PAC DSP: A Digital Signal Processor for Next-generation Mobile Platform

張傳華 Chuan-Hua Chang 廖宜道 I-Dao Liao
High Frequency IC Design Dept.

Abstract

PAC DSP is a 32-bit fixed-point low power high performance digital signal processor with 5-way VLIW pipeline developed at STC/ITRI targeted for the next generation mobile platform. It has one scalar cluster and two data stream clusters. Each data stream cluster contains two functional units and a distinct partitioned low power register file structure. PAC DSP has a rich, but optimized, instruction set which supports 8-bit and 16-bit SIMD operations. It is targeted to run at a maximum frequency of 250-300 MHZ. Along with the development of PAC DSP processor, a complete tool chain of compiler, assembler, and linker is also developed. High performance assembly code library will be provided as well for multimedia applications.

1. Introduction

In today's media-rich environment, processing of audio, video, and image information becomes essential for any consumer product that wants to be successful in the market. Almost all the new generation of wireless phones on the market right now has improved digital camera, video capturing, and playing capabilities. Even the MP3 players such as Apple's iPOD start to have photo viewing capabilities. Not before long, all MP3 players will transform into Personal Media Players and enable people to watch high quality full-length movies everywhere they go.

However, designing products targeting the media-rich consumer market poses real challenges to system designers in the areas of performance, power, and flexibility. All of these media processing applications require an enormous amount of computations under stringent real-time constraints in order to provide consumers with smooth and high quality media experiences. And besides the quality of audio/video, the lasting power of a mobile device is also a very important user experience. It becomes a real challenge for a system to perform many video/audio processing operations while reducing energy consumption. Also new media formats and standards (e.g. MPEG-4, H.264, etc.) that need to be incorporated into these devices are coming out all the time and being improved frequently in a very short time frame.

Thus, how to design a system that has high computational power with the most efficient energy consumption while maintaining high flexibility to quickly adapt to different media types in order to prolong the product life time becomes an important issue.

To address the above challenging issues, we

have developed PAC DSP, a 32-bit fixed-point high performance low power digital signal processor IP core as a key component for the next-generation media-rich mobile products. It provides high computational power from its parallel processing of multiple multimedia signal processing instructions. It achieves efficient energy consumption from its unique architecture/micro-architecture design. And it maintains high application flexibility from its general programmability.

The PAC DSP digital signal processor core can be used as a co-processor in a dual-core processor architecture platform (e.g. ITRI/PAC) or used standalone in a single-processor digital signal processing platform. The PAC DSP digital signal processor features a scalable VLIW (Very Long Instruction Word) architecture, a rich, but optimized instruction set, an innovative register file structure, and a high performance memory subsystem. It is targeted, but not limited, to the following application domain:

- Video and Image processing (H.264, MPEG-4, JPEG, Color space transform, etc.)
- Audio and Speech processing (MP3, AAC, and GSM speech processing, etc.)
- Voice processing and enhancement (digital hearing aid, voice-controlled gadgets, voice codec, etc.)

This paper provides an overview introduction to the PAC DSP digital signal processor core. It describes its key features including instruction set, architecture, micro-architecture design, memory architecture, power management strategies, and software support. A simple PAC DSP code example will be provided to demonstrate its competitiveness when compared with leading digital signal processors on the market.

2. PAC DSP Features

PAC DSP is a 32-bit, fixed-point, five-way VLIW Digital Signal Processor. It includes the following key features:

- **Rich set of scalar and VLIW instructions with 8-bit/16-bit SIMD operation support** targeted for multimedia applications. Most of the instructions are fully predicated which can be used to reduce the number of branch instructions and their impact to performance.
- **Scalable VLIW data path** for easy extension of the signal processing performance. The current PAC DSP implementation has two VLIW clusters each with its own multiply-accumulate unit.
- **Variable instruction word/packet length** for code size reduction.
- **Heterogeneous and partitioned register files** for register file read and write port reduction, thus, improving the register file read and write performance, and reducing power and area.
- **Large number of registers** which reduce the frequency of accessing memory. A total of 152 registers are built in PAC DSP among all clusters.
- **Dynamic power management** for power saving of PAC DSP core.
- **Customized instruction set and functional unit interface** for different application platforms to add different hardware accelerators which are used to enhance specific DSP operations.
- **Standard AMBA master/slave system bus interface** for easy integration with other platform components.
- **Optimized instruction and data cache**

/memory architecture allow the DSP kernel to access the needed data with low latency and high bandwidth while maintaining a reasonably large on-chip memory.

- **Maximum target frequency 250-300 MHZ** to provide needed performance for next generation's demanding high-quality multimedia encoding/decoding applications (e.g. H.264)

3. Processor Architecture

The PAC DSP core contains the following components as shown in Fig. 1, i.e. PAC DSP kernel, Application-specific Accelerators, Memory subsystem, Interrupt interface unit, Embedded ICE unit, power control unit, and Bus Interface unit. The PAC DSP kernel contains the main instruction pipeline and is the computation engine of the chip. The application-specific accelerators are used to enhance the computational power of PAC DSP kernel. One example of such an accelerator is a motion-estimation engine for video encoding applications. The accelerators execute in parallel with the PAC DSP kernel and interface with the kernel using either the PAC DSP data memory or special status lines.

3.1 The PAC DSP kernel

Fig. 2 shows a block diagram of the architecture of the PAC DSP kernel. It has three main components: a program sequence control unit, a scalar unit, and two clusters of VLIW data path.

The program sequence control unit dispatches

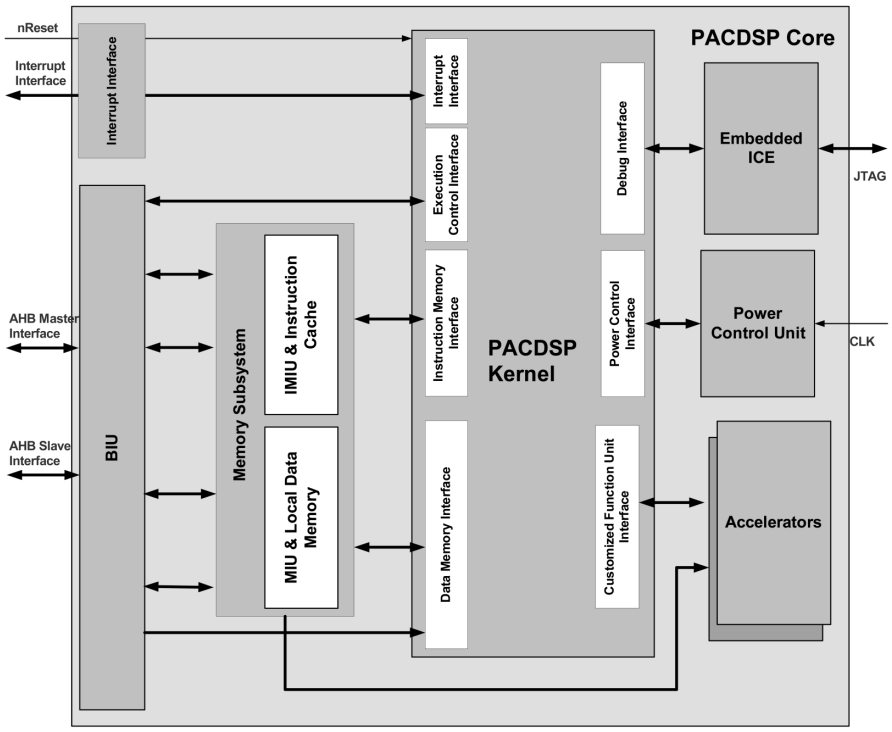


Fig. 1 PAC DSP Core architecture

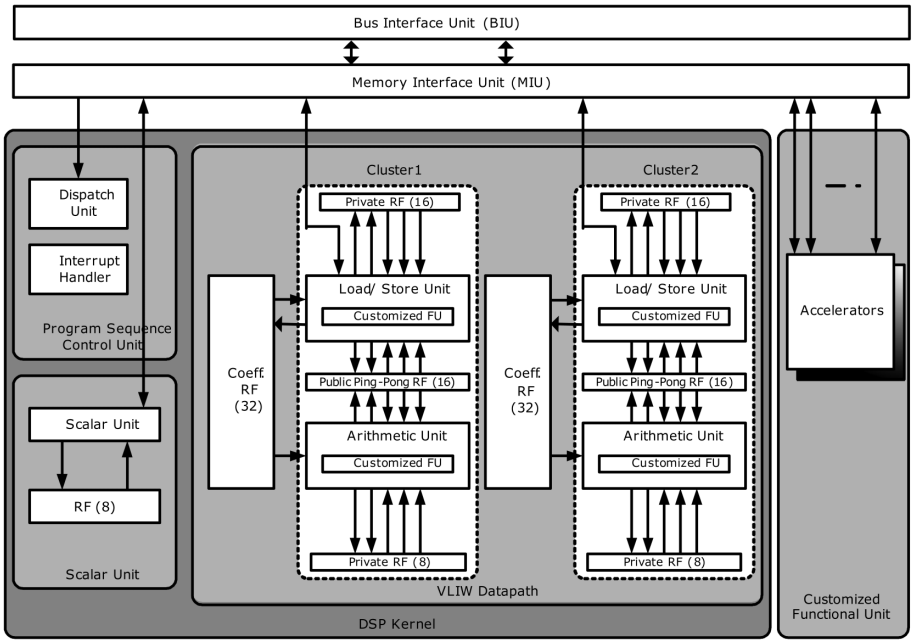


Fig. 2 The Architecture of PAC DSP Kernel

instructions to the scalar unit and VLIW data path. It also executes the control flow instruction and handles the interrupt and exception events. The program sequence control unit and the scalar unit use the same instruction slot in the instruction packet.

The scalar unit executes the scalar instructions and has 8 local registers. The scalar unit does not support SIMD type instructions. It also controls the power control interface and the customized functional unit interface.

The VLIW data path is composed of two clusters taking care of executing complex data operations in the program. The number of clusters in the VLIW data path can be scaled up or down based on target application's performance requirement. Each cluster contains a load/store unit (L/S) and an arithmetic unit (AU). Both units can execute instructions concurrently. Thus, two instruction slots in the instruction packet are allocated a cluster. Each cluster has its own register files structure. And communicating register file data between clusters is achieved using explicit "data broadcast" and "receive" instructions. The execution capabilities of the two clusters are the same.

3.1.1 Cluster Register File Structure

The register file structure in each VLIW data path cluster is illustrated in the following Fig. 3. It contains four distinct register files, Address register file (A0-A15), Accumulation register file (AC0-AC7), Coefficient register file (C0-C31), and Ping-Pong register file (D0-D15).

The Address register file has sixteen 32-bit Address Registers (A0-A15). It can only be accessed (read and written) by the L/S unit. The main use of the Address register file is to provide memory address pointers and various addressing

mode logic to the L/S unit, which is responsible for generating the load and store address to the data memory subsystem.

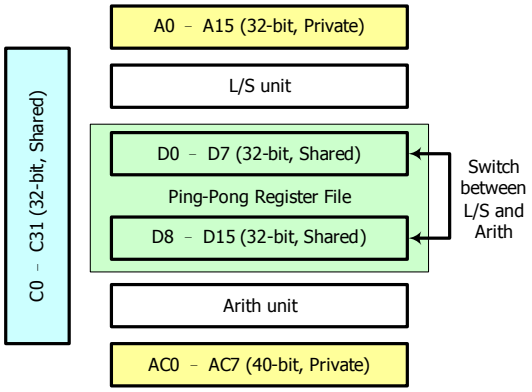


Fig. 3 Register file structure in a VLIW data path cluster.

The Accumulation register file has eight 40-bit Accumulation Registers (AC0-AC7). And it can only be accessed (read and written) by the Arithmetic unit. The accumulation registers are used mainly for storing intermediate multiplication and other computation results.

The Coefficient register file has thirty-two 32-bit Coefficient Registers (C0-C31). And it can be read by both L/S unit and Arithmetic unit. But it can only be written by load instructions in the L/S unit. The Coefficient register file is designed this way to provide a large space storing coefficient constants frequently required in digital signal processing algorithms and still maintain very low power consumption by limiting the number of read/write ports.

The Ping-Pong register file has sixteen 32-bit Ping-Pong Registers. The sixteen Ping-Pong registers are further divided into two banks of eight registers (D0-D7 and D8-D15). The Ping-Pong register file is shared between L/S and arithmetic units. Although both L/S Unit and Arithmetic Unit can share the ping-pong register file, the bank they access has to be different for the register read and write operations among the two functional units. This Ping-Pong register file access constraint is encoded in the instruction word and packet and is enforced by the compiler or the assembler generating the code. The advantage of the Ping-Pong register file structure lies in its ability to provide data communication between the two functional units effectively while consuming less power due to its reduced read/write ports.

3.1.2 Predicate Registers

PAC DSP has a fully-predicated instruction set architecture. Every instruction has a predicate register as its source operand. If the predicate register has a value of one, the instruction then will be executed. If the predicate register has a value of zero, then the instruction will not be executed or the execution result will be discarded. The PAC DSP branch instruction is using this predicate register to determine if the branch is taken or not as well.

PAC DSP has a globally visible predicate register file which can be read or written by all five functional units. The predicate register file has sixteen predicate registers (P0-P15) each of which is 1-bit wide. The predicate register P0 always contains the value of one. Thus, every unconditionally-executed instruction will have P0 as its source predicate register.

3.1.3 VLIW Execution Pipeline

The PAC DSP instruction execution pipeline has 7 stages. The operations performed in each

stage are illustrated in Fig. 4.

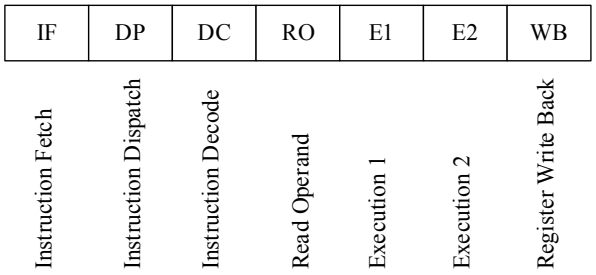


Fig. 4 PAC DSP Instruction Execution Pipeline.

In the IF stage 256 bits of instruction data is fetched from Instruction cache/memory into an instruction buffer. In the DP and DC stage the instruction packet in the instruction buffer is disassembled into instruction word and then up to five instruction words will be sent to the corresponding functional unit pipeline preparing for execution. In the RO stage the instruction source operands will be read from register files. Most of the basic and simple instructions will produce their results at the end of E1 stage. The result of a multiplication instruction will be produced at the end of the E2 stage. For memory load/store instructions, E1 stage is used to calculate the memory addresses. And the calculated memory address will be send to the memory subsystem at the end of the E1 stage. For the current version of PAC DSP, up to two such addresses can be sent to the memory subsystem. And for memory load instructions, the memory data will be coming back from data memory during the WB stage and written back to registers at the end of the WB stage. Pipeline stall will happen if the memory subsystem is not able to supply the memory data during the WB stage.

In the PAC DSP's VLIW style pipeline, hardware is not responsible for scheduling instructions into the execution pipeline based on data dependencies. So the design of instruction dispatch logic in the DP stage is less complex than a non-VLIW design employed in most of the traditional microprocessors. Compiler or advanced assembler need to schedule instructions properly in the program based on instruction latencies.

3.2 Memory Subsystem

The memory subsystem contains a 16K-32K byte direct-mapped instruction cache, and a 64-128K byte data memory. For performance enhancement, a 16K byte 2-way set-associative data cache may be added. The data memory/cache is dual-ported servicing the three clusters (1 scalar, 2 VLIW data path) in the DSP kernel. To simplify hardware design and reduce resource conflict, compiler is required not to put more than two load/store instructions in an instruction packet. This is designed under the assumption that the two ports of the data memory/cache will be used by the two VLIW data path clusters most of the time.

The data memory does not contain any instruction data. When a required instruction packet can not be found in the instruction cache, PAC DSP will try to find instructions in the platform memory space. This simplifies the interaction between the instruction cache and the data memory, thus simplifying the PAC DSP memory subsystem design. To maintain the coherence between the instruction cache and the platform DSP instruction memory, PAC DSP provides mechanisms for the platform software to flush/invalidate all instruction cache content. Future versions of PAC DSP chip may include separate instruction memory or

combined instruction/data memory to enhance program fetch performance.

The two-level data cache/memory architecture allows the data memory to be run with a slower frequency than the DSP kernel frequency without impacting the overall DSP kernel performance. The latency of a load instruction is very important to the performance of the DSP applications. Every latency increase of a load instruction not only delay the execution time of the consumer of the load data, but also add an extra delay slot which may be harder to fill in the application.

This two-level architecture provides a scalable road map for future PAC DSP core to increase its local memory size in the next generation process technology. In the current PAC DSP design, if a data cache is present, the data memory may be run at half of the DSP kernel frequency.

3.3 Bus Interface Unit

The bus interface unit is used to communicate with other key components in the platform: MPU, memory, and peripherals. PAC DSP core contains one or more master interface, and one slave interface. The slave interface allows another master component (MPU, DMA) on the system bus to access PAC DSP's data memory and control registers. The master interface allows the PAC DSP to access the instruction/data memory outside the PAC DSP domain, and control peripherals on the system bus. Current version of PAC DSP is targeted at implementing the AMBA bus protocol.

3.4 Interrupt Interface Unit

To support flexible platform design using PAC DSP, PAC DSP accepts interrupts from the platform. And to speed up the interrupt service routine processing, PAC DSP implements hardware shadowing of scalar registers for one of the interrupt types.

3.5 Embedded ICE Unit and JTAG Interface

For easy debugging and testing of PAC DSP core, an Embedded ICE unit and JTAG interface are built around the PAC DSP design for easy access to PAC DSP's internal states.

4. Instruction Set

PAC DSP digital signal processor instruction set is a fully predicated (conditional execution) instruction set and has the following instruction types:

- Data transfer instructions – within clusters and between clusters
- Compare instructions
- Arithmetic instructions
- Multiplication instructions
- Multiplication & Arithmetic instructions
- Logical instructions
- Shift instructions
- Memory load/store instructions
- Program flow control instructions
- DSP configuration control instructions – e.g. dynamic power configuration control.
- Special signal processing instructions – e.g. Butterfly, dot product, Sum-of-Absolute-Difference, etc.
- Bit-manipulation instructions

In addition to 32-bit and 16-bit operations, The PAC DSP instructions support 8-bit (byte), 16-bit (half-word) SIMD operations for many arithmetic instructions. It also supports double word (64-bit) load and store instructions to increase the load/store data bandwidth. To facilitate system design, instructions are added to perform word/half-word endian conversion on data stored in a register. To facilitate application development, instructions are added to speed up loading of un-aligned 32-bit data element.

Since PAC DSP has a scalar cluster and one or more VLIW data path clusters, the PAC DSP instructions can be classified into scalar cluster type instructions, VLIW data path type instructions, and common type instructions. The scalar cluster is responsible for program flow control so only scalar cluster unit has flow control change instructions (e.g. branch, break, continue, etc.). The VLIW data path cluster is responsible for data stream computation so it can execute compute-oriented instructions such as multiply, MAC, dot product, or butterfly etc. There are some common instructions that can be executed in both scalar clusters and VLIW data path clusters such as add, compare, memory access instructions.

To reduce program code size, PAC DSP employs a variable instruction word/variable instruction packet encoding scheme. The size of instruction word is varied from 16 bits to 48 bits. And the number of instruction word in an instruction packet is varied from zero to five. A packet header field in the instruction packet is used to encode this variability.

4.1 Memory Addressing Mode

PAC DSP provides three main addressing modes for calculating the memory address for the memory load and store instructions. They are linear addressing mode, bit-reverse addressing mode, and modulo addressing mode. Memory linear addressing mode has two variations: immediate data, and register indirect. For register indirect linear mode post-increment and pre-increment are supported. For bit-reverse and modulo addressing modes, only post-increment flavor is supported.

Bit-reverse addressing is useful for addressing the twiddle factors in 2^k point FFT computation. It is also called reverse-carry addressing. When the bit-reverse addressing mode is selected, the address computation will be performed with extra bit-reversing operations on all the source and destination operands. Fig. 5 shows an example of 3-bit bit-reverse address generation which illustrates the successive contents in A0 register when the “LB D0,(A0)+4” is being executed successively.

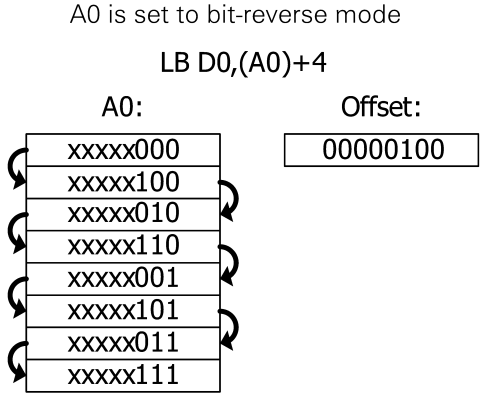


Fig. 5 Bit-Reverse Addressing Mode (3-bit)

Modulo addressing mode is useful for creating circular buffers for FIFO queues, delay lines, and sample buffers in the memory space. When modulo addressing mode is selected, address computation is performed with modulo M arithmetic, where M ranges from 1 to $+2^{32}$-1. Modulo M arithmetic causes the address register value to remain within an address range of size M, thus defining a buffer with a lower and an upper address boundary.

5. Power Management

PAC DSP digital signal processor is optimized in power consumption for the next-generation energy-sensitive, battery-powered devices such as smart phone, portable multimedia player, digital hearing aid, etc. The strategies for power reduction are employed from algorithm level to circuit level in PAC DSP core.

Algorithm level

In the application algorithm level, compiler-aided power-aware instruction scheduling will be used to reduce the overall power of the application from a more global point of view.

Architecture level

In PAC DSP architecture, register file read and write power are greatly reduced through register file port reduction using global/private register file partition scheme and global ping-pong register file structure.

Micro-architectural level

The scalable VLIW cluster in the PAC DSP core can be disabled based on the performance / power requirements of an application.

Register Transfer level

Fine-grain clock gating technique is used in every stage of the PAC DSP pipeline to dynamically turn off switching activities to save energy consumption.

Circuit level

Further power reduction is achieved by reducing leakage power through the use of multiple threshold cell element design flows during the PAC DSP core design process.

6. Software Development and OS Support

To support PAC DSP application and system development, a complete set of software development tools will be provided in conjunction with the PAC DSP system development platform. The tools include C compiler, Assembler, Linker, Loader, Instruction Set Simulator, and Debugger.

The high-level C compiler ensures that PAC DSP application can be developed in a programmer-friendly environment, thus reducing time-to-market and development cost for the final product.

For some multimedia applications requiring really high performance, PAC DSP high performance digital signal processing library will be provided to speed up the application performance and development.

For easy PAC DSP system management and multi-tasking support, a PAC DSP Micro Kernel and its interfaces will be provided to application

programs to facilitate common system tasks.

7. PAC DSP Code Example

In this section, we will present an example of implementing a 4x4 16-bit matrix multiplication Y=CX routine (Fig. 6) using PAC DSP instructions. This is used to demonstrate a few PAC DSP instructions and its register file architecture.

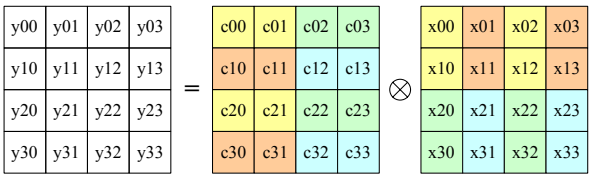


Fig. 6 4x4 16-bit matrix multiplication

Matrix C in this example is a constant coefficient matrix. The assembly code is written under the assumption that the sixteen constants are layout in memory in a row-based fashion (Fig. 7). It is further assumed that all of the sixteen constants have been loaded into registers preparing for continuous 4x4 matrix multiplication operations. Two successive 16-bit coefficients are stored in one 32-bit register.

m: C0		m+4: C1		m+8: C2		m+12: C3	
c00	c01	c02	c03	c10	c11	c12	c13
m+16: C4		m+20: C5		m+24: C6		m+28: C7	
c20	c21	c22	c23	c30	c31	c32	c33

Fig. 7 Memory layout of matrix C. "m" is the starting base address. C# is the 32-bit register that contains two 16-bit coefficients.

Matrix X is assumed to be layout in memory in a column-based fashion (Fig. 8). All data in matrix X will be loaded into registers in the assembly code. And similar to the constant coefficient, two successive matrix X 16-bit data in memory will be loaded into one 32-bit register for computation.

n: X0		n+4: X1		n+8: X2		n+12: X3	
x00	x10	x20	x30	x01	x11	x21	x31
n+16: X4		n+20: X5		n+24: X6		n+28: X7	
x02	x12	x22	x23	x03	x13	x23	x33

Fig. 8 Memory layout of matrix X. "n" is the starting base address. X# represents the 32-bit data that will be loaded into a 32-bit register

Matrix Y is assumed to be layout in memory in a row-based fashion (Fig. 9). Each element of matrix Y is 32 bits. The code presented in this section does not convert the 32-bit element to 16-bit element before storing it back to memory.

k	k+4	k+8	k+12
y00	y01	y02	y03
k+16	k+20	k+24	k+28
y10	y11	y12	y13
k+32	k+36	k+40	k+44
y20	y21	y22	y23
k+48	k+52	k+56	k+60
y30	y31	y32	y33

Fig. 9 Memory layout of matrix Y. Each element of matrix Y is 32 bits. "k" is the starting base address.

The PAC DSP code of this example is listed in Fig. 10. Every cycle in PAC DSP, five functional units are available for executing instructions. The computations of the sixteen elements of the matrix Y are equally distributed in PAC DSP's two VLIW data path clusters. Cluster 0 is responsible for elements y<0..3>0 in the first iteration of the code and y<0..3>2 in the second iteration of the code. Cluster 1 is responsible for elements y<0..3>1 and y<0..3>3, also in two iterations, respectively. The

order of generating these elements is arranged this way in order to reduce the number of accessing matrix X data from memory. This code uses "dot product" (dotp2 with two cycle latency) instruction to combine three operations (two 16-bit multiply and one 32-bit add) to increase the number of parallel operations every cycle. The dot product instruction multiply the 16-bit low-half pair and the 16-bit high-half pair of the two source operands and then add the results together to form a 32-bit data. Line 1 and line 2 of the code set up the addresses for loading and storing memory and conditions for loop control. Line 3 to line 16 constitutes the main loop body. In line 3, two special "double load word" (dlw with three cycle latency) instructions are used to load a total of 128 bits of data into four registers.

The total basic operations in this example contain 64 multiplications, 48 additions, 16 loads and 16 stores. However, with PAC DSP's special support of the "*dot product*" and "*double load word*" operations, this routine can be implemented with 16 PAC DSP instruction packets and will be completed in 30 cycles provided there is no memory stalls for the load/store instructions. On average, more than 4 operations are being executed per cycle in the PAC DSP pipeline.

	Scalar unit	C0 cluster		C1 cluster	
	S unit	L/S unit	Arith unit	L/S unit	Arith unit
1	mov R0<-0	mov A0<-n		mov A0<-n	
2	mov R1<-1	mov A1<-k-8		mov A1<-k-8	
3	seq R0,R1,P4,P5	dlw D0(D1)<-A0,0		dlw D0(D1)<-A0,8	
4	add R0<-R0,1	add A0<-A0,16		add A0<-A0,16	
5		add A1<-A1,8		add A1<-A1,8	
6			dotp2 D8<-D0,C0		dotp2 D8<-D0,C0
7			dotp2 D10<-D1,C1		dotp2 D10<-D1,C1
8			dotp2 D9<-D0,C2		dotp2 D9<-D0,C2
9		add A8<-D8,D10	dotp2 D11<-D1,C3	add A8<-D8,D9	dotp2 D11<-D1,C3
10		sw A8,A1,0	dotp2 D8<-D0,C4	sw A8,A1,4	dotp2 D8<-D0,C4
11		add A8<-D9,D11	dotp2 D10<-D1,C5	add A8<-D9,D11	dotp2 D10<-D1,C5
12		sw A8,A1,16+0	dotp2 D9<-D0,C6	sw A8,A1,16+4	dotp2 D9<-D0,C6
13		add A8<-D8,D9	dotp2 D11<-D1,C7	add A8<-D8,D9	dotp2 D11<-D1,C7
14		sw A8,A1,32+0		sw A8,A1,32+4	
15	(p4) B <line 3>	add A8<-D9,D11		add A8<-D9,D11	
16		sw A8,A1,48+0		sw A8,A1,48+4	

Fig. 10 PAC DSP example code of 16-bit 4x4 matrix multiplication

	S1/L1	D1	M1	D2	M2
1	mvk 0->A30	mvk n->A0		mvk n->B0	
2	mvk 1->A31	mvk k-8->A1		mvk k-8->B1	
3		lddw A0,0-> A7:A6		lddw B0,8->B7:B6	
4		add A0,16->A0		add B0,16->B0	
5		add A1,8->A1		add B1,8->B1	
6	cmpeq A30,A31->A2				
7	add A30,1->A30				
8			dotp2 A6,C0->A8		dotp2 B6,C0->B8
9			dotp2 A7,C1->A9		dotp2 B7,C1->B9
10			dotp2 A6,C2->A10		dotp2 B6,C2->B10
11			dotp2 A7,C3->A11		dotp2 B7,C3->B11
12			dotp2 A6,C4->A12		dotp2 B6,C4->B12
13		add A8,A9->A20	dotp2 A7,C5->A13	add B8,B9->B20	dotp2 B7,C5->B13
14		sw A20,A1,0	dotp2 A6,C6->A8	sw B20,B1,4	dotp2 B6,C6->B8
15	(A2) B <line 2>	add A10,A11->A21	dotp2 A7,C7->A9	add B10,B11->B21	dotp2 B7,C7->B9
16		sw A21,A1,16+0		sw B21,B1,16+4	
17		add A12,A13->A20		add B12,B13->B20	
18		sw A20,A1,32+0		sw B20,B1,32+4	
19		add A8,A9->A21		add B8,B9->B21	
20		sw A21,A1,48+0		sw B21,B1,48+4	

Fig. 11 TI C64 example code of 16-bit 4x4 matrix multiplication

A similar code using TI's C64 instructions is constructed in Fig. 11 as a comparison to PAC DSP's architecture. TI's C64 can perform eight instructions in a VLIW packet. However, in this example, due to resource constrains and data dependencies, only six out of the eight slots have ever been used. The main loop body is from line 3 to line 20. Line 1 and 2 are used just for setup.

TI C64 has similar "dot product" and "load double-word" instructions to PAC DSP chip. However, due to TI's long instruction latencies of these instructions (dotp2 with four cycle latency and lddw with five cycle latency), an iteration of this loop takes eighteen cycles. The whole routine takes 38 cycles to complete. Clearly, PAC DSP has an eight cycle advantage if PAC DSP and TI's C64 are running in the same frequency.

Unrolling the loop will reduce this cycle difference between PAC DSP and C64. However, doing this will increase the code size.

8. Conclusions

PAC DSP is a high performance low power VLIW processor with optimized digital signal processing instruction set. It is an ideal component

for system designers facing a challenge task of designing a competitive product processing rich media contents in a limited time frame. Complete software tools reduce the time-to-market for the product developed on PAC DSP core.

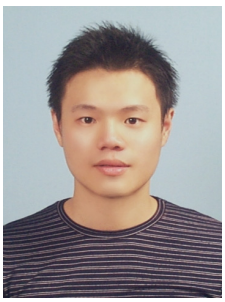
Authors

張傳華

Chuan-Hua Chang is a technical manager in the PAC DSP digital signal processor design group in SoC Technology Center / ITRI since 2004. He received his Ph.D. and Master degree in 1996 and 1992 respectively in Computer Science and Engineering from the University of Michigan, USA. From 2001 to 2004, he worked at Intel as architecture box lead designing Itanium 64-bit processor. From 1996 to 2000, he worked at DEC as senior architect, later, architecture box lead designing Alpha 64-bit processor. His research interests include computer architecture, processor micro-architecture, digital signal processing hardware/software, VLSI/CAD.

E-mail: chuanhua.chang@itri.org.tw

廖宜道



I-Tao Liao is the project leader of PAC DSP digital signal processor project in SoC Technology Center / ITRI since 2003.

He received his Master degree in 2000 in Computer Science from NCTU.

From 2001 to 2002, he joins the Radio/Channel processor project designing the architecture.

E-mail: udall@itri.org.tw