A COST EFFECTIVE MULTIMEDIA EXTENSION TO ARM7 MICROPROCESSORS

Ing-Jer Huang, Wen-Kai Huang, Rui-Ting Gu and Chung-Fu Kao

Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan

ABSTRACT

This paper presents a cost-effective multimedia extension to the ARM7 architecture (v4T), a low-cost embedded microprocessor. A *MultiMedia Extension (MME)* module has been constructed as a functional unit for the ARM7 processor core. It provides common media operations such as saturating addition and multiply-accumulation, *etc.* All operations of the MME module perform subword parallelism on byte or half-word entities. The SIMD-styled operations allow a higher processing throughput. The module has been successfully integrated with a synthesizable ARM7 core. The synthesis results show that the MME module is capable of reducing the execution cycle counts by 69% to 81% for several multimedia critical loops, with an average speedup of 3.41, while the gate count overhead is kept under 11%.

1. INTRODUCTION

Multimedia workloads have always played an important role in embedded applications, e.g. MP3 players, language learning players, digital cameras and video games. During the last decade, many computer architectures optimized for multimedia applications have been proposed. These architectures can be classified into three different categories:

1. Multimedia dedicated processors. Examples include Chromatic' s Mpact [1] and Philips TriMedia [2].
2. Signal processing coprocessors, for example, ARM' s Piccolo architecture [3].
3. Instruction set architecture extensions such as MAX for HP PA-RISC architecture [4], MMX for Intel x86 architecture [5] and VIS for UltraSPARC [6].

Among for these categories, dedicated processors may offer the best performance benefit. The microarchitecture that surrounds the VLIW-like core with a variety of media ports allows dedicated processors to provide the highest computing capability and diversify functionality.

Even so, many low price consumer products would mainly focus on low cost issues rather than high computing performance. Such products have relatively small chip area and well-defined workloads. From this viewpoint, we believe that ISA extension to existing processor is preferable for this kind of applications. However, many of such extended processors do not target the embedded market; this is an important motivation of our research. We are interested in adopting ISA extension to improve the multimedia capabilities of embedded microprocessors for low cost consumer products.

In this paper, we have implemented an *MultiMedia Extension (MME)* module to extend the media capability of the ARM7 architecture (v4T). The MME module is a small but useful functional unit for accelerating the execution for media algorithms. Our goal is not only to improve the media performance, but also to keep the small-area benefit of the ARM7 processor core for low cost applications.

Related works

ARM Limited has proposed two extensions to the ARM architecture in attempting to enhance the media processing performance: the Piccolo coprocessor and the signal processing instructions set extensions in ARM architecture v5TE [3]. The Piccolo coprocessor is a 16-bit engine that uses the ARM coprocessor interface to cooperate with the ARM core. The signal processing instructions in v5TE fall into two groups: multiplications and 32-bit saturating addition/subtraction. Besides, Intel has introduced the XScale architecture [7]. This ARM v5TE core-compliant microarchitecture surrounds the processor core with a multiply-accumulate coprocessor for efficient processing of audio media algorithms. However, none of the above efforts provide the subword parallelism for small data types. For example, the v5TE saturating addition handles only 32-bit additions and thus doesn' t fit very well for pixel operations in image applications. The main contribution of the MME module is to provide common multimedia operations with efficient parallelism for such data elements.

0-7803-7448-7/02/$17.00 ©2002 IEEE

The rest of this paper is organized as follows: Section 2 introduces the organization and functions of the MME module. Section 3 discusses the modification of the ARM7 processor core. Section 4 presents the results of the integration of the MME module with a synthesizable ARM7 processor core. Finally, section 5 concludes the paper with a discussion on future work.

2. OVERVIEW OF THE MME MODULE

The data-path of MME module is 32-bit wide. Hardware designers simply can treat this module as a pure combinational function unit. All operations, include 16-bit multiply-accumulate operations, are completed in one cycle.

2.1. Main structure

The key feature of multimedia applications is that the typical data size of operands is relatively small, for example, 8-bit pixels and 16-bit audio samples. Also, multimedia processing typically involves performing the same computation on a large number of adjacent data. These two properties lend themselves to the use of SIMD architecture. Base on SIMD structure, the MME module can operate on groups of four bytes and two half-words. These groups of 32 bits are referred to as packed data. For instance, one MME packed-bytes addition can handle the workload with one execution cycle that is operated by four normal ARM addition instructions with four cycles. (See Figure 1) It is the fundamental reason for the speedups achieved by the MME module. The significantly decrease of execution cycle counts results in overall performance improvement.

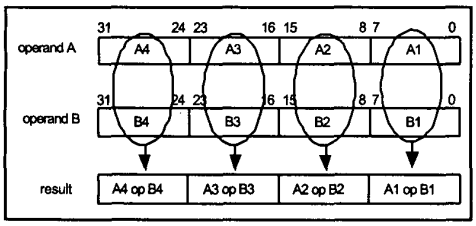

Figure 1. Parallel operation for packed data

2.2. Supported multimedia operations

In the literature, several studies [5], [8], indicate that the most frequent operations in multimedia algorithms can be summarized as follow:

- Packed data arithmetic operations
- Saturation arithmetic
- Fixed-point arithmetic
- Data rearrangement operations
- Data formatting operations
- Conditional execution
- Memory operation

Based on the above analysis, we implement the following operations in our current MME module. (See Table 1)

Operations	Description
PADD	Subword addition with saturation
PSUB	Subword subtraction with saturation
PCMPEQ	Subword compare for equal
PCMPGT	Subword compare for greater than
PSLL	Subword shift left logical
PSRL	Subword shift right logical
PSRA	Subword shift right arithmetical
MOV	Data movement
PMULLW	Multiply for low word product
PMULHW	Multiply for high word product
PMADD	Multiply-accumulation
PUNPACKBH	Unpack byte to half-word
PACKSSHB	Pack signed half-word to byte
PACKSSWH	Pack signed word to half-word
PACKUSWH	Pack unsigned word to half-word

Table 1. Operations of MME module

2.3. I/O interface

Figure 2 shows the block diagram of MME module. There are three functional components in the module. There is a small internal controller to provide the necessary control signals.

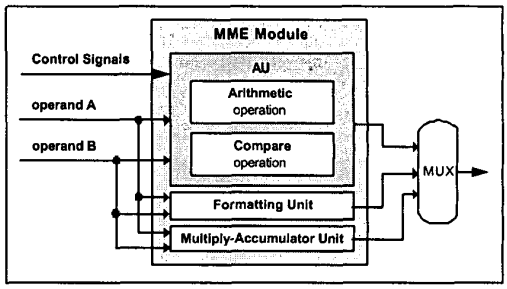

Figure 2. Block diagram of MME module

The arithmetic unit provides byte or half-word arithmetic operations with an optional saturation mode. The compare unit executes special compare instructions. The formatting unit performs data formatting operations and shift operations. The single cycle 16-bit multiplication with 32-bit accumulation is provided by the multiply-accumulator. Note that the I/O signals of MME module are quite general. These signals can be easily integrated into a microprocessor core. This characteristic makes MME module possible to be reused in different instruction set architectures. In our another research, we plan to integrate it with an 8-bit microcontroller for voice recognition for low cost consumer electronics products.

3. INTEGRATING THE MME MODULE TO THE ARM7 PROCESSOR CORE

This section describes how to integrate the MME module into the ARM7 architecture for multimedia extension. Figure 3 shows the block diagram of the ARM7 processor core. It has three-stage pipelined organization. For our study, we have implemented an academic synthesizable processor core, named N-ARM7TM, which implements the ARM architecture v4T. It supports 16-bit Thumb instruction mode (T) and has a Booth multiplier (M).

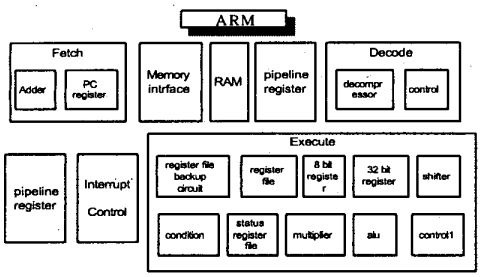

Figure 3. ARM7 block diagram

3.1. Definition of new instructions

There are several methods for extending new instructions to the original instruction set architecture.

1. One of the possible ways is to increase the instruction length for encoding of new instructions
2. The second method is to define new operating modes for execution of new instructions. The mode switching between original instructions and new ones is controlled by some flags. For example, the T-flag in ARM7 processors switches the processor between THUMB mode (16-bit instruction set) and ARM mode (32-bit instruction set).
3. The third way is to use the unused field in the original instruction encoding without modifying the instruction length or extending operating modes.

The first two methods would provide more encoding space for new instructions than the third method. However, a large amount modification in instruction decoder will be required for the first two methods. Besides, the existing executables for original instruction set will also need to be re-compiled. Therefore, we choose the third method.

From the encoding of ARM7 instructions, there are several unused bits in the "undefined" instruction. We use these bits to extend new instructions. Figure 4 shows the encoding of the "undefined" instruction for our extended ones. Fifteen new instructions had been defined for the new media instructions set.

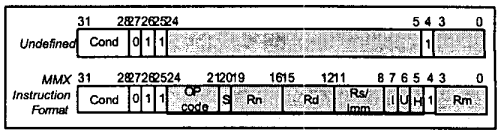

Figure 4. Encoding of new instructions

3.2. Hardware modification for the processor core

The usage of the MME module is similar to the usage of common functional units such as multipliers. The MME module receives the opcode, operands with related parameters and then produces the result of computation. This module is integrated into the execution stage. The data-path of MME module is parallel with existing ALU and multiplier in the processor, as shown in Figure 5.

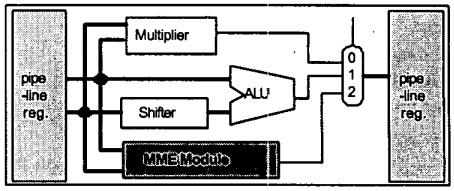

Figure 5. Schematic view of the execute-stage

The next step is to modify the instruction decoder for generating related control signals. Note that there is an internal decoder within the MME module. The internal decoder generates all necessary control signals from the input of MME module. Since the internal decoder has shared the decode responsibility for instruction decoder, it greatly simply the modification of the original control path.

4. EXPERIMENTS AND RESULTS

Both the ARM7 core and the MME module are Verilog RTL code and are synthesized using TSMC 0.35um cell library with the same global synthesis constraints to derive the gate-level design. There are 8993 Verilog statements for the original processor core, and 724 statements for the MME module. We call the MME enhanced processor N-ARM7TMX. The total amount of statements for N-ARM7TMX is 9847.

In order to analysis the speed overhead of the circuit, we measure the circuit delay for each pipeline stage of processor core. Table 2 shows the circuit delay of each stage. From this table, we see that the critical path has moved from the decode stage to the execution stage and is increased by 0.7 ns. This is due to the single-cycle multiply-accumulator of MME module. However, we believe that the delay of this path could be furthermore reduced if aggressive synthesis constraints are used. On the other hand, the gate count variation is summarized in Table 3. It shows that the gate overhead is kept under 11%. 5.5K gates are used for our multimedia extension.

	Fetch	Decode	Execute	Global critical path delay
N-ARM7TM	16.93	17.05	16.98	**17.05**
N-ARM7TMX	16.93	17.09	17.75	**17.75**

Table 2. Circuit delay of each stage (unit: ns)

	N-ARM7TM	N-ARM7TMX	Overhead
Gate count	50,387	55,761	11%

Table 3. Area overhead (unit: gates)

For the purpose of performance evaluation, we measure the execution time of N-ARM7TMX and N-ARM7TM for a given benchmark set. Within our experiments, five critical code portions of media algorithms are used as the benchmarks. They are *Chroma Keying* image overlay for a 256x256 picture; *Convolution* and *Correlation* for 512 audio samples; *Absolute Difference* and *Maximum Number* between two 256x256 pixel arrays.

Each benchmark is compiled into two versions of assembly code. The first version is the original ARM code for N-ARM7TM. The other one is functionally identical to the first version but further enhanced by new media instruction set from N-ARM7TMX. An example for Chroma Keying is given in Table 4, and more experiment results are shown in Table 5.

C code operation

```
For( index=0; index<65536; index++ ) {
 if( x[index]==Blue )
  new_image[index] = y[index];
 else
  new_image[index] = x[index];  }
```

Assembly code for N-ARM7TM

```
    LDR   R1, BLUE_VALUE
    ADR   R2, old_img
    ADR   R3, back_img
    ADR   R4, new_img
    MOV   R0, #0
Main_loop
    LDRB  R5, [R2,R0]
    CMP   R5,R1
    BNE   NO_OVERLAY
    LDRB  R5, [R3,R0]
    STRB  R5, [R4,R0]
    B     CONT
NO_OVERLAY
    STRB  R5, [R4,R0]
CONT
    ADD   R0, R0, #1
    AND   R0, R0, 0xff
    CMP   R0, #0x10000
    BLT   Main_loop
```

Assembly code for N-ARM7TMX

```
    LDR  R1, BLUE_VALUE
    ADR   R2, old_img
    ADR   R3, back_img
    ADR   R4, new_img
    MOV   R0, #0
Main_loop
    LDR   R5, [R2],#4
    LDR   R6, [R3],#4
    PCMPEQ  R11, R5, R1
    AND   R6, R6, R11
    BIC   R5, R5, R11
    ORR   R7, R5, R6
    STR   R7, [R4],#4
    ADD   R0, R0, #4
    CMP   R0, #0x10000
    BLT   Main_loop
```

Table 4. C and assembly code for Chroma Keying image overlay

Benchmarks	Static Instructions		Execution Cycles	
	N-ARM7TM	N-ARM7TMX	N-ARM7TM	N-ARM7TMX
Chroma Keying	16	15	1,145,287	332,142 (-71%)
Convolution	21	19	1,360,071	421,622 (-69%)
Correlation	24	20	1,573,802	503,716 (-68%)
Absolute Difference	14	10	1,141,801	283,115 (-75%)
Maximum Number	12	9	1,010,729	214,268 (-79%)
Overall	**87**	**73**	**6,231,690**	**1,754,863 (-72%)**

Table 5. Benchmark execution characteristics

The total execution cycle count is decreased by 68% to 79% for our benchmarks due to the use of new media instructions. Combining the information of Table 2 with Table 5, we can compute the execution time of each processor and derive the speedup.

$$Speedup = \frac{\text{Execution Time}_{(N\text{-}ARM7TM)}}{\text{Execution Time}_{(N\text{-}ARM7TMX)}} = \frac{6{,}231{,}690 \times 17.05}{1{,}754{,}863 \times 17.75} = 3.41 \quad (1)$$

The speedup shows that N-ARM7TMX is about 3.4 times faster than N-ARM7TM for these workloads while the area overhead is only 11%. Even N-ARM7TMX has a slightly lower clock rate, its average performance is still better than N-ARM7TM because the execution cycle count of N-ARM7TMX has been decreased significantly.

5. CONCLUSION

For low-cost embedded multimedia applications, we introduce a cost-effective media extension to ARM7 microprocessors. A *MultiMedia Extension (MME)* module has been proposed as a functional unit in ARM7 processor core. It adopts a SIMD structure to perform packed data operations, and there are fifteen media instructions that have been defined. Five critical code portions of media algorithms are used as the benchmark for our experiments. The result shows that the speedup of media enhanced processor is 3.41 while the area overhead is only 11%. In our future research, we are interested in integrating the MME module with an 8-bit microcontroller for voice recognition in low-cost consumer electronic products.

6. REFERENCES

[1] P. Kalapathy, "Hardware/Software Interactions on the Mpact," *IEEE Micro*, pp. 20-26, Mar./Apr. 1997.

[2] G.A. Slavenburg, S. Rathnam, and H. Dijkstra, "The Tri-media TM-1 PCI VLIW Media Processor," *Proc. Hot Chips VIII Symp.*, IEEE CS Press, Los Alamitos, Calif., 1996.

[3] S. Furber, *ARM system-on-chip architecture, 2nd*, Addison-Wesley, 2000.

[4] R.B. Lee, "Subword Parallelism with MAX-2," *IEEE Micro*, pp. 51-59, July/Aug. 1996.

[5] A. Peleg and U. Weiser, "MMX Technology Extension to the Intel Architecture," *IEEE Micro*, pp. 42-50, July/Aug. 1996.

[6] M. Tremblay et al., "VIS Speeds New Media Processing," *IEEE Micro*, pp. 10-20, July/Aug. 1996.

[7] Intel Corporate, *Intel Xscale Microarchitecture Technical Summary*, http://developer.intel.com/design/intelxscale/

[8] R.B. Lee, "Multimedia Extensions for General-Purpose Processors," *Proc. IEEE Workshop on Signal Processing Systems*, pp. 9-23, Nov. 1997.