An Embedded Low Power/Cost 16-Bit Data/Instruction Microprocessor Compatible with ARM7 Software Tools

Fu-Ching Yang

Department of Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung 804, Taiwan
Tel : 886-7-5254337
Fax : 886-7-5254301
e-mail : fcyang@esl.cse.nsysu.edu.tw

Ing-Jer Huang

Department of Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung 804, Taiwan
Tel : 886-7-5254337
Fax : 886-7-5254301
e-mail : ijhuang@cse.nsysu.edu.tw

Abstract - A 16-bit THUMB instruction set microprocessor is proposed for low cost/power in short-precision computing. It achieves 40% gate count, 51% power consumption and 160% clock frequency comparing to ARM7, even the performance is 67% better in narrow width memory at the same clock frequency. The ARM7 software is also compatible.

I Introduction

Embedded microprocessors are divided into 3 categories: 8-bit, 16-bit and 32-bit microprocessor, depending on the demand of performance, cost, power, and programbility. For simple control system witch requires extremely low cost and low power, 8-bit microprocessor is the best choice [2]. But it often has low programbility that the software designers have to write the program in assembly language. In contrast to 8-bit microprocessor, 32-bit microprocessor, for example, ARM have high programbility, high performance and are widely used for applications need high computation such as cellular phone and PDA [3]. As for 16-bit microprocessor, they have higher performance power than 8-bit microprocessors [4] and lower power consumption than 32-bit microprocessors, are often used in 16-bit applications such as disk driver controller, airbags and cellular communication [4].

For the demand for extreme low power and low cost in mobile devices, many microprocessor design companies support 32-bit and 16-bit instruction sets in a 32-bit architecture microprocessor, for example the ARM's THUMB instruction set [1] and MIPS's MIPS16e instruction set [5]. In this paper, since ARM based embedded processors are widely used in embedded system [3], our work is based on ARM, especially ARM7. In ARM7 architecture, by changing the processor mode for executing the 32-bit ARM instruction set or the 16-bit THUMB instruction set, trade off can be made among the code density, performance and energy efficiency according to application specifications. ARM instruction set targets to high performance applications while THUMB instruction set targets to low cost applications. THUMB instruction set's 16-bit instruction length allows it to approach twice the density of standard ARM code [1].

However, for applications that are primary short-precision operations, ARM only solves half the problem. The reason is that although the 32-bit microprocessor can change the processor mode to execute the 16-bit instructions compiled for those applications, the data path inside the processor is still 32-bit width. According to the observation by Ramon et al. [6], the upper bits of data path nearly have no change for applications requiring only short-precision operand. In this case, 32-bit architecture is not suitable comparing to 16-bit architecture. Because by removing the unused 16-bit data path, 16-bit architecture has smaller chip size and lower power consumption than 32-bit architecture.

In this paper, for the reason above, we propose a 16-bit ARM microprocessor called SYS16TM based on ARM7 microprocessor architecture that has a 16-bit instruction set and 16-bit data path. We have two challenges. One is how to reduce the processor data path from 32-bit to 16-bit. Another is how to maintain software compatibility of ARM7 so the software tool chain such as the compiler and the simulator can be reused as much as possible. Thus the ARM software programmer can port applications to SYS16TM in a short time and benefit of low power and low cost.

The experimental results show SYS16TM can achieve about 49% of power and 60% of cost reduction comparing to the original ARM7 microprocessor. Even the performance of SYS16TM is 68% better than ARM7 when considering 16-bit width memory.

II. Related work

For 32-bit microprocessors, the upper 16-bit of the data path inside the pipeline stage has no changes at most of the time when executing short-precision applications [6]. For those applications, 16-bit data path is better than 32-bit data path in power consumption and cost. Since ARM is the most popular embedded microprocessor, we decide to propose a 16-bit microprocessor based on ARM architecture. Because the software environment of ARM can be reused, we don't have to create compiler and simulator on our own. And thus achieve fast time-to-market.

Before we explore the effect of reducing data path from 32-bit to 16-bit, we first consider the difference of instruction sets in ARM7. As describe in ARM7 specification [1], ARM7 has two instruction sets called ARM and THUMB. ARM instruction set is 32-bit and THUMB instruction set is 16-bit. Each THUMB instruction has a corresponding ARM instruction. By compiling program into THUMB instructions, it can reduce code size. The programmer should judge by application complexity to decide which instruction set to use.

1-4244-0630-7/07/$20.00 ©2007 IEEE.

Related research about how to choose between ARM and TUMB instruction set has been carried out [16]. From their result, it shows that THUMB code is able to provide 70% of the code size of ARM. The similar results are carried out in our experimental result in section 6.1.

But the reduction of instruction length leads to more execution time [17]. For example, THUMB instruction set has shorter immediate and offset field comparing to ARM instruction set. It also sacrifices several features that ARM instruction set has, such as conditional execution instruction, register file access ability and so on [16]. Because of the limitation, more instructions required when a program is compiled in THUMB instruction set. According to our experimental result, the performance degradation is about 6%.

However, memory usually has narrow bandwidth in many embedded systems. In such system, 8-bit and 16-bit microprocessor can benefit from it. Take 16-bit bandwidth memory for example, every instruction needs to be fetch twice from memory in ARM mode, while in THUMB mode, fetching one instruction takes only one memory access. From the ARM7 specification [1], THUMB code provides 160% of the performance of an equivalent ARM7 processor connected to a 16-bit memory system. Same experimental results are obtained in section 6.3.

From the comparison between ARM and THUMB above, we know that THUMB consumes less memory space by code size reduction, thus reducing power consumption. And it also has better performance than ARM with low bandwidth memory which is commonly used in low cost embedded systems.

But for short-precision applications, even if we compile them to THUMB instructions, the power and cost reduction are not good enough. The reason is that the data path inside ARM7 is still 32-bit. In our paper, we not only implement THUMB instruction set as our instruction set architecture but also farther reduce the data path from 32-bit to 16-bit in order to achieve further power reduction.

Related work like Ramon et al. do [6], they also reduce the data path from 32-bit to 16-bit in his halfword-serial architecture. In this architecture, although the Clock-Cycle-Per-Instruction (CPI) is increased by 131%, the power consumption can be reduced by 44~58%. In many 16-bit embedded systems, because we more care about power and cost than performance, the loss of performance is acceptable.

There for, in this paper, we propose a 16-bit data/instruction microprocessor. It can achieve significant low power/cost at acceptable performance degradation. For narrow width memory such as 16-bit memory, even the performance is 68% better than ARM7. And the software tool chain can be reused, so that the ARM software engineers can still use their experience when developing application in SYS16TM.

III. Programmer's model

In order to reuse of ARM7 software environment, we only modify what is necessary. So our microprocessor can be compatible with ARM7 simulator.

A. Instruction set

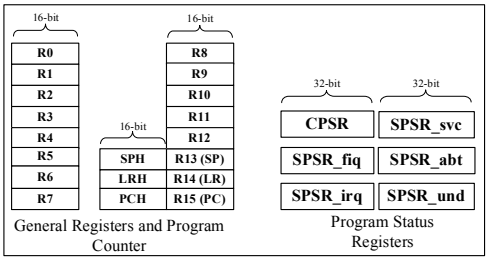

Fig. 1. Register file of proposed microprocessor.

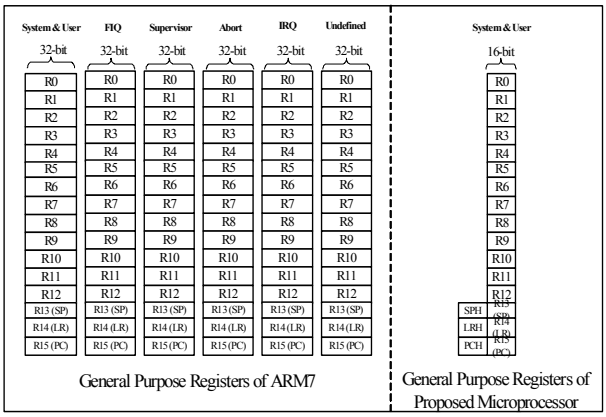

Fig. 2. Comparison between ARM7 register files and proposed microprocessor register files.

In this paper, the 16-bit microprocessor is designed based on THUMB instruction set. Following ARM7, THUMB is RISC load-store architecture simplified for high throughput. THUMB has complete set of logical, shift, bit manipulation, and arithmetic operations that operate on a register and either another register or a fixed length 3-8-bit immediate field according to different instruction format. Program can be executed in THUMB instructions independently without the support of ARM instruction set.

We implement all the THUMB instructions in our microprocessor except Branch Exchange instruction which is designed for changing between ARM and THUMB mode. Since we only implement THUMB instructions, this instruction format is no use at all. We replace it with new defined instruction.

B. Register files

For low power consumption, the data path is reduced from 32-bit to 16-bit as we mentioned in section 2. That is, the register file is also reduced from 32-bit to 16-bit. But for register PC (Program Counter), LR (Link Register) and SP (Stack Point), because these three registers are used for address recording and calculation, they have to keep in 32-bit to have 4 G byte address space. We use two 16-bit registers combined to form a 32-bit register for PC, LR and SP. The 16-bit register corresponding to the lower 16-bit of the 32-bit register is accessed by THUMB instructions. The upper 16-bit register is access by a new defined instruction call MOVH. Fig. 1 is the register file of our microprocessor. Like it shows, the upper 16-bit registers of PC, LR and SP are PCH, LRH and SPH [7] respectively.

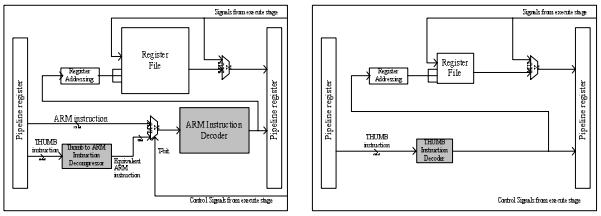

Fig. 3. Diagram of ARM decode stage and SYS16TM decode stage.

Besides, in ARM7, there are six register file for six different modes such as System, FIQ, Supervisor, Abort, IRQ and Undefined mode respectively. These register files are used for fast context switch. For example, if ARM7 is interrupted, registers don't have to be stored in memory by software. The hardware automatically stores them in another set of register file. But in our design, we only implement system mode register file while removing the other five. The primary reason for this decision is that we care about power and cost more than performance in 16-bit applications. It is necessary to keep the hardware simple. This modification helps reducing power and cost in two ways. First is that the power consumption of register file is great. Second is that the logic for switching among the six register files is complex. Fig. 2 is the comparison between ARM7 register files and SYS16TM. The proposed register file size is only 21% of ARM7 register file size. By taking account of the complex logic for switching among the different register files, the gate count can be smaller than 21%.

IV. Micro-architecture

In this section, we compare the difference between SYS16TM and ARM7, and explain how SYS16TM can be lower cost/power than ARM7. SYS16TM is designed in verilog hardware description language following coding guide lines [8] in top-down approach.

According to ARM7 data sheet, ARM7 has three pipeline stages. In order to be compatible with AMBA, SYS16TM is implemented in the same pipeline stages.

A. Fetch stage

At the fetch stage, PC increments by 2 instead of 4 in byte addressing memory because of the reduction of instruction width. It means we can use a 2-bit adder instead of 4-bit adder. Besides, PCH latch is added for address expansion of 32-bit. The rest of the design is still the same as ARM7. In this stage, chip area is similar to ARM7.

B. Decode stage

In the original ARM7 architecture, it has an ARM instruction decoder at the decode stage. As fig. 3a shows, the decoder decodes ARM instructions to a set of control signals for execution stage. But for Thumb instructions, they are translated into equivalent ARM instruction by De-compressor module before entering the ARM instruction decoder.

In SYS16TM, we only implement Thumb instruction set. It is unnecessary to translate THUMB instructions into ARM instructions and then use ARM decoder to decode them. So as fig. 3b shows, we remove De-compressor module, and modify the decoder to decode THUMB instructions directly. In this way, the complexity of decoder logic is much lower, and chip area is greatly decreased.

At this stage, the register file is also reduced. The reason why and how it is reduce is explained in section 3.2.

C. Execute stage

At the execute stage, functional units take control signals from the decode stage to execute. Their functionalities are still the same to ARM7, but the bandwidth is reduced to 16-bit including multiplier, shifter and adder.

In the original ARM7, multiply instructions take at most four cycles to complete. The multiplier is based on Booth algorithm [9] that uses a 32 x 8 multiplier internally. The multiplier calculates the result of a 32-bit multiplicand times a 8-bit multiplier at each cycle. According to Booth algorithm, if the upper 24 bits of multiplier are all ones or zeros, it takes only one cycle to complete calculation. But it can also take four cycles to complete in the worst case if the multiplier is not so formatted.

In SYS16TM, we use a 16 x 8 multiplier as core multiplier. Since the data path is reduced to 16-bit, the multiplier takes at most two cycles to complete. The finite state machine logic in ARM7 decoder that is responsible for generating the control signals for third and fourth cycle can be removed. Besides, Thumb instruction set doesn't have MLA instruction like ARM7 dose that sums up the result of multiply with another operand. So the logic of adder is also removed. After all the modification made to multiplier, the gate count of multiplier is greatly reduced to 24% of the original design.

Besides reducing the width of functional unit to 16-bit, certain functional units are removed entirely. Because Thumb instruction set is a subset of ARM instruction set, functional units operate by only ARM instructions are no long need in SYS16TM. For example, coprocessor instructions defined in ARM instruction set is not supported by Thumb instruction set, thus the coprocessor interface is removed in SYS16TM. Other instructions such as MLA, MRS, MSR, RSB, RSC, SWP and TEQ are treated in the same way.

V. Reuse of software tools

In order to reuse ARM7's software environment instead of designing the compiler and the simulator on our own, there are some coding guide lines to be followed. With little modification to coding style, we can reuse ARM7's compiler and simulator as much as possible. The software tools we used here is ARM Developer Suite (ADS) licensed from ARM [10].

A. Coding guide line 1: declare "short" instead of "int" in C language

C compiler treats integer variable as 32-bit and short variable as 16-bit. For example, if a variable is declared as short, compiler uses 16 bits memory space or 16 bits register to store it. Since SYS16TM only has 16-bit width, C compiler

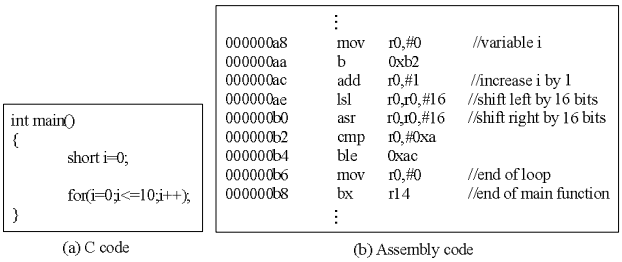

Fig. 4. Example C code and the corresponding assembly code.

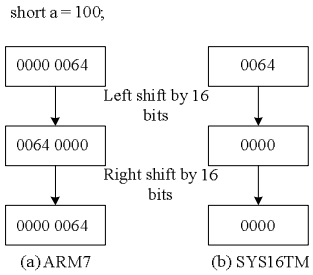

Fig. 5. Effect of the pair of shift instructions on ARM7 and SYS16TM.

```
0x00000000   B     rst_srv     //reset exception
0x00000002   NOP               //non operation
0x00000004   B     undef_srv   //undefined exception
0x00000006   NOP
0x00000008   B     SWI         //software int. exception
0x0000000A   NOP
0x0000000C   B     Abtp_srv    //prefetch abort exception
0x0000000E   NOP
0x00000010   B     Abtd_srv    //data abort exception
0x00000012   NOP
0x00000014   NOP               //reserve
0x00000016   NOP
0x00000018   B     IRQ_srv     //IRQ exception
0x0000001A   NOP
0x0000001B   B     FIQ_srv     //FIQ exception
0x0000001C   NOP
```

Fig. 6. Example of exception vector table of SYS16TM.

TABLE I
Address of exception vectors

Address	Exception
0x00000000	Reset
0x00000004	Undefined instruction
0x00000008	Software interrupt
0x0000000C	Abort (prefetch)
0x00000010	Abort (data)
0x00000014	Reserved
0x00000018	IRQ
0x0000001C	FIQ

must be aware of this limitation so that the registers are not overflow during the execution. In order to tell the compiler to operate on 16-bit, be sure to declare variables as "short". We run several benchmarks to confirm this solution in several ARM compilers such as TCC (Thumb C Compiler) [10] and GNU C compiler [11].

For assembly programming, software programmers can still use the same experience already learned from ARM7. One thing programmer should keep in mind is that the registers are now 16-bit.

B. Coding guide line 2: remove LSL, ASR pairs

By our observation, when variables are declared as "short", the compiler generates a pair of instruction consisting of LSL and ASR instructions. For example, fig. 4b is the assembly code compiled from C code in fig. 4a by thumb compiler. From the assembly code, every time the variable i increases, the pair of shift instructions are added after.

The reason why inserting this instruction pair is to make sure that value is not more than 16-bit. As fig. 5a shows, the 32-bit register storing 100 is left shifting by 16 bits and then right shifting by 16 bits. In the original ARM7 program model, it is necessary for making sure the upper 16 bits of the 32-bit register is zero. But for SYS16TM, registers is only 16 bits, fig. 5b shows this pair of instructions clean the registers to zero.

For SYS16TM, such pair of instructions should be taken care of after compiling. Without modification to thumb compiler, we develop a simple tool to replace the pair of instructions with NOP (No Operation) instructions.

C. Coding guide line 3: keep the exception vector addresses the same

Table 1 shows the address of exception vectors in ARM7. When exception occurs, program jumps to the specific address according to specific type of exception. At each exception vector address, there exists a branch instruction to jump to corresponding exception service routine (srv). In order to be compatible with ARM7 simulator, SYS16TM's hardware is designed in the way that has the same exception vector addresses comparing to ARM7. For software developers, there is one coding guide line to follow. That is after inserting branch instructions at each exception vector address, a NOP instruction should be inserted after each branch instruction. The reason is that the Thumb instruction length is two bytes, but the address space between two exception vectors is four bytes.

The left two bytes space should be inserted by a NOP instruction, so the program can jump to the correct address. Fig. 6 is the example of exception vector table of SYS16TM. Every entry of exception vector has a branch instruction that jumps to the corresponding service routine. Followed by each branch instruction is a NOP instruction.

By following the three coding guide line above, programs compiled from ADS thumb compiler can be executed on SYS16TM. But there is one limitation. It is that the compiler and simulator are not aware of the new defined instructions for accessing PCH, LRH and SPH discussed in section 3.2. Unless the applications are written in assembly code, the address space of SYS16TM is limited to 64 k due to compiler. In the future work, this problem will be solved by developing custom software tools for SYS1TM.

VI. Experimental Result

We implement both ARM7 and SYS16TM in verilog language following nLint design rules [12]. We verify both

designs in RTL level by comparing the result of RTL simulation with ARM7 simulator [13]. The test patterns used to verify both designs can achieve 100% of code coverage measured by VN-cover [14]. Further more, both designs are verified in FPGA.

The benchmarks we use are collected by their features in embedded system such as sort, image, network, math, and others.

A. Cost

We use Synopsys design compiler to synthesize in UMC 0.18 technology. Table 2 is the result after synthesis of both ARM7 and SYS16TM. As we can see, the gate count of SYS16TM is only 40% of ARM7 while the clock frequency is 51% faster than ARM7.

The great reduction of gate count is contributed primarily from the reduction of data path. For example, table 3 is the synthesis result of multiplier and register file of both ARM7 and SYS16TM. After the modification mentioned in section 4.3, the gate count of multiplier is reduced to 24% of the original design.

The other contributors of gate count reduction are the decoder and register file at the decode stage. Table As we can see from table 3, the gate count reduction of register is huge that is only 21% of ARM7 register file.

For the consideration of cost, memory size is also an important issue. Table 4 is the experiment result we obtain from running ARM7 compiler for code size comparison. The result shows that THUMB code size is average 61 % of ARM code size [1]. In other words, the memory size needed for SYS16TM is 39% less than ARM7. So not only the gate count of microprocessor is reduced, but also the required memory size.

TABLE II
Report after synthesis in UMC 0.18 technology

	ARM7	SYS16TM	SYS16TM/ARM7
Gate count	50,612	20232	40 %
Frequency	91 MHz	137 MHz	151 %

TABLE III
Gate count of multiplier and register file

	ARM7	SYS16TM	SYS16TM/ARM7
Multiplier	8800	2128	24 %
Register file	166,46	3453	21%

TABLE IV
Code size of ARM/THUMB instruction

Benchmark	Thumb code size (byte)	ARM code size (byte)	Relative density (Thumb/ARM)
qsort	96	144	0.67
dijkstra	124	244	0.51
Hanoi	88	180	0.49
gcd	34	64	0.53
Fibonacci	48	76	0.63
knight	438	120	1.04
jpeg_encoder	3,728	6,660	0.56
Geometry mean			0.61

B. Power

We use Synopsys PrimePower [15] for power estimation. Table 5 is the result obtained from running the same benchmarks on ARM7 and SYS16TM. The result shows that the power of SYS16TM is only 51% of ARM7.

For the power consumption of memory, we use Artisan's memory generator to estimate the power. The experimental result shows that even if the code size reduction of SYS16TM achieves 61%, the memory power reduction due to smaller memory size is not great. The factor effects the memory power is primarily memory clock frequency.

C. Performance

Because of the simplicity of logic and the reduction of data path, the critical path in SYS16TM is shorter than ARM7, resulting higher clock frequency than ARM7. From table 2, it shows that SYS16TM is 51% faster than ARM7.

Although SYS16TM is faster than ARM7, it suffers from longer execution path. As we discus in section 2, since the size of immediate fields, effective address offsets in Thumb instruction set is smaller than ARM instruction set, SYS16TM could take longer execution path than ARM7 to complete the same program. Table 6 is the number of instruction executed for the seven benchmarks. From the result, SYS16TM has to execute an average instruction of 12% more than ARM7. For embedded systems that concern about low cost/power primarily, the performance degradation is acceptable. Since SYS16TM achieve 60% less gate count, 39% less required memory size and 49% less power consumption, the performance degradation of 12% is tolerable.

D. For 16-bit bandwidth memory

For low cost embedded systems, low bandwidth memory is commonly used. In such systems, the performance and power consumption of SYS16TM can be better than ARM7. Table 7 is the result we obtain from ARM7 simulator. We simulate

TABLE V
Power of ARM7 and SYS16TM

	ARM7	SYS16TM	SYS16TM/ARM7
Average power	5.33 mW	2.7 mW	51%

TABLE VI
Executed instruction count of different benchmarks

Benchmark	# of Inst. (SYS16TM/ARM7)	Inst. Ratio
qsort	60720 / 50707	1.20
dijkstra	1393 / 1340	1.04
Hanoi	21492 / 21489	1
gcd	200 / 161	1.24
Fibonacci	4760389 / 4171137	1.14
knight	91971 / 71223	1.29
jpeg_encoder	1488009 / 1483328	1.00
Geometry mean		1.12

TABLE VII
Cycle count of SYS16TM and ARM7 equipped 32-bit and 16-bit bandwidth memory (ARM-32 : ARM7 with a 32-bit bandwidth memory; ARM-16 : ARM7 with a 16-bit bandwidth memory)

Benchmark	ARM-32	ARM-16	SYS16TM	ARM-32/ SYS16TM	ARM-16/ SYS16TM
qsort	91024	173262	101060	0.90	1.71
dijkstra	2344	4130	2513	0.93	1.64
Hanoi	49210	75732	42989	1.14	1.76
gcd	256	510	295	0.87	1.72
Fibonacci	9428047	16106249	9624465	0.98	1.67
knight	129678	244073	161026	0.81	1.52
jpeg encoder	2361939	4346607	2481146	0.95	1.75
Geometry mean				0.94	1.68

ARM7 and SYS16TM respectively with different bandwidth of memory running at same clock frequency. The result shows that when bus bandwidth is 32-bit, ARM7 only needs 94% cycle count of SYS16TM. But if we consider 16-bit bandwidth memory, ARM7 require 68% more cycle count than SYS16TM. In other words, SYS16TM requires only 60% (1/1.68) cycle count of ARM7. For real time applications, smaller cycle count means the processor can operate at smaller clock frequency. So the power consumption can be lower.

Further, if we consider cycle count with average power, SYS16TM can achieve significant low power consumption. As we can see from table 6, the power consumption of SYS16TM is only 51% of ARM7. Since the total energy for a given application is the average power consumption times the total execution time, the energy consumption of SYS16TM is about 0.51 x 0.60 = 30.6% of ARM7.

VII. Conclusion

In midrange performance embedded system applications, 16-bit microprocessor is sufficient. Especially when considering low bandwidth memory such as 8-bit or 16-bit memory, which comes up often in embedded systems, 16-bit instruction set has greater advantage than 32-bit instruction set. Although there are many 32-bit microprocessors, for example, ARM7 has the capability to execute programs in 16-bit instructions and achieves 39% code size reduction. But for short precision applications, the upper 16-bit of 32-bit microprocessor is rarely used, thus resulting waste of cost and power. By this observation, we propose a microprocessor architecture called SYS16TM that is derived from ARM7 to achieving lower cost/power and high performance. After some optimization of SYS16TM, it requires only 40% cost and 51% power of ARM7. Especially for low cost embedded systems that are usually equipped 16-bit bandwidth memory, SYS16TM takes only 60% cycle count of ARM7 to complete a same program. Thus, the total energy SYS16TM consumes is only 30.6% of ARM7. At last, unlike many microprocessors, SYS16TM still can reuse software environment as much as possible. Although there is one limitation on software tools reuse, in the future work, we will solve this problem by developing compiler and simulator for SYS16TM.

References

[1] "ARM7TDMI Data Sheet", Advanced RISC Machines Ltd., 1995.

[2] Cross, J. E. and Soetan, R. A., "Teaching microprocessor design using the 8086 microprocessor," in Proc. IEEE Conf. on Southeastcon '88, pp. 175-180, April 1998.

[3] Dac pham, et al., "A 1.2 W 66 MHz superscalar RISC microprocessor for set-tops, video games, and PDAs," in Proc. IEEE intl. Conf. on Solid-State Circuits, pp. 180-181, Feb. 1995.

[4] Bannatyne, R., "Migrating from 8- to 16-bit processors," in proc. Northcon /98 conf., pp. 150-158, Oct. 1998

[5] "MIPS32 Architecture for Programmers Volume IV-a : The MIPS16e Application-Specific Extension to the MIPS32 Architecture", MIPS Technologies, Inc., 2005.

[6] Canal, R., Gonzalez, A., and Smith, J.E., "Very low power pipelines using significance compression," in Proc. 33rd IEEE/ACM intl. Symp. On Microarchitecture, pp. 181-190, Dec. 2000.

[7] "PIC16C63A/65B/73B/74B Data Sheet", Microchip Technology Inc., 2000.

[8] Michael Keating and Pierre Bricaud, "Reuse Methodology Manual for System-on-a-Chip Designs. 2nd Edition", KLUWER ACADEMIC PUBLISHERS, 2002.

[9] David A. Patterson, John L. Hennessy, "Computer Organization and Design: The Hardware/Software Interface, Third Edition," Morgan Kaufmann, 2004.

[10] Advanced RISC Machines Ltd., "ARM Developer Suite : Compilers and Libraries Guide", Advanced RISC Machines Ltd., 2001.

[11] GNU, "GNU Compiler Collection", http://gcc.gnu.org/.

[12] NOVAS, "nLint", http://www.novas.com/Products/nLint/

[13] Advanced RISC Machines Ltd., "ARM Developer Suite : AXD and armsd Debuggers Guide", Advanced RISC Machines Ltd., 2001.

[14] TransEDA, "VN-Cover", http://www.transeda.com/products/vn-cover/details.php

[15] Synopsys, "PrimePower", http://www.synopsys.com/products/power/primepower_ds.pdf

[16] Arvind Krishnaswamy and Rajiv Gupta, "Profile guided selection of ARM and thumb instructions," in Proc. the joint conference on Languages, compilers and tools for embedded systems: software and compilers for embedded systems, pp. 56-64, 2002

[17] Bunda, J., Fussell, D., Athas, W.C. and Jenevein, R., "16-bit Vs. 32-bit Instructions For Pipelined Microprocessors," in Proc. 20th Annual International Symposium on Computer Architecture, pp. 237-246, May. 1993