CalmRISC^{TM} : A Low Power Microcontroller with Efficient Coprocessor Interface

Kyoung-Mook Lim, Seh-Woong Jeong, Yong-Chun Kim, Seung-Jae Jeong, Hong-Kyu Kim,
Yang-Ho Kim, Bong-Young Chung, Hyung-Lae Roh, and H.S. Yang
Samsung Semiconductor and KAIST
e-mail: lracer@samsung.co.kr

Abstract

This paper presents the low power architecture of CalmRISC, a low power 8-bit microcontroller consuming only 0.1mW per MIPS at 3.0V, and its efficient coprocessor interface. The architectural consideration of CalmRISC for low power consumption is presented. Some low power circuit design schemes as well as an efficient coprocessor interface scheme in CalmRISC are proposed and discussed.

1. Introduction

As the VLSI process technology advances, three remarkable trends in the competitive MCU industry have been a challenge to MCU (Microcontroller Unit) designers: low power, high performance, and system on a chip.

The power consumption of an MCU has been paid more attention, because it takes a significant portion of the operating power of the system as well as its power consumption becomes dominant in non-operating mode (or sleep mode) of the system.

The competition towards high performance in the MCU industry seems endless. As systems are getting more complex and advanced, system engineers expect an MCU to handle more complex tasks. Especially, it is very attractive to them if they can upgrade their systems by simply changing programs with a better performing MCU, which reduces both the development cost and the system cost itself.

RISC (Reduced Instruction Set Computer) has been known for its high performance. Recently, it has been focused because of its low power characteristics. Due to these facts, it has become a *de facto* standard architecture for low power and high performance MCU and has started to be widely accepted in 8-bit embedded MCUs [1][2].

The SOC (System On a Chip) approach has been very attractive to ASIC designers since cost, power consumption and system design complexity can be drastically reduced. The trend of SOC integration around an MCU core is more conspicuous, due to the fact that an MCU is a mandatory element to almost all electronic appliances and it is much easier to integrate devices around a programmable core. Especially, the integration of an MCU core and a DSP (Digital Signal Processing) core has been focused as the advanced VLSI process technology has made DSP more easily available than ever.

A simple solution to the integration of an MCU core and a DSP core is to put two independent cores on a single die, an MCU core for control tasks and a DSP core for signal processing algorithms. In this simple two-core approach, we can have the flexibility to choose any application specific pair of MCU cores and DSP cores, to fit the target application optimally. However, it suffers from several drawbacks: ill programmability, communication overhead, and hardware overhead.

Another way to support MCU and DSP capabilities on a chip is to use a single processor with both capabilities. For example, an MCU with MAC (Multiply and Accumulate) instructions or a DSP with powerful bit manipulation instructions and branch instructions fall into the category. This approach has some advantages over the simple two-core approach: reduced hardware cost due to no hardware duplication and no communication problem because it is a single processor, which also provides a single development environment. However, it lacks flexibility and extensibility. For example, to add more DSP horse power to the single processor means that more powerful DSP instructions with appropriate data paths should be added, which requires to design almost a new processor.

2. Low Power Architecture of CalmRISC

There are many important criteria in choosing an architecture for MCU cores, such as power consumption, performance, size, and interoperability. Especially, in embedded applications, a chip contains an MCU as well as other components like program ROM, data RAM, and peripheral logic, hence it is crucial to consider an architecture in the context of the whole chip, not just an MCU core. CalmRISC adopts a RISC (Reduced Instruction Set Computer) architecture. Except some 2-

word instructions, all instructions are fixed to 1 word and executed in a single clock cycle. The instruction set of typical RISC architectures is simple and regular. Therefore, control logic and data paths are relatively smaller than those of CISC (Complex Instruction Set Computer), and power consumption and die size can be reduced accordingly. Another advantage of the RISC architecture is that it is easy to utilize a pipeline scheme. With a well-designed pipeline scheme, we can obtain large timing margin for each instruction, low CPI (clock per instruction) near 1.0, and high throughput. A lower CPI or higher throughput MCU performs a given task with a smaller number of clock cycles, hence consumes potentially less power than an MCU with high CPI or low throughput. From the architecture standpoint, one of the most efforts centers around how to reduce CPI, and the RISC architectures have appeared as a natural choice.

As mentioned earlier, the power consumption of an MCU core itself has less meaning unless it is considered with other components in a whole chip. One of such components, which are strongly coupled with an MCU core, is a program memory. If an architecture mandates a larger program memory due to the ill programmability of the instruction set, it increases not only the die size (*i.e.*, cost) but also the power consumption. Hereinafter, we will consider three major instruction set architectures, which are well accepted in the MCU industry : Register-Memory, Memory-Memory, Load/Store instruction sets[3]. To make a fair comparison between the three instruction set architectures, we assume that an ALU instruction takes two operands, *op1* and *op2*, on which the ALU operation is performed, and then the result is stored into *op1*, *i.e.*, $op1 \leftarrow op1 \oplus op2$. CalmRISC has the register-memory instruction set architecture, where only *op2* can be a memory location except store instructions. In the memory-memory instruction set architecture, either *op1* or *op2*, but not both, can be a memory location. In the load/store instruction set architecture, memory locations can be referenced only in load and store instructions.

Table 1. Code size comparison for various instruction set architectures

	Number of instructions		
Operation	Reg.-Mem	Mem-Mem	Load/Store
M1 = M1 + M2	3	2	4
M1 = M2 + M3	3	3	4
M1 = M1 + R1	2	1	3
M1 = M2 + R1	2	2	3
R1 = M1 + M2	2	2	3
M1 = R1 + R2	2	2	2
R1 = R2 + M1	2	2	2
R1 = R2 + R3	2	2	2
Average	1.00	0.89	1.28

<Table 1> compares the number of instructions of the three instruction set architectures for some typical operations, assuming each operation is executed with equal frequency.

In the table, M_i means a memory operand and R_i means an internal register. It is shown that the memory-memory instruction set architecture has the best code density, and the load/store architecture is the worst. However, comparing only the average number of instructions might be misleading without careful consideration of other aspects, such as trade-off between average bits per instruction and addressing modes, and program memory access time.

◆ **Trade-off between average bits per instruction and addressing modes**: The memory-memory instruction set architecture needs memory addressing for *op1* and *op2*. On the other hand, the register-memory instruction set architecture needs memory addressing only for *op2*, and the load/store instruction set architecture restricts memory addressing only to load/store instructions. If addressing modes are the same, the memory-memory instruction set architecture has the largest average bits per instruction, the load/store instruction set architecture has the smallest one, and the register-memory architecture is somewhere in-between. In other words, if average bits per instruction is the same, the load/store instruction set architecture has the most powerful addressing modes, the memory-memory instruction set architecture has the least powerful addressing modes, and the register-memory architecture is in the middle. If an application program is large and needs powerful addressing modes, a memory-memory instruction set architecture MCU requires more instructions for compensating poor addressing modes, and the code size grows rapidly. We can conclude that the memory-memory instruction set architecture has best code density for small size application programs, the load/store instruction set architecture is better for large size application programs, and the register-memory instruction set architecture is good for small or intermediate size application programs.

◆ **Program memory access time**: The access time requirement of the program memory (normally, a ROM) is also an influential factor for the power consumption and the die size of a whole chip. Generally speaking, a shorter access time of a program memory (or a faster memory) results in larger size and more power consumption [4]. Since the CPI of a memory-memory instruction set architecture MCU is about 4 to 20, the clock frequency should be much higher to maintain the same throughput than in a RISC type MCU, which is the case for a register-memory or a load/store instruction set architecture. Therefore, a memory-memory instruction set architecture requires a faster program memory than the other architectures, which, in turn, can increase the die size and the power consumption. Even in the case where a memory-memory instruction set architecture MCU has a relatively small code length, it is not obvious whether the

actual die size and the power consumption are small.
From the observation we have made so far, the register-memory RISC architecture appears the most attractive choice for low-power 8-bit embedded applications, where the program memory is relatively small. Hence, we adopted the register-memory RISC architecture for CalmRISC since the most important design criterion is low power.

CalmRISC has a simple 3-stage pipeline architecture, and <Figure 1> depicts the CalmRISC pipeline structure and other alternative pipeline structures for an MCU with the register-memory instruction set architecture. Since the main design target of CalmRISC is low power consumption, deeper pipeline structures beyond 4 stages are excluded from the consideration.

In the first stage named IF (Instruction Fetch) of the CalmRISC pipeline (<Figure 1>-(a)), instructions are fetched from the program memory and an early decoding (*i.e.*, decoding the fetched instruction partially on the fly) for some instructions is performed. In the second stage dubbed as ID/MEM (Instruction Decode/Memory), fetched instructions are decoded and operands are fetched from the data memory or the register file. In the case that one of the source operands should be from the data memory or the destination is a data memory location, a simple early decoding detects it in the first stage and the required memory address is calculated in the first half of the ID/MEM stage. In the third stage named EX (Execution), an ALU operation and its write-back to the register file are performed. Since the pipeline structure is simple, there is no pipeline stall caused by data dependency. However, there exists a pipeline stall due to control dependency for conditional branch instructions. CalmRISC does not have a branch prediction mechanism in order to avoid power consumption on mispredictions. Due to the branch stall, the CPI of CalmRISC is slightly over 1.0, about 1.1 ~ 1.2, still much lower than that of a typical CISC machine.

<Figure 1>-(b) depicts an alternative 3-stage pipeline. CoolRISC™ adopts the alternative 3-stage pipeline [5]. In the alternative pipeline structure, operations after the IF stage, such as ID, MEM, EX, and WB, are performed earlier by a half cycle than in the CalmRISC pipeline structure. The advantage of the pipeline structure is that the control dependency from branch instructions is totally removed and the CPI is exactly 1.0, which is the ideal value for scalar processors. However, it is noted that the pipeline structure requires a faster program memory. In the second half of the first stage, both program memory access and instruction decoding should be done, and therefore it requires about two times faster program memory than CalmRISC. Unless the clock is sufficiently slow, the requirement may cause power or speed problems. In general, the power consumption of a program memory is comparable to or more than that of an MCU core in typical MCU products, and a two times faster memory consumes 2 ~ 4 times more power [4]. Since the program memory should be accessed at least once for an instruction, the program memory speed requirement results in 2 ~ 4 times more power consumption than the MCU core. If a faster program memory is not affordable for some reasons, it is inevitable to slow down the clock frequency (or throughput) to the half.

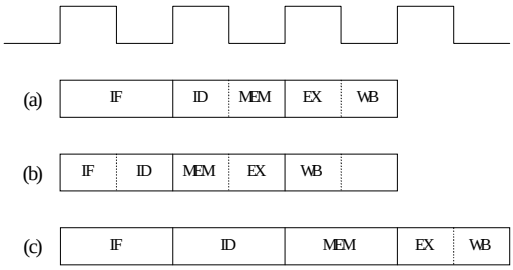

Figure 1. Pipeline structures : (a) CalmRISC 3-stage pipeline (b) Alternative 3-stage pipeline (c) Alternative 4-stage pipeline

<Figure 1>-(c) presents an alternative 4-stage pipeline. It is a deeper pipeline version of CalmRISC, where the second stage ID/MEM of CalmRISC pipeline is divided into two separate stages. At a first glance, the timing constraint of data address calculation and data memory seems to be relieved, compared to that of the CalmRISC pipeline (<Figure 1>-(a)). But, due to data dependency of registers from previous instructions, the data address calculation time and data memory access time is practically the same as that of the CalmRISC pipeline. Only the instruction decoding time margin is increased. Since the instruction decoding time is not on the critical path in the CalmRISC pipeline, there is little gain of timing margin in the alternative 4-stage pipeline over the CalmRISC pipeline. Since a conditional branch needs a 2-cycle stall, CPI of the alternative 4-stage pipeline is about 1.2 ~ 1.5, which means throughput degradation.

From the observations we have made so far, we can safely conclude that the CalmRISC pipeline outperforms the other alternatives in terms of power consumption.

3. Low Power Circuit Design in CalmRISC

The dynamic power consumption of CMOS logic is described as follows[6].

$$P = \sum_i \alpha_i C_i V_{dd}^2 f$$

The power reduction by lowering V_{dd} maintaining the same performance is mainly related to process technology. From the architectural standpoint, most of design efforts go to reducing CPI(Clock Per Instruction), hence the operating frequency *f*. CalmRISC achieves relatively low

CPI (1.1 ~ .12) by adopting Harvard RISC architecture.

To reduce the switching capacitance $\alpha_i C_i$, smaller cells are used whenever possible (e.g., on non-critical paths) and unnecessary transitions are avoided by a smart use of signal gating. Especially, the clock signals should be paid attention to because not only the clock signals have a larger routing capacitance but also their transition frequencies are much higher. Hence clock gating schemes are extensively used in the CalmRISC design [7]. An example is the 12 bit increment logic in PAGU (Program Address Generation Unit) in CalmRISC. For low power consumption of the increment logic, a ripple carry adder is used. The clocks of m LSB bits make a transition every cycle, and the clocks of the remaining (n-m) MSB bits are gated by carry output of m-th half adder. The clocks of (n-m) bits are toggled only when the carry output is 1. From simple mathematics, the power consumption can be minimized when m = 3. By the use of the clock gating scheme, we can reduce about 70% of the clock power consumption in PAGU.

In addition to the clock gating scheme discussed in the above, what CalmRISC uses extensively for power reduction is a selective input latching technique. Its basic idea is that any change of inputs to a combinational logic can be safely blocked by making those inputs filtered through latches, if the output changes of the combinational logic due to the input changes are not used by other logics. In CalmRISC, the selective input latching technique is especially effective to reduce power consumption in combinational logic between pipeline registers. In DAGU (Data Address Generation Unit), only when the memory address calculation is required, data are loaded in the input latches of the DAGU adder.

The power consumption from the short circuit current in CMOS logic is a significant portion of the total power consumption, especially when logic cells with minimum feature size are used. To reduce the short circuit current, a careful in-place optimization, where a weak gate is replaced by a stronger one with equivalent function, is iterated.

4. Coprocessor Interface in CalmRISC

The interface of a coprocessor with a host processor requires the following aspects should be scrutinized: synchronization of the host and the coprocessor executions, instruction issues to the coprocessor, data transfer between the host and the coprocessor, and flagging of the coprocessor status to the host processor.

The synchronization scheme between processors in a multi-processor system should be carefully designed to prevent resource conflicts, false data dependency, and deadlocks. In CalmRISC, synchronization overhead can be relieved since the coprocessors are passive (or non-autonomous) in a sense that they perform only the designated task at a designated time (or cycle) by the host (CalmRISC).

CalmRISC only provides a set of generic coprocessor instructions, and its instantiation to a specific coprocessor instruction set can differ from a coprocessor to another.

A coprocessor instruction is fetched and early decoded by CalmRISC. Once it is identified as a coprocessor instruction, CalmRISC indicates to the coprocessor the appropriate command through the coprocessor interface signals. Then the coprocessor performs the designated tasks at ID/MEM and EX stage. Hence IF from CalmRISC and then ID/MEM and EX from the coprocessor constitute the pipeline for the instruction.

Data transfers are accomplished efficiently through a single shared data memory. The coprocessor in CalmRISC accesses the shared data memory only at the designated time by CalmRISC at which CalmRISC is guaranteed not to access the data memory, and therefore there is no contention over the shared data memory. Another advantage of the proposed scheme is that the coprocessor can access the data memory in its own bandwidth. The only direct data transfer involving CalmRISC and the coprocessor is via CLD (coprocessor register load/store instruction) instructions.

Coprocessors of CalmRISC are passive, that is, do not have their own programs. To perform a branch operation according to a status of a coprocessor (which may be an outcome of a coprocessor instruction execution), CalmRISC conditional branch instruction directly refers to the value of some input signals where the coprocessor has put its status.

References

[1] CalmRISC Technical Reference Manual, Samsung Electronics Co., Ltd., 1999.

[2] “AVR Enhanced RISC microcontroller”, data book, May 1996, Atmel Corporation.

[3] M. Flynn, “Computer Architecture”, Jones and Bartlett publishers, 1995.

[4] 0.5μm 5V/3.3V Standard Cell Library Data Book, Samsung Electronics Co., Ltd., 1996.

[5] C. Piguet *et al.*, “Low-Power Design of 8-b Embedded CoolRisc Microcontroller Cores”, IEEE J. of Solid-State Circuits, Vol. 32, No 7, July 1997, pp. 1067-1078.

[6] A. Bellaouar, M. Elmasry, “Low-Power Digital VLSI design Circuits & Systems”, Kluwer Academic Publishers, 1995.

[7] M. Alidina, J. Monteiro, S. Devadas, “Precomputation-Based Sequential Logic Optimization for Low Power”, Proc. IEEE/ACM International Conf. On CAD-94, Nov 1994, pp. 74-81.