CalmRISC™-32: A 32-BIT LOW-POWER MCU CORE

Sangyeun Cho, Sanghyun Park, Sangwoo Kim, Yongchun Kim, Seh-Woong Jeong, Bong-Young Chung, Hyung-Lae Roh, Chang-Ho Lee*, Hun-Mo Yang*, Sung-Ho Kwak*, and Moon-Key Lee*

MCU Team, System LSI Business, Samsung Electronics Co., Yong-In, Korea
*Dept. of Electrical and Computer Engineering, Yonsei University, Seoul, Korea

Abstract
Architecting today's embedded processor core faces several important design challenges: low power, high performance, and system-on-a-chip considerations. Moreover, support for high-level language constructs and operating systems becomes increasingly critical for acceptance to various applications. CalmRISC™-32 effectively meets these challenges by incorporating a carefully designed instruction set, an energy-efficient pipeline design, debugging support with trace mode/CalmBreaker™ (an in-circuit debugger), and a generic, yet efficient coprocessor interface. Using a 0.25㎜ static CMOS standard cell library and compiled datapath cells, the first implementation of CalmRISC™-32 operates at 130MHz (under worst conditions) and consumes 150㎃/MHz at 2.5V. This paper presents a brief description of the instruction set, the overall microarchitecture, and the coprocessor interface of CalmRISC™-32.

I. INTRODUCTION

Need for low-power high-performance embedded processor cores has grown at a very fast pace in recent years. Proliferation of battery-powered portable electronic devices and their requirements for more computing performance are pushing this trend even further. It also becomes more important to incorporate into a processor core provisions for efficiently building a highly integrated system (system-on-a-chip) based on it. Whether or not a processor core satisfies certain criteria set by such application requirements can greatly enhance or diminish the core's usability. On the other hand, it can be a challenging and time-consuming task to design a new processor core with all these considerations well addressed.

Low-power design calls for significant efforts in all the levels of a design flow – from early architecture design stage down to target process, including microarchitecture design, logic design, and circuit/layout implementation. Software tools at each design stage should provide a projection or hint on power consumption, and (help) generate a low-power equivalent of the design. Besides, power-aware software development is also of growing importance. Recent low-power MCU cores reportedly consume sub-mA per MHz [7,9]

Performance-hungry portable devices are emerging rapidly. Various wireless communication devices like cellular phones, personal information systems like PDAs, and portable game stations all get more intelligent and complex, which is achievable only through a higher operating frequency (currently up to around 100MHz, or higher for future applications) or more work done per cycle. In many situations, the high performance goal conflicts with the low power requirements.

With the advance of process technology, system-on-a-chip (SoC) designs have appeared very attractive to system engineers and ASIC designers, which reduce cost, power, as well as board size. One of the most conspicuous SoC approaches is the merge of an MCU core and special-purpose functional blocks. Especially, when such functional blocks are programmable, an efficient programming environment like MDS (Microprocessor Development System) is crucial for the acceptance of the MCU core. Hence an MCU architecture in terms of SoC support should be understood not only from the hardware standpoint, but also from the software standpoint.

All these aspects motivated us to develop CalmRISC™-32, a new 32-bit general-purpose embedded processor core. It was designed for both low power and high performance: An initial implementation using a 0.25μm static CMOS standard cell library and compiled datapath cells operates at 130MHz under worst conditions, and consumes 150μA/MHz at 2.5V. Moreover, it features a generic yet efficient interface to customizable coprocessors, which greatly enhances its applicability. This paper focuses on introducing the key aspects of CalmRISC™-32, by discussing the instruction set architecture, the microarchitecture, and the coprocessor interface.

The rest of this paper is organized as follows. First, an overview of the CalmRISC™-32 instruction set is given in Section 2. Then we outline in Section 3 the microarchitecture of CalmRISC™-32, *i.e.*, datapath organization, pipeline design, etc., followed by Section 4, a discussion of the coprocessor interface upon which a DSP coprocessor and an FPU coprocessor have been successfully built. Lastly, Section 5 summarizes the paper.

II. INSTRUCTION SET ARCHITECTURE

Three design considerations affected the overall shape of the CalmRISC™-32 instruction set, which is a 32-bit Load/Store RISC architecture. First of all, low power requirements set important guidelines. Instruction width was determined to be 16 bits, which, compared with a 32-bit equivalent, leads to much smaller code size with little performance degradation [4]. A large portion of fetch-related power can be saved as

well. Accordingly many recent embedded cores implement a 16-bit instruction set or a special mode in which a 16-bit instruction format is used (on top of an existing 32-bit instruction set) [1,6,8,10]. A caveat is that the expressive power of instructions is inevitably impaired and so is the orthogonality of the instruction encoding, thus making the instruction decoder possibly have longer latency. Instruction grouping and placement on the instruction map was done in a way that facilitates easy pre-decoding (See Section III) and lowers bit transitions between consecutive instructions. Second, it was assumed that the users would mainly use high-level languages, such as C or C++, to program CalmRISC™-32. Among others, the addressing modes of memory access instructions were made powerful. Lastly, controller-oriented instructions and some other features unique to previous CalmRISC™ cores [7,5] were again included in the instruction set to minimize the users' effort of migrating to this newer core and to ease software porting.

Register View There are two logical general-purpose register banks, each containing sixteen 32-bit registers (from `r0` to `r15`). In *user mode*, only one bank is accessible. In *privileged mode*, however, all 32 registers in two banks can be accessed, by properly setting register selection bits in the *processor status register* or `sr`. Two registers, namely `r14` and `r15`, are reserved for saving the return address on procedure calls and for the stack pointer, respectively. There is a separate special register file used for saving the program counter (`PC`) and `sr` on various asynchronous exceptional conditions. Transferring data to and from the special registers is a privileged operation and can not be performed in user mode.

Use of "T" Bit "T" (test) bit is a bit stored in `sr` that is set or reset by compare instructions and certain arithmetic instructions. A conditional branch refers to this bit to determine whether or not it is taken and change the control flow of the program. Some arithmetic instructions take this bit as an implicit input, *e.g.*, `adc` (add with carry) and `sbc` (sub with carry) use the T bit as the carry-in. This bit is also used to execute or cancel conditional instructions, such as `inct` (increment if true) and `dect` (decrement if true).

Memory Access Instructions Two addressing modes are supported for 32-bit word, 16-bit half-word, and byte accesses: register-displacement (scaled) and register-register. There are separate stack-access instructions, `push`, `pop`, `pushq`, and `popq`. `pushq` and `popq` are used to push or pop four registers in sequence. Besides, program memory can be accessed with a single register pointer.

Arithmetic and Compare Instructions Various ALU operations, like `add`, `sub`, `adc` (add with carry), `sbc` (sub with carry), `and` (logical AND), `or` (logical OR), `xor` (logical XOR), `tst` (bit test), and `div1` (single-step division), are provided. Barrel shifter instructions include `sl` (shift left), `sr` (shift right), `sra` (shift right arithmetic), `rr` (rotate right), `rl` (rotate left), and `rrc` (rotate right with carry). For efficient multiplication, a pipelined, 2-cycle latency 16x16 multiply instruction is offered. Compare instructions, such as `cmp` `eq`/`ge`/`gt` (equal/greater than or equal to/greater than) or `cmpu` (compare unsigned) `ge`/`gt` set or reset the T bit according to the result of comparing two signed or unsigned numbers.

Control Instructions Control-changing instructions jump to a program location whose address is either computed using the current `PC` (called "PC-relative") or assigned with a value stored in a register. Conditional branches look at the T bit to determine if the branch is taken. Similarly, `brec` (branch on external condition) is taken if the specified external condition is true. There are four external condition signals fed into the CalmRISC™-32 core, typically driven by a coprocessor. Branches by default have a single delay slot following them. For most frequently executed short conditional branches, a non-delayed version is also provided to save the code size in case no instructions are available to fill a delay slot. Some other branches, especially related to exception handling, are also devoid of a delay slot due to different reasons.

Controller-Oriented Instructions There are instructions that directly manipulate a bit in a memory location: `bits` (bit set), `bitr` (bit reset), `bitc` (bit complement), and `bitt` (bit test). After executing these instructions, the specified bit in memory is set, reset, or complemented accordingly. In addition, they update the T bit based on the original value of the specified bit, and thus can be used as *test-and-set* (`bits`) and *test-and-reset* (`bitr`) primitives in order to implement semaphores. A byte-access instruction with a large, 8-bit displacement is provided so that access to a group of memory-mapped ports can be made efficiently using a single base register.

Coprocessor and System Instructions For coprocessor instructions, 13 bits (on the 16-bit instruction map) were allocated. Except for the data transfer instructions, these instructions are not executed in the core. Fetched coprocessor instructions are passed to the coprocessor via `COPIR` core pins. See Section 4 for detailed explanation. There is a privileged system instruction called `sys`, to issue a 5-bit system command, *e.g.*, to shut down the system clock.

Other Data Transfer Instructions Instructions are provided to transfer a value in a register to and from another general-purpose register, a special-purpose register, and the current PC. Also provided are instructions to assign an immediate value to specific bits in `sr`.

III. MICROARCHITECTURE

CalmRISC™-32 is a pipelined processor that follows the Harvard architecture. With separate instruction and data memory interfaces, most instructions are executed in a single cycle, except for a few multi-cycle instructions such as `pushq`/`popq`, 16-/32-bit immediate load instructions, and the controller-oriented `bit*` operations. Classical five stages constitute the pipeline: **Fetch** (instruction fetching) – **Decode** (instruction decoding and operand reading) – **Execute** (instruction execution and address calculation) – **Memory** (memory access) – **Writeback** (writing result to register file). Data dependencies between instructions in the pipeline are automatically enforced by the hardware interlock and forwarding mechanism.

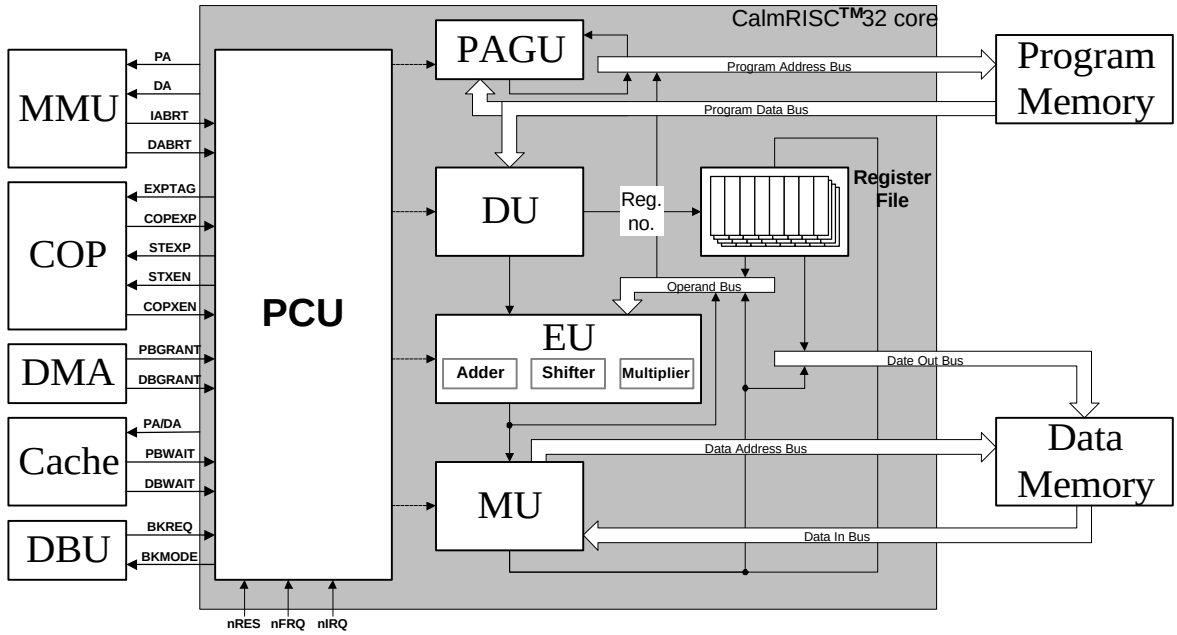

Figure 3.1 CalmRISC™-32 Datapath.

Figure 3.1 briefly shows the CalmRISC™-32 datapath. The datapath contains a 32-bit adder, a 32-bit barrel shifter, and a 16x16 multiplier for various computations. In our first prototype implementation, these datapath macro blocks were compiled, and all the other logic blocks used standard cells based on a 0.25μm technology [2].

The general-purpose register file contains thirty-two 32-bit registers, divided into four sets of eight registers. Depending on the current processor mode (user or privileged) and the register selection bits in `sr`, two out of the four sets (thus 16 registers) are selected for use. This hardware structure allows the previously described logical register view.

Two-Phase Clocking We used the two-phase clocking scheme for low-power circuit implementation. `ICLK`, the internal clock that drives the core, is first distributed to each pipeline stage via a buffered clock tree. Then each stage locally generates phase-1 and phase-2 clocks. Most local clocks are gated clocks, and they are activated only if the downstream logic needs re-evaluation, preventing unnecessary bit transitions to propagate downwards. We have also applied *slack borrowing* [3] to secure an enough timing margin in certain cases, especially in the Decode and Execute stages. Due to this clocking scheme, much care was paid to ensuring that signals that stem from a phase would not cause timing violations in the next phase registers. All the pipeline registers are implemented with latches rather than flip-flops.

Pre-Decoding of Instructions Fetched instructions are pre-decoded before they are latched in the IR (Instruction Register) in the Decode stage. The pre-decoder looks at three most significant bits of a fetched instruction, and decides if it is a coprocessor instruction, a branch, or else. Depending on the result, the instruction is processed by distinct decoders. For example, if the instruction is a branch, it is *not* latched in the IR, but in a separate latch in PAGU (see below).

Program Address Generation Unit (PAGU) Spanning from the Fetch stage to the Decode stage, PAGU contains a set of registers and adders to store and compute the next program address. When it is known that a branch instruction is fetched, the instruction is *not* latched and processed in the *Decoder Unit* (DU), but in PAGU. This separate processing of branch instructions saves much decoding power as branches appear frequently in a typical program. Another low-power technique used in PAGU is to use dual clocks for PC registers [7]. The technique is based on the observation that upper bits of PC are less frequently updated than lower bits as a sequence of instructions are executed. Therefore, the clock signal used for these upper bits can be held most of the time without causing any harm.

Decoder Unit (DU) DU forms the major part of the second pipeline stage. Instructions are decoded, and necessary control signals are generated and propagated to PCU, PAGU, and the Execution stage. Moreover, the necessary operands are read from the register file, or obtained from EU or MU through the forwarding mechanism if the most up-to-date operand values are found there.

Execution Unit (EU) EU implements the Execute stage of the pipeline. Most instructions are executed in this unit. Results are produced and passed to the Memory stage, and bypassed to DU if the result is to be used immediately by the following instruction in DU. To save power consumption, large datapath cells, namely adder, shifter, and multiplier, are driven by separate input latches. If the instruction in EU is a memory store operation, the operand to be stored is read in this stage.

Memory Unit (MU) MU interfaces to the data memory. It issues the data address and the related indication signals, such as `NDMCS` (data memory chip select), `DMWR` (data write) `DSIZE` (data size), etc. Processing of `bit*` operations is

also done in MU.

Pipeline Control Unit (PCU) PCU receives various pipeline status signals from each pipeline stage and from outside the core, and generates pipeline control signals. Each pipeline stage, using the control signals, decides to either proceed or stall.

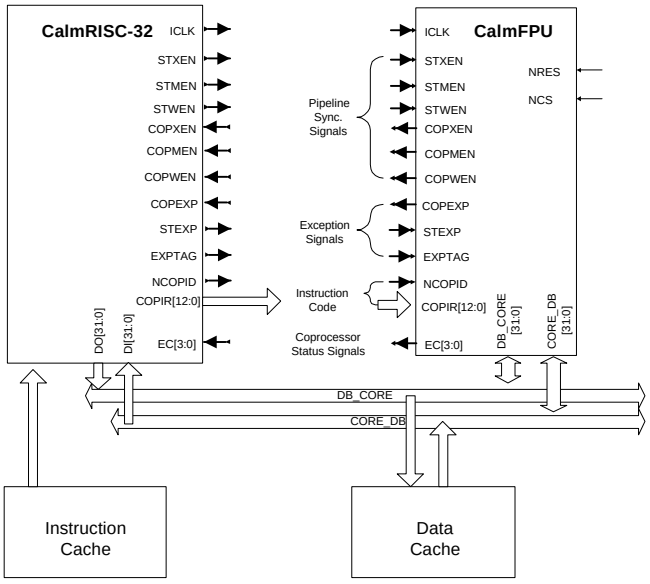

Figure 4.1 CalmRISC™-32 Coprocessor Interface.

IV. COPROCESSOR INTERFACE

The CalmRISC™-32 coprocessor interface is for a *passive* coprocessor, which brings instructions from the core instead of fetching them from memory with its own program counter. Coprocessor instructions are fetched and pre-decoded by the core, and passed to the coprocessor via `COPIR[12:0]` signals with a notification `NCOPID`. 13-bit coprocessor instructions, denoted "`cop imm:13`" in a generic form, are not executed in the core except coprocessor load/store instructions named `cld`. They can be categorized into two types of operations: (i) data transfer between the core and the coprocessor, and (ii) data transfer between the coprocessor and memory, managed by the core. In the former, transfer is between general-purpose registers in word. In the latter, also word transfer, three addressing modes are supported: register-displacement, register-register, and register pre-/post-increment/decrement. These `cld` instructions use global data bus as shown in Figure 4.1. CalmRISC™-32 core generates necessary data memory signals and bus requests.

A CalmRISC™-32 coprocessor accesses memory at the same pipeline stage as the core does, with help from a set of pipeline synchronization signals. CalmRISC™-32 and a coprocessor may be stalled for any reason, and two processors should be synchronized for the `cld` instructions or on exceptions. Two processors communicate their pipeline status with each other via the signals `STXEN`, `STMEN`, `STWEN` (from core to coprocessor), `COPXEN`, `COPMEN`, and `COPWEN` (from coprocessor to core). Each processor decides if a pipeline stage will advance to the next stage after consulting these signals. Figure 4.2 shows how a stall in the core causes a stall in CalmFPU™, an FPU coprocessor.

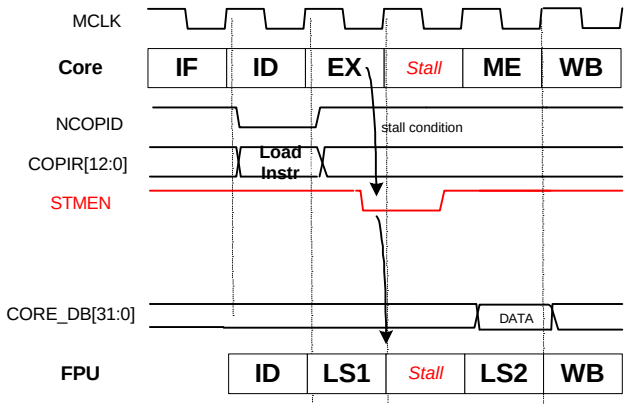

Figure 4.2 Pipeline Synchronization on Stalls.

When CalmRISC™-32 gets a data abort exception, *e.g.*, TLB miss, it is required that both the core and the coprocessor flush the instructions from the aborting instruction. `EXPTAG` and `STEXP` signals are used to ensure that this flushing action is taken correctly and to maintain the precise exception model. `EXPTAG` indicates that an instruction, which may cause an exception, is now entering the execution stage, and `STEXP` indicates that an exception actually occurred. On the other hand, if the coprocessor generates an exception, *e.g.*, floating-point overflow, underflow, and inexact exception, the core should save the `PC` of the faulting instruction and flush the following instructions. In this case, a coprocessor uses the `COPMEN` signal to stall the instruction in the core that corresponds to the potentially faulting instruction in the coprocessor. Figure 4.3 shows the situation. After stalling the core with `COPMEN`, the coprocessor asserts the `COPEXP` signal to flag a real exception.

Lastly, it is noted that the core always controls the program flow. For the core to branch based on the coprocessor state, *e.g.*, after an FPU instruction generates an outcome, a branch instruction called `brec` directly refers to the value of `EC[3:0]` signals driven by the coprocessor, as described in Section 2.

Besides the FPU coprocessor used as an example in this section, a DSP coprocessor called CalmMAC™-2424 has been designed and successfully integrated with CalmRISC™ [5]. They were then used to build a low-power digital music player chip, running MP3 decoder codes as an application.

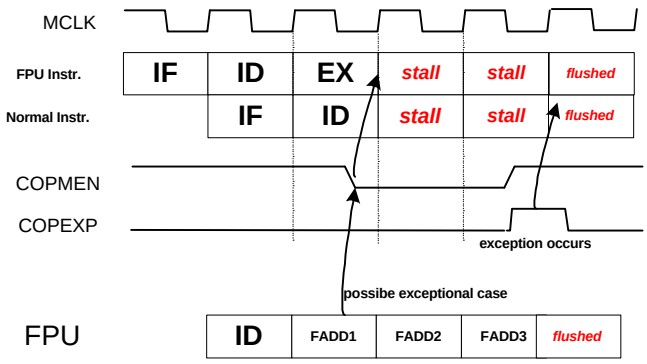

Figure 4.3 Pipeline Synchronization on Exceptions.

V. CONCLUDING REMARKS

CalmRISC^{TM}-32 is a RISC-style Load/Store architecture, implemented with various low-power techniques. In order to minimize the CPI (Clocks Per Instruction), the Harvard architecture is adopted. Its instruction width is set to be 16 bits, which alleviates the code density problem inherent in the RISC architecture. All the instructions are a single 16-bit word, except a few instructions with long immediates. Furthermore, most frequently executed branch instructions come both with and without a delay slot.

To better support SoC (system-on-a-chip) design, CalmRISC^{TM}-32 provides a generic, yet powerful coprocessor interface. With the inter-processor synchronization support, the core and a coprocessor efficiently communicate data and control.

An initial implementation of CalmRISC^{TM}-32 using a 0.25μm process runs at 130MHz under worst conditions and consumes about 150μA/MHz. The gate count of the core is around 31,000, and the die size is about 1.02mm^2.

Currently, we are designing customized datapath cells to replace the current compiled datapath cells for more speed. The design will be migrated to a more advanced 0.18μm process technology in a near future to offer more performance at even lower power consumption. Extensive software infrastructure, including development environment such as compilers, debuggers, and various libraries, and real-time operating systems are under development at the same time.

Thanks to its low power, high performance, and SoC support with various coprocessors, CalmRISC^{TM}-32 is poised to be an ideal MCU core for current and future mobile applications as well as general applications.

REFERENCES

[1] ARM7T RISC Processor, Advanced RISC Machines Ltd., 1995.

[2] ASIC STD110 0.25μm 2.5V CMOS Standard Cell Library Databook, Samsung Electronics Co., March 2000.

[3] K. Bernstein, K. M. Carrig, C. M. Durham, P. R. Hansen, D. Hogenmiller, E. J. Nowak, and N. J. Rohrer. *High Speed CMOS Design Styles*, Kluwer Academic Pub., 1998.

[4] J. Bunda, D. Fussell, W. C. Athas, and R. Jenevein. "16-bit vs. 32-bit Instructions for Pipelined Microprocessors," *Proc. Int'l Symp. Computer Architecture (ISCA)*, pp. 237 – 246, May 1993.

[5] H.-K. Kim, K.-M. Lim, Y.-C. Kim, S.-W. Jeong, E.-M. Kim, Y.-H. Kim, B.-Y. Chung, and H.-L. Roh. "A Customizable DSP Coprocessor Extension in CalmRISC^{TM} Microcontroller," *Proc. Int'l Conf. Signal Processing Applications & Technology*, November 1999.

[6] K. D. Kissel. "MIPS16: A High-density MIPS for the Embedded Market," Silicon Graphics MIPS Group, 1997.

[7] K.-M. Lim, S.-W. Jeong, Y.-C. Kim, S.-J. Jeong, H.-K. Kim, Y.-H. Kim, B.-Y. Chung, and H.-L. Roh. "CalmRISC^{TM}: A Low Power Microcontroller with Efficient Coprocessor Interface," *Proc. Int'l Conf. Computer Design (ICCD)*, pp. 299 – 302, October 1999.

[8] B. Moyer and J. Arends. "RISC Gets Small." *Byte Magazine*, February 1998.

[9] J. Scott, L. Lee, J. Arends, and B. Moyer. "Designing the Low-Power M•CORE Architecture," *Proc. Int'l Symp. Computer Architecture Power-Driven Microarchitecture Workshop*, pp. 145 – 150, July 1998.

[10] SuperH SH-4 Hardware Manual, Hitachi Inc., 1998.