

Low-Power Design of an Embedded Microprocessor Core

J.-M. Masgonty, C. Arm, S. Durand, M. Stegers, T. Schneider, C. Piguet
CSEM Centre Suisse d'Electronique et de Microtechnique SA
Neuchâtel, Switzerland

Abstract

Low-power consumption has emerged as a very important issue in the design of integrated circuits in CMOS technology. The basic idea behind low-power RISC-like architectures is to reduce the number of executed instructions and clock cycles for the execution of a given task. In addition to these architectural issues, important power savings have been obtained by lowering the supply voltage, by pipelining, by adopting gated clock techniques as well as by using hierarchical memories.

1. Introduction

Low-power CMOS microprocessors and microcontrollers are the key to the realization of portable products in which the power consumption is a more important issue than other performances. Low-power microprocessor cores embedded in ASICs are more and more required in various applications. The design of the low-power microprocessor presented in this paper has been addressed at the behavioral, architectural, logical and physical levels. At the behavioral level, the instruction set has been chosen to minimize the number of executed instructions for a given task. At the architectural level, the number of clock cycles per instruction has been reduced. At the logical level, low-power techniques such as gated clock have been implemented. At the physical level, a low-power cell library [1] has been used, resulting in very good performances.

2. Reduction of the number of instructions to perform a task

Low-power microprocessors have to be programmed in a "low-power programming style". This can be achieved by the minimization of the number N of instructions necessary to execute a given task. The μP is therefore in the idle state for a longer period, or the basic frequency f can be reduced as well as V_{dd} . Power is significantly reduced.

Some "programming" rules can be given. For instance, subroutines called only one or two times could be inserted directly in the program code while removing the CALL and RETURN instructions. It results in the reduction of the number of executed instructions. Small loops executed a small number of times could be unrolled to reduce the number of executed instructions while removing the loop counter (see Table I). These examples illustrate a general rule : less sequentialization in the software, at the price of more instructions in the program memory.

Routine	CoolRisc 88				PIC16C5x [2]	
	nb instr	nb exec code	nb instr	nb exec instr	code	instr
8-bit multiply linear		26		26	35	37
8-bit multiply looped		14		56	16	71
16-bit multiply linear	127		127		233	
16-bit multiply looped	31		170		333	

Table I shows the number of instructions in the code as well as the number of executed instructions for a $N*N$ multiplication with a $2*N$ result.

The instruction set has to be designed to reduce the number of executed instructions to perform a task. The 8-bit core contains 22 generic instructions and 129 assembly instructions (Figure 1). A very useful instruction to implement software multiplication is provided as CADS $r1, r2$. If the zero flag $Z=0$, $r1:= SHR(r1+r2)$ else $r1:= SHR(r1)$, executed in one clock cycle. Table I shows a comparison with an existing 8-bit microprocessor.

CoolRisc Core 88	
JUMP t<3>, addr<13> if t<3> = TRUE then PC0<13> := addr<13>	Immediate DATA Operations r<3>, data<8> r<3> := data<8> alu_op r<3>
Indexed JUMP t<3> if t<3> = TRUE then PC0 := ROMIdx	Bit Operations r<3>, bit<3> r<3>(bit<3>) := OP (test, set, clear, inv)(r<3>(bit<3>))
CALL addr<13> PC0<13> := addr<13>, PC1 := PC0+1, PC2 := PC1...	REG-REG ALU Operations r1<3>, r2<3> r1<3> := r2<3> alu_op r1<3>
Indexed CALL PC0 := ROMIdx, PC1 := PC0+1, PC2 := PC1...	RAM-REG ALU Operations r<3>, addr<8> r<3> := RAM(addr<8>) alu_op r1<3>
RETURN PC0 := PC1, PC2 := PC1...	Indexed RAM-REG ALU Op. r<3>, offset<8> r<3> := RAM(RAMIdx CC offset<8>) alu_op r<3>
RETURN from Interrupt PC0 := PC1, PC2 := PC1... + Restore flags	RAM Store r<3>, addr<8> RAM(addr<3>) := r<8>
Software CALL addr<13> PC0<13> := addr<13>, ROMIdx := PC0	Indexed RAM Store r<3>, offset<8> RAM(RAMIdx CC offset<8>) := r<3>
Indexed Software CALL PC0 := ROMIdx, ROMIdx := PC0	SET μP Frequency (Ck μ P := CkIn/2 ^f)
Software RETURN PC0 := ROMIdx	HALT (Ck μ P := 0)
	NOP
	Start ROM TEST
	End ROM TEST

Figure 1 Generic Instruction Set

3. Low-power Architectures

The choice of a Von Neumann or Harvard architecture is very important. The Von Neumann architecture contains only one memory (or one cache memory) for both instructions and data. It results in a high sequentialization of the instruction execution (and a larger number of Clocks Per Instruction or CPI), as the instruction fetch has to be performed before the operand fetch via the same bus. This is called the Von Neumann bottleneck. The Harvard architecture has separate address and data units, and their management can be performed in parallel. A Harvard architecture is clearly required to save power.

At the architecture level, RISC architectures result in less clock cycles (CPI) executed to perform an instruction [3]. The one-word instruction does not need several memory accesses as it is the case for multi-bytes instructions. The number of data memory accesses is limited while using a register bank instead of one or two accumulators. The CoolRisc 88 core contains a register bank with 8 registers and the CoolRisc 816 provides 16 registers.

An important design parameter in the design of μ P architectures is the number of Clocks Per Instruction (CPI). Architectures with a small CPI are more powerful, can execute more instructions in a given time, or can be supplied with a reduced V_{dd} while maintaining the same performances. The most powerful technique to achieve a low CPI is a pipeline architecture. The basic principle of a pipelined μ P is to execute N-cycles instructions in an overlapped fashion in a N-stages pipeline [4]. At each cycle, an instruction is provided to the pipeline, and an instruction is completed. The CoolRisc is a 3-stage pipelined core (Figure 2).

Pipeline present hazards that result in load or branch delays that slow down the execution. Data hazards occur when an instruction has to wait the result of the preceding instruction that is still in the pipeline. Branch delays occur when a conditional branch has to wait on the condition provided by the preceding instruction. No such delay can occur in the CoolRisc core. Data of the preceding instruction is bypassed to the next one if needed. Furthermore, the branch instruction is executed in only one clock cycle and the condition provided by the preceding instruction is always available. It results in a strictly CPI=1. It is not the case of other 8-bit microprocessors (see Table IV). Tables II and III show that the reduction of the number of Clocks Per Instruction is the key to high performances [5]. The CoolRisc with a CPI=1 outperforms the PIC16C5x with

CPI between 4 and 5, the MC68HCxx with CPI=4 and the 8048 with CPI=15.

μP	8-bits synchrone transmission		
	instr. code	exec. instr.	exec. clocks
8048	14	91	1365
MC68HCxx	20	133	532
PIC16C5x	11	74	296
CoolRisc 88	10	58	58

Table II Clocks Efficiency

routine	PIC16C5x nb clocks	CoolRisc 88 nb clocks
8-bit multiply linear		148
16-bit multiply looped	1332	170
16-bit division looped	1492	233
Floating-point addition	1072	212

Table III Number of Clocks

4. Low-Power Techniques

The gated clock technique has been extensively used in this design. Figure 2 shows the ALU with input and control registers that are loaded only when an ALU operation has to be executed. During the execution of another instruction (branch, load/store), these registers are not clocked thus no transitions occur in the ALU. This reduces the power consumption. A similar mechanism is used for the instruction registers, thus in a branch, which is executed only in the first pipeline stage, no transitions occur in the second and third stages of the pipeline. Gated clocks can be advantageously combined with the pipeline architecture. When input and control registers have to be implemented to obtain a gated clocked ALU, they are naturally used as pipelined registers. An interesting low power feature provided is the support of hierarchical memories. Frequently used code is stored in a fast and small ROM memory while infrequently parts of the code are stored in a large but slow memory. Since most of the time the small ROM memory is read, the power consumption is reduced.

5. Performances

MIPS (million of instructions executed per second) compare the throughput of microprocessor architectures with quite the same instruction set. The energy per instruction can be computed as the power * delay product,

i.e. $E/instr = C_{sw} * V_{dd}^2 * CPI$. One can choose the inverse function MIPS/Watt given by :

Error!= Error!which is independent of the frequency. However, microprocessors with high frequencies are more difficult to design and the best diagram to compare μP performances is the throughput/Watt as a function of the throughput. Table IV shows a comparison of 8-bit microprocessors (MIPS and MIPS/Watt).

6. Conclusion

The CoolRisc 88 microprocessor has been integrated in a 1μm process and first silicon is working. The maximum achieved frequency is 12 MHz providing 12 MIPS at 3.0 Volts. The core itself provides 5000 MIPS/Watt at 3.0 V (without memories). A family has been designed, i.e. 4, 8, 16 and 24-bit upgrade compatible cores with 1, 8 or 16 registers.

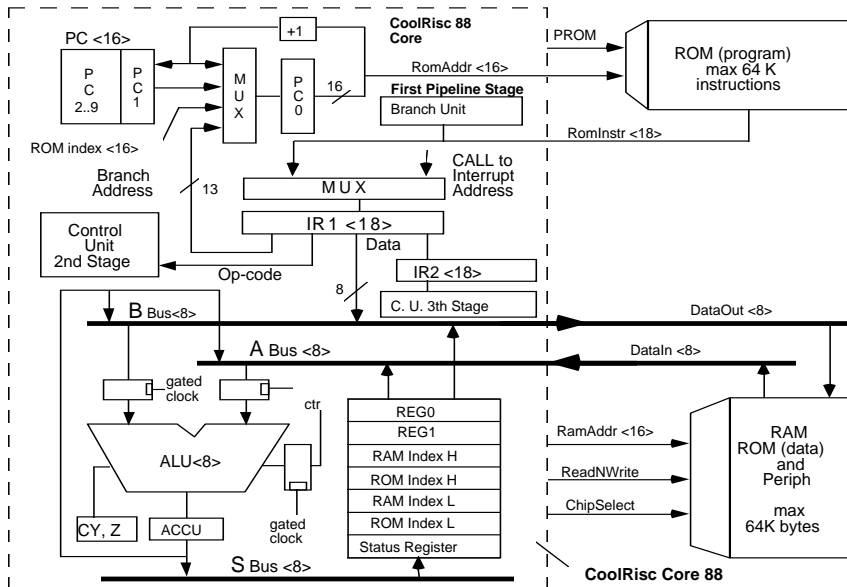


Figure 2. Architecture of the CoolRisc 88

Micro-processor	frequency for 1 MIPS	CPI	MIPS @ f	Vdd	Power @ 1 MIPS	MIPS/Watt
80C31	12	12.17	@ 20 MHz	3.0	30 mW	33
68HCxx	4	4	1 @ 4 MHz	3.0	8 mW	125
PIC16Cxx [2]	5	5	1.6 @ 8 MHz	3.0	7.5 mW	133
PIC17Cxx	5	5	3.2 @ 16 MHz	3.0	14 mW	71
Punch [6]	4	4	1 @ 4 MHz	3.0	1.2 mW	830
CoolRisc 81	1	1	14 @ 14 MHz	3.0	0.2 mW	5'000
CoolRisc 88	1	1	12 @ 12 MHz	3.0	0.3 mW	3'000
CoolRisc 816	1	1	10 @ 10 MHz	3.0	0.4 mW	2'500
CoolRisc 81	1	1	5 @ 5 MHz	1.5	0.05 mW	22'000
CoolRisc 88	1	1	4 @ 4 MHz	1.5	0.08 mW	13'000
CoolRisc 816	1	1	3 @ 3 MHz	1.5	0.09 mW	11'000

Table IV 8-bit Microcontroller Comparison

7. References

- [1] C. Piguet et al. "Low-Power Low-Voltage Digital CMOS Cell Design", Proc. PATMOS'94, October 17-19, 1994 Barcelona, Spain, pp. 132-139.
- [2] "Embedded Control Handbook 1994/95", Microchip Technology Inc, Sept. 1994.
- [3] D. A. Patterson, C. H. Séquin, "A VLSI RISC", IEEE Computer, Sept. 1982, pp. 8-21.
- [4] H. S. Stone, "High-Performance Computer Architecture", Addison-Wesley Publishing Company, 1987.
- [5] S. W. White et al. "How Does Processor MHz Relate to End-User Performance ?" Part I in IEEE Micro, August 1993, pp. 8-16, Part II in IEEE Micro, October 1993, pp. 79-88.
- [6] J-F. Perotto et al. "An 8-bit Multitask Micropower RISC Core", JSSC-29, No 8, August 1994, pp. 986-991.