The HF-RISC Processor: Performance Assessment

Sérgio F. Johann, Matheus T. Moreira, Ney L. V. Calazans and Fabiano P. Hessel
Pontifícia Universidade Católica do Rio Grande do Sul - Porto Alegre, Brazil
{sergio.filho, matheus.moreira, ney.calazans, fabiano.hessel}@pucrs.br

***Abstract*—This paper presents HF-RISC, a 32-bit RISC processor, along with its associated programming toolchain. The instruction set architecture of the processor is based on MIPS I and its hardware organization comprises three pipeline stages. The processor was synthesized in four different technology nodes for maximum frequency and simulated using CoreMark, an industry-standard performance evaluation benchmark. Using data obtained from synthesis and benchmarking we analyze the processor performance and compare it to similar commercial products. Obtained results indicate that HF-RISC is a good option for embedded design, as it presents performance figures similar to state-of-the-art ARM processors. Furthermore, its partially reconfigurable hardware organization allows the designer to explore performance and area trade offs.**

***Index Terms*—RISC; MIPS; FD-SOI; Design Space Exploration; Embedded systems;**

I. INTRODUCTION

Current embedded solutions require a relatively high processing power combined to low energy consumption. Yet, trends such as the Internet-of-Things (IoT) demand even greater levels of integration, as devices are connected to other systems and efficiency becomes a major concern. In such applications, several architecture characteristics have to be evaluated and processors must be designed for specific situations. Due to the the need for integration of IoT devices with other computer systems through network protocols such as IPv6 [1], 32-bit processor cores end up providing better performance and energy trade-offs, compared to 8 or 16-bit devices [2]. Embedded designs often have to trade raw performance improvements against power consumption. Thus, performance/power and similar ratios are more relevant than the traditional area, speed and power measurements isolated. Deep pipelines often incur in contentions among stages, which reduces the average number of executed instructions per clock (IPC) to less than 1. Keeping the number of pipeline stages low can take IPC closer to 1, simplifies the design [3] and reduces energy consumption [4].

In the processor proposed here this greatly simplifies the instruction set architecture (ISA) implementation (as no interlocks or forwarding units are needed to fix hazards). This approach is useful in lower clock frequency applications, where energy/MHz trade-offs need to be explored, rather than increasing clock frequency to a maximum. Industry currently employs the same principle, using 32-bit processors like the ARM Cortex-M family [5], with only 2 or 3 pipeline stages [2], in place of 8- and 16-bit microcontrollers. Such design choices aim to improve both performance and energy efficiency. We propose a 32-bit 3-stage pipeline processor based on the MIPS I instruction set architecture (ISA). The paper starts describing HF-RISC and its toolchain, exploring optimizations available at the hardware (HW) organization level. Next, it compares these optimizations, assessing the improvements offered in terms of performance, and compare these to state-of-the-art devices designed for embedded applications. The obtained results indicate a processing performance similar to that achievable with state-of-the-art solutions, but with a more efficient architecture in terms of silicon area. The results evidence that using HF-RISC can entail improvements in embedded design space exploration, as its different HW organization flavors allow trading off performance and area, which can in turn translate into power efficiency.

II. THE HF-RISC PROCESSOR

A. The HF-RISC Architecture

The architecture proposed here, HF-RISC, derives from the MIPS I ISA introduced in [6]. More specifically, the HF-RISC ISA is a small subset of the MIPS I ISA (with target on compatibility with existing tools and optimizing compilers), and the core has a specific organization, using few pipeline stages, along with a compact organization of the architecture components. The most relevant differences between the HF-RISC organization and a classic 5-stage MIPS are:

- Short, 3-stage pipeline, to simplify core design and to reduce chip area and energy consumption;
- No hazard or forward units, due to the short pipeline;
- Shared instruction and data memories, i.e. a von Neumann organization. Data accesses take 3 cycles;
- Fully synchronous, single clock edge design: registered memories are interfaced directly;
- 3-cycle branch delay when taken, with 2 branch delay slots;
- No unaligned loads/stores; no MMU; no exceptions;
- No co-processor, only memory-mapped peripherals (EPC, MASK, STATUS, VECTOR, and CAUSE registers);
- Configurable HW multiply unit, no HW division unit;
- A set of MCU-like peripherals: an optional UART, an interrupt controller, a running counter, two programmable counters, compare registers and a debug interface.

In terms of throughput, most HF-RISC instructions take just one clock cycle, but load and store instructions take three cycles each, due to the memory bus multiplexing and pipeline refill. Also, multiply instructions take several cycles, depending on the chosen hardware configuration. When a parallel multiplier is used, multiply instructions take 4 clock cycles and when a serial multiplier is used the latency is between 11 and 35 cycles. A side effect of the simple pipeline is the absence of explicit load delay slots of conventional MIPS organizations. Two branch delay slots arise due to pipeline design - the outcome of branches is discovered on the third pipeline stage, so the ALU can be reused. The compiler can schedule instructions in the first branch slot, reducing branch penalty to 2 cycles.

TABLE I
THE HF-RISC COMPLETE INSTRUCTION SET. (* ARE OPTIONAL)

Arith	Logic	Shift	Comp	Mem	Branch	Jump	Mult*
addiu	and	sll	slt	lui	beq	j	mthi
addu	andi	sra	sltu	lb	bne	jal	mfhi
subu	nor	srl	slti	lbu	bgez	jr	mtlo
	or	sllv	sltiu	lh	bgezal	jalr	mflo
	ori	srav		lhu	bgtz		mult
	xor	srlv		lw	blez		multu
	xori			sb	bltzal		
				sh	bltz		
				sw			

Table I presents the instruction set of HF-RISC. Only a minimal set of instructions were implemented (47 total, or just 41 if no HW multiply is used), such that the architecture performance is still acceptable for most applications and hardware complexity is reduced. Integer division and floating point operations execute through compiler-generated library calls.

B. The HF-RISC Organization

Figure 1 depict the stages of the HF-RISC pipeline and the tasks executed in each of these. In the *fetch* stage, memory is accessed and an instruction becomes available in one cycle. In this same cycle the PC is updated. In the *decode* stage an instruction is fed into the decoding and control logic, so values are registered for the next stage. Pipeline bubble insertion is performed in this stage for memory and branch operations. In the *execute* stage the register file is accessed and the ALU calculates the result of the operation. Address and data are put on the data bus (on store operations) or data are copied to the register file (on load operations). On logic/arithmetic operations, the ALU result is written to the register file. Branch outcomes are computed in this stage. Multiply operations write the result to HI and LO registers.

The register file is accessed only in the *execute* stage because the architecture is greatly simplified using this choice. Also, there is no need to implement forwarding logic on the pipeline, and less state information needs to be kept between stages. Another simplification concerns the behavior of load operations on data hazards. If the register file was accessed earlier, another pipeline stall would have to be used, along with additional logic surrounding the register file.

C. The HF-RISC Software Toolchain

To evaluate the performance of the architecture and build an adequate programming environment for measurements, we created a hardware abstraction layer (HAL), small C and runtime libraries, and used the GNU tools based on GCC 4.9.3 and Binutils 2.24. The compiler backend was modified to support all different processor configurations, including the absence of multiply and divide instructions and other microarchitecture features.

III. EXPERIMENTS AND DISCUSSION

A. Experimental Setup

In order to evaluate the different configurations of HF-RISC, we developed an automated flow that collects performance and area metrics. This paper does not consider power analysis due to space limitations. Figure 2 shows the devised flow, which is divided in two major task sets: (i) ISA Analysis; and (ii) ASIC Design and Analysis.

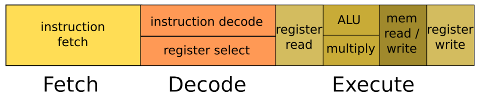


Fig. 1. The HF-RISC 3-stage pipeline and the stage tasks.

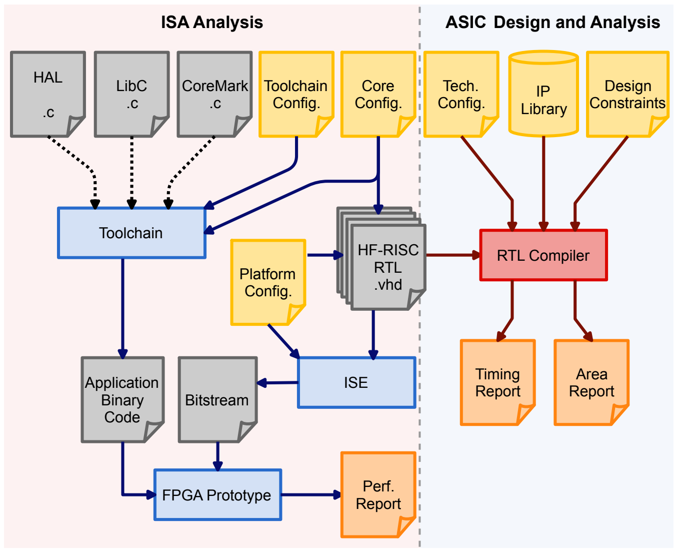


Fig. 2. Flow to extracting metrics for different HF-RISC configurations.

For the ISA analysis, we configure the RTL of the core using a set of “generic mapping” commands in VHDL. Using this RTL we relied on Xilinx ISE to generate a bitstream for prototyping the core in an FPGA. Note that to do so, one must also specify the target platform configuration to ISE, for setting the target FPGA, and to the RTL, for setting the right memory modules. The target board for all prototyping experiments in this paper was the Xilinx Spartan3 Starter Kit, which has a xc3s200 FPGA along with 1MB of SRAM. Once the core is prototyped in FPGA, a toolchain is used to compile the CoreMark benchmark targeting a specific HF-RISC configuration. The inputs to the toolchain are a set of runtime libraries, the source code of Coremark and a set of configurations for the toolchain. The latter defines the flags to be used and the target HF-RISC configuration. With these inputs, the toolchain generates a binary code that can be executed on the selected HF-RISC configuration. Note that this toolchain is an in-house design, which uses a modified version of the backend of GCC 4.9.3 compiler for MIPS and a set of scripts to automate the compilation, linking and binary generation processes. At this stage, the generated binary code can execute on the prototyped HF-RISC. After execution, the software outputs the performance of the processor in terms of Coremarks/MHz [7] through a serial interface. Note that we choose to prototype the design in order to obtain performance metrics because executing Coremark in the prototyped design is orders of magnitude faster than simulating its RTL description.

For the hardware design analysis, we synthesize different configurations of HF-RISC targeting a specific technology. To do so we rely on the Cadence RTL Compiler. The inputs to this part of the flow are: (i) the same RTL generated for the ISA analysis; (ii) a definition of a target technology; (iii) an IP library containing standard-cells to be used during synthesis; and (iv) a set of constraints that will guide the tool through the synthesis process, mainly defining clock constraints. Using these inputs, RTL Compiler synthesizes the design and outputs timing and area reports. The former reports if the tool was able to achieve timing closure considering the specified constraints for the target technology and IP library. The latter allows us to analyse the total cell area of the design.

B. Experimental Results

Using this experimental setup we evaluated three versions of HF-RISC: (i) one with a fast (parallel) multiplier (FM); (ii) one with a serial multiplier (SM); and (iii) one with no HW support for multiplication operations, where these operations must be done in software (SWM). For the ISA analysis, the three different versions received different application binary codes (HW for FM and SM and software multiply for SWM). We varied the compiler flags to exploit three different optimization levels: (i) *-Os*; (ii) *-O2*, and (iii) *-O3*. The basic compiler flags used are *-mips2 -mno-branch-likely -mpatfree -mfix-r4000 -mno-check-zero-division -msoft-float -fshort-double -nostdinc -fno-builtin -fomit-frame-pointer -G 8 -nohwdiv*. For the best optimization level (*O3extreme*) includes *-funroll-all-loops -fgcse-sm -finline-limit=500 -fno-schedule-insns* flags. On the SWM configuration, additional flags are *-mnohwmult -ffixed-lo -ffixed-hi*. Flags *-mpatfree*, *-mnohwmult* and *-nohwdiv* disable the generation of unaligned memory accesses, HW multiply and divide instructions, respectively. Such flags were included in the modified compiler (GCC 4.9.3) backend, while all other flags are present in the original compiler.

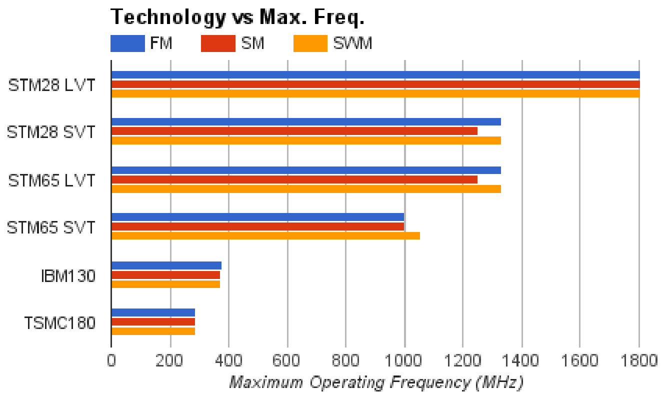
Table II presents the performance of different software and hardware configurations. In the case of agressive compiler optimizations (*-O3*), along with a parallel multiplier, the HF-RISC has a score of 2.01 CoreMark/MHz. This is comparable to similar class embedded processors (32-bit, in-order, short pipeline, HW multiply, von-Neumann organization) [7]. Furthermore, a score close to 3.0 CoreMark/MHz is expected if code and data memories are separated (Harvard machine). Note that, as Table II shows, the improvements provided by the agressive compiler optimizations come with a 3x cost in code size. In this way there is a clear trade off between performance and code size, which translates to memory requirements. We employ the best case Coremarks/MHz results as our baseline for comparisons, because using best case compiler configurations is standard in processors specifications.

The next set of experiments consists in evaluating the hardware design. To do so, we rely on six different technology flavors, which provide a broad perspective of the performance of the evaluated processor versions and enable a fair comparison with other processors. The employed technologies are STMicroelectronics FDSOI 28nm and Bulk 65nm, IBM Bulk 130nm and TSMC Bulk 180nm. For the first two technologies we employ flavors based on two types of transistors, low and standard threshold (LVT and SVT). For the other technologies we employ standard threshold transistors only. For STMicroelectronics technologies we use the IP libraries provided by the foundry and for IBM and TSMC we use the libraries provided by ARM. For each technology flavor we iterated the hardware analysis flow showed in Figure 2 until we reached maximum performance. We define maximum performance as the highest frequency achievable before the timing reports yield a negative slack, *i.e.* when the tool fails to meet clock requirements.

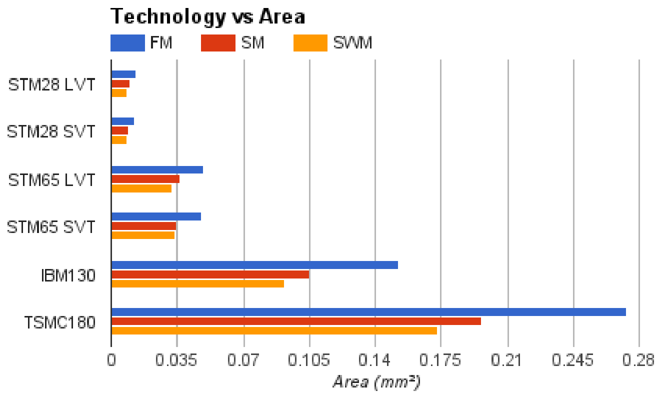
Figure 3(a) summarizes the obtained results for maximum frequency for each version of the processor and each technology flavor. Figure 3(b) shows the respective cell area for each of the designs. As the charts show, the designs achieved similar maximum frequencies regardless of having hardware support for multiplication or not. This is mainly because the pipeline of this design is unbalanced. However, the area overheads are clear and the FM design presents an overhead of 65% in the worst case. Regarding maximum frequency, in the best case, using LVT transistors in the FDSOI 28nm technology, the designs managed to operate at 1.8 GHz. In the worst case, using a bulk 180nm technology, the maximum frequency of the designs was 290 MHz.

TABLE II
COREMARKS PER MHZ AND CODE SIZE FOR EACH CORE CONFIGURATION WITH THREE DIFFERENT TOOLCHAIN CONFIGURATIONS.

Toolchain Config.		FM	SM	SWM
Os	*Coremarks/MHz*	1.42	1.22	0.68
	Code size (B)	11068	11068	11708
O2	*Coremarks/MHz*	1.68	1.43	0.96
	Code size (B)	11756	11756	12364
O3extreme	*Coremarks/MHz*	2.01	1.61	0.84
	Code size (B)	33500	33500	33420



(a)



(b)

Fig. 3. (a) Maximum frequency in MHz for each core configuration for six different target technologies and (b) respective cell area in mm².

TABLE III
MAXIMUM COREMARKS ACHIEVABLE AND NORMALIZED MAXIMUM COREMARKS PER MM² FOR EACH CORE CONFIGURATION FOR THE SIX TARGET TECHNOLOGY FLAVORS.

	Max. Coremarks			**Normalized Max. Coremarks per mm²**		
Technology	**FM**	**SM**	**SWM**	**FM**	**SM**	**SWM**
STM28 LVT	3636	2909	1745	0.94	1.00	0.67
STM28 SVT	2667	2000	1280	1.00	0.97	0.72
STM65 LVT	2667	2000	1280	0.99	1.00	0.72
STM65 SVT	2000	1600	1011	0.90	1.00	0.65
IBM130	755	593	356	0.87	1.00	0.68
TSMC180	580	457	274	0.91	1.00	0.68

Table III provides another perspective on these benchmark results. The Table shows the values in absolute maximum Coremarks and normalized maximum Coremarks per mm². The former is defined as Coremarks/MHz multiplied by the maximum frequency in MHz and the latter as maximum Coremarks divided by cell area and normalized to the best case across different processor configurations. As expected, the results show the improvements in maximum Coremarks allowed by technology scaling, a consequence of maximum frequency scaling. More interestingly, though, are the results for normalized maximum Coremarks per mm². As the Table shows, the FM and the SM designs presented similar scores, specially in more recent technology nodes. Thus, albeit the FM is more area hungry, the overhead pays off in terms of maximum Coremarks. In fact, this design presented at least 25% better maximum Coremarks when compared to the SM and at least 100% better than the SWM.

IV. DISCUSSION AND CONCLUSIONS

Table IV presents a survey of state-of-the-art devices used on several embedded applications, together with some HF-RISC results. All information in the Table was extracted from the Coremark website [7]. Note the selected devices present similar characteristics regarding architecture features (specifically: all cores are 32-bit, in order, scalar pipelines) and thus can be considered devices from a same class. Exceptions are the devices based on the ARM11 (superscalar ARM1176JZ-S, Broadcom BCM2835) architecture, included because they offer a similar performance in terms of Coremark/MHz.

As the Table shows, the FM version of HF-RISC is superior to most of the available devices in terms of Coremarks/MHz and is comparable to ARM processors like the ARM1176JZ-S and the Cortex-M0 employed in the STM32F051C8 device. The SM version, on the other hand provides more modest performance figures, being comparable to Xilinx MicroBlaze processors. It is harder to present a fair assessment of the absolute performance of these devices and put the HF-RISC in context. This is because information regarding the fabrication technology for the evaluated devices is scarce. Having that said, ARM provides information of maximum frequency for some of its processors and the respective fabrication technologies available. For the ARM1176JZ-S, for example, they report a maximum frequency of 772MHz in a bulk 65nm technology using standard threshold devices, which yields a score of 1605 maximum Coremarks. Note that, as Table III shows, the HF-RISC FM version in a similar technology yields 2,000 maximum Coremarks in its maximum operating frequency.

Another interesting comparison is with the STM32F051C8,

TABLE IV
COREMARKS/MHZ FOR STATE-OF-THE-ART DEVICES USED ON EMBEDDED APPLICATIONS.

Processor	**Compiler**	**Coremarks/MHz**
Microchip PIC24FJ64GA004	gcc-4.0.3	0.93
Analog Devices BF536	gcc-4.3.3	1.12
Xilinx MicroBlaze, 3-stage	gcc-4.1.1	1.48
HF-RISC v3.0 (SM)	gcc-4.9.3	1.61
Xilinx MicroBlaze, 5-stage	gcc-4.1.1	1.66
Broadcom BCM63281	gcc-4.2.3	1.68
NXP LPC1768	armcc 4.0	1.75
NXP LPC1768	Keil ARMCC	1.76
STMicro STM32F103RB	gcc-4.4.1	1.80
Marvell Kirkwood 88F6281	gcc-4.4.5	1.85
Broadcom BCM2835	gcc-4.6.3	1.86
Altera NIOS II/f	gcc-4.9.2	1.87
Marvell 88AP510 Armada 510	gcc-4.4.3	1.91
TI Stellaris LM3S9B96	Keil ARMCC	1.92
HF-RISC v3.0 (FM)	gcc-4.9.3	2.01
ARM ARM1176JZ-S	gcc-4.3.3	2.08
STM32F051C8	IAR 6.60	2.20

the top device in terms of Coremarks/MHz. The Cortex-M0 employed in this device has a maximum frequency of 50 MHz in a bulk 180 nm technology, which yields a maximum Coremarks of 110. This is substantially smaller than the results for the FM version of HF-RISC and the ARM1176JZ-S. The reason is that the STM device is designed for low area and low power. In fact, when we compare the processor area for the same technology, it is only 0.109 mm². Comparing to the 0.273 mm² of the HF-RISC FM version in a similar technology, this is a reduction of 2.5x. Obviously, the price paid is the reduction of 5.3x in maximum Coremarks. If we compare the Cortex-M0 to the SM version of HF-RISC, though, our design presents an area overhead of 89% only with an improvement in performance of 4.2x in terms of maximum Coremarks. In this way, there is a clear opportunity to explore the design space for embedded computing using the different versions of HF-RISC for different performance requirements.

As future work we will include a power analysis in our experimental setup to provide a broader comparison against state-of-the-art devices. This will allow exploring opportunities for contemporary applications like the IoT. As said before, this analysis was not included here due to the space limitations. Furthermore, we prototyped the SWM version of HF-RISC in a bulk 180nm technology and validated it on silicon. The next step is to prototype the other versions of the core in the same technology to explore their trade offs with precise on-chip measurements.

REFERENCES

[1] J. Gubbi *et al.*, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," *Fut. Gen. Comp. Syst.*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.
[2] J. Yiu, *The Definitive Guide to the ARM Cortex-M0*. Newnes, 2011.
[3] M. Labrecque *et al.*, "Custom Code Generation for Soft Processors," *SIGARCH Comput. Archit. News*, vol. 35, no. 3, pp. 9–19, Jun. 2007.
[4] O. Azizi *et al.*, "Energy-performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Analysis," in *ISCA*, 2010, pp. 26–36.
[5] ARM, "Cortex-M0+ Technical Reference Manual," Tech. Rep., 2012. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0484b/DDI0484B_cortex_m0p_r0p0_trm.pdf
[6] J. Hennessy *et al.*, "MIPS: A VLSI Processor Architecture," in *CMU Conference on VLSI Systems and Computations*, 1981, pp. 337–346.
[7] "Coremark - A benchmark provided by Embedded Microprocessor Benchmark Consortium (EEMBC)," http://www.eembc.org/coremark, accessed: 2015-11-8.