Floating Point Hardware for Embedded Processors in FPGAs:

Design Space Exploration for Performance and Area

Taciano A. Rodolfo, Ney L. V. Calazans, Fernando G. Moraes
Faculty of Informatics (FACIN)
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre, Brazil
e-mail: {taciano.rodolfo, ney.calazans, fernando.moraes}@pucrs.br

Abstract – **Although the use of floating point hardware in FPGAs has long been considered unfeasible or relegated to use only in expensive devices and platforms, this is no longer the case. This paper describes fully-fledged implementations of single-precision floating point units for a MIPS processor architecture implementation. These coprocessors take as little room as 6% of a medium-sized FPGA, while the processor CPU may take only 2% of the same device. The space exploration process described here values the area and performance metrics and considers variations on the choice of synthesis tool, floating point unit generation method and architectural issues like clocking schemes. The conducted experiments show reductions of up to 22 times in clock cycles count for typical floating point application modules, compared to the use of software-emulated floating point processing.**

Keywords – floating point hardware; FPGA; embedded processor; GALS design; prototyping.

I. INTRODUCTION

For many years, hardware floating point (FP) units (FPUs) were resources affordable only in expensive wall powered systems, because of their high complexity when compared to integer units and the low density of integrated circuits. Gradually, hardware FPUs reached the status of a commodity in systems such as desktop computers, high-end microprocessors and powerful game consoles. The current state of the art in integrated circuit density and performance allows designers to envision a near future when FP hardware will be commonplace in practically any system where they may bring some performance advantage. This includes several classes of embedded systems, and even mobile devices, if the FP hardware power dissipation problem is adequately controlled. FPGAs were once devices where designers avoided FP hardware due to high area overhead and low performance. This is no longer true. This paper explores the design space of FPGA embedded processors with FPUs.

To put the work in perspective, Figure 1 displays the design options spectrum to solve problems that can benefit from the use of floating point representations.

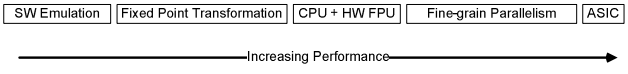

Figure 1. The spectrum of design options to deal with FP representations.

Each option has its own set of benefits and disadvantages, the choice of the method depending on several issues. Software emulation is the most straightforward option, since it relies simply on a compiler that can transform FP computations into an equivalent set of integer manipulations. This appears as the most used method in embedded processors. Its main problem is low performance. Transforming an FP specification to a fixed-point implementation may solve the software emulation performance problem, at the cost of increased design time and reduced portability. The next option is to use a processor coupled to some hardware FPU. This may have a dramatic impact on the performance, but may imply a significant overhead in silicon real estate and also in power dissipation. This option is the focus this paper. Specific problems such as radars and molecular dynamic simulations can benefit a lot from the fine-grain parallelism enabled by FPGA architectures, but this may lead to large hardware design time and reduce the flexibility and portability of the solution. Finally, the virtually infinite degrees of freedom of Application Specific Integrated Circuit (ASIC) design can take the most of area, performance and power optimizations. However, few embedded applications are able to afford the non recurring design and mask fabrication overheads implied by ASIC design flows.

The rest of this paper is organized in six Sections. Section II approaches related work. Section III describes the basic FP coprocessor design for a MIPS processor. Section IV presents the FPUs initially implemented for conducting the design space exploration process and Section V describes the CPU-FP coprocessor integration. The design space exploration is the subject of Section VI. The paper ends with some conclusions, ongoing and future work, in Section VII.

II. RELATED WORK

The amount of works proposing FPGA acceleration of specific problems demanding FP is quite large. However, approaches other than exploring FPGA fine-grain parallelism exist as well. The proposals of Chong & Parameswaran [1] and Beauchamp et al. [2] produce each a case to justify the inclusion of embedded FPUs in FPGAs. However, such pleads do not seem to have had effect so far on mainstream FPGA vendors.

When FPGA resources were more modest, some authors proposed configurable FP formats and FP module libraries. An example of the first is Dido et al. [3] that designed a specific, parameterizable, non-IEEE format for DSP design on FPGAs. Belanovic e Leeser [4] on another effort proposed a parameterizable module library, able to support the construction of FP operators adapted to the application in view.

Other works suggest that very high performance is achievable in FPGA FP computations, by carefully designing basic FP operators, as shown by Jie et al. [5]. Such works support the adoption of core based approaches used here.

Finally, there are works proposing the integration of embedded processors and FP hardware in FPGAs. This forms a complete solution, as investigated here. Papakonstantinou et al. [6] use the Altera Nios configurable soft core and FP custom instructions for accelerating MP3 processing. Their architecture is specific for the problem. Kadlec et al. [7] improve FP performance of a MicroBlaze soft processor with up to eight FP coprocessors, each controlled by a reconfigurable Picoblaze soft processor. The integration is strongly based on the EDK environment for connecting Microblaze and the coprocessor. Configuring of the coprocessor controllers may require assembly coding. In another work, Steiner et al. from Xilinx [8] showed how to accelerate FP with a FPGA-specific firm FPU coupled to an embedded hard core PowerPC processor. Again, EDK is used, which may impair flexibility. The approach described here uses an open source embedded processor enabling low level optimization of the processor/coprocessor interface. Also, the paper explores a generic processor plus FPU combination, favoring fast design of blocks, dozens of which can be replicated in FPGAs.

III. A MIPS-I COPROCESSOR 1 DESIGN

Although proposed almost 30 years ago, the MIPS-I architecture is still largely employed today, in several niche embedded applications. Its straightforward RISC architecture is well known and some encompassing open-source implementations are available. Due to these characteristics, this work uses the MIPS-I architecture and proposes an FP coprocessor for it that serves as basis to the experimentation.

A. The MIPS-I architecture and coprocessors

The MIPS-I basic block diagram appears in Figure 2. This is a 32-bit RISC processor with a load/store architecture, where all arithmetic and logic instructions operate on register values only. The MIPS CPU contains 32 32-bit general purpose registers, named $0 (which is in fact the constant zero, not a proper register) through $31. There are Harvard and von Neumann organizations proposed for this architecture, and pipelined implementations of the hardware have often been used.

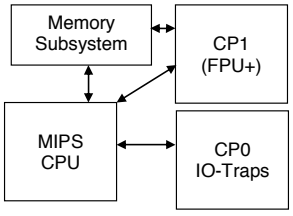

Figure 2. A typical structure for a MIPS-I architecture processor.

A MIPS-I architecture consists of a central processing unit (CPU) and functional units, or coprocessors, that perform auxiliary tasks or operate on other data types. The architecture supports operation with one to four co-processors two of which are displayed in the block diagram of Figure 2. The CP0 co-processor is used to control the system, performing tasks such as memory management (when needed), interrupt and exception handling, producing execution diagnostics, etc. The CP1 coprocessor is the FPU. The other two coprocessors are reserved for implementation of additional units, depending on the specific utilization devised for the architecture. CP0 is usually implemented within the processor, on the same chip. Other coprocessors, including CP1, can be implemented separately or as part of the same chip.

The CP1 is defined by the MIPS-I architecture as an IEEE 754-compliant floating point coprocessor, with its own 32 32-bit wide registers, that operates in both single and double precision. This work limits developments to single precision implementations, but most results can easily extend to treat a full single/double precision FPU version.

Most complex systems on chip (SoCs) built today reuse complex modules extensively. Accordingly, all experiments described herein assume a core based design approach, where available processor and FP cores are reused or generated automatically with design automation tools. HDL coding takes place only where reuse is not possible or feasible. For the MIPS CPU architecture, this work selected the Plasma open-source implementation [9] that presents structure and memory interface as depicted in Figure 3. Plasma is a von Neumann organization supporting all MIPS-I instructions, except for unaligned load and store instructions, because these are patented.

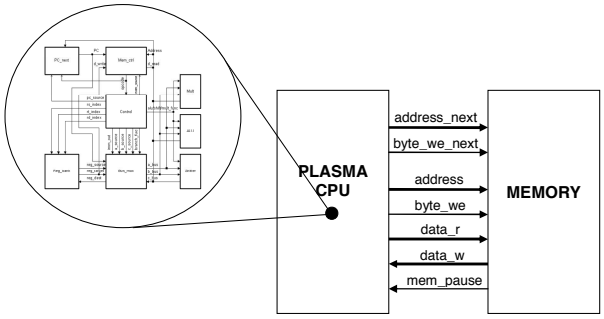

Figure 3. Plasma block diagram and its memory interface, taken from [9].

The latest available version of Plasma comes with a GNU **gcc** cross compiler and an operating system kernel supporting real time applications, besides software facilities like a TCP/IP protocol stack and a boot loader. The free VHDL embedded core also includes peripherals like a bidirectional serial port, DDR SDRAM controller, MAC Ethernet interface and a Flash controller. However, Plasma does not contain any memory management unit or FPU. The control coprocessor CP0 deals only with interrupt enabling/disabling and some exception treatment.

B. Structure of the Proposed MIPS Coprocessor 1

The structure of a fully-fledged MIPS coprocessor 1 (CP1) comprises three main elements: the FPU itself, a register bank similar to that of the CPU, and a controller. A datapath for the proposed CP1 appears in Figure 4. The FPU (fpu) and the register bank (bc_regs) are combined with an instruction register (ir) and some simple modules that enable

operations not supported by a typical FPU. These are needed to cover the whole set of MIPS-I FP instructions, including modules to execute absolute value (abs), negation (neg), conversions (cvt) and comparison (cmp) instructions.

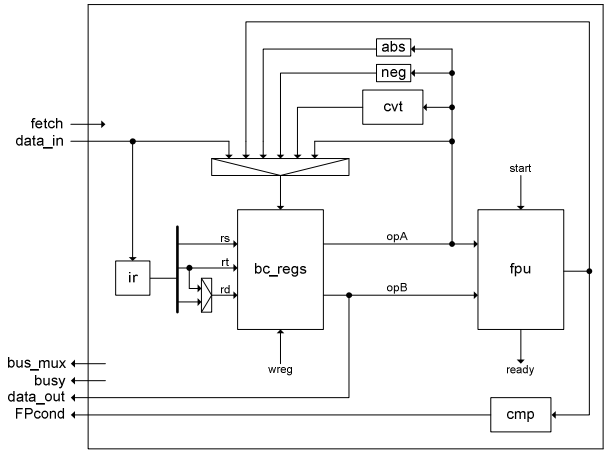

Figure 4. Block diagram of the proposed CP1 datapath.

The hidden controller, HFP_control, interacts with the MIPS CPU through a streamlined interface, the most relevant signals of which are depicted in Figure 4. Besides the basic 32-bit data transfer signals (data_in and data_out), the controller receives one CPU signal (fetch) and generates two signals used by external entities: FPcond signals the result of an FP comparison instruction by the CP1, enabling the CPU controller to execute control flow instructions based on FP conditions, while signal bus_mux routes data from FP data movement instructions outside the CP1 to the CPU or memory subsystems.

To the knowledge of the authors no MIPS-I CP1 is available in open-source format and little information other than manuals of commercial MIPS designs are found in the literature. The CP1 designs proposed are thus an original work unavailable on the Opencores Plasma project. The fpu block of the proposed CP1 is one of the sources of the design exploration process performed here. It is obtained from either reuse of open source code or from the use of automatic generators, as described in Section IV.

Another point to clarify is that Plasma had to be adapted to accommodate the new module. The CP1 operates in parallel with the CPU. It captures instruction words and decodes them to find out when an instruction has to be executed by it and not by the CPU. The Plasma correctly ignores FP instructions, but it has to suspend operation while an FP instruction is under execution. This takes place by OR-ing together signals mem_pause from the memory with the busy signal form the CP1 (See Figures 3 and 4). Also, since Plasma employs a von Neumann organization, instructions and data share the CPU input data bus. Thus, it was necessary to edit the Plasma interface, by adding a new signal (fetch) that indicates when an instruction fetch occurs. This enables the CP1 to decode instructions in parallel with the CPU.

IV. FPUs for Design Space Exploration

This paper adopts two methods to further the core based design approach of the design space exploration process. The first is the reuse of a generic FPU. The choice in this case was for the Opencores module FPU100 [10], a VHDL, IEEE 754-compliant, single precision soft core. The second method relies upon the use of a design automation tool capable of generating firm cores for FPGAs. The choice here was for the Xilinx Core Generator tool (Coregen), which enables the parameterized, automatic generation of hardware FP operators for Xilinx FPGAs. The tool was used to produce two versions of FPUs functionally and structurally equivalent to the FPU100, to enable fair comparisons.

A. The Open-Source FPU100 FPU

The FPU100 datapath block diagram appears in Figure 5. It contains four operators, one for performing addition and subtraction, one for square root and two others for the remaining arithmetic operations (multiplication and division). The core supports the four IEEE 754 rounding modes through the 2-bit control input rmode_i, and is capable to signal all exceptions predicted in the standard, through the 8-bit exception_o output. As usual, exception treatment is left to software. This work ignores exceptions. The communication protocol with this FPU uses two signals, start_i and ready_o, that signal the start and end of the FP computation for the operation specified by the fpu_op_i signal. As MIPS-I does not contain a square root instruction, the FPU100 was edited to remove this block for the purpose of this work.

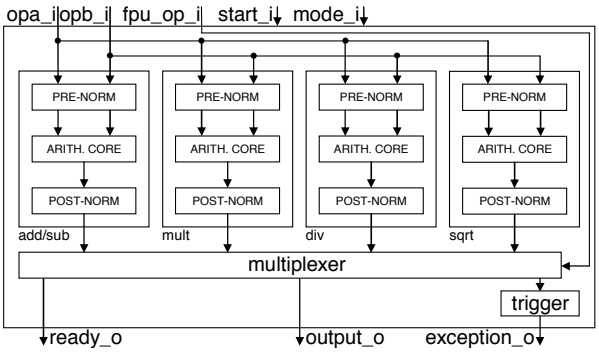

Figure 5. Block diagram for the FPU100 core datapath.

B. The Xilinx Coregen FPUs

Coregen is part of the ISE Design Suite for Xilinx FPGAs. It allows the user to choose among a large number of parameterizable hardware generators to support core based design for Xilinx FPGAs. Although some modules are in fact only available after the payment of licensing fees, many are included freely with the ISE license, available for academic institutions through the Xilinx University Program. This work employed the Floating Point hardware generator version 3.0, available in the ISE Design Suite version 10.1, with Service Pack 3.

Coregen allowed the production of two FPU versions with interface and functionality compatible with the FPU100, as illustrated in Figure 6. Besides the arithmetic

operators it was possible to automatically produce modules to perform comparisons and conversions eliminating the need for the external cvt and cmp modules of Figure 4.

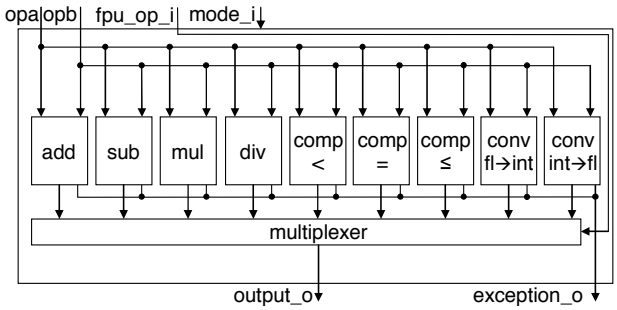

Figure 6. Block diagram for the Coregen FPU with minimum latency.

Coregen floating point hardware generator allows specifying each module's latency. The two FPU Coregen versions differ in this aspect. One uses minimum latency, while the other employs maximum latency for all modules. Since the minimum latency Coregen FPU takes exactly the same number of cycles for all operations, signals start_i and ready_o are not necessary and are accordingly eliminated. This is not the case for the maximum latency version which do present these signals on its external interface. Intermediate latencies between the minimum and maximum values are possible and their investigation is left as future work.

V. CP1S-PLASMA INTEGRATION

The development of the three functionally equivalent and structurally similar FPUs described in Section IV enabled producing single-precision CP1s containing each FPU. Next, took place the integration of each of these CP1s with the Plasma. Figure 7 depicts the general interconnection structure for these fully synchronous versions of the Plasma with Hardware Floating Point Unit (Plasma-HFP).

Besides the already cited modifications on the Plasma CPU (addition of the fetch output, see Section III), other changes in Plasma were needed to accommodate the complete functionality of the CP1s. This occurs because coprocessor-related instructions of MIPS-I are absent in the Plasma CPU. The instructions in question are the data movement instructions between CP1 and the memory (swc1 and lwc1), between the CPU and the CP1 (mtc1 and mfc1), and the conditional branch instructions based on CP1 results (bc1t and bc1f). The former two are more easily implemented in the CPU, because they need the base address register value from the CPU register bank. The latter two test the FPcond input coming from the CP1 to control the instruction flow. Decoding of other FP instructions is overlooked by the CPU, since these are taken over by the CP1 controller. In the proposed implementation, the memory does not need to insert wait states. Thus, the mem_pause signal is in fact under control of the CP1 only. Also, it was necessary to interpose the multiplexers appearing in Figure 7, to create a path for data exchanges between CPU, memory subsystem (RAM) and CP1s. In this version each multiplexer is controlled by the CP1 through the 4-bit signal bus_mux, one for each multiplexer. Also, the 32-bit memory interface accepts byte enable writes controlled by the mem_byte_we 4-bit signal. For the CP1s all writing deal with entire words only, explaining the enable all ("1111") signal.

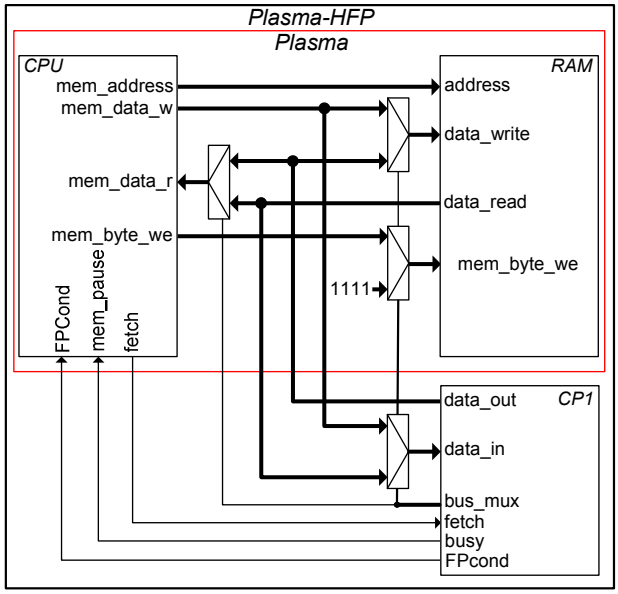

Figure 7. Overall structure of the fully synchronous Plasma-HFP organizations. Clock signals and less relevant signals are omitted.

The Plasma-HFP organizations described in this Section are the starting point for the design space exploration.

VI. DESIGN SPACE EXPLORATION

The main intent of this work is to investigate core based design processes for using FPU hardware in FPGA embedded processors. The primary method to conduct the investigation is to examine design trade-offs in performance, identified by the maximum operating frequency in synchronous systems, and FPGA area occupation, in LUTs. This is approached in Section VI.A. The results of this first step served to further the design space exploration, enabling to create a better implementation of the Plasma-HFP, as described in Section VI.B. Finally, Section VI.C brings some data on the acceleration achievable with the proposed organizations for FP intensive application modules.

A. *Basic Space Exploration*

To analyze the trade-offs, all three Plasma-HFP versions passed by synthesis and prototyping. All organizations were successfully prototyped on a Dinigroup DN8000K10PCI platform with a Xilinx Virtex4 XCV4FX100-10 FPGA. An example C program to exercise all single precision MIPS-I FP instructions was compiled, loaded on organization memory and run. The Xilinx Chipscope on-chip verification tool certified the process. Table I displays the results of this step.

The synthesis process used two distinct tools: XST, the ISE Design Suite 10.1 built-in tool and Synplify-Pro version 9.4 from Synplicity. Since exploring the best possible use of the tools was not an issue, both were explicitly parameterized

to use only default settings. For each organization, Table I displays four lines, the first of which containing data about the synthesis of the whole organization and the other three with data about the synthesis of selected modules: the whole CP1, the hidden controller and the FPU itself. Each of these three lines results from separate synthesis processes.

TABLE I. INITIAL DESIGN SPACE EXPLORATION DATA FOR A XILINX VIRTEX4 XCV4FX100-10 FPGA. THE 4-LINE GROUPS BRING GLOBAL AND PARTIAL DATA ABOUT EACH PLASMA-HFP VERSION. THE LAST 2 LINES RELATE TO THE CPU AND THE CP1 REGISTER BANK.

Module / Tool	XST		Synplify	
	Frequency (MHz)	Area (LUTs)	Frequency (MHz)	Area (LUTs)
Plasma-HFP100	28,52	13513 (16%)	34,10	8221 (9%)
- CP1	28,89	10285 (12%)	33,60	5399 (6%)
- Controller	330,24	1005 (1%)	179,00	1085 (1%)
- FPU	28,58	8328 (9%)	78,00	3283 (3%)
Plasma-HFPMin	5,84	7606 (9%)	5,60	7342 (8%)
- CP1	5,84	4553 (5%)	5,70	4596 (5%)
- Controller	759,71	137 (1%)	784,10	118 (1%)
- FPU	83,99	3283 (3%)	190,00	3489 (4%)
Plasma-HFPMax	61,50	8092 (9%)	35,20	7594 (8%)
- CP1	79,42	4815 (5%)	46,60	4810 (5%)
- Controller	478,48	116 (1%)	234,10	98 (1%)
- FPU	249,83	3532 (4%)	208,90	3582 (4%)
Plasma Only	66,07	3124 (3%)	38,50	2382 (2%)
Reg_bank_FPU	NA	1068 (1%)	NA	1062 (1%)

The first feature salient in the Table is that an embedded processor with FP hardware takes as little as 8% of the FPGA. Remember this family contains devices with as much as twice this amount of resources (3x more in Virtex5). Also, this module fits in some of the smallest family members.

Table I area information shows that Synplify consistently obtains better area results than XST. However, the difference is only significant for the FPU100. This is due to the fact that the FPU100 source code does not use Xilinx specific FPGA features like Block-RAMs or LUT-RAMs and Synplify has better RAM-inference algorithms than XST. For the Coregen versions this is probably not relevant, since this Xilinx tool is expected to use Xilinx FPGA features extensively. Thus, using XST or Synplify does not impact area significantly when using Coregen to produce FP modules. As for the operating frequency, the results point again to the use of Synplify as a better choice for the open-source FPU100. For Coregen, however, both tools give bad results for the min. latency version and the minor gains in area with regard to the maximum latency version do not justify the performance loss.

Finally, the partial data on operating frequencies of selected modules led to the conclusion that the CP1 design was not the best possible, because the main FP modules (the FPUs) had maximum operating frequency far larger than the corresponding CP1, except in one case. The analysis of this result and the measures taken to overcome this are the subject of the next Section.

B. *Enhancement of the Plasma-HFP System*

The first problem is to explain why the CP1 implementations obtained very low maximum operating frequencies compared to the respective FPUs. The next is to propose a solution for it. Finally, if the first problem can be solved the discrepancy between the maximum operating frequency of the CPU and its CP1 has to be overcome.

Referring to Figure 4, an analysis of the proposed CP1 organization revealed that the data interface between the CP1 register bank and the FPU was at the core of the problem, with a very long critical path. This can be explained in part by the large combinational multiplexers at each output of the register bank, associated to the combinational path after the inputs of the FPU. To solve this, the data interface has been registered and the controller behavior changed accordingly. This alone had a dramatic impact on the maximum operating frequencies of the CP1s, which become practically identical to the FPU operating frequency (near 250MHz). Also, the original register bank was written in generic VHDL and an analysis of the Plasma CPU showed that it contained functionally equivalent modules using hard features of both Xilinx and Altera FPGAs. Thus, the CP1 register bank was exchanged by the appropriate module, reducing the CP1 register bank area to one tenth of its original size.

To overcome the CPU-CP1 operating frequency discrepancy, the solution was the use of a globally asynchronous, locally synchronous (GALS) design approach for the embedded processor. Figure 8 shows the new organization.

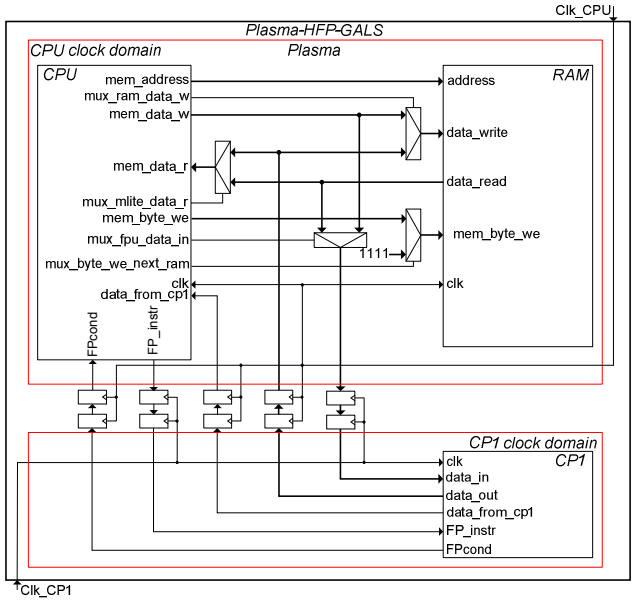

Figure 8. Proposed organization for GALS version of the Plasma-HFP.

In the Plasma-HFP-GALS, CPU and memory are located in one clock domain and the CP1 is at another clock domain. The exchange of data and control signals among the domains relies on the use of simple two-flop synchronizers [11]. Due to the separation of clock domains and to the intrinsic inefficiency added by the simple synchronizers, it was necessary to modify the CPU-CP1 interface. Two-flop synchronizers

imply a latency of at least two additional clock cycles on the receiving domain, which become at least four cycles if a handshake between domains is necessary. To avoid this as much as possible, all multiplexers remained in the CPU clock domain and their control migrated to the CPU, requiring Plasma code alterations again. With this new organization only two instructions suffered additional delay, move from coprocessor 1 and the store from coprocessor 1 instructions (MFC1 and SWC1). These instructions required the addition of the data_from_cp1 control line in the CP1-CPU interface. Both instructions now take 4 CPU cycles to execute, instead of the 2 cycles in the synchronous version.

C. Application Modules Behavior Comparison

The GALS version of the Plasma-HFP used as basis for development the Plasma-HFPMax synthesized with the XST tool, which provides the best area-performance trade-off. To evaluate the new area-performance trade-offs, four FP-intensive application modules (sine and cosine computations and FIR and IIR digital filters) have been adapted to run on the synchronous and GALS organizations. Table II compares the obtained results, using as reference the performance of a Plasma-only organization emulating FP instructions with integer instructions, a code produced by **gcc**.

TABLE II. PERFORMANCE OF 4 FP APPLICATION MODULES RUNNING ON 4 DIFFERENT ORGANIZATIONS – SW EMULATION, SYNCHRONOUS HFP, AND GALS HFP WITH 2 FP TO CPU FREQUENCY RELATIONS, 4 AND 8.

FP Organizations Accelerations	Application Module Clock Cycles for Execution			
	Sine	Cosine	FIR	IIR
# of FP Instructions	1124	1043	37	26
Sw Emulation	89898	81906	2662	3776
Synchronous HFPMax	11638	10729	884	396
- Sw Em. Accel.	7.7X	7.6X	3.0X	9.5X
GALS HFPMax 4X	9381	8671	805	224
- Sw Em. Accel.	9.5X	9.4X	3.3X	16.8X
- Sync HFP Accel.	1.2X	1.2X	1.1X	1.7X
GALS HFPMax 8X	8722	8065	775	172
- Sw Em. Accel.	10.3X	10.1X	3.4X	21.9X
- Sync HFP Accel.	1.3X	1.3X	1.1X	2.3X
- GALS HFP 4X Accel.	1.07X	1.07X	1.03X	1.3X

To highlight GALS systems behavior, the GALS HFPMax organization runs with two combinations of clock frequencies: 12.5 and 50MHz for CPU and CP1 respectively, and 25 and 200MHz. More relevant than the absolute frequency values is the relation between these, which directly affects overall performance. The Table accordingly brings results in clock cycle count. From the results, the potential benefits of using the organizations should be clear. Accelerations do depend a lot on the application and the consideration of what organization to use in a given embedded application must pass by the execution of the candidate code on each organization. However, improvements of more than 20 times over SW emulation can make a difference in many fields.

VII. CONCLUSIONS AND ONGOING WORK

This paper demonstrated that using FPGAs as a substrate to implement embedded processors with FP hardware is feasible and has a manageable cost, even for small less-than-state-of-the-art devices. It also demonstrates that the use of available soft and firm cores enables fast execution of design space exploration for embedded applications for FP manipulation. The experiments have a generic nature and should easily map to several other processor architectures and FPUs.

Several directions are available to extend the design space exploration described here. Concerning the FPU, interesting enhancements include consider double-precision hardware and exception treatment issues and their influence in the processor and software design. Since even in FPGAs power concerns are growingly important, the studies can be extended to account for power dissipation, to complete the spectrum of considered design issues. Another important aspect is the proposed use of GALS techniques. Two-flop synchronizers are simple and fast to use, but have strong impact on performance. The use of better synchronizers as proposed in [12] are currently under investigation.

ACKNOWLEDGMENT

This work receives partial support from the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq-Brazil) under grants 134400/2008-9 (PNM), 309255/2008-2 and 141247/2005-3.

REFERENCES

[1] Y. Chong, S. Parameswaran, "Flexible multi-mode embedded floating-point unit for field programmable gate arrays," in: 17th Int. Symp. on Field Programmable Gate Arrays (FPGA'09), pp. 171-180.

[2] M. Beauchamp, S. Hauck, K. Underwood, K. Hemmert, "Embedded floating-point units in FPGAs," in: 14th International Symposium on Field Programmable Gate Arrays (FPGA'06), pp. 12-20.

[3] J. Dido, N. Geraudie, L. Loiseau, O. Payeur, Y. Savaria, D. Poirier, "A flexible floating-point format for optimizing data-paths and operators in FPGA based DSPs," in: Proc. 10th Int. Symp. on Field Programmable Gate Arrays (FPGA'02), pp. 50-55.

[4] P. Belanovic, M. Leeser, "A library of parameterized floating-point modules and their use," in: 12th Int. Conf. on Field-Programmable Logic and Applications (FPL'02), Sep. 2002, pp. 657-666.

[5] S. Jie, Y. Ning, Z. Xiao-Yan, "An IEEE compliant floating-point adder with the deeply pipelining paradigm on FPGAs," in: 2008 Int. Conf. on Comp. Sc. and Soft. Eng. (CSSE'08), 2008, pp. 50-53.

[6] A. Papakonstantinou, Y. Kifle, G. Lucas, D. Chen. "MP3 decoding on FPGA: a case study for floating point acceleration," in: 4th Annual Reconfigurable Systems Summer Institute (RSSI'08), Jul. 2008, 4pp.

[7] J. Kadlec, R. Bartosinski, M. Danek, "Accelerating Microblaze floating point operations," in: 17th Int. Conf. on Field-Programmable Logic and Applications (FPL'07), Sep. 2007, pp. 621-624.

[8] G. Steiner, B. Jones, P. Alfke, "Floating Point: Have it Your Way with FPGA Embedded Processors," Xcell J. 1st Q. 2009, pp. 32-35.

[9] OpenCores, "Plasma-most MIPS I(TM) opcodes: overview," captured in: http://www.opencores.org/?do=project&who=plasma, Aug. 2009.

[10] OpenCores, "FPU100: overview," captured in: http://www. opencores.org /?do=project&who=fpu100, Aug. 2009.

[11] R. Ginosar. "Fourteen ways to fool your synchronizer," In: IEEE Int. Symp. on Asynchronous Circuits and Systems (ASYNC03), May 2003, pp. 89-96.

[12] R. Dobkin, R. Ginosar, "Two-phase synchronization with sub-cycle latency," Integration the VLSI Journal, 42(3), Jun. 2000, pp. 367-375.