An Architecture for Reconfigurable Multi-core Explorations

Olivier Serres, Vikram K. Narayana and Tarek El-Ghazawi
NSF Center for High-Performance Reconfigurable Computing (CHREC),
Department of Electrical and Computer Engineering,
The George Washington University,
Washington DC, USA
(serres, vikram, tarek)@gwu.edu

***Abstract*—Multi-core systems are now the norm, and reconfigurable systems have shown substantial benefits over general purpose ones. This paper presents a combination of the two: a fully featured reconfigurable multi-core processor based on the Leon3 processor. The platform has important features like cache coherency, a fully running modern OS (GNU/Linux) and each core has a tightly coupled reconfigurable coprocessor unit attached. This allows the SPARC instruction set to be extended for the running application. The multi-core reconfigurable processor architecture, including the coprocessor interface, the ICAP controller and the Linux kernel driver, is presented. The experimental results show the characteristics of the platform including: area costs, the memory contention, the reprogramming cost... Speedups up to $100\times$ are demonstrated on a cryptography test.**

I. Introduction

As per Moore's law, the number of transistors fitting on a single die keep increasing. To efficiently exploits those transistors, processors are using more and more cores. This present the user with more coarse grain parallelism that need to be exploited. FPGAs uses those transistor to allow very fine grain parallelism and to provide concurrency across the whole chip.

Reconfigurable processors try to bring the best of both world by combining a processor with some reconfigurable logic. It brings the processor ease of use for the non-critical part of the code and allows the exploitation of fine grain parallelism by custom hardware for performance critical portion of the code. Reconfigurable processor also have the great advantage of being able to be reprogrammed in the field to add new unforeseen features or corrections.

This paper presents a reconfigurable multi-core processor platform based on the Leon3 processor which is fully featured, including inter-core cache coherency, allowing it to run a complete SMP Linux kernel. Leon3 is based on SPARC v8 instruction set allowing it to run a great variety of applications, it is synthesizable to FPGAs but also available as ASICs. A fault tolerance version is also available for space applications.

The proposed reconfigurable coprocessor core is tightly coupled with the processor allowing it to be directly controlled by custom instructions issued by the SPARC core.

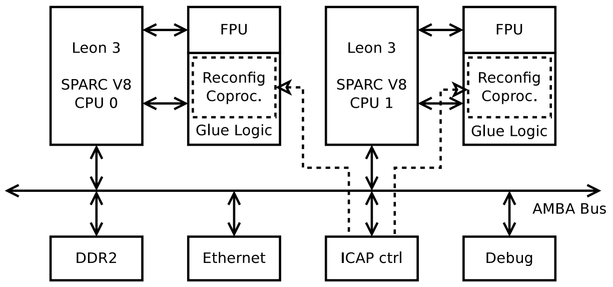

Figure 1. Leon3 SMP with reconfigurable coprocessors

The paper is organized as follows; first an overview of the related work is given in section II, next the system architecture is described in section III, section IV presents the experimental results, and section V concludes the paper.

II. Related Work

There has been a lot of reconfigurable processor research in the past, most notably: PRISC [1] is a RISC based microprocessor that includes reprogrammable function units, OneChip [2] is based on MIPS and integrates a tightly coupled reconfigurable core. Chimaera [3] integrates a superscalar processor with a reconfigurable function unit; the Chimaera C compiler has the ability to automatically map instructions to the reconfigurable unit. Recently, multi-core reconfigurable base systems are studied for example, Amalgam [4] which provide 4 cores and 4 reconfigurable cores without SMP, exchanging data by registers. In [5] a multi-core base system is simulated with Simics while the coprocessors run on a Virtex 5 FPGA. In [6], an heterogeneous multi-core system on chip is presented, however the system is based on Microblaze which is not suitable for running a full SMP OS like Linux.

III. System Architecture

A. Design Requirements

The design choices were guided by the following requirements. In order to be a realistic platform for multi-core coprocessor evaluation, processor cores supporting a full operating system, virtual memory, physical shared memory with cache coherency (to allow efficient data exchange and

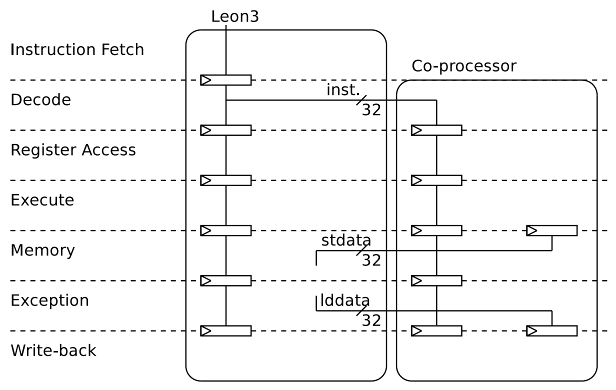

Figure 2. The Leon3 and the proposed coprocessor pipelines. Also represented, the timing for the load and store operations between the two units.

reuse between the processor cores) were needed. The processor core also add to use a small area on the chip to allow the placement of multiple cores on one FPGA. For those reasons, the OpenSPARC open-source processors (1 and 2) were not selected due to their important area usage per core. The OpenRISC 1200 processor does support SMP but the Linux port does not support this feature, yet. The Xilinx Microblaze and the Altera NIOS soft-core processors do not have support for cache coherency and modifications are prevented by their proprietary licenses. Leon3 was meeting all the design requirements (see Section III-B).

For the reconfigurable coprocessor, it was important for the coprocessor to be tightly integrated as it is shown to be one of the most versatile approach in its ability to provide speedup on a great variety of applications [7], as the data transfer cost between the processor and its coprocessor is greatly reduced compared to a loosely coupled approach. A tightly coupled approach also simplifies programming as one program controls both the processor core and its coprocessor.

B. Leon3

The Leon processors are developed by Aeroflex Gaisler, mainly for aero-spacial applications (they were originally developed by the European Space Agency).

The Leon3 32 bit processor synthesizable VHDL sources are released, under the GNU General Public License v2 (GPL), as part of the Gaisler IP library (GRLIB). The open-source version is fully featured to the exception of the radiation hardened Fault-Tolerance and the IEEE-754 Floating Point Unit (FPU). The open source nature of Leon3 makes it ideal for research [8].

Leon3 is compliant with the SPARC V8 specifications, allowing it to run a great variety of software and operating system, including GNU/Linux. Multi-processor is possible either in a symmetric or an asymmetric configuration (for example, see Fig. 1). Leon3 is based on a 7 stage pipeline, see Fig. 2.

Table I
THE SPARC INSTRUCTIONS RESERVED FOR THE COPROCESSOR

Instruction	op1	op3	Description
LDC	11	110000	Load Coprocessor
LDDC	11	110011	Load Double Coprocessor
LDCSR	11	110001	Load Coprocessor State Register
STC	11	110100	Store Coprocessor
STDC	11	110111	Store Double Coprocessor
STCSR	11	110101	Store Coprocessor State Register
STDCQ	11	110110	Store Double Coprocessor deferred-trap Queue
CPOP1	10	110110	Coprocessor Operate 1
CPOP2	10	110111	Coprocessor Operate 2
CB...	00	N/A	Branch on Coprocessor Condition codes

Leon3 also supports a wide variety of development features: it can be configured with a Debug Support Unit allowing, in conjunction with the debug monitor (GRMON), to fully debug the targeted system by reading/writing memory and registers, setting breakpoint and stepping through the code, attaching the GNU debugger (gdb), monitoring the transactions on the AMBA bus... Simulators are also developed by Aeroflex Gaisler like TSIM2 and GRSIM.

C. Coprocessor Interface

The Leon2 processor was widely used for reconfigurable processor research [9], [10] but unfortunately the simplified coprocessor interface which was provided with Leon2 was removed from the Leon3 source code release, making the development of coprocessor much more difficult (to the best knowledge of the authors, no publication describes a Leon3 processor connected to a coprocessor).

The Leon3 processor still exposes an interface for coprocessors. It is used to connect the GRLIB FPU unit. It uses the `rst`, `holdn`, `clk` signals plus two structures `fpc_in_type` and `fpc_out_type`. Those two structures basically expose all the signals from the main pipeline to the coprocessor. This makes the interface particularly complex to work with. In total there is 497 signals going in and out from the coprocessor. Having all those signals allow the coprocessor to reuse the registers used in the main pipeline and gives more freedom to the synthesis tools to optimize the coprocessor. However, this is not appropriate for a reconfigurable coprocessor implementation as all those signals would have to be preserved and routed to the reconfigurable region. This issue has been solved by designing a simplified interface for the coprocessor reducing the number of signals from 497 to 162. The VHDL records for the simplified interface are presented on Fig. 3.

The Leon3 core was also modified, as some signals from the coprocessor were not taken into account (`CPEN`, `holdn` and `dbg.data`). The implementation to store data from the coprocessor to the memory was also missing.

During the reconfiguration of the coprocessor, it is also important that the signals coming in from the coprocessor

```vhdl
type fpc_pipeline_control_type is record
  pc      : std_logic_vector(31 downto 0);
  inst    : std_logic_vector(31 downto 0);
  cnt     : std_logic_vector(1 downto 0);
  trap    : std_ulogic; -- instr. trapped
  annul   : std_ulogic; -- cancel next inst.
  pv      : std_ulogic; -- valid
end record;

type cp_out_type is record
  stdata : std_logic_vector(31 downto 0);
      -- data to memory
  exc     : std_logic;  -- except
  cc      : std_logic_vector(1 downto 0);
  ccv     : std_ulogic; -- branch code valid
  ldlock  : std_logic;  -- load lock
  holdn   : std_ulogic; -- hold
end record;

type cp_pipeline_control_type is record
  cnt     : std_logic_vector(1 downto 0);
  trap    : std_ulogic; -- trap
  annul   : std_ulogic; -- cancel next inst.
  pv      : std_ulogic; -- valid
end record;

type cp_in_type is record
  holdn   : std_ulogic; -- processor holdn
  flush   : std_ulogic; -- pipeline flush
  exack   : std_ulogic; -- exception acknowledge
  d       : fpc_pipeline_control_type;
  a       : cp_pipeline_control_type;
  e       : cp_pipeline_control_type;
  m       : cp_pipeline_control_type;
  x       : cp_pipeline_control_type;
  lddata : std_logic_vector(31 downto 0);
      -- data from memory
end record;
```

Figure 3. Reduced VHDL interface for the coprocessor

does not interfere with the processor pipeline. For this we used the fact that, due to the SPARC specification [11], the coprocessor has to be explicitly enabled before being used. This is done by setting the bit 13 of the Processor State Register (PSR) to 1. Trying to use the coprocessor instructions without enabling the coprocessor generates a `cp_disabled` processor trap. This is used to implement some glue logic that masks out the signals from the coprocessor when the coprocessor is disabled. The reconfiguration can now happens without interfering with the main processor pipeline.

The SPARC v8 architecture manual [11] also defines which instructions are available to control the coprocessor (See Table I). The two first sets of instructions (LD and ST) are used to transfer data to and from the coprocessor. Depending on the addressing mode, the load/store instructions have the following format:

op1 (2)	rd (5)	op3 (6)	rs1 (5)	i (1)	zero (8)	rs2

Effective address = r[rs1]+r[rs2].

op1 (2)	rd (5)	op3 (6)	rs1 (5)	i (1)	simm (13)

Effective address = current address + sign ext(simm).

In both those instructions, `rd` is the destination register and its signification is dependent on the coprocessor implementation. The rest of the load/store instructions is interpreted by the host processor and cannot be changed by reconfiguring the coprocessor. Timing information for the coprocessor load/store operations are shown in Fig. 2.

The following instruction format is suggested for the coprocessors:

op1 (2)	rd (5)	op3 (6)	rs1 (5)	opc (9)	rs2 (5)

op1 and op3 fields are also decoded by the processor so they cannot be changed. The other fields are implementation specific and can be re-purposed. With the two CPOP instructions, there is actually 25 bits available to implement the coprocessor instruction set.

The CB (Coprocessor Branch) instructions are interpreted by the processor and are not customizable, they rely on the condition codes (signal `cc`) returned by the coprocessor.

D. ICAP Controller

An Internal Configuration Access Port (ICAP) controller was written and connected to the Advanced Microcontroller Bus Architecture (AMBA). This module allows the reconfiguration of the coprocessors by allowing the processors to load new partial bitstreams. The controller behaves as an Advanced High-performance Bus (AHB) slave and it allows any processor core to reprogram any coprocessor.

E. Linux Kernel Driver

In order for the ICAP controller to be accessible by the user space programs, a kernel driver is needed. The kernel driver's first task is to register itself as a driver (via the `of_register_platform_driver` kernel function) for the ICAP controller. Once this is done, a new character device is created (`/dev/icap`) to allow the user space applications to communicate with the driver. When the Linux kernel probes the ICAP device, the driver maps the io device memory to the kernel address space (with `of_ioremap`). The following 4 IOCTL operations are provided to the applications to configure and reprogram the coprocessor:

- IOCTL_ICAP_COPROCESSOR_ENABLE: Modify the `psr` register of the user process to allow the use of the coprocessor instructions. This also configure the glue logic to connect the signals of the coprocessor to the main pipeline.
- IOCTL_ICAP_COPROCESSOR_DISABLE: Disable the coprocessor.
- IOCTL_ICAP_COPROCESSOR_STATUS: Get the current status of the coprocessor.
- IOCTL_ICAP_PROGRAM: Program the coprocessor using the bitstream at the virtual address provided by the user. The programming itself is protected by a mutex, so that two different cores cannot access the ICAP at the same time.

IV. Experimental Results

A. Testbed

The Xilinx ML-509 board was used for the experiments. The board is powered by a Virtex-5 XC5VLX110T FPGA. It

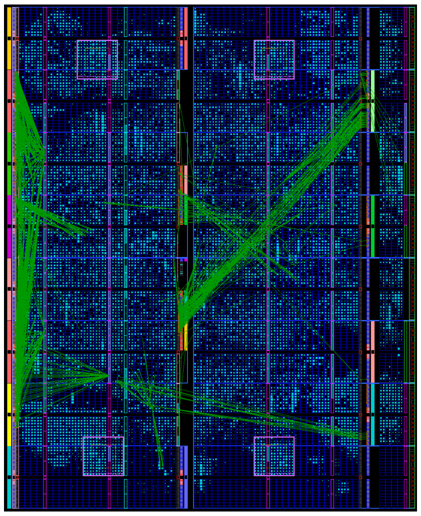

Figure 4. Placement of a 4 core SMP Leon3 design with reconfigurable coprocessors (the four squares represent the reconfigurable areas)

Table IV
MODIFIED STREAM BENCHMARK PERFORMANCE RESULT (MB/S)

# of cores	Copy	Scale	Add	Triad
1	46.8	8.4	40.0	11.1
2	64.0	16.6	60.0	21.7
3	66.2	24.9	62.6	31.6
4	68.6	33.7	64.0	41.7

provides 17, 280 slices, 148× 36Kb block rams (which can also be used as two independent 18Kb block rams) and 64 DSP48E slices (providing a 25 × 18 multiplier). The board also includes a JTAG interface, a Gigabit Ethernet port and 256 MB of DDR2 ram.

The following softwares were used: Synopsys Synplify Premier version E2011.03 to synthesize, Xilinx ISE version 13.1 to place and route the different designs, Leon3 from GRLIB version 1.1.0 B4100, LinuxBuild 1.0.0 to build the GNU/Linux environment with the Linux kernel version 2.6.36-4. All the designs target a clock speed of 70 Mhz.

B. Scalability/Performance

1) Area cost: The area cost for the Leon3 processor is evaluated as follows. A minimal configuration of Leon3 is chosen and extra features are added progressively. The increased area usage in terms of LUTS and block-rams is used to approximate the area cost of each features. Table II shows the total cost of fully functional configurations. Table III details the cost of individual features. Both tables can be used to easily evaluate the area cost of a given configuration.

2) Memory contention on the AMBA Bus: The AMBA Bus to access the memory is shared among all the cores; this limits the scalability of the multi-core approach. In order to evaluate this, a modified version of the OpenMP Stream benchmark [12] was used. On this modified version, the

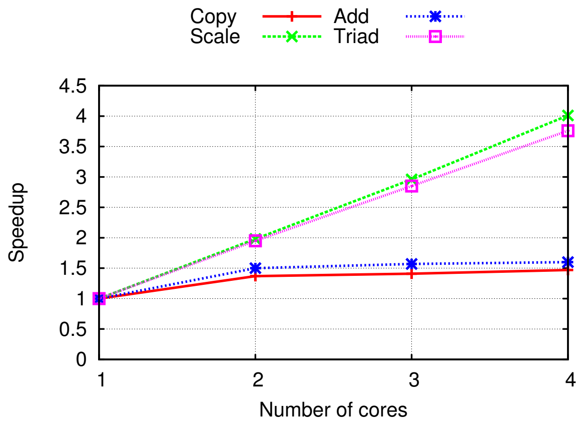

Figure 5. Scalability of the modified Stream test, showing the contention on the AMBA Bus

Table V
IMPLEMENTATION TIME FOR A 4 CORE SMP DESIGN

Implementation step	Full design	PR regions only
Synthesis	14m 55s	5s
Place and route	47m 24s	32m 28s
Total	62m 19s	32m 33s

floating point operations were replaced by integer operations (double data types were replaced by long long), due to the lack of floating point unit on our test platform. The results can be seen on Table IV. Fig. 5 shows the scalability on our platform. We can see that the scaling is very good for the more compute intensive benchmarks, but scaling stops at 2 cores for memory intensive operations. Due to this limited scaling, it is important that the caches are dimensioned appropriately and used efficiently. Coprocessors can also use block ram as buffer to reduce memory accesses to a minimum. For more demanding applications, Leon4 can be used. Leon4 has improved data paths with single cycle 64 bits load/store operations and a 128 bit wide AMBA 2.0 bus. Unfortunately, Leon4 is not yet available under an open source license.

C. Reconfiguration

The reconfiguration for a 79, 964 bytes bitstream takes 4.58 ms, this correspond to a reconfiguration speed of 16.6 MB/s. Most of this time is spent transferring the bitstream on the AHB bus and across the processor caches. The reconfiguration time could be reduced by either having a configuration cache in the ICAP controller or by making the ICAP controller a bus master so that it can fetch the bitstream directly from the DDR2 RAM.

D. Implementation time

Using a partial reconfiguration design flow has also the advantage of reducing the turn-around time to test new designs. Table V shows the implementation time for a

Table II
AREA COST FOR DIFFERENT CONFIGURATIONS OF LEON3

Configuration	**Slice resources**		**BRAM**		**DSP48Es**
	Registers	LUTs	18kB	36KB	
Leon3 minimal configuration	2,355 (3%)	3,615 (5%)	6	2 (5%)	
2 Leon3 cores (MMU, I1, D1, HW 2c., FPU full, debug, ICAP, DES)	16,886 (24%)	36,963 (53%)	50	20 (31%)	40
4 Leon3 cores (MMU, I1, D1, HW 2c., FPU full, debug, ICAP, DES)	-	76,431 (110%)	-	-	-
4 Leon3 cores (MMU, I1, D1, HW 2c., debug, ICAP, DES)	17,969 (25%)	32,258 (46%)	88	36 (56%)	16
6 Leon3 cores (MMU, I1, D1, HW 2c., debug, ICAP, DES)	25,493 (36%)	46,508 (67%)	126	52 (79%)	24

Table III
AREA COST PER ADDITIONAL FEATURE TO THE CONFIGURATION OF LEON3

Feature	**Slice resources**		**BRAM**		**DSP48Es**	Per core
	Registers	LUTs	18kB	36KB		
MMU (8 entries, combined, 4kB page size)	857	746				yes
I1 (8kB in 2 sets, 32 bytes/line, LRU)	77	250	4	2		yes
D1 (16kB in 4 sets, 32 bytes/line, LRU, AHB fast snooping)	700	2,028	12	4		yes
HW multiplier (5 cycles latency)	144	565			2	yes
HW multiplier (2 cycles latency)	93	362			4	yes
GRFPU lite	1,140	2,872				yes
GRFPU full	3339	9,423	22	8	16	yes
Coprocessor (DES core)	523	525	1			yes
Debug support (JTAG, Ethernet GRMon)	1,813	3,366	4	4		no
AHB ICAP controller	10	132	4			no

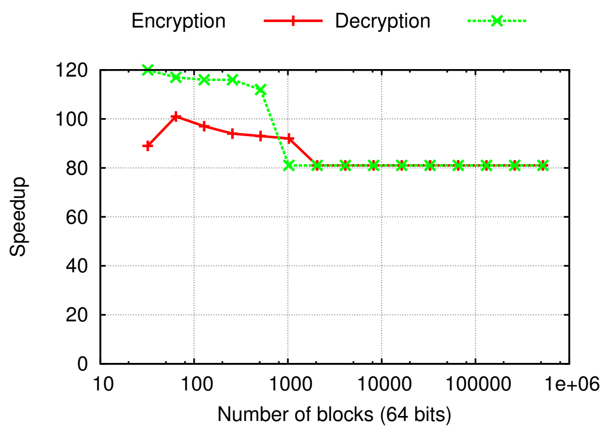

Figure 6. Sensibility to the dataset size for the DES coprocessor, using a warm cache (1 core)

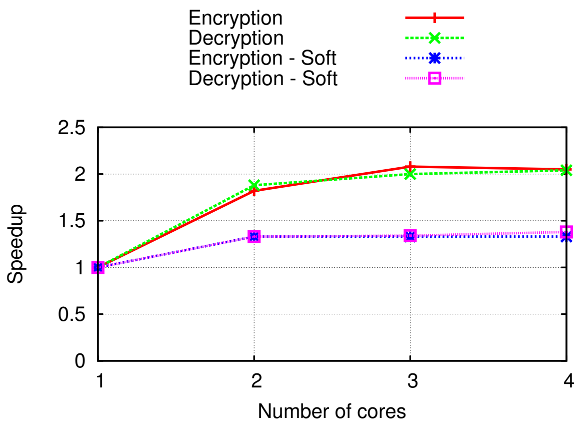

Figure 7. Scalability of the DES test

Table VI
DES PERFORMANCE RESULT (KIB/S)

	Hardware		**Software**	
Cores	**Encryption**	**Decryption**	**Encryption**	**Decryption**
1	6,826	6,826	87	87
2	12,423	12,800	108	116
3	14,184	13,626	108	117
4	14,045	13,948	108	120

4 core SMP design with reconfigurable coprocessors on a recent machine (equipped with two Intel Xeon X5667 CPUs running at 3.07GHz with 96 GB of RAM). The implementation time is reduced by 48%, allowing for much faster iterative coprocessor design.

E. Test Application: DES

To verify the functionality and the performance gains of the reconfigurable platform, a DES cryptography coprocessor was used. Encryption is a common application of partial reconfiguration as hardware implementations provides a good speedup for encryption algorithms. Also, the cipher can be changed based on the distant peer and it can also be updated [13]. For this work, two cores were used, one for encryption and one for decryption. Those cores are based on the Basic DES Block Cipher [14]. The design is not pipelined and perform a DES encryption in 17 clock cycles, however loading the next block of data (64 bits) can be performed at the same time as an encryption. The implementation is compared to the portable C DES functions [15] compiled with GCC and the -O3 level of optimization. The correctness of the core and of the software implementation was first verified using 250 commonly used test vectors for DES (including the test vectors from [16]). The results of the two implementations were also checked against each others. Speedup for 1 core and for different data sizes is shown on Fig. 6, an important improvment (100x) is observed against the C code. A drop in performance can be observed when the data do not fit in the cache anymore. Result for a multi-core design is presented in Table VI and in Fig. 7. In order to compare those results with the modified stream benchmark, the cipher speed needs to be multiplied by two (in order to account for both the loads and stores). We obtain a maximum data transfer speed of 28.4 MiB/s (versus 33.7 MiB/s for the scale benchmark). For both the software and hardware approach, scalability is good for 2 cores, fair for 3 cores and stops at 4 cores, showing the need for improvement to the interconnection between the cores for data intensive applications.

V. Conclusions

In this paper, a complete multi-core reconfigurable platform based on Leon3 was presented, including the coprocessor interface, the reconfiguration controller and the kernel driver. The platform runs a full Linux SMP operating system providing a rich environment and flexibility for the application programmers. Experimental results show the main platform characteristics like memory bandwidth and reconfiguration time. The coprocessor design time is also reduced thanks to the partial reconfiguration work flow, facilitating future research. The reconfigurable coprocessor also shows great speedup proving the value of this approach.

Acknowledgment

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. IIP-0706352. The authors also wish to acknowledge the Xilinx University Program (XUP) and the Sun Microsystems OpenSPARC University Program for their hardware and software donation which has been essential to complete this work.

References

[1] R. Razdan and M. Smith, "A high-performance microarchitecture with hardware-programmable functional units," in *Proceedings of the 27th Annual International Symposium on Microarchitecture, MICRO-27.*, Nov. 1994, pp. 172–180.

[2] R. Wittig and P. Chow, "OneChip: an FPGA processor with reconfigurable logic," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, Apr. 1996, pp. 126–135.

[3] Z. Ye, A. Moshovos, S. Hauck, and P. Banerjee, "CHIMAERA: a high-performance architecture with a tightly-coupled reconfigurable functional unit," in *Proceedings of the 27th International Symposium on Computer Architecture*, June 2000, pp. 225–235.

[4] D. Gottlieb, J. Cook, J. Walstrom, S. Ferrera, C.-W. Wang, and N. Carter, "Clustered programmable-reconfigurable processors," in *Proceeding of the IEEE International Conference on Field-Programmable Technology (FPT)*, Dec. 2002, pp. 134–141.

[5] L. Yan, B. Wu, Y. Wen, S. Zhang, and T. Chen, "A reconfigurable processor architecture combining multi-core and reconfigurable processing unit," in *Proceedings of the 10th IEEE International Conference on Computer and Information Technology (CIT)*, July 2010, pp. 2897–2902.

[6] T.-O. Kwok and Y.-K. Kwok, "On the design, control, and use of a reconfigurable heterogeneous multi-core system-on-a-chip," in *IEEE International Symposium on Parallel and Distributed Processing, IPDPS*, April 2008, pp. 1 –11.

[7] F. Barat and R. Lauwereins, "Reconfigurable instruction set processors: a survey," in *Proceedings of the 11th International Workshop on Rapid System Prototyping, RSP*, 2000, pp. 168–173.

[8] Gaisler Research, "GRLIB IP library user's manual, version 1.1.0 b4100," 2010.

[9] G. Cardarilli, L. Di Nunzio, R. Fazzolari, and M. Re, "Algorithm acceleration on LEON-2 processor using a reconfigurable bit manipulation unit," in *8th Workshop on Intelligent Solutions in Embedded Systems (WISES)*, July 2010, pp. 6–11.

[10] P. Guironnet De Massas and P. Amblard, "Experiments around SPARC Leon-2 for MPEG encoding," in *Proceedings of the International Conference on Mixed Design of Integrated Circuits and System (MIXDES)*, June 2006, pp. 285–289.

[11] R. Garner *et al.*, "The SPARC architecture manual: Version 8," 1992.

[12] J. D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, Dec. 1995.

[13] I. Gonzalez, S. Lopez-Buedo, and F. Gomez-Arribas, "Implementation of secure applications in self-reconfigurable systems," *Microprocessors and Microsystems*, vol. 32, no. 1, pp. 23–32, 2008.

[14] Steven R. McQueen, "Basic DES block cypher IP core, http://www.opencores.org/," 2003.

[15] Phil Karn and Jim Gillogly, "Software DES functions, http://www.ka9q.net," 1995.

[16] J. Gait, *Validating the correctness of hardware implementations of the NBS Data Encryption Standard.* US Dept. of Commerce, National Bureau of Standards, 1980, vol. 20.