A VLIW Processor With Reconfigurable Instruction Set for Embedded Applications

Andrea Lodi, Mario Toma, Fabio Campi, Andrea Cappelli, Roberto Canegallo, and Roberto Guerrieri

***Abstract*—This paper describes a new architecture for embedded reconfigurable computing, based on a very-long instruction word (VLIW) processor enhanced with an additional run-time configurable datapath. The reconfigurable unit is tightly coupled with the processor, featuring an application-specific instruction-set extension. Mapping computation intensive algorithmic portions on the reconfigurable unit allows a more efficient elaboration, thus leading to an improvement in both timing performance and power consumption. A test chip has been implemented in a standard 0.18-μm CMOS technology. The test of a signal processing algorithmic benchmark showed speedups ranging from 4.3$\times$ to 13.5$\times$ and energy consumption reduced up to 92%.**

***Index Terms*—Energy consumption, field-programmable gate array (FPGA), pipeline, reconfigurable processor.**

I. Introduction

PROCESSORS suitable for the last generation of embedded systems are facing opposite constraints. The first relevant factor is due to the sharp increase in non-recurring engineering and integration costs caused by the development of the most recent deep-submicron technologies. In order to amortize costs over high production volumes, and account for shorter time-to-market in embedded systems development, higher levels of flexibility are needed, thus ensuring reusability.

A second powerful factor is bound to the significant change in embedded processor workloads that has taken place over the last years. The typical application environment has gradually shifted toward an increasing computational complexity, requiring real-time elaborations of tasks such as image, audio, and video compression and recognition, telecommunications protocol stack management, and so on. This trend is described by the so-defined Shannon's Law [1]. Fig. 1 [3] shows algorithmic complexity for wireless applications compared with Moore's law [2]. It can be observed that the increase of computational requirements cannot be met by technological developments alone. In order to fill this gap, some kind of architectural breakthrough is needed. A new balance must be sought between silicon resources and the patterns for their computational utilization. The relation between required complexity and available energy in storage devices for portable applications appears even worse. Fig. 1 shows that the increase of capacity of a typical battery is negligible compared to

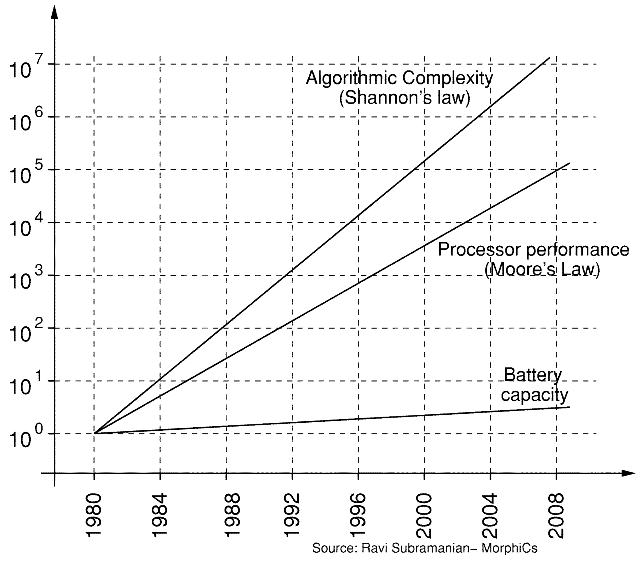

Fig. 1. Computational requirements versus Moore's law and battery storage.

both algorithmic complexity and technological development. Therefore, embedded processor design is facing a very strong push toward both higher flexibility and higher computational requirements, while in the case of portable applications, it is also subject to severe power consumption constraints.

An appealing option is to exploit the ever-improving programmable logic devices technology, combining standard processors with embedded configurable gate arrays in various ways [4]. Such an approach is broadly referred to as *reconfigurable computing*. This solution allows the user to configure a set of required functional units into the programmable hardware at deployment time, featuring a potentially infinite dynamic instruction set extension. The presence of reconfigurable hardware also allows reuse of silicon resources, reconfiguring the instruction set at run-time according to the currently executed algorithm.

In this paper, a new architectural model is proposed for an embedded processor based on reconfigurable computing. In our solution, we exploit a high degree of instruction level parallelism coupling a very-long instruction word (VLIW) processor, featuring a set of digital signal processing (DSP)-specific hardwired function units, with a custom designed gate array. The gate array is tightly integrated within the CPU instruction set architecture, behaving as part of both the control unit and the datapath. The processor, called *eXtended Instruction Set RISC* (XiRisc), is capable of executing a wide range of algorithms, including DSP functions, data encryption, telecommunication protocol handling, and multimedia elaboration. XiRisc significantly enhances the timing performance for a given computa-

Manuscript received April 1, 2003; revised June 26, 2003. This work was supported by STMicroelectronics.

The authors are with the Advanced Research Center for Electronic Systems (ARCES), University of Bologna, 40136 Bologna, Italy (e-mail: fcampi@deis.unibo.it).

Digital Object Identifier 10.1109/JSSC.2003.818292

tion, while reducing energy consumption with respect to low-power embedded processors. In comparison with other reconfigurable processor architectures, such results are achieved with a considerably smaller increase in silicon resource occupation and complexity in software development tools.

Section II overviews previous work related to processor and field-programmable gate array (FPGA) coupling. In Section III, we give a description of the computational model adopted and of the VLIW processor architecture. Section IV describes the configurable device in details both at the architectural and circuit level. Section V presents a silicon prototype of the processor, and the results achieved are discussed.

II. Related Work

The first experiments to couple a general purpose processor with an FPGA array in literature are probably the PRISM machine [5] and the Spyder machine [6]. In both cases, however, due to the limitations of FPGA technology at that time, the processor and the FPGA were located on separate chips, and the communication between the two was the bottleneck that severely limited the kind of applications that could benefit from these approaches.

The first architectures that can be defined as reconfigurable processors are PRISC [7], Chimaera [8], [9], and ConCISe [10]. In these examples, the reconfigurable array is tightly coupled to the processor core and limited to combinational logic only. Data is read and written directly to and from the processor register file, making the array an additional *function unit* in the processor pipeline. This makes the control logic simple, as almost no overhead is required in transferring data to the programmable hardware unit. The utilization of an integrated compiler tool is also eased by the fine grain of instructions mapped on the reconfigurable array. The boost in performance that can be achieved is severely limited by the combinational nature of the reconfigurable array.

Later attempts have been focused on introducing not combinatorial computation in the embedded gate array [11], [12]. In these cases, the allowed performance increase is more significant, but the definition of an integrated software development tool is not trivial. Furthermore, in many applications featuring this functional-unit approach, a severe bottleneck appeared in the access to data stored in memory. In all of the above described architectures, both the processor core and the embedded gate array had to be deeply modified in order to be coupled togheter. This prevents the easy reuse of existing commercial devices, thus severely increasing design costs.

To overcome the described limitations, later attempts have focused on a different architectural model, utilizing larger embedded configurable logic loosely coupled with existing standard processors. In the GARP machine [13], a custom-designed gate array works as a coprocessor for a standard million-instructions-per-second (MIPS) core. Data is exchanged between the two using dedicated *move* instructions, causing an overhead due to explicit communication. If the granularity of tasks mapped on the array is relatively high (in terms of required execution cycles), then the communication overhead may be considered negligible. The array can be considered a *configurable datapath* implementing *customized pipelines* that can be determined by the configuration. Each row implements a stage in the pipeline. This solution, if somehow imposing limits on the definition of possible array-based instructions, makes a direct implementation of a data flow graph quite straightforward, to the point that GARP can be programmed using a retargeted C compiler rather than involving HDL languages.

The Molen processor [14] is another example of reconfigurable architecture, where instructions are decoded by an arbiter determining which unit is targeted. "Normal" instructions are computed by the hardwired *core processor* (CP) while application-specific instructions are computed on the reconfigurable logic. One of the main points of interest is that the processor core does not need to be redesigned to support the reconfigurable unit. Nevertheless, the communication overhead introduced is comparable to that of processors based on the functional unit model like PRISC and Chimaera. Different from Garp and other previous attempts, Molen does not attempt to propose a mean for hardware/software co-compilation. In fact, tasks to be mapped on the programmable hardware unit are considered in the source code as atomic tasks, primitive operations *microcoded* in the processor architecture. Configurations are not determined by compilation, but defined as part of the processor design itself. This allows the architecture a large degree of freedom in the definition of the programmable array structure; in fact, Molen can exploit commercial FPGAs, taking advantage of the technology development in this field, while maintaining the basic architectural framework [15].

In conclusion, many solutions have been proposed in recent years, featuring various degrees of tradeoffs among performance, flexibility, and area. The availability of high-level programming tools (e.g., C language) not involving a hardware design (e.g., HDL description) is also a major factor that has to be taken into account.

A broad classification can be made according to the size of the hardware programmable logic and its degree of proximity to the CPU [16]:

- Loosely coupled architectures [13], [14] (coprocessor model) usually featuring large reconfigurable devices (often commercial FPGAs). Performance is improved in applications where it is possible to extract a computation-intensive coarse-grained task loosely interacting with the remaining application parts. Hardware/software partitioning is made manually, and the reconfigurable device is programmed using HDL languages.
- Tightly coupled architectures [7]–[12], [17] (functional-unit model), usually featuring smaller reconfigurable devices especially suited for fine-grained tasks strongly interacting with the processor execution flow. This allows good performances for a wider range of applications and the possibility of describing algorithms using high-level programming languages.

III. System Architecture

XiRisc is a VLIW processor based on the classic RISC five-stages pipeline [18]. It includes hardwired functional units

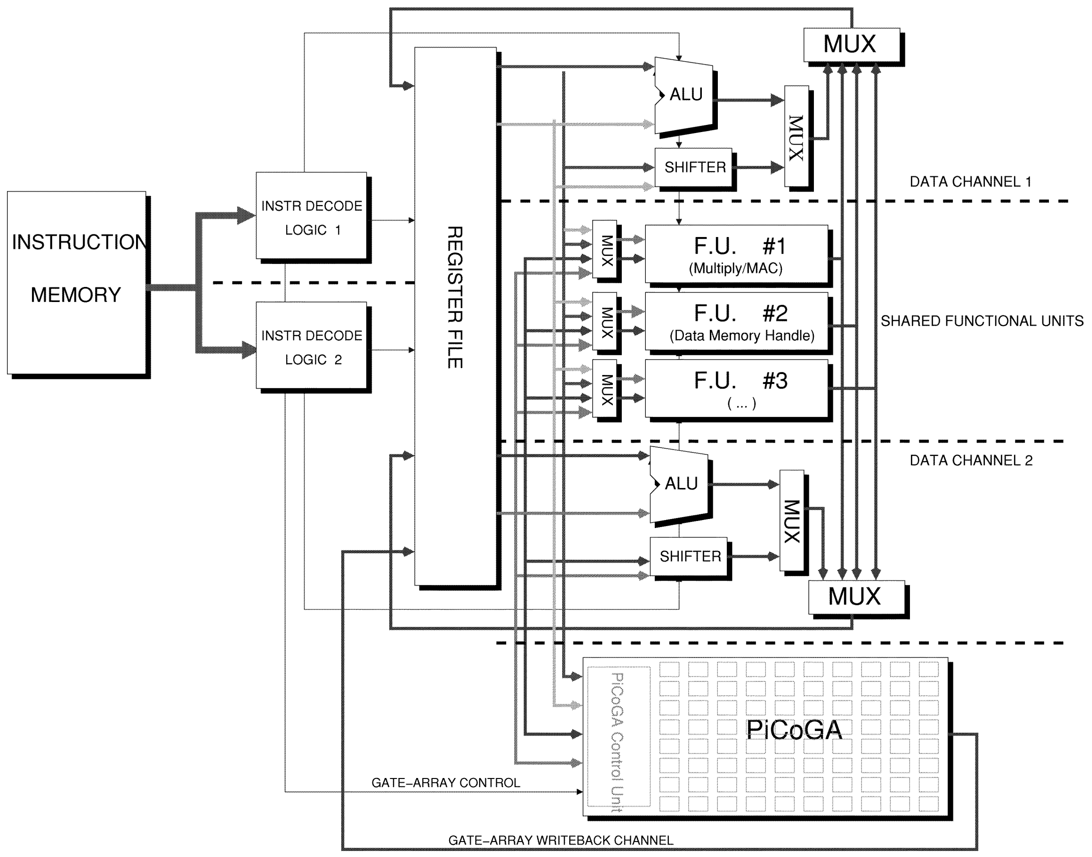

Fig. 2. System architecture.

for DSP calculations and an additional pipelined run-time configurable datapath (PiCo gate array, pGA or PiCoGA), acting as a repository of virtual application-specific functional units. XiRisc is a load/store architecture (Fig. 2), where all data loaded from memory are stored in the register file before they are actually computed by functional units. The processor fetches two 32-bit instructions each clock cycle, which are executed concurrently on the available functional units, determining two symmetrical separate execution flows called *data channels*. General-purpose functional units perform typical DSP calculations such as 32-bit multiply–accumulation, automatic hardware loop iteration, single-instruction–multiple-data (SIMD) ALU operations, and saturation arithmetic. The reconfigurable functional unit provides the capability of dynamically extending the processor instruction set with application-specific multicycle instructions, thus achieving run-time configurability. The architecture is fully bypassed, to maintain high data throughput through hardware resources.

The PiCoGA is tightly integrated in the processor core, just like any other functional unit, receiving inputs from the register file and writing back results to the register file, but differently than traditional functional unit in that more complex tasks can be executed. First, in order to better exploit instruction-level parallelism, the PiCoGA supports up to four source and two destination registers for each assembly instruction issued. Moreover, PiCoGA can hold an internal state across several computations, thus reducing the pressure on connection from/to the register file. Elaboration on the two hardwired data channels and the reconfigurable data path is concurrent, thus improving parallel computations. For instance, data memory access and PiCoGA elaboration may be done concurrently, thus reducing the memory bandwidth bottleneck. Synchronization and consistency between program flow and PiCoGA elaboration is granted by hardware stall logic based on a register locking mechanism, which handles read-after-write hazards.

Dynamic reconfiguration is handled by a special assembly instruction, which loads a configuration inside the array reading from an on-chip dedicated memory called *configuration cache*. In order to avoid stalls due to reconfiguration when different PiCoGA functions are needed in a short time span, several configuration may be stored inside the array, and are immediately available. Thus, the processor instruction set has been extended with two types of instructions, as shown in [19]:

- pGA-load, which loads a configuration inside the PiCoGA;
- pGA-op, which starts the computation of an application-specific function stored in the array.

The proposed computational model takes advantage of the synergy between different application specific functional units tightly integrated into the same core. A reconfigurable device behaving as a coprocessor needs to implement an entire computational kernel to achieve high throughput because the communication overhead to the processor core is otherwise considerable. As a consequence, when a kernel is composed of both functions suitable to be mapped in a reconfigurable device and operators which could not be efficiently implemented, it is often completely computed in the processor core, leaving the array

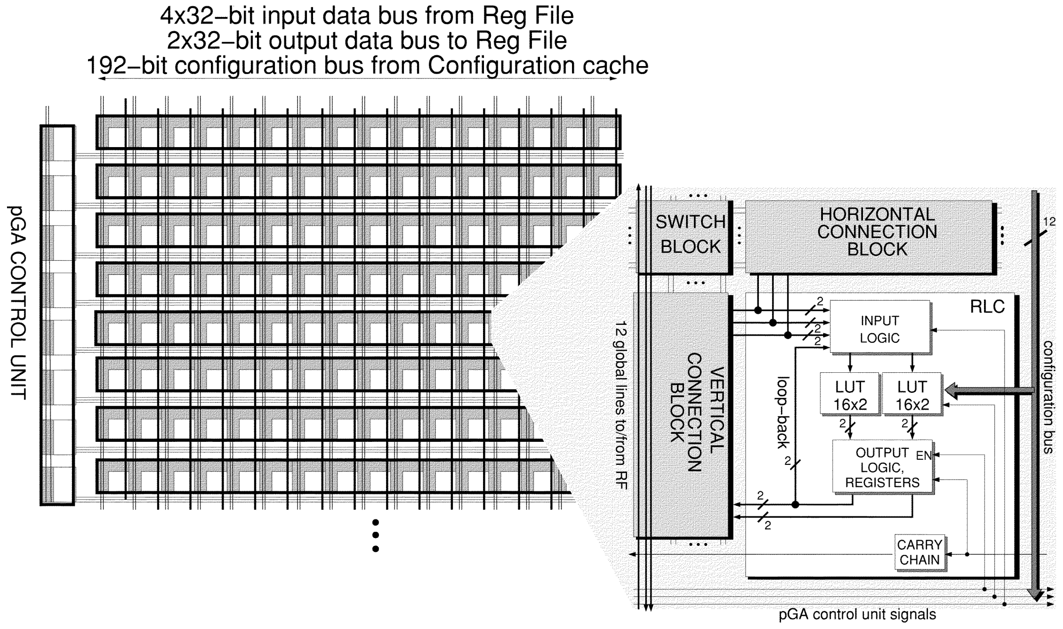
4x32−bit input data bus from Reg File
2x32−bit output data bus to Reg File
192−bit configuration bus from Configuration cache
pGA CONTROL UNIT
SWITCH BLOCK
HORIZONTAL CONNECTION BLOCK
VERTICAL CONNECTION BLOCK
12 global lines to/from RF
loop−back
RLC
INPUT LOGIC
LUT 16x2
LUT 16x2
OUTPUT LOGIC, REGISTERS
EN
CARRY CHAIN
configuration bus
12
2
pGA control unit signals

Fig. 3. PiCoGA structure.

unused. In our model, the communication overhead between the PiCoGA and the other functional units is small, thus allowing to distribute the different operations included in a single kernel to the functional unit that fits them best. Wide multipliers, variable shifters, and medium access controls (MACs), which are so difficult to implement efficiently in traditional reconfigurable devices, could be executed in dedicated hardwired functional units, while the configurable unit exploits parallelism of even small portion of kernels. In this way, the use of the PiCoGA increases considerably, justifying its cost in terms of area for a wide range of applications.

IV. Pipelined Configurable Gate Array

In the past, a few attempts have been carried out in order to design a configurable unit tightly integrated in a processor core. Their study led to some guidelines that have to be followed to achieve a significant gain in the performance of the overall system.

First, the configurable unit should be able to perform complex functions that require multicycle latency. The PiCoGA is designed to implement a peculiar pipeline where each stage corresponds to a piece of computation, so that high-throughput circuits can be mapped. The array is also provided with a control unit which controls pipeline activity, just as if it were an additional datapath. A sequence of PiCoGA instructions can then be processed, filling the pipeline in order to exploit parallelism.

Moreover, the configurable unit should preserve its state across instruction executions. A new PiCoGA instruction may use the results of previous ones stored on the array, thus reducing the pressure on the register file. Since most of the bit-level control logic would be computed in the standard processor pipeline, the configurable unit should have a granularity suitable for multibit datapath implementations. At the same time, the PiCoGA should be flexible enough to compensate the other functional units for the kind of computations that are not efficient.

Finally, a tight integration in the processor core gives the opportunity to use the PiCoGA in many different computational cores. Therefore, run-time reconfiguration is necessary to support new sets of dynamically defined instructions.

A. *PiCoGA Structure*

The PiCoGA is an array of rows, each representing a possible stage of a customized pipeline. The datapath width should comply with the processor data width, so each row is able to process 32-bit operands. As shown in Fig. 3, each row is connected to other rows with configurable interconnect channels and to the processor register file with six 32-bit global busses. In a single cycle, four words can be received from the register file and up to two words can be produced for writeback operations. The busses span the whole array, so that any row can access them, improving routability.

Pipeline activity is controlled by a dedicated configurable control unit, which generates three signals for each row of the array. The first one enables computation on the pipeline stage, allowing the registers in the row to sample new data. In every cycle, only rows having input data ready are activated. In this way, a state stored in flip-flops inside the array can be correctly held and at the same time unnecessary power dissipation is avoided. The second signal controls initialization steps of a state held inside the array, while the third enables a burst write of lookup tables (LUTs) with data available in the processor register file.

Each row is composed of 16 reconfigurable logic cells (RLCs) and a configurable horizontal interconnect channel. Vertical channels have 12 pairs of wires, while horizontal ones have only eight pairs of wires. Switch blocks adjacent to each RLC connect vertical and horizontal wires.

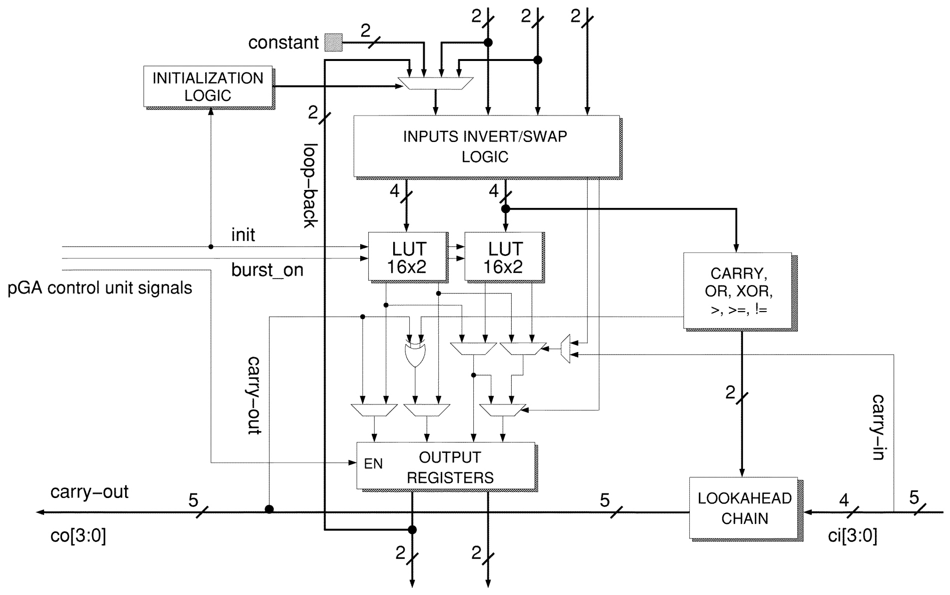

Fig. 4. Reconfigurable logic cell structure.

Since most of the remaining portion of control logic not mapped in the processor standard dataflow is implemented in the configurable control unit, the array core can be data-path oriented. Therefore, the PiCoGA has a 2-bit granularity for both interconnections and LUTs, except for input connection blocks which have 1-bit granularity. This can be considered a good tradeoff, since bit-level operators such as bit permutation, which are frequent in cryptography algorithms, are not well supported by other functional units.

B. Configuration Caching

Since the PiCoGA is tightly integrated in the processor core, it can be frequently used for many different computational kernels. Reconfiguration of traditional FPGAs can take hundreds or most frequently thousands of clock cycles, depending on the reprogrammed region size. Although computation can still continue on other processor resources, scheduling will hardly find enough instructions to avoid stalls. This could overcome any benefit from the use of dynamically configurable arrays. Furthermore, in some algorithms the function to be implemented is only known at the time it has to be executed, so that no preventive reconfiguration can be performed. In such cases, many computational kernels can hardly take advantage of the presence of a configurable unit.

Three different approaches have been adopted to overcome these limitations. First, the PiCoGA is provided with a first-level cache, storing four configurations for each reconfigurable logic cell [20], [21]. Context switch takes place in a single clock cycle, providing four immediately available PiCoGA instructions. Further increases in the number of functions simultaneously supported by the array can be obtained exploiting partial run-time reconfiguration (PRTR), which gives the opportunity for reprogramming only the portion of the PiCoGA needed by the configuration. As a consequence, different configurations can be simultaneously loaded in different regions of the same context of the array.

The PiCoGA may concurrently execute one computation instruction and one reconfiguration which configures the next instruction to be performed. By doing so, miss occurrences should be highly reduced, even when the number of used configurations is large.

Finally, reconfiguration time can be shortened exploiting a wide configuration bus to the PiCoGA. The RLCs in a row are programmed concurrently throug 192 dedicated wires, taking up to 16 cycles to have a complete reconfiguration. A dedicated second-level cache on chip (configuration cache) is needed to provide such a wide bus, while the whole set of available functions can be stored in an off-chip, possibly nonvolatile memory.

C. Reconfigurable Logic Cells

An RLC is composed of a cluster of two LUTs (Fig. 4). LUTs have 2-bit granularity, that is, 4-bit inputs and 2-bit outputs (4:2). A total of six inputs from the configurable interconnect channels are provided to the RLC which can be used to implement logic functions with different granularity combining the two LUTs together. A two level multiplexing stage controlled by two of the inputs performs the combination of LUT outputs. By doing so, mapping of either a 6:1, a 5:2, or a 4:4 logic function is allowed. An RLC contains four registers, one for each output, which are controlled by the configurable control unit.

RLC outputs are internally routed back to the input block, in order to implement the cascade of two LUTs or logic holding a state such as accumulators. A block controlled by the control unit is introduced on the feedback path to support different kinds of state initialization. At first, a 4:1 multiplexer is set to propagate either a constant value or a value coming from one of the RLC inputs, providing an initial value for the static variable held. Then, as soon as the control unit determines that the initial value is no longer needed, data coming either from the internal loop or from another RLC input are propagated by the multiplexer for normal operations.

A single RLC can implement a 2-bit adder using the two LUTs to compute both results with carry-in equal to 0 and 1. The

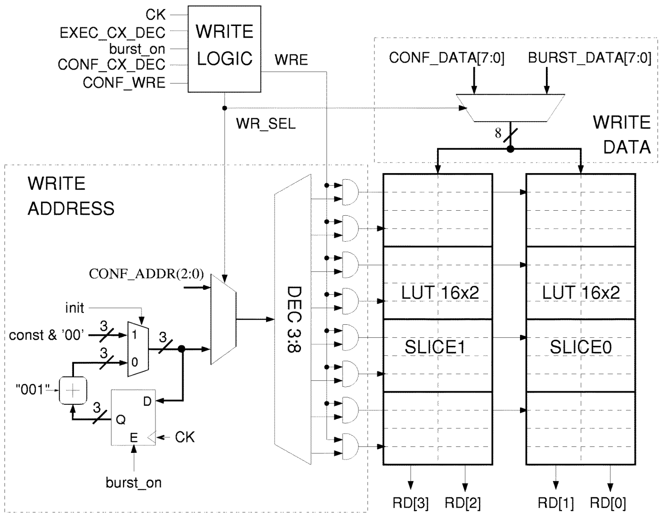

Fig. 5. Lookup table configuration and burst write.

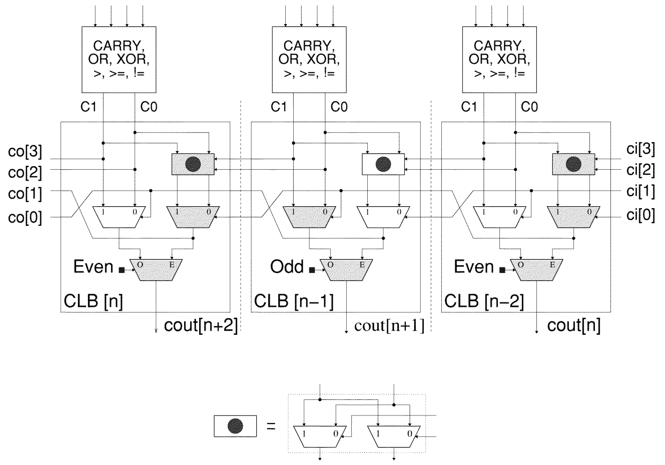

Fig. 6. Level-one lookahead carry chain logic.

same multiplexers, which combine LUT outputs, can be used to propagate the right result on the base of the carry-in value in a carry–select fashion. At the same time, a dedicated carry generation block computes sum carry-out signals, both in the case of carry-in equal to 0 and 1, which are fed into a carry chain block performing fast lookahead logic. Since the implementation of a 2-bit adder needs only two RLC outputs as result, the other two outputs can be used to route the final carry-out and the overflow bit, in the case of signed operands. Besides sum carry logic, other bit-serial computations have been introduced in the carry generation block, which could take advantage of the fast propagation chain, such as comparison, OR and XOR operators. Exploiting the input block of the RLC to invert signals, even more operators can be efficiently mapped in the same way.

Since LUTs are well suited for compactly storing data inside the PiCoGA, a mechanism for writing them at execution time has been provided (Fig. 5). Each LUT is addressed for configuration as a memory with four 8-bit words, in order to have fast reconfiguration of the array. Using the same addressing, a burst write mechanism is provided which is able to store four 2-bit data from the global lines in one clock cycle. It is thus possible to store all 32-bit registers of the processor register file into a single PiCoGA row in only eight clock cycles. Address generation for burst write is performed by a 3-bit counter controlled by two signals provided by the control unit for initialization (*init*) and write (*burst on*) phases.

D. Carry Chain Logic

Each RLC contains a dedicated carry chain block performing level-one lookahead logic. Dedicated wires along each row, directly connecting configurable cells, are also provided to have fast propagation of carry signals.

A standard carry–select architecture is implemented exploiting a 2-to-1 multiplexer, driven by the carry-in signal coming from the previous RLC, which selects the correct carry-out. If we consider the implementation of a $2N$-bit adder, the critical path delay of a simple carry-select architecture passes through N multiplexers, one for each RLC used. A level-one lookahead technique has been applied in order to reduce roughly by one half the total number of cascaded multiplexers. Adopting this kind of architecture, a group of RLCs in a row implementing an adder need to be characterized as a sequence of *even* RLCs alternated with *odd* ones. Odd and even RLCs use different logic for carry propagation. However, in order to maintain the placement of any unconstrained adder in a row, we designed identical RLCs, implementing both odd and even branches of the chain. An additional multiplexer is required to select which of the two branches is actually used, while four more signals dedicated to the chain have to be added to the standard carry-out signal. In Fig. 6, the utilized multiplexers are highlighted, showing that the critical path passes only through the multiplexer of even RLCs.

Special care is needed in the configuration of the RLC computing the least significant bit (LSB). It must be an odd-type RLC, and its carry generation block must be configured to produce identical outputs, in order to ignore carry chain input signals coming from the two previous RLCs.

With the proposed carry chain, the longest path passes through $(N/2)+2$ multiplexers if N is even and $((N+1)/2)+2$ if N is odd. Even though a standard-cells synthesis and automatic back-end flow was adopted for the design of RLC logic, comprising carry logic, a 32-bit addition at 150 MHz in a 0.18-μm technology was achieved.

E. Decoder-Based Multicontext Interconnections

In typical FPGAs, each switch connecting two lines in the routing channel is individually driven by a dedicated SRAM cell which stores a configuration bit. In the case of multicontext arrays, each SRAM needs to be replicated k times, in order to have k immediately available configurations, thus considerably increasing area occupation.

The architecture proposed is based on the introduction of a decoding stage for each line possibly connecting to wires in the routing channel (see Fig. 7), in order to reduce the number of configuration memories. If we consider the case of an input line to a logic block which can be connected to n wires of the routing channel, the number of memories needed becomes $m = k[\log_2 n]$ instead of kn. With regard to delays, the scheme adopted does not increase them with respect to the typical case, as only one pass transistor is passed when connecting two lines;

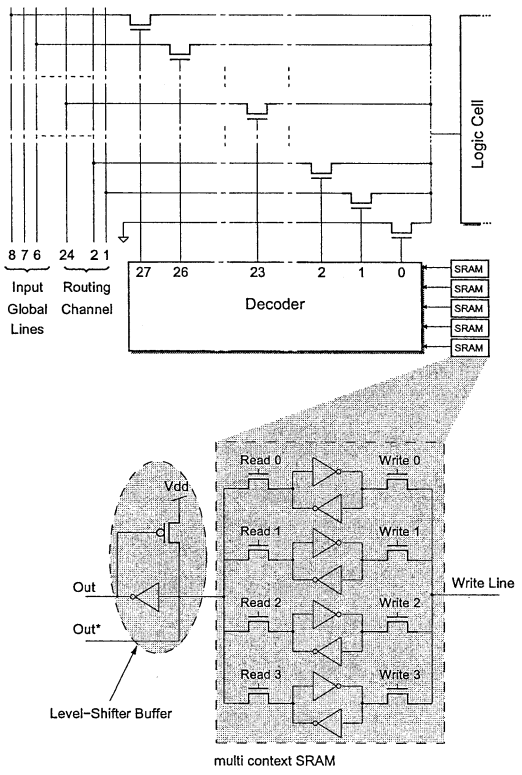

Fig. 7. Decoder-based multicontext structure applied to an *RLC* input.

on the contrary, the reduction in area should also reduce parasitic capacitance and delays.

The application of this approach to an output of a logic block obviously reduces routability, as only a single wire of the routing channel can be connected to the line from the RLC. However, in [22], the impact of a similar structure on routability was investigated, showing that only a small penalty has to be paid.

In order to achieve considerable area reduction, both memory cells and decoder have been carefully designed. Since memory area becomes more and more important as the number of contexts grows, single-ended cells were used to compose a 1-bit multicontext SRAM as shown in Fig. 7. Two different lines are provided for writing and reading, so that reconfiguration of a context can take place while another one is computing. In order to keep the pass gate used for writing small, the traditional six-transistors scheme has been extended with an additional nMOS (Fig. 8) which can be turned off when a high value needs to be stored. This solution offers a much more compact layout than the one designed simply increasing the pass gate width, and shows to be even more convenient when scaling to 0.13-μm technology.

With regard to the decoder, a special circuit has been designed in order to have minimum area occupation even at the expense of increased latency. In Fig. 9, the schematic of a 3-8 decoder is depicted where all transistors are minimum sized; similar schemes can obviously be obtained for any number n of lines that need to be connected. The decoder structure is based on an nMOS pull-down net and a pMOS tree, which minimizes the number of transistor in the pull-up net. Since in classic schemes most of the area occupation is due to the pull-down net, which needs $m = [\log_2 n]$ nMOS transistors for each decoder output, a different solution has been adopted. As depicted in Fig. 9, decoder outputs are connected two by two with $m - 1$ pass transistors so that only one more nMOS transistor is needed as a pull-down circuit for each output.

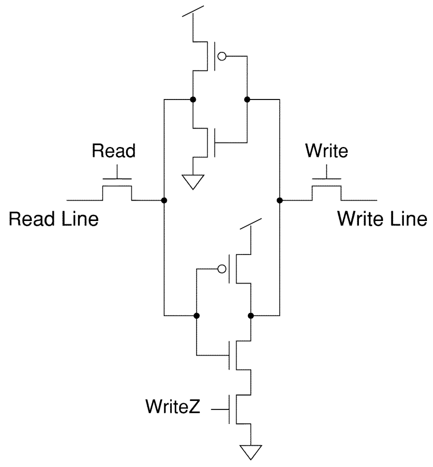

Fig. 8. SRAM cell schematic for each context.

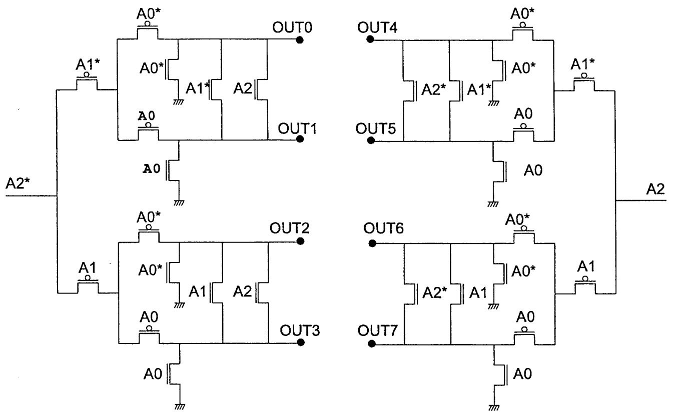

Fig. 9. 3:8 decoder scheme.

V. Results

Several digital signal processing algorithms (Table I) were implemented on the XiRisc processor and tested on a prototype chip, in order to measure power consumption, timing performance, and area cost. Area occupation is a major issue for reconfigurable architectures, especially those based on the coprocessor model, since computation-intensive portions of algorithms are entirely mapped on the embedded programmable device. On the other hand, XiRisc allows the programmer to choose an appropriate hardware/software partitioning, ensuring flexibility in the area–performance tradeoff. This is shown in Table II, where different implementations of the same algorithm are presented, exploiting various degrees of parallelism. A 24-row PiCoGA is compared with a 48-row one, considering that area occupations in a 0.18-μm technology are: 1.2 mm^2 for the VLIW core, 9 mm^2 for instructions and data cache, 2 mm^2 for configuration cache and PiCoGA interface logic, and 0.6 mm^2 for each PiCoGA row. The area increase shown in Table II compares a standard VLIW processor (including instruction and data caches) with a XiRisc processor (adding the

TABLE I
PiCoGA AREA REQUIRED AND SPEED-UP FOR SOME SIGNAL PROCESSING ALGORITHMS

Algorithm	Rows occupation	Speed-up
DES encryption	5	13.5x
Turbo Decoder	20	11.9x
CRC	11	4.3x
Median filter (parallel) (24bit word, 8 samples)	32	7.7x
Median filter (sequential) (8bit word, 256 samples)	17	6.2x
Motion estimation (MPEG)	24	12.4x
Motion prediction (MPEG)	12	4.5x

TABLE II
AREA VERSUS SPEED-UP TRADEOFF

		24 rows	48 rows
Area Increase		2.6x	4x
Speed-up	Turbo Decoder	11.9x	18x
	Motion estimation	12.4x	20x
	Median Filter (parallel)	N.A.	7.7x

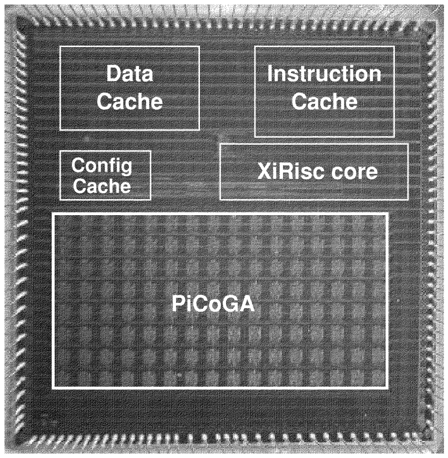

Fig. 10. Chip micrograph.

area contributions due to PiCoGA, configuration cache and interface logic). Analyzing these numbers and speed-up figures, a 24-row PiCoGA seems to be a good tradeoff for target applications. Considering technology scaling (i.e., < 0.1 μm), the best tradeoff will move toward implementations with more rows, thus increasing the advantages of the proposed architecture.

A prototype test chip (Fig. 10) that couples the VLIW processor with an eight-row PiCoGA has been fabricated using 0.18-μm 1.8-V, six-metal-layers CMOS technology. This is adequate for simple mappings (e.g., DES) and for basic measurements. This prototype has a large area overhead due to the presence of testing structures and layout inefficiencies, which requires a row area of 1.9 mm^2. Information concerning the chip is summarized in Table III.

A software development environment based on a customization of the GNU-Gcc toolchain [24] has been used to support

TABLE III
TEST CHIP INFORMATION

Process Technology	0.18 μm CMOS Process, 6 Metal Layers
Power supply	1.8V
Clock frequency	80MHz
Power consumption	120mW (average) processor core and memories 16mW/row active PiCoGA rows
Number of transistors	12M
Memory size	64Kbyte Instruction cache 64Kbyte Data cache 24Kbye Configuration cache
Die size	8x8 mm^2

TABLE IV
POWER CONSUMPTION FOR A STANDARD VLIW PROCESSOR

Algorithm	Energy Cons.	Processor Core	Data Mem.	Instr. Mem.
DES (64 bit)	2.47μJ (1444 cycles)	24%	39%	37%
CRC (1Kbyte)	23.14μJ (14385 cycles)	23%	38%	39%
Median Filter (40 samples, (window size 8)	7.24μJ (4381 cycles)	21%	41%	38%

TABLE V
ROW ACTIVITY RATE

Algorithm	Row activity rate
DES	4.0%
Turbo Decoder	2.1%
Motion prediction	1.3%
Median Filter (parallel)	1.75%

architecture programming and benchmarking. The availability of a software profiling environment offers an appropriate mean to manually determine critical computation kernel that should be implemented on the PiCoGA.

Table I shows speed-ups for several algorithms, calculated through VHDL logic simulations and confirmed by experimental results. Comparisons are made counting the number of execution cycles with respect to a DSP-like architecture, namely, a standard RISC enhanced by the most common DSP features. These figures, ranging from 4.3$\times$ to 13.5$\times$, prove the flexibility of the presented architecture which is effective for a wide range of different algorithms.

A special effort was made for low-power architectural and circuit design, since this is a key issue for embedded applications. By analyzing the main sources of power consumption for standard processor (reported in Table IV), it is clear that the main contribution (about 75%) is due to memory accesses (instruction and data), and every tested algorithm roughly presents the same distribution. These results were obtained through logic simulations, and are confirmed by measurements made on a previous prototype [25] which does not include the

TABLE VI
POWER CONSUMPTION FOR XiRisc PROCESSOR

Algorithm	Energy Cons.	Processor Core	Data Mem.	Instr. Mem.	PiCoGA
DES (64 bit)	0.187μJ (106 cycles)	22%	38%	36%	4%
CRC (1Kbyte)	6.42μJ (3352 cycles)	14%	29%	33%	24%
Median Filter (40 samples, (window size 8)	1.06μJ (4381 cycles)	20%	38%	34%	8%

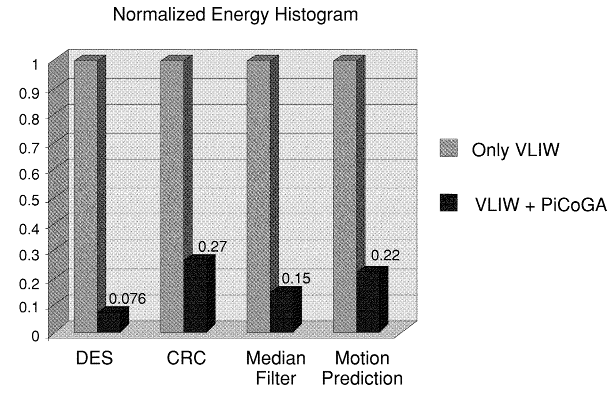

Fig. 11. Normalized energy consumption histogram.

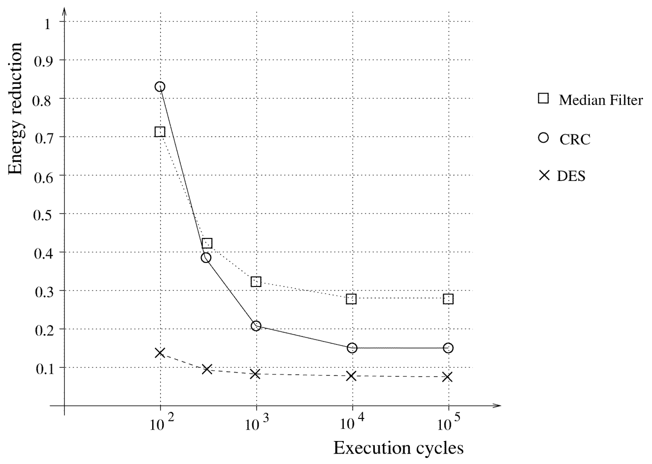

Fig. 12. Energy consumption versus number of clock cycles between two reconfigurations.

PiCoGA. Since a new instruction is fetched every clock cycle, the only way to reduce instruction memory energy consumption is to reduce the number of execution cycles. In fact, instruction memory consumption scales proportionally with speedup, therefore, a VLIW architecture enhanced by the PiCoGA achieves both speedup and instruction memory consumption reduction. Several tests have shown that access to data memory roughly scales with speedup. In fact, the execution on PiCoGA allows an improved efficiency for data management (e.g., data stored locally on PiCoGA), thus reducing data memory consumption. Additional power consumption due to PiCoGA computation is small compared to the overall value. In fact only PiCoGA rows involved in computations and activated by the row control unit require additional energy, while the unused portions of the PiCoGA are kept inactive. Measurements show that the average energy consumption for a computation which involves a row in one cycle is 200 pJ. Since the average PiCoGA activity rate is very low and each reconfigurable cell is unused for the most time, this technique is very effective. Activity rate values, obtained through logic simulations, are shown in Table V.

Combining simulation values from Table V with power consumption measurements of a PiCoGA row and the remaining XiRisc parts, it is possible to give a reliable PiCoGA consumption estimate also for algorithms that do not fit on the prototype chip, due to its limited number of rows. The final results are summarized in Fig. 11 and in Table VI, which show the advantages of the proposed architecture, compared with standard DSP, with an energy consumption reduction up to 92%.

A further contribution to energy consumption is due to dynamic reconfiguration, but this is not present during normal execution. In fact, since reconfiguration happens only once at the beginning of PiCoGA computations, the overall average power consumption depends on the number of execution cycles. Fig. 12 depicts this relation, and shows that the impact of reconfiguration consumption becomes negligible when a given configuration is used for more than 1000 cycles. Typically, a configuration is active for a far larger number of cycles. For instance, 1000 cycles are needed to encrypt 80 bytes with DES algorithm or to compute the parity check of 300 bytes. Therefore, in most situations, its possible to neglect reconfiguration overhead, considering only execution energy.

VI. CONCLUSION

A new architecture for reconfigurable computing, tightly integrating a run-time reconfigurable pipelined datapath (PiCoGA) with a VLIW processor core, has been presented. A prototype chip has been implemented, and testing proved the flexibility of this approach, allowing a more efficient elaboration of a wide range of signal processing algorithms. Through application-specific instructions mapped on pGA, speedups ranging from 4.3$\times$ to 13.5$\times$ are achieved, while instruction and data memory accesses are reduced allowing an energy consumption reduction up to 92%.

ACKNOWLEDGMENT

The authors would like to thank L. Lavagno, C. Passerone, and A. La Rosa of the Politecnico di Torino for software tools development.

REFERENCES

[1] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, 623–656, July, Oct. 1948.

[2] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, Apr. 1965.

[3] J. M. Rabaey, "Silicon platforms for the next generation systems—What role does reconfigurable hardware play?," in *Proc. 9th Int. Workshop Field Programmable Logic and Applications*, Aug. 2000, pp. 277–285. LNCS 1896.

[4] A. DeHon, "The density advantage of reconfigurable computing," *IEEE Computer*, vol. 33, pp. 41–49, Apr. 2000.

[5] P. Athanas and H. Silverman, "Processor reconfiguration through instruction-set metamorphosis," *IEEE Computer*, vol. 26, pp. 11–18, Mar. 1993.

[6] C. Iseli and E. Sanchez, "Spyder: a SURE (SUperscalar and REconfigurable) processor," *J. Supercomput.*, vol. 9, no. 3, pp. 231–252, 1995.

[7] R. Razdan and M. Smith, "A high-performance microarchitecture with hardware-programmable functional units," in *Proc. 27th Annu. Int. Symp. Microarchitecture*, Nov. 1994, pp. 172–180.

[8] S. Hauck, T. Fry, M. Hosler, and J. Kao, "The Chimaera reconfigurable functional unit," in *Proc. IEEE Symp. FPGAs for Custom Computing Machines*, Napa Valley, CA, Apr. 1997, pp. 87–96.

[9] Z. A. Ye, N. Shenoy, and P. Banerjee, "A C compiler for a processor with a reconfigurable functional unit," in *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, Feb. 2000, pp. 95–100.

[10] B. Kastrup, A. Bink, and J. Hoogerbrugge, "ConCISe: a compiler-driven CPLD-based instruction set accelerator," in *Proc. 7th Annu. IEEE Symp. Field-Programmable Custom Computing Machines*, Napa Valley, CA, Apr. 1999, pp. 92–100.

[11] R. Wittig and P. Chow, "OneChip: an FPGA processor with reconfigurable logic," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, Napa Valley, CA, Mar. 1996, pp. 126–135.

[12] J. Jacob and P. Chow, "Memory interfacing and instruction specification for reconfigurable processors," in *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, Monterey, CA, Feb. 1999, pp. 145–154.

[13] J. R. Hauser and J. Wawrzynek, "Garp: a MIPS processor with a reconfigurable coprocessor," in *Proc. 1997 IEEE Symp. Field Programmable Custom Computing Machines*, 1997, pp. 12–21.

[14] S. Vassiliadis, S. Wong, and S. Coţofană, "The MOLEN $\rho\mu$-coded processor," in *Proc. 11th Int. Conf. Field Programmable Logic and Application (FPL)*, 2001, pp. 275–285.

[15] S. Wong, S. Vassiliadis, and S. Coţofană, "Future directions of (programmable and reconfigurable) embedded processors," in *Proc. 2nd Workshop System Architecture Modeling and Simulation (SAMOS2002)*, 2002, pp. 1–18.

[16] W. H. Mangione-Smith, B. Hutchings, D. Andrews, A. DeHon, C. Ebeling, R. Hartenstein, O. Mencer, J. Morris, K. Palem, V. K. Prasanna, and H. A. E. Spaanenburg, "Seeking solutions in reconfigurable computing," *IEEE Computer*, vol. 30, pp. 38–43, Dec. 1997.

[17] J. Goodman and A. P. Chandrakasan, "An energy-efficient reconfigurable public-key cryptography processor," *IEEE J. Solid-State Circuits*, vol. 36, pp. 1808–1820, Nov. 2001.

[18] D. Patterson and J. Hennessy, *Computer Architecture: A Quantitative Approach*. San Mateo, CA: Morgan Kaufmann, 1996.

[19] F. Campi, R. Canegallo, A. Cappelli, R. Guerrieri, A. La Rosa, L. Lavagno, A. Lodi, C. Passerone, and M. Toma, "A reconfigurable processor architecture and software development environment for embedded systems," presented at the Reconfigurable Architectures Workshop, Nice, France, Apr. 2003.

[20] A. DeHon, "DPGA-coupled microprocessors: Commodity ICs for the early 21st century," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, Napa Valley, CA, Apr. 1994, pp. 31–39.

[21] S. Trimberger, D. Carberry, A. Jhonson, and J. Wong, "A time multiplexed FPGA," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, Napa Valley, CA, Apr. 1997, pp. 34–40.

[22] V. Baena-Lecuyer, M. A. Aguirre, A. Torralba, L. G. Franquelo, and J. Faura, "Decoder-driven switching matrices in multicontext fpgas: Area reduction and their effect on routability," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 1, 1999, pp. 463–466.

[23] A. Lodi, M. Toma, F. Campi, A. Cappelli, R. Canegallo, and R. Guerrieri, "A pipelined configurable gate array for embedded processors," in *Proc. ACM Symp. Field-Programmable Gate Arrays*, Feb. 2003, pp. 21–30.

[24] A. La Rosa, L. Lavagno, and C. Passerone, "A software development tool chain for a reconfigurable processor," in *Proc. Int. Conf. Compilers, Architecture and Synthesis for Embedded Systems*, 2002, pp. 93–98.

[25] F. Campi, R. Canegallo, and R. Guerrieri, "IP-reusable 32-bit VLIW Risc core," in *Proc. 27th Eur. Solid State Circuits Conf.*, Sept. 2001, pp. 456–459.

Andrea Lodi received the Electrical Engineering and the Ph.D. degrees from the University of Bologna, Bologna, Italy, in 1998 and 2002, respectively.

Since 1998, he has been a Consultant for STMicroelectronics in the fields of signal-processing algorithms and innovative architectures of systems-on-chips and reconfigurable devices. He is currently with the Advanced Research Center on Electronic Systems (ARCES), Bologna, Italy.

Mario Toma received the Dr.Eng. degree in electronics and the Ph.D. degree from the University of Bologna, Bologna, Italy, in 1998 and 2002, respectively.

Since 1999, he has been a Consultant for STMicroelectronics for the application of innovative CAD CMOS design platforms on digital system-on-chip design. He is currently with the Advanced Reasearch Center on Electronic Systems (ARCES), Bologna, Italy.

Fabio Campi received the M.Sc. degree in microelectronics and the Ph.D. degree in electronics and computing science from the University of Bologna, Bologna, Italy, in 1999 and 2003, respectively.

In 1995 and 1996, he was with the Tampere University of Technology, Tampere, Finland, as Visiting Student. Since 1999, he has been a Consultant for Central Research and Development, STMicroelectronics, for the application of innovative CMOS design platforms on digital system-on-chip design. He is currently with the Advanced Reasearch Center on Electronic Systems (ARCES), Bologna. His main research interests are VLSI system-on-chip design, embedded microprocessors, and development of advanced architectures and algorithms for digital signal processing.

Andrea Cappelli received the Dr.Eng. degree in electrical engineering form the University of Bologna, Bologna, Italy, in 2002, where he is currently working toward the Ph.D. degree.

Since 2002, he has also been a Consultant for STMicroelectronics.

Roberto Canegallo received the degree in electronic engineering from the University of Pavia, Pavia, Italy.

From 1992 to 1999, he was with STMicroelectronics, Agrate Brianza, Italy, conducting research on a wide variety of topics in mixed-analog systems, such as optical character recognition, image sensors, and multilevel nonvolatile Flash memories. In 1999, he joined the joint Laboratory ST/University of Bologna, Bologna, Italy. His current research interests include the development of three-dimensional high-bandwidth chip-to-chip communication.

Roberto Guerrieri received the Electrical Engineering and the Ph.D. degrees from the University of Bologna, Bologna, Italy.

He is currently an Associate Professor in electrical engineering with the University of Bologna. Beginning in 1986, he was visiting the Department of Electrical Engineering and Computer Science, University of California at Berkeley, and the Department of Electrical Engineering at the Massachusetts Institute of Technology, Cambridge. He has published more than 90 papers in various fields including numerical simulation of semiconductor devices, numerical solution of Maxwell's equations, and parallel computation on massively parallel machines. Recently, his work has focused on integrated silicon systems to solve various problems such as optical and capacitive smart sensors, integrated digital circuits for speech and video processing and analog circuits for fuzzy controllers. In 1998, he became Director of the Laboratory for Electronic Systems, a joint venture of the University of Bologna and STMicroelectronics for the development of innovative designs of systems on chip.

Dr. Guerrieri received the Best Paper Award from the IEEE in 1992 for his work in the area of process modeling.