Journal of VLSI Signal Processing 44, 135–152, 2006
© 2006 Springer Science + Business Media, LLC. Manufactured in The Netherlands.
DOI: 10.1007/s11265-006-7512-7

# Reconfigurable Coprocessor for Multimedia Application Domain

SALVATORE M. CARTA
*DMI—Department of Mathematics and Computer Science, University of Cagliari, Palazzo delle Scienze, Via Ospedale 72, 09124, Cagliari, Italy*
*e-mail: salvatore@unica.it*

DANILO PANI
*DIEE—Department of Electrical and Electronic Engineering, University of Cagliari, P.zza d'Armi, 09123, Cagliari, Italy*
*e-mail: pani@diee.unica.it*

LUIGI RAFFO
*DIEE—Department of Electrical and Electronic Engineering, University of Cagliari, P.zza d'Armi, 09123, Cagliari, Italy, and with INFM-S3*
*e-mail: raffo@unica.it*

**Published online:** 27 May 2006

**Abstract.** A new reconfigurable architectural template is presented. Such a template is composed of coarse-grained and fine-grained reconfigurable datapath and control to obtain performances at custom designed chip level. To show the adaptability/performance of such architectural template, the architecture has been customized (i.e. datapath and control features of the template have been properly sized) for multimedia application domain. To evaluate complexity and maximum clock frequency of the proposed architecture, it has been synthesized using Synopsys Design Compiler on a standard-cell 0.18 $\mu$m technology. Estimated number of transistors is 335 K, while maximum allowable frequency is 460 MHz. Performances have been evaluated comparing the number of clock cycles and the processing time required to process application domain dominant kernels with commercial devices: we obtained up to 95% reduction with respect to ARM and up to 94% reduction with respect to TMS320C5510 in terms of clock cycles.

**Keywords:** reconfigurable computing, digital signal processing, domain-specific architectures, multimedia

## 1. Introduction

Reconfigurable Computing Systems (RCSs), also called adaptive computing systems, are processing systems with redundant computational and connectivity resources, able to change their functionality at run-time, according to the task in execution in that moment. RCSs represent a trade-off between the maximum flexibility ensured by general purpose microprocessor or digital signal processors and the maximum efficiency in terms of speed and power dissipation offered by Application Specific Integrated Circuits (ASICs) [1, 2]. The use of reconfigurable redundant hardware

Corresponding author: Salvatore M. Carta, DMI—University of Cagliari, Palazzo delle Scienze, Via Ospedale, 72 - 09124, Cagliari, Italy
e-mail: salvatore@unica.it

resources gives the possibility of: (i) maximizing execution speed by the use of specialized parallel or pipelined hardware architectures suited for the target application, (ii) minimizing power dissipation switching-off unused hardware resources and exploiting locality of computation.

We propose a new reconfigurable architectural template (section 2) which exploits mixed coarse-grained and fine-grained [3] reconfigurable datapath and control elements to obtain performances at ASICs level on computational tasks based on repetitive execution of a reduced set of operations on multidimensional array data (vectors or matrix). Such kind of tasks are the core of very common algorithms for image and voice processing and encryption, and since they are usually computationally expensive, they normally represent the dominant kernels of them (i.e. kernels that determine the time performance of the system for the task considered) [4]. The architectural template determines: (i) execution partitioning between a reconfigurable coprocessor in charge of execute dominant kernels and a standard host processor in charge of execute not-dominant kernels and control the system, (ii) processor-coprocessor interaction. System integrability and scalability is maximized by the use of memory mapping for coprocessor/processor communications. In this way coprocessor can be easily coupled with each kind of existing processor, and a cluster of coprocessors in parallel can be used to implement thread-level parallelism.

These features are not in general a pure innovation: other reconfigurable platforms in literature use one or more of them. Algorithm partitioning between a host processor and a reconfigurable customized coprocessor is used in [5–7], and the use of reconfigurable coarse-grained datapaths is common to many architectures [5, 8–13], and also the use of memory mapping is very common. The innovation introduced by our approach substantially relies on reconfigurable coprocessor core architecture. It is optimized for the execution of the target kind of kernels: four specialized sub-units are used for data buffering, data processing, indexes and loops signal generation, control. According to the specific application domain considered, the coprocessing subsystem is then customized in terms of: (i) number and size of memory elements, (ii) number and functionality of processing units, (iii) source, intermediate and result data width, (iv) auxiliary control elements (such as counters or address generators) number and functionality, etc. The result is a reconfigurable coprocessor optimized for a specific application domain, able to execute kernels with a high level of power/frequency efficiency, avoiding extra cycles due to loops condition evaluation, to array indexes calculation and to data fetching of standard Von Neumann or Very Long Instruction Word (VLIW) computing architectures for large classes of algorithms. Architectural power efficiency of the proposed approach is ensured by: (i) supply voltage reduction made possible by the reduction of the clock frequency required to real time execution of dominant kernels, (ii) average capacitance reduction by the use of small local memories instead of a large centralized memory (locality of reference exploitation), (iii) average switching activity reduction by temporal correlation exploitation ensured by the minimization of hardware resources (time sharing) [5, 14]. To hide to programmers all the architecture/coprocessor specific issues and to allow continuous improvements a set of high-level parametric C functions have been developed.

In this paper we customize our reconfigurable architectural template for multimedia application domain (section 4). The system (standard host processor plus reconfigurable coprocessor plus memory) has been modelled and validated at Register Transfer Level (RTL) using Verilog Hardware Description Language (HDL), and the coprocessor has been synthesized at gate level using commercial tools. The architecture has been tested on a set of representative kernels of multimedia application domain: auto-correlation and cross-correlation filters, FIR filters, IIR filters, forward DCT and inverse DCT. The control of the architecture with a level of automation allowing the programmer to ignore the detailed hardware structure, is a complex task and cannot be fully explained in the pages of this paper, nevertheless a short description of the solution adopted is given.

Simulations show (section 5) the usefulness of the proposed approach, through the comparison of its performances with standard DSP and RISC processing systems, in terms of number of cycles required and processing time.

## 2. Architectural Template Definition

The classic general purpose processing scheme based on the interaction between a standard processor and the main data/program memory through standard data and address buses often cannot comply with real-time constraint in common image/voice processing application. In our architectural template (see Fig. 1) we propose (according to the reconfigurable computing paradigm)

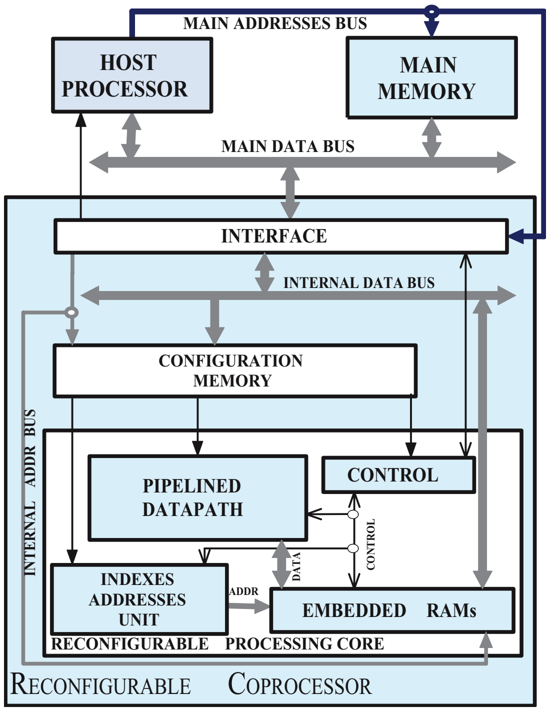

*Figure 1.* Kernel oriented reconfigurable coprocessing system template.

to connect a reconfigurable subsystem to the main bus. Such subsystem is in charge of executing dominant kernels (i.e. the most computational expensive tasks) of the application, the rest of execution remains to the general purpose processor. Reconfigurable paradigm is used to better adapt processing core datapath and control to exploit intrinsic algorithm parallelism. The configuration of the subsystem is determined by a configuration memory, whose content is set by the processor according to the specific task required by the program at the specific moment.

In Fig. 2 the processor-coprocessor iteration is depicted. During standard elaboration (step a) coprocessor is switched off. Each time a dominant kernel execution is required, the host processor activates the coprocessor, and its configuration memory is filled in with specific configuration bits (not all configuration memory locations need to be written) (step b). In step c, kernel input data are copied from main data memory to the proper coprocessor embedded RAMs. During step d, coprocessor performs kernel processing. At the end of step d, kernel processing results are moved to main data memory during step e. All data movement (configuration bits from main memory to coprocessor configuration memory, source data from main memory to embedded coprocessor RAMs and result data from coprocessor embedded RAMs to main data memory) are performed using standard load or store processor instructions. Alternatively a DMA technique can be used to reduce host processor overhead. To simplify the treatment we refer to the former solution.

Reconfigurable Coprocessor is partitioned in three main modules (see Fig. 1): a Reconfigurable Processing Core, a Configuration Memory and an Interface.

## 2.1. Interface

The Interface has two roles: to connect internal data and address buses with main data and address buses, and to synchronize coprocessor and host processor when switching from one to another of the processing steps listed above. The latter functionality can be implemented using a simple Finite State Machine (FSM), while the former using three-state or multiplexer logic, depending on the host system bus implementation. Template specification doesn't imply any specific interface standard, so the designer is free to chose the most suited one (AMBA, OCP, VCI, etc.).

## 2.2. Configuration Memory

Configuration Memory stores the bit stream that defines coprocessor configuration. It is based on an array of registers whose outputs are permanently connected to: (i) control inputs of switches and/or pass transistors determining data routing in reconfigurable processing modules and interconnection networks; (ii) input ports of parametric processing elements (i.e. input which determines the max counting value of a counter).

## 2.3. Reconfigurable Processing Core

Reconfigurable Processing Core is composed of (see Fig. 3): (i) Embedded RAMs, (ii) Pipelined Datapath, (iii) Addresses/Indexes Unit (iv) Control Unit and (v) Interconnection Network.

***2.3.1. Embedded RAMs.*** Unit functionality is kernel data storage. The implementation is based on a cluster of multiple memory elements (RAMs) to store source and result data arrays. RAMs addresses streams come from the Indexes/Addresses Unit, RAMs control signals come from Control Unit, whereas kernel data are exchanged with Pipelined Datapath through internal Pipelined Datapath multi-bus reconfigurable network. Embedded RAMs exchange data with

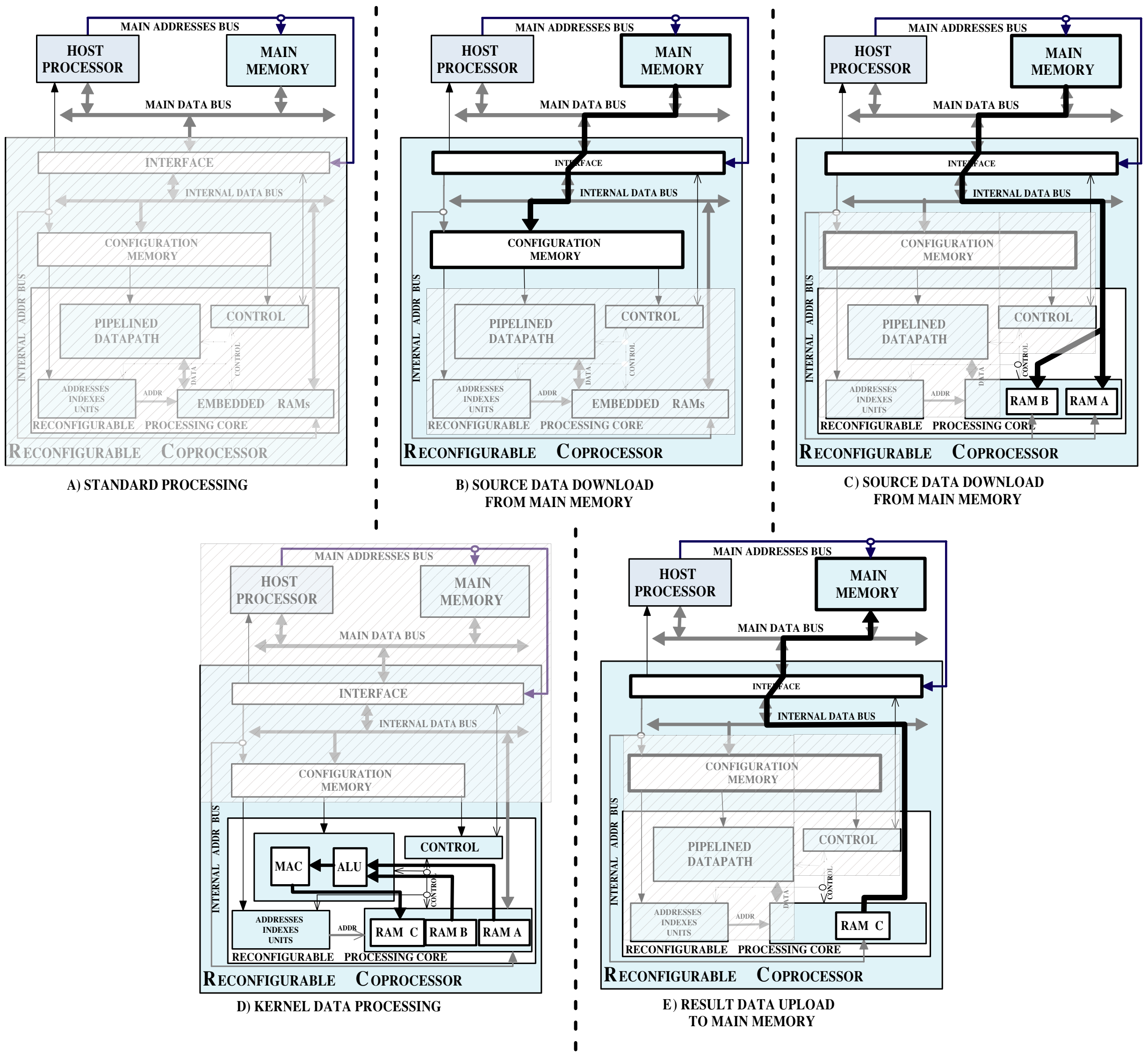

*Figure 2.* Kernel oriented reconfigurable coprocessing system main steps.

Pipelined Datapath (during kernels processing, step d in Fig. 2) or with host system memory through Interface (during kernels data download and upload, steps c and e in Fig. 2).

***2.3.2. Pipelined Datapath.*** Unit functionality is kernel source data processing to generate kernel result data. The implementation is based on a cluster of mixed-grained high level processing units (integer Arithmetic Logic Units (ALUs), integer multipliers (MULs), Field Programmable Gate Arrays (FPGAs), shifters, etc.) and on a set of pipelined registers. Processing elements and pipeline registers can be connected according to specific kernel in execution using a fully reconfigurable multi-bus connection network.

***2.3.3. Addresses/Indexes Unit.*** Unit functionality is twofold: generation of addresses streams for embedded RAMs and generation of loops-end signals used by Control Unit to generate control signals. The implementation is based on a cluster of programmable counters connectible through a reconfigurable network to generate loop indexes signals. The counter dedicated to outermost loop control receives start signal from top level control module. A set of processing units (ALUs, MULs, etc), customized using configuration

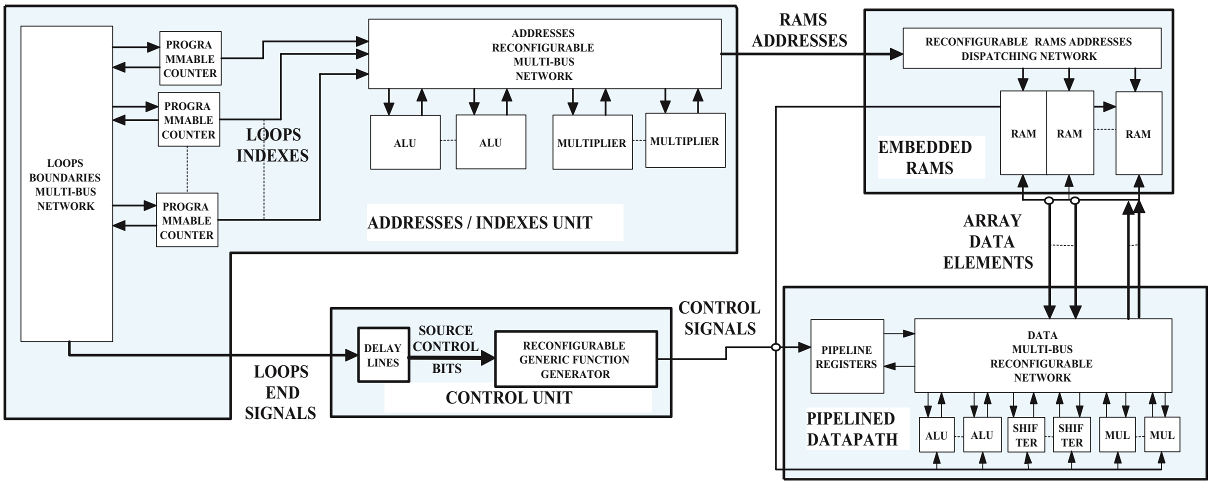

*Figure 3*. Reconfigurable processing core architectural template.

memory and connected through a reconfigurable network, processes indexes streams to generate addresses streams for embedded RAMs.

***2.3.4. Control Unit.*** This unit processes loop-end signals to generate control signals (RAMs and pipeline registers write enables) for Pipelined Datapath and Embedded RAMs. The irregular structure of the task performed requires a full programmable implementation, using a generic reconfigurable combinatorial function generator and delay lines.

***2.3.5. Reconfigurable Interconnection Networks.*** Reconfigurable Interconnection Networks have been implemented adopting an irregular mesh technique [15]. This technique is derived from FPGAs architectures and is based on sets of buses connected to logic or memory modules through Connect Boxes (C-Box). Each segment of a bus can be connected to neighbour segments of the same bus or to segments of other buses through Switch Boxes (S-Box).

Fig. 4 shows the implementation of a 4-port single-bus S-Box and of a 5-port double-bus C-box, based on the use of single switches. Each single switch can be implemented using pass-transistors or a couple of three-state non-inverting buffers. An $m$-port S-box selectively connects $m$ different segments of the same bus using $m$ arrays of $n$-switches, where $n$ is the number of bit-lines of the bus. An S-box connecting $p$

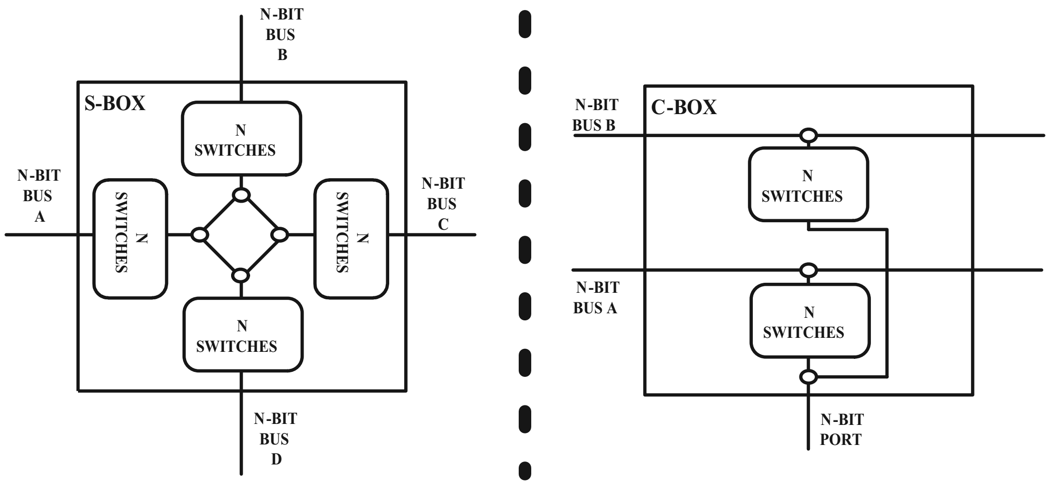

*Figure 4*. S-box and C-box implementation.

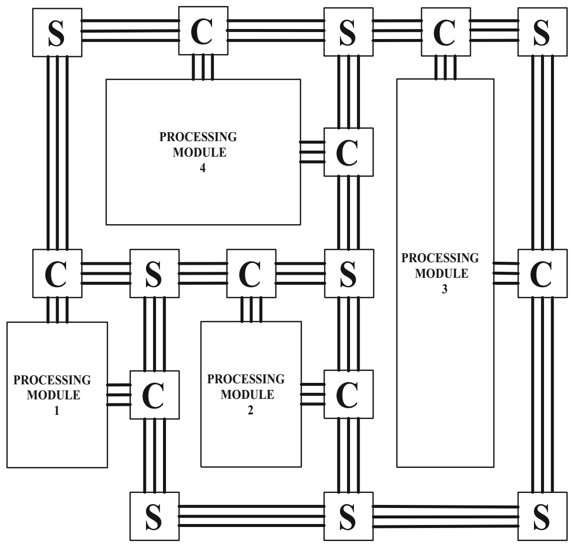

*Figure 5.* Example of irregular-mesh connected network of modules.

buses is implemented simply replying $p$-times this structure. A C-Box selectively connects the port of a processing module to one or more buses using a number of arrays of switches equal to the number of buses.

Fig. 5 shows an example of triple-bus connected, reconfigurable, irregular-mesh. C-Boxes are placed on the sides of modules (each C-box corresponding to one of the ports), whereas S-Boxes are placed on the corners of each module.

## 3. Template Customization and Optimization

The described template is very general. It must be customized for a specific application domain to have an efficient processing system both in terms of performance and used resources/area. The customization process can be subdivided into three main tasks: (i) target application domain kernel identification, (ii) coprocessor customization; (iii) support library development

### *3.1. Dominant Kernel Identification*

A representative set of applications in the domain considered is selected, and dominant kernels are identified performing applications profiling by standard software development tools. As an example we defined a set of applications representative of multimedia domain (the ones identified in section 4).

### *3.2. Coprocessor Customization*

Reconfigurable Coprocessor architectural template customization consists in the quantitative definition of all coprocessor internal modules (how many counters, how many ALUs, how many RAMs, data size, etc). The activity of such hardware resources are scheduled to optimize the execution of the task. The limit on hardware resources leads to the definition of a set of constraints on kernels to be executed (maximum number of loops handled, maximum data size, overall maximum number of data, etc). To optimize the number of resources (taking into account both area/complexity and performances) the architecture is customized for a specific application domain. Each dominant kernel defined in 3.1 have to be mapped on a generic architectural template, composed of an unlimited number of modules. The exact number of ALUs and MULs in the Reconfigurable Datapath and in the Addresses/Indexes Unit, the exact number of Embedded RAMs, data granularity, etc. are defined considering the worst case of the mapping process for the whole set of dominant kernels. This approach ensures hardware resources minimization, while reconfigurability guarantees an adequate level of flexibility. Other features to define at customization time, according to implementation related issues, are interface protocol and Configuration Memory physical implementation.

### *3.3. Support Library Development*

Considering dominant kernels defined in 3.1, a library of parametric C-callable functions are developed. Functions embed all coprocessor-specific issues (configurations, data movement, etc.), so the programmer can ignore the great part of reconfigurable architecture details. The approach is similar to the one used for high performance VLIW DSPs to exploit all processing capabilities of the processor without the need of a deep architecture knowledge. Each function, written in C-code, performs four sub-tasks (corresponding to host processor activities described in Fig. 2): (i) copies coprocessor configuration from main memory to Configuration Memory (optional); (ii) copies kernel source data from main memory to coprocessor Embedded RAMs; (iii) starts coprocessor processing; (iv) copies kernel result data from coprocessor Embedded

RAMs to main memory (after coprocessor acknowledge).

## 4. Customization for Multimedia Application Domain

The target application domain considered is voice and video processing. To represent the application domain the following list of kernels extracted from three GSM codecs (full-rate RPE-LTP [16, 17]; half-rate VSELP [18, 19]; enhanced full-rate ACELP [20, 21]), and MPEG [22–24] standards have been chosen:

i) two kinds of FIR filtering from GSM voice coding.
ii) IIR filtering from GSM voice coding
iii) 8 × 8 inverse DCT from MPEG
iv) 8 × 8 forward DCT from MPEG
v) vectors cross-correlation from GSM voice coding
vi) vectors auto-correlation from GSM voice coding
vii) convolution from GSM voice coding

### *4.1. Reconfigurable Processing Core Architecture*

Reconfigurable Processing Core architecture descends from architectural template described in Fig. 3. All undefined features of the template have been sized basing on the kernels listed above.

***4.1.1. Pipelined Datapath.*** Processing modules are: (i) one 16-bit inputs, 32-bit output MUL, (ii) one 16-bit ALU performing a reduced set of operations (sum, subtract, logical operations), (ii) one 40-bit ALU performing a reduced set of operations (sum, subtract, logical operations), (iii) one 40-bit barrel shifter (performing n-positions shift left, n-positions shift right, n-positions rotate left, n-positions rotate right). Pipeline registers are two 32-bit registers and two 40-bit registers. Reconfigurable Interconnection Network is based on a double bus, two 16-bit and one 8-bit. Bus segments are connected using: 64 16-bit 3-port S-boxes, 16 8-bit 3-port S-boxes, 10 16-bit C-boxes, 6 32-bit C-boxes, 5 40-bit C-boxes. The architecture is depicted in Fig. 6.

***4.1.2. Embedded RAMs.*** Unit block diagram is depicted in Fig. 7. The unit is based on four 16-bit embedded RAMs, two having 256 locations and two having 512 locations. Four address channels coming from Addresses/Indexes Unit are dispatched to the 8 RAM addresses inputs (4 inputs for read addresses and 4 inputs for write addresses) through reconfigurable network. Reconfigurable Interconnection Network is based on a double 9-bit bus. Bus segments are connected using: 16 9-bit 3-port S-boxes, 8 9-bit C-boxes.

***4.1.3. Indexes/Addresses Unit.*** Unit block diagram is showed in Fig. 8. The architecture is based on four 9-bit up-down counters, four 9-bit multipliers, four 9-bit simplified ALUs and two reconfigurable Interconnection Networks. Each counter generates synchronized streams of 9-bit indexes and has configurable start and stop index bounds coming from configuration registers or from another counter of the cascade through Reconfigurable Interconnection Network. ALUs and multipliers generate addresses starting from indexes and from constant data coming from configuration registers. Count-signals Reconfigurable Interconnection

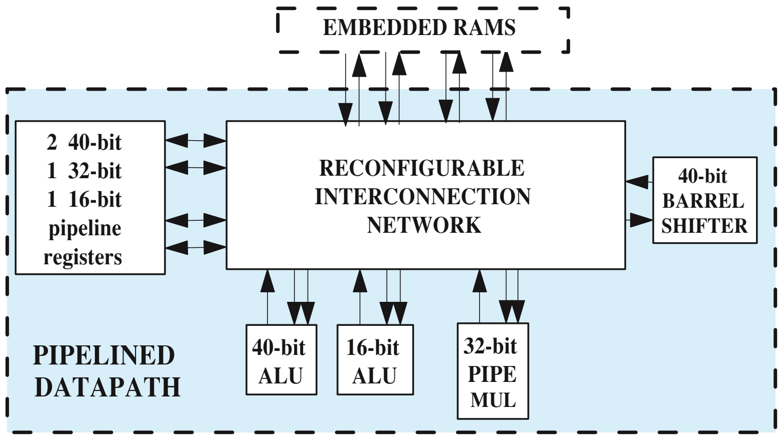

*Figure 6.* Pipelined datapath block diagram.

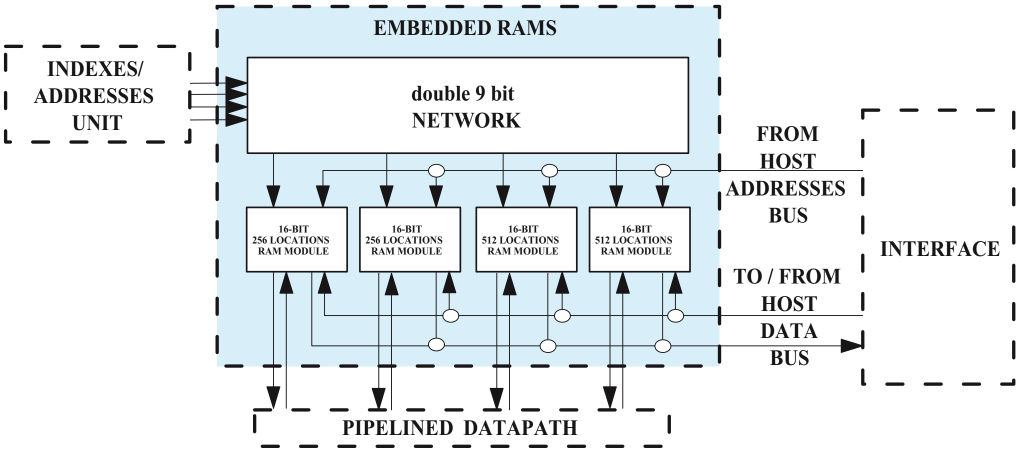

*Figure 7.* Embedded RAMs block diagram.

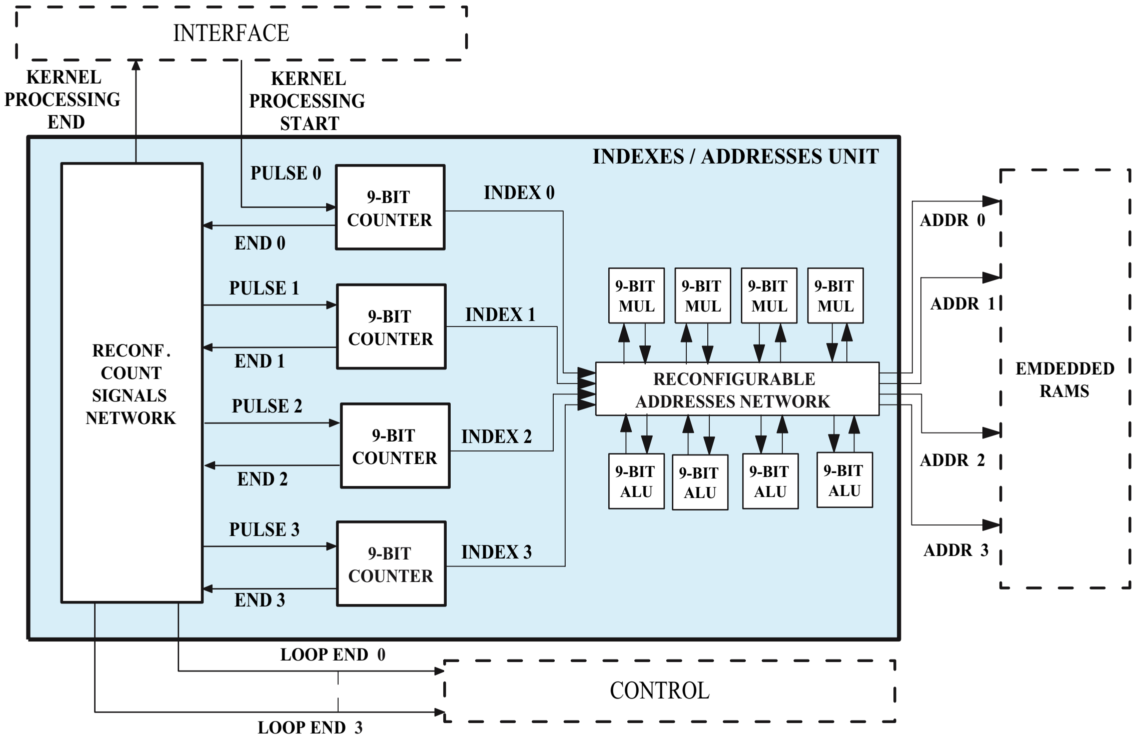

*Figure 8.* Indexes/Addresses unit block diagram.

Network is based on a double 1-bit, bus. Bus segments are connected using: 20 1-bit 3-port S-boxes; 10 1-bit C-boxes. Reconfigurable Addresses Interconnection Network is based on a double, 9-bit bus. Bus segments are connected using: 36 9-bit 3-port S-boxes; 20 9-bit C-boxes.

***4.1.4. Control Unit.*** Control Unit processes 4 loops-end signals coming from cascaded counters of Indexes/Addresses Unit to generate 8 write enable signals for the 4 pipeline registers of the Pipelined Datapath and for the 4 RAMs. Each signal could be delayed up to two cycles using a dedicated delay line. For each

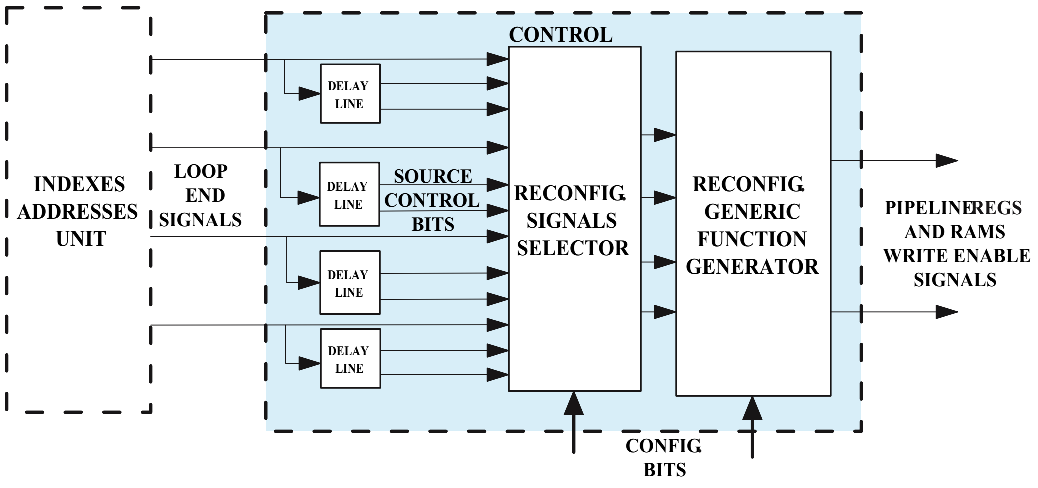

*Figure 9*. Control unit block diagram.

kernel the write enable signals needed are a function of 4 of the 24 resulting signals (4 in input and 8 delayed). Pre-selection module and generic combinatorial function generator are configured depending on the kernel. Generic combinatorial function generator is based on a programmable Look-Up Table (LUT) implemented using an 8-bit – 16 locations RAM. Reconfigurable Signals Selector is implemented using four 24 × 1 1-bit multiplexers. Block diagram is shown in Fig. 9.

## 4.2. *Interface*

Interface protocol is not strictly related to the application domain chosen. Interface has been designed to support AMBA 32-bit bus interfacing, so its implementation is similar to a standard AMBA slave interface augmented with some features needed to synchronize host processor and core processing unit.

## 4.3. *Kernel Processing Example*

To explain Reconfigurable Processing Core behaviour we consider, as an example, the execution of a cross-correlation between two vectors of 16-bit data, with 40-bit accumulation. Kernel code is presented in Fig. 10.

Execution flow is based on the three inner loops, controlled by i, j and k indexes. Inside these loops body, the

```
signed_32  ACC;

signed_16  A[160], B[287], C[512];

short i, j, k;

for (i=0 ; i<4 ; i=i+1)

 for (j=0 ; j<128 ; j++)

 {

   for (k = 0; k < 40; k++)

     ACC = ACC+A[40*i+k]*B[40*i+j+k];

   C[128*i+j] = extract_most_significant(ACC);

  }
```

*Figure 10*. Example of a kernel (C code).

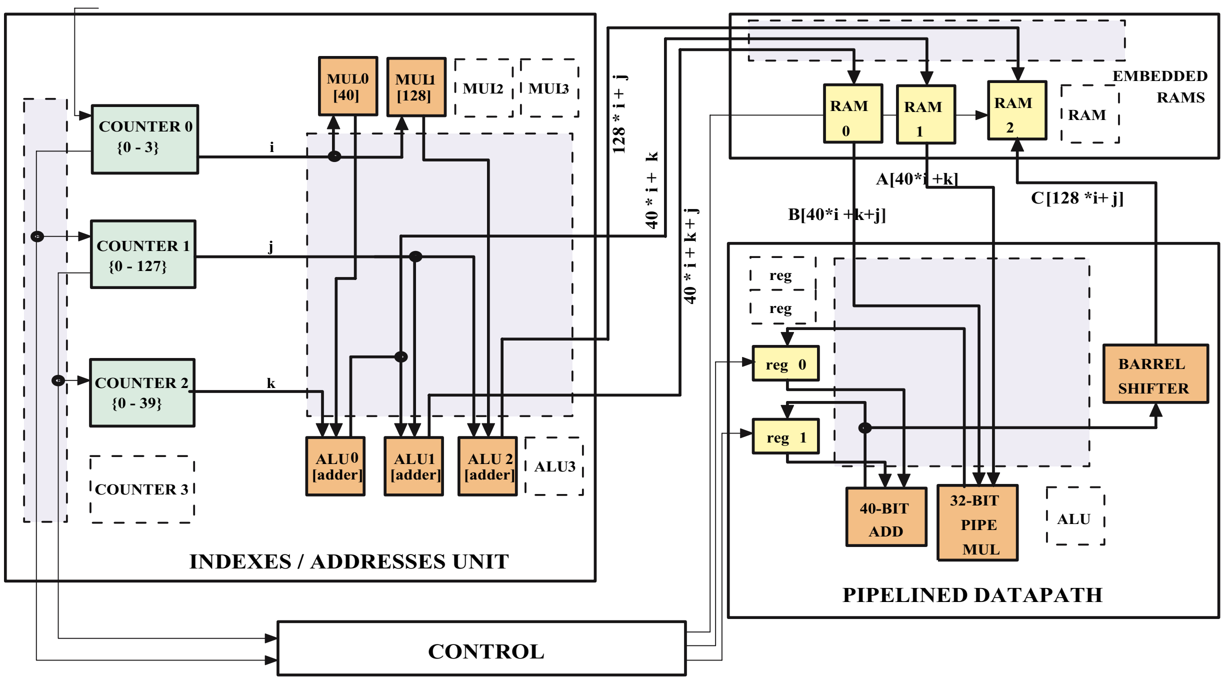

*Figure 11.* Reconfigurable processing core configured for cross-correlation.

elements of vectors A and B are multiplied and accumulated into vector C. Reconfigurable Processing Core configured for the execution of this kernel is shown in Fig. 11.

This dedicated architecture implements four pipeline stages (with throughput of 1 clock cycle): (i) operands and results addresses computation, (ii) operands fetching from embedded RAMs 0 and 1 (vectors A and B), (iii) 16-bit operands one stage pipelined multiplication and 32-bit result storage in register 0, (iv) 32-bit sum with 40-bit partial result sum and storage in register 1. At the end of each internal loop the 16 most significant bits of the final accumulated result are saved in RAM 2 (vector C) using the barrel shifter, which was properly configured depending on the accumulation maximum value. Configured cascaded counters generate i, j and k indexes with the correct synchronization and boundaries. Indexes are sent in input to the Addresses Unit where the network of configured constant adders and constant multipliers manipulate them to generate three synchronized streams of addresses for the three RAMs used. Output data ports of the two RAMs containing source vectors are connected to multiplier inputs, whereas accumulation register (reg 1) output is connected the barrel shifter which is connected to the input port of RAM used to store results vector.

The number of cycles required for cross-correlation example kernel processing on the Reconfigurable Coprocessing Subsystem is about 20,500. The number of cycles required for other, not mandatory tasks (data I/O and configuration), is about 5,000. The same piece of algorithm is executed by ARM9 processor in about 400,000 cycles, by TMS320C5510 TI DSP (no assembly optimisations were used) in about 105,000 cycles.

The substantial reduction of clock cycles (about 94% for RISC processor and about 76% for DSP) is due to the fact that the architecture depicted in Fig. 11 (after configuration) is optimized as much as a dedicated architecture. The implementation exploits the inner parallel/pipelined structure of nested loops algorithms. Processors (RISC or DSP) spend cycles executing: (i) loop conditions evaluation (ii) addresses calculation, (iv) data fetching and processing. Reconfigurable coprocessor cuts off the most part of this cycles: (i) loop conditions evaluations are performed in zero cycles, due to counters connected to control logic; (ii) addresses calculation is also performed in zero cycles due to counters, adders and multipliers of Addresses/Indexes Unit; (iii) data fetching and processing is performed using embedded RAMs and reconfigurable datapath connected in pipeline, with no wait states, leading to a further cycles saving.

## 5. Modeling and Evaluation of Reconfigurable Coprocessor for Multimedia

To evaluate our approach we have performed the following steps: (i) Reconfigurable Coprocessor for multimedia has been modelled using a standard HDL language and an RTL-behavioural description style; (ii) coprocessor has been validated coupling it with a standard general purpose processing system (host-processor plus memory) also modelled in HDL; standard development tools have been used to program the host-processor and HDL simulation to execute effective processing, (iii) Reconfigurable Coprocessor was implemented at the gate-level (embedded RAMs excluded) using standard synthesis tools (Reconfigurable Interconnection Networks switches were implemented using three-state buffers, Configuration Memory were implemented using an array of 32-bit registers, all C-Boxes and S-Boxes considered in this paper are single bus), (iv) a set of parametric functions implementing multimedia dominant kernels were coded using ansi-C language; (v) cycle level performances of the system have been compared to standard processing platforms (a RISC processor and a fixed point DSP).

### *5.1. Modeling and Validation*

The whole system has been modelled using Verilog HDL. Coprocessor functionality has been validated through Verilog simulation, performed using Synopsys VCS simulation environment [25]. A graphical description of simulation framework is depicted in Fig. 12.

To model the host-processor, a Verilog cycle level compliant version of ARM9TM processor [26] has been used. Data and instruction memories have been also modelled using Verilog behavioral description. Algorithms code has been compiled using ARM SDT [27] and the output binary code has been converted in a Verilog readable form and loaded into Instruction Memory.

### *5.2. Evaluation*

To evaluate coprocessor basic figures of merit two analysis have been performed: (i) gate level implementation, (ii) dominant kernel execution comparison.

***5.2.1. Gate-level Implementation Results.*** The reconfigurable coprocessor has been synthesized using Synopsys Design Compiler [25] on CMOS 0.18 $\mu$m technology library (UMC18$\mu$1P6M) [28], with the only exception of embedded RAMs. Transistor count of synthesizable part of the coprocessor were estimated using synthesis reports, while for embedded RAMs we considered 6 transistors for each memory cell (bit) and a 1.2 coefficient to consider auxiliary logic (sense amplifiers, decoders, etc.) [29]. Computational

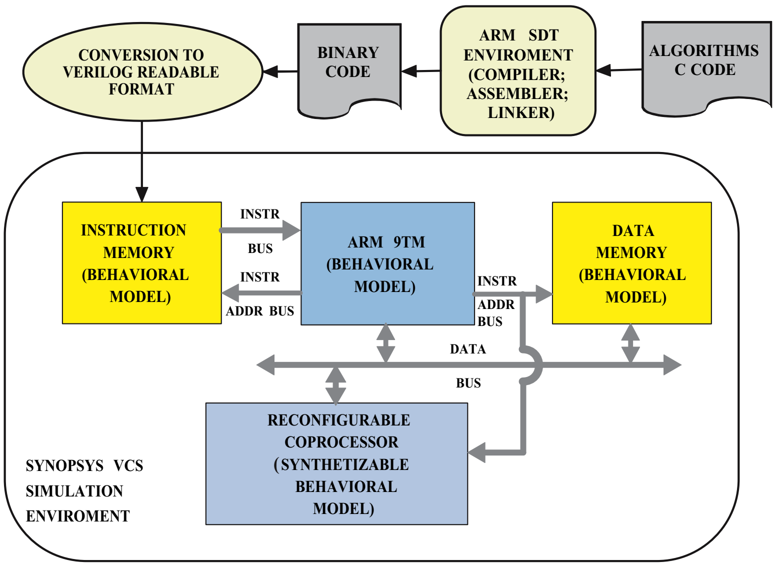
CONVERSION TO VERILOG READABLE FORMAT
BINARY CODE
ARM SDT ENVIROMENT (COMPILER; ASSEMBLER; LINKER)
ALGORITHMS C CODE
INSTRUCTION MEMORY (BEHAVIORAL MODEL)
INSTR BUS
INSTR ADDR BUS
ARM 9TM (BEHAVIORAL MODEL)
INSTR ADDR BUS
DATA MEMORY (BEHAVIORAL MODEL)
DATA BUS
RECONFIGURABLE COPROCESSOR (SYNTHETIZABLE BEHAVIORAL MODEL)
SYNOPSYS VCS SIMULATION ENVIROMENT

*Figure 12.* Coprocessor verification flow.

and control elements of Reconfigurable Processing Core (ALUs, MULs, Counters, control delay lines, etc.) require 10.53 K equivalent gates (42.12 K transistors); Reconfigurable Interconnection Networks require 23.62 K equivalent gates, (94.48 K transistors); embedded RAMs require 24.5 K bits (176.9 K transistors); Configuration Memory was implemented using 29 32-bit registers (928 bits), requiring 5.92 K equivalent gates, (23.68 K transistors); Interface requires 3.24 K equivalent gates (12.96 K transistors). The overall gate level implementation requires 40.07 K equivalent gates (160.28 K transistors) and 24.5 K bits of memory (176.4 K transistors) for a total amount of 335.68 K transistors. It is worth to note that a large percentage of standard-cells (more than 58%) is dedicated to Reconfigurable Interconnection Networks, i.e. to standard-cell implemented three-state buffers used to realize switches in C-boxes and S-boxes. The incidence of this kind of feature to overall area can be substantially reduced using custom pass-transistors instead of standard-cell three-state buffers to implement switches (for a single switch, area reduction is up to 70%). The intrinsic regularity of the architecture and the use of pipelined computational elements in datapath and control allow a very high computational speed: estimated maximum allowable frequency is 460 MHz.

***5.2.2. Dominant Kernels Compared Execution Results.*** Table 1 shows the comparison of cycles and processing time required for specific tasks on: (i) RCS (reconfigurable coprocessor plus host processor), (ii) a 32-bit RISC processor (ARM9TM), (iii) a 16-bit fixed-point Digital Signal Processor (Texas Instruments TMS320C5510) [30]. Since different technologies lead to difference operative frequencies the most significant comparison is in terms of clock cycles because it can be considered technology independent

Results have been obtained using respectively: the simulation framework depicted in Fig. 12, ARM standard development tools [27], simulator and profiler, Texas Instruments Code Composer Studio Development Tools (TMDSCCS5000-1) [31], simulator and profiler.

The profiling has been executed on optimized C source code from [19, 21, 24] and from TI processors' benchmarks (for MPEG FDCT and IDCT). For both processors we have set the highest level of optimization in the C compiler. The same code has been analyzed to create the configuration for our reconfigurable coprocessor.

Table 1 is organized in eight columns. First column shows kernel description, columns 2 and 3 show the number of clock cycles and the processing time required for kernel execution respectively on ARM9TM RISC (200MHz) processor and on TMS320C5510 DSP (200MHz). Column 4 shows the overall number of clock cycles and processing time required for kernel execution on Reconfigurable Coprocessing System. The number of clock cycles is the sum of 3 contributes: cycles spent for configuration step (worst case), for data I/O and for processing. These quantities are showed in the fifth column in disaggregate form. Column 6 and 7 shows the percentage reduction in clock cycles and in processing time obtained comparing reconfigurable coprocessing system with respect to ARM9TM and TM320C5510 processors.

To better evaluate the potentiality of our approach we defined two additional tables. Table 2 gives a quantitative description of kernel operators. It presents the types of operators involved in the computation of each kernel, and for each operator, how many times it is used. Operators are classified in data processing operators and vectors/matrix addresses processing operators. Table 3 shows coprocessor resources utilization during each kernel processing. Resources are classified in datapath resources and Indexes/Addresses Unit (IAU) resources. Since the IAU resources are combinatorial (with the only exception of counters) the percentage of utilization corresponds to the percentage of allocated units for the generation of addresses and control signals for every kernel.

***5.2.3. Comparison with Previous Works.*** Many reconfigurable architectures have been presented in scientific literature to address specific issues related to hardware accelerators. Trying to compare our approach with these previous works, it is worth to consider both the philosophy and the architectural structure. From a structural point of view, reconfigurable architectures can be classified according to their granularity and kind of interconnection network. Since the interconnection network is a secondary issue, primarily we can distinguish between coarse-grained, fine-grained and mixed-grained architectures.

Fine-grained reconfigurable systems are closest to FPGAs: they exploit the fine granularity to build up datapaths with different data widths in the most flexible way. Examples of such architectures are GARP [7] and CHESS [32]. This kind of fine-grained approach involves however some pitfalls. Reconfigurable

*Table 1*. Comparison result.

| | | | Rec. Coproc. | | | |
|---|---|---|---|---|---|---|
| Kernel description | ARM9TM cycles time [$\mu$s] | C5510 cycles time [$\mu$s] | Cycles time [$\mu$s] | Configuration data I/O processing — | Red. vs RISC cycles % time % | Red. vs DSP cycles% time % |
| IIR Filter I (VSELP) | 7370 | 2463 | 1340 | 500 / 450 / 390 | 82 | 46 |
| | 36.85 | 12.31 | 2.91 | — | 92 | 76 |
| IIR Filter II (VSELP) | 7559 | 2526 | 1350 | 500 / 450 / 400 | 82 | 47 |
| | 37.79 | 12.63 | 2.93 | — | 92 | 77 |
| FIR Filter (VSELP) | 7200 | 2467 | 1350 | 500 / 450 / 400 | 81 | 45 |
| | 36 | 12.33 | 2.93 | — | 92 | 76 |
| Auto Corr. I (VSELP) | 26743 | 7720 | 5140 | 500 / 3200 / 1440 | 81 | 33 |
| | 133.71 | 38.6 | 11.17 | — | 92 | 71 |
| Auto Corr. II (ACELP) | 1502879 | 332567 | 72900 | 500 / 8400 / 64000 | 95 | 78 |
| | 7514.39 | 1662.63 | 158.47 | — | 98 | 90 |
| Auto Corr. III (ACELP) | 46996 | 13061 | 5330 | 500 / 2245 / 2585 | 89 | 59 |
| | 234.98 | 65.3 | 11.58 | — | 95 | 82 |
| Cross Corr. I (VSELP) | 396307 | 105285 | 25380 | 500 / 4400 / 20480 | 94 | 76 |
| | 1981.53 | 526.42 | 55.17 | — | 97 | 90 |
| Cross Corr.II (ACELP) | 15233 | 4544 | 1920 | 500 / 600 / 820 | 87 | 58 |
| | 76.16 | 22.72 | 4.17 | — | 95 | 82 |
| Synth Filter I (ACELP) | 7584 | 2407 | 1350 | 500 / 450 / 400 | 82 | 44 |
| | 37.92 | 12.03 | 2.93 | — | 92 | 76 |
| Synth Filter II (RPE-LPT) | 64464 | 43576 | 2600 | 500 / 1140 / 960 | 96 | 94 |
| | 322.32 | 217.88 | 5.65 | — | 98 | 97 |
| Convolution (ACELP) | 15162 | 4504 | 1920 | 500 / 600 / 820 | 87 | 57 |
| | 75.81 | 22.52 | 4.17 | — | 94 | 81 |
| Rest Eval (ACELP) | 11860 | 2570 | 1590 | 500 / 650 / 440 | 87 | 38 |
| | 59.3 | 12.85 | 3.45 | — | 94 | 73 |
| Search (RPE-LTP) | 60919 | 17586 | 4170 | 500 / 430 / 3240 | 93 | 76 |
| | 304.59 | 87.93 | 9.06 | — | 97 | 90 |
| FDCT (MPEG2) | 6716 | 1498 | 1032 | 500 / 128 / 1024 | 85 | 31 |
| | 33.58 | 7.49 | 2.24 | — | 93 | 70 |
| IDCT (MPEG2) | 16136 | 3159 | 1032 | 500 / 128 / 1024 | 94 | 67 |
| | 80.68 | 15.79 | 2.24 | — | 97 | 86 |

*Table 2.* Number and kind of operators applied to data and indexes for kernel execution.

| | Data operators | | | Addresses operators | |
|---|---|---|---|---|---|
| Kernel description | sums/subs | Muls | Shifts | sums/subs | Muls |
| IIR Filter I (VSELP) | 390 | 390 | 39 | 429 | 0 |
| IIR Filter II (VSELP) | 400 | 400 | 40 | 440 | 0 |
| FIR Filter (VSELP) | 400 | 400 | 40 | 440 | 0 |
| Auto Corr. I (VSELP) | 1440 | 1440 | 1440 | 2952 | 2916 |
| Auto Corr. II (ACELP) | 0 | 64000 | 0 | 320000 | 1600 |
| Auto Corr. III (ACELP) | 0 | 2585 | 45 | 2585 | 0 |
| Cross Corr. I (VSELP) | 20480 | 20480 | 20480 | 61952 | 20992 |
| Cross Corr. II (ACELP) | 820 | 820 | 0 | 820 | 0 |
| Synth. Filter I (ACELP) | 440 | 400 | 40 | 400 | 0 |
| Synth. Filter II (RPE-LPT) | 1920 | 960 | 960 | 120 | 0 |
| Convolution (ACELP) | 0 | 820 | 40 | 820 | 0 |
| Rest Eval (ACELP) | 440 | 440 | 40 | 880 | 0 |
| Search (RPE-LTP) | 3320 | 3240 | 3240 | 6480 | 0 |
| FDCT (MPEG2) | 576 | 512 | 0 | 1152 | 1152 |
| IDCT (MPEG2) | 576 | 512 | 0 | 1152 | 1152 |

*Table 3.* Hardware percentage utilization.

| | Datapath resources | | | IAU resources | | |
|---|---|---|---|---|---|---|
| Kernel description | ALUs | Mul | Shifter | Counters | ALUs | Mul |
| IIR Filter I (VSELP) | 50% | 100% | 100% | 50% | 50% | 0% |
| IIR Filter II (VSELP) | 50% | 100% | 100% | 50% | 50% | 0% |
| FIR Filter (VSELP) | 50% | 100% | 100% | 50% | 50% | 0% |
| Auto Corr. I (VSELP) | 50% | 100% | 100% | 75% | 50% | 75% |
| Auto Corr. II (ACELP) | 0% | 100% | 100% | 75% | 100% | 25% |
| Auto Corr. III (ACELP) | 0% | 100% | 2% | 50% | 25% | 0% |
| Cross Corr. I (VSELP) | 50% | 100% | 100% | 75% | 100% | 50% |
| Cross Corr. II (ACELP) | 50% | 100% | 0% | 50% | 25% | 0% |
| Synth. Filter I (ACELP) | 60% | 100% | 100% | 50% | 25% | 0% |
| Synth. Filter II (RPE-LPT) | 100% | 100% | 100% | 50% | 25% | 0% |
| Convolution (ACELP) | 0% | 100% | 20% | 50% | 25% | 0% |
| Rest Eval (ACELP) | 50% | 100% | 10% | 50% | 50% | 0% |
| Search (RPE-LTP) | 54% | 100% | 100% | 50% | 50% | 0% |
| FDCT (MPEG2) | 62% | 50% | 0% | 100% | 100% | 100% |
| IDCT (MPEG2) | 62% | 50% | 0% | 100% | 100% | 100% |

platforms have been conceived to overcome the slowness in configuration of traditional FPGAs due to the high amount of configurable elements and serial configuration interfaces, and the relatively low performances of such substrates in terms of computational speed and power consumption. As a matter of fact, FPGA represent the most flexible substrate for the implementation of any design but pay this flexibility with limited performances. If a reconfigurable platform is composed of reconfigurable elements with fine granularity (2-4 bits) all the limits of FPGAs arise again: a huge interconnection network that occupies the greatest part of the chip area, an intrinsic slowness in terms of clock frequency and a poor exploitation of the properties of multimedia algorithms to deal with large data words.

To overcome these limits, coarse-grained architectures like PipeRench [8], MorphoSys [9], DReAM [10], KressArray [11], MATRIX [13] adopt reconfigurable units with a larger data word. In this manner the interconnection network can be kept simpler and also the speed of a large reconfigurable unit is higher than that of an equivalent block that could descend from the interconnection of finer units in a fine-grained architecture. At the same time such architectures are not flexible enough to deal with variable data width. Other examples of coarse-grained architectures but where every functional unit is a small processor are RAW [12] and REMARC [6].

Our approach belongs to the category of mixed-grained architectures since the granularity of the elements is quite different into the array, allowing the presence of heterogeneous elements like FPGA blocks, barrel shifters, ALUs, multipliers with different data width and so on. A similar approach has been exploited in the Pleiades [3] template, even if the Maia [5] chip that descends from the same template doesn't contain multigranular elements.

It should be noted that this is only a raw classification because some of the architectures cited above, like [5, 8], and the one presented in this paper, have a granularity specifiable at synthesis time and take all the advantages of the other two approaches. Another key aspect briefly introduced above is the homogeneity of the processing elements. Undifferentiated cells are very common in this type of architectures [6–13, 32] because the most general approach is to create a fabric of identical cells able to perform any arbitrary task. This choice limits the ability to create domain-specific instances of the architecture able to obtain the highest performances for that particular application domain. Our heterogeneous model shows better performance with respect to bit-byte-word level architectures based on absolutely identical "tiles".

The two works that are closer to our reconfigurable coprocessors are [5] and [10]. In [5] a reconfigurable coprocessor based on the Pleiades templates is presented. The Maia architecture is a reconfigurable coprocessor for CELP speech coding, and it aims to speed-up computational kernels by means of the application-specific datapath, reducing also the power consumption. The greatest difference compared to our work is that Maia is a self-timed asynchronous implementation with all circuit blocks (including the top level design of the chip) designed using a full-custom design methodology. This is because the self-timed data-driven approach is not suited for automated synthesis due to its intrinsically difficult timing verification. All this characteristics make the instantiation of a coprocessor from the Pleiades template not very straightforward compared to standard cell synchronous design approaches exploited in our work. Furthermore the data-driven approach requires an handshake protocol that represents an overhead in communication time. Finally, the control part of Maia is based on two Address Generation Processors (AGPs) that works on sequential codes to perform the addresses generation, whereas our control part is less complex and doesn't require any micro-code routine but only some parameters for the cascaded counters that are the basis of the address generation units.

The DReAM architecture [10] is based on an array of undifferentiated functional units. These units have fixed data width of 8 bits, that is quite narrow for today's applications. Every unit also presents a look-up table based multiplier, hence its scalability is reduced by the look-up tables sizes limits. In our approach multipliers are standard pipeline structures which can be easily optimized by means of synthesis tools. The adoption of undifferentiated cells allows a greater flexibility of the system compared to our approach. Nevertheless, this advantage is overcome by the higher hardware redundancy that is a slightly inefficient solution with respect to hardware usage considering that the authors stress the domain specific approach. In our reconfigurable architecture hardware redundancies have been limited dimensioning the structure considering all the main kernel types identifiable in the considered domain. Finally, in DReAM the adoption of a regular mesh with hierarchical interconnection structure avoids specific optimizations so that the advantages in terms of generality aren't balanced by the efficiency of the network. It should be noted that DReAM, like also the largest part of the mentioned previous works, is not a customizable template but a specific architecture. From this point of view all the mentioned drawbacks highlighted for this last previous work are common to the other similar not customizable reconfigurable architectures.

## 6. Discussion and Conclusions

A new reconfigurable architectural template for execution acceleration of a wide class of dominant kernels is proposed. Dominant kernels considered are based on repetitive execution of a reduced set of operations on multidimensional arrays (vectors or matrix). Our approach uses reconfigurable computing paradigm to accelerate execution: (i) cycles required for loops conditions calculation and evaluation were eliminated by the use of a reconfigurable dedicated processing-scheduling units; (ii) cycles required for array indexes calculation and array data fetching were eliminated by the use of dedicated reconfigurable processing units and reconfigurable Embedded RAM clusters; (iii) the number of cycles required for array data processing was minimized by the use of high parallel-pipelined reconfigurable datapath.

The system is based on a reconfigurable coprocessor coupled to a host processor. Being based on a classic memory mapped communication scheme and on a simple interrupt-based signalling technique, coprocessor is intrinsically conceived to be easily coupled with each kind of processor. More coprocessors can be also easily clustered in the same system to allow thread level parallelism. System functionality can be sized to the chosen application domain simply tuning the number of resources (i.e. to handle up to N nested loops, N counters are needed in Loops Unit, to process floating point data is sufficient to add one or more floating point units to replace or support ALUs and multipliers in the Pipelined Datapath, etc.), without redesigning the system for each new application domain. The benefits of our approach increase proportionally to the number of iterations of the loops, because the overhead of cycles needed for system reconfiguration is fixed, whereas the number of cycles saved for (i), (ii) and (iii) increases with the iterations.

Many reconfigurable architectures are available in scientific literature. Trying to relate our architecture to the state of the art, they can be mainly classified according to their granularity. Primarily we can distinguish between coarse-grained, fine-grained and mixed-grained architectures. Fine-grained ones are closest to FPGAs: they exploit the fine granularity to build up datapaths with different data widths in the most flexible way [7, 32]. Using such a fine-grained approach may cause on one hand an excessive overhead in reconfiguration and on the other hand does not exploit the properties of multimedia algorithms to deal with large data words. Coarse-grained architectures [5, 8–11, 13] are often not enough flexible to deal with variable data width. This is only a raw classification because some of the architectures cited above, like the one presented in this paper, have a granularity specifiable at synthesis time [5, 8] and take all the advantages of the other two approaches. Undifferentiated cells are very common in this type of architectures [6–13, 32] because the most general approach is to create a fabric of identical cells able to perform any arbitrary task. This choice limits the ability to create domain-specific instances of the architecture able to obtain the highest performances for that particular application domain. Our model shows a mixed-grained architecture that includes coarse-grained modules like multipliers, adders and shifters, and fine-grained FPGA modules obtaining better performance with respect to bit-byte-word level architecture based on absolutely identical "tiles" (i.e. [12]). Moreover, with respect to similar works (i.e. [5]) our system is a standard synchronous architecture synthesized from a Verilog RTL code, without any hand-made optimization and no microprocessor unit for data address generation and internal control is required.

To validate our approach an instance of the proposed reconfigurable coprocessing template customized for multimedia application domain has been modelled using Verilog HDL, synthesized at the gate level and tested on dominant kernel extracted from standard multimedia algorithms. A library of parametric C functions implementing multimedia dominant kernels allows high level programmers to ignore the complex inner structure of the architecture. Results show a reduction of the number of cycles for dominant kernels processing from 81% (Auto Corr I extracted from GSM VSELP) up to 96% (Synth. Filter II extracted from GSM RPE-LTP) compared to a standard 32-bit RISC processor (ARM9TM), whereas a reduction from 31% to 94% compared to top class fixed point DSP (TI TMS320C5510), for the same kernels. The software profiling, architectural and hardware design and simulation methodology has been presented as a general approach easily portable to other application domains.

## Acknowledgment

The authors wish to thank Oreste Villa and Romolo Camplani for the development of part of the code, Massimo Barbaro for useful suggestions and support.

## References

1. R. Tessier, W. Burleson, "Reconfigurable Computing for Digital Signal Processing: A Survey," *Journal of VLSI Signal Processing*, no. 28, 2001, pp. 7–27.

2. K. Compton, S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM Computing Surveys*, vol. 34, No. 2, June 2002, pp. 171–210.
3. J. M. Rabaey, "Reconfigurable Computing: The Solution to Low Power Programmable DSP," *Proceedings 1997 ICASSP Conference*, Munich, April 1997.
4. R. Sueyoshi, M. Iida, "Configurable and Reconfigurable Computing for Digital Signal Processing," *IEICE Transactions on Fundamentals*, vol. E85-A, No. 3, March 2002, pp. 591–599.
5. A. Abnous, "Low-Power Domain-Specific Processors for Digital Signal Processing," *PhD thesis*, Dept. of EECS, UC Berkeley, CA, USA, 2001.
6. T. Miyamori, K. Olukotun, "REMARC: Reconfigurable Multimedia Array Coprocessor," *IEICE Trans. on Information and Systems*, vol. E82-D, No. 2, February 1999, pp. 389–397.
7. J. H. Hauser, "Augmenting a Microprocessor with Reconfigurable Hardware," *PhD thesis*, Dept. of EECS, UC Berkeley, CA, USA, 2000.
8. S.C. Goldstein, H. Schmith, M. Moe, M.Budiu, S. Cadambi, R.R. Taylor, R. Laufer, "PipeRench: A Coprocessor for Streaming Multimedia Acceleration," *Proc. of The 26th Annual Internation Symposium on Computer Architecture*, Atlanta, Georgia, May 1999.
9. H. Sing, M. Lee, F.J. Kurday, N. Bagherzadeh, E.M. Chaves Filho, "MorphoSys: an Integrated Configurable System for Data-Parallel and Computation Intensive Applications," *IEEE Transactions on Computer*, vol. 49, no. 5, May 2000, pp. 465–481.
10. J. Becker, M. Glesner, A. Alsolaim, J. Starzyk, "Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communication Systems," *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00)*, Napa, CA, USA, April 17–19, 2000.
11. R. Hartenstein, R. Kress, "A Datapath Synthesis System for the Reconfigurable Datapath Architecture," *Asia and South Pacific Design Automation Conference, ASP-DAC '95*, August 1995.
12. E. Waingold, M. Taylor, D. Srikrishna, V. Sakar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, A. Agarwal, "Baring It All to Sofware: Raw Machines," *IEEE Computer*, September 1997, pp. 86–93.
13. E. Mirsky, A. DeHon, "MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources," *FCCM '96 - IEEE Symposium on FPGAs for Custom Computing Machines*, Napa, CA, April 1996.
14. K. Murakami, H. Magoshi, "Trends in High-Performance, Low-Power Processor Architectures," *IEICE Transactions on Electronics*, vol. E84-C, no. 2, February 2001, pp. 131–138.
15. A. Varma, C. S. Raghavendra, "Interconnection Networks for Multiprocessors and Multicomputers: Theory and Practice," *IEEE Computer Society Press*, 1994.
16. E.T.S.I., "Full Rate speech transcoding—Digital cellular telecommunication system (Phase 2+)—GSM 06.10 version 8.0.2," http://www.etsi.org, 1999.
17. J. Degener, C. Bormann, "Source code for the GSM Full Rate speech codec," http://kbs.cs.tu-berlin.de/∼jutta/toast.html, Technische Universitaet Berlin.
18. E.T.S.I., "Half rate speech transcoding—Digital cellular telecommunication system—GSM 06.20 version 8.0.1," http://www.etsi.org, 1999.
19. E.T.S.I., "ANSI-C code for the GSM half rate speech codec – Digital cellular telecommunication system—GSM 06.06 version 8.0.1," http://www.etsi.org, 1999.
20. E.T.S.I., "Enhanced Full Rate (EFR) speech transcoding—Digital cellular telecommunication system (Phase 2+)—GSM 06.60 version 8.0.1," http://www.etsi.org, 1999.
21. E.T.S.I., "ANSI-C code for the GSM Enhanced Full Rate speech codec—Digital cellular telecommunication system—GSM0 6.53 version 8.0.1," http://www.etsi.org, 1999.
22. ISO/IEC JTC1/SC29/WG11, "ISO/IEC DIS 13818-1 Information technology—-Generic coding of moving pictures and associated audio information: Systems," July 1996.
23. ISO/IEC JTC1/SC29/WG11, "ISO/IEC DIS 13818-2 Information technology—Generic coding of moving pictures and associated audio information: Video," July 1996.
24. C. Lee, M. Potkonjak, W. H. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems," *Department of Computer Science and Electrical Engineering*, University of California at Los Angeles, July 1996.
25. Synopsys Inc., http://www.synopsys.com.
26. ARM Limited, http://www.arm.com.
27. ARM Corporation, "ARM Software Development Toolkit: Version 2.50," Reference Guide, ARM DUI 0041C, Nov. 1998.
28. UMC, http://www.umc.com.
29. J.M. Rabaey, A. Chandrakasan and B. Nicolic, "Digital Integrated circuits—a Design Perspective, second edition," *Prentice Hall Electronics and VLSI Series*, 2003.
30. Texas Instruments Inc., http://www.ti.com.
31. TMS320C55x Instruction Set Simulator Technical Overview, Texas Instruments Inc., SPRU599A, July 2002.
32. A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, B. Hutchings, "A Reconfigurable Arithmetic Array for Multimedia Applications," *Proceedings 7th ACM International Symposium on Field-Programmable Gate Arrays, Monterey (CA)*, February 1999, pp. 135–143.

**Salvatore M. Carta** (1997 Electronic Eng. Master. 2002 Electronics and Computer Science PhD) joined the Department of Electrical and Electronics Engineering of the University of Cagliari, Italy in 1998 as PhD student. From 2005 he has been assistant professor in Department of Mathematics and Computer Science of the University of Cagliari. His research interests focus mainly on architectures, software and tools for embedded and portable computing, with particular emphasis on: languages, architectures and compilers for reconfigurable and parallel computing; Networks-on-chip; Operating systems for multiprocessor-systems-on-chip; low power real-time scheduling algorithms.
salvatore@unica.it

**Danilo Pani** (2002 Electronic Eng. Master, 2006 Electronics and Computer Science PhD) joined the Department of Electrical and Electronics engineering of the University of Cagliari, Italy in 2002 as Electronics and Computer Science PhD student. His primary research interests are in the area of Digital Signal Processing architectures and systems, Biomedical Engineering, Reconfigurable Systems and Cooperative VLSI architectures for distributed computing.
pani@diee.unica.it

**Luigi Raffo** (1989 Master, 1994 Electronics and Computer Science PhD) joined Department of Electrical and Electronics Engineering of the University of Cagliari, Italy in 1994 as assistant professor. From 1998 he has been professor of Digital System Design, Integrated Systems Architectures and Microelectronics at the same Department. His research activity is mainly in the design of low-power analog and digital architectures/chips. He has been project manager of many local and international projects. He is author of more than 50 international papers in the field.
raffo@unica.it