A Run-Time Word-Level Reconfigurable Coarse-Grain Functional Unit for a VLIW Processor

Natalino G. Busa'
Philips Research Laboratories
High Tech Campus, Prof. Holstlaan 4, 5656AA
Eindhoven, The Netherlands
Natalino.Busa@philips.com

Carles Rodoreda Sala
Sony España S.A. Design Division
Poligono Industrial Can Mitjans,
Viladecavalls - Barcelona (Spain)
Carles.Rodoreda@eu.sony.com

ABSTRACT

Nowadays, new DSP applications are offering combined and flexible multimedia and telecom services. VLIW processor architectures, which include dedicated but inflexible functional units, are usually tuned to a single specific application. In order to accelerate a wide range of applications, we propose a VLIW processor containing a novel run-time reconfigurable functional unit (RC-FU). Only a few hundred bits and few cycles are necessary to configure a new coarse-grain operation on the RC-FU unit. After reconfiguring its internal datapath and micro-program, the RC-FU can execute a number of look-alike DSP functions, such as 8-point DCT or 4-point FFT. The RC-FU itself is a VLIW processor and the configuration contexts are generated using a high-level synthesis tool. The proposed RC-FU provides high processing power and can be efficiently tuned to the requirements of a variety of DSP applications.

Categories and Subject Descriptors

C.1.2 [**Processor Architectures**]: Single Data Stream Architectures – *RISC/CISC, VLIW architectures.*

General Terms

Algorithms, Performance, Design

Keywords

Reconfigurable logic, VLIW processors, Architectural Synthesis

1. INTRODUCTION

Each day, new DSP applications are developed offering yet more features and support for multiple and emerging standards. Currently, we are witnessing the convergence of different sorts of video, audio and telecom application on a single, often portable, device. Hence, in order to efficiently perform the required applications, such devices must be equipped with dedicated and computationally powerful processing hardware. Paradoxically, the designed hardware must also be highly configurable and programmable [1].

Three sorts of processor's architectures are currently used in order to execute the current DSP applications, namely, Application Specific Architectures (ASA), RISC-like architectures, and VLIW architectures (see Figure 1).

Application Specific Architectures are highly optimized and tailored for a specific functionality [2]. A large number of dedicated computational resources are deployed in the architecture, providing very high computational power. On the other hand, such architectures are often not programmable and cannot be easily adapted for the execution of other applications.

RISC-like processor architectures, on the other hand, are highly programmable and can cope with a broad range of (irregular) DSP applications, but usually deliver modest or low performance.

VLIW architectures are located in between these two extreme architectural solutions [3]. Many commercially available DSP processors can be placed in this category. The demanded computational performance can be achieved by including in the datapath one or more dedicated ASUs (Application Specific Functional Units). Those ASUs are capable of executing complex coarse-grain operations, characterized by an instruction latency of tens of cycles, and processing blocks of data as single atomic operations. Typical coarse-grain operations are, for instance, 4-points FFTs, 8-point FIR filters. Coarse-grain ASUs contain internally a large number of internal fine-grain resources such as adders and multipliers [4].

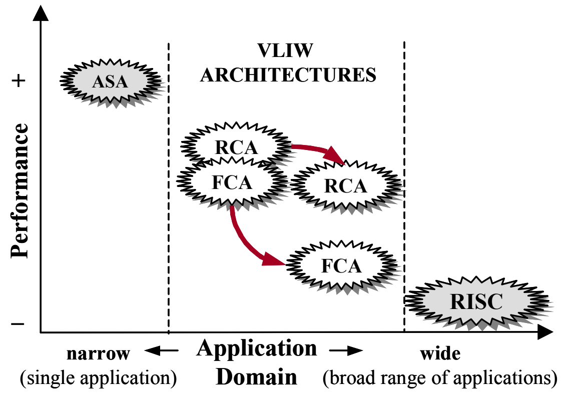

Figure 1. Performance vs Flexibility tradeoff

When the coarse-grain operations are implemented in a dedicated, non-configurable ASU's internal architecture, only a limited set of applications can benefit from the computational acceleration the ASU provide. Modern DSP applications are continuously evolving in terms of provided standards and services. While the insertion of fixed custom ASUs in a VLIW datapath increases the processor performance for a given application, at the same time it reduces the flexibility and the programmability of the VLIW processor. If the application changes, then the processor's performance drops considerably [1]. Therefore, such a Fixed Coarse-Grain VLIW Architecture (FCA) would deliver very low performance while moving away from the original application domain (see Figure 1).

In order to overcome this flexibility-performance dilemma [5], we propose a novel VLIW architecture containing a Run-Time Word-Level Reconfigurable Coarse Grain Functional Unit (RC-FU). Various coarse-grain operations can be reconfigured on the proposed RC-FU, in order to accelerate cycle budget critical DSP loops. In this way, a Reconfigurable Coarse Grain VLIW Architecture (RCA) would provide high processing power for a wide domain of DSP applications with dissimilar computational requirements (see Figure 1). The coarse-grain operation will be available after a run-time configuration step of a few cycles. During this time, the RC-FU's internal resources, such as the controller, the datapath network and the data memories, will be re-programmed using a few hundreds configuration bits. In this way, the RC-FU's instruction set can be adjusted and tailored *at run-time* to the requirements of the application code, which is about to be executed. By reconfiguring the RC-FU coarse grain instructions, hence the VLIW Instruction Set, the processor will execute fewer and more compact instructions.

The paper is organized as follows. Section 2 depicts the current state and the available results on re-configurable VLIW architectures. In Section 3, advantages and drawbacks of fixed coarse-grain VLIW architectures will be examined. In Section 4 and 5, the characteristics of the RC-FU will be introduced together with the integration of functional unit in a hierarchical VLIW processor architecture. Section 6, describes the programming paradigm, which has been adopted for the run-time reconfiguration of the RC-FU instruction set. The experimental results are reported in Section 7, followed by some conclusions.

2. RELATED WORK

Mihai Sima et. al. [3] have designed a Reconfigurable Functional Unit (RFU) for a Trimedia/CPU64 VLIW datapath. A 16-cycle coarse-grain 8-point IDCT operation has been configured on the above-mentioned RFU. The coarse-grain operation is configured at bit-level and mapped on an ACEX EP1K100 FPGA from Altera, which is embedded in the VLIW processor's datapath. In this way, adders and multipliers bit-level architectures are re-configured any time a new coarse-grain operation is configured. Usually, those basic FUs can be re-used and the reconfiguration of their bit-level architectures is not strictly necessary.

In the VLIW processor template Lx [1], word-level re-configuration is possible by customizing the functionality provided by processing clusters. However, when the application domain is identified, the provided flexibility is removed and the hardware is frozen for the final implementation.

The Concise RFU [6], is word and bit level reconfigurable at run-time. Anyhow, the RFU coarse-grain operations cannot be executed in parallel with other operations because the RFU is embedded in a RISC-like architecture. In this case, the ILP is limited to the acceleration provided by the RFU itself.

In the Chimaera architecture [7], the RFU's configuration context is quite big (32x1674 bits). Furthermore, the RFU could negatively influence the timing of the whole processor, because of possible long signal propagation paths in the logic circuits configured in the RFU.

Each of the above-mentioned frameworks has some similarities with the proposed architecture [5]. Nevertheless, none of them offers, at the same time, a mixed coarse/fine grained VLIW Instruction Level Parallelism, small reconfiguration context (less then 500 bits), low run-time reconfiguration latency.

3. FIXED COARSE GRAIN ASUs

Coarse-Grain ASUs are characterized by an instruction latency of tens of cycles. They can process blocks of several data words as single atomic operations and manifest complex Input/Output activity patterns [4].

In terms of code size, the introduction of fixed coarse-grain ASUs in a VLIW datapath can be very beneficial. In order to control an ASU, only a few bits wide command must be encoded in the VLIW instruction word. On the other hand, the ASU steers a large amount of fine-grain internal FUs. The program code compiled for a VLIW architecture containing coarse-grain ASUs is denser, both in terms of code length and code width, when compared to a VLIW architecture containing only fine-grain FUs. The length of the code is shorter because of the higher instruction level parallelism provided by the ASUs and the code width is small because the control of the ASU's internal hardware resources is not encoded in the VLIW instruction word (compare architecture and microcode aspect ratio of case *a* and *b* in Figure 4). Nevertheless, such a fixed coarse-grain ASU has some limitations, which could lead to an inefficient hardware utilization of its internal resources.

The ASU's hardware is fixed, because it is tuned and tailored for a specific application. Each day, new DSP applications are developed. This could imply that the coarse-grain operations provided by the available ASUs do not match any more with the requirements of a new DSP application. In this case, no dedicated hardware is exploited to accelerate the application, and the VLIW processor delivers poor performance [1]. If the coarse-grain operations are not performed, then a large portion of the datapath resources could remain idle for long periods during the execution of the DSP application. Furthermore, those hardware resources are hidden in the ASU and they are not accessible directly from VLIW controller.

Once silicon is produced, a modification in the DSP application may require the design of new coarse-grain ASUs. Therefore, either new silicon must be produced or the application's performance decay.

A last, but not least, limitation of inflexible coarse-grain ASUs comes as a direct consequence of the Amdahl's law [8]. Once that the critical loops are accelerated thanks to the specialized coarse-grain operations provided by the ASU, other portions of the code become the bottleneck of the application and other loops become critical.

4. A RECONFIGURABLE RC-FU

The RC-FU (Run-Time Word-Level Reconfigurable Coarse Grain Functional Unit) is itself a VLIW processor. The reconfigurable coarse-grain operations are stored in the RC-FU microcode memory. The datapath consists of many fine-grain FUs such as adders and multipliers as well as medium-grain units such as complex multipliers, complex rotators, and butterfly units as depicted in the bottom layer of Figure 2. Those internal FUs are partially reconfigurable, for instance, in terms of processing precision and word-level functionality.

We define a RC-FU reconfiguration context as the total amount of data (microcode, interconnect, constants) needed to configure a new coarse-grain operation. The reconfigured resources are shaded in gray in the RC-FU diagram of Figure 2. According to which RC-FU internal resource is actually reconfigured, the configuration process can be divided in the following steps:

- **Controller Microcode**: When a new microcode is downloaded in the RC-FU's controller instruction memory, a new coarse-grain operation is available.
- **Datapath Interconnect Network**: The datapath busses, which convey data among the internal RC-FU computational resources, can be re-arranged.
- **Internal Data ROMs**: According to the configured coarse-grain operation a number of constants could be programmed into the internal RC-FU memories.

In order to gain the best performance results, a careful analysis of the application critical loops must be performed. Once identified the right functions to be accelerated, the routines are removed from the application program code and then mapped on the RC-FU word-level reconfigurable hardware. As a result of this last step, a number of RC-FU configuration contexts are generated. They are downloaded at run-time to the RC-FU architecture. From that moment on, a new coarse-grain operation is available in the VLIW Instruction Set.

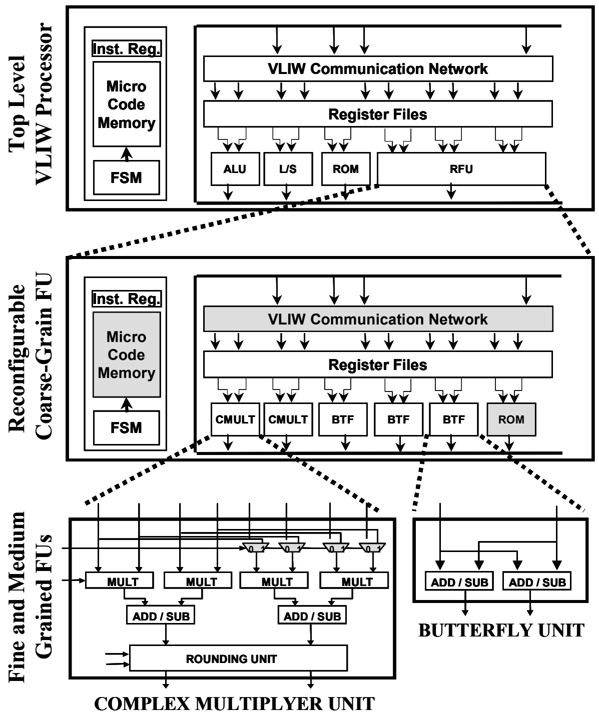

Figure 2. Hierarchical VLIW Processor architecture, including a RC-FU

In the proposed setup, a new configuration context is generated by exporting the controller microcode ROM of the VLIW RC-FU as generated by the high-level synthesis tool A|RT Designer [9]. The tool can compile a new microcode, given the C code of a new coarse-grain function, for a VLIW architecture where the datapath resources have been frozen (the RC-FU datapath cannot be bit-level reconfigured, nor can the type and the number of the RC-FU's internal resources change at run-time).

5. A HIERARCHICAL VLIW TEMPLATE

The VLIW architecture is composed of three layers as depicted in Figure 2. It is a hierarchical template because the RC-FU, which is a re-configurable coarse-grain ASU of the top VLIW processor, is itself a VLIW processor. In this way, given a DSP application, the traditional HW-SW partitioning problem can be revisited into a HW-SW mapping problem. The problem is now defined as how to identify the right hardware layer at which the software routines are executed.

In the proposed template, the hardware hierarchical components (see Figure 2) are related to the following software programming entities:

- **Application**: the top VLIW processor executes the DSP application. Irregular code and non-critical loops of the application are executed by the standard FUs of this processor, while the critical DSP routines are accelerated by the RC-FU. The top VLIW processor is responsible for the correct configuration of the RC-FU.
- **Function**: the RC-FU VLIW processor executes dedicated and application specific DSP routines such as 8-point DCTs or 4-points FFTs.
- **Operation**: At this level the medium-grain operations are processed in terms of fine-grain simple additions and multiplications.

6. RC-FU PROGRAMMING PARADIGM

The instruction set of the RC-FU is relatively simple and consists of only five different commands. This is a consequence of the fact that, for the top-level VLIW processor, the RC-FU internal functionality is not visible and any coarse-grain operation is encoded with the very same command (see below "RFU_RUN").

- **RFU_CONFIG**: By executing this command, a configuration word is stored internally in the RC-FU. The word could be stored either in the microcode memory of the RC-FU controller or in separate configuration registers, in order to define the datapath interconnect network and the content of the RC-FU internal memories.
- **RFU_RESET**: It resets the RC-FU to a known state.
- **RFU_RUN**: By executing this command, the last configured RC-FU coarse grain operation is actually executed. The latency of the RFU_Run command is, in fact, the latency of the coarse-grain operation.
- **RFU_WRITE / RFU_READ**: Input/Output data blocks for the coarse-grain operation are provided and retrieved using these two RC-FU commands.

An example of code showing how the top layer VLIW processor steers the RC-FU is given in Figure 3. Once the RC-FU is configured and initialized, it can be used as a normal coarse-grained operation, for instance, to accelerate the innermost loop of a FFT function. Once the cycle budget critical DSP function is performed, the RC-FU can be configured for the next computational intensive DSP loop. In order to execute a RC-FU operation, the input data is transferred to the unit, the coarse-grain operation is executed and then the output data is read back. A data dependency exists between these three operations, which lengthen the execution latency of the innermost loop. Anyhow, the latency of the loop can be reduced to the latency of the "RFU_Run" operation, by applying loop folding/unrolling techniques [10].

In order to obtain the required computing acceleration, a sufficient data bandwidth must be available between the RC-FU and the Top Level VLIW processor. In this case, a VLIW architecture with multiple Load/Store FUs would be the best match for such a data greedy RFU.

```
// Download FFT Radix4 microcode to the RFU
int RFU_ContextBaseAddr=FFT_UCODE_BASEADDR;
int RFU_ContextLength  =FFT_UCODE_LENGTH;
for(addr=0;addr<RFU_ContextLength;addr++){
   ucode_addr = addr+ RFU_ContextBaseAddr;
   ucode_instr= RFU_CONTEXT[ucode_addr];
   RFU_Config(addr, ucode_instr);
}
RFU_Reset();
// FFT 1K main loop
for(s=0; s<stages; s++)
 for(c=0; c<clusters; c++)
  for(r=0; r<radixstep; r++)
      ...
      // The 4-point FFT coarse-grain op.
      RFU_Write(RADIX4_INPUT[8]);
      RFU_Write(TWIDDLE_COEFF[8]);
      RFU_Run();
      RFU_Read(RADIX4_OUTPUT[8]);
      ...
```

Figure 3. Segment of code configuring and executing a RC-FU coarse-grain operation

7. EXPERIMENTAL RESULTS

A VLIW processor containing a RC-FU (architecture (e) in Figure 4) has been compared with other 6 VLIW architectures. The various VLIW processors have been tested according to run-time performance, code size and silicon utilization.

The architectures pictured in Figure 4 are built using the following functional resources. The VLIW datapaths (b), and (c), contain, respectively, a 4-point FFT and an 8-point DCT fixed coarse-grain ASU, while datapath (d) contain both the two ASUs. The datapath (e) contains a RC-FU, while (a) is a RISC-like datapath (1 adder, 1 multiplier). Finally, template (f) is a flattened VLIW architecture, provided in two flavors: (f ′) medium-grain (2 CMULTs, 3 BTFs) and (f ″) fine-grain (10 adders, 8 multipliers). The architectures (b), (c), (e), and (f), contain the same amount of computational resources, even though differently clustered. Contrarily, architecture (d) contains twice as many FUs, and the VLIW processor (a) is a stripped down template with no extra hardware to accelerate the application code.

Two different application kernels have been compiled for the above-mentioned architectures, namely a 1024-point FFT, and a SQCIF (128x96 pixels) 2D-FDCT. Finally, both kernels are combined in the third application. When the two kernels are combined, the advantages of the VLIW processor containing a RC-FU are substantial, as shown in Figure 4 and Tables 4 and 5.

Together with the diagrams of the various VLIW architectures, Figure 4 shows the aspect ratio of a microcode containing two critical loops demanding different dedicated coarse-grain operations. The best architecture, for what code density is concerned, is the one with the smallest microcode. Architecture (a) does not have to control many FUs and its microcode is narrow, but none of the two critical loops is accelerated. Architecture (b) and (c) have a wider microcode but they can both accelerate only one of the two loops.

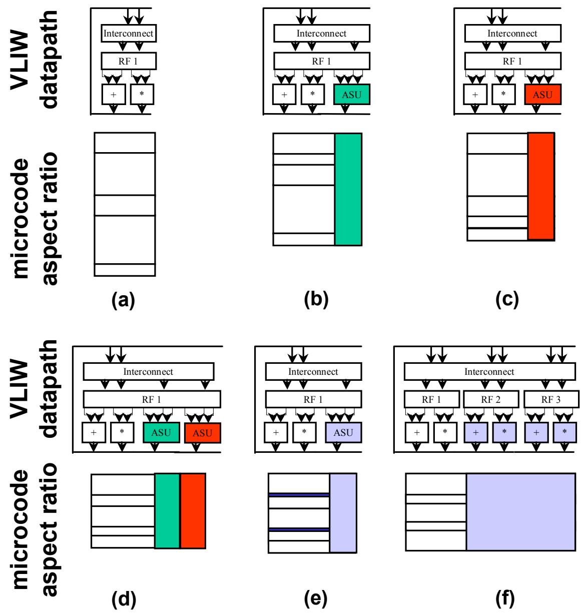

Figure 4. The compared VLIW datapaths, and their relative controller's microcodes, as compiled for an application containing two different DSP loops

On the other hand, architectures (d), (e) and (f) can accelerate both critical loops. Additionally, only architecture (e) has a compact microcode while the other two require a large microcode instruction width. Furthermore, architecture (d) achieves high execution performance by deploying twice as many hardware resources in the VLIW datapath.

In the architecture (f) the very large instruction set is constantly available during the whole application, but it is effectively exploited only during the small portion of code of the two critical loops. On the contrary, in architecture (e), the instruction set is tuned at run-time, in order to temporarily match the requirement of critical loop, which is about to be executed. The black stripes in the microcode (e) in Figure 4 represent the code overhead due to the run-time reconfiguration of the RC-FU and, hence, of the VLIW instruction set.

In the proposed experiment, the RC-FU's internal datapath consists of two Complex Multipliers (CMULTs) and three Butterfly Units (BTFs). The CMULT unit itself consists of four multipliers and three ALUs. It can be word-level reconfigured in order to perform either a complex multiplication or four MAC operations. The CMULT is a 3-cycle pipelined unit. The BTF unit consists of two ALUs. It can be used in order to perform single-cycle butterfly operations as well as 4-input addition/subtraction operations. For the given experiment, four different configuration contexts have been generated. Therefore, the following four different coarse-grain operations can be configured on the RC-FU, as reported in Table 1.

Table 1. Some RC-FU's coarse-grain operations

RC-FU Coarse-Grain Ops.	Operation Latency (cycles)	RC-FU Controller area (bits)
4-point FFT	23	475
8-point DCT	15	275
8-point IDCT	17	325
8-taps FIR Filter	12	225

The RC-FU has eight 16-bit wide input and output ports. The controller microcode for this particular experiment consists of a RAM of 20 instructions by 25 bits. The configuration latency, for the above-reported coarse-grain operations varies from 2 to 4 cycles. Usually, for typical DSP applications, the overhead due to the RC-FU reconfiguration latency can be neglected. Anyway, if this is not the case, the critical loops must be restructured, in order to reduce/hide the run-time reconfiguration latency overhead. This can be done, for instance, by clustering those critical DSP loops, which are using the same coarse-grain operation, hence the same RC-FU configuration [11]. The controller microcode as well as the RTL VHDL for the RC-FU have been generated using the architectural synthesis tool A|RT Designer [9].

Before testing the various architectural solutions on the combined application (FFT + DCT kernels), the FFT and the FDCT kernels have been tested separately. Both kernels have been compiled for the proposed seven architectures, and then benchmarked according to run-time performance and code density.

When the application is so well defined, as in the proposed 1024-Point FFT kernel (see Table 2), a RC-FU cannot be exploited adequately. On the contrary, a fixed coarse-grain 4-point ASU offers the best performance - code density tradeoff. A flattened architecture is nonetheless the fastest solution, even though not code-efficient at all.

Table 2. Results for a 1024-point FFT kernel

	Performance (cycles)		Microcode Area (bits)
	4-point FFT	Application	Area
(e) RC-FU	23	33'589	7420
(b) 4-point FFT	20	29'749	4685
(f ') CMULTs, BTFs	20	29'749	7395
(f ") ALUs, MULTs	18	27'290	8712
(a) No acceleration	55	91'213	7384

The same considerations, as already mentioned for the 1024-Points FFT, can be derived for the FDCT application kernel (see Table 3). The coarse-grain fixed 8-point forward DCT (c) is the best tradeoff, while (f ″) is the fastest. Still, the RC-FU is not successfully exploited.

Table 3. Results for the SQCIF 2D-FDCT kernel

	Performance (cycles)		Microcode area (bits)
	8-point DCT	Application	Area
(e) RC-FU	15	53'952	7334
(c) 8-point DCT	12	44'736	6713
(f ') CMULTs, BTFs	12	44'736	10875
(f ") ALUs, MULTs	10	38'592	12887
(a) No acceleration	31	103'104	6780

Conversely to what said above, when the two kernels are combined in a single application code (see Figure 4 and Table 4 and 5), then the architecture including a RC-FU offers the best performance - code tradeoff. It is almost as fast as the flattened architectures (f ′) and (f ″) but its application code is much more compact.

The architectures (b) and (c) cannot speed up both loop kernels, hence delivering poor performance. Architecture (d) delivers good performance but deploys twice as many resources as in the RC-FU and exploits them poorly (only one of the two ASUs is active at a time).

Table 4. FFT + 2D FDCT kernels: combined performance

	Performance (cycles)		
	4-point FFT	8-point DCT	Application
(e) RFU	23	15	87'568
(b) 4-point FFT	20	31	132'853
(c) 8-point DCT	55	12	135'949
(f ') CMULTs, BTFs	20	12	74'485
(f ") ALUs, MULTs	18	10	65'882
(d) RADIX4 + 1DFDCT	20	12	74'485
(a) No acceleration	55	31	194'317

Table 5. FFT + 2D FDCT kernels: code density

	Microcode Area (bits)		
	Width	Length	Area
(e) RFU	86	173	15246
(b) 4-point FFT	85	164	14233
(c) 8-point DCT	85	198	17106
(f ') CMULTs, BTFs	145	126	18270
(f ") ALUs, MULTs	182	119	21599
(d) RADIX4 + 1D-FDCT	109	126	15234
(a) No acceleration	60	236	14160

The microcode width of architectures (e), (b), and (c) is almost the same, but the VLIW architecture containing an RC-

FU delivers a much higher performance. In general, the higher the number of critical loops requiring different coarse-grain operations and the more advantageous will be to include an RC-FU in the VLIW processor's datapath.

8. CONCLUSIONS

A novel coarse-grain run-time reconfigurable FU (RC-FU) for a VLIW template has been proposed. The RC-FU can be configured to implement a number of DSP functions, typically used in multimedia and telecom applications. The configuration context needed to configure the RC-FU is very modest and can be downloaded in few cycles. Yet, a variety of DSP functions can be implemented only acting on the word-level configuration and on the interconnection of the RC-FU's internal resources. The novel RC-FU is itself a VLIW architecture, whose configuration context can be generated using an architectural synthesis tool.

The proposed processor architecture combines some of the characteristics typical of RISC and CISC architectures, for what Instruction Set design is concerned.

Formerly CISC (Complex Instruction Set Computer) processors were designed to provide an extended set of complex operations that were internally translated in a micro-program made of yet simpler operations. This approach revealed soon its advantages but also its drawbacks. Many of the complex operations added in the instruction set were rarely used, increasing with no actual benefit the complexity of the processor's control unit. On the other hand, the fast and simpler RISC (Reduced Instruction Set Computer) processors are lacking efficient hardware support for telecom and multimedia applications, loosing in code density and performance.

The proposed hierarchical VLIW processor architecture could combine the advantages given by the two above-mentioned classic approaches. A VLIW machine with modest instruction level parallelism (RISC-like architecture) is enhanced with a complex RC-FU (Run-time reconfigurable coarse-grain Functional Unit). The complex coarse-grain operations executed by the RC-FU are described by means of micro-programs, just like in CISC architectures. But, the RC-FU micro-program as well as some characteristics of the RC-FU internal hardware architecture are run-time reconfigurable, allowing a high exploitation of the RC-FU internal hardware resources while executing a large variety of telecom and multimedia applications.

9. REFERENCES

[1]. Paolo Faraboschi et al, "Lx: A Technology Platform for Customizable VLIW Embedded Processing", ISCA 2000, Vancouver, British Columbia Canada.

[2]. Marco Bekooij et. al. "Power-Efficient Application-Specific VLIW Processor for Turbo Decoding", IEEE ISSCC 2001

[3]. Mihai Sima, at. al. "An 8x8 IDCT Implementation on an FPGA-augmented TriMedia", in proc. of IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM), California, May 2001

[4]. Natalino Busá, et al. "Scheduling coarse grain operations for VLIW processors", Madrid, Spain, 2000, ISSS.

[5]. Reiner Hartenstein "A Decade of Reconfigurable Computing: A visionary Retrospective", in Proc of Design Automation and Test in Europe (DATE) 2001.

[6]. Bernardo Kastrup et. al "ConCISe: A Compiler-Driven CPLD-Based Instruction Set Accelerator" in proc. of IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM), California,April 1999

[7]. Zhi Alex Ye et.al. "CHIMAERA: A High-Performance Architecture with Tightly-Coupled Reconfigurable Functional Unit", ISCA2000, Canada

[8]. G. Amdhal "Validity of the single processor approach to achieving large scale computing capabilities", in proc. AFIPS 1967 Spring Joint Computer Conf. Atlantic City, N.J. (USA) pp. 483-485

[9]. Adelante Technologies, http://www.adelantetech.com

[10]. Ramlakan Gupta et. al. "Synthezising a Long Latency Unit within a VLIW processor", 14th Inter. Conf. on VLSI Design 2001, India, pp 460 -465.

[11]. Rafael Mestra et. al. "A framework for reconfigurable computing: task scheduling and context management", in IEEE Trans. On VLSI Systems, vol. 9, no.6, pp.858-873