Reconfigurable Microprocessor and Microcontroller – Architectures and Classification

Christian Siemers and Harald Richter

Technical University of Clausthal, Institute for Computer Science, Julius-Albert-Str. 4, D-38678 Clausthal-Zellerfeld, Germany, siemers|richter@informatik.tu-clausthal.de

Abstract. Reconfigurable microprocessors and microcontroller are facing their market introduction on a broad base. What are the advantages of these new devices, the challenges, and is a classification of different architectures available? This paper gives an introduction into this new field and provides some hints for applying reconfigurable microcontroller to different classes of applications.

1 Introduction

The microprocessor world is becoming more complicated. While the item *processor* was solely used for the Central Processing Unit (CPU) of the von-Neuman-model, this has changed through the last years: A processor is something like the heart of a computational machine.

The connection of the processor and – related to this – of the item *software* with the von-Neuman-model, specifically with the execution model of von Neuman, is still present but becoming more loosely. Therefore the first question arises how significant distinctions between several classes of processors can be obtained. The second question concerns the consequences of these distinctions.

Reconfigurable processors as well as microcontroller mark a significant new area: After developing different processor and execution models as well as finding characteristics to distinguish between them, reconfigurable processors contain nothing less than joining at least two different paradigm inside. They are by definition of hybrid nature, and this makes computing much more powerful and efficient at the price of more sophisticated programming.

This paper intends to discuss some aspects of these reconfigurable devices. Chapter 2 gives an introduction to some different execution paradigm and the resulting consequences. Chapter 3 discusses the overall architecture of computing machines from a three-tiers-approach: software, micro-architecture and fabric. This leads to an explanation of different approaches on the market.

Chapter 4 introduces the role of peripheral elements inside computing machines. These elements, formerly introduced by von Neuman, Goldstine and Burkes as the necessary part for connecting computer and the outside world, tend to add functionality inside for own computing power. This chapter presents finally a classification scheme for microcontroller with integrated reconfigurable processor kernel as well as reconfigurable peripheral elements.

As some suppliers of (re-)configurable processors are present on the market, some of them are discussed concerning their classification inside the scheme in chapter 5.

2 Sequential versus Configurable Computing

The distinction between sequential computing following the execution paradigm by von Neuman and configurable computing used inside programmable logic devices (PLDs), specifically inside the well-known FPGA devices (field-programmable gate arrays), was discussed several times before. One introduction into this topic is given in [1] and shall be summarised inside this chapter.

This paper is published in: Caspar Grote, Renate Ester (eds.): ”Proceedings Embedded World 2004 Conference”, pp.271-287, Nuremberg, Germany, February 2004

2.1 *Execution Dimension*

Figure 1 shows a simple part of a software program, written in C. This program consists of an infinite loop (`while(1)`) and an `if`-construct inside to distinguish between two cases.

```
#define INPORT_ADR 0x1000
#define OUTPORT_ADR 0x2000

unsigned char *a = INPORT_ADR;
unsigned char *b = OUTPORT_ADR;

main()
{
   while(1) {
      if( *a == 0 )
         *b = 1;
      else
         *b = 0;
   }
}
```

Figure 1. Example program written in C

The translation for a von-Neuman-based processor consists of several instructions. Figure 2 shows this translation for a RISC-like architecture: While the details are not important, the principle is that the program consists not only of several instructions but a well-defined sequence. The execution of this program is performed by executing the first instruction, followed by the next. Data-dependent branches lead to several ways the program may take, shown in figure 3. It should be noted that this kind of execution is maintained even in superscalar architectures with instruction level parallelism (ILP).

```
L0:  MOV   R1, #a      ; Address of a
     MOV   R2, #b      ; in R1, of b in R2

L1:  LD    R3, (R1)    ; Inport bits in R3
     CMP   R3, #0      ; IF-condition
     BNE   L3          ;

L2:  MOV   R4, #1      ; IF-branch
     JMP   L4          ;

L3:  MOV   R4, #0      ; ELSE-branch

L4:  ST    (R2), R4    ;
     JMP   L1          ;
```

Figure 2. Translation of the source-code from figure 1 into assembler code

The semantic interpretation of this program leads to a complete other solution: The program and the executing microprocessor (or microcontroller) behave like a NOR-gate with 8 inputs, and this could be directly implemented in hardware. Figure 4 shows one solution using an AND-gate and some inverting elements, this implementation is used inside e.g. PLDs

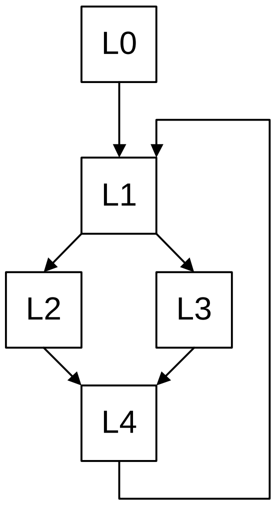

Figure 3. Control flow graph using basic blocks

As PLDs are also programmable – this means that the way the IC works is finally defined outside the factory – the algorithm shown in figure 4 must be mapped into the IC. This is done by structuring pre-defined logic blocks and their communication lines inside. In the case of figure 4, 8 inputs were used, connected to an inverting element each, and the inverting elements were connected to the inputs of an AND-gate with at least 8 outputs. The difference in execution, compared to the von-Neuman-paradigm, is that after structuring has taken place, the complete program works as one instruction: No sequence is performed inside.

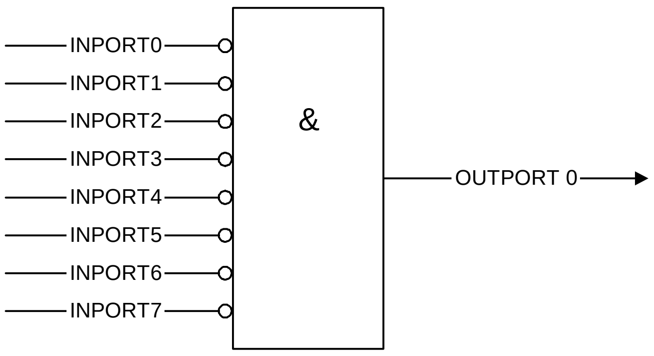

Figure 4. Optimised solution for implementation inside PLDs

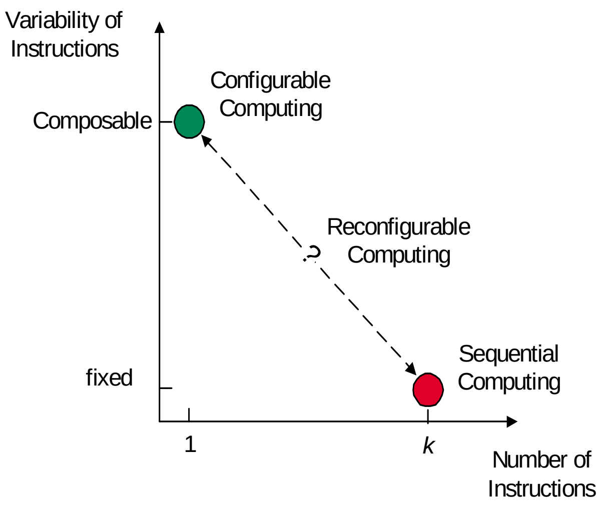

Figure 5. Configurable and Sequential Computing

This leads to the item *execution dimension*: While execution in the von-Neuman paradigm is performed through a sequence of pre-defined instruction – sequence in the sense of sequence in time – , execution in the PLD paradigm is done by structuring pre-defined instructions in space. Space means the two-dimensional IC area (where two-dimensional could be replaced by three-dimensional in the future), and this distinction leads to *computing in time versus computing in space*. Figure 5 shows the classification of these two computing paradigm with respect to the variability of the executed instruction(s) and the number of instructions per algorithm.

The two computing paradigm appear to be well separated from each other, but a transitional paradigm is known with reconfigurable computing. In this case, an algorithm may be translated into a sequence of configurations.

2.2 Characteristic Timing Values for the Computing Paradigms

A close look on the characteristic timing values for several kinds of executing hardware (this shall be called 'processor' for the rest of this paper) shows further significant distinctions. Table 1 summarises the values.

	(fixed Hardware)		← *(programmable Hardware)* →			
	Custom VLSI	ASIC	PLD, one-time progr.	PLD (Conf. Comp.)	Reconfigurable Computing	µP
Binding time	- fabrication - first mask	metal mask	fuse program	load config.	load config.	cycle
Binding duration	- infinite -		infinite	reset	user-defined	cycle
Programming time	- fabrication -		seconds	µs - ms	cycle	cycle

Table 1. Characteristic timing values for several hardware architectures

The most significant value to distinguish between the three discussed programmable hardware platforms is the binding duration. This defines the time period how long the machine is fixed to execute the actual instruction, and this must change inside a microprocessor but will be fixed inside a PLD. Reconfigurable devices may change by user-definition, and exactly this is the new degree of freedom.

The two characteristics of (re-)configurable computing – computing in space and the user-defined binding time to a specific algorithm – lead directly to two application fields: high performance computing and real-time applications. For high performance computing, the execution in space is very worthy, because long execution times by the time sequence are omitted. High performance does not mean that supercomputing is concerned: High performance within embedded systems means that relatively high performance standards are met using microprocessors with other important characteristics like good power/performance relationship.

Meeting real-time constraints is sometimes hard using von-Neuman processors, because the sequential execution implies that the necessary program for reaction are just not available. This can be significantly improved using (re-)configurable elements, because programs executing in these elements are always available.

Serving performance as well as real-time capabilities are the most important motivations for combining processors and configurable structures to reconfigurable processors. While the distinction between configurable and reconfigurable computing is well-known, characteristics are vanishing in this combination, and therefore both items are used synonymous for the purposes of this paper.

3 *Three Tiers for Computing Machines: Software, Architecture and Fabric*

The role of processors is to execute applications, and these applications are described by 'software'. General-purpose processors are designed to fit all applications, neglecting that the application's demand for program and/or data size might run out every boundary. The question is whether general-purpose processors are doing their job efficiently or not [3].

3.1 General-Purpose and Custom Processors

"You can't fit a square peg into a round hole." Actually, given a large enough hole, you can. This is the answer for general-purpose processors to fit an application into a processor-based system. Figure 6 demonstrates the meaning of this rule for a very simple application.

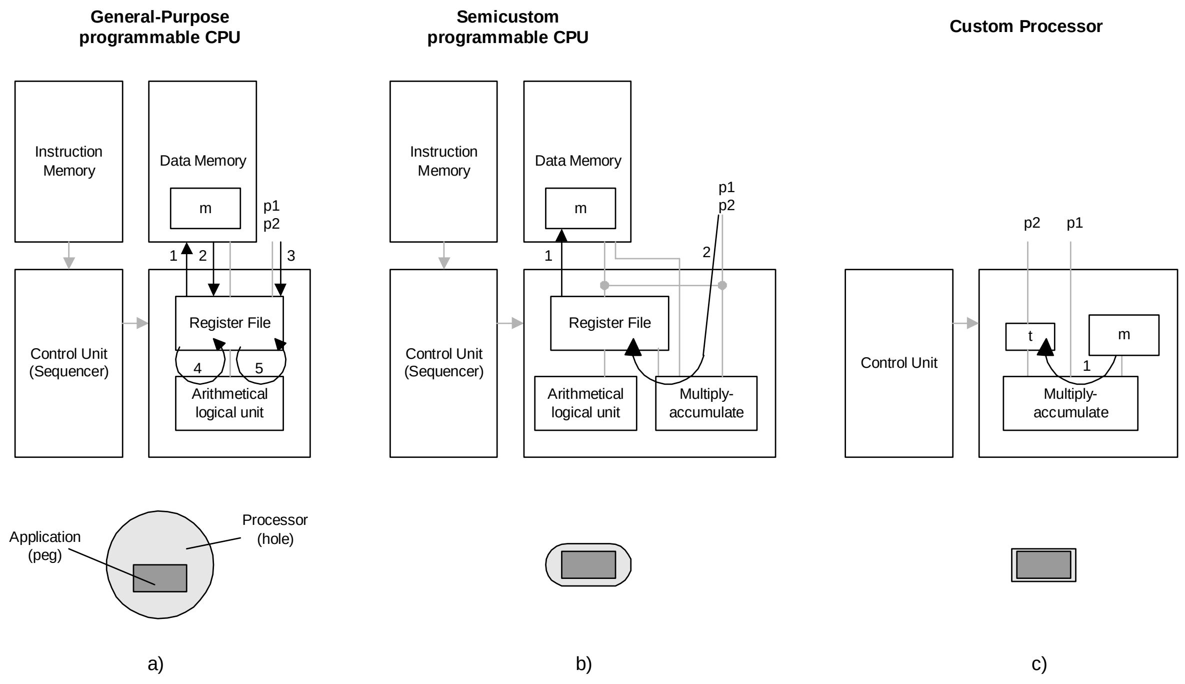

Figure 6. Processor types executing a very simple application
a) General-purpose processor b) semicustom processor c) custom processor

The simple program consists of the implementation of the statement

$$t = t + m[i] * p1$$

It is further assumed that this statement is part of a loop, and after finishing the loop, the result would be written into port *p2*. This multiply-accumulate operation is frequently used within digital signal processing application. A general-purpose processor would use the following sequence of steps to implement this operation:

1. Move *i* from the register file to a data memory address register.
2. Read *m*[*i*] into the register file.
3. Store *p1* in the register file.
4. Read *m*[*i*] and *p1* from the register file, multiply them, and store the result in the register file.
5. Read *t* and *m*[*i*] * *p1* from register file, add them, and store the result in *t* in the register file.

The semicustom processor in figure 6b solves the same algorithm using only 2 steps: Addressing the data memory by the index *i* and simultaneously reading port *p1*, the memory cell and *t* from register file, multiplying *p1* and *m*[*i*], adding the result to *t* and storing the result in *t* in the register file. The custom processor finally executes this within one step.

What's the lesson learned from this? In fact, a custom processor would execute the specific application in a much more efficient and/or accelerated way, on the other side the general-purpose processor is programmable for 'all' other applications too. This leads to the question whether a reconfigurable processor could unify these advantages, and at which price this could be done.

The actual answer to this flexibility/efficiency trade-off is to use semicustom processor wherever applicable. This has lead to e.g. the class of DSPs, the digital signal processors, and to microcontroller. While DSPs have to handle data streams, microcontroller consist of a microprocessor kernel and at least one integrated peripheral function or memory part on the same die. The interesting news is that the microprocessor kernel is often optimised to handle external events and to follow control algorithms with their segmented structure. Both examples of semicustom processors are very frequently used, and if reconfigurable processors want to play an important role in the digital world, they have to compete with at least one of these semicustom processor subclasses.

3.2 Structures of Processor Classes

Figure 7 shows what actually takes place in the processor market and architectures.

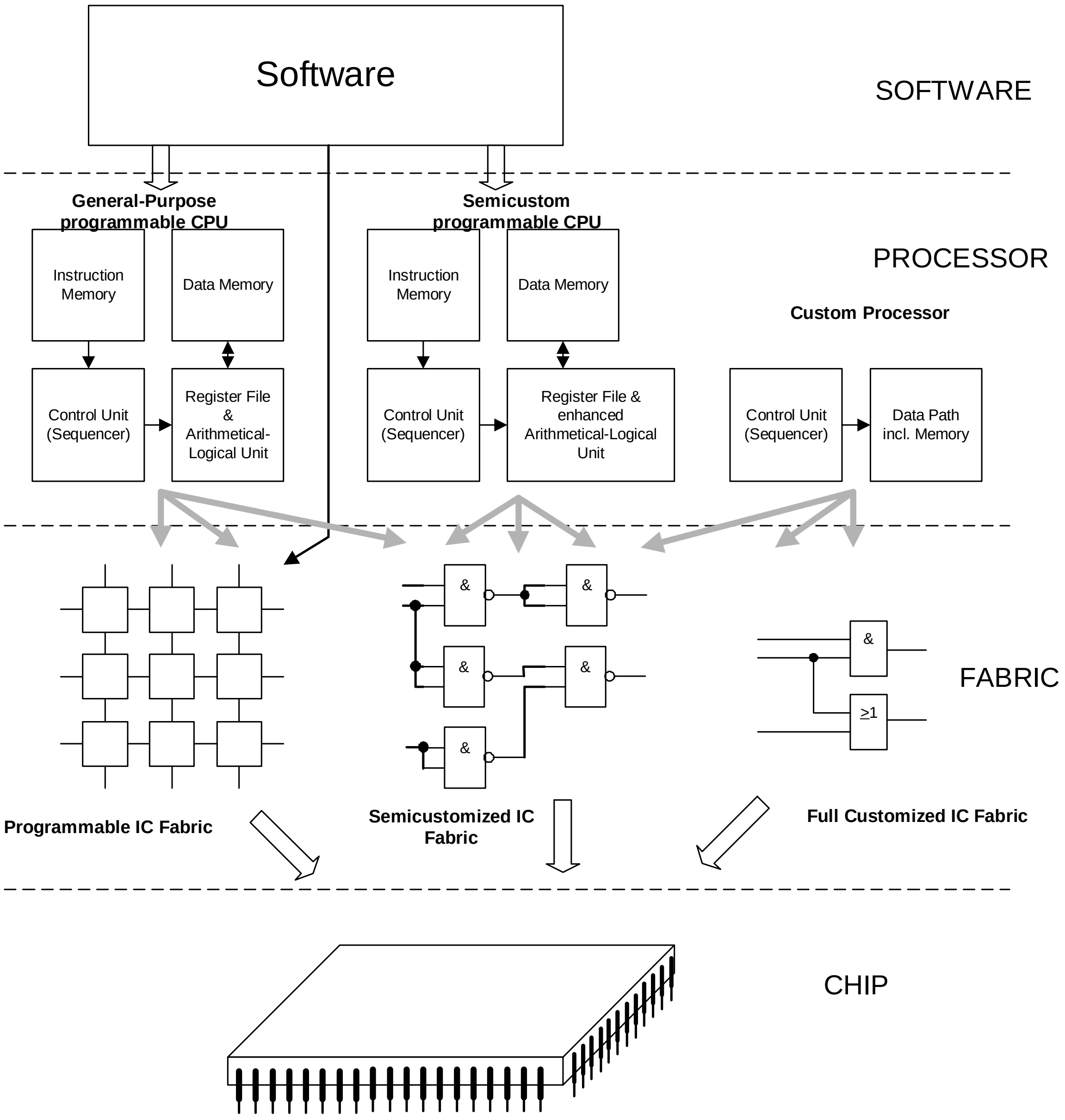
Software
SOFTWARE
General-Purpose programmable CPU
Semicustom programmable CPU
PROCESSOR
Instruction Memory
Data Memory
Control Unit (Sequencer)
Register File & Arithmetical-Logical Unit
Register File & enhanced Arithmetical-Logical Unit
Custom Processor
Data Path incl. Memory
&
≥1
FABRIC
Programmable IC Fabric
Semicustomized IC Fabric
Full Customized IC Fabric
CHIP

Figure 7. Mapping of processor architecture and IC fabric

Three tiers are identified for the use and the structures of processors: Software, architecture and fabric (the chip is just the product but an additional tier).

The software plays an important role for all general-purpose and semicustom processors and may play an important role for custom processors, if they are mapped to programmable IC fabrics. The architecture tier might be separated into general-purpose designs, in almost all cases following the von-Neuman-principle, in semicustom designs, again following the von-Neuman-execution principle but now with enhancements like specialised Harvard memory architectures and additional units inside the processor, and the custom processors. The latter are not limited to any computing paradigm, but design pattern like cellular automaton are of course existing.

All these architectural subclasses may be mapped (and will be mapped in the future) on at least three kinds of fabric: Custom, semicustom and programmable, whereas combinations are of course possible. In this three-tier-presentation, some combinations are well-known, e.g. the combination of a custom processor with a programmable IC fabric: This is a typical application using an FPGA (field-programmable gate array) or a PLD (programmable logic device).

Reconfigurable processors are either general-purpose processors or semicustom processors, mapped partly or even complete to a programmable IC fabric.

3.3 The Role of Peripheral Elements

Inside typical embedded systems applications, the microprocessor kernel is only half of the game, the other part is located inside the peripheral elements. These elements, defined in the von-Neuman-architecture as the connecting unit, are of increasing importance as shown by some studies.

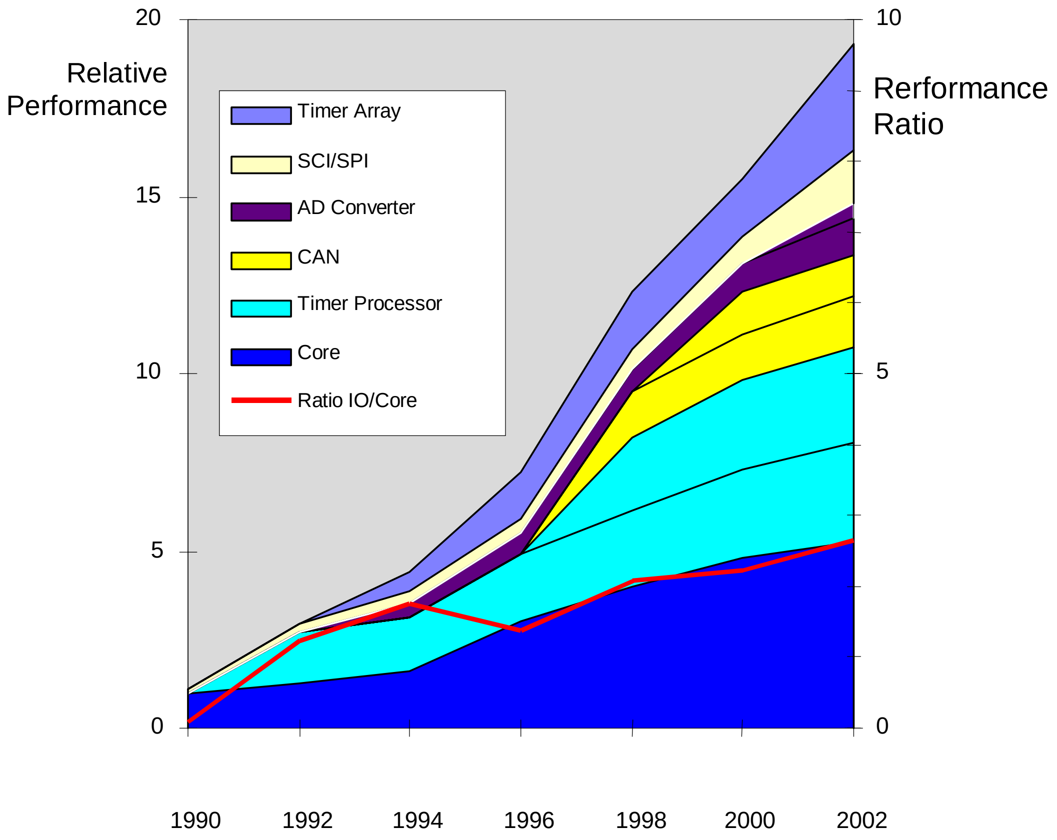

Figure 8. Performance ratio of Input/Output and core

Figure 8 shows the ever growing performance requirements of peripheral elements in the past years. For typical applications in the area of embedded systems, many peripheral elements like bus connections (CAN), timer arrays or processors and AD converter are used, and they have some computational power integrated inside. If this computing power is omitted, the processor kernel must provide it additionally to its own tasks. Beside all considerations concerning real-time capabilities, just for providing the same speed the processor speed (e.g. provided by the clock rate) should be enhanced by a factor of more than 3.

This shows one possible way for the future: Providing peripheral elements with configurable computing capacities makes them more flexible, enhances the real-time capabilities and relieves the processor core from most often periodic tasks.

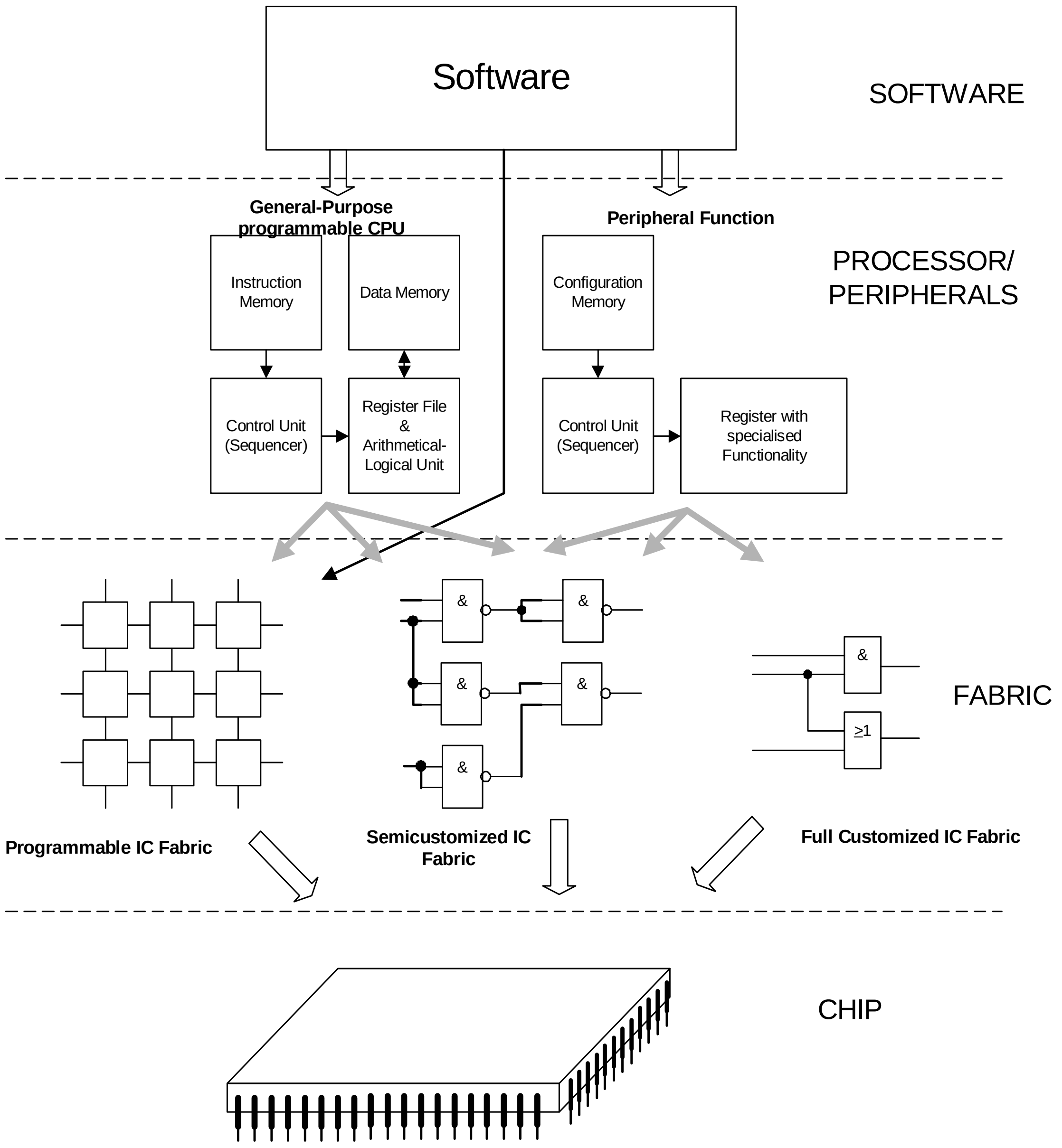

Figure 9. Microcontroller architecture with core and peripheral elements

Figure 9 shows the three-tiers presentation of this microcontroller architecture. It is important to note that the software plays now two roles, because the core itself is programmed by the software as well the peripheral elements. As it is assumed that the peripheral elements are configured – which is in fact a slightly different, more hardware-related kind of software – the software development becomes significantly more complex: This is the price for the improvements.

4 *Classification of Reconfigurable Microcontroller*

4.1 Overview of Reconfigurable Parts inside Microcontroller

An actual microcontroller might be configurable in two main parts: The processor core and/or peripheral elements. Therefore we can identify two classes for reconfiguration: *core reconfiguration* and *I/O reconfiguration.*

The processor core itself may contain reconfigurable parts and sequential working parts clearly separated from each other and loosely coupled through signalling lines. This concept is comparable to a coprocessor approach, where the coprocessor is completely reconfigurable. Another well-known item for this approach is “customised computing machine” (CCM, [4]).

The other approach integrates reconfigurable elements in parts of the core. This is used for defining new operations and instructions. Both approaches are discussed in the following sections.

4.2 The Coprocessor Approach

Figure 10 shows a sample architecture using a RISC-like microcontroller core with a reconfigurable coprocessor loosely coupled to the core.

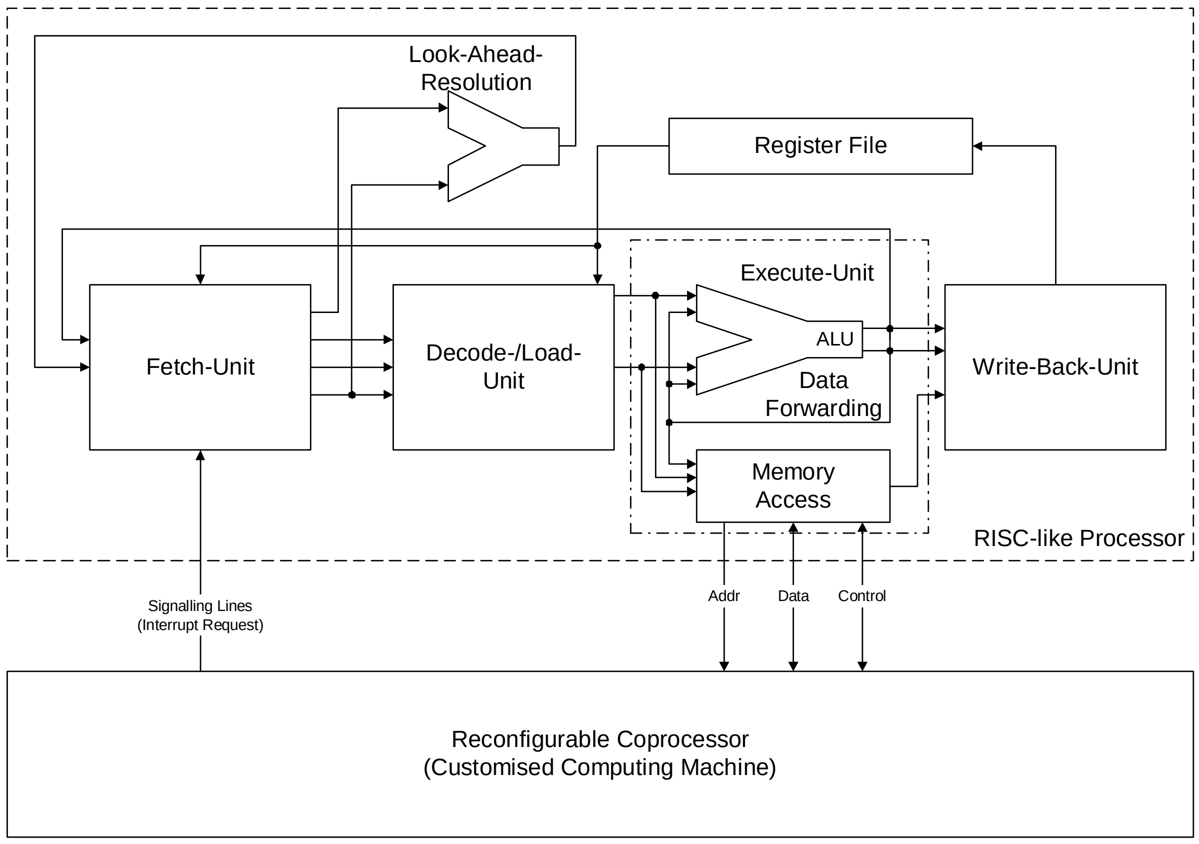

Figure 10. Coupling a RISC-like microcontroller core with a reconfigurable coprocessor

The core architectures shows a 4-stage pipeline including data forwarding (for resolving data dependences) and a look-ahead resolution (for computing branch addresses), while the memory interface is omitted for clearance. Instead of this, a reconfigurable coprocessor is coupled to the core, and this interface requires at least one signalling line to the fetch-unit and a bus system for transferring data using the memory access subunit.

The microcontroller core looks at the coprocessor as part of the memory or as part of the input/output-space, if available. A small number of addresses (e.g. 16) must be mapped exclusively in the address space of this coprocessor, and the processor may submit commands and data and receive results.

This kind of memory-mapped coupling is the absolute minimum. For practical purposes, it is desirable to provide the coprocessor with service request capabilities. These signalling lines, often used as input to the interrupt request system, are of great interest, if the coprocessor works completely asynchronous to the microcontroller core, indeed a very normal use-case. The interrupt request will be coupled to the fetch-unit as shown in figure 10.

This class of reconfigurable microcontroller architecture is very useful for core as well as peripheral applications, because it might be exclusively mapped to support any computation. If the coprocessor has additional links to the input/output system of the microcontroller – in the simplest case, input and/or output pins are connected to the coprocessor – it might act independent of the microcontroller core.

The software interface to support this architecture is comparatively simple, because the application must be partitioned on system or application level. This is a well-known hardware/software co-design approach, where main tasks are identified on system level, modelled using a system description language (which actually might be "C" too), simulated or estimated concerning their runtime behaviour and then partitioned and mapped to different execution units.

Please note that this approach is not simple, it is only *comparatively* simple. Currently the main work is performed by the system designer, and after partitioning the application, the parts often must be coded once more to meet the requirements of the executing system: While the native high-level language of microcontroller is something like "C" or "C++", the native language for configurable systems is VHDL (Very High-Speed Integrated Circuits Hardware Description Language) or Verilog, and even worse for reconfigurable systems there exists no "native language".

4.3 The Reconfigurable Core Approach

Figure 11 uses the same 4-stage pipelining RISC-architecture for discussion of the other approach, which uses reconfigurable enhancements inside parts of the core.

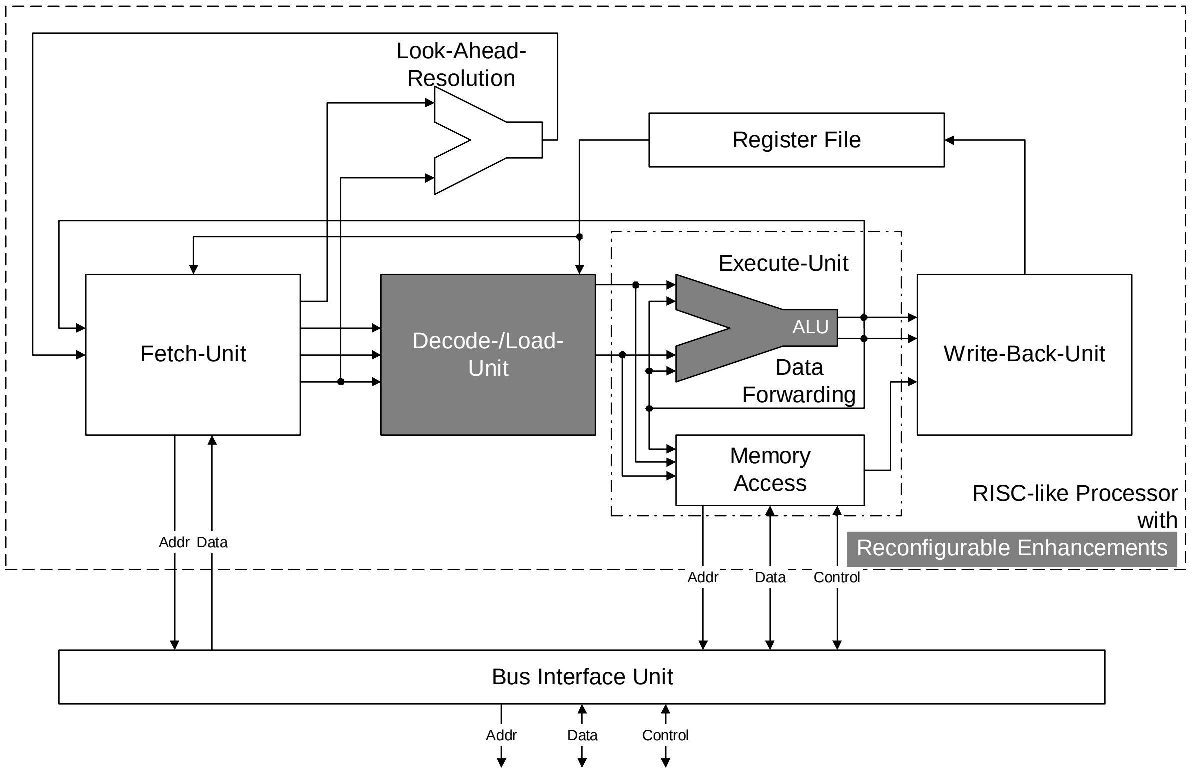

Figure 11. Enhancing a RISC-like core

The most obvious way to create new operations is to locate reconfigurable parts inside the arithmetical-logical unit (ALU). Most of the microcontroller instructions are executed inside this unit (exceptions are branches, jumps and other control flow instructions). If the reconfigurable part consists of pre-defined operations, connectable through configurable data paths, this would obviously create new, application-specific operations and therefore instructions.

Figure 12 visualises a variant of this approach taken from [5]. This system, called Universal Configurable Blocks (UCB), connects some arithmetical units and logical units with the register file and integrates even local RAM for data storage. This system was adopted in [6] to a RISC-like microprocessor showing significant performance improvements up to 1.36 instructions per cycle.

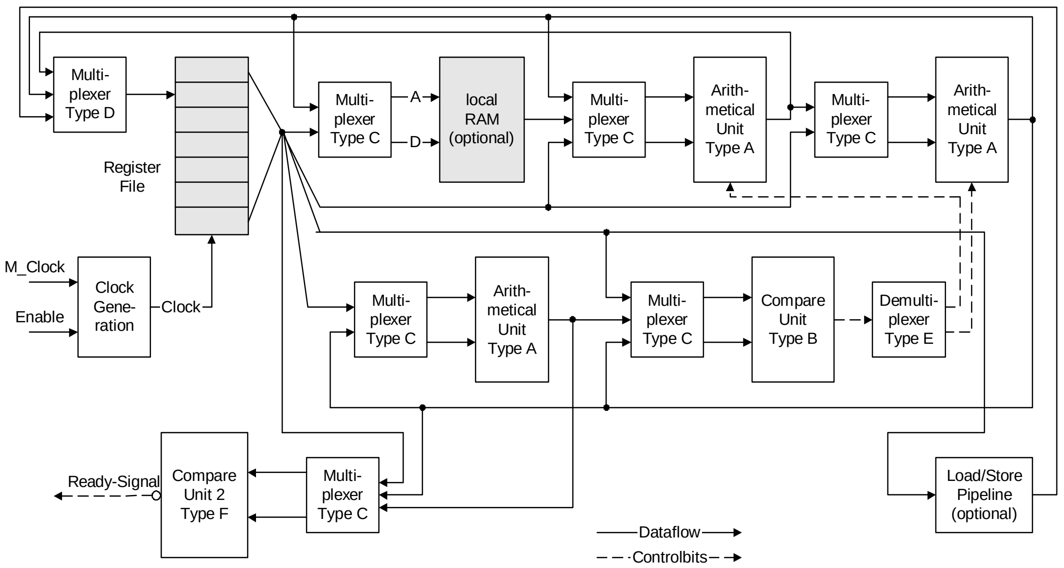

Figure 12. Reconfigurable structure for use inside microcontroller cores

The executing unit is one part of the reconfigurable approach inside the core, the decode/load stage is the other affected unit. It is necessary to decode the new instructions and to load the required register contents to enable the execution part to do the job. Unfortunately this provides additional complexity.

The last remark leads directly to the hardware-software-interface. The development system must provide some capacities to use the additional reconfigurable structures inside the core, otherwise the enhancements would be either useless or only usable by specialists. For general use, the automatic generation of the configurations by the development system is a strong requirement.

The main approach for using reconfigurable structures inside the microarchitecture is to enhance performance. Actual compiler technology analyses the sourcecode in many ways and generates an intermediate representation [7], which is then translated to assembler code in the compiler backend. An analysis of the performance issues of loops or only operations is missing inside this technology.

The profiling analysis, whether measuring or estimating the behaviour, will be one key element of the development tool, especially the compiling part. While the data base for the compiler backend is constant in the ‘classical’ part, it is subject to change when using reconfigurable structures inside the core. Figure 13 b) shows a simplified description of the additional parts. The generation and estimation of profiling information is the base for extracting configurations to enhance performance, which in turn influence the execution time.

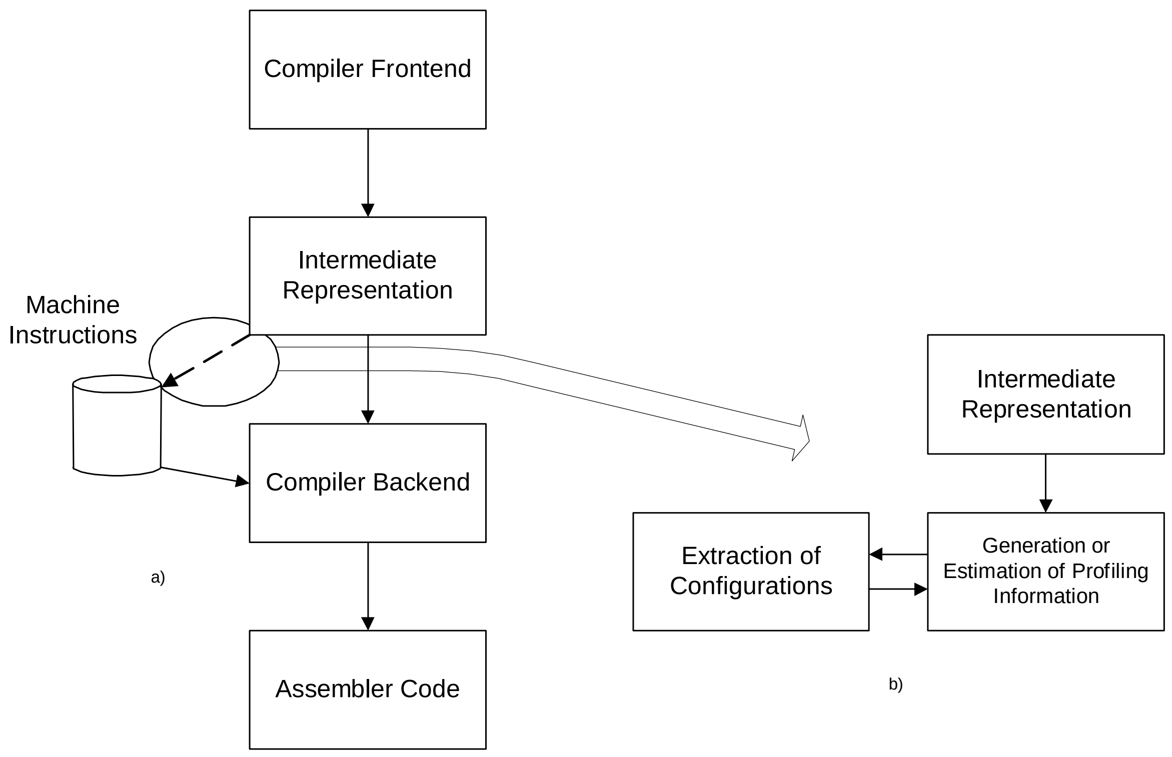
Compiler Frontend
Intermediate Representation
Machine Instructions
Compiler Backend
Assembler Code
a)
Intermediate Representation
Extraction of Configurations
Generation or Estimation of Profiling Information
b)

Figure 13. Compiler technology for reconfigurable structure inside microcontroller core
a) simplified normal control flow b) extension for integrating configurable instructions

Obviously this is a complex task, and the optimum results is hard to find and will depend on runtime values (e.g. input data). The extracted configurations will be a part of the data base for code generation, and the compiler backend depends on this added information.

4.4 The Classification System

As shown before, reconfigurable structures inside a microcontroller may be located in the input/output area, as an add-on in the near of the processor core or directly inside the core. This is the main classification characteristic.

The other characteristic feature is the operational granularity. This was already used for classifying FPGAs (Field-Programmable Gate Arrays) from CPLDs (Complex Programmable Logic Devices) in the PLD-related world, but this time the granularity differs even more.

We can separate bit-grained from word-grained architectures, where word-grained includes mixed architectures called multi-grained. The bit-grained approach uses FPGA-like structures inside the IC. The size of an intrinsic operation is exactly 1 bit, and this makes the configurable architecture very flexible but includes a lot of configuration overhead.

On the other side, the word-granularity defines an intrinsic size of operations, e.g. 16 or 32 bit. This feature makes operations of word size faster, omits most of the configuration overhead but is limited in its flexibility. These structures are called FPFA, Field-Programmable Functional Arrays.

Figure 14 shows the final classification using these two characteristics, the coupling to the processor core and the granularity. Neglecting the granularity, this leads to three classes of reconfigurable microcontroller (parts): cASIP (configurable Application-Specific Instruction Set Processor), CCM (Customised Computing Machine) and cIOP (configurable Input/Output Processor).

	Core-embedded	Co-processor approach	I/O processor
Bit-grained granularity	Augmented FPGA (cASIP)	Embedded FPGA (CCM)	Embedded FPGA (cIOP)
Word- or Multi-grained granularity	Augmented FPFA (cASIP)	Embedded FPFA (CCM)	Embedded FPFA (cIOP)

Figure 14. Classification of configurable microcontroller

The cASIP class enables primarily performance enhancements, because operations may be tailored to the application-specific requirements. On the other side, CCMs may also enhance performance, but this time, complete application parts must be mapped to the application-specific co-processor. If these mapped parts are exclusive to a specific task, this task can be made real-time capable too.

This is the flexibility of the CCM approach. If application parts can be separated and completely mapped to the CCM, both performance and real-time behaviour can be supported.

The cIOP approach is similar to the CCM with the extension that peripheral elements are directly supported. From the architectural viewpoint, CCM and cIOP are sometimes not distinguishable, and only the application decides how to use the resources. A very good use-case is the usage of cIOPs for network or interconnection purposes. This reduces workload from the processor core and enhances the overall system performance as well as the real-time behaviour of the connection.

5 The Actual Market of Reconfigurable Processors

Last not least a look at the market should be part of an classification. In this specific case, off-the-shelf examples for the classes shall be discussed, and therefore this market view is not complete.

5.1 Altera NIOS and Xilinx MicroBlaze

Altera with its NIOS design and Xilinx with MicroBlaze, both are typical softcore implementations: The microcontroller core is defined by a configuration inside the FPGA (Altera APEX, Xilinx Virtex). Thus optimised for direct use, the design is in fact changeable. This enables the developer to integrate more functionality inside the core and to use reconfigurable structures.

Therefore both architectures can be mapped to all classes: Without core design change, the ICs may be used as CCM and/or cIOP, with change the ICs are part of the cASIP-class. In either case, the FPGAs provide bit-grained operations.

5.2 Atmel AT94k

Figure 15 shows the coupling of the microprocessor core and the embedded FPGA inside the AT94k-architecture from Atmel Inc. This early example of coupling both kinds of computing machines on one die is a typical FPGA-based CCM or cIOP.

The coupling of microcontroller and embedded FPGA is done via 16 addresses, an 8 bit data bus and some control lines. The usage of the FPGA part is not limited, it may be programmed for core applications (CCM) as well as peripheral tasks (cIOP).

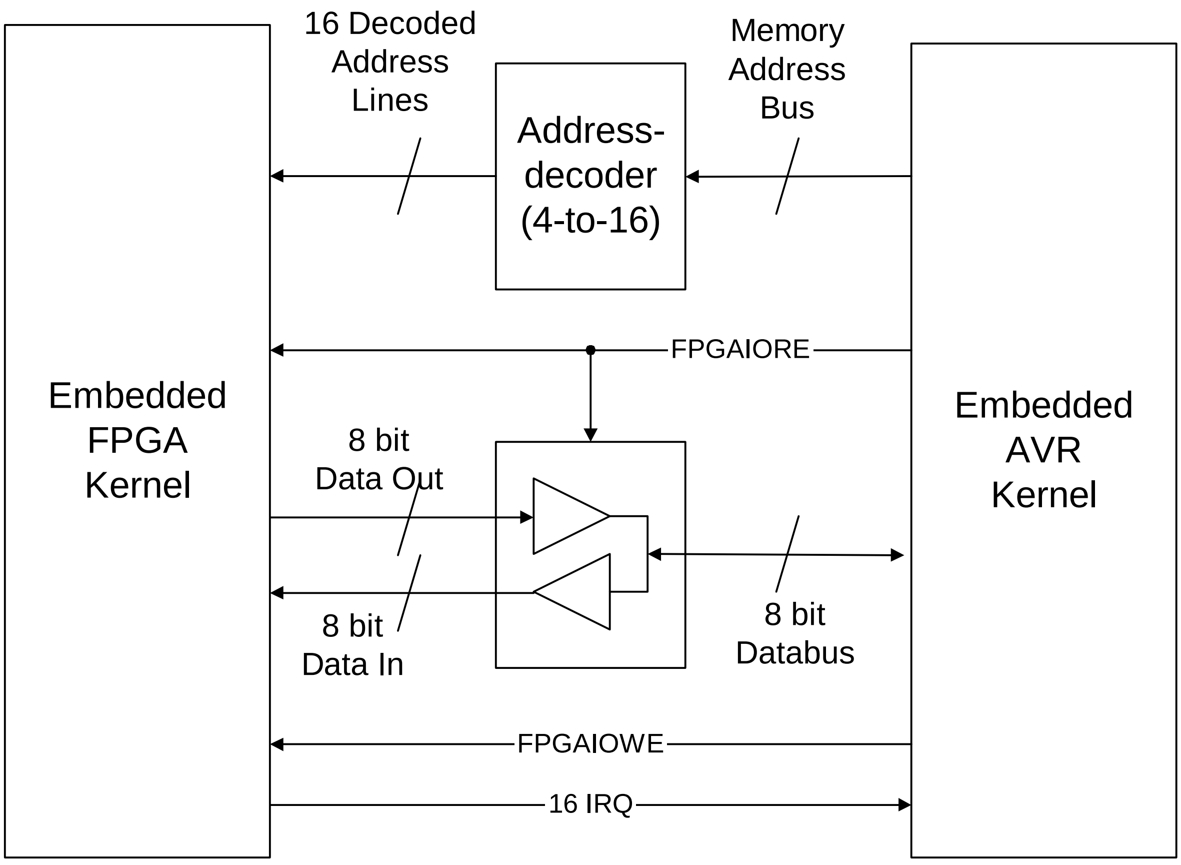

Figure 15. Coupling of microcontroller core (AVR) and embedded FPGA inside AT94k

5.3 PACT XPP

The eXtreme Processing Platform XPP of PACT Corp. is known as high performance and highest performance/power ratio architecture. Figure 16 [1] [8] shows the overall structure of the architecture with a supervisor configuration manager (SCM), one configuration manager (CM) per block and processing array cluster (PAC). While the configuration manager are implemented on microcontroller core, the PAC is in fact a reconfigurable structure.

Figure 17 visualises the above mentioned reconfigurable structure: The PAC consists of some processing array elements (PAE) with storing, routing and computational capacities. The exact amount of PAEs per PAC depends on the actual architecture, but 48 to 64 are good values.

The computational capacity per PAE is located in the ALU object providing operations at word level (e.g. 24 or 32 bit operational size). Forward register (FREG) and backward register (BREG) provide storage capacities as well as vertical routing resources for a height of 1 cell, while horizontal resources and implemented as segmented routing lines for the complete width of the PAE.

The XPP was developed as co-processor and is therefore classified as CCM or embedded FPFA. The CCM approach does not mean that stand-alone application are not possible. If the application is completely data-flow dominated, it could be a good solution to use this architecture stand-alone. Nevertheless, the SCM and CMs must be used for management inside, but this is not the processor part in an application.

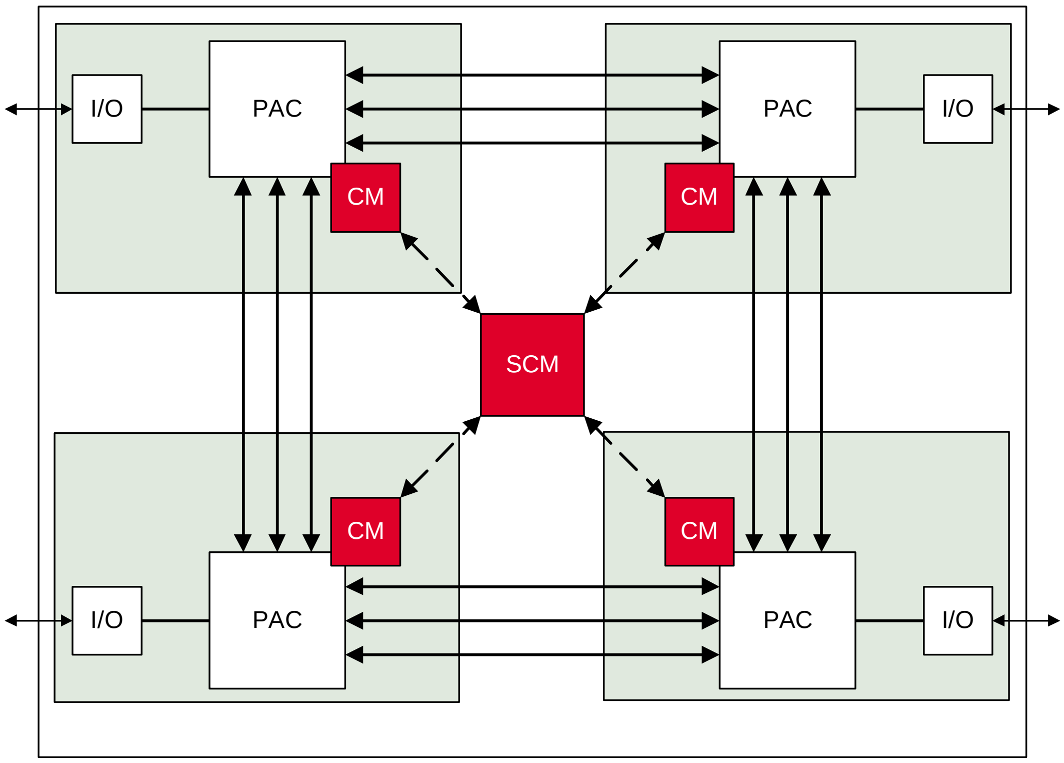

Figure 16. XPP (extreme Processing Platform) of PACT

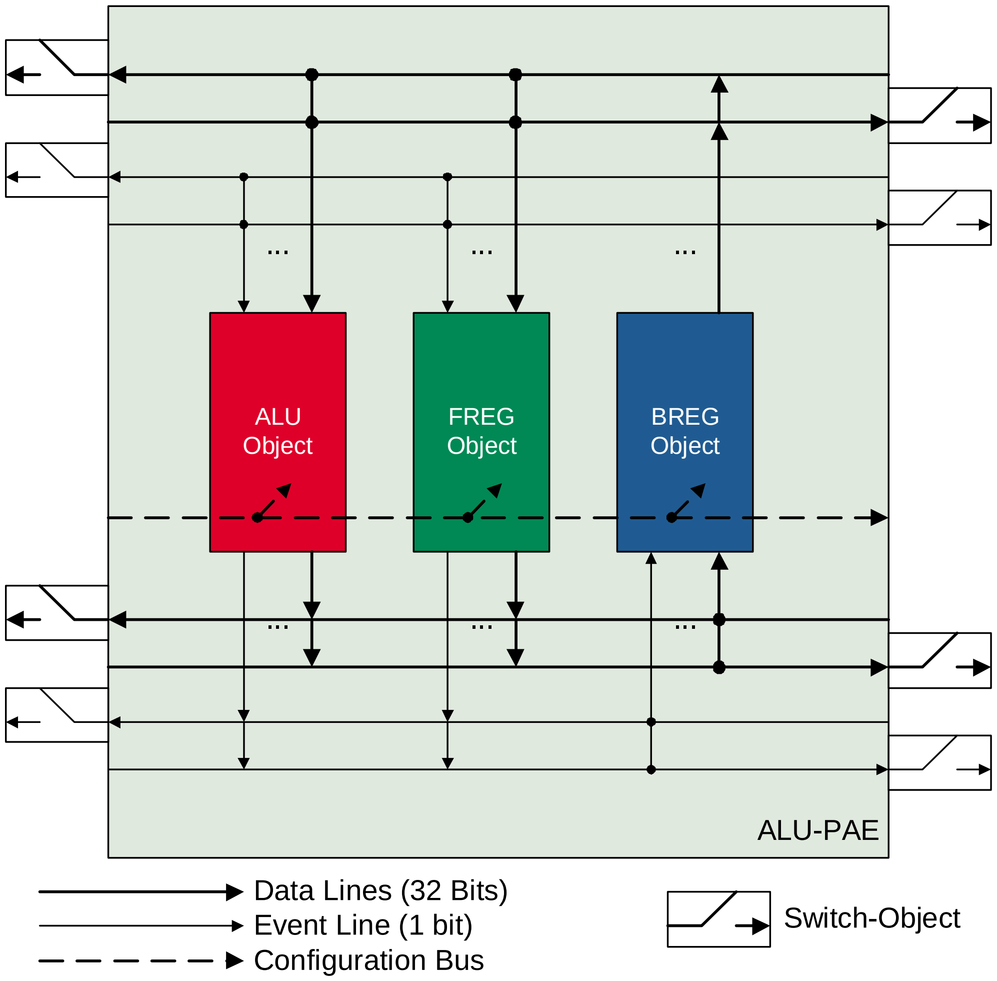

Figure 17. Processing Array Element

5.4 *Tensilica Xtensa*

Tensilica Corp. calls its Xtensa processor core *a configurable, extensible and synthesizable processor core* [9]. This means that the system or hardware designer can configure the processor core or even extend it, not the application software designer. All configuration or extension is performed at design time.

But beside this, the Xtensa architecture has everything a cASIP needs. Figure 18 shows a simplified version of the Xtensa architecture omitting everything from data and instruction caches, internal interfaces and exception handling. It is limited to the very internal part of the core.

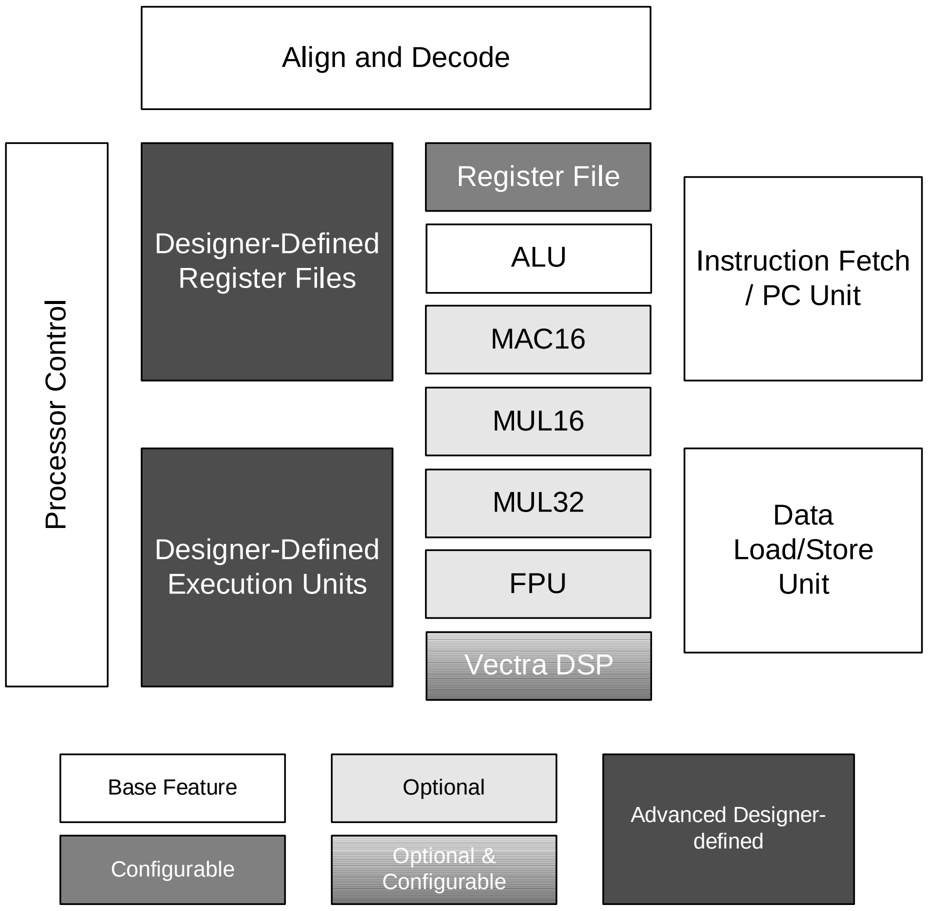

Figure 18. Simplified Xtensa architecture [9]

The base features contain the fetch unit, the data load/store unit, align and decode and the processor control unit. The execution unit itself contains just the ALU as base feature, even the register file is configurable in the sense that it is parameterised by the design engineer.

Optional parts can be included or not, while optional and configurable parts can be included or not, and if, they must be additional parameterised. This design work is done by choices and parameters, and the corresponding language is TIE, Tensilica Instruction Extension language, which is verilog-like. The most interesting part are the advanced designer-defined regions, which must be defined explicitly using TIE.

The software interface contains the definition of instruction mnemonics, operands, encoding and semantics. To obtain the best possible performance, the software interface includes a simulator and profiler to identify the 'hot spots' of the program.

Tensilica's Xtensa is not reconfigurable in the sense of this paper, because this is not done during runtime but synthesize (design) time. But the Xtensa architecture and software interface shows the way future products could proceed: Using reconfigurable elements instead of advanced designer-defined areas but leaving the rest of the software including the C-compiler will be a good system approach, if the reconfigurable parts fit the user's demands of the future.

References

[1] Christian Siemers, ”Configurable Computing – Ansätze, Chancen und Herausforderungen“. Tagungsband Embedded World 2003, Nürnberg, Februar 2003, S. 631–648 (in German language).

[2] André Dehon, John Wawrzynek, ”Reconfigurable Computing: What, Why and Implications for Design Automation”. Design Automation Conference DAC99, San Francisco, 1999.

[3] Frank Vahid, “The Softening of Hardware”. IEEE Computer 36(4), pp. 27–34 (2003).

[4] Reiner W. Hartenstein, Jürgen Becker and Rainer Kress, ”Custom Computing Machines vs. Hardware/Software Co-Design: From a Globalized Point of View”. 6th Int. Workshop on Field Programmable Logic and Appl., FPL'96, Darmstadt, Sept. 1996, Lecture Notes in Computer Science 1142, Springer 1996.

[5] Christian Siemers, Volker Winterstein, ”The Universal Configurable Block/Machine – An Approach for a Configurable SoC-Architecture”. The Journal of Supercomputing 26, pp.309-331 (2003).

[6] Sascha Wennekers, Christian Siemers, ”Reconfigurable RISC – a New Approach for Space-efficient Superscalar Microprocessor Architecture” in: Proceedings of the International Conference on Architecture of Computing Systems ARCS 2002, Karlsruhe, Germany, April 2002. Springer Lecture Notes in Computer Science 2299, pp. 165–178.

[7] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, “Compilers – Principles, Techniques and Tools”. Addison-Wesley Publishing Company, Reading, Massachusetts.

[8] Christian Siemers, ”Rechenfabrik: Ansätze für extrem parallele Prozessoren”. c‘t 2001, H. 15, S. 170 .. 179 (in German language).

[9] http://www.tensilica.com