Architecture, Memory and Interface Technology Integration of an Industrial/ Academic Configurable System-on-Chip (CSoC)

Jürgen Becker

Universitaet Karlsruhe (TH)
Institut fuer Technik der Informationsverarbeitung
D-76128 Karlsruhe, Germany
becker@itiv.uni-karlsruhe.de

Martin Vorbach

PACT XPP Technologies AG
Muthmannstr. 1
D-80939 Munich, Germany
martin.vorbach@pactcorp.com

Abstract

This paper describes the actual status and results of a dynamically Configurable System-on-Chip (CSoC) integration, consisting of a SPARC-compatible LEON processor-core, a commercial coarse-grain XPP-array of suitable size from PACT XPP Technologies AG, and application-tailored global/local memory topology with efficient Amba-based communication interfaces. The given adaptive architecture is synthesized within an industrial/academic SoC project onto 0.18 and 0.13 mm UMC CMOS technologies at Universitaet Karlsruhe (TH). Due to exponential increasing CMOS mask costs, essential aspects for the industry are now adaptivity of SoCs, which can be realized by integrating reconfigurable re-usable hardware parts on different granularities into Configurable Systems-on-Chip (CSoCs).

1 Introduction and Motivation

Systems-on-Chip (SoCs) has become reality now, driven by fast development of CMOS VLSI technologies. Complex system integration onto one single die introduce a set of various challenges and perspectives for industrial and academic institutions. Important issues to be addressed here are cost-effective technologies, efficient and application-tailored hardware/software architectures, and corresponding IP-based EDA methods. Due to exponential increasing CMOS mask costs, essential aspects for the industry are now flexibility and adaptivity of SoCs. Thus, in addition to ASIC-based, one new promising type of SoC architecture template is recognized by several academic[2] [16] [17] [18] [19] [20] and first commercial versions [4] [5] [6] [8] [10] [11] [13]: Configurable SoCs (CSoCs), consisting of processor-, memory-, probably ASIC-cores, and on-chip reconfigurable hardware parts for customization to a particular application. CSoCs combine the advantages of both: ASIC-based SoCs and multichip-board development using standard components [3].

This contribution provides the academic case study results of a CSoC project, integrating the dynamically reconfigurable eXtreme Processing Platform (XPP) from PACT [10] [11], [12] (see figure 1). The XPP architecture realizes a new runtime re-configurable data processing technology that replaces the concept of instruction sequencing by configuration sequencing with high performance application areas envisioned from embedded signal processing to co-processing in different DSP-like application environments.

The adaptive reconfigurable data processing architecture consist of following components:

- Processing Array Elements (PAEs), organized as Processing Arrays (PAs),
- a packet oriented communication network,
- a hierarchical Configuration Manager (CM) tree, and
- a set of I/O modules.

This supports the execution of multiple data flow applications running in parallel. A PA together with one low level CM is referred as PAC (Processing Array Cluster). The low level CM is responsible for writing configuration data into the configurable objects of the PA. Typically, more than one PAC is used to build a complete XPP device. Doing so, additional CMs are introduced for configuration data handling. With an increasing number of PACs on a device, the configuration hardware assumes the structure of a tree of CMs. The root CM of the tree is called the supervising CM or SCM. This unit is usually connected to an external or global RAM.

The basic concept consists of replacing the Von-Neumann instruction stream by automatic configuration sequencing and by processing data streams instead of single machine words, similar to [1] (see figure 1). Due to the XPP´s high regularity, a high level compiler can extract instruction level parallelism and pipelining that is implicitly contained in algorithms [12]. The XPP can be used in several fields, e.g. as image/video processing, encryption, and baseband processing of next generation wireless standards. 3G systems, i.e. based on the UMTS standard, will be defined to provide a transmission scheme which is highly flexible and adaptable to new services. Relative to GSM, UMTS and IS-95 will require intensive layer 1 related operations, which cannot be performed on today´s processors [14] [15]. Thus, an optimized HW/SW partitioning of these computation-intensive tasks is necessary, whereas the flexibility to adapt to changing standards and different operation modes (dif-

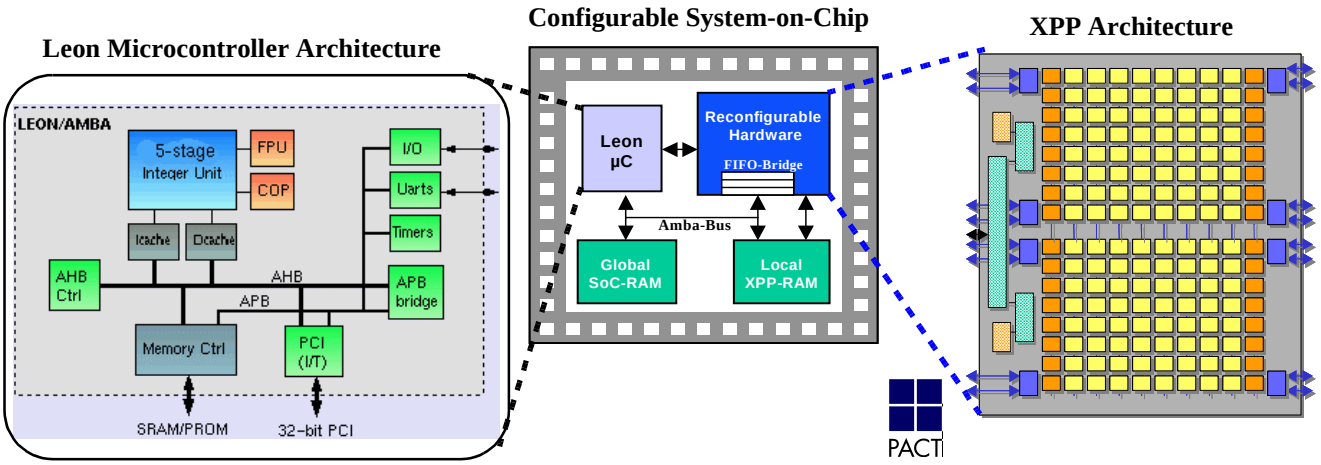

Figure 1: XPP-/Leon-based CSoC Architecture

ferent services, QoS, BER, etc.) has to be considered. Therefore, selected computation-intensive signal processing tasks have to be migrated from software to hardware implementation, e. g. to ASIC or coarse-grain reconfigurable hardware parts, like the XPP architecture.

2 XPP-based CSoC Architecture

Our CSoC architecture (figure 2) consists of an XPP-core from PACT, one LEON µcontroller, and several SRAM-type memory modules. The main communication bus is chosen to be the AHB from ARM [23]. The size of the XPP architecture will be either 16 ALU-PAEs (4x4-array), or 64 ALU-PAEs (8x8-array), dependent on the application field. To get an efficient coupling of the XPP architecture to AHB, we design an AHB-bridge which connects both IO-interfaces on one side of the XPP, input and output interfaces, to the AHB via one module.The AHB specification grants only communication between masters and slaves. There is no option provided for communication between two homogeneous partners, e.g. master to master. Usually the main controller, a processor or a µcontroller, on an AHB-based SoC is master, the RAM as a passive component on the bus is designed as a slave. Thus, if we choice our XPP-AHB-bridge to be a slave, there is no possibility for a connection between XPP and a RAM-module. Otherwise if we choice our bridge to be a master, no communication between the main µcontroller and XPP were allowed. Therefore we choice an unusual method and combine two ports, one master and one slave port as a dual port in the same bridge. This combination allows us to be flexible enough to process various application scenarios. In this way the XPP is able to handle the data from a RAM- module or gets a stream from another master on the CSoC. The CM unit implements a separately memory for faster storing and loading the XPP configurations. If there isn't enough memory space for storing the configurations in local memory, its possible to use the global CSoC memory to do that. The AHB-bridge for CM will be a single ported SLAVE-AHB-bridge. The transfers of the configurations from global memory to the Configuration Manager will be done by LEON. Therefore the CM have to send a request to LEON and start new configuration transfer.

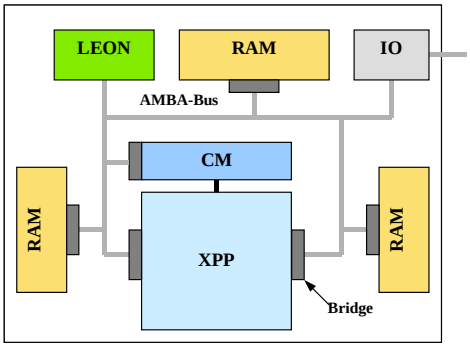

Figure 2: CSoC RAM Topology

The µcontroller on our CSoC is a LEON processor. This processor is a public domain IP core. The LEON VHDL model implements a 32-bit processor conforming to the SPARC V8 architecture. It is designed for embedded applications with the following features on-chip: separate instruction and data caches, hardware multiplier and divider, interrupt controller, two 24-bit timers, two UARTs, power-down function, watchdog, 16-bit I/O port and a flexible memory controller. Additional modules can easily be added using the on-chip AMBA AHB/APB buses. The VHDL model is fully synthesisable with most synthesis tools and can be implemented on both FPGAs and ASICs. The LEON µprocessor acts as a master on our CSoC-architecture. The program data for LEON will be transferred via AHB. In this manner there are two options where the main memory for LEON could be located: intern on die or extern on separate modules.

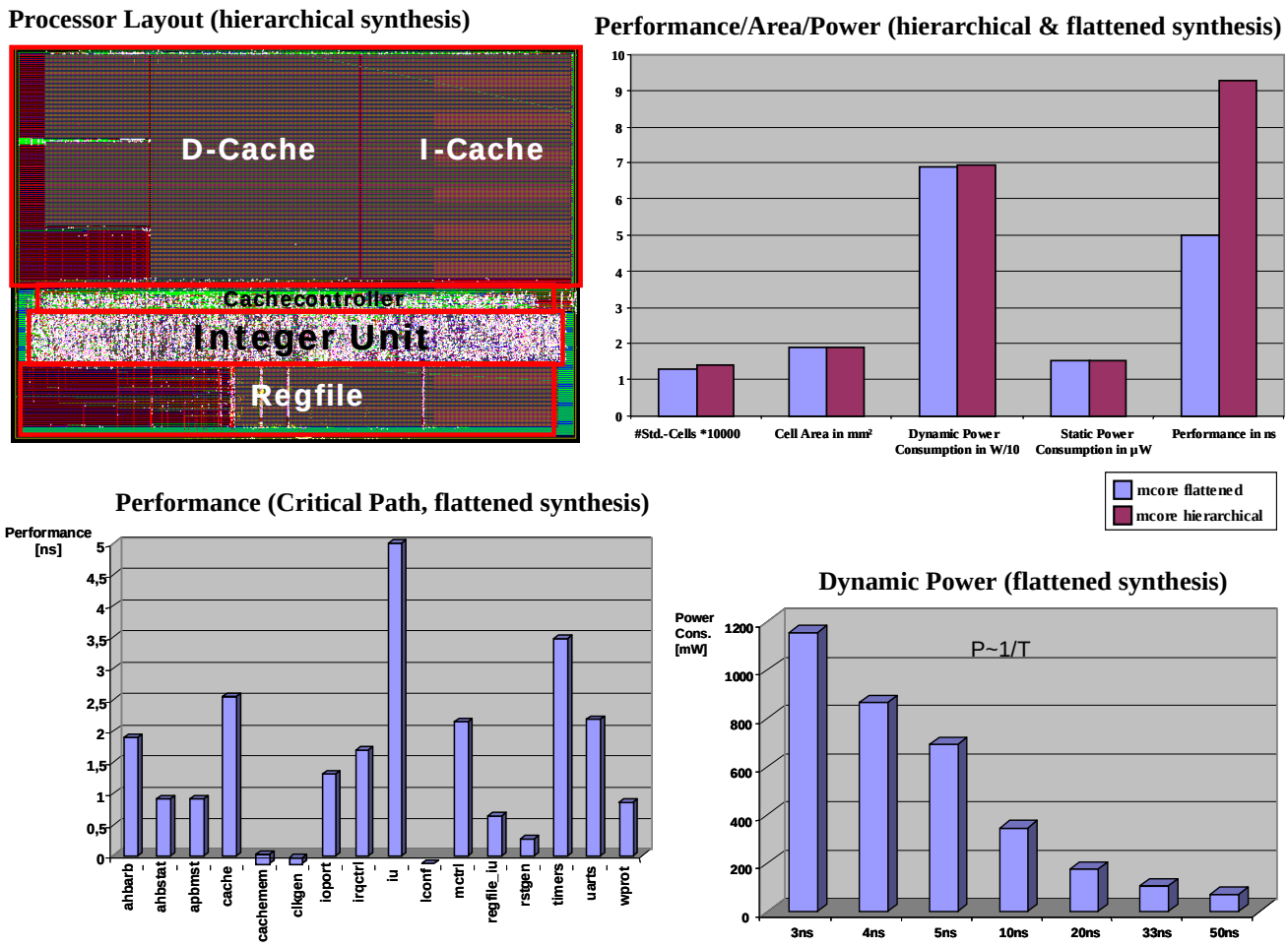

Figure 3: LEON RISC Processor Standard Cell Synthesis obtained at Universitaet Karlsruhe (TH)

The local memory module on CSoC is used to store the LEON programs, data for XPP computation and XPP configurations. The theoretical bandwidth of AHB at 100 Mhz and 32bit bit width is 400 MBytes/sec. That's enough to serve the XPP-architecture with data and configurations and to handle the program data of the LEON efficiently. The interface of the memory module to the AHB is realized as a slave. That's because this module is a passive module only and can not start any kind of transactions on the AHB. Moreover, there will be an external RAM interface implemented, which allows to connect extern memory to the CSoC. This module is a part of the LEON IP-core.

The prior AHB specification [23] from ARM allows only one transaction per cycle. That means that if one master and one slave are communicating at a time step, the other modules on the bus have to wait till this communication is done. This kind of transactions block completely the whole bus. The solution for this restriction is the multi-layer AHB. This concept allows multiple transactions at the same time. Instead of using simple multiplexers and decoders on the bus, we use now single decoders and multiplexers per slave to choice the right master for communication. Thus, the bus is divided in several sub-busses allowing simultaneous transactions between various masters and slaves. Each of the 4 parallel operating high-throughput bridges connecting the SRAM-banks to the XPP can achieve a data throughput of of 400 MB/sec operating in 100 Mhz, e.g. a complete on-Chip data throughput of 1600 MB/sec, which is sufficient for multimedia-based applications like MPEG-4 algorithms applied to video data in PAL-standard format (see section 4). The three SRAM memory modules provide up to 3 MB on-chip. The complete size is splitted into 2x1,2 MB and 1x0,6 MB modules, for storing two pictures and code with XPP configurations. Within this flexible multi-layer AHB interface concept the XPP can operate either as slave (Leon processor is master) or as master itself. The communication between the CSoC and the outside world will be realised throught a master/slave AHB/PCI host bridge. The AHB master ability admits the direct transfers from PCI to internal RAM without involvement of the main µcontroller, or, the slave ability admits transfers between all masters and PCI-Bridge.

3 eXtreme Processing Platform - XPP

The XPP architecture is based on a hierarchical array of coarse-grain, adaptive computing elements called *Processing Array Elements (PAEs)* and a *packet-oriented communication network*. The strength of the XPP technology originates from the combination of array processing with unique, powerful run-time reconfiguration mechanisms. Since configuration control is distributed over several *Configuration Managers (CMs)* embedded in the array, PAEs can be configured rapidly in parallel while neighboring PAEs are processing data. Entire applications can be configured and run independently on different parts of the array. Reconfiguration is triggered externally or even by special event signals originating within the array, enabling self-reconfiguring designs. By utilizing protocols implemented in hardware, data and event packets are used to process, generate, decompose and merge streams of data. The XPP has some similarities with other coarse-grain reconfigurable architectures like the KressArray [21] or Raw Machines [22] which are specifically for stream-based applications. XPP's main distinguishing features are its automatic packet-handling mechanisms and sophisticated hierarchical configuration protocols.

3.1 Array Structure

An XPP device contains one or several *Processing Array Clusters (PACs)*, i.e. rectangular blocks of PAEs. Each PAC is attached to a CM responsible for writing configuration data into the configurable objects of the PAC. Multi-PAC devices contain additional CMs for configuration data handling, forming a hierarchical tree of CMs. The root CM is called the supervising CM or SCM. The XPP architecture is also designed for cascading multiple devices in a multi-chip. A CM consists of a state machine and internal RAM for configuration caching. The PAC itself contains a configuration bus which connects the CM with PAEs and other configurable objects. Horizontal busses carry data and events. They can be segmented by configurable switch-objects, and connected to PAEs and special I/O objects at the periphery of the device.

A PAE is a collection of PAE objects. The typical PAE shown in figure 4 contains a BREG object (back registers) and an FREG object (forward registers) which are used for vertical routing, as well as an ALU object which performs the actual computations. The ALU object's internal structure is shown on the bottom left-hand side of the figure. The ALU implemented performs common fixed-point arithmetical and logical operations as well as several special three-input opcodes like multiply-add, sort, and counters. Events generated by ALU objects depend on ALU results or exceptions, very similar to the state flags of a classical microprocessor. A counter, e.g., generates a special event only after it has terminated. The next section explains how these events are used. Another PAE object implemented in the prototype is a memory object which can be used in FIFO mode or as RAM for lookup tables, intermediate results etc. However, any PAE object functionality can be included in the XPP architecture.

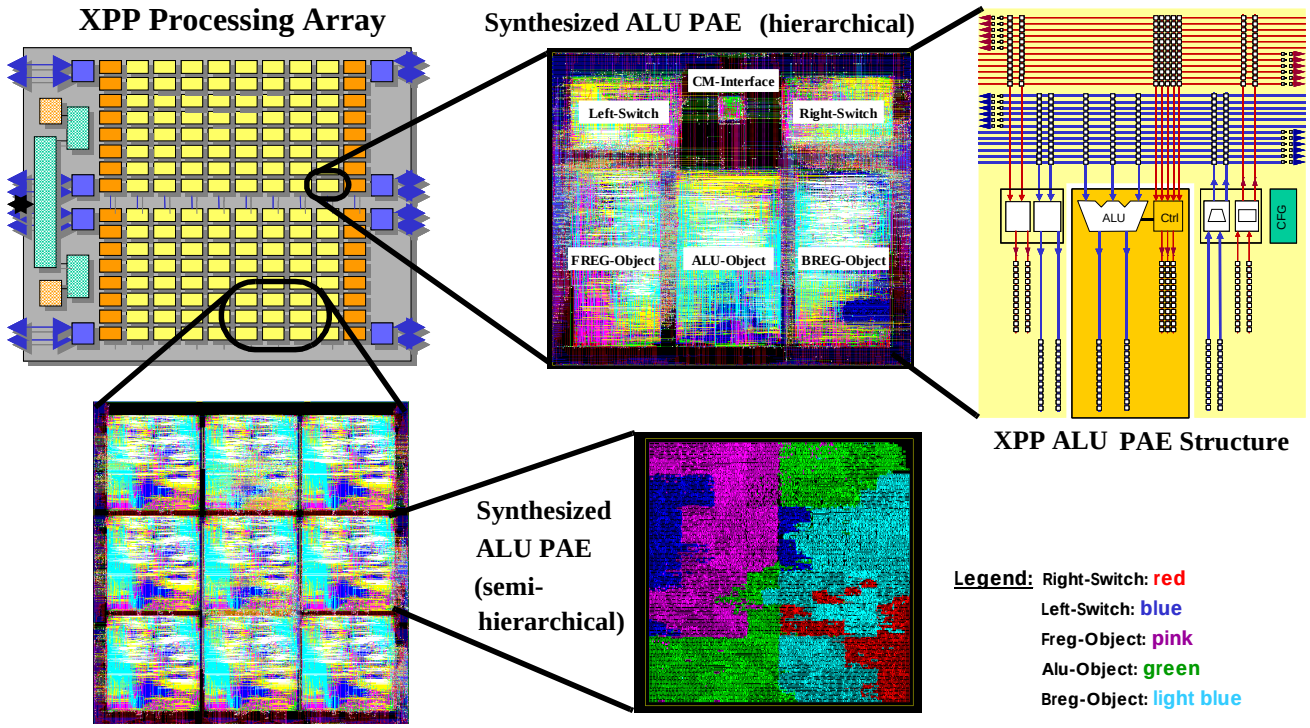

Figure 4: XPP ALU Structure and Standard Cell Synthesis Layout from Universitaet Karlsruhe (TH)

3.2 Packet Handling and Synchronization

PAE objects as defined above communicate via a packet-oriented network. Two types of packets are sent through the array: data packets and event packets. Data packets have a uniform bit width specific to the device type.
In normal operation mode, PAE objects are self-synchronizing. An operation is performed as soon as all necessary data input packets are available. The results are forwarded as soon as they are available, provided the previous results have been consumed. Thus it is possible to map a signal-flow graph directly to ALU objects, and to pipeline input data streams through it. The communication system is designed to transmit one packet per cycle. Hardware protocols ensure that no packets are lost, even in the case of pipeline stalls or during the configuration process. This simplifies application development considerably. No explicit scheduling of operations is required. Event packets are one bit wide. They transmit state information which controls ALU execution and packet generation. For instance, they can be used to control the merging of data-streams or to deliberately discard data packets. Thus conditional computations depending on the results of earlier ALU operations are feasible. Events can even trigger a self-reconfiguration of the device as explained below.

3.3 Configuration

The XPP architecture is optimized for rapid and user transparent configuration. For this purpose, the configuration managers in the CM tree operate independently, and therefore are able to configure their respective parts of the array in parallel. Every PAE stores locally its current configuration state, i.e. if it is part of a configuration or not (states „configured“'or „free“). If a configuration is requested by the supervising CM, the configuration data traverses the hierarchical CM tree to the leaf CMs which load the configurations onto the array. The leaf CM locally synchronizes with the PAEs in the PAC it configures. Once a PAE is configured, it changes its state to „configured“. This prevents the respective CM from reconfiguring a PAE which is still used by another application. The CM caches the configuration data in its internal RAM until the required PAEs become available. Hence the CMs' cache memory and the distributed configuration state in the array enables the leaf CMs to configure their respective PACs independently. No global synchronization is necessary.
While loading a configuration, all PAEs start to compute their part of the application as soon as they are in state „configured“. Partially configured applications are able to process data without loss of packets. This concurrency of configuration and computation hides configuration latency. Additionally, a prefetching mechanism is used. After a configuration is loaded onto the array, the next configuration may already be requested and cached in the low-level CMs' internal RAM. Thus it need not be requested all the way from the SCM down to the array when PAEs become available.

3.4 CSoC Standard Cell Synthesis

The LEON processor architecture, illustrated in figure 1, has been first synthesized onto UMC 0.18 µm technology with SYNOPSYS and CADENCE (Silicon Ebsemble) tools, available through the european joined academic/ industry project EUROPRACTICE. The LEON processor core needs in 0.18 µm CMOS technology approx. 1.8 mm^2 and can be clocked up to 200 Mhz. The LEON layout, obtained at Universitaet Karlsruhe (TH), with area/performance and dynamic power consumption results are shown in figure 3, whereas the synthesis was done hierarchically (synthesized netlists and place&route performed separately for different LEON modules) and completely flattened. The major advantage in using flattened synthesis was the better performance for the critical path of LEON in its Integer Unit (see 5 ns cycle rate for IU in). First non-optimized area/ performance synthesis results in 0.13 µm UMC CMOS technology needs 0.7 mm^2 and can be clocked up to 300 Mhz. corresponding.
XPP ALU-PAEs have been synthesized in 0.13 µm CMOS technology in different synthesis strategies (with CADENCE SE): hierarchical, semi-hierarchical, and flattened. The semi-hierarchical strategy gives the best performance/area trade-offs, still allowing a parametrizable modular design flow (see figure 4).

4 Application Examples Performance

PACT did in the year 2000 a first evaluation board based on 0.25 µm technology for their XPP 128 chips. Based thereupon, the promising performance results in figure 5 compared to a parallel VLIW type DSP of Texas Instruments are obtained [10], [11]. The dhrystone 2.1 benchmark reports 1,400 iteration/s/MHz for LEON by using 4K + 4K caches and a 16x16 multiplier, e.g. ARM9TDMI reaches 1,200 iterations/s/MHz. This academic study of LEON and XPP with efficient RAM-topology promises a high boost in performance and flexibility. First digital TV application performance results were obtained by evaluating corresponding MPEG-4 algorithm mappings onto the introduced LEON/XPP CSoC and based on the UMC 0.13 µm CMOS technology synthesis results. Based on this

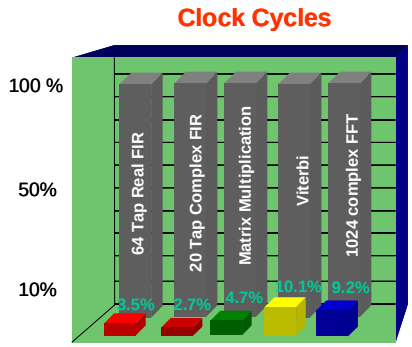

Figure 5: Performance: TI C6203 vs. PACT XPP 128

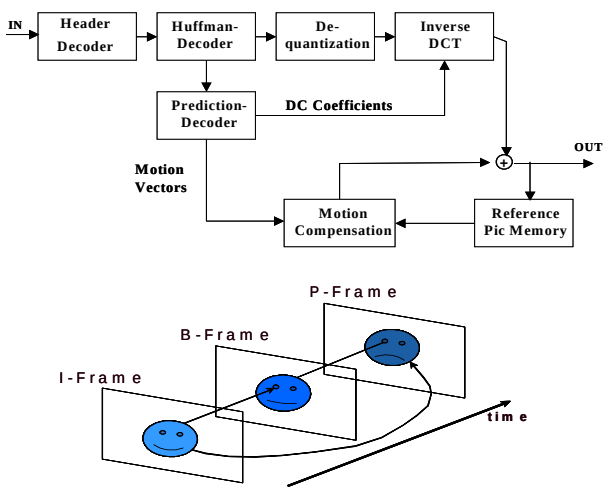

Figure 6: Main MPEG-4 Algorithm Modules

coarse-grain CSoC version, performance/cost results of an MPEG-4 application have been analyzed, whereas the Inverse DCT (see figure 6) applied to 8x8 pixel blocks can be performed by an 4x4 XPP-Array in 74 clock cycles. Since the IDCT is one of the most complex operations in MPEG-4 algorithms, the preliminary clock frequency of 100 Mhz based on UMC 0.13 µm CMOS technology seems to be more than sufficient for this real-time digital TV application scenario.

5 Conclusions

This paper introduced the status and first performance/area results of an academic case study integrating an industrial dynamically Configurable System-on-Chip (CSoC), consisting of a LEON RISC processor core, a PACT XPP-array of suitable size (4x4 or 8x8 ALU PAEs) and efficient global/local memory topologies with efficient multi-layer Amba-based communication interfaces. The adaptivity and multi-purpose usability of CSoCs promises an attractive potential for embedded system industry in different application areas, e.g. (wireless) communication (-> multi-standard, different bandwidth and services), automotive (multi-purpose architecture platforms for all kind of control and multi-media in cars), etc. The exponential increasing of CMOS mask costs demands urgently such adaptivity, which can be realized by integrating reconfigurable reusable silicon parts on multiple granularities into CSoCs, demonstrating attractive perspectives, especially with short time-to-market (risk minimization), flexibility (adaptivity) and cost (multi-purpose -> volume increase) constraints.

6 References

[1] R. W. Hartenstein, J. Becker et al.: A Novel Machine Paradigm to Accelerate Scientific Computing; Special issue on Scientific Computing of Computer Science and Informatics Journal, Computer Society of India, 1996.

[2] J. Becker, T. Pionteck, C. Habermann, M. Glesner: Design and Implementation of a Coarse-Grained Dynamically Reconfigurable Hardware Architecture; in: Proc. of IEEE Computer Society Annual Workshop on VLSI (WVLSI 2001), Orlando, Florida, USA, April 19-20, 2001

[3] J. Becker (Invited Tutorial): Configurable Systems-on-Chip (CSoC); in: Proc. of 9th Proc. of XV Brazilian Symposium on Integrated Circuit Design (SBCCI 2002), Porto Alegre, Brazil, September 5-9, 2002

[4] Xilinx Corp.: http://www.xilinx.com/products/virtex.htm.

[5] Altera Corp.: http://www.altera.com

[6] Triscend Inc.: http://www.triscend.com

[7] Triscend A7 Configurable System-on-Chip Platform - Data Sheet http://www.triscend.com/products/dsa7csoc_summary.pdf

[8] LucentWeb] http://www.lucent.com/micro/fpga/

[9] Atmel Corp.: http://www.atmel.com

[10] PACT Corporation: http://www.pactcorp.com

[11] The XPP Communication System, PACT Corporation, Technical Report 15, 2000

[12] V. Baumgarte, F. Mayr, A. Nückel, M. Vorbach, M. Weinhardt: PACT XPP - A Self-Reconfigurable Data Processing Architecture; The 1st Int´l. Conference of Engineering of Reconfigurable Systems and Algorithms (ERSA´01), Las Vegas, NV, June 2001

[13] Hitachi Semiconductor: http://semiconductor.hitachi.com/news/triscend.html

[14] Peter Jung, Joerg Plechinger., “M-GOLD: a multimode basband platform for future mobile terminals”,CTMC'99, IEEE International Conference on Communications, Vancouver, June 1999.

[15] Jan M. Rabaey: System Design at Universities: Experiences and Challenges; IEEE Computer Society International Conference on Microelectronic Systems Education (MSE´99), July 19-21, Arlington VA, USA

[16] S. Copen Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor, R. Laufer "PipeRench: a Coprocessor for Streaming Multimedia Acceleration" in ISCA 1999. http://www.ece.cmu.edu/research/piperench/

[17] MIT Reinventing Computing: http://www.ai.mit.edu/projects/transit dpga_prototype_documents.html

[18] N. Bagherzadeh, F. J. Kurdahi, H. Singh, G. Lu, M. Lee: "Design and Implementation of the MorphoSys Reconfigurable Computing Processor "; J. of VLSI and Signal Processing-Systems for Signal, Image and Video Technology, 3/ 2000

[19] Hui Zhang, Vandana Prabhu, Varghese George, Marlene Wan, Martin Benes, Arthur Abnous, "A 1V Heterogeneous Reconfigurable Processor IC for Baseband Wireless Applications", Proc. of ISSCC2000.

[20] Pleiades Group: http://bwrc.eecs.berkeley.edu/Research/Configurable_Architectures/

[21] R. Hartenstein, R. Kress, and H. Reinig. A new FPGA architecture for word-oriented datapaths. In Proc. FPL’94, Prague, Czech Republic, September 1994. Springer LNCS 849.

[22] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, and P. Finch. Baring it all to software: Raw machines. IEEE Computer, pages 86-93, September 1997

[23] ARM Corp.: http://www.arm.com/arm/AMBA