Chapter 6
Smart Chips for Smart Surroundings – 4S

Eberhard Schüler, Ralf König, Jürgen Becker, Gerard Rauwerda, Marcel van de Burgwal, and Gerard J.M. Smit

Abstract The overall mission of the 4S project (Smart Chips for Smart Surroundings) was to define and develop efficient flexible, reconfigurable core building blocks, including the supporting tools, for future Ambient System Devices. Reconfigurability offers the needed flexibility and adaptability, it provides the efficiency needed for these systems, it enables systems that can adapt to rapidly changing environmental conditions, it enables communication over heterogeneous wireless networks, and it reduces risks: reconfigurable systems can adapt to standards that may vary from place to place or standards that have changed during and after product development.

In 4S we focused on heterogeneous building blocks such as analogue, hardwired functions, fine and coarse grain reconfigurable tiles and microprocessors. Such a platform can adapt to a wide application space without the need for specialized ASICs. A novel power aware design flow and runtime system was developed. The runtime system decides dynamically about the near-optimal application mapping to the given hardware platform.

The overall concept was verified on hardware platforms based on an existing SoC and in a second step with novel silicon. DRM (Digital Radio Mondiale) and MPEG4 Video applications have been implemented on the platforms demonstrating the adaptability of the 4S concept.

6.1 Project Partners

1. PACT XPP Technologies AG, Germany
2. CTIT, The University of Twente, The Netherlands
3. ITIV, Karlsruhe Institute of Technology KIT, Germany (coordinator)

E. Schüler (✉)
PACT XPP Technologies AG, Walter-Gropius-Str. 15, 80807 Munich, Los Gatos, Germany
e-mail: eberhard.schueler@t-online.de

J.M.P. Cardoso and M. Hübner (eds.), *Reconfigurable Computing: From FPGAs to Hardware/Software Codesign*, DOI 10.1007/978-1-4614-0061-5_6,

4. ATMEL Corporation, Germany
5. IMEC, Inter University Micro-Electronics Centre, Belgium
6. WMC, Twente Institute for Wireless and Mobile Communications B.V., The Netherlands
7. ASICentrum s.r.o., Czech Republic
8. Thales Communications, France
9. dicas, Dicas digital image coding GmbH, Germany
10. Harman/Becker Automotive Systems GmbH, Germany
11. Recore Systems, The Netherlands

- Project Coordinator: Eberhard Schüler, PACT XPP Technologies AG, Germany
- Start Date: January 2004
- End Date: December 2007
- EU Program: 6th Framework Program for Research and Technological Development, Information Society Technologies, Pushing the limits of CMOS and preparing for post-CMOS, Contract IST-1908. Instrument: Specifically Targeted Research Project (STREP)
- Global Budget: 9.7 M€
- Global Funding by EU: 5.4 M€
- Contact Author: Eberhard Schüler, Email: eberhard.schueler@t-online.de

6.2 Introduction

The development of energy efficient computing and software architectures for future Ambient Systems was at the core of the 4S project. Ambient Systems (also known as *ambient intelligence or ubiquitous computing)* are networked embedded systems wirelessly integrated with everyday environments and supporting people in their activities. The architecture of Systems-on-Chips (SoC) architecture suitable for Ambient Devices poses a lot of challenges: these devices have a very small energy budget, they are always operational (although quite often in a low-power mode), are small in size but require high processing performance. State-of-the-art architectures could not provide the processing power required by a fully operational Ambient Device given the tight energy limitations. To realize devices within the energy budget, flexible and highly efficient hardware *and* software architectures are needed. Moreover, without significant energy reduction techniques and energy-efficient adaptive architectures, battery-life constraints will severely limit the capabilities of these devices.

6.2.1 Heterogeneous Reconfigurable Computing

Reconfigurable systems offer the required flexibility and can adapt processing resources dynamically to the demand of applications. We distinguish several processing structures in a heterogeneous reconfigurable system, for example: bit-level

reconfigurable units (e.g. FPGAs), word-level (coarse-grained) reconfigurable units, and general-purpose programmable units (DSPs and microprocessors). Unlike microprocessors and DSPs, reconfigurable units adapt the layout and functionality of hardware elements to the task to be performed. This enables exploiting the implicit parallelism of the algorithm and provides much higher performance for a given energy budget.

The adaptability of the architecture enables the system to be targeted at multiple applications. The architecture and firmware can be upgraded at any time (even when the system is already installed and running) and reconfigurability allows adaptation of the architecture for testing and maintenance purposes during and after the production phase.

Typically, some algorithms are more suitable for bit-level reconfigurable architectures (e.g. Software defined Radio PN-code generation), others for DSP-like hardware and others for word-level reconfigurable platforms (e.g. FIR filters or FFT algorithms). Most stream-based algorithms can be mapped efficiently to word-level reconfigurable architectures [10]. Application designers or, at best, high-level compilers choose the most efficient processing unit for the type of processing needed for a given application task.

This raises the question how to map various application tasks to such a heterogeneous platform and which tools will enable the application designer to benefit from a specific hardware architecture. In 4S we addressed the question by means of a runtime operating system and communication framework as described in subsequent sections.

6.2.2 Energy Efficiency

Perhaps the most significant property of ambient and unobtrusive wireless devices is their resource limitation, in terms of energy, memory and processing power. In 4S we qualified energy as the premium resource, and built time/energy adaptive architectures that adequately operate under these constraints. We applied energy-efficient design principles at all layers, including those where these considerations were not consequently applied in combination in the past. For example: analogue/digital tradeoffs, energy-efficient data processing and communication, run-time reconfiguration for efficiency, as well as high-level tools for (energy-) efficient mapping of applications to dynamic reconfigurable architectures.

Furthermore, heterogeneous reconfigurable Systems on Chip (SoC) can adapt supply voltage and clock frequency per module and so provide additional potential to save power.

6.2.3 Flexibility

Ambient Devices are able to choose from a wide range of services from various wireless access networks in its surrounding, each having their own characteristics, costs, and ownership. They operate in a heterogeneous environment where it is

possible to use a combination of networks (simultaneously), each of which is optimised for some particular service. For example, Ambient Devices will have to support multiple multimedia standards, e.g. different audio and video standards (such as MP3, MPEG4, H.264). In addition, devices should be prepared to perform e.g. speech recognition/generation algorithms, signal processing algorithms for biometrics (e.g. fingerprint recognition) and security algorithms. Finally, the architecture should have the flexibility to anticipate future emerging standards.

6.2.4 Development Tools for Reconfigurable Architectures

Programming heterogeneous reconfigurable SoCs is considered to be non-trivial. Thus software designers must be supported by sophisticated tools. Furthermore, new tools must be accepted by application software designers. In 4S we developed a set of new tools on top of existing well known standards. This strategy not only provides a clearly defined migration path from state-of-the-art tools but also provides flexibility and less risk because only small steps are required to launch a new architecture for commercial products.

6.2.5 Structure of Subsequent Sections

In the following section we first introduce the 4S heterogeneous reconfigurable platform from the hardware and runtime system perspectives. The operating system section introduces the concepts of the runtime framework. Then we describe the demonstration platforms, the realisation of implemented heterogeneous hardware and the silicon implementations. The Proof of Concept section presents experimental results based on two target applications. The 4S chapter is finished with an outlook and the conclusion summarizing the achieved results.

6.3 The 4S Approach

6.3.1 The Essential Ideas

The 4S project targets the system architecture, hardware building blocks and the supporting energy efficient design and runtime tools.

6.3.1.1 Heterogeneous System Architecture

The system architecture consists of a set of *heterogeneous building blocks* like analogue blocks, hardwired ASIC functions, reconfigurable blocks, signal processors

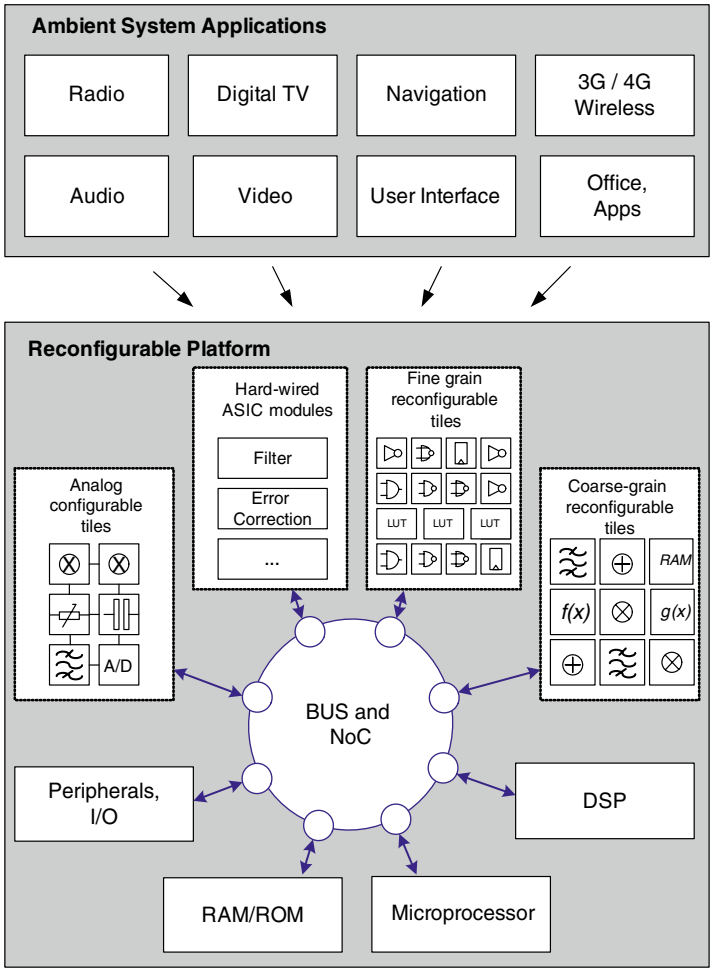

Fig. 6.1 The 4S heterogeneous hardware architecture

and microprocessors. We name the processing building blocks "tiles". The entirety of the building blocks must support a *wide spectrum* of possible applications for Ambient Devices, including wireless communication as well as audio and video processing. As those application fields have different requirements that cannot always be handled efficiently by one single tile, several tiles with different capabilities are included in the architecture.

Figure 6.1 shows the main building blocks of a universal Ambient System. For the specific realisation of an Ambient Device not all of these building blocks might be needed. If – for example – the purpose of a device is the reception of audio streams, no building blocks for video processing are needed in this special device and thus can be omitted in order to save power and silicon area. For another device that is targeted for mobile video reception other building blocks will be included or left out.

In that sense the proposed architecture provides the building blocks needed to support most of the demands of Ambient Systems but the amount of blocks used for an individual product might change from application field to application field. However, within one field of application the platform provides full flexibility.

6.3.1.2 Heterogeneous SoC Infrastructure

A flexible communication infrastructure connects the building blocks. A standard shared bus system (e.g. AMBA) provides interfaces for all heterogeneous blocks. Most of existing and commercially available IP blocks use standard busses. However, a shared bus system is not optimal for high bandwidth data streams which are common in multimedia and wireless applications requiring guaranteed bandwidth. Therefore, we propose a Network-on-Chip (NoC) which unloads busses from high bandwidth and uniform data streams.

6.3.1.3 Power Aware Runtime System

The main task of the runtime system is to map the applications or single functions at a specific point in time on available heterogeneous hardware resources. With existing standard operating systems the mapping to hardware was defined manually during compile time. Thus a specific function can only be executed on the initially planned hardware building block. One of the 4S inventions was the development of a flexible runtime system and spatial mapping tool that allows for dynamic scheduling and mapping of functions to hardware building blocks. The purpose of dynamic spatial mapping is to allow loading of new applications to the platform during runtime and to minimise the overall power consumption of the Ambient System. To fulfil this task the runtime system takes into account user demands, the environment such as current wireless coverage as well as the status of the system such as current battery status.

To support this, a runtime system and the corresponding design time tool flow have been developed in the 4S project. The development environment consists of specific tools for each tile for generation of binary code needed to configure and operate each block. The application software uses the configuration and communication mechanisms of the underlying real-time operating system and the runtime system. On top-level a XML-description specifies and characterizes the applications, the building blocks and the overall system architecture in order to gain decision criteria for the dynamic runtime mapping.

To allow for early design space exploration and concept verification, a co-simulation *framework* was developed. It allows not only to co-simulate already existing building blocks but also to verify newly developed hardware blocks, software drivers and their interaction with other parts of a system.

Figure 6.2 shows the tool flow. At top level the system properties such as the system topology, interconnect and available bandwidth are specified. The second step is describing the application in form of a task graph. The graph's nodes specify

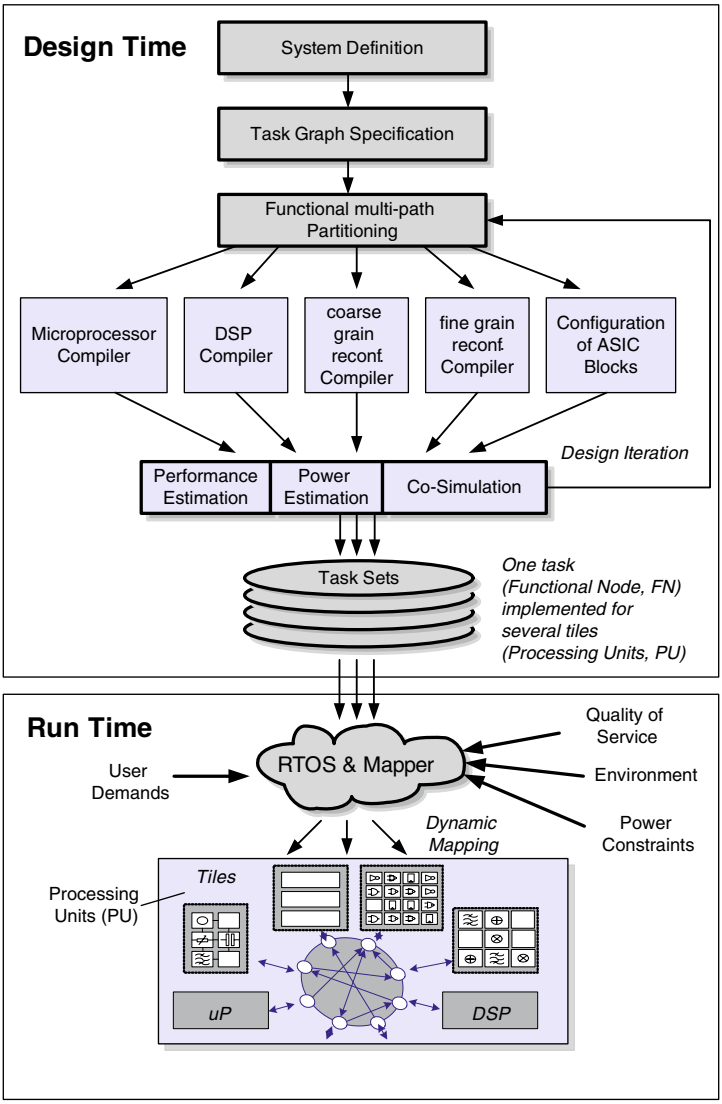

Fig. 6.2 Tools flow

the function to be performed while the edges specify the communication with other tasks. Then the designer has to decide which task should be implemented on which hardware tile. Note that in the context of the operating framework OSYRES as described later, a task is a "Functional Node" (FN) and a hardware tile a "Processing Unit" (PU). Where possible a task is implemented functional identically for several tiles. This allows the runtime system to choose among different realisations according to the actual system status and external requirements. The third step is to implement

and characterize all tasks per tile in terms of energy, processing time and bandwidth requirements. This characterization can be done with RTL design compiler power tools or – preferably – with power estimators as part of the design environment of the specific hardware block. Co-simulation now allows to verify the application and to perform design iterations and optimization where required.

The output of the design-time activities is a database of so-called Task Sets. Each Task Set contains the executables implemented for one or more hardware tiles combined with the performance model and a XML-file describing the realisation and connection of the different implementations.

The runtime system finally uses Task Sets as input. The runtime scheduler continuously analyses the current system state, checks parameters such as requested and running applications, battery status and wireless coverage and then tries to find the most power efficient mapping. The scheduler defines which task should be executed on which hardware tile whenever the system status changes (spatial mapping).

Based on the scheduler's decision, the runtime system loads the tasks to the hardware tiles and starts them. Sometimes it may be required to remove a running task from one tile and to load and re-launch it on another. Since the current application must not be interrupted during re-mapping, special switch points are defined in the application. Switch points should be defined where the application state complexity is small and when buffers have sufficient data to bridge short term intermitted processing, since remapping may take some time.

6.4 Realisation

The 4S concept makes use of several hardware blocks which are capable to perform functionally identical software tasks. To support this in an efficient way existing hardware IP has been newly designed or extended:

- The *Montium* Reconfigurable core
- A NoC routers and AHB interfacing for Montium Tiles
- The *XPP* reconfigurable processor was extended by new instructions and features to improve the area and power efficiency
- The RF front end chip was extended by configurable features
- New ASIC cores for software defined radio (SDR) such as the Digital Down Converter (DDC), the Viterbi Decoder and the Signal Analyser have been implemented.

6.4.1 Montium Reconfigurable Tile

The *Montium* is a coarse-grained reconfigurable processor core and targets the 16-bit DSP algorithm domain. The Montium architecture originates from research by the University of Twente and has been further developed by Recore Systems.

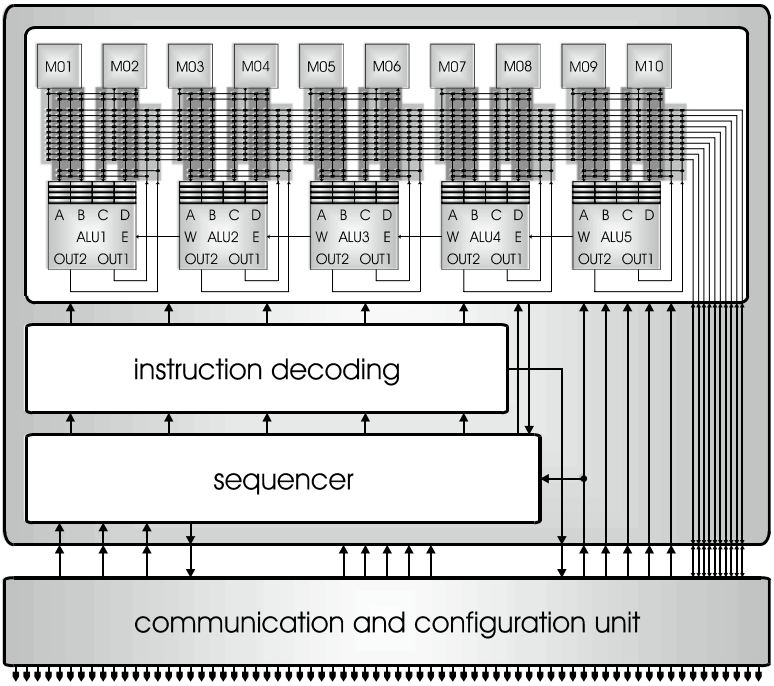

Fig. 6.3 Montium processing tile

A single Montium processing tile is shown in Fig. 6.3. At first glance the Montium architecture bears a resemblance to a VLIW processor. However, the control structure of the Montium is very different. The lower part of Fig. 6.3 shows the Communication and Configuration Unit (CCU) and the upper part show the coarse-grained reconfigurable Montium Tile Processor (TP).

6.4.1.1 Communication and Configuration Unit

The CCU implements the network interface controller between the NoC and the Montium TP. The CCU provides configuration and communication services to the Montium TP. The definition of the network interface depends on the NoC technology that is used in a SoC in which the Montium processing tile is integrated [3].

The CCU enables the Montium TP to run in "streaming" as well as in "block" mode. In "streaming" mode the CCU and the Montium TP run in parallel. Hence, communication and computation overlap in time. In "block" mode, the CCU first reads a block of data, then starts the Montium TP, and finally after completion of the Montium TP, the CCU sends the results to the next processing unit in the SoC (e.g., another Montium processing tile or external memory).

6.4.1.2 Montium Tile Processor

The TP is the computing part of the Montium processing tile. The Montium TP can be configured to implement a particular DSP algorithm. DSP algorithms that have been implemented on the Montium are, for instance, all power-of-2 FFTs upto 2,048 points, non- power-of-2 FFT up to 1,920 points, FIR filters, IIR filters, matrix vector multiplication, DCT decoding, Viterbi decoders, and Turbo (SISO) decoders [4].

Figure 6.3 reveals that the hardware organization of the Montium TP is very regular. The five identical arithmetic logic units (ALU1 through ALU5) in a tile can exploit data level parallelism to enhance performance. This type of parallelism demands a very high memory bandwidth, which is obtained by having 10 local memories (M01 through M10) in parallel. The small local memories are also motivated by the locality of reference principle. The data path has a width of 16-bits and the ALUs support both signed integer and signed fixed-point arithmetic. The ALU input registers provide an even more local level of storage. Locality of reference is one of the guiding principles applied to obtain energy efficiency in the Montium TP.

A relatively simple sequencer controls the entire Montium TP. The sequencer selects configurable TP instructions that are stored in the decoder blocks of Fig. 6.3. For (energy) efficiency it is imperative to minimize the control overhead. The Montium TP instructions, which comprise ALU, AGU, memory, register file, and interconnect instructions, are determined by a DSP application designer at design time. All Montium TP instructions are scheduled at design time and arranged into a Montium sequencer programme. By statically scheduling the instructions as much as possible at compile time, the Montium sequencer does not require any sophisticated control logic which minimizes the control overhead of the reconfigurable architecture.

The Montium TP has no fixed instruction set, but the instructions are configured at configuration time. During configuration of the Montium TP, the CCU writes the configuration data (i.e., instructions of the ALUs, memories and interconnects, sequencer and decoder instructions) in the configuration memory of the Montium TP. The size of the total configuration memory of the Montium TP is about 2.6 kByte. However, configuration sizes of DSP algorithms mapped on the Montium TP are typically in the order of 1 kByte. For example, a 64-point fast Fourier transform (FFT) has a configuration size of 946 bytes, which typically takes about 500 clock cycles required for configuring the Montium TP. Hence, the Montium TP can be configured for FFT-64 in less than 5 μs (assuming a configuration clock of 100 MHz). By sending a configuration file containing configuration RAM addresses and data values to the CCU, the Montium TP can be configured via the NoC interface. The configuration memory of the Montium TP is implemented as a 16-bit wide SRAM memory that can be written by the CCU. By only updating certain configuration locations of the configuration memory, the Montium TP can be partially reconfigured.

6.4.1.3 Montium Design Methodology

The Montium development tools start with a high-level description of an application (e.g. in C/C++ or MATLAB) and translate this description to a Montium TP configuration. The Montium design methodology to map DSP applications on the Montium TP is divided into three steps:

1. The high-level description of the DSP application is analyzed and computation intensive DSP kernels are identified.
2. The identified DSP kernels or parts of the DSP kernels are mapped on one or multiple Montium TPs that are available in a SoC. The DSP operations are programmed on the Montium TP using an embedded C language, called MontiumC.
3. Depending on the layout of the SoC in which the Montium processing tiles are applied, the Montium processing tiles are configured for a particular DSP kernel or part of the DSP kernel. Furthermore, the channels in the NoC between the processing tiles are defined.

6.4.2 Circuit Switched Network-on-Chip

The Network-on-Chip was designed as a circuit switching NoC, i.e. communication between any two tiles in the SoC goes via pre-configured routes. These routes are configured in the routers using a dedicated configuration interface of the routers [5]. Typically, in the applications considered in this project data streams are fixed for a relatively long time. Therefore, a connection between two tiles is required for a long period (e.g. seconds or longer). A large amount of the traffic between tiles will need a guaranteed throughput, which can be easily guaranteed in a circuit-switched connection. Current SoC architectures have a large amount of wiring resources that give enough flexibility for streams with different bandwidth demands. Internally, a circuit switching router has a minimal amount of control in its data pad (e.g. no arbitration). This increases the energy- efficiency per transported bit and the maximum throughput.

The network-on-chip consists of two circuit switched routers. It enables concurrent communication between the four Montium processing tiles and the rest of the SoC via an AHB bridge.

The network router interface consists of two identical unidirectional physical channels, each containing four 20-bit wide lanes. A lane consists of 16-bit data, 2-bit flit type (FT), 1-bit data valid and 1-bit acknowledge signals.

Each Montium processing tile is connected to a router via two unidirectional physical channels. In order to send data to the network a tile has to send a 20-bit network packet onto one of its output lanes. Up to four packets can be sent in parallel by a single tile, by using all outgoing lanes.

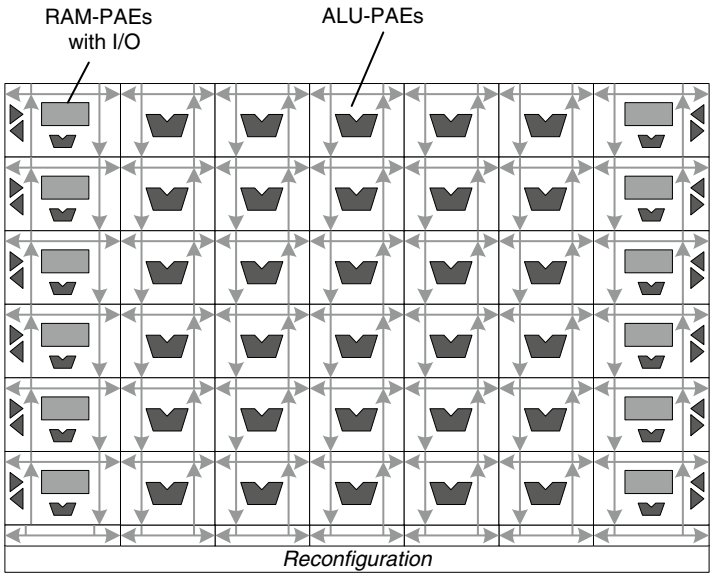

Fig. 6.4 XPP Array

6.4.3 XPP Reconfigurable Tile

The XPP Array is a coarse grained reconfigurable processing unit designed by PACT XPP Technologies [6]. The XPP Array (Fig. 6.4) is built from a rectangular array of two types of Processing Array Elements (PAEs): Those in the centre of the array are *ALU-PAEs*. At the left and right side are *RAM-PAEs* with *I/O*. An ALU-PAE contains three integer ALUs, two in top-down direction and one in bottom-up direction. A RAM-PAE contains two ALUs, a small RAM block and an I/O object. The I/O objects provide access to external streaming data sources or destinations.

The horizontal routing busses for point-to-point connections between XPP objects (ALUs, RAMs, I/O objects, etc.) are integrated in the PAEs. Separate busses for 16-bit data values and 1-bit events are available. Furthermore, vertical routing connections are provided within the ALU-PAEs and RAM-PAEs

An application is described as a flow graph where the nodes define the function to be performed (e.g. addition, multiplication). Each node is then mapped to one ALU while the graph's edges define the connections. Data flows through the network of operators. The event network can steer the data flow based on calculation results. This enables conditional execution and while loops. The strength of the XPP Array originates from the combination of parallel array processing with fast run-time reconfiguration mechanisms [7, 8]. PAEs can be configured while neighbouring PAEs are processing data. Entire algorithms can be configured and run independently on

different parts of the array. A configuration typically requires only a few thousand cycles. This is several orders of magnitude faster than reconfiguration of large FPGAs.

In the course of the 4S project the XPP Array was extended in order reduce area and thus power consumption for a given application. One strategy to achieve this is automatic cutting of small sub-graphs from the overall graph and mapping them to PAEs which execute this graph sequentially, but at higher frequency [9].

6.4.3.1 Streams

The basic communication concept of the XPP-III architecture is based on *streams*. A *data stream* is a sequence of single data packets travelling through the flow graph that describes the algorithm. In addition to data packets, state information packets ("events") are transmitted through independent event connections. Event packets containing one bit of information are used to control the execution of the processing nodes or may synchronize external devices.

The XPP communication network enables automatic synchronization of packets. An XPP object (e.g. an ALU) operates and produces an output packet only when all input data and event packets are available. The benefit of the resulting self-synchronizing network is that only the number and order of packets travelling through a graph is important. There is no need for the programmer or compiler to care about absolute timing of the pipelines during operation.

6.4.3.2 XPP Design Tools

The XPP comes with a complete tool chain [8]. The array is either programmed with a proprietary mapping description language or a vectorizing C Compiler. The C-Compiler extracts implicit parallelism form standard C-code (e.g. inner loops) and converts it to control and data flow graphs which then are mapped on the PAEs. Data flow graphs are simulated and visualized graphically.

6.4.3.3 Power Estimation

The XPP tool chain was extended by an A*pplication Power Estimator*. The power requirement of the XPP Array is highly dynamic. A PAE which is not contributing to calculating in a specific clock cycle consumes nearly no power (besides leakage and clock tree). The tool parses the simulator output per cycle and integrates the power consumption for a specific task. The power tool was calibrated with the Synopsys power compiler for a specific silicon library (e.g. ST CMOS 90 nm). The output of this tool in terms of nJ per task, described in XML format is then used by the *Spatial Mapper* (as described later). The power consumption over time can also be visualized graphically. This enables to profile the application partitioning and reconfiguration strategy in terms of power.

6.4.3.4 XPP Hardware

In the 4S project an existing processor, the XPP64-A, with 64 24-bit wide ALU PAEs and 16 RAM-PAEs was used. The chip is mounted on an evaluation board with a MIPS processor which is responsible for administrative tasks, XPP reconfiguration and communication. A video board provides a frame buffer and video output enabling test and demonstration of video algorithms. This board was linked through parallel streaming interfaces to the basic verification platform's FPGA board. The FPGA multiplexes several data streams originating from the SoC (i.e. ARM or DMA) to be processed by the XPP Array acting as coprocessor.

6.4.4 Demonstrators

Hardware and software for two demonstrator platforms have been designed. The first, the *Basic Verification Platform (BCVP)* is used as a multi-purpose platform:

- Operating System verification at an early project stage
- Test and verification of new hardware tiles on FPGA
- Interfacing to other board-level hardware tiles and legacy hardware
- Application test and benchmarking
- Early demonstration of the 4S concept.

The second demonstrator hardware, the *Highly Integrated Verification Platform,* replaces some modules of the basic platform by new chips that integrate new hardware tiles on silicon.

6.4.4.1 Basic Verification Platform

This platform is built from existing boards and chips. The specification was driven by the 4S target applications which are the radio broadcast standard Digital Radio Mondiale (DRM) and MPEG4 video decoding. Following the needs of those applications the hardware is built from:

- A SoC board based on an ATMEL SoC that integrates peripherals and two ARM processors
- An FPGA extension board with interfaces
- An Radio RF front-end based on an ATMEL RF chip
- An XPP Evaluation board with video interface which is connected to the FPGA extension board

Most evaluation and demonstration tasks were performed on this platform. The FPGA was used for the implementation and initial verification of the Montium reconfigurable cores and associated NoC routers. Another FPGA configuration provides a parallel interface to the XPP evaluation board with the XPP64A reconfigurable processor chip. The interface is capable to multiplex several independent data streams which are required for the video decoding application.

6.4.4.2 Highly Integrated Verification Platform

To show that the complexity of a heterogeneous architecture can be handled on silicon, a SoC (named *Annabelle,* ATMEL CMOS 130 nm) and analogue RF frontend (*Amelie*) have been designed and produced. The hardware platform reuses most modules of the Basic Verification Platform. However the SoC board and Radio front end have been replaced by boards for the new silicon chips.

6.5 Silicon Implementation

Implementation of a complex System on Chip on silicon was one of the project objectives. As outlined in the introduction one first needs to specify the application field. In 4S we decided to demonstrate the DRM on silicon. It became obvious that two chips need to be designed. One is a flexible RF radio front end, the second was the heterogeneous SoC. The chips integrate all functions which are required for a universal digital radio receiver. The DRM requires lots of signal processing power and flexibility.

6.5.1 Analogue RF Frontend

RF processing is a typical field where analogue hardware is required. We have developed a widely configurable analogue RF building block which connects directly to the SoC. The chip provides the analogue building block of the proposed heterogeneous tiled 4S architecture model. Though this device is not part of the overall task mapping scheme – simply because no alternative implementation exists – it is mandatory for the DRM proof of concept.

The new analogue RF chip is an advanced AM/FM/DRM frontend tuner with fast PLL integrated on a single chip. It represents a complete, automatically adjustable AM/FM/DRM front end, containing an analogue AM/DRM up/down conversion and FM down conversion system for an Intermediate Frequency (IF) of 10.7 MHz which allows an economic filter concept. The impedance driver at the IF output is designed for the A/D converter which is integrated in the new digital SoC. The chip registers are programmed through a 3-wire serial protocol originating from the SoC.

The fast tuning concept realized in this part is based on patents and allows lock times less than 1 ms for a jump over the FM band with a step width of 12.5 kHz. An automatic tuner alignment is provided by built-in D/A converters for gain and offset compensation (to support the computer-controlled alignment). The frequency range of the IC covers the AM/DRM band as well as the FM broadcasting band, the Japan band and the Weather band.

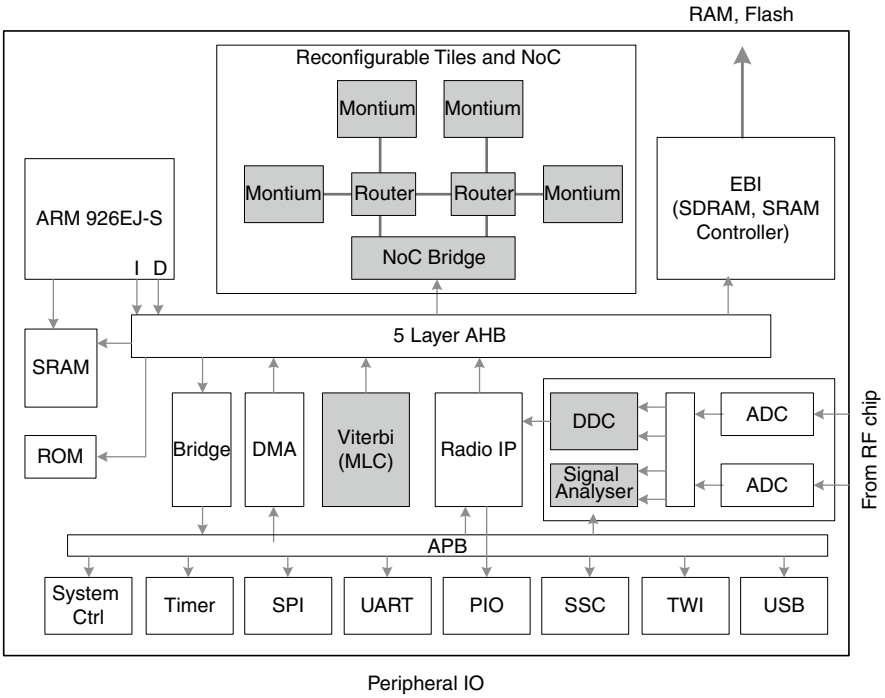

Fig. 6.5 SoC block diagram, new modules are shaded

6.5.2 Digital System on Chip

The SoC (Fig. 6.5) incorporates the complete digital signal processing chain required for a software defined radio receiver. Analogue inputs from the RF chip are converted by two integrated A/D converters. The digital samples are fed into new signal analysers providing quick information about the signal quality and to two Digital Down Converters (DDC) which provide the digital input to be processed by the processor tiles. The mentioned digital modules have been designed in the course of the project. The Viterbi decoder is also a new ASIC tile that unburdens the processors and provides best performance/power ratio for this task. Baseband processing is performed by the reconfigurable fabric which contains four reconfigurable Montium cores that are connected to the NoC. The reconfigurable fabric is connected to the AHB bus and serves as a slave to the AMBA system. In fact, the reconfigurable fabric acts as a reconfigurable co-processor for the ARM926 processor. Computation intensive DSP algorithms are typically offloaded from the ARM926 processor and processed on the coarse-grained reconfigurable Montium cores inside the reconfigurable fabric.

6.5.3 Runtime System and Tools

6.5.3.1 RTOS

Systems on chip need to perform several tasks with partially hard real-time constraints. In 4S the *eCos* real-time operating system was chosen as target operating system [1]. eCos, released under the GNU GPL license, is during design-time widely configurable to the final processor footprint and has been designed to support applications with real-time requirements. It is fully pre-emptive and provides minimal interrupt latencies, the necessary synchronization primitives, scheduling policies and interrupt handling mechanisms needed for real-time applications. eCos also provides all the functionality required for general embedded application support including device drivers, memory management, exception handling, C, math libraries, etc. The eCos' hardware abstraction layer (HAL) was adapted to support the digital SoCs, e.g. communication with the reconfigurable tiles, hard-wired accelerators and the peripherals. For booting services the *RedBoot* debug and bootstrap tool was used. Both 4S demonstrator platforms are using eCos.

6.5.3.2 OSYRES Framework

The goal of *OSYRES* is to provide an abstraction of the multi-processing platform. OSYRES [2] hides the details of the communication between functional nodes and hides the details of instantiation and allocation of functional nodes on a multi-processor platform. The OSYRES framework has been used for proof of concept and implementation of the DRM and MPEG4 applications.

OSYRES is built on top of eCos and provides the 4S-specific features. The architecture of OSYRES is optimized to support of data flow applications. The design of a data flow application differs from the traditional sequential designed application. In a *sequential* designed application, functions are called in a sequence, which is defined by the application's state machine. In a *data flow* application, functions are triggered by the availability of data or messages at their input ports. In a multi-processor system with a run-time function allocation, the arrival of messages and thus the order at which functions are called is not predictable anymore unless they are explicitly synchronized. In OSYRES synchronization is explicitly done via messages that automatically synchronize two functions. In addition, functions may run in parallel on different processors. This requires a different way of defining the application in *Functional Nodes* (*FN)*.

6.5.3.3 Application Specification

An application in OSYRES is defined by a set of task graphs consisting of Functional Nodes connected by channels described in a data flow graph (see Fig. 6.6). A task

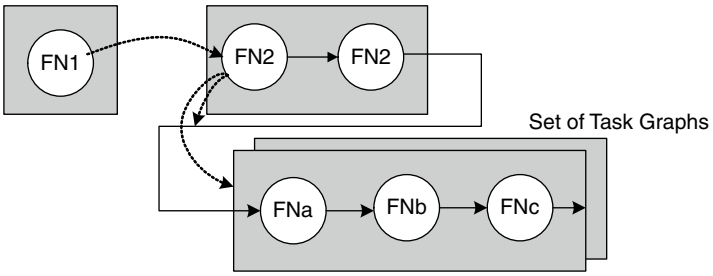

Fig. 6.6 Functional Node (FN) definition

graph is a combination of functionality, which is always instantiated as a whole. It is possible to instantiate multiple instances of the same task graph. A task graph instantiation is performed by one of the already running functional nodes.

A FN will always be instantiated on one processing device. It is a self-contained functional entity, which communicates only via its ports with other FNs. These ports form the end-points of *uni-directional channels*. The channels will be instantiated on the communication links between the processing devices such as a bus or Network on Chip (NoC).

The approach of task graphs and self-contained entities, which communicate via well-defined messages results by definition in a modular design of the overall application.

6.5.3.4 Functional Node Design Concept

An OSYRES Functional Node is defined by its functionality at its ports. There may be different Functional Node implementations for each different type of processing unit but the external behaviour must be the same for all implementations. The FN implementation itself does not take any resource of the processing unit, except for the storage of the functional implementation image. At run-time instances of functional implementations are created and it is only then, when they take resources from the processing unit. The amount of resources that is needed for an instance of a FN implementation will be defined at design time and specified in the system definition (memory, MIPS, energy) characterizing the node.

All instances of a Functional Node implementation executed by the same processor type use the same code. To allow multiple instances, each instance has its own administration, which is created at FN instantiation. This administration is the storage of all FN local data.

In addition of the operational functionality, a FN also contains special functionality to control the instantiation of new instances. Control events may be events to *Create*, *Delete*, *Start* and *Stop* a FN instance as well as channel connection events like *Add* and *Remove* ports.

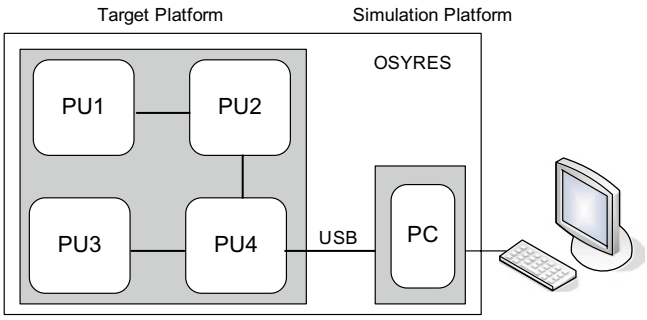

Fig. 6.7 Simulation with a PC and four Processing Units (PU)

6.5.3.5 Application Verification

As OSYRES can be implemented on different platforms, there is an implementation for the most common development or simulation environments; e.g. OSYRES for Windows using the Visual C development environment. Before the real target platform is available, one can already start simulating the application on such a common PC platform, e.g. by using simulators as MatLab for the actual processing.

On this simulation platform, applications can be verified in two steps. The first step is the verification of the application structure. In this step, the task graph structure and the messaging between FNs is verified. The second step is the verification of the functionality of the FNs individually. As the interfaces are clearly defined, only module-level test is needed. After successful verification, the overall application can be simulated first on the PC platform and in a later stage on the target platform. By combining the OSYRES platforms of the target (e.g. SoC) and the common PC platform, one unified OSYRES platform is obtained. This allows a smooth transfer from the simulation to the target platform. Starting from a running application on the simulation platform, FNs are gradually transferred to the target platform.

Figure 6.7 shows an example where the target is connected with the simulation PC via a USB cable. Starting from an application running in the simulation environment, functional processes are transferred to the target. The abstraction of communication in OSYRES assures the connections within the application, regardless where the functional processes are located.

6.5.3.6 System Definition

OSYRES provides an abstraction of the multi-processing platform. Figure 6.8 shows an example of a system platform which could be a dual ARM System on Chip, with one shared bus on which the memory and three peripherals are connected.

Each ARM processing device runs its own operating system i.e. eCos. Within this local RTOS peripherals and memory areas are allocated to one of the processing devices.

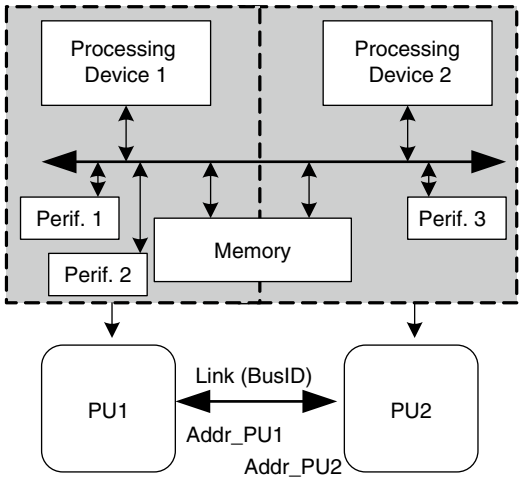

Fig. 6.8 Sample system platform with two Processing Units (PU)

Specific bus drivers are added to access the connected communication medium, in this example a shared bus.

A *Link* in OSYRES is defined as a method to access the other processing device via the communication medium. This method is implemented in a *Link driver,* which is built on top of the RTOS bus driver.

A *Processing Unit (PU)* in OSYRES is defined by the RTOS defined functionality and the OSYRES specific functionality as the communication layer, scheduler and control network as well as the previously mentioned Link drivers. In the context of the 4S demonstrator platforms a PU may be an ARM core, a Montium processor, the XPP processor or a hardwired core such as Viterbi.

The OSYRES functionality is built on top of the local RTOS. The OSYRES communication layer offers access to the platform-wide communication layer. The OSYRES control network provides the system-wide instantiation of Functional Nodes. Each PU has a local instantiation of this control network in the form of a *PU Manager*. This PU manager is responsible for the instantiation of the FNs on the PU. All PU Managers communicate with one central *Control Node*. This Control Node is located on one of the PUs and includes the run-time spatial mapping algorithm.

Reconfigurable structures as introduced in the previous sections may run FNs as well. In order for the OSYRES platform to control the reconfigurable processing devices, it needs to be adapted to control the reconfigurable structures.

These reconfigurable devices are considered as Processing Units without an operating system and will be controlled by a remote PU Manager, which runs on one of the generic processing devices like the ARM. The remote PU Manager translates control commands, like Create, Start and Stop into device specific actions (like Load, Configure and Start) towards the reconfigurable device. In addition, the remote PU manager controls the communication with the reconfigurable device, which does not support the OSYRES communication protocol. This allows the integration of OSYRES on devices which are not able to run an operating system.

6.5.4 Spatial Mapper

Common practice is to map applications to a hardware platform at design-time. In 4S we researched methods and algorithms how to perform the mapping at run-time. Run-time mapping offers a number of advantages over design-time mapping. It offers the possibility

- to adapt dynamically to available hardware resources. The available resources may vary over time due to applications running simultaneously or adaptation of the algorithms to the environment.
- to enable unforeseeable upgrades after first product release time, e.g. new applications and new or changing standards.
- to circumvent defective parts of a SoC. Larger chips mean lower yield. The yield can be improved when the mapper is able to avoid faulty sections of the chip. Also aging can lead to faulty parts that are unforeseeable at design-time.

The mapping tool *SMIT* (Spatial MappIng Tool) [11] is integrated into OSYRES and determines at runtime a near-optimal mapping of a given set of applications to the 4S heterogeneous architecture. The application is modelled as a set of communicating Functional Nodes. In the context of the 4S project "optimal mapping" means that the energy consumption of the SoC is minimized with regard to the mapping, while other constraints are still satisfied. Other optimization criteria such as user response time, memory consumption etc. could be the optimization target but were not the focus in 4S. SMIT uses the performance characteristics that are determined for each functional node implementation at design-time.

6.5.4.1 Inputs

A number of inputs are required for SMIT:

Application description
An application is assumed to be given by a set of task graphs, consisting of Functional Nodes and interconnections between processes needed for communication. This partitioning is typically done manually by an experienced designer, but future tools may perform partitioning also automatically. Additional Quality of Service requirements for the application such as required timing behaviour (e.g. throughput, latency) are also specified in the application description.

Library description
For each Functional Node of an application, one or more implementations have to be provided. A FN implementation is the implementation of a FN on a particular type of tile, e.g. object code for an ARM or configuration data for an FPGA, Montium or XPP. A FN implementation is annotated with performance figures, e.g. the amount of energy it takes to execute the function on a particular tile of the architecture or the amount of cycles it takes to execute the function on a particular tile of the architecture.

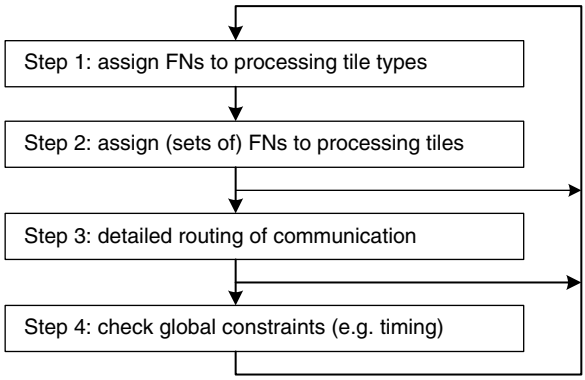

Fig. 6.9 Hierarchical mapping algorithm

Architecture description
The heterogeneous SoC architecture consists of multiple tiles of different types (e.g. ARM, reconfigurable cores) interconnected by a Network-on-Chip (NoC) or Bus. For each tile a number of characteristics have to be provided on beforehand, such as the type of the tile, the amount of available memory, the clock frequency, etc. The NoC consists of routers and links. The links are used to interconnect routers or a tile with a router. It is possible to have different links in parallel between the same source and destination. Also the NoC characteristics have to be provided, such as the topology of the network, the frequency of the clock of the network, latency per router, etc.

6.5.4.2 Mapping Approach

The mapping of an application to a multi-processor architecture is a General Assignment Problem (GAP), which is known to be NP-complete. To deal with the complexity of the mapping problem, SMIT employs a hierarchical iterative approach (Fig. 6.9). The idea is to solve the problem using multiple levels. At each level a particular decision is made that reduces the search space. Decisions of higher levels are considered to be fixed at the lower levels. On higher levels not all details are taken into account to improve the speed of evaluation.

In other words, higher levels use a higher abstraction. This hierarchical approach has the danger that decisions which seem to be promising on a higher level due to the underlying assumptions of the abstraction level, show to be bad or even lead to resulting sub problems which are infeasible. The allowed maximum runtime of the mapper routine is restricted so far that at least a *good* mapping solution, can be achieved. Good mapping guarantees that the Quality of Service requirements of the application are met. *Good* mapping is preferred over *optimal* mapping which would again consume too much processing power and energy.

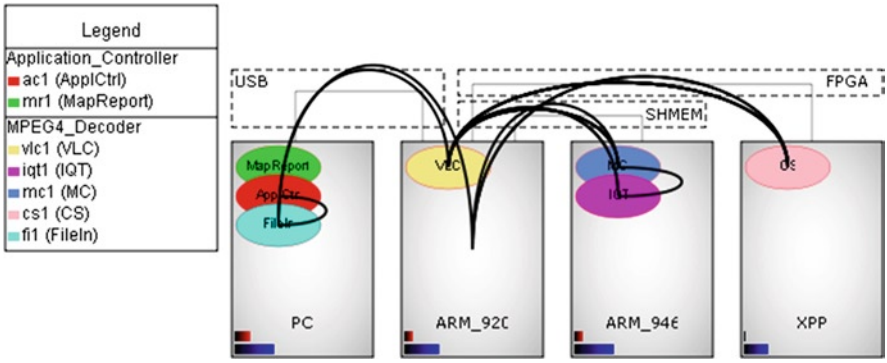

Fig. 6.10 Sample visualization of FNs

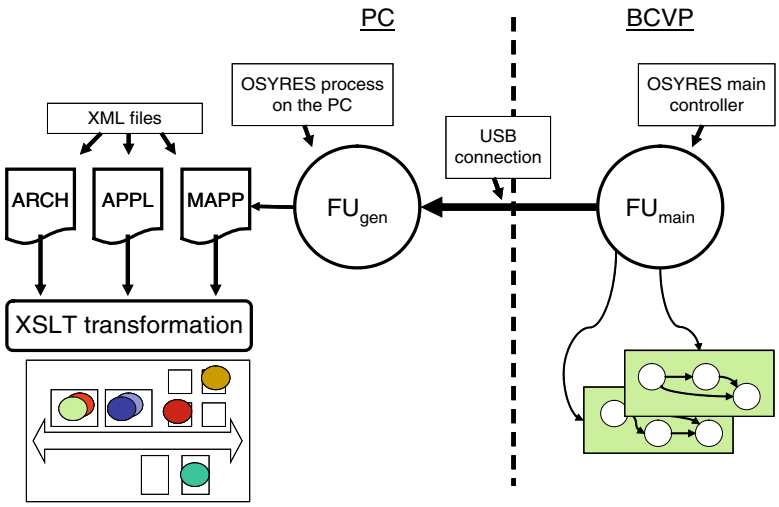

Fig. 6.11 Visualization of mapping using OSYRES

6.5.5 Mapping Visualization

For debugging and demonstration purposes, the mapping proposed by SMIT can be displayed on screen (Fig. 6.10). The visualization tool performs this task by utilizing the proposed XML tool flow in combination with the platform transparent communication layer of OSYRES (Fig. 6.11).

For visualisation two instances of OSYRES are launched: one instance is active on the target platform and the other instance is executed on a PC. Both instances communicate via an USB connection. The mapping information that SMIT provides is sent to the USB interface by a Functional Node (*FUmain*) of OSYRES.

A second Functional Node (*FUgen*) of OSYRES running on the PC reads the USB interface and converts the received data to an XML file with the mapping information, which is graphically displayed. The mapping information in XML can be combined with the already present architecture and application descriptions in XML.

The data received by FUgen contains a set of IDs that can be found in the XML files describing the architecture and the application. As the processes (i.e. FN instantiations) and channels in a task graph have unique IDs and the processing tiles and links in the hardware architecture have unique IDs too, the received mapping information only needs to include a mapping of process IDs and process implementation IDs to processing tile IDs and channel IDs to link IDs, respectively. The current mapping is exported to XML by the *FUgen* OSYRES process.

This approach demonstrates also the benefits of the OSYRES concept: it is easy for the programmer to distribute the functional nodes to several of heterogeneous processing units without the need to care about communication details.

6.6 Proof of Concept

The overall concept was verified through several scenarios targeting MPEG4 Video and DRM. Applications have been designed according to the proposed tool flow, i.e. application definition in XML and mapping to OSYRES functional nodes and links. For demonstration the Basic Verification Platform was used since it was available early in the project and provided reconfigurable cores (Montium implemented on the FPGA and the XPP evaluation board), two ARM processors and hardwired tiles. In the following section some of the scenarios which have been successfully demonstrated are presented.

6.6.1 MPEG4

The MPEG4 Video case stands for a typical video decoding application. The application was chosen in addition to DRM to demonstrate the usability of the concept on very different application fields.

6.6.1.1 MPEG4 Task Graph and Mapping Scenarios

The Task Graph in Fig. 6.12 shows the nodes of the MPEG4 decoding chain which is composed out of four Functional Nodes that have been implemented to run on several tiles (i.e. *PUs* in the OSYRES context):

- Variable Length Coding (VLC): ARM920
- Inverse Quantization (IQT): ARM920, ARM946, XPP

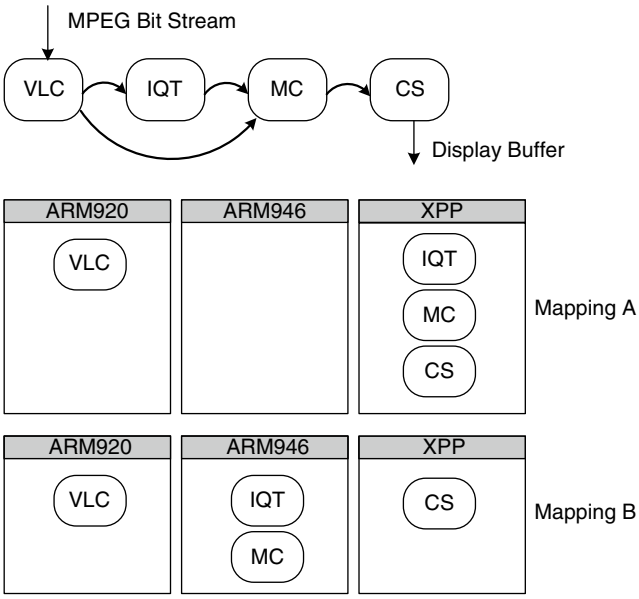

Fig. 6.12 MPEG4 Task Graph and two mapping scenarios

- Motion Compensation (MC): ARM920, ARM946, XPP
- Colour Space Conversion (CS): XPP

Each implementation provides also a performance model (XML) as input for the spatial mapper. The performance models describe the energy for each function, memory requirements, processing resource utilization and communication costs. The performance models have been generated through simulation and a new power estimation tool for the XPP. Energy for the ARM processors has been calculated using datasheet values (mW/MHz) and by profiling the number of clock cycles which are required for a specific function. Based on the Task Graph and performance models SMIT mapped the FNs to the available hardware resources. OSYRES provides the abstraction layer for FN creation and channel establishing. Besides VLC and CS, which can only be executed on specific tiles, SMIT had the choice to distribute the other tasks freely. Note that the dynamic reconfiguration properties of the XPP are used to execute the Functional Nodes sequentially.

The application's switching points have been specified at design time. It is not feasible to allow switching at any point in time in an application since the number of states to be saved and transferred to the target FN varies during execution. Furthermore it may be difficult to transfer processing states from architecture to another, because e.g. the internal data and state representation may differ. For MPEG4 best switching points are I-Frames, since those frames do not need information from previous or subsequent frames.

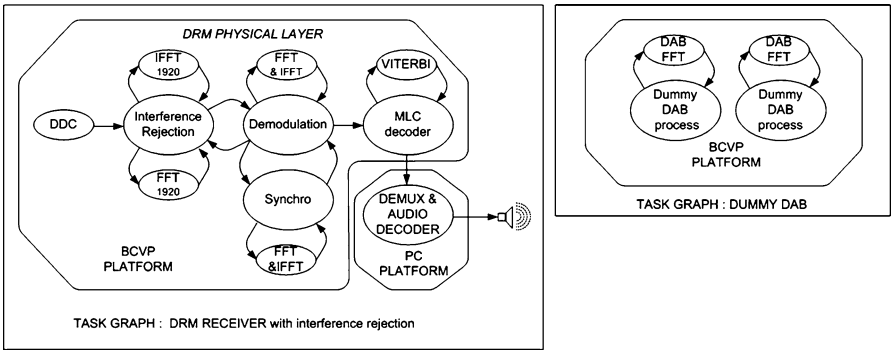

Fig. 6.13 DRM with interference rejection and DAB task graphs

For demonstration purposes the SMIT has been forced to reassign mapping by adding placeholder FNs which can only be executed on a specific tile. SMIT then had to recalculate the mapping. Note that the example in Fig. 6.12 shows only a subset of potential mappings.

A live demonstration with the basic verification platform and XPP showed that with an MPEG4 stream seamless switching between several mappings could be done in real-time without disturbing or interrupting the video output.

6.6.2 DRM

The Digital Radio Mondiale (DRM) physical layer was ported to the OSYRES environment. According to the OSYRES model, the DRM physical layer was described as task graph (Fig. 6.13). The FFT and IFFT used in the DRM interference rejection were implemented for both, the ARM and Montium. Additional features allowed to switch on/off the interference rejection and to launch a background receiver (which is used to scan for alternative frequencies). In addition a "placeholder" DAB receiver (of which only the FFT was implemented) could be added. Several usage scenarios were specified according to the typical needs of a car radio receiver. For demonstration purposes the switching between scenarios was done by keystrokes mimicking varying reception quality. The tests have been performed on the basic verification platform. Available hardware resources (i.e. OSYRES *PUs*) were the ARM processors, the hardwired down converters (DDC), the Viterbi decoder and three reconfigurable Montium tiles. For simplicity reasons audio decoding was performed on a PC linked via USB.

Figure 6.14 shows the task graph of a scenario where interference rejection is added e.g. because of reception getting worse. Interference rejection requires in addition calculating a FFT1920 and an IFFT1920. SMIT selected from all possible mapping the most power efficient resource assignment as shown. So, the most compute-intensive tasks are mapped on the Montium.

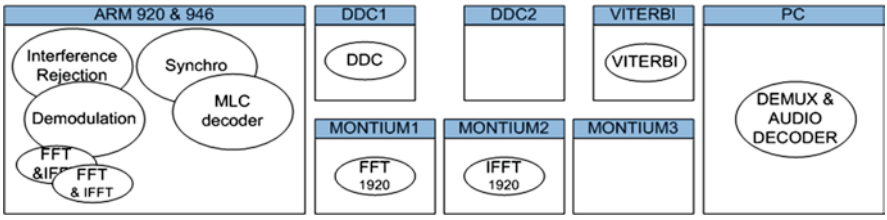

Fig. 6.14 DRM receiver with interference mapping (FFT1920 and IFFT1920)

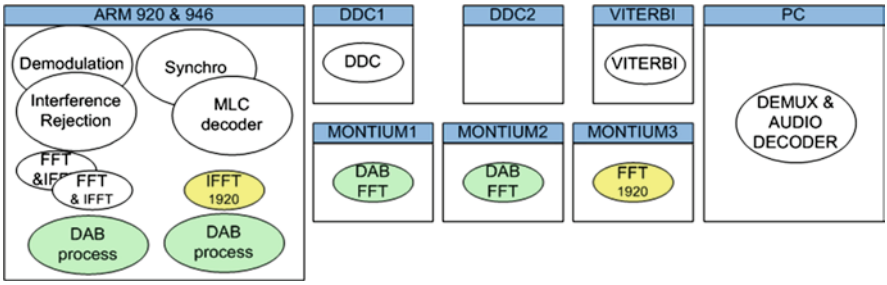

Fig. 6.15 Mapping with additional DAB FFTs

Another scenario in Fig. 6.15 starts again from a DRM receiver and an already mapped DAB receiver running in background (which delivers traffic information in a car radio). In this scenario the DAB has utilized already two Montium tiles to compute the DAB FFTs. Now interference rejection was switched on: SMIT mapped the FFT1920 to the third Montium tile and the IFFT to the ARM. In any case the mapping was performed without interruption of audio.

6.6.2.1 SMIT Mapper and OSYRES Benchmarking

Experiments on the basic platform delivered the following figures on an ARM9 running at 86 MHz while six FNs have been mapped to four PUs.

- SMIT mapping algorithm: 0.7 ms
- OSYRES task graph generation overhead: 13 ms (2 ARMs and 2 Montiums)
- Message transfer: ~0.15 ms (NoC and Montium driver).

6.6.2.2 SoC Demonstration

In order to prove that a complex heterogeneous platform can be implemented on silicon using state-of-the-art ASIC design tools, the Highly Integrated Concept Verification Platform was realized and functionally tested and an estimation of the

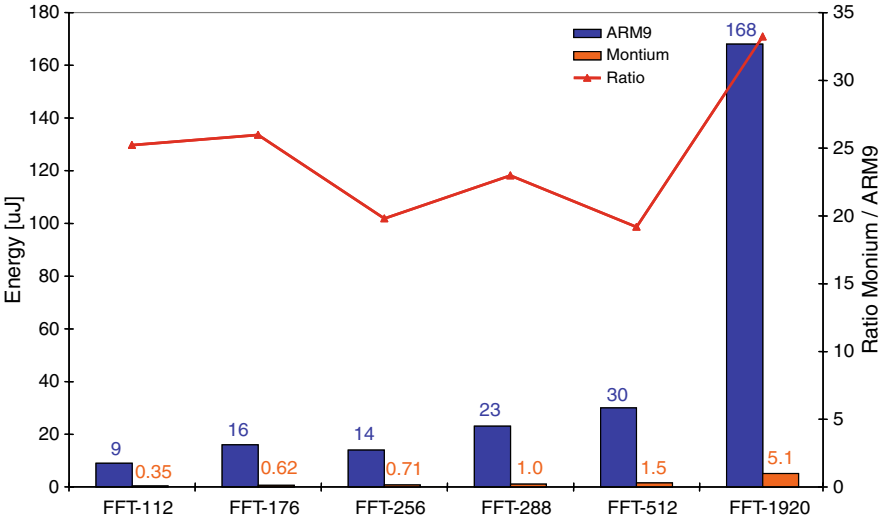

Fig. 6.16 ARM9 – Montium FFT energy comparison

dynamic power was made. The test suite demonstrated that all devices including the Montium and routers were fully functional ("first time right"). Dynamic power estimation was performed on the post-layout netlist. Fig. 6.16 shows the ARM9 and Montium comparison of energy for single FFTs. The power reduction factor is between the factor 20 and 33 in favour of reconfigurable technology compared to general purpose processors. Absolute values of dynamic power consumption of the Montium are in the range of 0.24–0.6 mW/MHz. Reconfiguration and control overhead of the reconfigurable hardware is about 10%. In addition, power reduction comes in combination with higher processing power.

6.6.3 Outlook

The intention of 4S was to establish a sample heterogeneous reconfigurable platform and to map tasks to the platform resources during runtime. The mapping algorithm was setup to optimize overall power consumption. With current SoC architectures that have only a small number of heterogeneous cores (< 10) the spatial mapper has only a limited set of mapping scenarios to choose from and optimisation potential is limited. The situation changes if dozens or even hundreds of cores are available. With those complex platforms it is not feasible anymore to define the mapping at design time, thus a runtime tool is mandatory to perform the mapping. The mapping algorithm does not necessarily need optimising energy consumption only. Also other criteria such as QoS may steer the mapping.

An important outcome of 4S is that task mapping can be controlled dynamically during runtime without interrupting the application. The following list gives an outlook to future scenarios which may benefit from the 4S approach:

- Devices with silicon geometries below 65 nm will suffer from severe process variations even on the same die. There will be processing cores which can only operate with reduced frequency while others achieve full frequency. Based on chip test results a performance model for each of cores can be provided and the runtime system automatically maps tasks according to requested application needs without overloading the slower cores.
- Future chips will suffer from defects and silicon lifetime constraints. A spatial mapper can adapt to the availability and current performance of cores during the lifetime of a product. This redundancy concept is completely transparent for application software.
- The 4S concept enables the most aggressive power reduction concepts such as frequency and voltage scaling and dynamic power switching. The operating system maps tasks seamlessly to the configured performance.
- Application software does not need to be tailored to a specific heterogeneous device architecture. The same application software will run on different heterogeneous platforms and hardware re-spins. For example, imagine high end and entry level mobile devices, all running the same application software that automatically distributes its tasks to the available resources.

The mapping concept is currently part of research activities within Thales. They extend the 4S mapping concept by further optimisation parameters. Lessons learned from 4S are further investigated in e.g. the *CRISP* project. The CRISP project studies heterogeneous reconfigurable many-core SoCs, runtime mapping and self-repairing reconfigurable core concepts for improved dependability. An extended version of the XPP reconfigurable processor was implemented in the MORPHEUS [18] project on silicon. Both projects are discussed in specific chapters of this book.

6.7 Conclusion

The 4S project researched innovative hardware concepts along with compile-time and run-time tools to realize a computational powerful and flexible platform for Ambient Systems. On the hardware side, we targeted the development of a dynamically reconfigurable System-On-Chip (SoC) architecture. It basically offers the required flexibility to adapt its hardware resources to diverse processing requirements that result from the demands of different applications as well as environmental changing conditions (e.g. different communication protocols or fading channels). Different to reconfigurable architectures like RAW [16], Pleiades [17], PicoChip [12], MorphoSys [14], Cell [13] or Imagine [15] which are complex multicore designs whose focus is not primarily put on power efficiency, our approach particularly targeted embedded devices where maintaining an ultra-low power profile is of great

importance. Therefore we investigated and developed a *heterogeneous* SoC platform concept, composing miscellaneous hardwired as well as configurable analogue and reconfigurable digital core buildings blocks, interconnected by a Network on Chip (NoC) communication infrastructure.

We thereby were able to contribute to the research of reconfigurable architectures as we applied a holistic view on the overall system to achieve energy efficiency. Unlike in the past, various aspects have, therefore, been investigated in depth in an overlapping manner, i.e. tradeoffs in combining reconfigurable analogue and digital building blocks, design-time application characterization for energy-efficient data processing on heterogeneous processing tiles and dynamic migration of tasks to reconfigurable processing tiles to maintain or even improve quality of service and energy efficiency. With respect to the technique mentioned last, 4S also extended the run-time tools by a system-aware middleware solution, called OSYRES. Based on user demands, environmental, power and quality of service constraints its spatial mapper (SMIT) evaluates an optimal assignment of tasks to tiles and thus efficiently exploits the flexibility of computation in time and space. The decision where to place tasks is thereby not limited to the starting time of applications. It rather reacts dynamically on any changes to the mentioned criteria. Thereby, the proposed methodology, to realize applications upon a task graph model, enabled overcoming the drawback of ending and restarting an application to handle task migration on heterogeneous systems. Instead, the migration process is handled by the middleware through interaction with the running application in background, so that the user is unable to recognize the remapping process.

With all the developed methodologies, techniques and hardware concepts the outcome of 4S project had a significant impact on energy efficient embedded computing targeting applications with run-time dynamic behaviour. We have shown that heterogeneous dynamically reconfigurable architectures are very suited to realize flexible and energy efficient embedded computing platforms. This was not only done on theoretical and prototyping level but also verified based on two highly complex chips (analogue and digital) that have been realized within the project. The exploitation of the flexibility of the hardware is thereby given by a set of compile- and run-time tools that enable dynamic mapping of task to computational tiles without interruption of running applications. For the first time ever, this seamless mapping technique has been shown on a heterogeneous dynamically reconfigurable SoC architecture based on two complex applications, MPEG4 and DRM.

Besides technical related aspects, 4S also stimulated commercial interests and activities in reconfigurable computing by attracting distinguished partners of Europe's leading ICT industry. Upon that, the ideas, concepts, methods and hardware designs researched in 4S have been the basis for further European projects, e.g. CRISP [19] and MORPHEUS [18]. Finally, all major industrial and academic partners of the 4S project have perused their activities exploiting reconfigurable technology by participating in consecutive projects.

Acknowledgements The authors wish to acknowledge Jens Becker (ITIV, Karlsruhe Institute of Technology) and Jan Stoter (WMC) for review and, last but not least, all project partners, reviewers and EU project officers for their contributions that enabled to successfully fulfil the goals of the 4S project.

References

1. eCos: http://ecos.sourceware.org
2. OSYRES: http://www.ti-wmc.nl
3. Marcel D. van de Burgwal, Gerard J.M. Smit, Gerard K. Rauwerda and Paul M. Heysters, Hydra: an Energy-efficient and Reconfigurable Network Interface, Proceedings of the 2006 International Conference on Engineering of Reconfigurable Systems & Algorithms, 26–29 June 2006, Las Vegas, USA, pp. 171–177, CSREA Press, ISBN 1-60132-011-6
4. Gerard K. Rauwerda, Paul M. Heysters and Gerard J.M. Smit, Towards Software Defined Radios Using Coarse-Grained Reconfigurable Hardware, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 16, no. 1, pp. 3–13, January 2008
5. P.T. Wolkotte, G.J.M. Smit, G.K. Rauwerda, L.T. Smit, An Energy-Efficient Reconfigurable Circuit Switched Network-on-Chip. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) – 12th Reconfigurable Architecture Workshop (RAW 2005)*, 4–8 Apr 2005, Denver, Colorado, USA. 155. IEEE Computer Society. ISBN 0-7695-2312-9
6. XPP-III Processor Overview (White Paper), URL: www. pactxpp.com
7. Reconfiguration on XPP-III Processors (White Paper), URL: www. pactxpp.com
8. Programming XPP-III processors (White Paper), URL: www. pactxpp.com
9. M. Weinhardt, M. Vorbach, V. Baumgarte, and F. May: Using Function Folding to Improve Silicon Efficiency of Reconfigurable Arithmetic Arrays, Proceedings of the IEEE International Conference on Field-Programmable Technology FPT'04, Brisbane, Australia, Dec. 2004
10. G.J.M. Smit, A.B.J. Kokkeler, P.T. Wolkotte, P.K.F. Hölzenspies, M.D. van de Burgwal, and P.M. Heysters. The Chameleon Architecture for Streaming DSP Applications. *EURASIP Journal on Embedded Systems*, 2007. 78082. ISSN 1687–3955
11. L. T. Smit, G. J. Smit, J. L. Hurink, H. Broersma, D. Paulusma, and P. T. Wolkotte, "Run-time assignment of tasks to multiple heterogeneous processors," in *5TH PROGRESS Symposium on Embedded Systems*. STW Technology Foundation, 2004, pp. 185–192.
12. A. Duller, G. Panesar, and D. Towner. Parallel Processing – the picoChip way! In J. Broenink and G. Hilderink, editors, *Communicating Processing Architectures 2003*, pages 125–138, 2003.
13. J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the cell multiprocessor. In *IBM Journal of Research and Development*, 2005.
14. Guangming Lu, H. Singh, Ming-Hau Lee, N. Bagherzadeh, F.J. Kurdahi, E.M.C. Filho, and V. Castro-Alves. The MorphoSys dynamically reconfigurable system-on-chip. In *Evolvable Hardware, 1999. Proceedings of the First NASA/DoD Workshop on*, pages 152–160, 1999.
15. Jung Ho Ahn, W.J. Dally, B. Khailany, U.J. Kapasi, and A. Das. Evaluating the imagine stream architecture. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pages 14–25, 2004.
16. M. Taylor et al., "The RAW microprocessor: a computational fabric for software circuits and general-purpose programs," *Micro, IEEE*, vol. 22, no. 2, pp. 25–35, 2002.
17. H. Zhang et al., "A 1-V heterogeneous reconfigurable DSP IC for wireless baseband digital signal processing," *Solid-State Circuits, IEEE Journal of*, vol. 35, no. 11, pp. 1697–1704, 2000.
18. F. Thoma et al., "MORPHEUS: heterogeneous reconfigurable computing," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, 2007, pp. 409–414.
19. CRISP project, Cutting-edge Reconfigurable ICs for Stream Processing, http://www.crisp-project.eu