Chapter 15

SYSTEMC AND OCAPI-XL BASED SYSTEM-LEVEL DESIGN FOR RECONFIGURABLE SYSTEMS-ON-CHIP

Kari Tiensyrjä[1], Miroslav Cupak[2], Kostas Masselos[3], Marko Pettissalo[4], Konstantinos Potamianos[3], Yang Qu[1], Luc Rynders[2], Geert Vanmeerbeeck[2], Nikos Voros[3] and Yan Zhang[1]

[1]*VTT Electronics, P.O.Box 1100, FIN-90571 Oulu, Finland;* [2]*IMEC, Kapeldreef 75, B 3001 Leuven, Belgium;* [3]*INTRACOM SA, P.O. Box 68, GR-19002 Peania, Attika, Greece;* [4]*Nokia Technology Platforms, P.O.Box 50, FIN-90571 Oulu, Finland*

Abstract Reconfigurability is becoming an important part of System-on-Chip (SoC) design to cope with the increasing demands for simultaneous flexibility and computational power. Current hardware/software co-design methodologies provide little support for dealing with the additional design dimension introduced. Further support at the system-level is needed for the identification and modeling of dynamically re-configurable function blocks, for efficient design space exploration, partitioning and mapping, and for performance evaluation. The overhead effects, e.g. context switching and configuration data, should be included in the modeling already at the system-level in order to produce credible information for decision-making. This chapter focuses on hardware/software co-design applied for reconfigurable SoCs. We discuss exploration of additional requirements due to reconfigurability, report extensions to two C++ based languages/methodologies, SystemC and OCAPI-xl, to support those requirements, and present results of three case studies in the wireless and multimedia communication domain that were used for the validation of the approaches.

Keywords: co-design; communication; configuration overhead; context switching; design space exploration; dynamic reconfiguration; mapping; multimedia; OCAPI-xl; partitioning; reconfigurable; reconfigurability; SystemC; system-on-chip; wireless.

P. Boulet (ed.) Advances in Design and Specification Languages for SoCs, 255–269.

1. Introduction

Reconfigurable systems have raised a lot of research interest in recent years, and various reconfigurable architectures and technologies have been proposed [Compton et al., 2002]. Reconfigurable hardware combines the capability for post fabrication silicon reuse by different application tasks with computational efficiency due to the hardware-like spatial computation style. This fact allows the efficient adaptation to different operating conditions and/or different standards. The presence of reconfigurable resources on-chip also allows post shipment functionality modifications/upgrades and bug fixing capability similar to that of software.

Reconfigurability does not however come at no cost. The reconfiguration introduces extra delays as well as area and energy overheads. A further concern is technology portability, since embedded reconfigurable blocks are available only for a limited number of silicon processes. Additionally, current design methodologies and tools do not provide efficient support for dealing with this new design dimension at the system level.

A number of reconfigurable technologies are commercially available. Off-the-shelf Field Programmable Gate Arrays (FPGAs) available by companies such as Xilinx and Altera offer system-level densities of logic, memories and hardwired resources including processor cores [Virtex II Pro]. Due to their high unit costs FPGAs are not suitable for large volume consumer applications. Embedded FPGAs that can be integrated in customized SoCs are also commercially available [Varicore]. Coarse grain reconfigurable architectures include functional units of word level granularity such as PACT XPP technology [XPP]. However their commercial presence is limited compared to FPGAs mainly due to the difficulties in developing efficient mapping tools for such architectures.

The cost of deep submicron semiconductor technologies and the increasing design costs in state of the art semiconductor designs push for a move from conventional SoCs towards heterogeneous partly reconfigurable SoCs. Especially for signal processing type applications, coarse grain reconfigurable architectures are likely to dominate [Srikanteswara et al., 2003] since bit level granularity architectures (such as different FPGA flavors) offer high flexibility at the too high expense of power and area.

The rest of the chapter is organized as follows: The next section describes related research. Section 3 discusses specific requirements imposed by reconfigurability on the SoC design flow. Support extensions based on SystemC and OCAPI-xl languages and tools are described in Section 4. Section 5 summarizes three case studies in the domain of wireless communication that were used for experimenting and validating the approaches. Conclusions are drawn in Section 6.

2. Related Research

There have been several approaches researched towards developing reconfigurable architectures and associated software tools. However, they concentrated mainly to develop novel architectures, and their tools do not directly address the problem of modeling the reconfigurability at system-level. Such projects include e.g. Garp [Callahan et al., 2000], Xputer [Hartenstein, 2001], RAW [Taylor et al., 2002] and PipeRench [Goldstein et al., 2000].

Most existing high level design approaches for reconfigurable systems target compilation of C or C-like descriptions of targeted applications on reconfigurable architectures. In [Venkataramani et al., 2003] the compilation of a single assignment C-like language on the Morphosys coarse grained reconfigurable architecture is described. In [Cardoso et al., 2003] the compilation of C programs on the XPP reconfigurable platform is described. The approach also considers the temporal partitioning of the targeted behavioral description. Recently also a few co-compilation and co-synthesis type [Becker et al., 2003] design approaches for reconfigurability have been published.

The reconfigurable hardware brings a new dimension to system partitioning. The functional blocks of executable specification are partitioned into parts that will be implemented with software, hardware or dynamically reconfigurable blocks. The dynamic reconfiguration requires partitioning to address both temporal and spatial dimensions. Such an automatic partitioning is in a general case still an unsolved problem, but in specific cases solutions for temporal partitioning [Bobda, 2003], for task scheduling [Noguera et al., 2003] and for context management [Maestre et al., 2001] have been proposed.

3. Reconfigurability Requirements for Design Methodology

Reconfigurability manifests itself throughout the design flow. The System-Level Design (SLD) phase is the main focus of this paper and a more detailed diagram of it is shown in Fig. 15.1. The latter phases of the design flow are already more or less technology/vendor-specific and design flows and tools provided by vendors need to be used.

The SLD phase identifies reconfigurability needs, scenarios, constraints, and functionality in C or C++. It analyses and estimates the functional blocks with respect to reconfigurable implementation, decides on system partitioning and performs a system-level simulation to estimate the performance and resource impacts.

At the SLD phase the handling of reconfigurability requires an approach that addresses the three possible resource classes, i.e. software, fixed hardware and reconfigurable hardware. The SLD phase should support the following tasks:

- Unified description of system functionality in e.g. C

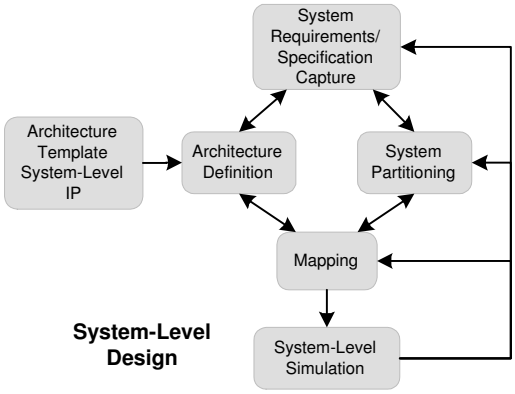

Figure 15.1. System-level design.

- Analysis and estimation to support justified partitioning decisions
- Reuse of architectures and Intellectual Property (IP) blocks
- Fast and efficient design space exploration
- Modeling of the effects of reconfiguration.

There are three major issues related to the reconfigurable technology that need to be modeled at the system level in order to get reliable information about the trade-offs between area, speed and total cost: the computational capacity requirements of functional blocks, the required resources needed for the largest reconfigurable context, and the delays and memory consumption caused by the reconfiguration.

The approach of this work is to take a holistic view to the overall SoC design flow in order to identify parts of the co-design methodology, where the inclusion of reconfigurability has the largest effects. It should be noted that reconfigurability does not appear as an isolated phenomenon, but as a tightly connected part of the overall SoC design flow. We do not constrain the system-level design to a specific architecture instance, but the designer defines the granularity by decomposing the functionality and explores reconfigurability through estimation, modeling, transformation and performance simulation. Reuse is supported through templates, and design space exploration allows alternatives to be studied in order to fine-tune the architecture, partitioning and mapping. The main properties of the explored reconfigurable technology alternatives are annotated to the system-level design. All the main decisions are already made at the exit of the system-level design phase.

4. Extending Systemc and OCAPI-xl for Support of Reconfigurability

SystemC is a standard modeling language based on C++, class libraries and a simulation kernel that provides the basic mechanisms for the system level modeling.

We have adopted SystemC language and tools [Grötker et al., 2002] as a base environment, on top of which we build our extensions for support of modeling, design space exploration and performance evaluation of reconfigurable parts at the system level.

4.1 SystemC Based Support

For designing of reconfigurable parts at system level, we developed: 1) an estimation method and tool for estimating execution time and resource consumption of function blocks on dynamically reconfigurable logic to support system partitioning, 2) a SystemC based modeling method and tool for reconfigurable parts to allow fast design space exploration through 3) system-level simulation using transaction-level models of the system.

Estimation Approach. The estimation approach applies basic principles of high-level synthesis, and is used for selecting candidate components that could benefit from implementation on a reconfigurable resource [Qu et al., 2003].

The starting point is the functional description given as a C-language algorithm. The designer decides the granularity of partitioning by decomposing the algorithm down to function blocks. A single function block may then be assigned to either software, reconfigurable logic or a fixed functional unit. Each of the function blocks will be individually studied and the set of estimation information will be fed into the system-level partitioning decision phase.

Together with the profiling information that is collected by running the executable algorithms with certain application data, the C codes of function blocks are transformed into a control data flow graph (CDFG) using the SUIF compiler of the Stanford University [SUIF]. For a selected instruction-set processor, the software (SW) estimator produces the estimated execution time. The hardware (HW) estimator produces the estimated HW execution time and the estimated HW resource utilization for each individual function block. The estimates can be used to narrow the design space in order to obtain a satisfactory solution with a few iterations.

Reconfigurability Modeling. The modeling method and associated tool transforms candidate components presented as SystemC modules to use a special SystemC template called Dynamically Reconfigurable Fabric (DRCF) as depicted in Fig. 15.2 [Pelkonen et al., 2003].

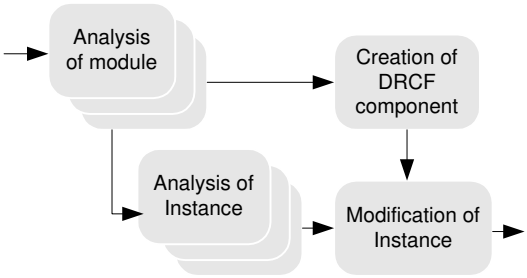

Figure 15.2. SystemC modeling method.

The template of the DRCF contains a configuration scheduler and an input splitter that routes data transfers to the correct instances. An example of a DRCF component is shown as part of a SoC model on the right hand side of Fig. 15.3. The configuration scheduler checks the target of each interface method call, forwards it if the target is active, or activates a context switch if the target is not active. This process automatically models context switching and the memory bus traffic. The automatic model transformer keeps the functionality of candidate components unchanged. The approach provides the means to test the effects of implementing some components in dynamically reconfigurable hardware.

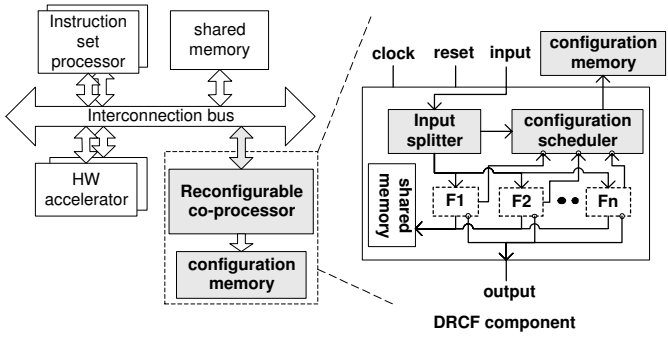

Figure 15.3. SoC model and DRCF template.

System-Level Simulation. As reconfigurability adds a new dimension to the design space, a method of analyzing performance of the resulting system in the early phase of design is needed. This can be achieved using SystemC transaction-level workload operation models. The timing information of com-

putation can be obtained from an estimation tool or from designers' experience. The factors related to the communication are architecture-dependent and can be set as parameters.

4.2 OCAPI-xl Based Support

To allow modeling of reconfigurability features at system level, we developed: 1) new software process type in OCAPI-xl [Ocapi-XL], 2) coupling of OCAPI-xl to SystemC for co-simulation, and 3) context switching from one resource towards another (software, reconfigurable hardware).

Software Processes Scheduling Extension. In the high-level software model of computation, concurrency is considered at the processor level. This means that for every process there is a separate processor assumed. Naturally, in real life this will typically not be the case. In realistic software implementation an operating system allows all the processes to be assigned to the same software processing resource. So from the performance point of view, the processes are not running concurrently, but they are sequentialized by the operating system scheduler onto the processing unit. To model such a behavior in the OCAPI-xl performance model, a separate process type, procManagedSW, has been introduced as depicted in Fig. 15.4.

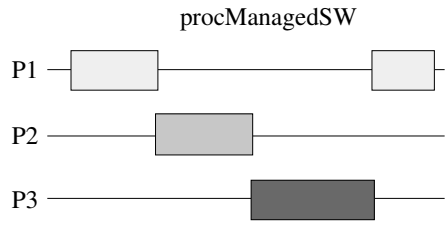

Figure 15.4. Sequentializing computation over time.

To be able to create a process of the type procManagedSW, the designer must first create a scheduling object. This scheduler will perform the actual sequentialization of all the processes that are attached to this object. The way this is done is defined in one of the member methods of this scheduling object. The user can define its own scheduling objects to model the behavior of the scheduler present in the target operating system.

It is important to realize that switching between the different SW tasks is not penalty-free. In order to come to the most accurate performance results, context-switching overhead is also considered in the performance model. The user can define extra context switching time for every process created, which is then applied to that process during the OCAPI-xl simulation.

SystemC Implementation of OCAPI-xl Threaded Process Extension. SystemC provides an implementation of a thread library bundled together with an event-driven simulation engine with notion of virtual time. Thus SystemC is used for implementation of the threaded-process extension with the additional bonus of automatically having an OCAPI-xl/SystemC co-simulation environment. Such an environment brings together the advantages of OCAPI-xl and SystemC. The essential idea of the common OCAPI-xl/SystemC environment is to let the whole OCAPI-xl part run in a single SystemC thread and to use SystemC synchronization mechanisms inside a modified OCAPI-xl simulation kernel to synchronize with the rest of the (SystemC) system. Since the OCAPI-xl kernel is single-threaded there are no thread compatibility problems created by this setup. The basic structure of the OCAPI-xl/SystemC co-simulation environment is shown in Fig. 15.5.

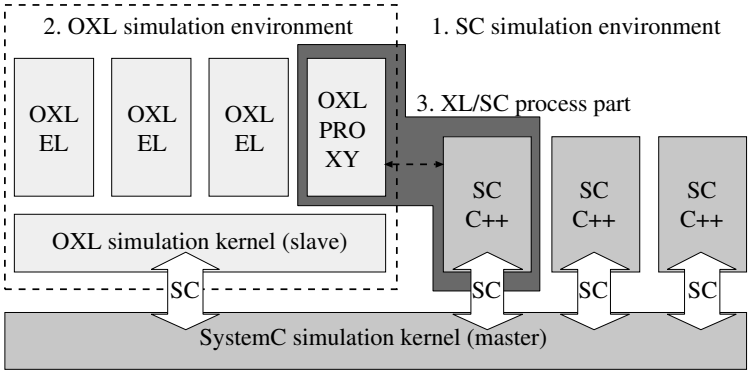

Figure 15.5. OCAPI-xl/SystemC co-simulation environment structure.

It consists of three domains within the SystemC environment:

1 The native SystemC processes controlled only by the SystemC simulation kernel

2 The OCAPI-xl domain running in a single SystemC thread and controlled by the OCAPI-xl kernel acting as a slave to the SystemC simulation kernel

3 OXL/SystemC processes running C++/SystemC code with access to OCAPI-xl's communication and synchronization primitives. These processes contain internally two parts: the SystemC part running the C++ or SystemC code, and a small OCAPI-xl proxy process, which is active when the process is executing an OCAPI-xl synchronization / communication primitive.

High-Level Modeling of Context Switching. The ability to reschedule a task either in hardware or software is an important asset in a reconfigurable system-on-chip. To support this feature, high-level implementation and management of hardware/software relocatable tasks in OCAPI-xl have been modeled. The aim is to model a pre-emptive relocation of tasks from the reconfigurable logic to the SW and vice versa. The model supports spatial temporal scheduling in hardware and software.

The OCAPI-xl code below illustrates the example of coding context switching for a task P1, switching between the different contexts (High-Level HW and ManagedSW), and simulating its behavior.

```
procDRCF P1("P1");
  //-- initial context: High-Level HW (default period of 10)
  P1.context(HLHW);
  //-- second context: SW under Round-Robin scheduler(RR)
  P1.context(ManagedSW, \&RR);
  //-- next context: High-Level HW with period of 2
  P1.context(HLHW, 2);
{
    //-- here goes "normal" OCAPI-xl task code

    //-- upon this operator the task will switch itself to the next context
    switchpoint();

    //-- here goes some more task code
}
  //-- and run the simulation for 2000 cycles
run(2000);
```

5. Design Cases

The SystemC and OCAPI-xl based approaches and extensions have been applied in the WCDMA, WLAN and MPEG-4 design cases in order to validate them at the system level, and to get experiences on the detailed and implementation design of reconfigurability on selected demonstrator platforms. The three cases represent different reconfiguration scenarios:

- The WCDMA detector case represents a study of applying partial dynamic reconfiguration in a mobile terminal
- The WLAN case presents a static reconfiguration scenario to allow generation of a family of wireless networking systems
- The MPEG-4 case presents a scenario where tasks are relocated between software and reconfigurable hardware.

5.1 WCDMA Detector

The application is an adaptive linear minimum mean-square error (LMMSE) detector [Heikkila, 2001] that is used in the downlink part of the WCDMA system. When compared to traditional RAKE detectors, it achieves 1- 4 dB

better performance in challenging channel conditions, uses a channel equalizer for performing multi-path correction instead of multiple RAKE fingers, and can be scaled to higher data rates by increasing clocking rates.

In the downlink data channel, each frame has 15 slots (2560 chips/slot) in a time period of 25 ms. The detector contains an adaptive filter, a channel estimator, a multi-path combiner and a correlator bank. In addition to the detector part, the searcher (code, frame and slot synchronization), de-interleaver and channel decoder of the WCDMA receiver are shown in Fig. 15.6. Starting with the C-representation of the WCDMA detector, the SystemC based approach was applied.

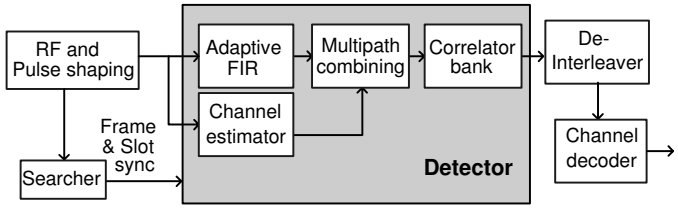

Figure 15.6. WCDMA base-band receiver.

The high-level estimation of the briefly optimized C code show that 1078, 1387, 463 and 287 FPGA look-up tables (LUTs) are required for the adaptive filter, channel estimator, combiner and correlator respectively. Based on the estimated resources, three different SystemC models were created in the system partitioning, mapping and performance simulation phase. For the fixed system, the processing time achieved was 1.12 ms per slot in an FPGA running at 100 MHz and the resource consumption was as mentioned above. In the case of dynamic partial reconfiguration, the processing was partitioned in two contexts, one containing the channel estimator and the other the rest three blocks. The benefit was that almost 50% of resource reduction could be achieved when compared to the fixed system, but at the cost of 8 times longer processing time. The pure software system resulted in 30 times longer processing time than the dynamic partial reconfiguration. In the partial reconfiguration case, device data such as configuration speed were taken from Xilinx Virtex-II Pro datasheet, which showed overwhelming reconfiguration latency that is almost 8 times of the processing time.

The WCDMA detector was implemented through the detailed and implementation design phases on the Memec Design's Virtex-II Pro FF1152 P20 Development Board using the Xilinx Embedded Development Kit (EDK) and ISE tools. Controlling of the WCDMA detector was handled by a SW process running on a PowerPC hardcore embedded in the FPGA platform. The run-time reconfiguration is realized using the Xilinx SystemACE solution. In

the implementation, 920 LUTs and 4 Block RAMs are required for the context containing the channel estimator, and 1254 LUTs, 6 Block RAMs and 12 Block Multipliers for the other context.

5.2 WLAN

The WLAN case presents a scenario, where reconfigurability is exploited to allow generation of a family of wireless networking systems. Specifically a dual mode WLAN - outdoor fixed wireless access system-on-chip is targeted. The WLAN was developed first based on the Hiperlan/2 standard [ETSI]. The system-on-chip realizes both MAC and Physical layer functionalities of the standard. The Hiperlan/2 physical layer is based on Orthogonal Frequency Division Multiplexing (OFDM). The MAC layer of Hiperlan/2 is based on a TDD/TDMA approach. Functionality upgrades/modifications will be developed in a second step to allow operation of the targeted system-on-chip in outdoor environments under a fixed wireless access scenario (in a non-overlapping fashion with Hiperlan/2). The simpler choice for the integration of the extra functionality after fabrication would be in the form of software upgrades. However due to the complexity of certain parts of the extra functionality (mainly corresponding to physical layer tasks) acceleration may be required. This can be achieved by including reconfigurable resources in the targeted system-on-chip.

The high level exploration for the development of the targeted dual standard system-on-chip was based on ANSI-C and OCAPI-xl. The Hiperlan/2 system was explored at the first step. Following an algorithmic exploration for the physical layer using MATLAB, a unified ANSI-C model of the targeted functionality (physical and MAC layers of Hiperlan/2) was developed. Physical layer blocks were modeled as parameterized procedures that produce the minimum amount of data required by the next procedure in the flow (pipelining). The procedures are parameterized with respect to different factors such as input data bit widths. Shared data for the inter module communication are presented as global variables. Communication between MAC layer modules is activated when all data produced by each module are ready (no pipelining). The physical layer part of the ANSI C model has a size of 9000 code lines while the MAC part includes 10000 code lines. Both parts use a common library of 1000 code lines.

Using the ANSI C model as input an OCAPI-xl model of the complete targeted functionality was developed. High level exploration was performed using high level OCAPI-xl processes (HLHW, HLSW) to evaluate different partitioning solutions. The complex tasks that are expected to be upgraded/duplicated in the future, i.e. need to be realized in reconfigurable hardware, were modeled as hardware (HLHW) processes. The physical layer part of the OCAPI-xl

model has a size of 13000 code lines while the MAC part includes 8000 code lines. Both parts use a common library of 500 code lines.

A prototype of the targeted system-on-chip was developed on the ARM Integrator platform. The platform hosts the AMBA AHB bus with all the required support peripherals for its operation (arbiter, interrupt controller etc.). Two ARM7 TDMI microprocessors on two separate boards - "core modules" are included in the platform and realize the software parts of the targeted system-on-chip. One processor acts as protocol processor while the other runs lower MAC functionality and controls the modem. The size of the code running on the protocol processor is 1.4 Mbytes while the size of the code running on the modem control processor is 50 Kbytes. The tasks that will be realized on ASIC or reconfigurable blocks in the targeted system-on-chip are mapped on two FPGAs - "logic modules". Each "logic module" hosts a Xilinx Virtex E 2000 FPGA. The total utilization of the two FPGAs resources is: 405 I/Os, 31450 function generators, 23416 CLBs and 14912 D flip flops.

5.3 MPEG-4 Decoder

The initial C/C++ code was obtained from FDIS (Final Draft International Standard) sources. First, the ATOMIUM toolset [ATOMIUM] for pruning of code, data transfer and storage exploration and advanced source-to-source transformations were applied to increase the performance and reduce the power of the system. Applying automatic pruning with functionality test bench reduced code to 40 % of its original size and further manual reorganization and rewriting reduced the code size by factor 5.4. The memory optimizations were driven in two phases, frame-based to macro-block-based data flow transformation and introduction of block-based data flow, with the aim of reducing the number of accesses and improvement of the locality of data. In the next steps of the design, platform dependent optimizations were applied. The targeted platform for implementing the MPEG-4 decoder was Xilinx's Multimedia Development Board containing Virtex-II FPGA with a single embedded MicroBlaze soft processor, surrounded with external ZBT memory banks.

After the optimization phase, the design proceeded with modeling of performance estimation in OCAPI-xl environment. A number of small tests were done on the board to find the operator execution times and memory access times and to select proper memory architecture. Taking into account the ATOMIUM analysis results from optimization phase, the most CPU time and memory access demanding blocks of the decoder were selected to become the candidates for HW acceleration. From the OCAPI-xl point of view, the C/C++ code representing their behavior was rewritten to OCAPI-xl managed SW processes and later refined to high-level hardware processes for clock-true sim-

ulation and OCAPI specific processes for HDL code generation. Exploiting OCAPI-xl Operation Set Simulator, the design was modeled in two flavors:

- Configured as pure SW version of MPEG-4 decoder running on soft processor core
- Configured as HW accelerated version, where most cycle demanding blocks have been implemented in reconfigurable HW.

OCAPI-xl high-level modeling of context switching extension allowed the performance estimation of two reconfigurable scenarios, with the ability to model the reconfiguration time. The design was mapped on xc2v2000 Virtex-II FPGA with 46% utilization (5000 slices) for HW accelerator and 71% utilization (7703 slices) for the whole decoding system. It uses 33% of available 18x18 multipliers, 76% of block RAMs and 52% of LUTs.

6. Conclusions

Reconfigurability is a promising technique in a SoC to obtain software-type flexibility, while maintaining hardware-type computational capacity. The related implementation technologies and architectures are still under development. It looks obvious that different reconfigurable technologies will become accepted in different application and business domains. Static reconfiguration does not change the design methodology radically, but dynamic reconfiguration adds a new dimension to SoC design flow.

We presented support extensions to SystemC and OCAPI-xl that address the system-level design of reconfigurable parts in the context of the SoC design flow. The effects of reconfigurability are studied before committing to a specific reconfigurable technology or architecture instance. Experiences of extending SystemC and OCAPI-xl show that this is a viable approach and credible data for system-level decision-making can be produced with reasonable effort. All the material and tools of SystemC are still valid.

The case studies represented three different reconfigurability scenarios: in the WCDMA detector case dynamic partial reconfiguration was explored, in the WLAN case static reconfiguration was applied for obtaining a family of networking systems, and in the MPEG-4 decoder case task relocation between software and reconfigurable hardware was studied.

Acknowledgments

This work is partially supported by the European Commission under the contract IST-2000-30049 ADRIATIC, and partially by the participating organizations: IMEC, INTRACOM, NOKIA, STMB and VTT.

References

ATOMIUM: `http://www.imec.be/design/atomium`.

Becker J., Hartenstein R. (2003). *Configware and Morphware Going Mainstream*. Journal of System Architecture, Vol. 49. pp. 127-142.

Bobda C. (2003). *Synthesis of Dataflow Graphs for Reconfigurable Systems using Temporal Partitioning and Temporal Placement*. Dissertation. University of Paderborn. 90 p.

Callahan T., Hauser J., Wawrzynek J. (2000). *The Garp Architecture and C Compiler*. IEEE Computer., pp 62 - 69.

Cardoso J.M.P., Weinhardt M. (2003). *From C Programs to the Configure-Execute Model*. Proc. of 2003 Design Automation and Test in Europe Conference and Exhibition. Munich, Germany, March 3 - 7, 2003, pp. 576 - 581.

Compton K., Hauck S. (2002). *Reconfigurable Computing: A Survey of Systems and Software*. ACM Computing Surveys, Vol. 34, No. 2. 171-210.

ETSI: *Broadband Radio Access Networks (BRAN)*. HIPERLAN type 2; Physical (PHY) layer, V 1.2.1 (2000-11).

Goldstein S.C., Schmit H., Mihai B., Cadambi S., Matt M., Taylor R.R. (2000). *PipeRench: A Reconfigurable Architecture and Compiler*. IEEE Computer. April 2000, pp. 70 - 77.

Grötker T., Liao S., Martin G., Swan S. (2002) *System Design with SystemC*. Kluwer Academic Publishers, Boston, 240 p.

Hartenstein R. (2001). *Reconfigurable Computing - Architectures and Methodologies for System-on-Chip*. SoC technology seminar "Enabling Technologies for System-on-Chip Development". Tampere, Finland, November 19-20.

Heikkila M.J. (2001) *A Novel Blind Adaptive Algorithm for Channel Equalization in WCDMA Downlink*. 12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, Volume:1, pp . A-41 - A-45.

Maestre R., Kurdahi F.J., Fernandez M., Hermida R., Bagherzadeh N., Singh H. (2001). *A Framework for Reconfigurable Computing: Task Scheduling and Context Management*. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 9, Issue 6, pp. 858 - 873.

Noguera J., Badia R.M. (2003). *System-Level Power-Performance Trade-offs in Task Scheduling for Dynamically Reconfigurable Architectures*. Proc. of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems. pp. 73 - 83.

Ocapi-XL: `http://www.imec.be/design/ocapi`.

Pelkonen A., Masselos K., Cupak M. (2003) *System-Level Modeling of Dynamically Reconfigurable Hardware with SystemC*. 17th International Par-

allel and Distributed Processing Symposium (IPDPS 2003). Nice, France, 22 - 26 April 2003. IEEE Computer society, pp. 174 - 181.

Qu Y., Soininen J.-P. (2003) *Estimating the Utilization of Embedded FPGA Co-Processor*. 2003 Euromicro Symposium on Digital Systems Design (DSD 2003): Architectures, Methods and Tools. Antalaya, Turkey, 3 - 5 Sept. 2003. IEEE Computer Society. Los Alamitos, pp. 214 - 221.

Srikanteswara S., Palat R.C., Reed J.H., Athanas P. (2003). *An Overview of Configurable Computing Machines for Software Radio Handsets*. IEEE Communications Magazine, pp. 134 - 141.

SUIF: http://suif.stanford.edu/.

Taylor M.B., Kim J., Miller J., Wentzlaff D., Ghodrat F., Greenwald B., Hoffman H., Johnson P., Lee J.-W., Lee W., Ma A., Saraf A., Seneski M., Shnidman N., Strumpen V., Frank M., Amarasinghe S., Agarwal A. (2002). *The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs*. IEEE Micro, Volume: 22, Issue: 2, pp. 25 - 35.

Varicore: `http://www.actel.com`.

Venkataramani G., Najjar W., Kurdahi F., Bagherzadeh N., Bohm W., Hammes J. (2003). *Automatic Compilation to a Coarse Grained Reconfigurable System-on-Chip*. ACM Transactions on Embedded Computing Systems, Vol. 2, No. 4, pp. 560-589.

Xilinx Virtex II Pro: `http://www.xilinx.com`.

XPP: `http://www.pactcorp.com`.