

SystemC-based Design Methodology for Reconfigurable System-on-Chip

Yang Qu, Kari Tiensyrjä and Juha-Pekka Soininen
VTT Electronics, P.O. Box 1100 (Kaitoväylä 1), FIN-90571 Oulu, FINLAND
E-mail: yang.qu@vtt.fi

Abstract

Reconfigurable system is a promising alternative to deliver both flexibility and performance at the same time. New reconfigurable technologies and technology-dependent tools have been developed, but a system-level design methodology to support system analysis and fast design space exploration is missing. In this paper, we present a SystemC-based system-level design approach. The main focuses are the resource estimation to support system analysis and reconfiguration modeling for fast performance simulation. The approach was applied in a real design case of a WCDMA detector on a commercially available reconfigurable platform. The run-time reconfiguration was used and the design showed 40% area saving when compared to a functionally equivalent fixed system and 30 times better in processing time when compared to a functionally equivalent pure software design.

1. Introduction

Reconfigurability is becoming an important issue in design of System-on-Chip (SoC) because of the increasing demands of silicon reuse, product upgrade after shipment and bug-fixing ability. The reconfigurability is usually achieved by embedding reconfigurable hardware into the system. The result is a heterogeneous SoC that has the advantages of both the reconfigurable hardware and traditional types of computing elements such as general-purpose processors (GPP) and application-specific integrated circuit (ASIC).

A number of reconfigurable technologies are commercially available. The Xilinx [1] and the Altera [2] provide fine-grain FPGA platforms. They contain embedded processor cores, which make it possible to design a rather complex system in such FPGA platforms. The PACT XPP technologies [3] and the QuickSilver [4] provide coarse-grain reconfigurable computing platforms, which are suitable for DSP-type tasks. The Triscend A7S [5] and the Motorola MRC6011 [6] are configurable

SoCs, which bring both high flexibility and high performance.

However, the reconfigurability does not come free of costs. The power consumption, reconfiguration latency and related overhead are main concerns to system designers. How to address these problems and evaluate the effects of reconfiguration in early phase of the design are not supported in existing system-level design methodologies and tools.

In this paper, we present a system-level design methodology and supporting tools for design of reconfigurable SoC (RSoC). The work is an extension of the modeling methodologies presented in [7, 8]. The main advantage of the approach is that it can be easily embedded into a SoC design flow to allow fast design space exploration for different reconfiguration alternatives without going into implementation details.

The structure of the paper is as follows. Related research is presented in Section 2. Our design methodology and supporting tools are presented in Section 3. A real design case has been carried out using the design methodology and the results are presented in Section 4. Analysis of the case study and future work are presented in Section 5. The conclusions are given in Section 6.

2. Related Work

There are several research works such as Garp [9], MorphoSys [10], and PipeRench [11], toward developing novel reconfigurable architectures and associated software development tools. However, these are architecture-centric design frameworks that do not specifically address system-level design issues. In the heterogeneous reconfigurable system that we are studying, the reconfigurable hardware is an add-on processing unit to the system, so we concentrate on how to extend the existing design flow and tools to support the design of RSoC.

A few research works focus on the system-level issues of more advanced reconfigurable devices such as multi-context devices or the devices that allow a task to be freely allocated at run time. These include context

management [12], task relocation [13] and on-line scheduling [14]. In this work, we focus on more realistic reconfigurable technology, which allows run-time reconfiguration (RTR) but uses only single context devices. In addition, dynamic relocation is not taken into account since routing constraints restrict such flexibility in practice at the moment. Although this looks like a limitation of the reusability of our approach, our models and techniques are not bound to a particular reconfigurable technology and they can be easily extended in the future.

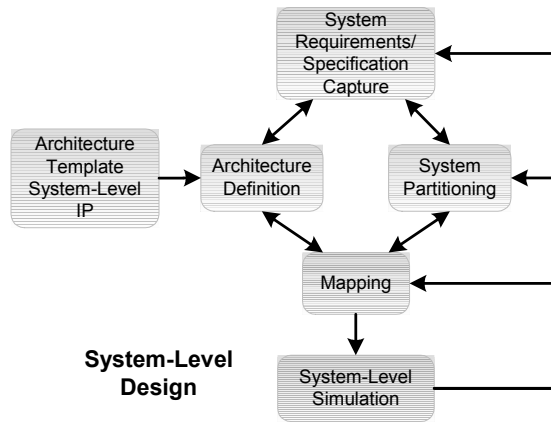


Figure 1. A generic system-level design flow

3. SystemC-based Design Methodology

A generic view of the system-level design flow is depicted in Figure 1. The following new features are identified in each phase when reconfigurability is taken into account [15]:

- *System Requirements and Specification Capture* needs to identify requirements and goals of reconfigurability
- *Architecture Definition* needs to model the reconfigurable resources at abstract level and include them in the architecture models
- *System Partitioning* needs to analyze and estimate the functions of the application for software, fixed hardware and reconfigurable hardware
- *Mapping* needs to map functions allocated to reconfigurable hardware onto the respective architecture model
- *System-Level Simulation* needs to observe the performance impacts of architecture and reconfigurable resources for a particular system function.

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. Our

approach is not intended to be a universal solution to support the design of any reconfigurability. Instead, we focus on a case, where the reconfigurable components are mainly used as co-processors in SoCs.

SystemC 2.0 [16] is selected as the backbone of the approach since it is a standard language that provides designers with basic mechanisms like channels, interfaces and events to model various kinds of communication and synchronization styles in system designs. More sophisticated mechanisms for the system-level design can be built on top of the basic constructs. More specifically, our system-level models operate on transaction-level of abstraction. The performance simulation is based on the estimates of computational complexity of each block, estimates of communication and storage capacity requirements, and characteristics of the architecture and mapped workload.

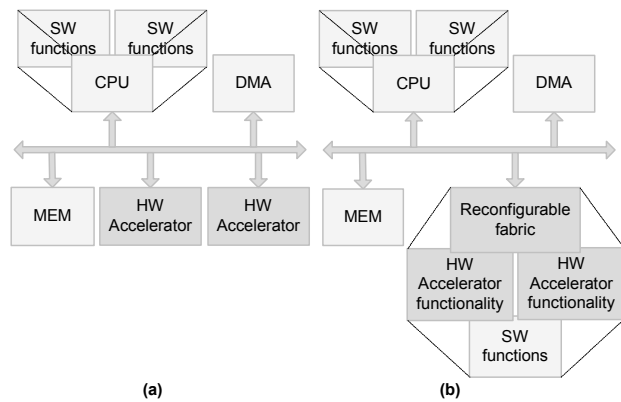


Figure 2. (a) An initial fixed SoC architecture and (b) a modified architecture using reconfigurable hardware

In the SystemC-based approach, we assume that the design does not start from scratch, but it is a more advanced version of an existing device. The new architecture is defined partly based on the existing architecture and partly using the system specification as input. The initial architecture is often dependent on many things not directly resulting from the requirements of the application. The company may have experience and tools for certain processor core or semiconductor technology, which restricts the design space and may produce an initial hardware/software (HW/SW) partition. Therefore, the initial architecture and the HW/SW partition are often given at the beginning of system-level design. The SystemC extension is designed to work with a SystemC model of the existing device to suit the design considering RTR hardware.

The way that the SystemC-based approach incorporates dynamically reconfigurable parts into

architecture is to replace SystemC models of some hardware accelerators, as shown in Figure 2(a), with a single SystemC model of reconfigurable block, as shown in Figure 2(b). The objective of the SystemC-based extensions is to provide a mechanism that allows designers to easily test the effects of implementing some components in the dynamically reconfigurable hardware. The provided supports in the SystemC approach include:

- Estimation and analysis support for design space exploration and system partitioning [17]
- Reconfigurability modeling using standard mechanisms of SystemC and a transformation tool to automatically generate SystemC models of the reconfigurable hardware [8].

3.1. Definitions

The terms and concepts specific to the SystemC based approach used in the rest of the paper are defined as follows:

- **Candidate Components:** Candidate components denote those application functions that are considered to gain benefits from their implementation on a reconfigurable hardware resource. The decision whether a task should be a candidate component is clearly application dependent. The criterion is that the task should have two features in combination: flexibility (that would exclude an ASIC implementation) and high computational complexity (that would exclude a software implementation). Flexibility may come either from the point that the task will be upgraded in the future or in view of hardware resource sharing with other tasks with non-overlapping lifetimes for global area optimization.
- **Dynamically reconfigurable fabric (DRCF):** The dynamically reconfigurable fabric is a system-level concept that represents a set of candidate components and the required reconfiguration support functions, which later on in the design process will be implemented on a reconfigurable hardware resource.
- **DRCF component:** The DRCF component is a transaction-level SystemC module of the DRCF. It consists of functions, which mimic the reconfiguration process, and the instances of SystemC modules of the candidate components to present their functionality during system-level simulation. It can automatically detect reconfiguration request and trigger the reconfiguration process when necessary.
- **DRCF template:** The DRCF template is an incomplete SystemC module, from which to create the DRCF component.

3.2. Estimation approach to support system analysis

System analysis is mainly the phase to make HW/SW partitioning and the initial architecture decision. In the design of reconfigurable SoC, system analysis also needs to focus on studying the trade-off of performance and flexibility. The estimation approach is developed to support the system analysis in the task of identifying candidate components that are to be implemented in the RTR hardware.

The estimation approach focuses on a reconfigurable architecture in which there is a RISC processor, an FPGA-type RTR hardware unit, and a system bus as a communication channel. It starts from function blocks represented using C-language and it can produce the following estimates for each function block: software execution time in terms of running the function on the RISC core, mappability of the function and the RISC core [18], execution time in terms of running the function on the RTR hardware, and resource utilization of the RTR hardware. The framework of the estimation approach is shown in Figure 3.

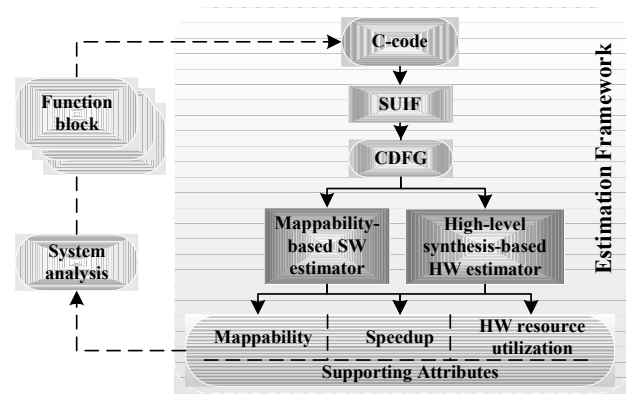


Figure 3. The estimation framework

The starting point is the functional description given in ANSI-C language. The designer decides the granularity of partitioning by decomposing the algorithm down to function blocks. A single function block may then be assigned to SW, RTR hardware or a fixed accelerator. Each of the function blocks will be individually studied and the set of estimation information will be fed into the system-level partitioning phase.

In the estimator, a SUIF-based [19] front-end pre-processor is used to extract Control-Data Flow Graphs (CDFG) from the C code. Then some well-known high-level synthesis tasks [20] are carried out to produce the estimates. As-Soon-As-Possible (ASAP) and As-Late-As-Possible (ALAP) scheduling are used to determine the

critical paths, based on which we estimate the execution time. A modified version of Force-Directed Scheduling (FDS) [21] is used to estimate the hardware resources required for the computation units and the storage units of the tasks. Finally, allocation algorithms are used to estimate the hardware resources required for interconnections with multiplexer types of interconnection units. The current estimator targets a Virtex2-like FPGA [1] in which the main resources are LookUp-Tables (LUTs) and multipliers.

The ultimate goal of the estimation approach is to make candidate component selection, which is an application-dependent procedure. In current design framework, the selection is carried out manually based on designers' experience and design constraints. When global resource saving is an issue, the resource estimates are important inputs. However, to make justified decisions, other information, such as power consumption and task dependence should be included as inputs. Current approach does not include automated tools to support the power and the task dependence analysis.

3.3. Modeling of RTR hardware and the supporting transformation tool

The modeling of reconfiguration overhead is divided into two steps. In the first step, different technology-dependent features are mapped onto a set of parameters, which are the size of the configuration data, the clock speed of configuration process and the extra delays apart from the loading of the configuration data. In the second step, a parameterized SystemC module that models the behavior of run-time reconfiguration process is created. It has the ability to automatically capture the reconfiguration request and present the reconfiguration overhead during performance simulation. Thus, designers can easily evaluate the trade-offs between different technologies by tuning the parameters.

A general model of RSoC is shown in Figure 4. The left hand side depicts the architecture of the RSoC. The right hand side shows the internal structure of the DRCF component.

The DRCF component is a single hierarchical SystemC module, which implements the same bus interfaces as other HW/SW modules do. A configuration memory is modeled, which could be an on-chip or off-chip memory that holds the configuration data. Each candidate component ($F1$ to F_n) is an individual SystemC module that implements the top-level bus interfaces with separate system address space. The Input Splitter (IS) is an address decoder and it manages all incoming Interface-Method-Calls (IMCs). The Configuration Scheduler (CS) monitors the operation states of the candidate components and controls the reconfiguration process.

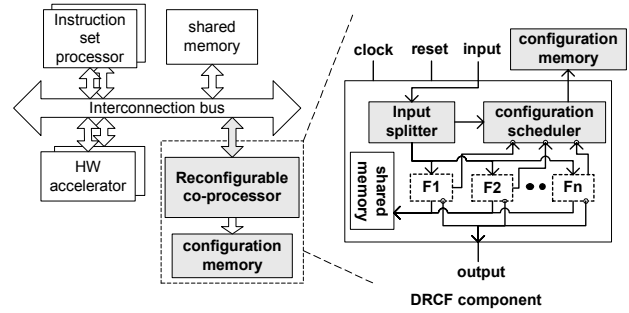


Figure 4. A generic model of RSoC

The DRCF component works as follows. When the IS captures an IMC to a candidate component, it will hold the IMC and pass the control to the CS, which decides if reconfiguration is needed. If so, the CS will call a reconfiguration procedure that uses the parameters specified in the first step to generate the memory traffic and the associated delays to mimic the reconfiguration latency. If the CS detects the RTR hardware is loaded with another module, a request to reconfigure the target module will be put into a FIFO queue and the reconfiguration will be started after the RTR hardware has no running module. After the CS finishes the reconfiguration loading, the IS will dispatch the IMC to the target module. This is a generic description of the context switching process, and designers can develop different CS models when different types of RTR hardware are used such as partial reconfiguration or multi-context device.

The context switching with pre-emption is a common approach in operating systems, the implementation of which doesn't introduce too much overhead because of the regularity of the register organization in GPP. In the DRCF component, the pre-emption technique is not supported because of the very high implementation costs of context switching.

The current modeling method is for non-blocking IMCs. For blocking IMCs, semaphore-based techniques and OS models are needed in the SW side, which are not directly supported in current version of SystemC. These models will be developed when language supports are available in the future.

In order to reduce the coding effort, we have developed a tool that can automatically transform SystemC modules of the candidate components, which however must follow a pre-defined coding pattern, into a DRCF component. The inputs are SystemC files of a static architecture and a script file, which specifies the names of the candidate components and the associated design parameters such as configuration latency. The tool contains a hard-coded DRCF template. It first parses the input SystemC code to locate the declarations of the

candidate components. Then the tool creates a DRCF component by filling the DRCF template with the declarations and making the appropriate connections. Finally, in the top-level structure, the candidate components are replaced with the generated DRCF component. During simulation, data related to reconfiguration latency will be automatically captured and saved in a text file for analysis. A VCD (Value Change Dump) file will also be produced to visualize the reconfiguration effects.

3.4. Link to low-level design

The low-level design is divided into detailed design and implementation design. The output of the detailed design is the intermediate representation of the system, in which SW is represented as C or assembly code and HW is represented as RTL-HDL code. The implementation is the phase where binary code for SW, bitstream for RTR HW and layout for ASICs are generated.

In our approach, automatic code generation for low-level design is not provided and designers should manually or using other tools to transform the SystemC representation of the reconfigurable system to low-level code such as C code for SW implementation and VHDL code for HW implementation. The implementation of the reconfiguration is technology-dependent and is outside the scope of the design methodology.

In our work, we used the Dynamic Circuit Switching (DCS)-based technique [22] to carry out the cycle-accurate co-simulation between the functions mapped onto the RTR hardware and the functions mapped onto the static part of the system. A VHDL module for each of the function mapped onto the RTR hardware is manually created. Multiplexers and selectors are inserted after the outputs of the modules and before the inputs of the modules. They are automatically switched on or off according to the configuration status. In the cycle-accurate simulation model, the reconfiguration is modeled as pure delay.

4. A WCDMA detector case study

We selected a WCDMA detector design case to validate the SystemC-based approach. We targeted on an RTR-type of implementation and the implementation platform was the VP20FF1152 development board from Memec Design group [23], which contains one Virtex2P XC2VP20 FPGA [1].

4.1. System description

The whole WCDMA base-band receiver system is depicted in Figure 5. The case study focuses on the

detector portion (shaded area in Figure 5) of the receiver and a limited set of the full features were taken into account. The detector case used 384 kbits/s user data rate without handover.

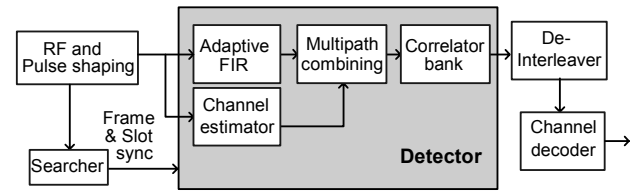


Figure 5. The WCDMA base-band receiver system

The detector contains an adaptive filter, a channel estimator, a multi-path combiner and a correlator bank. The adaptive filter is performing the signal whitening and part of the matched filtering that are traditionally implemented with the RAKE receiver. The channel estimator module calculates the phase references. In the combiner part, the different multi-path chip samples are phase rotated according to the reference samples and combined. Finally the received signal is de-spread in the correlator bank. When compared to traditional RAKE based receiver concepts, this WCDMA detector achieves 1 - 4 dB better performance in vehicular and pedestrian channels.

4.2. System-level design

The design started from the C-representation of the system. It contained a main control function and the four computational tasks, which lead to a simple system partition that the control function was mapped onto SW and the rest onto RTR hardware. The estimation tool was used first to produce the resource estimates. The results are listed in Table 1, where LUT stands for look-up table and register refers to word-wide storages. The multiplexer refers to the hardwired 18x18 bits multipliers embedded in the target FPGA.

Table 1. Estimates of FPGA resources required by the function blocks

Functions	LUT	Multiplier	Register
Adaptive filter	1078	8	91
Channel estimator	1387	0	84
Combiner	463	4	32
Correlator	287	0	17
Total	3215	12	224

Based on the resource estimates, the dynamic context partitioning was done as following. The channel estimator

was assigned to one context (1387 LUTs), and the other three processing blocks were assigned to a second context (1078 + 463 + 287 = 1828 LUTs). This partition resulted in both balanced resource utilization and less interface complexity compared to other alternatives.

A SystemC model of a fixed system was then created, which had two purposes in the design. The first was to use its simulation results as reference data, so the data collected from the reconfigurable system could be evaluated. The second purpose was to automatically generate the reconfigurable system model from it via the transformation tool.

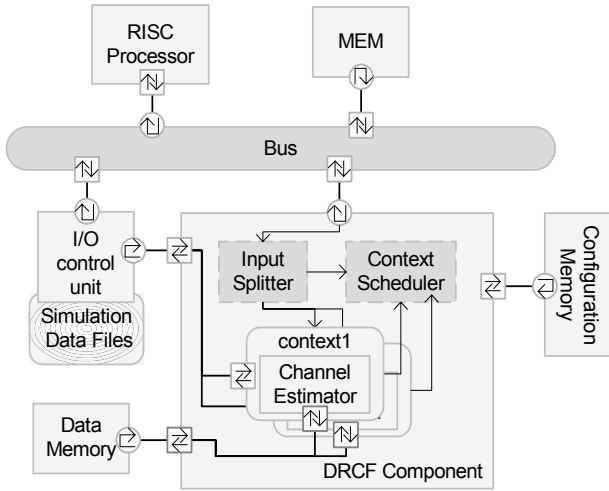


Figure 6. Reconfigurable system model of the WCDMA detector

In the fixed system, each of the four detector functions was mapped to an individual hardware accelerator, and pipelined processing was used to increase the performance. A small system bus was modeled to connect all of the processing units and storage elements. The channel data used in the simulation was recorded in text files, and the processor drove a slave I/O module to read the data from the file. The SystemC models were described at transaction level, in which the workload was derived based on the estimation results but with manual adjustment. The results showed that 1.12 ms was required for decoding all 2560 chips of a slot when the system was running at 100 MHz.

The transformation tool was used to automatically generate the reconfigurable system model, which is depicted in Figure 6, from the fixed model. The reconfiguration latency of the two dynamic contexts was derived based on the assumption that the size of the configuration data was proportional to the resource utilization, the number of LUTs required. The total available LUTs and size of full bitstream were taken from

the Xilinx XC2VP20 datasheet. Some accurate approaches can be used to derive the reconfiguration latency. For example, the latency is related only to the region allocated to the dynamic contexts. In the current work, these have not been studied.

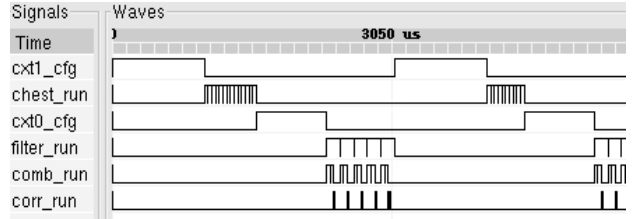


Figure 7. Simulation waveform shows the reconfiguration latency

The performance simulation showed that the system required two reconfigurations per processing each slot of data. This is presented by the *cxt0_cfg* and *cxt1_cfg* in Figure 7. When the configuration clock was running at 33 MHz and the configuration bit-width was 16, the reconfiguration latency was 2.73 ms and the solution was capable of processing 3 slots of data in a frame.

4.3. Detailed design and implementation

Vendor-specific tools were used in the system refinement and implementation phases. The task was divided into interface refinement, configuration design, SW design, and RHW design. Xilinx modular design flow [1] was used in the low-level reconfiguration design at the implementation phase.

Table 2. HW synthesis results

Functions	LUT	Multiplier	Register (bits)
Adaptive filter	553	8	1457
Channel estimator	920	0	2078
Combiner	364	4	346
Correlator	239	0	92

One of the two PowerPC cores in the FPGA was used and C code was manually generated out of the SystemC code. The SW/HW interface was mapped to register-based communication and the direct memory access was used in the SW side. The RTL-VHDL code was manually generated for the four computational tasks, and the Xilinx IP cores were used for other peripherals. The synthesis results are listed in Table 2. When considering the estimation, the results are over-estimated at about 55% in average. The main reasons are the assumption of fixed-length computation and the technique of mapping

multiplexers directly to LUTs [17]. A DCS-based VHDL wrapper was created to enable the cycle-accurate simulation to verify the system behavior.

The reconfiguration was implemented using SystemACE CF solution [1] and the control function of the SystemACE module was inserted into the C code. Modular design flow was used to generate the partial bitstream. The size of the configuration data was 279 KB for the context 1 and 280 KB for the context 2. 21 bus macros were used to connect the static region and the dynamic region, as depicted in Figure 8 showing the routed design graph of the assembled design of one dynamic context and the static context. The total resource utilization is listed in Table 3. The processing time of one slot of data was 9.66 ms, out of which 8.6 ms was associated with the reconfiguration latency.

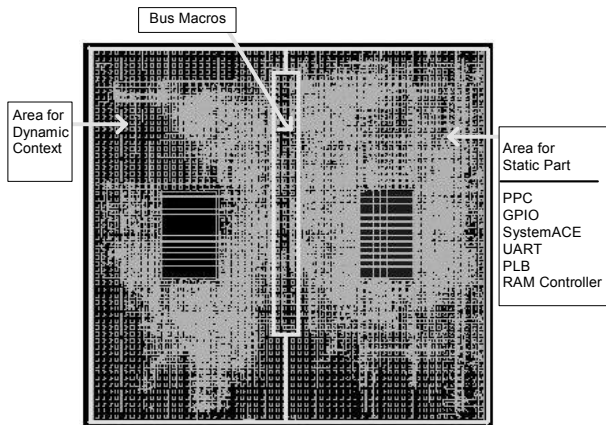


Figure 8. Routed design of one dynamic context and the static context

4.4. Comparison with other approaches

In addition to the implementation of the dynamic reconfiguration approach, a fixed hardware implementation and a pure software implementation were made as reference designs.

In the fixed-hardware implementation, the processing blocks were mapped onto static accelerators and the scheduling task was mapped onto SW that ran on the PPC core. The resource requirements were 4632 LUTs (24% of available resources), 55 Block RAMs (62%) and 12 Block Multipliers (13%). The system was running at 100 MHz. The execution time for processing one slot of data was 1.06 ms. Compared to the fixed reference system, the dynamic approach achieved almost 50% resource reduction in terms of the number of LUTs, but at the cost of 8 times longer processing time.

For the full software implementation, the design was done as a standalone approach and no operating system

was involved. Everything was running in a single PPC core and data were entirely stored in internal BRAMs. For the same clock frequency, the processing time of one slot of data was 294.6 ms, which was over 30 times of the processing time in run-time reconfiguration case. This did not fulfill the real-time requirements.

Table 3. Resource utilization in Xilinx XC2VP20

	LUT	BRAM	MUL	Reg/bit	PPC
static	1199	41	0	1422	1
dynamic	1534	7	12	1855	0
total	2733	48	12	3277	1

5. Analysis and discussion

The main advantage of the SystemC-based approach is that it can be easily embedded into a SoC design flow to allow fast design space exploration for different reconfiguration alternatives without going into implementation. Considering the design at detailed level and implementation level is time consuming, usually taking from weeks to months, the SystemC-based approach can result in remarkable improvements in the design process both from time and quality points of view.

The potential benefit of using the run-time reconfiguration approach is obviously the significant reduction of reconfigurable resources. Compared to a completely fixed implementation, the reduction of LUTs can be up to 50%. Compared to a full software implementation, the run-time reconfiguration approach is over 30 times faster. The commercial off-the-shelf FPGA platform caused limitations on the implementation of run-time reconfiguration. Although the selected approach used partial reconfiguration, the required configuration time affected the performance a lot in the data-flow type WCDMA detector design case. The ratio of computing to configuration time was about 1/8 in this design case.

The WCDMA detector design case is based on sample by sample processing and is not ideal for demonstrating the RTR benefits. It was selected for the validation of the SystemC-based design methodology. Through the design case, the estimation approach and the DRCF modeling approach have shown their usefulness by providing reasonably accurate results without going into low-level implementation. With C code and test data that were available in the WCDMA detector design case, the design at the system-level took only less than a week.

There are a few research directions foreseen at the moment. The SystemC 3.0 is expected to support well the RTOS modeling, and so the DRCF component could be extended to enable new reconfiguration services. Architecture-dependent mapping techniques can be developed in the estimation approach in order to increase

the estimation accuracy. A tool to automatically generate the VHDL wrapper from SystemC code for RTR simulation would further make the approach much easier to use. High-level SystemC synthesis or C-based synthesis could speed up the detailed design remarkably. The current approach does not address the power issue, especially power variance associated with the reconfiguration. Methods and tools for power analysis would be of great interest.

6. Conclusions

The SystemC-based approach to support design of RSoC is described in this paper. The main focus is how to enable fast design space exploration at the system level. The estimation approach to support system analysis, the reconfiguration modeling technique and a tool to support automatic SystemC code generation are provided. Our models work at the transaction level, so the system performance can be quickly evaluated in simulation but without losing too much information details from the system architecture. A design of WCDMA detector design case has been carried out and it has been fully implemented in a real reconfigurable platform. The design case has validated that the SystemC-based approach is very useful in providing the supports to RSoC design at the system level.

7. Acknowledgements

This work was previously supported by the European Commission under the contract IST-2000-30049 ADRIATIC, and is currently supported by Tekes (National Technology Agency of Finland) and VTT under EUREKA/ITEA contract 04006 MARTES.

8. References

- [1] Xilinx, www.xilinx.com
- [2] Altera, www.altera.com
- [3] PACT XPP technologies, www.pactcorp.com
- [4] QuickSilver Technologies, www.qstech.com
- [5] Triscend, "A7 Field Configurable System-on-Chip datasheets", www.triscend.com (2004)
- [6] Motorola, www.motorola.com
- [7] A. Pelkonen, K. Masselos, M. Cupak, "System-Level Modeling of Dynamically Reconfigurable Hardware with SystemC", Proc. IPDPS'03, 2003, pp. 174-181
- [8] Y. Qu, K. Tiensyrjä, K. Masselos, "System-level modeling of dynamically reconfigurable co-processors", Proceedings of the 14th International Conference on FPL (LNCS 3203), 2004, pp. 881-885
- [9] T.J. Callahan, J.R. Hauser, J. Wawrzynek, "The Garp Architecture and C Compiler", IEEE Computer, Vol. 33, Issue. 4, 2000, pp. 62-69
- [10] H. Singh, et al, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications", IEEE Transactions on Computers, Vol. 49, Issue. 5, 2000, pp. 465-481
- [11] S.C. Goldstein, et al, "PipeRench: a reconfigurable architecture and compiler", Computer, Vol. 33, Issue. 4, 2000, pp. 70-77
- [12] R. Maestre, et al, "A Framework for Reconfigurable Computing: Task Scheduling and Context Management", IEEE Transactions on VLSI Systems, vol. 9, issue. 6, 2001, pp. 858-873
- [13] J. Tabero, et al, "A Low Fragmentation Heuristic for Task Placement in 2D RTR HW Management", FPL04 (LNCS 3203), 2004, pp. 241-250
- [14] C. Steiger, H. Walder, M. Platzner, "Operating Systems for Reconfigurable Embedded Platforms: Online Scheduling of Real-Time Tasks", IEEE Transactions on Computer, Vol. 53, No. 11, 2004, pp. 1393-1407
- [15] K. Tiensyrjä, et al, "SystemC and OCAPI-XL Based System-Level Design for Reconfigurable Systems-on-Chip". Forum on Specification & Design Languages (FDL 2004), 2004, pp. 428-439
- [16] T. Grötter, et al, System Design with SystemC. Kluwer Academic Publishers, Boston, May 2002, 240 pp.
- [17] Y. Qu, J.-P. Soininen, "Estimating the utilization of embedded FPGA co-processor", Euromicro Symposium on Digital Systems Design (DSD 2003), 2003, pp. 214 - 221
- [18] J.-P. Soininen, et al, "Mappability estimation approach for processor architecture evaluation", Proceeding of 20th IEEE Norchip Conference, 2002, pp. 171-176
- [19] R.P. Wilson, et al, "SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers". Proceedings of the 7th ACM SIGPLAN symposium on Principles and practice of parallel programming, 1994, pp. 37-48
- [20] D.D. Gajski, et al, "High-level synthesis: Introduction to chip and system design", Kluwer Academic Publishers, Boston, 1997
- [21] P.G. Paulin, J.P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASICs", IEEE Transactions on CAD of Integrated Circuits and Systems, Vol. 8, No. 6, 1989, pp. 661-679
- [22] P. Lysaght, J. Stockwood, "A simulation tool for dynamically reconfigurable field programmable gate arrays", IEEE Transactions on VLSI Systems, vol. 4, issue. 3, 1996, pp. 381-390
- [23] Memec, www.memec.com