

SystemCoDesigner: Automatic Design Space Exploration and Rapid Prototyping from Behavioral Models

Christian Haubelt, Thomas Schlichter,
and Joachim Keinert
University of Erlangen-Nuremberg, Germany
haubelt@cs.fau.de

Mike Meredith
Forte Design Systems, San Jose, USA
mmeredith@ForteDS.com

ABSTRACT

SystemCoDesigner is an ESL tool developed at the University of Erlangen-Nuremberg, Germany. SystemCoDesigner offers a fast design space exploration and rapid prototyping of behavioral SystemC models. Together with Forte Design Systems, a fully automated approach was developed by integrating behavioral synthesis into the design flow. Starting from a behavioral SystemC model, hardware accelerators can be generated automatically using Forte Cynthesizer and can be added to the design space. The resulting design space is explored automatically by optimizing several objectives simultaneously using state of the art multi-objective optimization algorithms. As a result, SystemCoDesigner presents optimized hardware/software solutions to the designer who can select any of them for rapid prototyping on an FPGA basis. Thus, SystemCoDesigner bridges the gap from ESL to RTL and increases the confidence in early design decisions.

Categories and Subject Descriptors

J.6 [Computer-Aided Engineering]: Computer-aided design (CAD); C.3 [Special-Purpose and Application-Based Systems]: Real-time and embedded systems

General Terms

Algorithms, Design

Keywords

ESL Design, Design Space Exploration, Rapid Prototyping

1. INTRODUCTION

Design decisions taken at the Electronic System Level (ESL) have by far the most significant impact on the design quality of the final product. As most design decisions are taken in early design phases at high levels of abstraction, there is a need for methodologies and tool support helping the designer to select the best design alternatives. Hence, the next step in Electronic Design Automation (EDA) will target ESL methodologies. Emerging ESL tools are expected to decrease design times by a factor of 10 while improving design quality by a factor of 10. However, the main obstacle to reach this productivity gain can be identified as the design gap from ESL to RTL (Register Transfer Level).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC 2008, June 8–13, 2008, Anaheim, California, USA
Copyright 2008 ACM ACM 978-1-60558-115-6/08/0006 ...5.00

Several commercial *behavioral* or *high-level synthesis* tools are available, e.g., *Cynthesizer* by Forte Design Systems [5], *CatapultC* by Mentor Graphics [18], and *NEC's CyberWorkBench* [21]. Moreover, tools for automatic code generation for embedded processors exist [15, 20]. However, efficient synthesis approaches from ESL models to hardware/software implementations are still missing.

To overcome the limitations of currently available tools, SYSTEMCODESIGNER has been developed at the University of Erlangen-Nuremberg, Germany. In order to handle the complexity of complete systems, it uses an actor-oriented approach which is a widely accepted paradigm in system level synthesis [14, 13]. Here, SystemC [6] is going to become an industry de facto standard which is also used by SYSTEMCODESIGNER. This allows for integration of Forte's behavioral synthesis tool called Cynthesizer into its ESL design space exploration. That way, SYSTEMCODESIGNER is the first tool which reduces the number of manual steps as much as possible and provides a correct-by-construction generation of rapid prototypes from a behavioral model. This is achieved by starting the design process from an executable SystemC behavioral model. Integration of Forte Cynthesizer allows for automatic generation of hardware accelerators for each SystemC module. Furthermore it is possible to compile the modules of the SystemC model for embedded processor cores. By help of an automatic design space exploration, these implementations are assembled to an overall system which can be optimized in terms of throughput, latency and resource requirements. The designer can then select those solutions which fit best his requirements. Automatic generation of a configuration bitstream for FPGAs allows for fast prototyping. Consequently, this significantly simplifies the comparison of different implementation alternatives of the SystemC model not only by simulation, but also on a concrete hardware platform.

The rest of the paper is structured as follows: After presenting related work in Section 2, an overview of the SYSTEMCODESIGNER design flow is given in Section 3. Section 4 discusses the automatic generation of hardware accelerators using Forte Cynthesizer. Section 5 is devoted to design space exploration and rapid prototyping as implemented in SYSTEMCODESIGNER. Finally, Section 6 presents the results obtained by applying our proposed methodology to a Motion-JPEG decoder example using SYSTEMCODESIGNER and Cynthesizer. As a result, it can be seen that SYSTEMCODESIGNER can increase the confidence in early design decision by providing tight estimations on the final design quality at a high level of abstraction and by requiring only a minimal user interaction.

2. RELATED WORK

Tools providing ESL design space exploration are *Sesame* [23], *MILAN* [19], *CHARMED* [11], and *WIZARD* [2]. Furthermore, several publications in the area of design space exploration exist using different optimization strategies and considering different objectives [12, 17]. However, all these approaches are not able to au-

tomatically generate a prototype of an optimized implementation.

An approach called *Koski* supporting automatic design space exploration and synthesis is presented in [10]. The input specification is given as a Kahn process network modeled in UML. The Kahn processes are refined using Statecharts. The target architecture consists of platform-dependent and platform-independent software as well as synthesizable communication and processing resources. Another approach for automatic mapping of applications to FPGA platforms is the *Daedalus* design flow [28]. It automatically converts a Matlab or C loop program into a Kahn process network. This process network can be transformed into a hardware/software system by instantiating processors and IP cores and connecting them with FIFOs. An automatic design space exploration is supported. Finally, in [7], the *PeaCE* approach is presented. Starting from a Ptolemy application model, it provides a seamless co-design flow from functional simulation to system synthesis. Moreover, PeaCE supports the generation of an FPGA prototype. The target architecture is basically a multi-processor system. All three, *Koski*, *Daedalus*, and *PeaCE* do not support automatic generation of hardware accelerators by means of a behavioral synthesis.

The System-on-chip Environment (SCE) [1] developed at the Center for Embedded Computer Systems allows for automatic refinement of SpecC behavioral models. Besides hardware synthesis and embedded software synthesis, SCE also supports communication synthesis. Behavioral synthesis to generate hardware accelerators can be included in the design flow. However, in contrast to SYSTEMCODESIGNER, the automatic design space exploration is excluded, i.e., the mapping of so called SpecC behaviors to processors and IP cores has to be done manually.

Finally, a tool called *Cascade* is provided by CriticalBlue [3]. Starting from C/C++ or assembler code, Cascade generates hardware accelerators and corresponding interfaces to the processor core. However, in contrast to SYSTEMCODESIGNER, the generated accelerators are more fine-grained as these accelerators are expected to replace single assembler instructions.

3. DESIGN FLOW

The overall design flow of the SYSTEMCODESIGNER ESL design methodology is based on (i) a behavioral SystemC model of an application, (ii) generation of hardware accelerators for some or all SystemC modules using behavioral synthesis, (iii) determination of their performance parameters like required hardware resources, throughput and latency or estimated software execution times, (iv) design space exploration for finding the best candidate architectures, and (v) rapid prototype generation for FPGA platforms. Figure 1 shows the design flow implemented in SYSTEMCODESIGNER.

The first step in our ESL design flow is to describe the application in form of a SystemC behavioral model. In order to allow as much automation as possible, we restrict the application domain to multi media and networking, i.e., streaming applications, and require the SystemC model to be written using the SYSTEMOC library [4]. In a SYSTEMOC description each SystemC module is defined by a finite state machine specifying the communication behavior and methods controlled by the finite state machine. These methods are executed atomically and data consumption and production is only done after computing a method. Hence, SYSTEMOC resembles *FunState* (Functions driven by State machines) [25] and realizes a rule-based model of computation [22].

As an example, Figure 2(a) shows a Motion-JPEG decoder in SYSTEMOC. It consists of several modules also called *actors* processing a stream of data. These modules are interconnected by edges representing communication. The latter one is realized by special FIFO channels with peak and poke operations. The *Parser* analyzes the processed JPEG stream and extracts important control information like image dimensions and quantization strengths. The

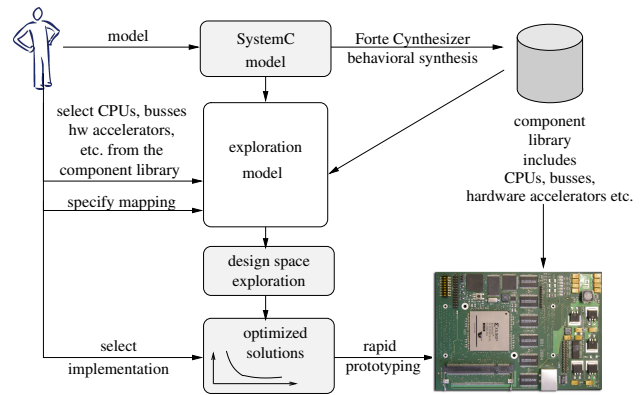


Figure 1: ESL design flow using SYSTEMCODESIGNER: The SystemC behavioral model of the application is used for both automatic design space exploration and creation of hardware accelerators allowing for rapid prototyping.

Huffman Decoder, the zero-run length decoder (*InvZrl*) and the *DC-Decoder* are responsible for entropy decoding. The resulting 8×8 frequency blocks are transformed into pixel values by help of the three following actors. The *Frame Shuffler* is responsible for re-ordering the pixels arriving block by block into a raster scan order. The *YCbCr Decoder* finally converts the images into the RGB color space before they are stored as *Portable Pixmap Files* [24].

In the case that an application is already available in form of a SystemC specification, the latter one can be directly transformed into a SYSTEMOC model. This is important in order to allow for automatic analysis and optimization as well as for automatic performance evaluation during design space exploration as described in Section 5. In order to transform a SystemC application into a SYSTEMOC description, the input SystemC application is required to only communicate via SystemC FIFOs and that its functionality is implemented via a single SystemC thread (`SC_THREAD`). Then a *control data flow graph* (CDFG) of the method defining the thread can be established in a similar way as done by any C++ parser. In order to increase the level of abstraction, all subgraphs of the CDFG not containing any read or write CDFG nodes can be collapsed into a single node. Finally, we collect sequences of read or write operations on a given port into a single node and transitions are annotated with the number of read or written tokens. This is shown in Figure 2(b) (SystemC module) and Figure 2(c) (SYSTEMOC actor). The finite state machine controlling the communication behavior of the SYSTEMOC actor checks for available input data (e.g., $\#i_1 \geq 1$) and available space on the output channels (e.g., $\#o_1 \geq 1$) to store results. Furthermore, constant methods called *guards* (e.g., `check`) can be used to test values of internal variables and data in the input channels. If the predicates annotated to a state transition evaluate to true, this transition can be taken and the annotated methods called *action* (e.g., `transform`) will be performed atomically. These actions can access the values of the SYSTEMOC FIFOs by help of the bracket operators indicating a read or write offset as shown in Figure 2(c).

Each SYSTEMOC actor can then be transformed into both hardware accelerators and software modules. Whereas the latter one is achieved by simple code transformations, the hardware accelerators are built by the help of Forte Synthesizer [5]. This allows for quick extraction of important performance parameters like the achieved throughput and the required area by a particular hardware accelerator. In case a Xilinx FPGA is the target platform, hardware resources in form of flip flops, look-up tables, and block RAMs are estimated. These values can be used to evaluate different solutions found during automatic design space exploration.

The performance information together with the executable spec-

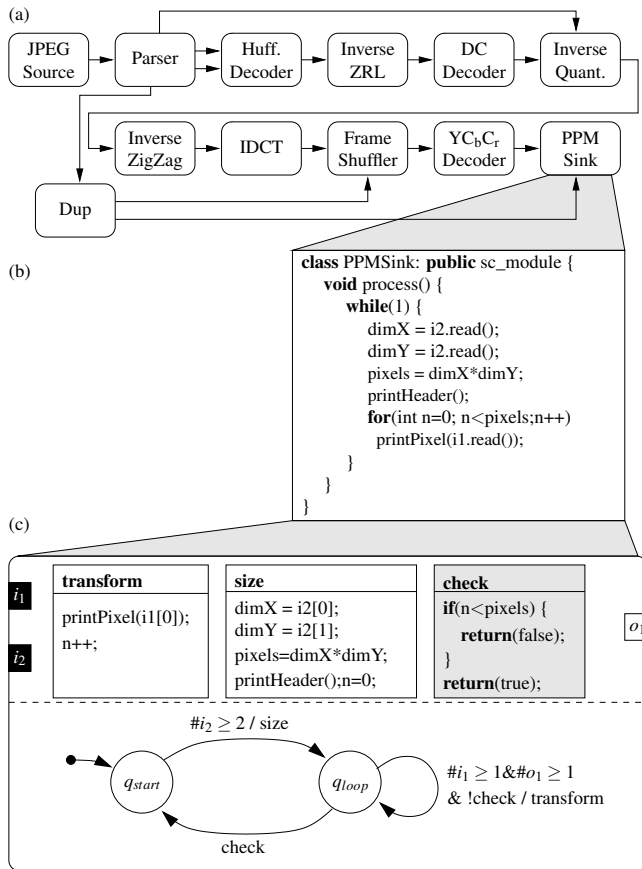


Figure 2: Block diagram of a Motion-JPEG decoder. Each block corresponds to a task performing a particular operation. Communication is illustrated by edges. b) shows an extract of the PPM SystemC code which can be transformed into the SYSTEMOC finite state machine given in c).

ification and a so called *architecture template* serves as the input model for design space exploration. The architecture template is represented by a graph which contains all possible hardware accelerators, processors, memories, and the communication infrastructure from which the design space exploration has to select those ones which are necessary in order to fulfill the user requirements in terms of overall throughput and chip size. The fewer components are allocated, the fewer hardware resources are required. This, however, in general comes along with a reduction in throughput, thus leading to tradeoffs between execution speed and implementation costs. Multi-objective optimization together with symbolic optimization techniques [16] are used to find sets of optimized solutions.

From this set of optimized solutions the designer selects a hardware/software implementation best suited for his needs. Once this decision has been taken, the last step of the proposed ESL design flow is the rapid prototyping of the corresponding FPGA-based implementation. As we use an actor-oriented application modeling, complex optimized implementations can be assembled by interconnecting the hardware accelerators and processors with special communication modules provided in a component library. Furthermore, the program code for each microprocessor is generated. Finally, the entire platform is compiled into an FPGA bit stream using, e.g., the Xilinx Embedded Development Kit (EDK) [29] tool chain for Xilinx FPGAs.

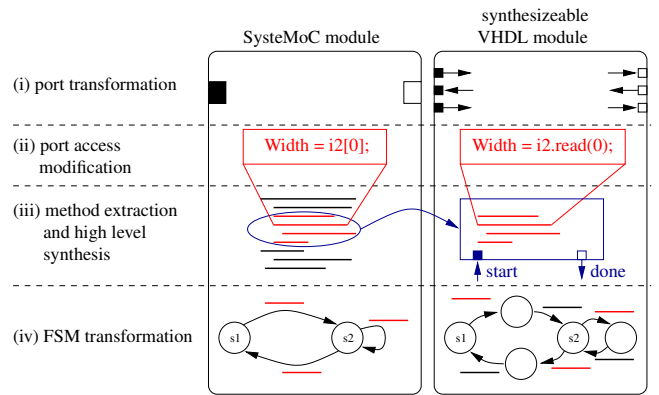


Figure 3: Transformation steps of a SYSTEMOC actor to a synthesizable SystemC module.

4. AUTOMATIC ACTOR SYNTHESIS

After creation of a SYSTEMOC behavioral model for the considered application, the next step in the SYSTEMCODESIGNER design flow consists in the automatic generation of hardware accelerators for some or all SYSTEMOC actors. This is an important step for extraction of performance parameters in form of execution times, area values, and other non-functional properties. These are required for the evaluation of each implementation during automatic design space exploration.

SYSTEMCODESIGNER uses Forte Synthesizer for this hardware synthesis step. As all behavioral synthesis tools typically do not support all possible SystemC language constructs, e.g., recursion, dynamic memory allocation, and thus usage of dynamic data structures, only SYSTEMOC modules obeying these restrictions can be synthesized to hardware.

The transformation of a SYSTEMOC actor into a hardware accelerator is done in four steps which are depicted in Figure 3. These steps are: (i) port transformation, (ii) port access modification, (iii) method extraction and behavioral synthesis, and (iv) FSM transformation.

The *port transformation* replaces each SYSTEMOC port interfacing an abstract SYSTEMOC communication channel by a SystemC hardware signal port. This is necessary to be able to drive the hardware signals of the communication primitives provided in our component library (see Figure 1) which is a prerequisite for assembling any hardware/software implementation later on. These communication primitives are hardware implementations of the SYSTEMOC channels which allow for peak and poke operations (see Section 3).

The *port access modification* replaces all port accesses occurring in the actions and guards by method calls (cf. Figure 2(b) and (c)). These methods are implemented by the SystemC hardware signal ports and drive the signals of the hardware implementations of the SYSTEMOC channels. As, however, the latter ones demand a communication protocol requiring at least one clock cycle, simple sequential method calls would result in bad system performance. Consequently, to allow concurrent writes to different hardware ports, we create processes for each SYSTEMOC output port that handle the hardware protocol. Concurrent reads from input ports are implemented by splitting the read access into two functions: the first function only starts the read access and does not wait for a clock edge, whereas the second function waits for the result from the input port.

The *method extraction and behavioral synthesis* phase copies the corresponding methods from the SYSTEMOC description into a SystemC module. Each of these methods can be started by help of a `start` signal which is generated by the controlling state machine described later on. Furthermore, the SystemC module provides a

done signal indicating that the execution of the method has been terminated. As SYSTEMOC actions cannot be executed in parallel, Forte Synthesizer is able to share resources between them thus leading to reduced resource requirements. SYSTEMOC guard functions on the other hand are allowed to execute in parallel if necessary in order to make the decision which transition of the finite state machine to take as fast as possible. To generate alternative hardware accelerators with different properties in speed and size, the designer can add constraints for the behavioral synthesis which translates the SystemC module into RTL code.

Finally, the *FSM transformation* automatically translates the abstract SYSTEMOC syntax of the finite state machine into a VHDL description. This finite state machine implementation is the controller process of all synthesized methods in the previously generated SystemC module using the *start* and *done* signals. Within each state of the finite state machine, the predicates of all outgoing transitions as for instance the number of available data and available space as well as guard functions are checked in parallel. Possible conflicting port accesses by different guard-functions are resolved. Data caching allows reducing the access time to repeatedly read FIFO data values. If a transition is taken, the corresponding method is called using the *start* signal. After receiving the *done* signal, the requested data is removed from the input channels and made visible on the output channels. Furthermore, the state variable of the module is set to the next state.

After termination of these steps, a VHDL file is generated which instantiates both the controlling FSM process as well as the RTL code of the SystemC module generated by Forte Synthesizer. Then, this VHDL file is synthesized using *Synplify Pro* from *Synplify* [27]. From the synthesis reports the information required for automatic design space exploration like method execution delays, required look-up tables (LUT), flip flops (FF), and block RAMs (BRAM) can be extracted.

5. AUTOMATIC DESIGN SPACE EXPLORATION AND RAPID PROTOTYPING

Having the application modeled in SYSTEMOC on the one hand, and the synthesized hardware accelerators for each SYSTEMOC actor on the other hand, an automatic design space exploration (DSE) can be performed. The latter one explores optimal (or near optimal) solutions in terms of throughput, latency or required chip size by allocating processors, memories, buses, and hardware accelerators and binding the SYSTEMOC actors and channels to the resources.

The first step is to formalize the particular instance of this system synthesis problem by providing a so called *architecture template*. The latter one specifies all possible hardware accelerators, processors, memories, buses as well as their interconnection. From this set, the automatic design space exploration has to select a subset in order to form an implementation. The target architecture in principle allows for hardware only, software only, and mixed hardware/software designs of an application

Once the designer has decided about the possible architecture template for the considered application, a formal model has to be built which serves as input to the design space exploration. It consists of three parts, namely (i) the application itself, (ii) the architecture template, and (iii) the mapping constraints. For each possible mapping, we annotate the action execution times of a SYSTEMOC actor occurring when the actor is bound to the hardware resource identified by the mapping edge. The action-accurate delay annotation improves the throughput and latency estimations, when the execution times for the different actions and hence actor invocations are not identical. The hardware sizes are directly associated to the hardware resources of the architecture model.

As soon as the designer has set up the formal model, the automatic design space exploration can be executed. It searches for

optimal implementations, i.e., an allocation of hardware resources and a binding of SYSTEMOC actors and channels onto these hardware resources. The optimization is guided through multi-objective evolutionary algorithms [16]. As a result of the design space exploration, we obtain several optimized or *non-dominated* implementation alternatives approximating the set of Pareto-optimal solutions. The evaluation of a concrete implementation bases on several *objectives*, like latency, throughput and hardware costs in form of required LUTs, flip flops and BRAM modules (for FPGA targets).

Whereas the necessary hardware cost for FPGA platforms can be determined by simply summarizing the resource characteristics from behavioral synthesis and IP core data sheets, throughput and latency calculation is more complex. Due to the data-dependent behavior of many applications, the timing information can only be obtained by simulation of the application using system level performance models which take the binding to hardware accelerators, softcore processors, and buses into account. This is established by the *Virtual Processing Components* (VPC) framework [26] which simulates a particular architecture modeled in SystemC in order to derive average case estimations for latency and throughput. In order to allow for fast simulation, VPC builds a precompiled simulation binary which is able to simulate all possible bindings for the chosen architecture template. Thus, the allocation and binding can be loaded at simulation startup by simple configuration instead of generating multiple simulation binaries, thus leading to fast exploration times. Each time an action of an actor is executed, the execution time needs to be simulated together with effects resulting from resource contention. For this purpose, a scheduling policy for each resource has to be defined. Preemptive scheduling is established by a sophisticated event protocol implemented in the VPC framework [26].

After termination of the design space exploration, automatic rapid prototyping is the last step in the SYSTEMCODESIGNER design flow (see Figure 1). Starting from the results obtained by the automatic design space exploration, it performs all steps required to generate an FPGA configuration file by interconnecting the previously generated hardware accelerators and softcore processors with the required communication primitives.

The rapid prototyping itself works fivefold: First, for each allocated CPU resource, a MicroBlaze subsystem including memory and bus resources is instantiated. Secondly, the allocated hardware accelerators are automatically inserted from a component library as Verilog netlists. The latter ones might either have been generated manually in order to allow for hand made optimizations, or they are automatically derived from the SYSTEMOC model as explained in Section 4. Next, the communication resources are inserted from the component library which provides different implementations of the abstract SYSTEMOC FIFO. Two different kinds of this FIFO are available, one using block RAMs for the internal buffer and the other one distributed RAM on Xilinx FPGAs. The size of the FIFO is directly specified in the SYSTEMOC model. In the case of hardware software communication, special *bridges* are inserted connecting the SYSTEMOC FIFO to the MicroBlaze FSL links. The result of these three steps is a hardware description file (*mhs*-file in case of the Xilinx Embedded Development Kit [29]).

In the fourth step, the code required for each MicroBlaze processor is generated by an automatic transformation of the SYSTEMOC actors into C++ code (see also [8]). If several actors are bound to the same processor, a simple round-robin schedule policy is deployed. Software to software communication is implemented via a special software library allowing the exchange of data within the MicroBlaze by reads and writes to local memory buffers. Finally, the platform specific bit stream is generated by using several Xilinx synthesis tools performing place and route and software compilation.

Table 1: Motion JPEG development effort

Development activity	Person days
Specification and interface definition	4
Module implementation	16
Integration	3
Debugging	11
Code Adaption for Synthesis	4

6. RESULTS

For the experimental evaluation of the SYSTEMCODESIGNER design flow, we use the example of a Motion-JPEG decoder shown in Figure 2(a). The Motion-JPEG decoder case study consists of 8,000 SYSTEMOC lines of code, supporting interleaved and non-interleaved baseline profile without sub-sampling. Table 1 illustrates the development effort spent for realization of the complete Motion-JPEG decoder and its different hardware/software implementations. The first item represents the activity of breaking the JPEG standard [9] down into a graph of actors. Module implementation encompasses the encoding of the actor’s communication state machine as well as the corresponding actions and guards. During the integration phase, the actors have been connected to the complete system. Thanks to the actor-oriented methodology, this step could be accomplished very quickly, as the interface specification is relatively simple due to the applied FIFO communication semantics. Then debugging has been required in order to detect errors in both the specification and the module implementations. Finally, coding constructs not supported by the behavioral synthesis tool (see Section 4) have been identified and replaced by equivalent instructions.

For the exploration of the Motion-JPEG example we created an architecture template using one MicroBlaze softcore processor, 224 FIFO links and 19 modules generated by behavioral synthesis. For the hardware SYSTEMOC FIFOs connecting two hardware IP cores, two alternative implementations have been taken into account, one using BRAM, the other distributed RAM. The complete specification includes 319 mapping edges for the actors resulting in about $5 \cdot 10^{33}$ possible implementation alternatives. Thanks to the integration of Forte Cynthesizer, the hardware accelerators for the different actors could be obtained directly from the SYSTEMOC specification. Hence, whereas traditional system design mostly requires the creation of both a so called *golden model* and the corresponding RTL implementation, behavioral synthesis helped avoiding this redundant effort. Furthermore, as SYSTEMOC offers a higher level of abstraction compared to RTL, the designer can progress more quickly. Taking the number of lines of code as a measure for complexity, we figured out that RTL design would have been 8-10 times more costly than the SYSTEMOC specification.

Determination of precise method execution times for both hardware and software implementations is an important prerequisite for accurate performance evaluation. For this purpose, we have implemented a hardware profiler running on the FPGA which observes the program counter of the deployed MicroBlaze processor. The latter one offers a corresponding interface making this approach feasible. The hardware profiler obtains the entrance and exit points of each module method which can be obtained by special compiler tools. By observation of the program counter, the hardware profiler can detect when the processor starts, respectively terminates the execution of a method. Together with a counter acting as a clock device, the hardware profiler can derive the execution time spent in the considered action.

With the SYSTEMOC application, the architecture template, and the corresponding mapping constraints, the design space has been explored using SYSTEMCODESIGNER. The objectives taken into account during design space exploration have been (i) throughput,

Table 2: Results of a SYSTEMCODESIGNER design space exploration on a Linux workstation with a 2400 MHz Intel Core2 CPU and 3GB of RAM.

parameter	value
solutions evaluated	7,600
non-dominated solutions	366
exploration time	2d17h46min
simulation time	30.44s/solution

Table 4: Influence of the MicroBlaze cache for software only solutions

	Processing time for four images
without cache	146.3 s
with cache	42.9 s

(ii) latency, (iii) number of required flip flops, (iv) look-up tables and (v) block RAMs resp. multipliers¹. Table 2 gives the results of a single exploration run for the Motion-JPEG decoder. The exploration has been stopped after 300 iterations, which corresponds to a run time of 2 days, 17 hours and 46 minutes.² The simulation time per solution is about 30 seconds for Motion-JPEG streams consisting of four color QCIF frames with 176×144 pixels each. As a result, 366 non-dominated solutions were found, each of them representing an arbitrary hardware/software implementation.

Table 3 provides the performance values and resource utilization of different implementations found during the design space exploration. We used the previously described rapid prototyping approach to generate the different implementations, in order to determine these numbers. The estimated values during design space exploration are given in parenthesis. Latency and throughput are estimated using the VPC simulation decoding a JPEG stream consisting of four color QCIF frames (176×144 pixels).

The differences in the required hardware sizes occurring between the predicted values during automatic design space exploration and those measured in hardware can be explained by post synthesis optimization like elimination of useless BRAMs as well as by the influence of the MicroBlaze configuration, in particular the number of FSL links (see also [8]). The discrepancy between the VPC estimations for latency and throughput and those measured for hardware-software solutions is due to schedule overhead. The underlying reason is that VPC deploys an event-based simulation where the evaluation of guards is performed in zero-time by the simulation kernel, which is not true for the final hardware implementation. Furthermore, we observed a tremendous influence of the cache enabled on the MicroBlaze processor.

This is illustrated in Table 4 by help of a software-only solution. It shows the overall execution time for processing four QCIF frames. As it can be seen, the simple enabling of both an instruction and a data cache with 16 kBytes each results in a performance gain of factor 4. This is due to the fact that both instruction code and program data is stored on an external memory with relatively huge access times. As the cache behavior, however, depends on the program execution history, a simple re-ordering of the program code can lead to significant changes in the action execution times which cannot be taken into account by the VPC framework.

¹A Xilinx Virtex II FPGA (XC2V6000) has been selected as target platform for the implementations running at a clock frequency of 50 MHz.

²Each iteration corresponds to the full evaluation of a full population of several so called individuals where each individual represents a specific hardware/software solution of the Motion-JPEG example.

Table 3: Hardware synthesis results and estimations used during design space exploration

No. of SW Actors	Latency	Throughput	LUTs	FFs	BRAM
0	15.63 ms (12.61 ms)	65.0 fps (81.1 fps)	40 467 (44 878)	14 508 (15 078)	47 (72)
1	23.49 ms (25.06 ms)	43.0 fps (40.3 fps)	35 033 (41 585)	11 622 (12 393)	72 (96)
8	6 275 ms (4 465 ms)	0.16 fps (0.22 fps)	15 064 (17 381)	7 540 (8 148)	63 (63)
all	10 030 ms (8 076 ms)	0.10 fps (0.13 fps)	1 893 (2 213)	1 086 (1 395)	29 (29)

7. CONCLUSIONS

In this paper, we presented SYSTEMCODESIGNER, an ESL tool developed at the University of Erlangen-Nuremberg, Germany. SystemCoDesigner offers a fast design space exploration and rapid prototyping form behavioral SystemC models. In order to automate both design space exploration as well as rapid prototyping, the behavioral model has to be written using a special SystemC library called SYSTEMOC that separates communication behavior from data transformations. Together with Forte Design Systems, a fully automated approach was developed by integrating behavioral synthesis into SYSTEMCODESIGNER's design flow. Forte Cynthesizer is used to automatically generate hardware accelerators that can be added to the design space. As a result of the design space exploration, SYSTEMCODESIGNER presents optimized hardware/software solutions to the designer who can select any solution for rapid prototyping. We have shown an example of a Motion-JPEG decoder specified as SystemC behavioral model which was automatically optimized and prototyped using SYSTEMCODESIGNER and Cynthesizer.

8. REFERENCES

- [1] S. Abdi, J. Peng, H. Yu, D. Shin, A. Gerstlauer, R. Doemer, and D. Gajski. *System-on-chip environment (SCE Version 2.2.0 beta): Tutorial*. UC Irvine, Irvine, CA, July 2003. Tech. Rep. CECS-TR-03-41.
- [2] C. Chantrapornchai, E. H.-M. Sha, and X. S. Hu. Efficient design exploration based on module utility selection. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 19(1):19–29, 2000.
- [3] <http://www.criticalblue.com>.
- [4] J. Falk, C. Haubelt, and J. Teich. Efficient Representation and Simulation of Model-Based Designs in SystemC. In *Proc. of FDL'06*, Darmstadt, Germany, Sept. 2006.
- [5] <http://www.forteds.com>.
- [6] T. Grötter, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.
- [7] S. Ha, C. Lee, Y. Yi, S. Kwon, and Y.-P. Joo. Hardware-software codesign of multimedia embedded systems: the PeaCE approach. In *RTCSA*, pages 207–214, 2006.
- [8] C. Haubelt, J. Falk, J. Keinert, T. Schlichter, M. Streubühr, A. Deyhle, A. Hadert, and J. Teich. A SystemC-based Design Methodology for Digital Signal Processing Systems. *EURASIP Journal on Embedded Systems, Special Issue on Embedded Digital Signal Processing Systems*, 2007:Article ID 47580, 22 pages, 2007. doi:10.1155/2007/47580.
- [9] ITU. *Digital Compression and Coding of Continuous-Tone Still Images - Requirements and Guidelines*. CCITT, T.81 edition, 09 1992.
- [10] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, M. Hännikäinen, T. D. Hämäläinen, J. Riihimäki, and K. Kuusilinna. UML-Based Multiprocessor SoC Design Framework. *ACM Transactions on Embedded Computing Systems*, 5(2):281–320, May 2006.
- [11] V. Kianzad and S. S. Bhattacharyya. CHARMED: A Multi-Objective Co-Synthesis Framework for Multi-Mode Embedded Systems. In *Proc. of ASAP'04*, pages 28–40, Galveston, U.S.A., Sept. 2004.
- [12] M. Kim, S. Banerjee, N. Dutt, and N. Venkatasubramanian. Design space exploration of real-time multi-media MPSoCs with heterogeneous scheduling policies. In *CODES+ISSS*, pages 16–21, 2006.
- [13] E. A. Lee and S. Neuendorffer. Actor-oriented Models for Codesign: Balancing Re-Use and Performance. In *Formal Methods and Models for System Design*, pages 33–56. Kluwer Academic Publishers, Norwell, MA, USA, 2004.
- [14] E. A. Lee, S. Neuendorffer, and M. J. Wirthlin. Actor-Oriented Design of Embedded Hardware and Software Systems. *Journal of Circuits, Systems, and Computers*, 12(3):231–260, 2003.
- [15] R. Leupers. *Code Optimization Techniques for Embedded Processors – Methods, Algorithms, and Tools*. Kluwer Academic Publishers, Nov. 2000.
- [16] M. Lukaszewicz, M. Glaß, C. Haubelt, and J. Teich. Efficient symbolic multi-objective design space exploration. In *Proceedings of the 13th Asia and South Pacific Design Automation Conference (ASP-DAC 2008)*, pages 691–696, Seoul, Korea, Jan. 2008.
- [17] S. Mamagkakis, D. Atienza, C. Poucet, F. Catthoor, D. Soudris, and J. M. Mendias. Automated exploration of pareto-optimal configurations in parameterized dynamic memory allocation for embedded systems. In *Proc. of DATE*, pages 874–875, 2006.
- [18] <http://www.mentor.com>.
- [19] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis. Rapid Design Space Exploration of Heterogeneous Embedded Systems Using Symbolic Search and Multi-Granular Simulation. In *Proceedings of Languages, compilers and tools for embedded systems*, pages 18–27, Berlin, Germany, June 2002.
- [20] P. K. Murthy and S. S. Bhattacharyya. *Memory Management for Synthesis of DSP Software*. CRC Press, 2006.
- [21] <http://www.cyberworkbench.com>.
- [22] H. D. Patel, S. K. Shukla, E. Mednick, and R. S. Nikhil. A Rule-Based Model of Computation for SystemC: Integrating SystemC and Bluespec for Co-Design. In *Proceedings of International Conference on Formal Methods and Models for Co-Design*, pages 39–48, 2006.
- [23] A. D. Pimentel, C. Erbas, and S. Polstra. A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels. *IEEE Transactions on Computers*, 55(2):99–112, 2006.
- [24] <http://netpbm.sourceforge.net/doc/ppm.html>.
- [25] K. Strehl, L. Thiele, M. Gries, D. Ziegenbein, R. Ernst, and J. Teich. FunState - An Internal Design Representation for Codesign. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(4):524–544, Aug. 2001.
- [26] M. Streubühr, J. Falk, C. Haubelt, J. Teich, R. Dorsch, and T. Schlipf. Task-Accurate Performance Modeling in SystemC for Real-Time Multi-Processor Architectures. In *Proc. of DATE*, pages 480–481, Mar. 2006.
- [27] <http://www.synplicity.com>.
- [28] M. Thompson, H. Nikolov, T. Stefanov, A. Pimentel, C. Erbas, S. Polstra, and E. Deprettere. A Framework for Rapid System-level Exploration, Synthesis, and Programming of Multimedia MP-SoCs. In *Proceedings of CODES-ISSS'07*, pages 9–14, 2007.
- [29] XILINX. *Embedded SystemTools Reference Manual - Embedded Development Kit EDK 8.1ia*, October 2005.