A Framework for Rapid System-level Exploration, Synthesis, and Programming of Multimedia MP-SoCs

Mark Thompson†, Hristo Nikolov‡, Todor Stefanov‡,
Andy D. Pimentel†, Cagkan Erbas†, Simon Polstra†, Ed F. Deprettere‡

†Department of Computer Science
University of Amsterdam, The Netherlands
{thompson,andy,cagkan,spolstra}@science.uva.nl

‡Leiden Embedded Research Center
Leiden University, The Netherlands
{nikolov,stefanov,edd}@liacs.nl

ABSTRACT

In this paper, we present the Daedalus framework, which allows for traversing the path from sequential application specification to a working MP-SoC prototype in FPGA technology with the (parallelized) application mapped onto it in only a matter of hours. During this traversal, which offers a high degree of automation, guidance is provided by Daedalus' integrated system-level design space exploration environment. We show that Daedalus offers remarkable potentials for quickly experimenting with different MP-SoC architectures and exploring system-level design options during the very early stages of design. Using a case study with a Motion-JPEG encoder application, we illustrate Daedalus' design steps and demonstrate its efficiency.

Categories and Subject Descriptors

J.6 [**Computer-aided Engineering**]: Computer-aided design

General Terms

Performance, design

Keywords

Design space exploration, system-level design and synthesis, rapid prototyping

1. INTRODUCTION

The complexity of modern embedded systems, which are increasingly based on heterogeneous MultiProcessor-SoC (MP-SoC) architectures, has led to the emergence of system-level design. To cope with this design complexity, system-level design aims at raising the abstraction level of the design process. Key enablers to this end are, for example, the use of architectural platforms to facilitate re-use of IP components and the notion of high-level system modeling and simulation [7]. The latter allows for capturing the behavior of platform components and their interactions at a high level of abstraction. As such, these high-level models minimize the modeling effort and are optimized for execution speed, and can therefore be applied during the very early design stages to perform, for example, architectural Design Space Exploration (DSE). Such early DSE is of paramount importance as early design choices heavily influence the success or failure of the final product.

System-level design for MP-SoC based embedded systems typically involves a number of challenging tasks. For example, applications need to be decomposed into parallel specifications so that they can be mapped onto an MP-SoC architecture [10]. Subsequently, applications need to be partitioned into HW and SW parts since MP-SoC architectures often are heterogeneous in nature. To this end, MP-SoC platform architectures need to be modeled and simulated to study system behavior and to evaluate a variety of different design options. Once a good candidate architecture has been found, it needs to be synthesized, which involves the synthesis of its architectural components as well as the mapping of applications onto the architecture. To accomplish all of these tasks, a range of different tools and tool-flows is often needed, potentially leaving designers with all kinds of interoperability problems. Moreover, there typically remains a large gap between the deployed system-level models and actual implementations of the system under study, known as the *implementation gap* [11]. Currently, there exist no mature methodologies, techniques, and tools to effectively and efficiently convert system-level system specifications to RTL specifications.

In this paper, we present the Daedalus framework which addresses these system-level design challenges. Daedalus' main objective is to bridge the aforementioned implementation gap for the design of multimedia MP-SoCs. It does so by providing an integrated and highly-automated environment for system-level architectural exploration, system-level synthesis, programming and prototyping. Whereas our prior publications reported on several of Daedalus' components in isolation (e.g., [21, 15, 13]), this paper focuses on how the different components fit together as the pieces of a puzzle, resulting in a system-level design environment that addresses the entire design trajectory with an unparalleled degree of automation. We will illustrate the framework and its design flow using a case study with a Motion-JPEG encoder application.

The next section provides a birds-eye overview of Daedalus, after which the three subsequent sections present the three core tools that constitute Daedalus in more detail. More specifically, Section 3 explains how multimedia applications are automatically decomposed in parallel specifications. Section 4 describes how – given the parallel application(s) – promising candidate architectures can be found using our system-level modeling, simulation and exploration methodology and toolset. In Section 5, we explain how selected candidate architectures can be automatically and rapidly synthesized, programmed and prototyped. Section 6 presents a Motion-JPEG case study to illustrate Daedalus' design flow. In Section 7, we present related work, after which Section 8 concludes the paper.

CODES+ISSS'07, September 30–October 3, 2007, Salzburg, Austria.

2. THE DAEDALUS FRAMEWORK

In Figure 1, the design flow of the Daedalus framework is depicted. As mentioned before, Daedalus provides a single environment for rapid system-level architectural exploration, high-level synthesis, programming and prototyping of multimedia MP-SoC architectures. Here, a key assumption is that the MP-SoCs are constructed from a library of pre-determined and pre-verified IP components. These components include a variety of programmable and dedicated processors, memories and interconnects, thereby allowing the implementation of a wide range of MP-SoC platforms. The remainder of this section provides a high-level overview of Daedalus, after which the subsequent sections zoom in on its core components and how they interact with the rest of the design flow.

Starting from a sequential application specification in C or C++, the KPNgen tool [21] allows for automatically converting the sequential application into a parallel Kahn Process Network (KPN) [8] specification. Here, the sequential input specifications are restricted to so-called static affine nested loop programs, which is an important class of programs in, e.g., the scientific and multimedia application domains. By means of automated source-level transformations [17], KPNgen is also capable of producing different input-output equivalent KPNs, in which for example the degree of parallelism can be varied. Such transformations enable application-level design space exploration.

The generated or handcrafted KPNs (the latter in the case that, e.g., the input specification did not entirely meet the requirements of the KPNgen tool) can subsequently be used by our Sesame modeling and simulation environment [15] to perform system-level architectural DSE. To this end, Sesame uses (high-level) architecture model components from the IP component library. Sesame allows for quickly evaluating the performance of different application to architecture mappings, HW/SW partitionings, and target platform architectures. Such DSE should result in a number of promising candidate system designs, of which their specifications (system-level platform description, application-architecture mapping description, and application description) act as input to the ESPAM tool [13]. This tool uses these system-level input specifications, together with RTL versions of the components from the IP library, to automatically generate synthesizable VHDL that implements the candidate MP-SoC platform architecture. In addition, it also generates the C/C++ code for those application processes that are mapped onto programmable cores. Using commercial synthesis tools and compilers, this implementation can be readily mapped onto an FPGA for prototyping. Such prototyping also allows for calibrating and validating Sesame's system-level models, and as a consequence, improving the trustworthiness of these models.

Ultimately, we aim at traversing Daedalus' design flow – going from a sequential application to a working MP-SoC prototype in FPGA technology with the application mapped onto it – in a matter of hours. Evidently, this would offer great potentials for quickly experimenting with different MP-SoC architectures and exploring design options during the early stages of design. As our case study in Section 6 shows, we are well underway of achieving this goal.

3. PARALLELIZING APPLICATIONS

Today, traditional imperative languages like C or C++ are still dominant with respect to implementing applications for SoC-based architectures. It is, however, difficult to map these imperative implementations, with typically a sequential model of computation, onto MP-SoC architectures that allow for exploiting task-level parallelism in applications. In contrast, models of computation that inherently express task-level parallelism in applications and make

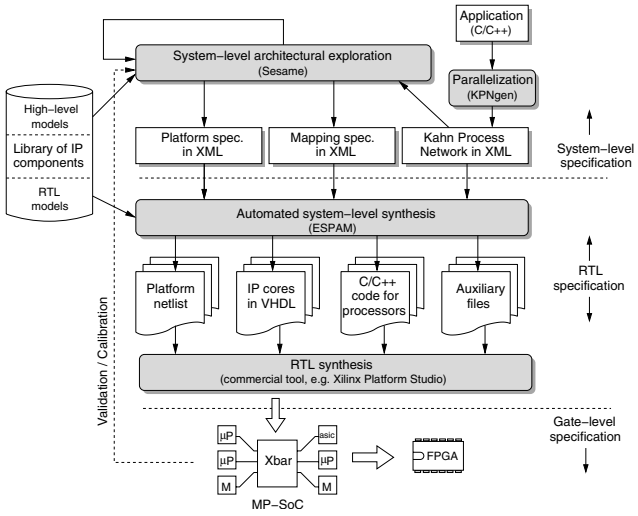


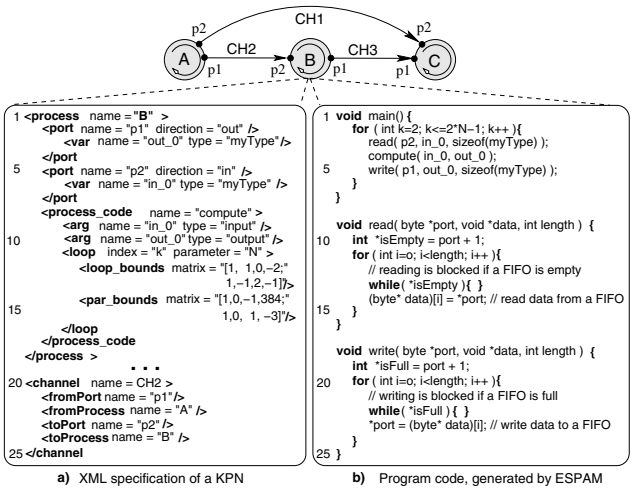
Figure 1: The Daedalus design flow.

communications explicit, such as CSP [5] and Process Networks [8], allow for easier mapping onto MP-SoC architectures. However, specifying applications using these models of computation usually requires more implementation effort in comparison to sequential imperative solutions.

In Daedalus, we start from a sequential imperative application specification (C/C++) which is then *automatically* converted into a Kahn Process Network (KPN) [8] using the KPNgen tool [21]. This conversion is fast and correct by construction. In the KPN model of computation, parallel processes communicate with each other via unbounded FIFO channels. Reading from channels is done in a blocking manner, while writing to channels is non-blocking. We use KPNs for application specifications because this model of computation nicely fits the targeted media-processing application domain and is deterministic. The latter implies that the same application input always results in the same application output, irrespective of the scheduling of the KPN processes. This provides complete scheduling freedom when, as will be discussed later on, mapping KPN processes onto MP-SoC architecture models for quantitative performance analysis and design space exploration.

As mentioned before, KPNgen's input applications need to be specified as so-called static affine nested loop programs to allow for automatic parallelization of applications. As a first step, KPNgen can apply a variety of source-level transformations to these specifications in order to, for example, increase or decrease the amount of parallelism in the final KPN [17]. Subsequently, the C/C++ code is transformed into single assignment code (SAC), which resembles the dependence graph (DG) of the original nested loop program. Hereafter, the SAC is converted to a Polyhedral Reduced Dependency Graph (PRDG) data structure, being a compact mathematical representation of a DG in terms of polyhedra. Finally, a PRDG is converted into a KPN by associating a KPN process with each node in the PRDG. The parallel KPN processes communicate with each other according to the data dependencies given in the DG.

In Figure 2, a Kahn Process Network example is given in which three processes (A, B and C) are connected using three channels (CH1-3). Figure 2(a) shows the XML description of Kahn process B as generated by KPNgen. The XML describes both the topology of the KPN (i.e., how the processes are connected together, see e.g. lines 20-25) as well as the communications and computations performed by processes. In our example, process B executes a function called *compute* (line 8). The function has one input argument (line 9) and one output argument (line 10). The relation between the



a) XML specification of a KPN **b)** Program code, generated by ESPAM

Figure 2: A Kahn Process Network example.

function arguments and the communication ports of the process is given in lines 3 and 6. The function has to be executed $2*N-2$ times as specified by the polytope in lines 12-13. The value of N is between 3 and 384 (lines 14-15).

From the XML specification, Daedalus allows for automatically generating the C/C++ code implementing the behavior of each KPN process. This is done by the ESPAM tool, which will be discussed later on. Figure 2(b) shows, for example, the generated C code for process B (some variable declarations have been omitted). The code contains the main behavior of a process, together with the read/write communication primitives. In accordance with the XML specification in Figure 2(a), the function *compute* – which is derived from the original sequential application specification – is part of a loop that iterates $2*N-2$ times. For synthesis purposes, Daedalus also allows for generating the code for the read and write communication primitives, as shown in Figure 2(b). Currently, these primitives are implemented using polling and memory-mapped I/O. Note that the implementation of the write primitive is blocking since at implementation level FIFO channels are bounded in size.

4. DESIGN SPACE EXPLORATION

Given a (set of) KPN application specification(s) – as for example generated by KPNgen or devised by hand – and the components in Daedalus' IP library, the Sesame system-level simulation framework [15] addresses the problem of finding a suitable and efficient target MP-SoC platform architecture. Figure 3 illustrates Sesame's layered infrastructure for the case in which a Motion-JPEG application is studied with a crossbar-based distributed-memory MP-SoC as target architecture. Sesame deploys separate application and architecture models, where an application model describes the functional behavior of an application and an architecture model defines architecture resources and captures their performance constraints. After explicitly mapping an application model onto an architecture model, they are co-simulated via trace-driven simulation. This allows for evaluation of the system performance of a particular application, mapping, and underlying architecture. Essential in this methodology is that an application model is independent from architectural specifics and assumptions on hardware/software partitioning. As a result, a single application model can be used to exercise different hardware/software partitionings and can be mapped onto a range of architecture models, possibly representing different architecture designs or modeling the same architecture design at various levels of abstraction.

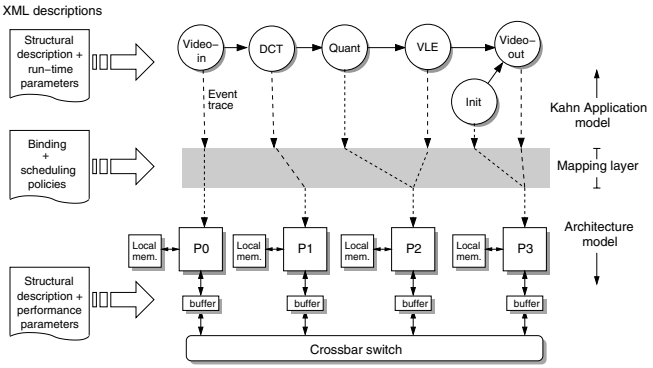


Figure 3: Sesame's layered infrastructure.

For application modeling, the computational and communication behavior of the KPN application specifications are captured using *application event traces*. The computation and communication events in these traces typically are coarse grained, such as *Execute(DCT)* or *Read(channel_id, pixel-block)*. To generate the application events, the C/C++ code of each Kahn process is instrumented with annotations that describe the application's computational actions. In addition, Sesame provides read and write communication primitives that generate communication events as a side-effect. So, by executing the KPN model, each process generates its own trace of application events, representing the workload that is imposed on the underlying MP-SoC architecture model.

An architecture model simulates the performance consequences of the computation and communication events generated by an application model. To this end, each component in the architecture model is parameterized with performance parameters specifying the latencies of computation events like *Execute(DCT)*, communication transactions, and memory accesses. This approach allows to quickly assess, e.g., different HW/SW partitionings by simply experimenting with the latency parameters of processing components in the architecture model: a low computational latency refers to a HW implementation while a high latency mimics a SW solution.

To bind application tasks to resources in the architecture model, Sesame provides an intermediate *mapping layer*. It controls the mapping of Kahn processes (i.e. their event traces) onto architecture model components by dispatching application events to the correct architecture model component. The mapping also includes the mapping of Kahn channels onto communication resources in the architecture model. The mapping layer has two additional purposes. First, the event dispatch mechanism in the mapping layer provides a variety of static and dynamic policies to schedule application tasks (i.e., their event traces) that are mapped onto shared architecture model components. Second, the mapping layer is also capable of dynamically transforming application events into (lower-level) architecture events in order to facilitate flexible refinement of architecture models [15].

The output of system simulations in Sesame provides the designer with performance estimates of the system(s) under study together with statistical information such as utilization of architectural components (idle/busy times), the contention in a system (e.g., network contention), profiling information (time spent in different executions), critical path analysis, and average bandwidth between architecture components. Such results allow for early evaluation of different design choices, identifying trends in the systems' behavior, and can help in revealing performance bottlenecks early in the design cycle. Here, the exploration process is also facilitated by the fact that system configurations (bindings, scheduling and arbitration policies, performance parameters, and so on) are specified

using XML descriptions. Hence, different system configurations can be rapidly simulated without remodeling and/or recompilation.

As a result of the design space exploration with Sesame, a small set of promising MP-SoC platform instances can be selected for automatic synthesis (see next section). Each selected platform instance is specified using two XML files. One describing the architectural platform at the system level, i.e. which IP components are used in the platform and how they are interconnected. And the other describing how application tasks are mapped onto the platform components.

5. SYSTEM-LEVEL SYNTHESIS

The system-level specifications that result from DSE – describing (the structure of) the application and platform architecture as well as the mapping of the former onto the latter – are given as input to the ESPAM tool for system-level synthesis [13]. To guarantee correctness-by-construction, ESPAM first runs a consistency check on the provided platform instance. This includes finding impossible and/or meaningless connections between system-level platform components as well as parameter values that are out of range. Subsequently, ESPAM refines the abstract platform model to a parameterized RTL model which is ready for an implementation on a target physical platform. The refined system components are instantiated by setting their parameters based on the target physical platform features. Finally, ESPAM generates program (C/C++) code for each programmable processor in the multiprocessor platform in accordance with the application and mapping specifications. To this end, it uses the XML specifications generated by KPNgen. In addition, ESPAM also provides the support for scheduling the code in the case multiple application processes are mapped onto a single processor in the platform. Currently, this code scheduling is performed statically.

The output of ESPAM, namely an RTL specification of the MP-SoC platform, is a model that can adequately abstract and exploit the key features of a target physical platform at the register transfer level. It consists of four parts (as shown in Figure 1): 1) a *platform topology* description defining in greater detail the structure of the multiprocessor platform; 2) *hardware descriptions of IP cores* containing predefined and custom IP cores used in 1). These IP cores, which are selected from Daedalus' IP component library, include programmable as well as dedicated processors, various memory components (FIFO buffers, random access memory, etc.), and different interconnects (point-to-point links, shared bus with various arbitration mechanisms, and a crossbar switch). For programmable processors, ESPAM currently uses PowerPCs and Microblazes since it targets Xilinx Virtex-II-Pro FPGA technology for prototyping the synthesized MP-SoCs. ESPAM also automatically generates custom IP cores needed as a glue/interface logic between components in the platform; 3) the *program code for processors* — as mentioned before, to execute the software parts of the application on the synthesized multiprocessor platform, and 4) *Auxiliary information* containing files which give tight control on the overall specifications, such as defining precise timing requirements and prioritizing signal constraints.

With the above descriptions, a commercial synthesizer can convert an RTL specification to a gate-level specification, thereby generating the target platform gate-level netlist (see the bottom part of Figure 1). At this moment, ESPAM facilitates automated MP-SoC synthesis and programming using Xilinx VirtexII-Pro FPGAs and therefore uses the Xilinx Platform Studio (XPS) tool as a back-end to generate the final bit-stream file that configures the FPGA. However, our framework is general and flexible enough to be targeted to other physical platform technologies as well.

6. A CASE STUDY

This section presents a case study in which we applied Daedalus to explore different implementation options for a Motion-JPEG (M-JPEG) encoder application mapped onto a heterogeneous MP-SoC architecture. The case study illustrates Daedalus' design steps and demonstrates its potentials to quickly experiment with different MP-SoC architecture designs during the very early stages of design.

The KPN specification of the M-JPEG application was derived from sequential C code using the KPNgen tool as described in Section 3. A small manual modification (taking no longer than 30 minutes) to the original M-JPEG code was necessary to meet the KPNgen input requirements. The resulting Kahn application specification consists of 6 processes, as shown in the top part of Figure 3. Generating the KPN specification is a one-time effort since the same specification is used for all subsequent implementation and exploration steps.

To study target MP-SoC architecture instances for the M-JPEG application, we selected a crossbar-based MP-SoC platform with up to 4 processors (MicroBlaze or PowerPC) and distributed memory. At the bottom part of Figure 3, a 4-processor instance of this platform is depicted. We modeled this platform architecture with the Sesame framework. The processor, memory and interconnect components in our architecture model were taken directly from Daedalus' high-level model component library. Only the performance parameters specific to the selected platform architecture needed to be specified, such as the latencies for computational actions, the latencies for setting up and communicating over the crossbar, and so on. We determined the values of these performance parameters by a combination of measurements on an ISS simulator (for the computational latencies on the MicroBlaze and PowerPC processors) and on the actual hardware itself. Note that this needs to be done only once for each application, since the values can be reused throughout the exploration process. More information about the calibration of our architectural performance models can be found in [16]. Moreover, the mapping layer in our system-level model is configured such that it models the static scheduling scheme as facilitated by the ESPAM framework (see Section 5). To this end, for shared architecture components, the mapping layer dynamically groups trace events that originate from the same Kahn process and interleaves these event groups in the same manner as would be the result of ESPAM's static scheduling.

In our design space exploration experiments, we selected three degrees of freedom, namely the number of processors in the platform (1 to 4), the type of processors (MicroBlaze or PowerPC) and the mapping of application processes onto the processors. For the sake of simplicity, the network configuration (crossbar switch) as well as the buffer/memory sizes remained unaltered (although these could also have been included in the exploration). For this particular case study, we were able to exhaustively explore the resulting design space – consisting of 10,148 design points – using system-level simulation, where the M-JPEG application was executed on 8 consecutive 128x128 resolution frames for each design point. As can be seen in Table 2, this design space sweep took 2.5 hours, demonstrating Sesame's efficiency. Figure 4 shows for three platform instances the relation between mappings and system performance, where we sorted the different mapping instances on performance. It clearly illustrates the importance of finding a good mapping since non-optimal mappings on larger MP-SoC platforms may perform worse than a good mapping on smaller MP-SoCs.

To validate our DSE experiments, we selected a number of design points with random application-to-architecture mappings and synthesized and prototyped them using ESPAM. The results of these validation experiments are shown in Figure 5. Note that a

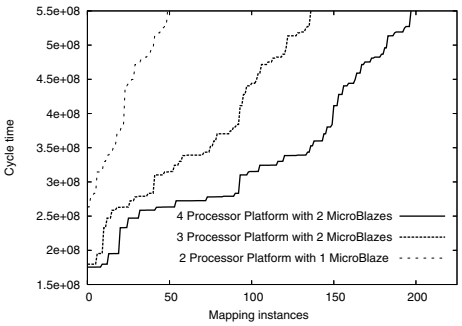


Figure 4: Performance of mappings on different platforms.

synthesized platform can only contain up to two PowerPCs due to the Xilinx Virtex-II-Pro FPGA chip (xc2vp20) that is used for prototyping. For the chosen design points, our abstract system-level simulations adequately show the correct performance trends, with an average error of 12% and worst-case error of 19%. The inaccuracies in terms of absolute cycle numbers are mainly caused by the modeling of the PowerPC processors. This because these processors are connected to the crossbar using a bus that is also used for access to the processor's local data and instruction memory. Since we do not explicitly model (contention on) this bus, our abstract PowerPC performance model is too optimistic.

Naturally, we also used our exploration results to find the best mapping for each platform instance. The graph on the left-hand side in Figure 6 shows the best design points found by our DSE for purely MicroBlaze based platforms, together with the real measurements from the prototypes of these design points. Clearly, our abstract performance models quite accurately reflect the performance behavior of the actual systems. When introducing one PowerPC in the platform, as depicted on the right-hand side in Figure 6, the absolute errors become larger (due to the inaccuracy of our current high-level PowerPC model, as explained above) but the correct performance trend is still shown. For MP-SoCs with more than two processors, this inaccuracy seems to be amortized again.

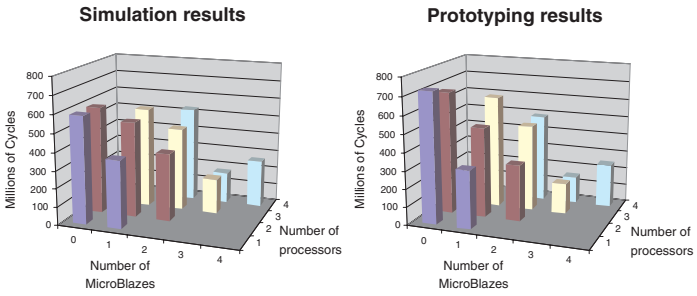


Figure 5: Validation experiment: simulation results (left) and actual measurements (right).

To give an impression of overall resource utilization of the multiprocessor systems that are generated by Daedalus' ESPAM tool, Table 1 shows the utilization of FPGA resources for an MP-SoC containing 4 MicroBlazes. Here, we recognize FPGA resource utilization for the entire MP-SoC, as well as specific utilization results for the Communication Controllers (CCs) that glue together the processors with the interconnect and the crossbar interconnect itself. As can be seen, the MP-SoC only takes about 40% of the FPGA slices, of which about 5% is used for the communication components. We note that the high BRAM usage reported in the last column is due to the complexity of the M-JPEG application,

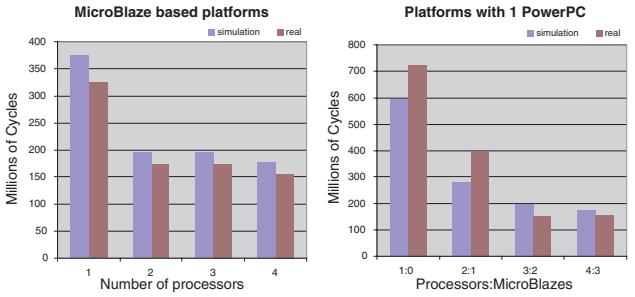


Figure 6: Best mapping results for MicroBlaze based platforms (left) and platforms with one PowerPC (right).

which causes each processor's local program and data memory to be quite large. We emphasize that the high BRAM usage is not caused by the implementation of the communication memories (the FIFO memories to which the Kahn channels are mapped), since they only use a maximum of 9 BRAMs.

Table 1: Resource Utilization for MicroBlaze based system.

	#Slices	#4-Input LUT	#Flip-Flops	#BRAMs
4 proc. system	3653 (39%)	4748 (25%)	2357 (12%)	85 (60%)
4 CCs	288 (2%)	468 (2%)	116 (1%)	—
4-Port crossbar	397 (3%)	587 (3%)	56 (1%)	—

Table 2 shows a breakdown of the execution time for each step in Daedalus' design flow in the case that one selected MP-SoC platform instance (a 4-processor MicroBlaze based architecture) is synthesized and implemented. The processing times were measured on a 1.8 GHz Pentium 4. Note that some of the steps only need to be performed once (such as the KPN derivation), after which, for example, the synthesis and physical implementation stages can be iterated several times to prototype different platform instances. The results from Table 2 demonstrate that the entire design trajectory, from sequential application specification to MP-SoC prototype executing the parallelized application on top of it, takes only a matter of hours. Evidently, this allows designers to quickly *prototype and assess* different platform instances and implementation choices during the very early design stages. Also noticeable is the fact that the system-level DSE component (Sesame) still requires a relatively high amount of manual effort. The manual effort listed in Table 2 is mainly due to the construction of the platform architecture model and the adaptation/construction of scripts that perform the automatic design space exploration. Not taken into account is the calibration of model components, which is a one-time effort for every application that is studied.

Table 2: Processing Times (hh:mm:ss).

Tool	KPN Derivation	Syst.level DSE	Syst.level to RTL conv.	Physical Impl.	Manual effort
KPNgen	00:00:22	–	–	–	00:30:00
Sesame	–	02:30:00	–	–	01:30:00
ESPAM	–	–	00:00:24	–	00:10:00
XPS tool	–	–	–	02:09:00	–

7. RELATED WORK

Systematic and automated application-to-architecture mapping has been widely studied in the research community. The closest to our work is the Koski MP-SoC design flow [18]. Koski also provides a single infrastructure for modeling of applications, automatic architectural design space exploration, and automatic system-level synthesis, programming, and prototyping of selected MP-SoCs.

But unlike Daedalus, Koski does not allow for parallelization of applications, nor design space exploration at application level. Koski requires applications to be specified by hand in UML. Other examples of related work can be found in [20, 9, 2, 4]. However, these efforts are limited to processor-coprocesor architectures [20], only provide a limited degree of automation [9, 2], or do not provide an automated step towards the register transfer level [4].

Companies such as Xilinx and Altera provide design tool chains attempting to generate efficient implementations starting from descriptions higher than (but still related to) the register transfer level of abstraction. The required input specifications are still so detailed that designing a single processor system is still error-prone and time consuming, let alone designing alternative multiprocessor systems. In contrast, Daedalus raises the design to an even higher level of abstraction allowing the exploration, design and programming of multiprocessor systems in a short amount of time.

Work focusing on the mapping of applications onto MP-SoCs, in the form of programming models, can be found in, e.g., [14, 6].

With respect to Daedalus' DSE component (i.e., Sesame), there are a number of related architectural exploration environments (e.g., [3, 1, 12, 19]) that facilitate flexible system-level performance evaluation by providing support for mapping a behavioral application specification to an architecture specification. In comparison to most related efforts, Sesame tries to push the separation of modeling application behavior and modeling architectural constraints at the system level to even greater extents. Doing so, it aims at optimizing the potentials for model re-use during the exploration cycle.

8. CONCLUSIONS

In this paper, we presented the Daedalus framework that tries to bridge the so-called implementation gap between system-level platform specifications and the actual physical implementations of these platforms. To this end, Daedalus focuses on the design of multimedia MP-SoC platforms. As such, it provides an integrated and highly-automated environment for system-level architectural exploration, system-level synthesis, programming and prototyping. Such a framework offers remarkable potentials for quickly experimenting with different MP-SoC architectures and exploring system-level design options during the very early stages of design. We illustrated Daedalus' design steps and demonstrated its efficiency using a case study with a Motion-JPEG encoder application. Main research directions for the future are increasing the level of automation even further, relaxing the constraints put on the input application specifications (e.g., handling more dynamic applications), developing more advanced design space steering and pruning techniques by means of e.g. genetic algorithms, and the inclusion of high-level power modeling during DSE.

9. REFERENCES

[1] A. Cassidy, J. Paul, and D. Thomas. Layered, multi-threaded, high-level performance design. In *Proc. of the Design, Automation and Test in Europe*, March 2003.

[2] D. Lyonnard et al. Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System-on-Chip. In *Proc. of the Design Automation Conference (DAC'2001)*, June 18-22 2001.

[3] F. Balarin et al. Metropolis: An integrated electronic system design environment. *IEEE Computer*, 36(4), April 2003.

[4] A. Gerstlauer and D. Gajski. System-level abstraction semantics. In *Proc. 15th Int. Symposium on System Synthesis (ISSS'02)*, pages 231–236, Oct. 2-4 2002.

[5] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8), August 1978.

[6] A. A. Jerraya, A. Bouchhima, and F. Pétrot. Programming models and hw-sw interfaces abstraction for multi-processor SoC. In *Proc. of the Design Automation Conference (DAC)*, pages 280–285, 2006.

[7] K. Keutzer et al. System level design: Orthogonalization of concerns and platform-based design. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 19(12), Dec. 2000.

[8] G. Kahn. The semantics of a simple language for parallel programming. In *Proc. of the IFIP Congress 74*, 1974.

[9] M. J. Rutten et al. A Heterogeneous Multiprocessor Architecture for Flexible Media Processing. *IEEE Design & Test of Computers*, 19(4), 2002.

[10] G. Martin. Overview of the MPSoC Design Challenge. In *Proc. Design Automation Conference (DAC)*, San Francisco, USA, July 24-28 2006.

[11] A. Mihal and K. Keutzer. Mapping concurrent applications onto architectural platforms. In A. Jantsch and H. Tenhunen, editors, *Networks on Chips*, pages 39–59. Kluwer Academic Publishers, 2003.

[12] S. Mohanty and V. K. Prasanna. Rapid system-level performance evaluation and optimization for application mapping onto SoC architectures. In *Proc. of the IEEE International ASIC/SOC Conference*, 2002.

[13] H. Nikolov, T. Stefanov, and E. Deprettere. Multi-processor system design with ESPAM. In *Proc. of the Int. Conf. on HW/SW Codesign and System Synthesis (CODES-ISSS'06)*, pages 211–216, Oct. 2006.

[14] P. G. Paulin et al. Parallel Programming Models for a Multiprocessor SoC Platform Applied to Networking and Multimedia. *IEEE Trans. on VLSI Systems*, 14(7), 2006.

[15] 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.

[16] A. D. Pimentel, M. Thompson, S. Polstra, and C. Erbas. On the calibration of abstract performance models for system-level design space exploration. In *Proc. of the Int. Conf. on Embedded Computer Systems: Architectures, MOdeling, and Simulation (IC-SAMOS)*, pages 71–77, 2006.

[17] T. Stefanov, B. Kienhuis, and E. F. Deprettere. Algorithmic transformation techniques for efficient exploration of alternative application instances. In *Proc. of the Int. Symposium on Hardware/Software Codesign (CODES)*, pages 7–12, May 2002.

[18] T. Kangas et al. UML-based multi-processor SoC design framework. *ACM Trans. on Embedded Computing Systems*, 5(2):281–320, May 2006.

[19] T. Kogel et al. Virtual architecture mapping: A SystemC based methodology for architectural exploration of system-on-chip designs. In *Proc. of the Int. workshop on Systems, Architectures, Modeling and Simulation (SAMOS)*, pages 138–148, 2003.

[20] T. Stefanov et al. System design using Kahn process networks: The Compaan/Laura approach. In *Proc. of the Int. Conference on Design, Automation and Test in Europe (DATE)*, pages 340–345, Feb. 2004.

[21] S. Verdoolaege, H. Nikolov, and T. Stefanov. PN: a tool for improved derivation of process networks. *EURASIP Journal on Embedded Systems*, 2007.