

System-on-Chip Environment: A SpecC-based Framework for Heterogeneous MPSoC Design

Rainer Dömer, Andreas Gerstlauer, Junyu Peng, Dongwan Shin, Lukai Cai, Haobo Yu, Samar Abdi,
and Daniel D. Gajski

Center for Embedded Computer Systems, University of California, Irvine

Abstract—

The constantly growing complexity of embedded systems is a challenge that drives the development of novel design automation techniques. C-based system-level design addresses the complexity challenge by raising the level of abstraction and integrating the design processes for the heterogeneous system components. In this article, we present a comprehensive design framework, the System-on-Chip Environment (SCE) which is based on the influential SpecC language and methodology. SCE implements a top-down system design flow based on automatic model refinement with support for heterogeneous target platforms consisting of custom hardware components, embedded software processors, dedicated IP blocks, and complex communication bus architectures. Starting from an abstract specification of the desired system, models at various levels of abstraction are automatically generated through successive step-wise refinement, resulting in a pin- and cycle-accurate system implementation. The seamless integration of automatic model generation, estimation and verification tools enables rapid design space exploration and efficient MPSoC implementation. Using a large set of industrial-strength examples with a wide range of target architectures, our experimental results demonstrate the effectiveness of our framework and show significant productivity gains in design time.

Index Terms—Embedded Systems, System-Level Design, SpecC, MPSoC.

I. INTRODUCTION

The rising complexity of embedded systems challenges the established design techniques and processes. Novel, non-traditional design approaches become necessary in order to keep up with the increasing demands of higher productivity.

A well-known technique to address the system design challenge is system-level design which raises the level of abstraction, exploits the reuse of intellectual property (IP), and integrates the traditionally separate design processes of the heterogeneous system components. By combining the design flows of hardware units, software processors, third-party IPs and the interconnecting bus architectures, system-level design emphasizes the system perspective of the overall design task and enables design space exploration across domains. However, successful system design depends on efficient design automation techniques and, in particular, effective tool support.

In this article, we describe the System-on-Chip Environment (SCE), a system-level design framework based on the SpecC language and methodology [1], [2]. SCE realizes a top-down refinement-based system design flow with support of heterogeneous target platforms consisting of custom hardware com-

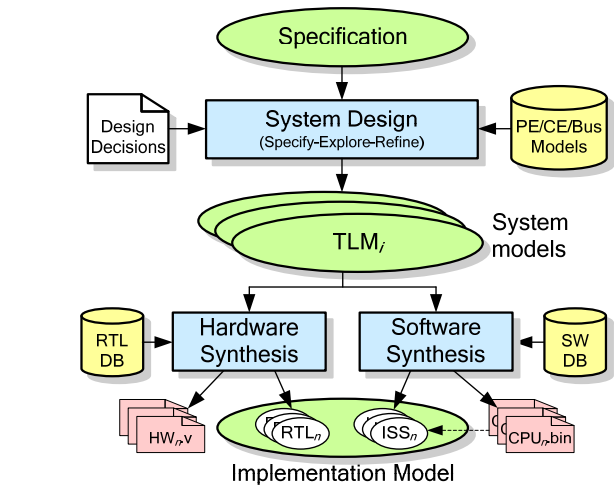


Fig. 1. System-on-Chip Environment (SCE) design flow.

ponents, embedded software processors, dedicated IP blocks, and complex communication bus architectures.

A. SCE Overview

Fig. 1 shows the design flow with SCE in an overview. Starting with an abstract specification model in the system design phase, the designer automatically generates Transaction Level Models (TLM) of the design, successively at lower levels of abstraction. Based on component models from the system database and design decisions made by the user, the generated models carry an increasing amount of implementation detail.

SCE follows a *specify-explore-refine* methodology. The design process starts from a model specifying the design functionality (*specify*). At each following step, the designer first explores the design space (*explore*) and makes the necessary design decisions. SCE then automatically generates a new model by integrating the decisions into the previous model (*refine*).

After the system design phase is complete, the hardware and software components in the system model are implemented by the hardware and software synthesis phases, respectively. As a combined result, a pin- and cycle-accurate implementation model is generated. Also, binary images for the software processors, as well as register-transfer level (RTL) descriptions in Verilog for the hardware blocks, are created for further

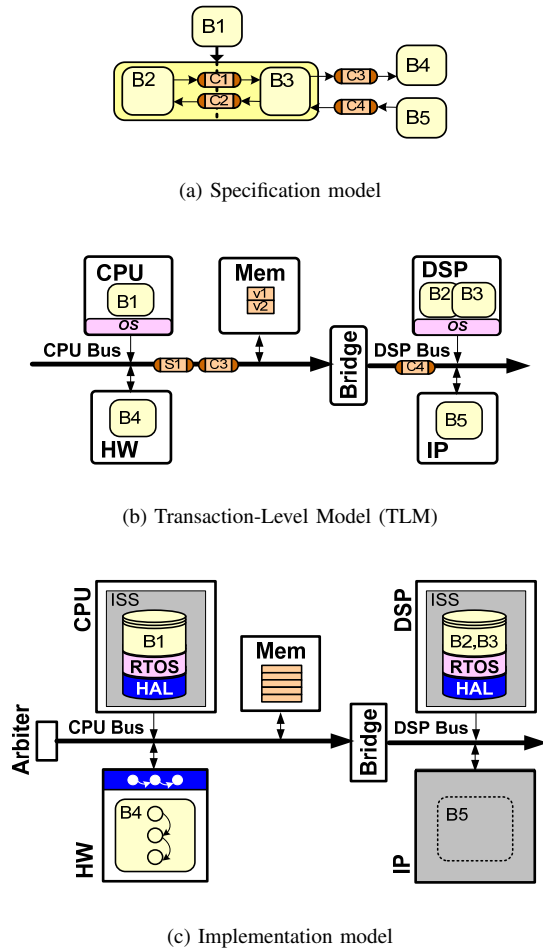


Fig. 2. Generic SCE design models.

synthesis and manufacturing of the intended Multi-Processor System-on-Chip (MPSoC).

The three main design models used in the SCE design flow are shown in more detail in Fig. 2. Fig. 2(a) depicts a simple generic specification model. The model consists of a hierarchy of five behaviors and four communication channels described in SpecC. Except for the system functionality this model is free of any implementation details [3]. During the system design phase, it will be mapped to a platform architecture (see Section III-A) and single-threaded processing elements (PEs) will be scheduled (Section III-B). Communication elements (CEs), such as bus bridges and transducers, and system busses will be added to the model as well (Section III-C and Section III-D).

As a result of each of these model refinement steps, a TLM is generated, as shown in Fig. 2(b). Depending on the number of implementation decisions taken, the TLM accurately reflects the number and type of PEs in the architecture, the mapping of behaviors to the PEs, and the mapping of channels to the system busses and CEs. Note that the communication in this model is still at the abstract transaction level.

After hardware and software synthesis (see Section III-E and Section III-F, respectively), a cycle-accurate implementation model is generated, as illustrated in Fig. 2(c). In this model, embedded software is represented in detailed layers,

including the real-time operating system (RTOS) and the hardware abstraction layer (HAL). Custom hardware blocks, on the other hand, are represented accurately by RTL finite state machine (FSM) models. Finally, system communication is also refined down to a pin- and cycle-accurate level.

B. Related Work

Traditionally, system design is dominated by simulation-centric approaches with horizontal integration of models at specific levels of abstraction. Approaches range from the co-simulation of different low-level languages [4]–[6] to the combination of heterogeneous models of computation in a common simulation environment [7]. In between, C-based system-level design languages (SLDLs), such as SystemC [8] and Handel-C [9], emerged as vehicles for transaction-level modeling (TLM) [10]. Most cases, however, are limited to simulation only and lack vertical integration with synthesis flows that provide a path to implementation.

The first attempts at providing system design environments were approaches for hardware/software co-design. Examples of such environments include COSYMA [11], COSMOS [12], and POLIS [13]. These approaches, however, are based on architecture templates consisting of a single microcontroller assisted by a custom hardware co-processor, and are thus limited to narrow target architectures.

More recently, design environments emerged that provide support for more complex multi-processor systems. The OCAPI system [14], [15] is based on an object-oriented modeling of designs using a C++ class library and focuses on reconfigurable hardware devices (FPGAs). Around the TLM concept, several SystemC-based approaches exist that deal with assembly, validation and to some extent automatic generation of communication [16]–[18]. Metropolis [19], [20] is a modeling and simulation environment based on the platform-based design paradigm. The key idea is to separate function, architecture and model of computation into separate models. Although Metropolis allows co-simulation of heterogeneous PEs as well as different models of computation, a refinement or verification flow between different abstraction levels has not emerged. None of the above frameworks provide a comprehensive, automated approach for the design of complete MPSoCs from abstract specification down to final implementation.

SCE was built on experiences obtained from its predecessor, SpecSyn [21]. While SpecSyn was based on the SpecCharts language, an extension of VHDL, SCE is based on SpecC, which extends ANSI-C for hardware and system modeling.

With respect to our previous publications, this article is the first comprehensive, cohesive, and complete description of the SCE tool framework [22], [23]. This work describes in detail the integration of tools that realizes an efficient top-down system design flow, all the way from an abstract system specification down to a pin- and cycle-accurate implementation. We also report the combined results of six design experiments that demonstrate the effectiveness of the SCE framework using real-world examples. Furthermore, this article describes for the first time the integration of verification tools into the design flow and framework.

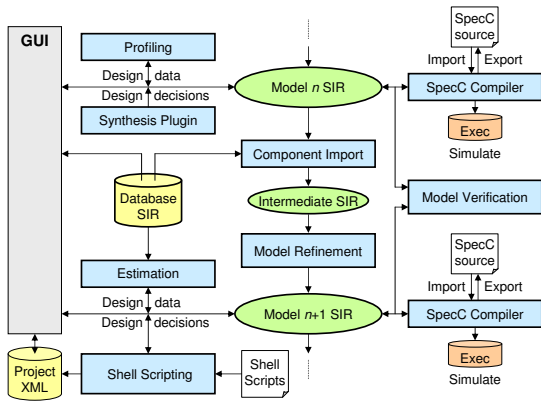


Fig. 3. SCE software architecture.

II. SCE ARCHITECTURE

SCE is based on the separation of design tasks into two distinct steps: decision making and model refinement. Model refinement takes design decisions and generates a new model of the design reflecting and implementing the decisions [2].

In SCE, model refinement is automated. Decisions, on the other hand, can be entered manually or through a tool box of automated synthesis algorithms. Together, SCE supports an interactive and automated system design process. Automatic model generation removes the need for error-prone and tedious model rewriting. Instead, designers can focus on design exploration and decision making.

Fig. 3 shows the generic software architecture for each task in the SCE design and refinement flow. In each step, design decisions can be entered by the user through a graphical user interface (GUI), via a command-line scripting and shell interface, or with the help of automated synthesis plugins implementing optimizing algorithms. Based on the design decisions, a refinement process generates a new design model from the input model automatically. SCE includes a set of databases with models of available system components. Database components are selected as part of decision making and component models are imported into the design during refinement. Furthermore, profiling and estimation tools perform analysis of design models in order to provide feedback about design quality metrics to the designer and/or synthesis plugins. Finally, verification tools support formal equivalence checks between input and output models in each step.

Overall, the SCE framework is formed by the combination of point tools. These tools exchange information through command line interfaces and design models. In general, all tools operate on a given design model. Design decisions, profiling data and meta-information about the design are stored as annotations attached to the corresponding objects in the design and database models. All models and databases in SCE are described and captured in the form of SpecC Internal Representation (SIR) files [24]. Using the SpecC compiler (scc), SCE models and databases can be imported from and exported into source files in standard SpecC language format at any time.

A. Graphical User Interface

The main interface between the designer and the tools is the *sce* GUI shown in Fig. 4 [22]. The GUI provides various displays and dialogs for browsing of design models and databases, interactive decision entry, and graphical analysis of profiling and estimation results. Furthermore, it includes menus and tool bars to trigger simulation, profiling, refinement, synthesis and verification actions. For each action, specific command-line tools are called and executed as needed where the GUI supplies the necessary parameters, captures the output and handles (normal or abnormal) results.

In each session, multiple candidate designs and models can be explored and generated. Information about design models and their relationships, including project-specific compiler and simulator parameters, are tracked by the GUI and can be stored in project files in a custom XML format, allowing for persistent storage, documentation and exchange of meta-information about the exploration process.

B. Simulation and Profiling

All design models in the SCE flow are executable for validation through simulation. Using the SpecC compiler and simulator (scc), models can be compiled and executed at any time. SCE also includes profiling tools to obtain feedback about design quality metrics. Based on a combination of static and dynamic analysis, a retargetable profiler (scprof) provides a variety of metrics across various levels of abstraction [25]. Initial dynamic profiling derives design characteristics through simulation of the input model. The tool then combines the obtained profiles with target PE, CE and bus characteristics. Thus, SCE profiling is retargetable for static estimation of complete system designs in linear time without the need for time consuming re-simulation or re-profiling.

The profiling results can also be back-annotated into the output model through refinement. By simulating the refined model, accurate feedback about implementation effects can then be obtained before entering the next design stage.

Since the system is only simulated once during the exploration process, the approach is fast yet accurate enough to make high-level decisions, since both static and dynamic effects are captured. Furthermore, the profiler supports multi-level, multi-metric estimation by providing relevant design quality metrics for each stage of the design process. Therefore, profiling guides the user in the design process and enables rapid and early design space exploration.

C. Databases

In the SCE design flow, the system is gradually refined using system components from a set of databases [26]. Specifically, SCE includes databases for processing elements (PEs), communication elements (CEs), operating system models, bus or other communication protocols, RTL units and software components. The database components are described as SpecC objects (behaviors or channels). The SpecC hierarchy for a component object in the database defines its structure and functionality for simulation and synthesis. In addition, meta-data, such as attributes, parameters and general information, is stored in the form of annotations attached to the components.

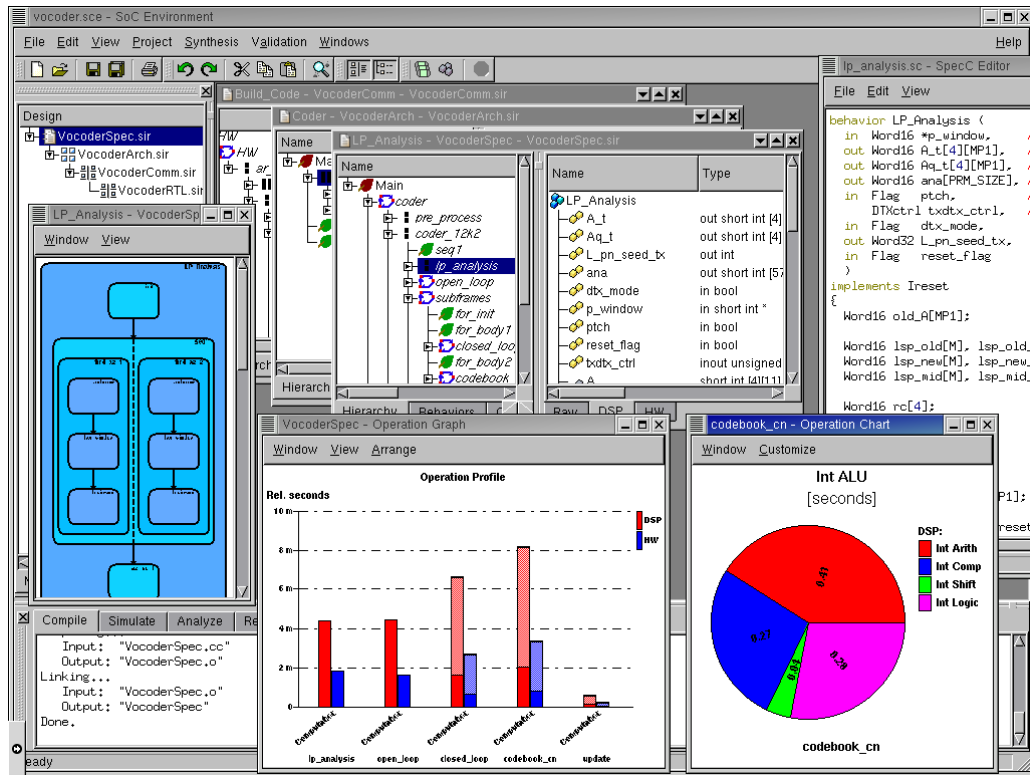


Fig. 4. SCE graphical user interface (GUI).

D. Model Generation and Refinement

At the core, SCE uses a set of refinement tools for automatic model generation. Based on the decisions annotated into the input design model (through the GUI, scripting or synthesis plugins), each refinement tool automatically generates a new output design model.

SCE includes refinement tools for architecture refinement (*scar*), OS refinement (*scos*), network refinement (*scnr*), communication refinement (*sccr*), and hardware (*scrtl*) and software (*sc2c*) synthesis (see details in Section III).

In general, model refinement in SCE is based on a layering of functionality. With each refinement step, a new implementation layer is inserted into the model, usually as an extra level of hierarchy. Each new level adds functionality in the form of additional design objects. Keeping implementation functionality organized as a stack of layers increases observability and transparency in the model. Note that later levels still can be merged or combined for optimization across layers.

E. Verification

SCE also integrates a formal verification tool *scver*. Our equivalence verification technology is based on *Model Algebra* [27], which is a formalism for symbolic representation and transformation of system level models. The formalism itself consists of a set of objects and composition rules. The objects are behaviors, synchronization channels, variables and ports. The composition rules for control flow, blocking and non-blocking communication, and hierarchy allow creation of formal models. Functionality preserving transformation rules

are also defined on model algebraic expressions. Each of these transformation rules are proven sound with respect to a trace-based notion of functional equivalence.

The incorporation of Model Algebra based verification in SCE follows the refinement flow. Well-formed models in SpecC can easily be translated to respective model algebraic expressions. Once the original and refined models are converted, *scver* applies the transformation rules to derive the original model from the refined model. The derived model is equivalent to the refined model by virtue of the soundness of the transformation rules. The derived model is then checked for isomorphism against the original model and the differences, if any, are reported. It must be noted that the number and order of transformation rules used for the model derivation step depend on the type of refinement. Since the key concept in SCE is the well-defined semantics of models at different abstraction levels, the order of transformation rules can be easily established. Therefore, equivalence verification becomes not only tractable, but straightforward.

F. Scripting Interface

SCE supports scripting of the complete environment from the command line without the need for the GUI. For scripting purposes, a GUI-less command shell, *scsh*, of SCE is available. *scsh* is based on the same libraries as the SCE GUI (not including the GUI layer itself), and offers interactive command-prompt based or automatic script based execution.

The SCE shell is based on an embedded Python interpreter that is extended with an API for low-level access to SCE core functionality and internals. For user-level scripting, a complete

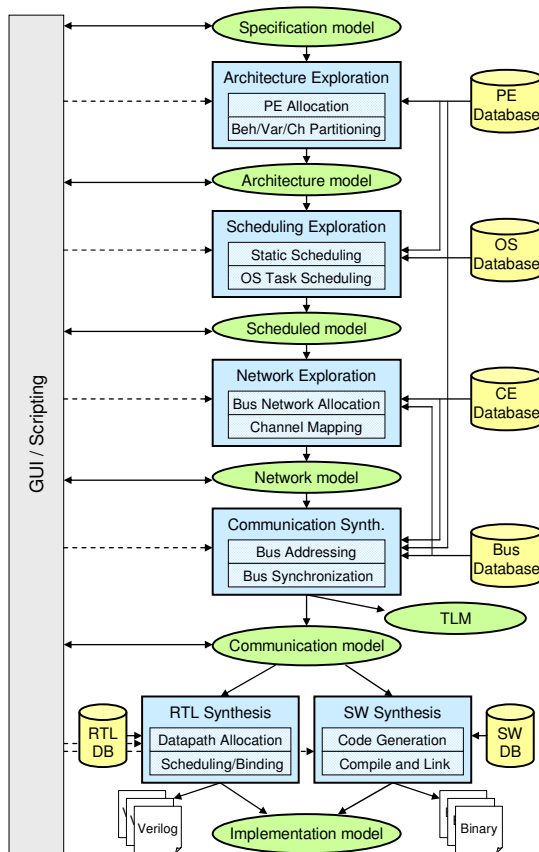


Fig. 5. Refinement-based tool flow in SCE.

set of high-level tools on top of the SCE shell are available. Provided scripts include command-line utilities for component allocation (`sce_allocate`), mapping/partitioning (`sce_map`), scheduling (`sce_schedule`), connectivity definition (`sce_connect`), component import (`sce_import`), and project handling (`sce_project`). These scripts provide a convenient command-line interface for all SCE high-level functionality and decision entry. Together with command-line interfaces to refinement tools and to the SpecC compiler, a complete scripting of the SCE design flow, through shell scripts or via Makefiles, is available.

III. SCE DESIGN FLOW

Fig. 5 shows the refinement-based tool flow in SCE from the initial abstract specification down to the final implementation model. In particular, the SCE flow consists of six specific tools which we will describe in the following sections.

A. Architecture Exploration

The first step in the SCE design flow, architecture exploration, defines the target platform and, under a set of design constraints, maps the computational parts of the specification model onto that platform. The target architecture consists of a set of PEs, i.e. software processors, custom hardware blocks and memories.

Given the specification model as input, architecture exploration consists of two tasks, PE allocation and partitioning. PE

allocation defines the target architecture by selecting system components (software and hardware processors, memories) from the PE database. Partitioning then maps behaviors and variables to the allocated PEs and memories, respectively. Furthermore, complex channels (e.g. queues and semaphores) are mapped to specific PEs for implementation in a client-server, remote procedure call (RPC) fashion.

Following the design decisions of PE allocation and partitioning, the SCE architecture refinement tool `scar` then automatically generates the output architecture model where the system components communicate through optimized abstract message-passing channels [28].

B. Scheduling Exploration

A key feature in the SCE design flow is the early evaluation of different scheduling strategies for software processors that are sequential and physically can only execute one task at a time. To evaluate different static and dynamic scheduling algorithms, such as round-robin or priority-based scheduling, we utilize a high level RTOS model on each processor in the system [29]. Our abstract RTOS model is written on top of the SpecC language and does not require any specific language extensions. It supports all the key concepts found in modern RTOS, including task management, real-time scheduling, pre-emption, task synchronization, and interrupt handling.

The SCE scheduling refinement tool `scos` automatically inserts the RTOS model with the user-defined scheduling strategy into the design model. The refined model can then be simulated for accurate observation of dynamic scheduling behavior in the multi-tasking system. Since our abstract RTOS model requires only minimal overhead in simulation time, this approach enables early and rapid design space exploration.

C. Network Exploration

Network exploration defines the system communication topology and maps the given communication channels onto a network of busses and communication elements (CEs), i.e. bridges and transducers. For this, network refinement inserts the required CEs from the database into the model and implements the end-to-end communication over point-to-point links between PEs and CEs [30].

In the input architecture model, PEs communicate via abstract, typed end-to-end channels and memory interfaces. During network exploration, communication media and bridges and transducers are allocated for the system busses and communication elements (CEs), respectively. Furthermore, the connectivity of PE and CE ports to the busses is defined. Finally, architecture-level end-to-end channels are mapped onto the allocated bus network.

Based on the network decisions, the SCE network refinement tool `scnr` generates the new network model such that it reflects the selected network topology. `scnr` also implements the top layers of the communication protocol stacks in each PE and CE in order to realize typed end-to-end architecture level communication over untyped point-to-point links between components in each network segment.

D. Communication Synthesis

Next, the task of communication synthesis is to implement the point-to-point logical links between stations over the actual bus media, and to select and define the final pin- and bit-accurate parameters of the communication architecture under a set of constraints. Communication refinement then inserts protocols and bus-functional component descriptions from the bus and PE/CE databases, respectively, and generates a refined communication model that implements the communication links in each network segment over the actual, shared bus protocol and bus wires. In addition to this pin-accurate model (PAM), our communication refinement also generates a fast-simulating TLM of the system, which abstracts away the pin-level details of individual bus transactions [30].

In the input network model, communication in each network segment is described as a set of logical links. During communication synthesis, the designer (through the GUI, scripting or using synthesis plugins) defines the bus parameters such as address and interrupt assignments for each logical link over each bus. Based on these decisions, the SCE communication refinement tool `sccr` inserts low-level (transaction-level down to pin-accurate) models of busses and components from the databases, and generates a new communication model (PAM or TLM) of the design. In the output model, PE and CE components are refined to implement the lower communication layers for synchronization, addressing and media accesses over each bus interface. On top of bus models from the bus database, the models hence implement all system communication down to the level of timing-accurate bus transactions (TLM) or cycle-accurate events for sampling/driving of bus wires (PAM).

E. RTL Synthesis

The task of RTL synthesis is to generate structural RTL from the behavioral description of the hardware components in the design. The SCE RTL synthesis tool `scrtl` supports automatic decision making through plugins. The designer can choose an algorithm to apply to all or only parts of their design. Critical parts of the design, on the other hand, can be manually pre-assigned or post-optimized [31].

Both designers and algorithms can rely on a set of estimates to aid them in the decision making. SCE includes RTL-specific profiling and analysis tools that provide feedback about a variety of metrics including delay, power, and variable lifetimes.

RTL synthesis in SCE takes full advantage of the designers' insight by allowing them to enter, modify or override their decisions at will. On the other hand, tedious and error-prone tasks including code generation are automated.

F. Software Synthesis

For implementing the software components in the system model, SCE relies on a layer-based modeling of the programmable processors and the software stack executing on them. Our embedded processor model supports task scheduling and interrupt handling [32].

Based on the layered modeling, the SCE software synthesis tool `sc2c` automatically generates embedded software code

for each processor from the system model [33]. More specifically, we generate efficient ANSI-C code from the SLDL code of the mapped application, and compile and link it against the selected RTOS. The resulting software binary can then be used for cycle-accurate instruction-set simulation within the system model, as well as for the final implementation.

IV. EXPERIMENTS AND RESULTS

We have applied SCE to a large set of industrial-strength examples. In the following, we will first demonstrate the SCE design flow in detail as applied to a case study. Next, we summarize our experiences with different examples and show exploration results. Finally, we will present a set of verification experiments.

A. Modeling Experiment

In order to demonstrate the overall SCE design flow, we have applied the flow to the example of a mobile phone baseband platform [34]. The specification model of the system is shown in Fig. 6. The design combines a JPEG encoder for processing of digital pictures taken by a camera and a voice encoder/decoder (vocoder) for speech processing based on the mobile phone GSM standard. Both JPEG and Vocoder processes are hierarchically composed of subbehaviors implementing the encoding and decoding algorithms in nested and pipelined loops and communicating through abstract message-passing channels. At the top level, a channel *Ctrl* between the two processes is used to send control messages from the JPEG encoder to the vocoder.

In the scheduled model obtained after architecture partitioning¹ and scheduling (Fig. 7), a *ColdFire* processor is running the JPEG encoder in software assisted by a hardware IP component for DCT (*DCT_IP*). Under control of the processor, a *DMA* component receives pixel stripes from the camera and puts them in the shared memory *Mem*. A *DSP* is running concurrent speech encoding and decoding tasks. Tasks are dynamically scheduled under the control of an operating system model [29] that sits in an additional OS layer *DSP_OS* of the DSP. Note that on the *ColdFire* side, no operating system is needed and the OS layer *CF_OS* is empty. The encoder on the DSP is assisted by a custom hardware coprocessor (*HW*) for the codebook search. Furthermore, four custom hardware I/O processors perform buffering and framing of the vocoder speech and bit streams.

Table I summarizes the design decisions made for implementing the communication channels in the example. As a result of the network exploration, the network is partitioned into one segment per subsystem with a transducer *Tx* connecting the two segments (Fig. 8). Individual point-to-point logical links connect each pair of stations in the resulting network model. Application channels are routed statically over these links where the *Ctrl* channel spanning the two subsystems is routed over two links via the intermediate transducer.

During communication synthesis, all links within each subsystem are implemented over a single shared medium. In

¹For space reasons, we omit the unscheduled architecture model here.

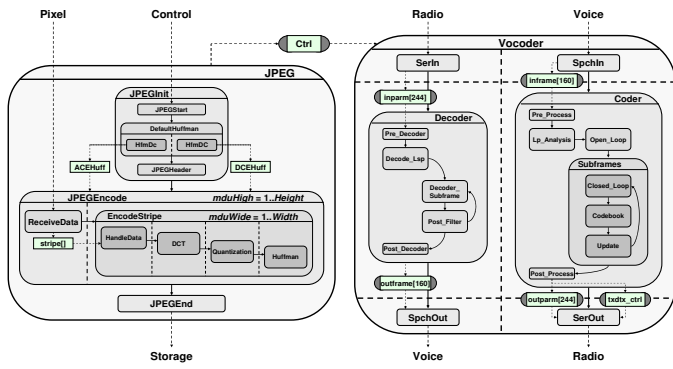


Fig. 6. Baseband example: specification model.

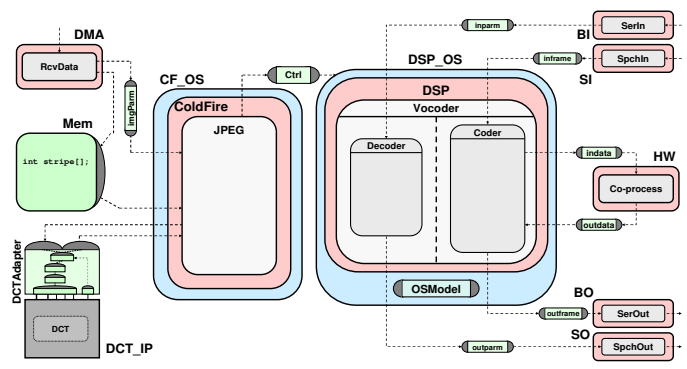


Fig. 7. Baseband example: scheduled architecture model.

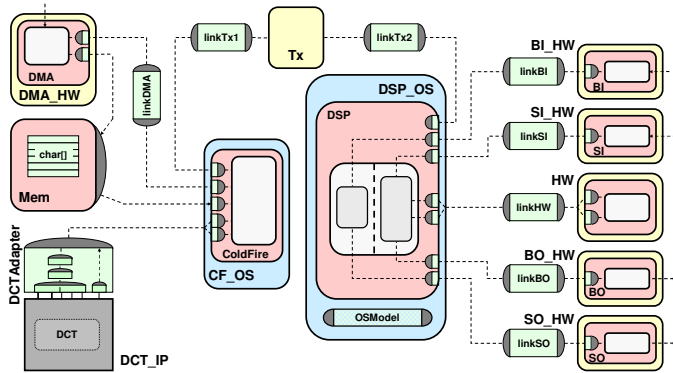


Fig. 8. Baseband example: network model.

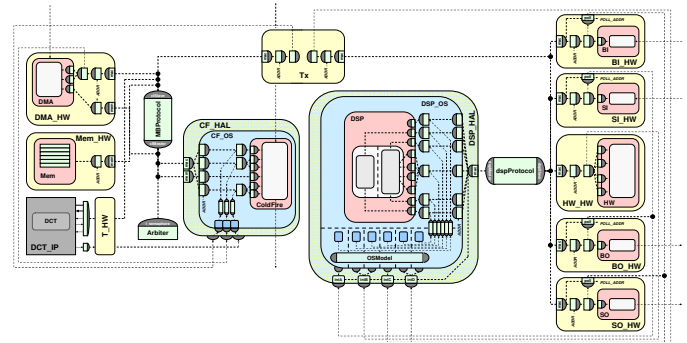


Fig. 9. Baseband example: transaction-level model (TLM).

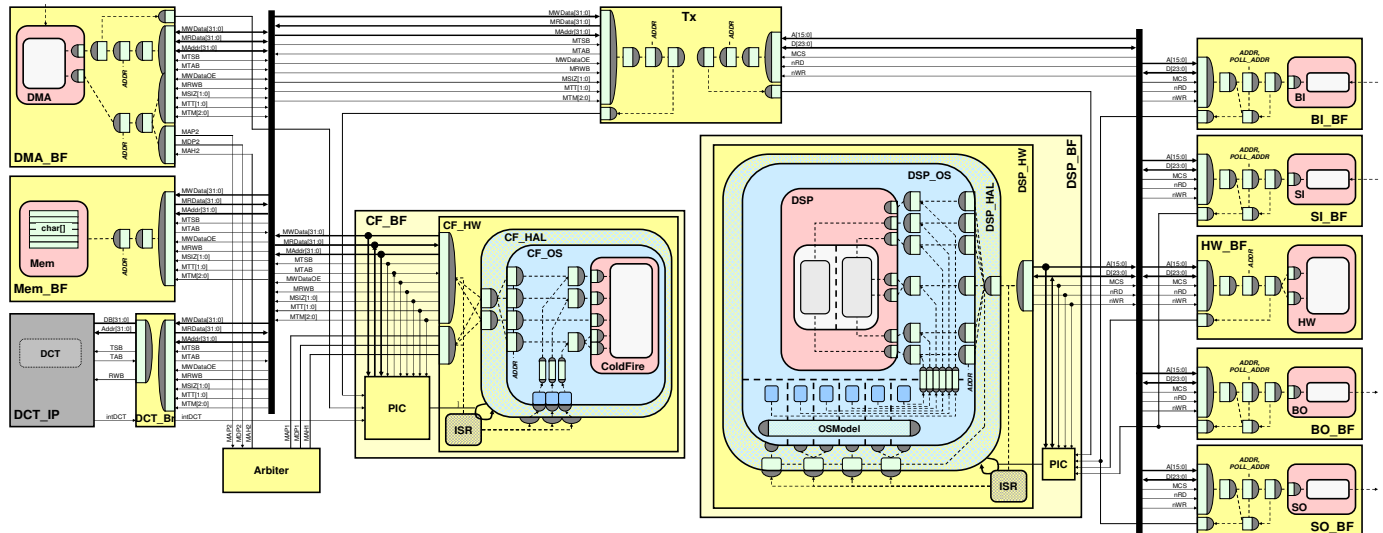


Fig. 10. Baseband example: pin-accurate model (PAM).

TABLE II
MODELING AND SIMULATION RESULTS FOR BASEBAND EXAMPLE.

| Model | ColdFire subsystem | | | DSP subsystem | | | System | |
|---------------|--------------------|-------------|------------|---------------|-------------|---------------|--------|-------------|
| | LOC | Simul. time | JPEG delay | LOC | Simul. time | Vocoder delay | LOC | Simul. time |
| Specification | 1,819 | 0.02 s | 0.00 ms | 9,736 | 1.31 s | 0.00 ms | 11,481 | 2.25 s |
| Architecture | 2,779 | 0.03 s | 9.66 ms | 11,121 | 1.21 s | 8.39 ms | 13,866 | 2.56 s |
| Scheduled | 3,098 | 0.02 s | 22.63 ms | 13,981 | 1.20 s | 12.02 ms | 17,020 | 2.00 s |
| Network | 3,419 | 0.02 s | 22.63 ms | 14,319 | 1.22 s | 12.02 ms | 17,658 | 2.03 s |
| TLM | 5,765 | 1.04 s | 24.03 ms | 15,668 | 27.4 s | 13.00 ms | 21,446 | 92.3 s |
| PAM | 5,916 | 14.3 s | 24.02 ms | 15,746 | 34.8 s | 13.00 ms | 21,711 | 2,349 s |
| RTL-C | 7,991 | 14.9 s | 23.48 ms | 23,661 | 147 s | 12.88 ms | 33,511 | 2,590 s |

TABLE I
COMMUNICATION DESIGN PARAMETERS FOR BASEBAND EXAMPLE.

| Channel | Network | | Link | | |
|----------|---------|------------|-------|--------|--|
| | Routing | Addr. | Intr. | Medium | |
| imgParm | linkDMA | 0x00010000 | int7 | cfBus | |
| stripe[] | Mem | 0x0010xxxx | - | | |
| hData | linkDCT | 0x00010010 | int1 | | |
| dData | | | | | |
| Ctrl | linkTx1 | 0x00010020 | int2 | dspBus | |
| | linkTx2 | 0xB000 | intA | | |
| inframe | linkSI | 0x800x | intB | | |
| outparm | linkBO | 0x950x | | | |
| indata | linkHW | 0xA000 | intD | | |
| outdata | | | | | |
| inparm | linkBI | 0x850x | intC | | |
| outframe | linkSO | 0x900x | | | |

both cases, the native ColdFire and DSP processor busses are selected as communication media. Within the segments, unique bus addresses and interrupts for synchronization are assigned to each link. On the ColdFire side, the memory is assigned a range of addresses with a base address plus offsets for each stored variable. On the DSP side, two of the four available interrupts are shared among the four I/O processors. In those cases, additional bus addresses for slave polling are assigned to each link (base address plus one). Finally, a bridge *DCT_Br* is inserted to translate between the *DCT_IP* and ColdFire bus protocols.

As a result, SCE communication synthesis generates two models, a fast-simulating TLM (Fig. 9), and a pin-accurate model (PAM, Fig. 10) for further implementation. In the TLM, link, stream, and media access layers are instantiated inside the OS and hardware layers of each station. Inside the processors, interrupt handlers that communicate with link layer adapters through semaphores are created. Interrupt service routines (*ISR*) together with models of programmable interrupt controllers (*PIC*) model the processor's interrupt behavior and invoke the corresponding handlers when triggered.

In the PAM, additionally the communication protocol layers are instantiated. Components are connected via pins and wires driven by the protocol layer adapters. On the ColdFire side, an additional arbiter component regulates bus accesses between the two masters, *DMA_BF* and *CF_BF*.

Table II summarizes the results for the example design. Using the refinement tools, models of the example design were automatically generated within seconds. A testbench common to all models was created which exercises the design by simultaneously encoding and decoding 163 frames of speech on the vocoder side while performing JPEG encoding of 30 pictures with 116x96 pixels. We created and refined both models of the whole system and models of each subsystem separately. Note that code sizes (Lines of Code, LOC) in each case include the testbenches. Since testbench code is shared, the size of the system model is less than the sum of the subsystem model sizes. All models were simulated on a 2.7 GHz Linux workstation using the QuickThreads version of the SpecC simulator.

Fig. 11 plots simulation times on a logarithmic scale, i.e. the graph shows that simulation times generally grow exponentially with each new model at the next lower level

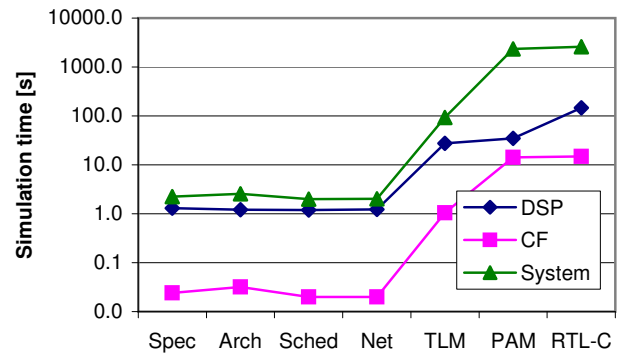


Fig. 11. Simulation speeds for the baseband example.

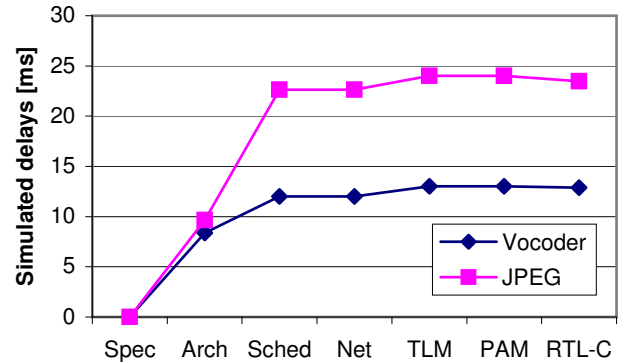


Fig. 12. Simulated delays in the baseband example.

of abstraction. On the other hand, results of simulated overall frame transcoding (back-to-back encoding and decoding) and picture encoding delays in the vocoder and JPEG encoder, respectively, are shown in Fig. 12. As can be seen, with each new model, measured delays linearly converge towards the final result.

Note that initial specification models are untimed and hence do not provide any delay measurements at all. Beginning with the architecture level, estimated execution delays are back-annotated into the computation blocks. As expected, scheduling has a large effect on simulation accuracy where abstract OS modeling enables evaluation of scheduling decisions at native simulation speeds². Depending on the relation of communication versus computation, introducing bus models and communication delays at the transaction-level further increases accuracy, potentially at the cost of significantly longer simulation times. On the other hand, TLMs allow for accurate modeling of communication close or equivalent to pin-accurate models but at higher speed.

Our results show that with increasing implementation detail at lower levels of abstraction, accuracy (as measured by the simulated delays) improves linearly while model complexities (as measured by code sizes and simulation times) grow exponentially. All in all, our results support the choice of intermediate models in the design flow that allow for fast validation of critical design aspects at early stages of the design process.

²Note that since the amount of simulated parallelism decreases, simulation is potentially even faster than at the specification level

TABLE III
DESIGN EXAMPLES AND TARGET ARCHITECTURES.

| Examples | | Buses (Masters → Slaves) |
|-----------|----|--|
| JPEG | A1 | CF → HW |
| Vocoder | A1 | DSP → HW |
| | A2 | DSP → HW1,HW2 |
| | A3 | DSP → HW1,HW2,HW3 |
| MP3float | A1 | CF → HW1 |
| | A2 | CF → HW1,HW2,HW3 HW1 ↔ HW3 HW2 ↔ HW3 |
| | A3 | CF → HW1,HW2,HW3,HW4 HW1 ↔ HW3 ↔ HW5 HW2 ↔ HW4 ↔ HW5 |
| MP3fix | A1 | ARM → 2 I/O |
| | A2 | ARM → 2 I/O, LDCT, RDCT |
| | A3 | ARM → 2 I/O, LDCT, RDCT LDCT ↔ I/O RDCT ↔ I/O |
| Baseband | A1 | DSP → HW, 4 I/O, T CF,DMA → Mem,BR,T,DMA BR → DCT_IP |
| Cellphone | A1 | ARM → 4 I/O, 2 DCT, T LDCT,RDCT → I/O DSP → HW, 4 I/O, T |

B. Exploration Experiments

In order to demonstrate our approach in terms of design space exploration for a wide variety of designs, we applied SCE to the design of six industrial-strength examples: stand-alone versions of the JPEG encoder (*JPEG*) and the GSM voice codec (*Vocoder*), floating- and fixed-point versions of an MP3 decoder (*MP3float* and *MP3fix*), the previously introduced baseband example (*Baseband*), and a *Cellphone* example combining the JPEG encoder, the MP3 decoder and the GSM vocoder in a platform mimicking the one used in the RAZR cellphone. For each example, we generated different architectures using Motorola DSP56600 (*DSP*), Motorola ColdFire (*CF*) and ARM7TDMI (*ARM*) processors together with custom hardware coprocessors (*HW*, *DCT*) and *I/O* units. We used various communication architectures with *DSP*, *CF*, *ARM* (AMBA AHB) and simple handshake busses.

Table III summarizes the features and parameters of the different design examples we tested. For each example, the target architectures are specified as a list of masters plus slaves for each bus in the system where the bus type is implicitly determined to be the protocol of the primary master on the bus. For example, in the case of the *MP3float* design, the ColdFire processor communicates with dedicated hardware units over its *CF* bus whereas the *HW* units communicate with each other through separate handshake busses. For simplicity, routing, address and interrupt assignment decisions are not shown in this table.

Table IV shows the results of exploration of the design space for the different examples. Overall model complexities are given in terms of code size using lines of code (LOC) as a metric. Results show significant differences in complexity between input and generated output models due to extra implementation detail added between abstraction levels.

Note that manual refinement would require tremendous effort (in the order of days). Automatic refinement, on the

other hand, completes in the order of seconds. Our results therefore show that a significant productivity gain can be achieved using SCE with automatic model refinement.

C. Verification Experiments

We implemented the SCE equivalence verification tool *scver* to verify the refinements above network level. Since the lowest abstraction level of communication in Model Algebra is the channel, models below network level in the SCE flow could not be directly translated into model algebraic representation.

The results for verification of architecture, scheduling and network refinements are presented in Table V. We used two benchmarks, namely the JPEG encoder and Vocoder as shown in column 1. The model algebraic representation was stored in a graph data structure, with nodes being the objects and edges being the composition rules. Column 5 shows the total transformations applied to derive Model 1 from Model 2 using the transformation rules of Model Algebra. As we can see, since the order of transformation is decided, it only took a few seconds to apply them even for representations with hundreds of nodes and edges. The verification time also includes the time it took to parse the SpecC models into model algebraic representation and to perform isomorphism checking between the derived and original model graphs.

The results demonstrate that the SCE tool flow based on well-defined model abstractions and semantics enables fast equivalence verification.

V. SUMMARY AND CONCLUSION

In this work, we have presented SCE, a comprehensive system design framework based on the SpecC language. SCE supports a wide range of heterogeneous target platforms consisting of custom hardware components, embedded software processors, dedicated IP blocks, and complex communication bus architectures.

The SCE design flow is based on a series of automated model refinement steps where the system designer makes the decisions and SCE quickly provides estimation feedback, generates new models automatically, and validates them through simulation and formal verification. The effective design automation tools integrated in SCE allow rapid and extensive design space exploration. The fast exploration capabilities, in turn, enable the designer to optimize the system architecture, the scheduling policies, the communication network, and the hardware and software components, so that an optimal implementation is reached quickly.

We have demonstrated the benefits of SCE by use of six industrial-size examples with varying target architectures, which have been designed and verified top-to-bottom. Compared to manual coding and model refinement, SCE achieves productivity gains by orders of magnitude.

SCE has been successfully transferred to and applied in industrial settings. SER, a commercial derivative of SCE has been developed and integrated into ELEGANT, an environment for electronic system-level (ESL) design of space and satellite electronics that was commissioned by the Japanese Aerospace Exploration Agency (JAXA). ELEGANT and SER

TABLE IV
RESULTS FOR EXPLORATION EXPERIMENTS.

| Examples | | Model Size (LOC) | | | | | Refinement Time | | | | |
|-----------|----|------------------|-------|-------|-------|-------|-----------------|--------|--------|--------|---------|
| | | Spec | Arch | Sched | Net | PAM | scar | scos | scnr | scrr | Total |
| JPEG | A1 | 1806 | 2409 | 2732 | 2780 | 4642 | 0.27 s | 0.37 s | 0.16 s | 0.21 s | 1.01 s |
| Vocoder | A1 | 7385 | 8449 | 9594 | 9775 | 10679 | 2.29 s | 1.30 s | 0.62 s | 0.56 s | 4.77 s |
| | A2 | | 8508 | 9632 | 9913 | 10989 | 2.41 s | 1.36 s | 0.75 s | 0.69 s | 5.21 s |
| | A3 | | 8535 | 9659 | 9949 | 11041 | 2.64 s | 1.69 s | 0.79 s | 0.64 s | 5.76 s |
| MP3float | A1 | 6900 | 6963 | 28190 | 28204 | 29807 | 0.82 s | 3.24 s | 0.90 s | 0.90 s | 5.86 s |
| | A2 | | 7181 | 28275 | 28633 | 31172 | 0.93 s | 2.66 s | 1.11 s | 1.48 s | 6.18 s |
| | A3 | | 11069 | 28736 | 30202 | 32795 | 4.66 s | 4.05 s | 7.10 s | 1.88 s | 17.69 s |
| MP3fix | A1 | 13363 | 13724 | 17131 | 17270 | 21593 | 0.95 s | 1.37 s | 0.58 s | 0.95 s | 3.85 s |
| | A2 | | 16040 | 18300 | 18564 | 23228 | 3.28 s | 1.68 s | 0.85 s | 1.20 s | 7.01 s |
| | A3 | | 16023 | 18748 | 19079 | 24471 | 2.72 s | 1.76 s | 1.97 s | 0.95 s | 7.40 s |
| Baseband | A1 | 11481 | 13866 | 17020 | 17658 | 21711 | 4.27 s | 2.46 s | 1.24 s | 1.02 s | 8.99 s |
| Cellphone | A1 | 16441 | 18653 | 21936 | 22570 | 30072 | 3.86 s | 3.10 s | 1.31 s | 1.22 s | 9.49 s |

TABLE V
RESULTS FOR EQUIVALENCE VERIFICATION.

| Examples | Refinement | Model 1 | | | Model 2 | | | #Transformations | Verification Time |
|----------|--------------|---------|--------|--------|---------|--------|--------|------------------|-------------------|
| | | Type | #nodes | #edges | Type | #nodes | #edges | | |
| JPEG | Architecture | spec. | 148 | 219 | arch. | 180 | 257 | 1602 | 1.6 sec. |
| | Scheduling | arch. | 180 | 257 | sched. | 180 | 287 | 2740 | 2.1 sec. |
| | Network | sched. | 180 | 287 | net. | 201 | 253 | 2852 | 2.1 sec. |
| Vocoder | Architecture | spec. | 436 | 761 | arch. | 528 | 882 | 6131 | 3.3 sec. |
| | Scheduling | arch. | 528 | 882 | sched. | 528 | 881 | 7065 | 3.7 sec. |
| | Network | sched. | 528 | 881 | net. | 569 | 933 | 7229 | 3.8 sec. |

have been successfully delivered to JAXA's suppliers and are currently being introduced into the general market [35].

Acknowledgements

The authors would like to thank all members of the CECS SpecC group who have contributed to SCE over the years. Special thanks go to David Berner, Pramod Chandraiah, Quoc-Viet Dang, Alexander Gluhak, Eric Johnson, Raphael Lopez, Gunar Schirner, Ines Viskic, Shuqing Zhao, and Jianwen Zhu.

REFERENCES

- [1] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao, *SpecC: Specification Language and Design Methodology*. Kluwer Academic Publishers, 2000.
- [2] A. Gerstlauer, R. Dömer, J. Peng, and D. D. Gajski, *System Design: A Practical Guide with SpecC*. Kluwer Academic Publishers, 2001.
- [3] A. Gerstlauer and R. Dömer, *SCE Specification Model Reference Manual, Version 2.2.0 beta*, Center for Embedded Computer Systems, University of California, Irvine, July 2003.
- [4] P. Coste, F. Hessel, P. L. Marrec, Z. Sugar, M. Romdhani, R. Suescun, N. Zergainoh, and A. A. Jerraya, "Multilanguage design of heterogeneous systems," in *Proceedings of the International Symposium on Hardware-Software Codesign (CODES)*, Rome, Italy, May 1999.
- [5] P. Gerin, S. Yoo, G. Nicolescu, and A. A. Jerraya, "Scalable and flexible cosimulation of SoC designs with heterogeneous multi-processor target architectures," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, Yokohama, Japan, January 2001.
- [6] *ModelSim SE User's Manual*, Mentor Graphics Corp.
- [7] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *International Journal of Computer Simulation*, vol. 4, no. 2, pp. 155–182, April 1994.
- [8] T. Grötter, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*. Kluwer Academic Publishers, 2002.
- [9] M. Aubury, I. Page, G. Randall, J. Saul, and R. Watts, *Handel-C Language Reference Guide*. Oxford University Computing Laboratory, August 1996.
- [10] F. Ghenassia, *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Springer, 2005.
- [11] A. Österling, T. Brenner, R. Ernst, D. Herrmann, T. Scholz, and W. Ye, "The COSYMA system," in *Hardware/Software Co-Design: Principles and Practice*, J. Staunstrup and W. Wolf, Eds. Kluwer Academic Publishers, 1997.
- [12] C. A. Valderrama, M. Romdhani, J.-M. Daveau, G. F. Marchioro, A. Changel, and A. A. Jerraya, "Cosmos: A transformational co-design tool for multiprocessor architectures," in *Hardware/Software Co-Design: Principles and Practice*, J. Staunstrup and W. Wolf, Eds. Kluwer Academic Publishers, 1997.
- [13] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara, *Hardware-Software Co-Design of Embedded Systems: The POLIS Approach*. Kluwer Academic Publishers, 1997.
- [14] G. Vanmeerbeeck, P. Schaumont, S. Vernalde, M. Engels, and I. Bolsens, "Hardware/software partitioning for embedded systems in OCAPI-xt," in *Proceedings of the International Symposium on Hardware-Software Codesign (CODES)*, Copenhagen, Denmark, April 2001.
- [15] P. Schaumont, S. Vernalde, L. Rijnders, M. Engels, and I. Bolsens, "A programming environment for the design of complex high speed ASICs," in *Proceedings of the Design Automation Conference (DAC)*, San Francisco, CA, June 1998.
- [16] W. O. Cesário, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, L. Gauthier, and M. Diaz-Nava, "Multiprocessor SoC platforms: A component-based design approach," *IEEE Design and Test of Computers*, vol. 19, no. 6, November/December 2002.
- [17] D. Lyonnard, S. Yoo, A. Baghdadi, and A. A. Jerraya, "Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip," in *Proceedings of the Design Automation Conference (DAC)*, Las Vegas, NV, June 2001.
- [18] K. van Rompaey, D. V. I. Bolsens, and H. D. Man, "CoWare: A design environment for heterogeneous hardware/software systems," in *Proceedings of the European Design Automation Conference (EuroDAC)*, Geneva, Switzerland, September 1996.
- [19] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: An Integrated Environment for Electronic System Design," *IEEE Computer*, vol. 36, no. 4, April 2003.
- [20] A. L. Sangiovanni-Vincentelli, "Quo Vadis SLD: Reasoning about Trends and Challenges of System-Level Design," *Proceedings of the IEEE*, vol. 95, no. 3, pp. 467–506, March 2007. [Online]. Available: <http://chess.eecs.berkeley.edu/pubs/263.html>
- [21] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems*. Prentice Hall, 1994.
- [22] L. Cai, A. Gerstlauer, S. Abdi, J. Peng, D. Shin, H. Yu, R. Dömer, and

- D. Gajski, "System-on-chip environment (SCE version 2.2.0 beta): Manual," Center for Embedded Computer Systems, University of California, Irvine, Tech. Rep. CECS-TR-03-45, December 2003.
- [23] S. Abdi, J. Peng, H. Yu, D. Shin, A. Gerstlauer, R. Dömer, and D. Gajski, "System-on-chip environment (SCE version 2.2.0 beta): Tutorial," Center for Embedded Computer Systems, University of California, Irvine, Tech. Rep. CECS-TR-03-41, July 2003.
- [24] I. Viskic and R. Dömer, "A Flexible, Syntax Independent Representation (SIR) for System Level Design Models," in *Proceedings of the EuroMicro Conference on Digital System Design*, Dubrovnik, Croatia, Aug. 2006.
- [25] L. Cai, A. Gerstlauer, and D. D. Gajski, "Retargetable profiling for rapid, early system-level design space exploration," in *Proceedings of the Design Automation Conference (DAC)*, San Diego, CA, June 2004.
- [26] A. Gerstlauer, L. Cai, D. Shin, H. Yu, J. Peng, and R. Dömer, *SCE Database Reference Manual, Version 2.2.0 beta*, Center for Embedded Computer Systems, University of California, Irvine, July 2003.
- [27] S. Abdi and D. Gajski, "Verification of System Level Model Transformations," *Springer International Journal of Parallel Programming*, vol. 34, no. 1, pp. 29–59, March 2006.
- [28] J. Peng and D. D. Gajski, "Optimal message passing for data coherency in distributed architecture," in *Proceedings of the International Symposium on System Synthesis*, October 2002, pp. 20–25.
- [29] A. Gerstlauer, H. Yu, and D. D. Gajski, "Rtos modeling for system level design," in *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, Munich, Germany, March 2003.
- [30] A. Gerstlauer, D. Shin, J. Peng, R. Dömer, and D. D. Gajski, "Automatic Layer-Based Generation of System-On-Chip Bus Communication Models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 26, no. 9, pp. 1676–1687, September 2007.
- [31] D. Shin, A. Gerstlauer, R. Dömer, and D. D. Gajski, "An Interactive Design Environment for C-based High-level Synthesis of RTL Processors," *IEEE Transactions on VLSI Systems*, 2007, accepted for publication, June 22, 2007.
- [32] G. Schirner, A. Gerstlauer, and R. Dömer, "Abstract, Multifaceted Modeling of Embedded Processors for System-Level Design," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, Yokohama, Japan, Jan. 2007.
- [33] H. Yu, R. Dömer, and D. Gajski, "Embedded software generation from system level design languages," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, Yokohama, Japan, Jan. 2004.
- [34] A. Gerstlauer, "Modeling flow for automated system design and exploration," Ph.D. dissertation, University of California, Irvine, U.S.A., April 2004.
- [35] *CECS eNews Volume 7, Issue 3*, <http://www.cecs.uci.edu/enews/CECSeNewsJul07.pdf>, Center for Embedded Computer Systems, University of California, Irvine, July 2007.