Chapter 8

EMBEDDED DESIGN PRACTICE

Both commercial and academic tools are available for the design of embedded systems. These tools come in three categories: system-level design, software design, and hardware design.

In this chapter, we will discuss the tools and frameworks available for these various examples of system design. We will also present examples of embedded system design and results for applications, such as JPEG encoder and an MP3 decoder. These results demonstrate the potential impact of the embedded system modeling, synthesis and verification technologies that have been discussed in this book.

8.1 SYSTEM LEVEL DESIGN TOOLS

The semiconductor revolution would not have been sustainable without the help of Electronic Design Automation (EDA) tools. Historically, the breakthrough of EDA came with the availability of the first Computer-Aided Design (CAD) tools for hardware synthesis (see Section 8.3). As we move to higher and higher levels of abstraction, new classes of tools gradually emerged with each new level. In recent years, we have seen a push towards development of so-called Electronic System-Level (ESL) tools. However, while there are many approaches that claim to provide ESL solutions, such as C-to-RTL tools implementing high-level synthesis of a single hardware unit (described in more detail in Section 8.3), true system-level solutions have to span the complete design space across hardware and software boundaries.

As described in detail throughout this book, a system-level design flow is typically separated into two parts: a frontend and a backend. The system design

D.D. Gajski et al., *Embedded System Design: Modeling, Synthesis and Verification*,
DOI: 10.1007/978-1-4419-0504-8_8,

frontend takes a description of the application and target architecture at its input. Applications are given in some MoC to describe the desired system behavior to be implemented. Target architectures can be given in the form of architectural constraints, associated parameters, architecture templates or complete pre-defined system-level netlists. In the frontend, application computation and communication is then mapped onto and implemented on the selected or synthesized target architecture. In the process, Design Space Exploration (DSE) is performed to optimize design metrics under a set of constraints. At the output of the frontend, models of the system at various levels of abstraction are generated for virtual prototyping of the system design. Predominantly, such system models will be TLMs described in some SLDL such as SystemC. Models can be simulated or analyzed to provide feedback about the feasibility and quality of the generated design. In addition, modeling guidelines such as the SystemC TLM standard [150] promise to enable easy exchange of component or design models between companies or design divisions and across tool chains.

In the backend, high-level system descriptions are then further synthesized down to a hardware or software implementation for each PE in the system. ESL design flows thereby rely on the availability of corresponding software or hardware synthesis tools (see Section 8.2 and Section 8.3, respectively). On the software side, final target binaries for each processor are produced. On the hardware side, high-level synthesis of behavioral, C-based component models down to RTL descriptions is performed. In both cases, synthesized PE models can be re-integrated into system TLMs for cycle-accurate co-simulation with the rest of the systems. On the software side, binaries are executed in an ISS that is integrated into the overall system simulation environment. On the hardware side, RTL or gate-level models in SLDL form are inserted for this purpose. As a result, a virtual prototype of the system platform is generated.

In the end, however, the desired result at the output of a system-level design flow is a physical system prototype or a system implementation that is ready for further manufacturing. Therefore, generated software binaries should be ready to be directly loaded into target processors and RTL models should be created in the form of standard HDL code (e.g., VHDL or Verilog) such that they can feed into traditional logic and physical synthesis processes.

Overall, being based on existing commercial or proprietary backend tools, the goal of system-level design tools is to develop and apply design automation techniques to the steps in the frontend. At any level, the first set of tools to always emerge are modeling and simulation solutions that allow designers to capture models and execute them in a validation environment. Consequently, most currently available commercial system-level approaches are focused on providing models and simulators either at the application, SLDL/TLM or HDL/RTL/ISS level. Looking ahead, academic research, in contrast, is aimed at the development of subsequent system-level synthesis and verification tools, which build

on modeling solutions to provide an automated path from abstract system specification down to synthesized system models and eventually a system prototype or implementation.

8.1.1 ACADEMIC TOOLS

METROPOLIS

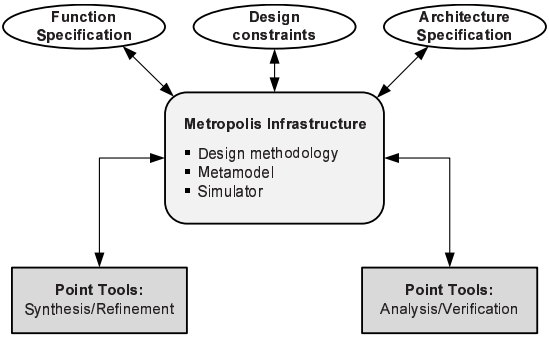

FIGURE 8.1 Metropolis framework

Metropolis [12] is a modeling and simulation environment originally developed at UC Berkeley. Metropolis is based on a Platform-Based Design (PBD) paradigm (Figure 8.1) [164] in which the target system architecture, called a platform, is assumed to be given or at least significantly pre-determined at the input of the system design flow. This constrains and simplifies the design space exploration process. In addition, a pre-defined and pre-determined platform facilitates the reuse of common design patterns across different design instances. Therefore, PDB follows a meet-in-the-middle approach and the system design problem is reduced to the mapping of a desired function onto the given target platform to create a specific design instance.

Metropolis provides a general, proprietary metamodeling language that is used to capture separate models for functionality (system application behavior), architecture and their mapping. The metamodel employs a fundamental discrete event-based execution model with concurrent processes communicating through channels (called media). In a similar manner to other SLDLs, functionality is described in the form of event-driven process networks that are general in the sense that many classes of MoCs can be represented. In addition, functionality can be annotated with non-functional constraints. The architecture is defined by using processes and media to describe available services (e.g., tasks) and resources (e.g., CPUs, memories or buses), respectively. Quantities can be associated with the architecture to model metrics such as delays. Finally,

given a system functionality and architecture, synthesis or refinement is performed by defining a mapping between the two in the Metropolis metamodel as a set of additional constraints synchronizing their event execution.

Metropolis itself does not define any specific design tools but rather a general framework and language for modeling with support for simulation, validation and analysis of models. Metropolis includes a frontend for parsing of metamodels and a backend for translation of metamodels into C++/SystemC simulation code. In addition, several backend point tools have been integrated into the Metropolis environment to support automatic scheduling, communication design, verification, or hardware synthesis. For example, the xPilot system (see Section 8.3.1) can be plugged into Metropolis to provide high-level synthesis of hardware blocks.

SYSTEMCODESIGNER

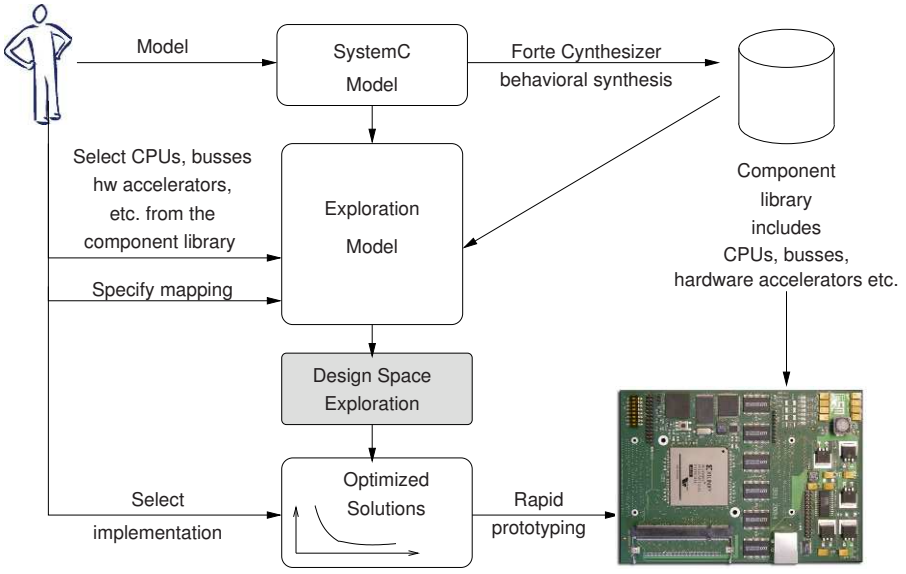

FIGURE 8.2 SystemCoDesigner tool flow

SystemCoDesigner is a system-level design space exploration environment developed at the University of Erlangen-Nuremberg in Germany (Figure 8.2) [105]. At its input, SystemCoDesigner supports applications written in a dynamic dataflow oriented MoC targeted towards streaming applications. Such input models are captured using a well-defined subset of SystemC called SysteMoC. In SysteMoC, applications are modeled as a graph of atomic actors that communicate via FIFO queues. Internally, the behavior of each actor is described in the form of an FSMD. In contrast to SDF models, SysteMoC sup-

ports applications in which actor production and consumption rates can vary dynamically at runtime. Thus, the SysteMoC model is similar to a KPN with the restriction of atomic process executions.

Once the application has been defined, SystemCoDesigner will automatically generate a library of software and hardware implementations of all actors. Software implementations are created through simple transformation of the SysteMoC input into C code. On the hardware side, Forte's Cynthesizer tool (see Section 8.3.2) is used for high-level synthesis of all actors down to RTL descriptions. All generated actor implementations are stored in a component library and are annotated with performance, area and other metrics obtained during synthesis.

Given an application, the annotated component library, and an architecture template, SystemCoDesigner can perform a fully automatic, multi-objective exploration of the design space. With the architecture template, the designer can thereby constrain the search space and restrict possible target architectures in terms of the number and type of available processors or the allowed mappings of actors to processor types. Design space exploration is performed using genetic algorithms to drive and guide the automatic search process. For every new candidate architecture selected by the search, a SystemC performance TLM is automatically generated and simulated. The generated virtual architecture model represents the mapping and scheduling of actors on the selected processors, where actors are annotated with corresponding estimated performance metrics from the component library. Simulation results are then fed back into the search algorithm to evaluate the current design point and direct the next iteration of the evolutionary exploration process.

As a result of the exploration process, a set of Pareto-optimal design solutions is obtained and presented to the user. From this optimal set, the designer can visualize the design space and subsequently select an applicable implementation option. After an architecture has been chosen, SystemCoDesigner can prototype the selected implementation on a Xilinx FPGA platform. The platform is assembled, and pre-synthesized hardware implementations of respective actors are inserted. For actors mapped into software, code is generated, compiled and linked together with other actors into a binary for each processor. Finally, the resulting bitstream is downloaded into the FPGA for rapid prototyping of the final target implementation.

DAEDALUS

Daedalus [145] is another system-level design environment targeted towards streaming, multimedia-type applications. Deadalus is a joint project between the University of Amsterdam and Leiden University in the Netherlands. It combines several tools under a common, XML-based infrastructure to provide application capture, modeling and simulation, and backend platform synthesis

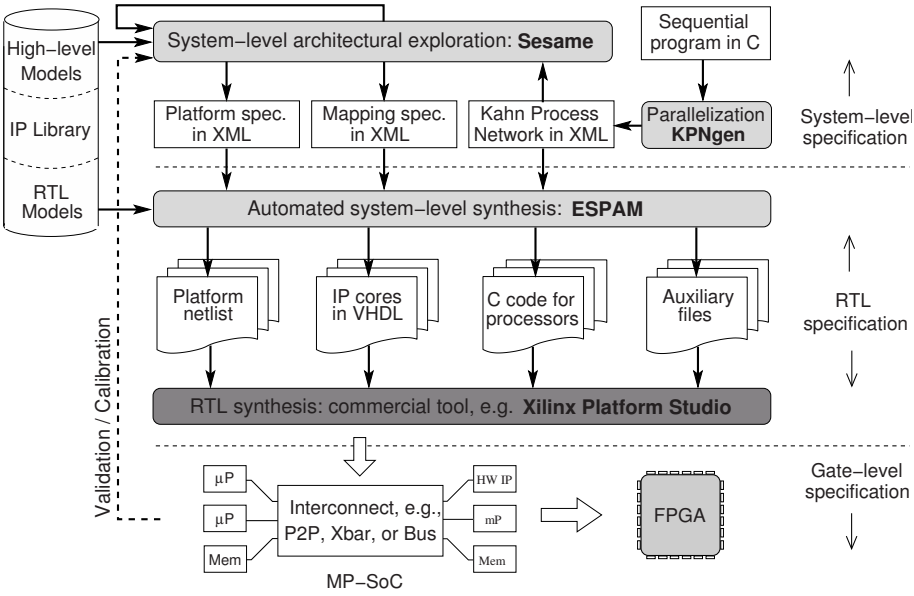

FIGURE 8.3 Daedalus tool flow

functionality (Figure 8.3). At its input, Daedalus accepts applications modeled in a KPN MoC (see Section 3.1.1) that is represented in an XML format. In addition, through a tool called KPNgen, Daedalus can perform automatic conversion of a well-defined subset of sequential C descriptions into a parallelized KPN suitable for input into the Daedalus design flow.

Daedalus supports target architectures consisting of multiple programmable processors and pre-defined hardware IPs. IP components are stored in a library that contains both high-level, functional as well as RTL component models. Given an input KPN and an IP library, a modeling and simulation tool called Sesame allows the designer to assemble a target architecture and perform a mapping of KPN processes onto architectural components. In case multiple processes are mapped to the same processor, Sesame will try to statically schedule processes or insert a lightweight OS kernel. For performance evaluation purposes, processor and IP models in the component library are annotated with tables of estimated execution latencies for typical function-level operations. Sesame links KPN processes to operational latencies of library components they are mapped to. As a result, Sesame will automatically generate a high-level, timed simulation TLM of the specified platform for quick evaluation of selected candidate target architectures. Sesame also allows integration of low-level component models such as cycle-accurate ISSes into the simulation environment. Furthermore, Sesame supports optional automation of the design

space exploration process through analytical design space pruning and heuristic search methods such as genetic algorithms.

Given a KPN application, a platform architecture specification and an application-to-architecture mapping (all in XLM form), a final backend synthesis tool called ESPAM automatically generates a description of the selected system implementation. Pre-defined RTL models of all hardware IPs are pulled out of the component library and C code is generated for all KPN processes mapped to programmable processors. Finally, code for each processor is compiled and a hardware models are assembled into a system VHDL model for further synthesis, download and prototyping on an FPGA platform.

PEACE

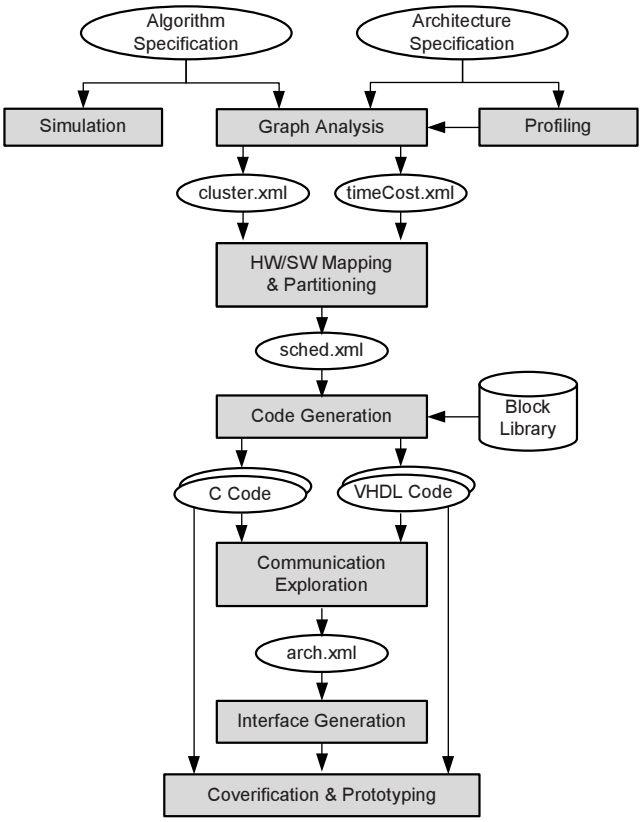

FIGURE 8.4 PeaCE tool flow

PeaCE (Ptolemy extension as a Codesign Environment) [83] is yet another hardware/software co-design framework targeted towards multimedia applications. As the name implies, it is based on Ptolemy [28] as the framework for

modeling applications. Ptolemy is a general framework for composition and co-simulation of a wide variety of heterogeneous MoCs in a hierarchical fashion. However, of the many MoCs supported in Ptolemy, PeaCE only accepts combinations of extended SDF and FSM models at its input.

PeaCE realizes a codesign flow from specification over system synthesis down to system prototyping in several steps (Figure 8.4). In a first step, the Ptolemy application model is translated into C code for functional simulation at the specification level. In addition, given a user-defined architecture template consisting of a list of processing elements, performance estimates of application tasks are obtained by profiling each functional block on an ISS of each processor. Annotated application and architecture specifications entered through the user interface are then translated into a generic XML-based format. Operating on this intermediate representation, automatic or manual component selection and HW/SW partitioning is performed. During this step, communication overhead is assumed to be proportional to amount of data transferred. Resulting mapping and scheduling information is stored in another XML-based, intermediate file. Based on this information, code for all processing elements is generated and co-simulated to obtain memory and communication traces. Next, traces are used to drive manual or automatic communication architecture exploration, results of which are stored in an XML-based architecture description. Finally, hardware and software interfaces are generated and the complete system platform is assembled for accurate co-simulation or FPGA-based prototyping.

PeaCE has recently been extended towards multi-processor software development in a framework called HOPES [115]. At its input, HOPES supports a parallel programming model called Common Intermediate Code (CIC), where CIC code can be generated from extended UML descriptions or Ptolemy-based PeaCE application models. CIC provides a high-level, rich and generic API for control or data-oriented code parallelization and inter-process communication. Generic CIC descriptions can be automatically translated into optimized, platform-specific code for a given multi-processor target architecture. Generated code can then be simulated in an ISS-based virtual prototyping environment or downloaded into the real processors of the chosen MPSoC platform.

SCE

The System-on-Chip Environment (SCE) [52] was developed at UC Irvine as the successor of the SpecSyn [64, 63] tool set (the successor of SCE, called ESE, is described in Section 8.4.1). Both SpecSyn and SCE support a PSM MoC (see Section 3.1.2) at their inputs and follow a Specify-Explore-Refine methodology (see Section 1.3). SpecSyn is based on a PSM extension of VHDL called SpecCharts [185]. In contrast, SCE uses the C-based SpecC SLDL (see Section 3.2.3) as the basis for describing all design models throughout the complete design flow. The SpecC language and technology has been standardized

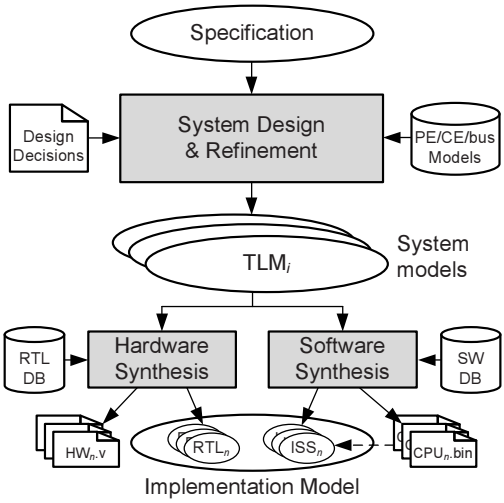

FIGURE 8.5 SCE tool flow

[51] and a derivative of the SCE system-level design frontend has been commercialized and integrated into a complete SpecC-based ESL design solution commissioned by the Japanese Aerospace Exploration Agency (JAXA) [73].

As shown in Figure 8.5, SCE consists of a system design frontend and hardware/software synthesis backend. The design process starts with an abstract specification of the desired system functionality written in SpecC PSM form. In the interactive frontend, the specification is automatically compiled down onto a user-defined MPSoC architecture through a series of architecture, scheduling, network and communication exploration and synthesis steps.

Design decisions such as allocation of architecture components out of the PE, CE and bus databases, scheduling of processes, and mapping of specification processes and channels onto allocated PEs, CEs and buses are entered by the designer through a scripting or graphical user interface. To aid the user in the exploration process, SCE includes retargetable profiling and estimation tools that provide feedback about specification characteristics and effects of decisions on design quality metrics. In addition, SCE supports a plugin mechanism for inclusion of optimizing algorithms that perform automated decision-making.

At its output, the SCE frontend automatically generates TLMs of the system design at successively lower levels of abstractions following a gradual, stepwise refinement processes. Automatically generated TLMs integrate high-level performance models with timing-annotated processes running on top of abstract OS and processor models to provide fast yet accurate analysis and design validation without the need for slow instruction-set simulation.

In a backend process, hardware and software processors in the TLMs are then individually synthesized further down to their cycle-accurate RTL and instruction set implementations, respectively. On the hardware side, application processes and automatically generated bus interfaces are synthesized into VHDL or Verilog descriptions following a high-level hardware synthesis process. Resulting RTL models are ready to be further synthesized and manufactured following traditional logic and physical design processes. On the software side, code for application tasks, middleware and bus drivers is automatically synthesized into final target binaries ready for download into the processors. In addition, a cycle-accurate implementation model of the system is generated that allows for co-simulation of hardware RTL models with software instruction-set simulators (ISSs) running final target binaries.

8.1.2 COMMERCIAL TOOLS

COFLUENT

CoFluent Studio by CoFluent Design [43] is a commercial spin-off based on the MCSE methodology (Méthodologie de Conception des Systèmes Electroniques, also known as CoMES, Co-design Methodology for Electronic Systems) and tool set originally developed at the University of Nantes in France [33]. CoFluent studio is a modeling and simulation environment for early, high-level design space exploration. As a graphical frontend for SystemC, it allows capturing of application functionality, system architecture and their mapping. Application models are specified as networks of timed processes. Processes are described purely by annotated delay estimates, by their functionality given in the form of C, C++ or SystemC code, or as a combination of both timing and behavior. Processes communicate through high-level, message-passing channels, queues, events and shared variables that can also be annotated with estimated communication latencies. The resulting application model can be simulated for early functional and performance evaluation supported by a rich set of built-in graphical analysis and visualization capabilities.

In the next step, an architecture platform can be graphically defined and assembled out of generic processing element or interconnect components. Through drag-and-drop, the designer can map application elements onto the specified platform, and CoFluent studio will generate a SystemC TLM of the resulting architecture for simulation and virtual prototyping. In contrast to other approaches (see below), no detailed component, ISS or bus models are employed. Instead, computation and communication remains at a high level, described as time-annotated processes and message-passing transactions. CoFluent Studio does, however, insert network-level models of communication stacks and OS models for dynamic scheduling of processes mapped onto software pro-

cessors. All combined, this allows for fast timed simulation at early stages of the design process (at the expense of reduced accuracy).

SPACE CODESIGN

Space Codesign is a recent startup coming out of the École Polytechnique de Montréal in Canada [170]. Its main product is SpaceStudio, which provides a SystemC-based system-level integrated development environment (IDE) built on top of Eclipse (see Section 8.2.1). A specific focus of Space Codesign is support for the increasingly important embedded software development process. Designers can create process-based SystemC application models out of pre-defined library blocks or by importing and wrapping existing C, C++ or SystemC code, where application processes can communicate through message-passing or shared memory channels. Next, a system architecture can be graphically assembled and the application can be partitioned by dragging application blocks onto previously allocated hardware or software processors. As a result, SpaceStudio (through a tool called Elix) will generate a SystemC TLM of the chosen platform where timing-annotated processes are grouped into bus-functional processor blocks, integrated with an OS simulation and connected through register- and cycle-accurate bus models.

All SystemC application models and TLMs generated through SpaceStudio can be simulated for analysis and performance evaluation. High-level models are based on native, host-compiled execution of application processes for fast simulation. In addition, a tool called Simtek will allow creation of cycle-accurate, transaction-level virtual platforms by replacing host simulation of software processes with processor ISS models. Finally, a tool called GenX will take virtual platform TLMs create through SpaceStudio and synthesize them down to a Xilinx FPGA prototyping platform. Software processes are compiled for the selected processor and linked against the target RTOS. Hardware IPs are replaced with pre-designed RTL descriptions, and custom hardware blocks are synthesized using third-party high-level synthesis tools such as Mentor Catapult or Forte Cynthesizer (see Section 8.3.2). Finally, components are assembled into a system netlist for input to the Xilinx FPGA platform synthesis process.

COWARE

CoWare technology started originally as a project at IMEC in Belgium to develop a process-based system-level modeling framework [188]. Since its commercialization, CoWare has evolved into a suite of products that provide a frontend for SystemC TLM capture, modeling and simulation [46]. At the core of the product portfolio, the CoWare Platform Architect is a graphical environment for capturing and assembling virtual system platform models at the cycle-approximate implementation level. CoWare includes an extensive li-

brary of detailed component models for hardware IPs, programmable processors and system buses. Hardware IPs are provided either in RTL or bus-functional behavioral form. For programmable processors, ISS models are employed. With the acquisition of LISATek [91], CoWare gained the capability to design application-specific and configurable embedded processors, including generation of associated custom ISSes and software tool chains. Different component models are integrated under a standard SystemC TLM framework using register- and protocol-accurate transactional interconnect models.

Virtual platform models captured and assembled through CoWare's Platform Architect and associated Model Library and Processor Designer can then be simulated using CoWare's own performance-optimized SystemC simulation kernel. Platform Architect thereby includes advanced capabilities for debugging, visualization and analysis of simulation results in order to aid the designer in the overall exploration, platform design and embedded software development process.

SOC DESIGNER

Carbon's SoC Designer [37] is another tool for platform architecture capture and modeling that dates back to simulation technology originally developed at the University of Aachen in Germany. Initially, this technology was marketed under the product name MaxSim by a spin-off called AXYS. Later on, AXYS got acquired by ARM and MaxSim was renamed to ARM RealView SoC Designer. Eventually, ARM sold the SoC Designer product family to Carbon Design Systems.

Similar to other virtual platform tools, SoC Designer includes a graphical user interface for assembling of system architectures out of pre-defined library or user-made custom components. SoC Designer integrates cycle-accurate hardware, ISS and bus models in a proprietary simulation setup. To avoid the need for expensive context switches necessary in typical event-driven SLDL or HDL simulations, components are statically scheduled into a single-threaded, straight-line C/C++ executable that calls individual blocks cycle-by-cycle in a round-robin fashion. This allows for fast yet fully cycle-accurate system simulation. However, components need to be modeled in a specific cycle-callable fashion. SoC Designer includes a frontend for component model development. In addition, existing SystemC, Matlab and VHDL or Verilog RTL models can either be integrated into or co-simulated with the SoC Designer simulation framework.

VAST AND VIRTUTECH

In contrast to virtual platform approaches based on standardized modeling backplanes and languages such as the SystemC, both VaST [187] and Virtutech

[189] are providers of software-centric virtual prototyping solutions based on proprietary simulation technologies. To achieve faster simulation speeds compared to an interpreted ISS, such approaches are based on binary translation or compiled instruction-set simulation of software code. In all cases, simulations are functionally accurate but techniques can vary in terms of achievable simulation bandwidth and cycle-approximate timing accuracy.

Both VaST and Virtutech include graphical environments (called CoMET and Simics, respectively) to integrate software simulators with models of peripherals and other hardware in order to provide full system simulation. In contrast to event-driven system simulation in typical SLDLs, hardware models are directly integrated into the software execution loop, reducing the need for context switches and further speeding up simulations. However, this requires proprietary models to be developed for each hardware block or peripheral. While both companies provide a large library of standard components and graphical frontends to aid in component model development, recent extensions include support for integration of standard SystemC models in such virtual prototyping environments.

On top of virtual prototypes of the platform hardware created with VaST or Virtutech tools, embedded software can then be developed and validated. Both approaches include corresponding software development environments coupled with extensive debugging capabilities (called METeor and Hindsight).

8.1.3 OUTLOOK

In recent years, ESL design concepts, methods and methodologies have experienced increasing interest and adoption in industry. This trend has been accompanied by a growing number of commercially available tools mainly aimed at modeling, simulation and virtual prototyping of complete system platforms and architectures. As technologies mature, we can expect that more and more of the advanced synthesis and design automation solutions currently under development in academia will be transfered into such commercial settings. On the one hand, as described in the following chapters, tools are already emerging that can provide an automated path to implementation from such system-level virtual platform models. On the other hand, additional research and development efforts will be necessary to provide future tools for automation of the design and design space exploration process at the system level. Only automation of the ESL design flow from specification down to implementation will provide the necessary productivity gains that will enable us in the future to close the gap between continously increasing application complexities and exponentiallly growing technological and device-level capabilities.

8.2 EMBEDDED SOFTWARE DESIGN TOOLS

The close relation between embedded software and the underlying customized hardware platform demands special procedures when developing embedded software, for example in terms of: cross compiling, host/target debugging, and testing. With specialized hardware, the embedded software development also needs to take measures for system booting and hardware specific functionality such as system diagnostics and analysis. By its nature, embedded software design has to deal with hardware-specific tools, such as processor specific instruction set simulators, hardware simulators and emulators, and distributed debuggers. This hardware dependency necessitates the use of special development tools.

To aid the development process, hardware vendors provide development environments geared toward their products. For example, the processor IP vendor ARM, provides RVDS (RealView Development Suite) for developing software for various platforms based on ARM cores. The suite integrates ARM cross compilers, enhanced debug capabilities, ARM specific code optimization options, and libraries for common devices (such as flash devices). Similarly, RTOS vendors offer development support tools. Examples include the Tornado tool suite from WindRiver, and MULTI, the integrated development environment from GreenHills. Such development environments are typically point solutions supporting a fixed system architecture. They are less applicable in a scenario in which the target platform remains flexible until the final stage of system design (e.g. complex multi-processor systems), and which may be composed out of heterogeneous components.

Many programmable logic device vendors also provide an embedded software design tool as a part of their design environment. The SOPC Builder from Altera is an example of this, as is the Embedded Development Kit (EDK) from Xilinx. Both of these tools let system designers define and implement a custom platform out of standard building blocks and user defined hardware components. Once the developer has defined and implemented the platform, these tools synthesize the hardware and produce custom software libraries (e.g. for accessing a programmable interrupt controller) reflecting the target's hardware configuration. By generating customized libraries, embedded software design tools like the SOPC Builder and EDK provide some level of abstraction above the hardware (e.g. resolving addressing and basic device access). However, the designer has to manually develop the embedded software on top of the provided low-level primitives for basic device access. Common to both examples is the focus on the vendor specifics of the target platform in terms of processor and RTOS selection. For example, Altera currently supports the NIOS processor with uC/OS-II, whereas Xilinx supports PPC and Microblaze with Xilkernel.

In this way, these vendor-supplied tools are point solutions, that help developers only in case of matching target platforms.

In addition to development tools, simulation environments are important for development of customized embedded systems as development of a hardware prototype is time consuming and a parallel development of hardware and software is desired. HW/SW co-simulation is one approach that allows for an overlapped development of software and hardware, as the SW can be developed on top of a virtual prototype of the hardware. The nature of a suitable approach for simulation highly depends on the envisioned platform complexity, the desired amount of observable simulation features, the required prototype's equivalence to the final software code, and in the needed simulation speed. For simple single core architectures, using an instruction set simulator or processor emulator may suffice. Similarly, for a system that uses one homogeneous RTOS type and does not feature complicated HW interaction, a minimal model may be sufficient, such as a host-compiled RTOS. In a host-compiled RTOS, a modified version of the target RTOS, together with the developed application, is compiled to run on top of host operating system. However, performance limitations make simple solutions such as these infeasible for complex multi-processor SoCs.

In summary, there are many different tools and methodologies currently available for designers to use in developing embedded software. However, these tools are typically point solutions, specific to a vendor or platform. In addition, current techniques rely on the manual development of software. To achieve higher design productivity, a more global approach is desirable, one that can target a wide range of platforms and has, furthermore, a path to synthesis.

Next, we will outline some academic and commercially available tools for embedded software development and generation.

8.2.1 ACADEMIC TOOLS

ECLIPSE

The open source Eclipse [59], is a multi-language software development platform. It consists of an Integrated Development Environment (IDE) with a flexible plug-in system. The IDE provides a source code editor with a rich set of source annotation and browsing capabilities, integrates a compiler, a source code debugger and many more facilities to aid the software development process. Eclipse's primary focus is the Java language, hoverer with various plug-ins it addresses many other languages as well, such as C/C++, Cobol, Python, Perl, PHP. Eclipse's well defined plug in system makes it very attractive for customized extensions.

With the popularity of Eclipse IDE, many academic and commercial providers use Eclipse as a platform for their own products with a wide range

of specific functionalities. For example, plug-ins exist for UML-based capturing and development (e.g. IBM Telelogic Rhapsody [94]). They extend the IDE with an interface to graphically capture UML-diagrams and later generate structural source code (e.g. class hierarchy) out of the diagrams. Many Eclipse plug-ins more specifically target embedded software development. One example is the Tensilica Xtensa Xplorer IDE [97]. It provides a GUI for customizing an Xtensa processor, integrates a specific cross compilation tool chain and furthermore offers co-simulation and emulation integration. Another Eclipse plug-in example addresses automotive software component design following the AUTOSAR standard, Greensys' Autosar Builder [67]. It supports developing AUTOSAR Software Component (SW-C), ECU and System descriptions at the applications level, integrates their validation and end emulation. Many more plug-ins exist, which we can not enumerate there. The wide range of highly specialized plug-ins make Eclipse an very versatile and powerful software development environment.

POLIS

The POLIS system [11] developed at UC Berkeley is a hardware/software co-design environment with a focus on reactive systems. POLIS allows the user to specify the application in a high level language such as the Esterel or using a graphical as FSM notation. The input specification is internally converted into a co-design finite state machine (CFSM) model. Each FSM within a CFSM represents a component in the system. Using this CFSM, POLIS allows the designer to partition the design, formally verify it, co-simulate as well as synthesize portions of the system. Software generation is performed by transforming the CFSM sub-network chosen for SW implementation into an S-Graph, and subsequent C code generation. In addition an application specific scheduler and drivers are generated for each partitioned design.

DESCARTES

DESCARTES [162] is a software synthesis environment that targets real-time signal processing systems. It focuses specially on optimization techniques for mapping data flow oriented block diagrams onto a DSP. It provides a combination of different mapping and optimization strategies that allow comfortable synthesis of real-time code which is highly adapted to application-specific needs as imposed by constraints on memory consumption, sampling rate, or latency.

DESCARTES uses a data flow description (Asynchronous Data Flow (ADF) and an extended Synchronous Data Flow (SDF)) as an input. The work provides scheduling algorithms defining the order of execution for each computation kernel (node) in the data flow following input constraints of latency, throughput and memory consumption. It generates C code for each computation kernel

that then is compiled using a DSP specific C compiler. With the choice of input model, DESCARTES is tightly coupled to the signal processing domain. In contrast, a flexible generic C-programming model is desirable over these specific input models to cater to the needs of a broader programming audience and to capture a wider range of application domains.

8.2.2 COMMERCIAL TOOLS

MATHWORKS: REAL-TIME WORKSHOP

MathWorks offers a range of packages that are centered around Matlab, a numerical computing environment and programming language. Simulink [132] is a commercial model-based design tool for modeling, simulation and analysis of multi-domain systems. As an input, Simulink has a graphical user interface for assembling a system as a block diagram describing the system functionality. Blocks within Simulink are typically library defined containing standard signal processing (e.g. filters) and control functions. They are connected and hierarchically composed to express the system either as discrete timed or continues timed models. Simulink is tightly integrated into the Matlab environment, and widely used for simulation and design in the control theory and the digital signal processing domain.

On top of Simulink, MathWorks offers Real-Time Workshop [131] for the synthesis of an software implementation. It generates stand-alone C code for algorithms modeled in Simulink. The generated code can be used in many real-time and non-real-time applications, as well as for simulation acceleration and hardware-in-the-loop testing. Real-Time Workshop generates ANSI/ISO C or C++ code from discrete, continuous, or hybrid Simulink models for execution on a wide range of target platforms. It can target bare processors without any operating system, as well as multi-tasking systems with an RTOS.

DSPACE: TARGETLINK

TargetLink [53] is a code generator, by dSpace. It integrates into the Matlab/Simulink environment and is similar to the above discussed Real-Time Workshop. It uses Matlab/Simulink as a graphical editor for system capture. However, it comes with an own library of block components for graphical design composition.

TargetLink provides generation of production code out of a Matlab/Simulink model for a wide range of target processors and platforms. TargetLink mainly addresses the design of automotive systems. It supports targeting OSEK/VDX-compliant operating systems [92] for integration of the generated function code onto an Electronic Control Unit (ECU).

dSpace offers both hardware and software solutions for the automotive design. For validation and testing of applications, it provides three levels of model testing. Model-in-the-Loop (MiL) executes the original model, validating functionality and dimensioning of the algorithm. Software-in-the-Loop (SiL) executes the generated software code on the simulation host, for validation of the implementation. Hardware-in-the-Loop (HiL) executes the final software on an actual ECU. The inputs and outputs of the ECU are controlled by a Matlab/Simulink model simulating the physical control environment.

In summary, dSpace TargetLink, offers a comprehensive solution for the design, synthesis and test of automotive designs with a focus on software. Current development extensions are addressing the emerging AUTOSAR [9] as multi-core ECU platforms.

ESTEREL TECHNOLOGIES: SCADE

Esterel Technologies' commercial SCADE suite [57] is a development environment for system and software engineers targeted for safety-critical applications. With its editor complex systems are captured using a graphical notation for hierarchical composition of data flow and safe state machine notations. SCADE comes with a rich library of predefined blocks for operators, linear functions, digital functions, filters, state machines and model composition. The product is internally based on the synchronous data-flow programming language Lustre [85]. The tool suite is mainly used in the aerospace and defense domains.

SCADE offers a C code generator (KCG) that is certified for the development of airborne systems and equipment, which allow the production use of the generated code. The code generator translates each block of the system specification into a software implementation that can be integrated for execution on a target processor.

For the analysis of generated code, SCADES integrates with external tools for Worst Case Execution Time (WCET) and stack utilization analysis. They provide WCET and stack utilization information at the model level, detailed for each function block within the specification. These analysis capabilities, provide design quality feedback about maintaining timely execution and staying withing resource constraints, which are important for safety critical systems early in the process supporting an efficient design.

In addition, Esterel Technologies offers gateway integration with other modeling environments that allow importing specifications and requirements into SCADES. For example, it provides a gateway for importing of discrete controllers prototyped in Matlab/Simulink. It further integrates with Rhapsody UML/SysML for high-level system requirements. These gateways expand the coverage of SCADES tool suite to other modeling approaches.

UML/SYSML PRODUCTS

The Unified Modeling Language (UML) [147] is an standardized language for the specification of software systems. It is a language for specifying, visualizing, constructing, and documenting the artifacts of a system with an emphasis on the earliest part of a design process. UML is a modeling language, in contrast to a a programming language. It therefore focuses on capturing relevant information required for understanding the design problem, solving it, and guiding implementation of the solution. It excludes any irrelevant information that may hinder that progress.

UML defines 13 different datagram types with a wide range of modeling system structure, system behavior and the interaction of system elements. With this range of diagram styles it is apparent that the designer has great flexibility in capturing system structure. UML provides means to capture boundaries, requirements and system interaction. On the other hand UML by itself is not very suitable to concisely express formulas. For capturing algorithms in the system, UML often relies on embedded C, C++, or Java code as a description.

The Systems Modeling Language (SysML) [149] an extension of a subset UML by using UML's profile mechanism. SysML reduces UML's restriction to software-centric systems, and is positioned as a modeling language for systems engineering applications. It only uses 7 out of the 13 UML diagrams, but extends it by additional diagrams and concepts. For example, it adds requirement diagrams for capturing parametric constraints between structural elements, which aid performance and quantitative analysis. It also introduces additional MoCs by extending the behavior of UML activities for the modeling of continuous and probabilistic systems. The use of UML and SysML for system level design of SoCs is discussed in [116, 126].

Many commercial products for model-based development exist, which are based on UML/SysML. Examples include IBM Telelogic Rhapsody [94], Spark Systems' Enterprise Architect [175], Gentleware's Poseidon for UML [69] and Artisan Software's Artisan Studio [169]. These tools offer graphical editors for capturing UML/SysML diagrams, the analysis and consistency validation. In addition these tools offer generation of targeted code for framework integration. The framework code itself may not contain all algorithm code, however provides a start framework for manual software development.

8.2.3 OUTLOOK

With the increasing attention to embedded software design, the tool support for developing embedded applications has significantly improved in the recent years. Vendors of hardware (e.g. FPGA) and software products (e.g. RTOS) provide an added value to their products by offering integrated development en-

vironments with specialized support for their own product. In addition, many domain specific specialized solutions guide the application development for example in the automotive and signal processing domain. A stronger focus on better structured, reusable, and expandable software implementations is noticeable, for example through utilizing component-based principles such as in AUTOSAR or through tighter connecting documentation and implementation as seen in an UML-based process.

The complexities of future platforms will continue to grow. We will see systems with diverse distributed heterogeneous components as well as systems with many cores. As platform complexities grow, manually implementing embedded software will become infeasible, especially when considering the decreasing time-to-market. Therefore, there is an essential need to further simplify the modeling and development of software and systems. In particular, design environments are needed, which enable abstract development of complex systems at the algorithm level, which automate the implementation process through automatic synthesis of both hardware and software, and which allow the designer to focus on essential functional aspects without the burden of low-level implementation details.

8.3 HARDWARE DESIGN TOOLS

Research and tool development for hardware design-automation began four decades ago, and progressed through four phases. The 1970s embodied the concept phase, which gave birth to basic definitions for the languages, design methods, and tools necessary for standard and custom processors. The 1980s introduced the algorithm phase, which saw a flurry of research activities defining algorithms for allocation, binding, and scheduling in a new field called High-Level Synthesis (HLS). During the decade which followed, these new approaches were consolidated with the emergence of several seminal books on HLS and the first commercial tools. Finally, the first decade of this century ushered in the acceptance phase, during which the concept of automatically generating custom hardware components from high-level programming languages (C-to-RTL) has become accepted and applied to many custom designs by industrial designers world-wide.

The concept phase began with Bell and Newell's seminal book on computer structures [16], which introduced Instruction-Set Processor (ISP) notation. ISP was intended to precisely and unambiguously describe the behavior of instruction-set processors. This behavior was characterized by the existence of an interpretation algorithm that fetches, decodes, and executes "instructions" stored in the memory. The ISP concept was refined by Barbacci at CMU who introduced the Instruction-Set Processor Specification (ISPS) for the simula-

tion, evaluation, and synthesis of simple processors [14, 15, 13]. Barbacci, along with Siewiorek, also developed an initial system for the synthesis of processors called CMU RT-CAD System in 1976 [168]. That opened broader investigations into the different aspects of synthesis process such as internal representations [133], component allocation [84] and processor architecture selection [179]. At this same time, Zimmermann and Marwedel at Kiel developed the MIMOLA design method to design of digital processors from a very high-level behavioral specification [199, 127]. A key feature of this method is the synthesis from application programs expected to run on that processor. This was the first attempt at C-to-RTL compilation.

In the 1980s, research on algorithms for HLS spread to many different countries. This research was focused on languages and representations, algorithms and methodologies, and tools and environments. In terms of languages every research group used a different subset since standard languages such as C or VHDL were not synthesizable [122]. In the representation domain CDFG became popular at this time [151]. Allocation, binding and scheduling algorithms were the most popular topic for research [155, 128, 10, 34, 49, 156, 153]. This was a time of great diffusion of new ideas. Different methodologies for the design of controllers, datapaths or complete custom processors were introduced based on different design paradigms [26, 50, 152, 176, 177, 181, 153, 134]. Similarly, many HLS tools came into use, the most prominent being the Yorktown Silicon Compiler from IBM [25] which included high-level, logic and layout synthesis, CATHEDRAL from IMEC in Belgium [160], which focused on multiprocessor DSP applications, as well as The System Architect's Workbench from CMU [176], and Design Environment from U of Karlsruhe [36].

The consolidation phase of HLS in 1990s is characterized by the appearance of several books defining the seminal work in the field. Don Thomas and associates published a book on CMU's System Architect's Workbench in 1990 [178], followed by several other books by different authors on different aspects of HLS, including timing constraints [114], methodologies and algorithms [61], digital signal processing [186], synthesis and optimization [139], and component reuse [102]. Several edited books concerning the issues involved in HLS [35, 138] were also published in that period. 1990s were also characterized by the appearance of EDA companies offering commercial tools. Wakabayashi introduced NECs Cyber synthesis tool [191], Synopsys introduced Behavioral Compiler (BC) [110], and Mentor introduced Monet [56]. Those early tools followed basic principles of HLS as described in the above mentioned HLS books. For example, BC accepted a behavioral description in a subset of VHDL or Verilog. It converted the input description into a CDFG representation that exposed control and data dependences. In order to perform technology-specific scheduling BC converted data flow graph in each basic block of CDFG into gates in order to produce accurate delay estimates. This

way BC could schedule two operations into the same clock cycle as long as their joint delay was smaller than the clock cycle. After scheduling, BC performed allocation and binding and synthesized the control FSM with gates. The last step was logic optimization of the generated datapath and controller.

The early tools showed the possibility of HLS automation. However, there were several obstacles for commercial success. Designers had to use a tool-dependent subset of HDLs instead of a standard programming language such as C or Java. Datapath and controller architectures were overly simple without pipelining or data forwarding. The controller was implemented as an FSM with gates, so that later upgrade or changes needed re-synthesis. Since the controller did not use control or program memory, it was not possible to execute large programs. Even when the synthesized result was acceptable, interfacing the synthesized component into a larger system was not well defined.

The largest obstacle to widespread acceptance of HLS was the market's unpreparedness for processor-level abstraction. This has changed dramatically in this decade because of increased system complexities. The new HLS tools use a standard programming language as the input and generate RTL in a HDL as the output so synthesized designs can be prototyped with FPGA tools. Moreover, the quality of these tools has improved through the use of more sophisticated algorithms. At the same time the complexity of synthesized components increased from special functions with a FSM controller to custom processors with a programmable controller.

8.3.1 ACADEMIC TOOLS

GAUT

The GAUT tool from UBS [157] is an academic and open-source HLS tool dedicated to digital signal processing applications. It generates an independent custom processor with custom interface that allows it to be inserted into any system. Starting from an algorithmic bit-accurate specification written in C/C++, GAUT extracts the potential parallelism before performing the allocation, scheduling and binding tasks. The mandatory synthesis constraints are the throughput, the clock period, and the target technology while the optional design constraints are I/O timing diagram and the variables-to-memory mappings. GAUT synthesizes a potentially pipelined architecture composed of a processing unit, a memory unit, a communication interface unit that uses a globally-asynchronous, locally-synchronous protocol.

GAUT generates an IEEE P1076 compliant RTL level VHDL file. This VHDL file is an input for commercial, off the shelf, logical synthesis tools such as ISE/Foundation from Xilinx, Quartus from Altera, or Design Compiler from

Synopsys. GAUT also generates a SystemC cycle-accurate simulation model for simulation-based validation.

NO-INSTRUCTION-SET COMPUTER

The No-Instruction-Set Computer (NISC) from UCI [40] is an attempt to overcome two of the weaknesses of HLS: programmability and metric closure. Most HLS designs are special function components with a fixed controller that implements the FSM of the special function executed in the datapath. Such a controller is usually implemented with gates which limit the FSM size to a couple of hundred states. The first problem with such an implementation is that the complete design has to be re-synthesized for any change or upgrade in the given function. The other problem is that this type of implementation can not support large amounts of code. To solve this problem NISC uses a programmable controller with a control-word memory that stores control words for every clock cycle. This way large codes can be accommodated and even dynamically up-loaded.

The second HLS weakness is that during synthesis and optimization the required metrics must be estimated. The exact value of delay, power, and performance is not known until the final layout. The finalized metrics values or metric closure is needed to fine tune the architecture and the application code. NISC solves this problem by separating the allocation and datapath structure generation from scheduling and binding performed by the NISC compiler. Therefore, making it possible to a create complete structure with all the metrics known before compilation. If the final results are not acceptable, the datapath can be modified and the application code recompiled. Furthermore, NISC methodology leads to the concept of standard architecture-cells or templates that can be stored in the library and used by different application designers. Having several such templates per application domain greatly simplifies the methodology and tools on lower levels of abstraction.

A NISC tool set as shown in Figure 8.6 consists of three different components: a datapath generator, a NISC Compiler, and an RTL Generator. The datapath generator is used to create a datapath structure for a given application. This task can be done automatically by profiling the application code in C, compiling usage statistics, selecting components and connectivity for the given performance metrics and generating a Generic Netlist Representation (GNR) of the datapath. A datapath template can be also selected from the template library, or designers can specify their own datapath by creating a GNR through GUI. The NISC cycle-accurate compiler [161] compiles the application for a given datapath. It converts the application code into a control-words stream controlling datapath on each clock cycle. The RTL Generator produces the RTL description for inputing to FPGA or ASIC tools. It converts the datapath

and controller GNR into RTL with control words generated by the compiler loaded into the control-word memory in the controller.

If synthesized results are not satisfactory, the datapath structure and/or application code can be modified. This can be done manually by rewriting the application code and GNR or automatically through code refinement or datapath refinement tools.

A NISC enables the designers to control every aspect of the design. The designer can select the exact points for improvement and then make the changes quickly. For example, by changing the GNR description of the datapath architecture, the designer can reduce a critical path delay or fix complex multiplexers and connections that consume too much power or make the layout unroutable. Since datapath can be an input in NISC technology, the designer can selectively explore options for quality metrics. For example, a designer can focus on dynamic power minimization by modifying the connections or gating or latching them in the datapath description and quickly see the effect on the final results.

SPARK HIGH LEVEL SYNTHESIS

SPARK tool from UCSD [140] is a C-to-VHDL high-level synthesis framework that employs a set of innovative compiler, and synthesis transformations to improve the quality of high-level synthesis results. The SPARK parallelizing high-level synthesis methodology is targeted particularly to multimedia and image processing applications along with control-intensive microprocessor functional blocks.

As shown in Figure 8.7, SPARK takes a behavioral description in ANSI-C as input. It also takes additional information as input, such as a hardware

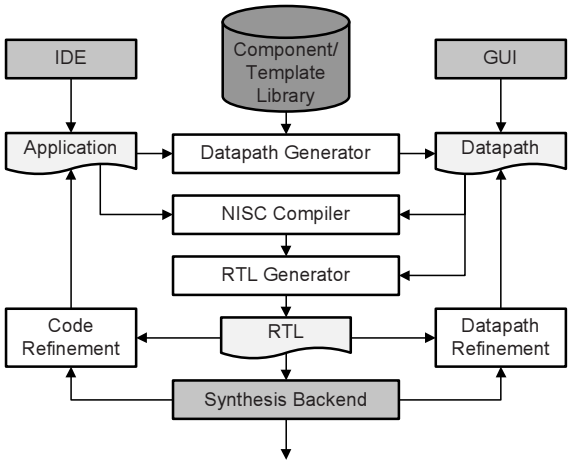

FIGURE 8.6 NISC technology tools

resource library, resource and timing constraints and user directives for the various heuristics and transformations. SPARK stores the input behavior in a hierarchical intermediate representation, a CDFG derivative with dependences across basic blocs. This is critical for enabling coarse-level transformations and making global decisions about code motion.

SPARK first applies a set of coarse-grain and fine-grain code transformations to the input description during a pre-synthesis phase before performing the traditional high-level synthesis tasks of scheduling, allocation and binding. The transformations in the pre-synthesis phase include (a) coarse-level code restructuring by function inlining and loop transformations, (b) transformations that remove unnecessary and redundant operations such as common sub-expression elimination (CSE), copy propagation, and dead code elimination (c) transformations such as loop-invariant code motion, induction variable analysis (IVA) and operation strength reduction, which reduce the number of operations within loops and replace expensive operations with simpler operations.

The pre-synthesis phase is followed by the scheduling and allocation phase. Resource allocation and module selection are done by the designer and are given as input to the synthesis tool through a hardware resource library. The scheduler is organized into two parts: the heuristics that perform scheduling

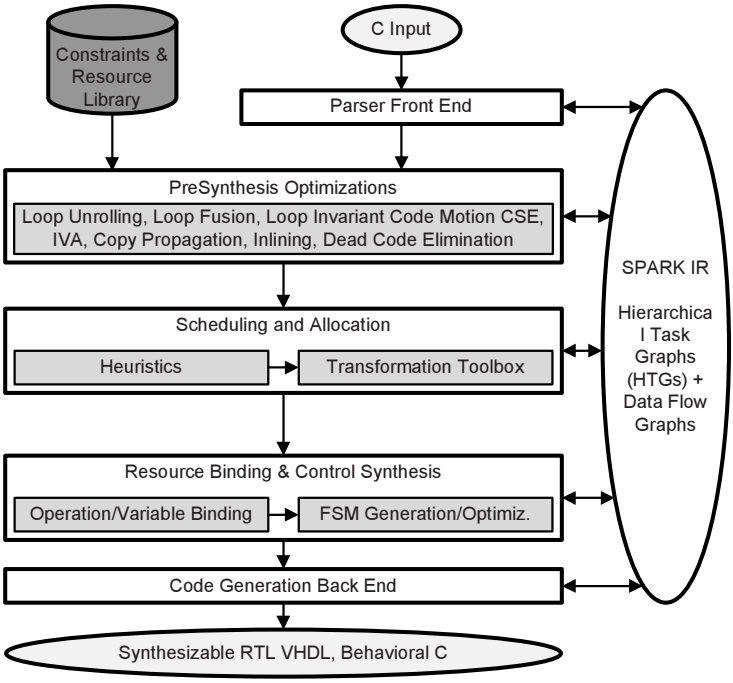

FIGURE 8.7 The SPARK Synthesis Methodology

and a transformations toolbox. The transformations toolbox contains speculative code motion transformations, the percolation and trailblazing code motion techniques, dynamic renaming of variables et cetera. The synthesis transformations include chaining operations across conditional blocks, scheduling on multi-cycle operations, and resource sharing.

Besides the traditional high-level synthesis transformations, the scheduling phase also employs several compiler transformations applied "dynamically" during scheduling. These dynamic transformations, such as dynamic CSE and dynamic copy propagation, exploit the new opportunities created by code motions. A branch balancing technique also dynamically adds scheduling steps in conditional branches to enable code motions, specifically those code motions that duplicate operations in conditional branches.

Passes from the toolbox are called by a set of heuristics that guide how the code refinement takes place. The heuristics and the underlying transformations that they use are kept completely independent from each other. This allows the heuristics to employ the various transformations as and when required, thus enabling a modular approach that allows the easy development of new heuristics.

The scheduling phase is followed by a resource binding and control synthesis phase. This phase binds operations to functional units, ties the functional units together, allocates and binds registers, generates the steering logic and generates the control circuits to implement the schedule. The focus of resource binding approach is to minimize the interconnect between functional units and registers. After binding, SPARK generate a FSM controller for the scheduled and bound design.

Finally, a back-end code generation pass generates a synthesizable RTL VHDL. SPARK also has back-end code generation passes that generate ANSI-C and behavioral VHDL. These behavioral output codes represent the scheduled and optimized design. The output C code can be used in conjunction with the input C code to perform functional verification and also, to improve visualization for the designer on the effects of the transformations applied by SPARK on the design.

XPILOT SYNTHESIS SYSTEM

The xPilot is a behavioral synthesis system being developed at UCLA [183, 41]. The goal of xPilot is to provide novel platform-based behavior synthesis technologies to optimize logic, interconnects, performance, and power simultaneously, so that designers can improve both design productivity and quality of results.

The overall design flow of the xPilot system is shown in Figure 8.8. xPilot accepts synthesizable C or SystemC as input. The behavioral description is first parsed and optimized by the UIUC LLVM compiler infrastructure. A System-

level Synthesis Data Model (SSDM) is then constructed from the LLVM's internal representation. The basic building blocks in SSDM are processes and channels. A process describes the behavior of one module, and each process uses a CDFG to capture its behavior. Each process interacts with other processes through ports and channels.

Each channel implements some interface to implement certain communication protocols. In total, an SSDM defines a process network to model the concurrent behavior of a complex system. On top of SSDM, xPilot performs platform-based synthesis and physical-aware optimizations during scheduling and resource binding; these construct an optimized State Transition Diagram (STG) and an associated datapath model. At the back end, xPilot generates RTL implementations together with constraint files such as multi-cycle path constraints and physical location constraints, to leverage the existing logic synthesis and physical design toolset.

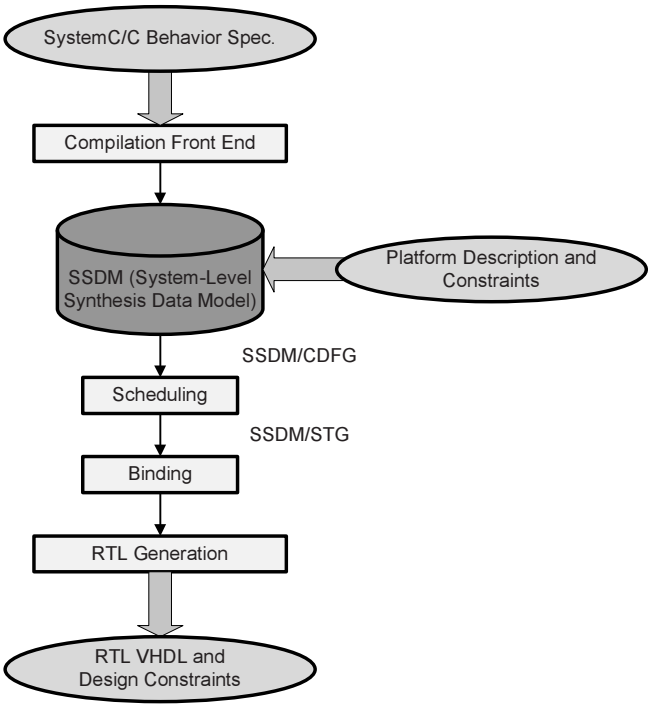

FIGURE 8.8 xPilot Synthesis System

8.3.2 COMMERCIAL TOOLS

CATAPULT SYNTHESIS

Catapult from Mentor [137] takes a behavioral description written in ANSI C++ and a set of user directives as input and generates an RTL that is optimized for the specified target technology. The input specification is behavioral and does not include any notion of explicit parallelism, time, state or interface protocol or the design structure. Required directives specify the selected component library and the clock period. Optional directives control hardware details such as interface and memory mappings, how much parallelism to implement in loop unrolling and loop pipelining, hardware hierarchy and block communication, latency or cycle constraints for scheduling, the number and/or type of hardware resources for allocation , etc. Catapult supports native C++ integer types as well as C++ bit accurate integer and fixed-point datatypes are supported for synthesis. The generated RTL faithfully reflects the bit-accurate behavior specified in the source.

One of the advantages of keeping the input untimed is that a very wide range of interfaces and design structures can be generated without changing the input specification. Another advantage is that errors that are created through manual coding are avoided. The interface and the design structure of the generated design are all under the control of the user via synthesis directives. Interface synthesis maps the data transfer that is implied by passing of function arguments to a variety of hardware interfaces such as wires, registers, handshaked registers, memories, buses or more complex user-defined interfaces. All the necessary signals and timing constraints are generated during the synthesis process so that the generated RTL conforms and is optimized for the desired interfaces.

Hierarchy or block-level concurrency can be also specified by user directives with Catapult. For example, a C function can be synthesized as a separate hardware block instead of being inlined in its caller(s). The blocks are connected with the appropriate communication channels and the required handshaking interfaces are generated to guarantee the correct execution of the specified behavior. The blocks may be synthesized to be driven by different clocks. The clock domain crossing logic is generated by Catapult. Communication is optimized depending on user directives to enable maximal block-level concurrency using FIFOs and ping-pong memories to enable block-level pipelining and thus improved throughput.

All the HLS synthesis steps are aware of accurate component area and timing numbers for the target technology (ASIC or FPGA) for the RTL synthesis tool of choice. Accurate timing and area numbers for components are essential for generating an RTL that meets the timing and area constraints. During synthesis, Catapult queries the component library so that it can allocate a variety of combinational or pipelining components with different performance and area

tradeoffs. The queried component library is pre-characterized for the target technology and the target RTL synthesis tool. Component libraries can also be built by the user to incorporate specific characterization for memories, buses, I/O interfaces or other pieces of functionality such as pipelined components..

The synthesis process generates the required verification infrastructure in SystemC so that the input stimuli from the original C++ testbench may be applied to the generated RTL to verify its functionality against the original input specification using simulation. The synthesis process also generates the required verification infrastructure for sequential equivalence checking between the input specification and the generated RTL. Catapult has been successfully used in over 200 ASIC tapeouts and several hundred FPGA designs. Typical applications include computation-intensive algorithms in communications and video and image processing.

CYNTHESIZER

Cynthesizer from Forte [58] takes a SystemC module containing hierarchy, multiple processes, interface protocol and algorithm and produces RTL Verilog optimized to a specific target technology and clock speed. The target technology is specified by a user provided library file or, for FPGA implementation, by identifying the targeted Xilinx or Altera tools.

The input to the high-level synthesis flow used with Cynthesizer is a pin- and protocol-accurate SystemC model. The designer puts untimed high-level C++ in a hardware context using SystemC to represent the hardware elements such as ports, clock edges, structural hierarchy, bit-accurate data types and concurrent processes.

Clocked thread processes are used for the majority of the module functionality. They contain an infinite loop that implements the bulk of the functionality along with the reset code that initializes I/O ports and variables. Within a thread, the designer can combine untimed computation with cycle-accurate communication. A hybrid scheduling approach is used in which the protocol sections are scheduled in a cycle-accurate way, honoring the clock edges specified by the designer as SystemC wait statements. The computation code is written without any wait statements and scheduled by the tool to satisfy latency, pipelining and other constraints given by the designer. Triggered methods can also be used to implement behaviors that are triggered by activity on signals in a sensitivity list, similar to a Verilog 'always' block. This allows a mix of high-level and low-level coding styles to be used if needed.

Complex subsystems are built and verified by combining modules using structural hierarchy just as it would be done in Verilog or VHDL. The high-level models used as the input to synthesis can be simulated directly to validate both the algorithms and the way the algorithm code interacts with the interface protocol code. Multiple modules are simulated together to validate that

they interoperate correctly to implement the functionality of the hierarchical subsystem.

In order to ensure that the synthesized RTL meets timing at a given clock rate using a specific foundry and process technology, a high-level synthesis tool requires accurate estimates of the timing characteristics each operation. Cynthesizer uses an internal datapath optimization engine to create a library of gate-level adders, multipliers, etc. The timing and area characteristics of these components are used by Cynthesizer to make tradeoffs and optimize the RTL. Designers have the option of using the gates for implementation or of giving their logic synthesis tool RTL representations of the datapath components.

Cynthesizer produces RTL Verilog for use with logic synthesis tools. The RTL consists of a finite state machine and a set of explicitly instantiated datapath components such as multipliers, adders, and multiplexors. More complex custom datapath components that implement arithmetic expressions used in the design are automatically created, and the user can specify sections of C++ code to be implemented as datapath components. The multiplexors directing the dataflow through the datapath components and registers are controlled by a conventional finite state machine implementation.

SystemC is a good fit for high-level synthesis because it combines the high-level and object-oriented features of C++ with hardware constructs that allow a designer to directly represent structural hierarchy, signals, ports, clock edges etc. This provides a very efficient design and verification flow in which behavioral models of multiple modules can be concurrently simulated to verify their combined algorithm and interface behavior. Most functional errors can be found and eliminated at this high-speed behavioral level instead of through time-consuming RTL simulation. Once the behavior is functionally correct, the models that were simulated are used directly for synthesis, eliminating opportunities for mistakes or misunderstanding.

PICO

PICO tools developed by Synofra [45] support the development of custom processors or application engines for a system platform consisting of standard CPUs and DSPs, memories, IF components such as DMAs or USBs and complex application engines such as video codecs and wireless modems. PICO provides a fully automated, performance-driven, application engine synthesis methodology that enables true algorithmic-level input specification. It produces C-to-RTL mapping under performance constraints in terms of throughput and cycle-time. The key to PICO's approach is the use of an advanced parallelizing compiler in conjunction with an optimized, compile-time configurable architecture template to generate an application-engine RTL.

PICO uses C/C++ language as the preferred mode of input specification at the algorithmic level to allow the user to specify functionality as a sequential pro-

gram. PICO's parallelizing compiler automatically extracts parallelism from the input specification to meet the desired performance based on its analysis of program dependencies and external resource constraints. PICO is intended for applications that process data streams such as audio, video, imaging, security, wireless, networking applications, among others. There is large amount of parallelism in such applications at various levels of granularity. These applications consist of a sequence of transformations expressed as multiple loop-nests encapsulated in a C procedure that is executed repetitively on a stream of data blocks.

One invocation of this procedure is called a task. PICO optimizes parallelism on task-level, loop-level, iteration-level, and instruction level at the same time to satisfy performance and cost constraints. Given the parallelism available in the application code at various levels, the PICO compiler exploits this parallelism without violating the sequential semantics of the application code by following the well-defined model of Kahn process networks, in which a set of sequential processes communicate via streams through unbounded FIFOs. This Kahn process network concept is implemented in PICO with an architectural template defined by a Pipeline of Processing Arrays (PPA). Each of the top level loop-nests in the C procedure is mapped to a custom processor called Processing Array (PA) which communicates with other PAs via one or more FIFOs or memories. Each PA is structured like a wide Very Long Instruction Word (VLIW) processor that is customized to execute only one program: a loop iteration.

Along with the hardware RTL and its related software, PICO also produces SystemC-based TLM models of the hardware at two levels of abstraction: an untimed programmer's view and a timed programmer's view. Knowledge of the target technology and its design trade-offs is embedded as a part of a macrocell library which PICO tools use as a database of hardware building blocks. The library consists of pre-verified, parameterized, synthesizable RTL components such as register, adders, multipliers and interconnect elements. These macrocells are independently characterized for various target technologies and various macrocell parameters. PICO uses these characterization data for its internal delay and area estimates.

CYBERWORKBENCH

CyberWorkBench (CWB) from NEC is a C-based high-level synthesis and verification tool that has been in development since 1990s [190, 191, 144]. The main idea behind the CWB is an "all-in-C" approach in which all the modules in the design are described in the behavioral C language. CWB also supports legacy RTL blocks as black boxes, which are called as C functions. At the same time the synthesis, verification, and debugging tasks are all done in the C source code.

CWB targets general SoC platforms which normally contain several CPUs or DSPs, in addition to custom HW modules and some pre-designed or fixed RTL or gate level IP modules that are connected directly or through buses in the platform.

Initially, each custom HW module is described in a specialized behavioral C called Cyber-C. Once its functionality is verified through the C simulator and debugger, the HW module is synthesized with the behavioral synthesizer. The custom processors are also synthesized from their C description in the CWB environment. Legacy RTL blocks are described as functions and handled as black boxes. The CPU bus and other bus interface circuits are also automatically generated using a CPU bus library. After synthesis and verification of each module, the CWB environment allows designers to create a cycle-accurate simulation model for the entire platform including CPUs, DSPs and custom HW modules. With this model designers can verify both the functionality and the performance of their design, as well as the embedded software running on the CPU, DSP and custom processors. The behavioral synthesis is fast enough to allow designers to modify and synthesize HW modules and embedded software many times. The input C code can also be debugged with a formal verification tool that checks properties and model assertions. These global properties and in-context assertions are described in the original input C code. The equivalence between the behavioral C and the generated RTL can be verified dynamically and statically.

Currently, the platform-level parallelization is left to the system designers. They partition the input C code into individual HW modules and embedded software based on the performance results of the cycle simulation or FPGA prototyping.

BLUESPEC

Bluespec tools from Bluespec provide an alternative to the standard C-based HLS technology by focusing on components that do not fall into the loop-and-array paradigm: processors, caches, interconnects, bridges, DMAs, I/O peripherals, and similarly others. These components are characterized by heterogeneous, irregular and complex parallelism for which the sequential computational model of C is not expressive enough. They use a language in which the concurrent behavior of a system is expressed as a collection of rewrite rules. Each rule has a guard expressed by a Boolean predicate on the current state, and an action that transforms the state of the system. These rules can be applied in parallel, that is, any rule whose guard is true can be applied at any time. The only assumption is that each rule is an atomic transaction, meaning that each rule observes and delivers a consistent state, relative to all other rules. The rules and their ordering are described in Bluespec System Verilog (BSV).

BSV allows designers to specify the micro architecture precisely, but with powerful generative and parameterization mechanisms which allow a single source to flexibly represent a family of micro architectures, within which different choices may be appropriate for different metric optimizations. Thus BSV provides synthesis from very high level description with a precisely-specified micro architecture in the parameterized program structure. Bluespec Compiler compiles a BSV description into Verilog RTL or SystemC while Bluspec Simulator simulates Bluespec designs with cycle accuracy.

8.3.3 OUTLOOK

The last thirty years of research and development into high-level synthesis has proven profitable, as evinced by the increasing supply of HLS tools and by designers' acceptance of C-to-RTL concepts. Though there has been much progress in the concepts, algorithms, and methods for HLS, there is more work ahead, which is driving the surge in HLS research and tools [45].

Although some tool suppliers are offering specific languages that support efficient descriptions of functionality or architecture, most of the market is settling on C/C++ for describing input functionality. That decision is leading to increasing efforts in pre-synthesis compilation to increase possible concurrency for future optimization and to improve the quality of the synthesized design.

The synthesized architecture is usually the set of storage and functional-unit components connected through multiplexers. Still, much work must be done to improve the architecture by adding busses, control and datapath pipelining, and programmable controllers in order to move the architecture into direction of custom processors. Some suppliers offer specific pipelined-blocks architecture for "loop-and-array" applications, but there is no conclusion on standard architecture-cells or templates that will make C-to-RTL compilation more efficient, as in the compilation of C to instruction-sets.

Moreover, the problem of interfacing synthesized components and merging them into a system platform is still grossly under solved. As with component architecture, there is a need for standard interface-cells so that any two synthesized components can be easily connected. With availability of architecture and interface standard cells and an efficient compilation from C, the directions of the IP industry in the future still remain to be answered.

8.4 CASE STUDY

So far we have looked at a variety of system level, software and hardware design tools. Many tools are available publicly or commercially to assist with

different aspects of embedded system design. We advocate that there will be a need for new tool-sets or design environments that integrate different aspects of embedded system design. These developments will be crucial to the evolution of a model based design and verification methodology for embedded systems. In the long term, there will be no distinction between hardware and software at the design entry stage. The next generation of embedded system design tools will focus on applications and enable non-experts to design embedded systems.

In this section, we will present a case study for the design of an industrial size application, the MP3 decoder. We use the Embedded System Environment (ESE) tool set [39] to present the model based design of the MP3 decoder on four heterogeneous embedded platforms. The ESE tool flow embodies the design methods and principles that have been discussed in this book. We will present results that demonstrate the speed and accuracy of automatically generated models, the quality of the synthesized design and the productivity gains that results from using ESE. The case study is meant to motivate designers to adopt the embedded system design methods and principles presented in this book.

8.4.1 EMBEDDED SYSTEM ENVIRONMENT

ESE consists of two parts, the front end and the back end, as shown in Figure 8.9. The input to front end is the system specification consisting of an application model mapped to a given platform. It automatically generates a TLM of the system for fast and early design evaluation. The back end reads this TLM and synthesizes the required software and hardware to produce the

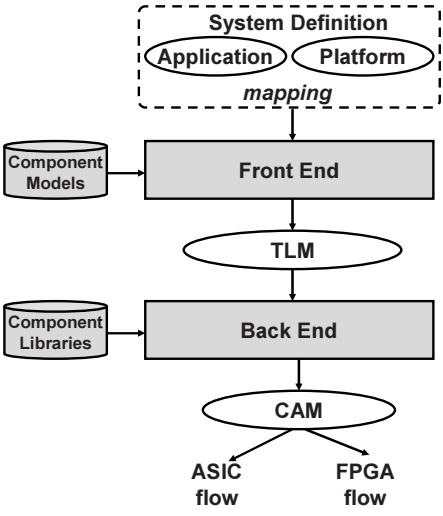

FIGURE 8.9 ESE tool flow

cycle accurate model (CAM). The CAM is the hand-off point to standard FPGA and ASIC design automation tools. Therefore, ESE enables a structured and automated design flow from an abstract specification to an implementation, based on well defined design decisions.

The application, platform and mapping entry in ESE are simplified by an intuitive Graphical User Interface (GUI). The application is captured as a set of concurrent communicating processes. Each process has an associated C/C++ description. Channels are used to specify communication between processes. These channels provide a rich set of user level communication mechanisms, such as handshake, FIFO and asynchronous read/write.

The hardware platform is composed in the GUI from a set of processing elements (PEs), buses, and interface components called transducers. The software platform is defined by configuring the software parameters of the processing elements. These configurations include the RTOS definition, task scheduling policy and memory management. A mapping from application to platform may also be defined graphically in ESE. The C/C++ processes are mapped to PEs. Channels are mapped to buses or routes in the hardware platform.

ESE FRONT END

The goal of ESE front end is to enable fast and early design space exploration by automatically generating fast and accurate TLMs from the system specification. The details of the TLM generation process are shown in Figure 8.10. The basic idea is to automatically generate a high speed TLM that can be simulated to obtain metrics about the design; these metrics may be performance, power, reliability, security and so on. Once the metrics are obtained, the designer may

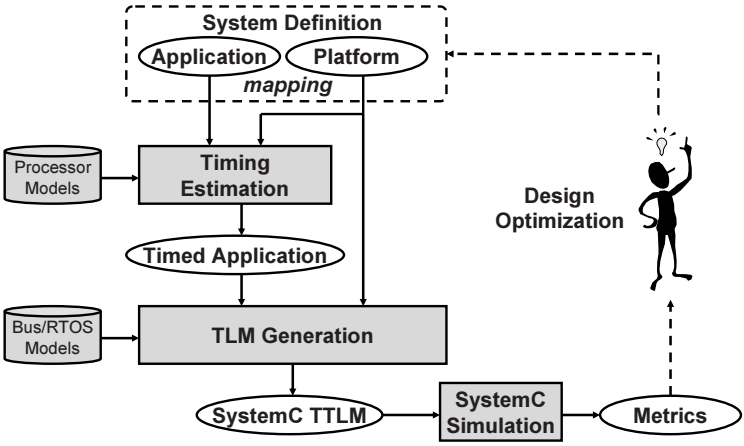

FIGURE 8.10 System level design with ESE front end

either be satisfied with them or go back to change either the application model, the platform or the mapping decisions. A practical design space exploration flow requires the capability to generate TLMs quickly. Therefore, manually coding the TLMs is not an option. TLMs must also provide reliable metrics. Perfect accuracy is desirable, but marginal error may be tolerated for a higher simulation speed.

The metric estimation supported by ESE generated TLMs is timing. Timing is annotated inside the TLM such that TLM simulation can predict timing for any input. ESE uses a retarget-able technique to automatically annotate cycle-approximate timing to the TLM. Data models of the PEs, buses and RTOS are used for timing annotation. The PE data model includes the data path and the memory hierarchy information of the PE. Therefore, it includes the number and type of architectural components and the size and configuration of caches. The bus model defines the bus transaction delays for various bus modes such as word, burst or pipelined transfer. The RTOS model includes methods for dynamic scheduling of processes and inter-process communication inside the PE.

The TLM generation occurs in two steps. The first step is the computation timing estimation where the application process code is instrumented with delays. The process code is converted into a Control Data Flow Graph (CDFG) representation. Each CDFG node represents a basic block in the application process. Based on the mapping of the process to a given PE, each basic block is statically scheduled on the PE data path. The scheduling provides the number of cycles needed to execute the basic block. The memory model of the PE is used to estimate the overhead of data and instruction cache misses. The scheduling and memory overhead delays are added to predict the delay for the basic block. This prediction is done for all the basic blocks in all the processes of the application model.

The processes, annotated with computation timing, are instantiated inside PE models. The executable models of the buses, transducers and RTOSes are instantiated and linked with the PE models. The RTOS model is used to capture resource contention and dynamic scheduling of processes mapped to the same PE. The abstract channel communication between the processes is transformed into sequence of bus transactions, based on the mapping of channels to buses and routes. The final result of the above steps is the timed TLM (TTLM) written in SystemC, which is the *de facto* language for system level modeling. The SystemC TLLM can be compiled natively on the host machine and simulated to obtain timing metrics. These metrics can then be used for design optimization as explained earlier.

ESE BACK END

After the design optimization steps are completed, a satisfactory designed is obtained at the system level. However, this design is still in the form of a TLM, which is suitable for simulation but for ready for implementation with standard EDA tools. The TLM must be transformed into the aforementioned CAM for hand-off to ASIC and FPGA implementation tools. The synthesis of the CAM from TLM is supported by the ESE back end as shown in Figure 8.11.

There are three modules in the back end, each working on different parts of the TLM. The software synthesis module produces the PE specific C/C++ code for software implementation. Naturally, the PEs in consideration for SW synthesis are embedded processors such as CPUs or DSPs. The application code is imported as is from the TLM. If an RTOS is present, the RTOS model is replaced with the actual RTOS kernel library from the database. Finally, the communication layers are generated. The communication layers implement the abstract channel based communication in the TLM using processor specific code. The synchronization with external processes is implemented using interrupt or polling. If interrupts are used, the specific interrupt handlers are generated and instantiated for each channel. If a polling option is chosen, then the HW polling flag management code is generated. The abstract data transfer of the TLM is implemented by creating an address map for the transactions and generating specific load and store transactions. Once all the code is generated, the cross-compiler for the embedded processor is used to generate the software binary.

For hardware implementation, the RTL code for the specific hardware PE must be generated. If a RTL model of the PE is already available in the IP

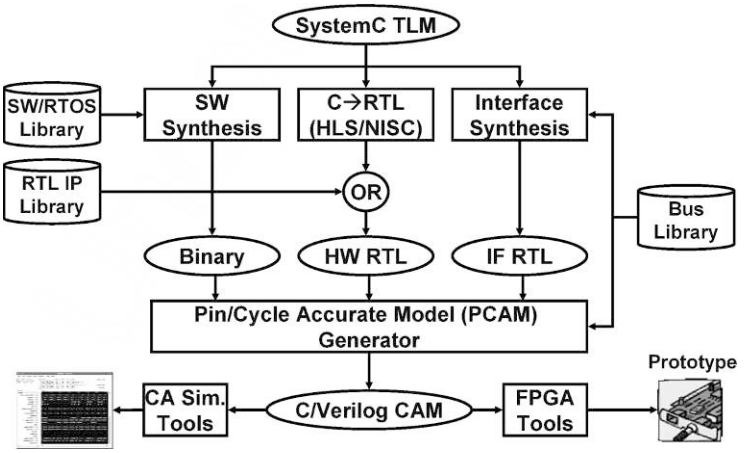

FIGURE 8.11 SW-HW synthesis with ESE back end

database, the C model in the TLM is simply replaced with this RTL model. If a RTL implementation is not available, it must be synthesized from the C model in the TLM. For this purpose, an industrial high level synthesis (HLS) tool may be used. ESE also supports generation of PE RTL model using the No Instruction Set Computer (NISC) technology [40]. The NISC technology is based on the programmable controller design of hardware PEs as explained in Section 6.1. A suitable data path template is selected based on the C profile of the process mapped to the hardware PE. Then, the NISC compiler is used to translate the C code of the process into control words to drive the data path. A Verilog RTL description of the data path and the control memory is automatically generated from the NISC tools for hardware implementation.

The final step in CAM generation is the RTL generation of the communication structure of the system. The bus protocol library is used to instantiate the bus controllers for all the buses in the system. The RTL description of all the transducers is also generated automatically based on the mapping of channels to routes in the platform. Interrupt controllers are also instantiated and configured, if needed.

The CAM produced by the ESE back end consists of C or binary code for all the software PEs in the system and RTL Verilog code for all the hardware PEs, buses and transducers. The CAM may be simulated using standard Verilog simulators available commercially. Since the Verilog code is synthesize-able, it can be input to logic synthesis tool for ASIC implementation. Alternately, ESE produces FPGA-ready description of the CAM for prototyping on FPGAs. Therefore, ESE enables a well defined and automated path from system specification to a software/hardware implementation.

8.4.2 DESIGN DRIVER: MP3 DECODER

As explained earlier, ESE provides model automation, estimation and software/hardware synthesis from abstract system representation. The tool support

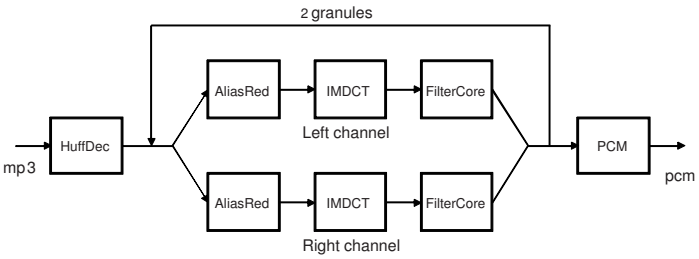

FIGURE 8.12 MP3 decoder application model

in ESE facilitates design of complex embedded systems for large applications. In order to demonstrate the efficacy of ESE, we have chosen the MP3 decoder application as a design driver. The MP3 decoder is an ideal application in many ways. It is reasonably complex, with over 13000 lines of C code, to justify a system level design approach. It is modular with well defined functions to demonstrate partitioning and hardware-software implementation. Since it typically has streaming input and output, there are real time constraints that require an application specific implementation. Finally, MP3 decoder designs are pervasive and highly relevant to mobile multimedia devices.

MP3 APPLICATION

The functional block diagram of the MP3 decoder [182] is shown in Figure 8.12. The input to the decoder is an MP3 data stream consisting of frames. Each frame of MP3 data is decoded using huffman decoding function (*HuffDec*). The frame is then split into *granules* that are sent to two channels, *left channel* and *right channel*, for stereo decoding. The two channels are data independent, so they can work on completely independent sections of the granules. Each granule section undergoes a sequence of transforms, namely alias reduction (*AliasRed*), inverse modified discrete cosine transform (*IMDCT*), and discrete cosine transform (*DCT*). Finally, the decoded granules are combined into pulse code modulated (*PCM*) frames that are ready to be sent to speaker.

In order to play the streaming MP3 file without dropped frames, the decoding rate must be at least 36 frames per second. As a result, after compensating for I/O delays, each frame must be decoded within 26.12 milliseconds (ms). Therefore, we have a real time constraint on the execution time of the decoder application. If a pure software implementation meets the required constraint, it would be an ideal implementation. Otherwise, a multi-core implementation, may be required. The decoding can be sped up by adding specialized hardware PEs for the compute intensive IMDCT and DCT functions. The data independence between the two decoding channels can also be used to parallelize the left and right channel transforms.

MP3 DESIGN FLOW

A design space exploration exercise is done with ESE to implement the MP3 decoder on a suitable platform that meets the real time constraint of 26.12 ms on the frame delay. In other words, the delay for each frame from the beginning of huffman decoding to the end of PCM output must be less than 26.12 ms. During this design space exploration, we start with a pure software implementation and incrementally move the compute intensive functions to hardware processors until the timing constraint is satisfied. The timed TLMs generated by ESE and

simulated with a sample MP3 file, as input, to estimate the performance of the design and to determine if it meets the timing constraint.

The chosen underlying implementation technology is Xilinx Virtex-II FPGA [196]with a maximum clock rate of 100 MHz. For software implementation, a Xilinx Microblaze (MB) processor is used on the FPGA chip. MB interfaces with the open peripheral bus (*OPB* and an off-chip SRAM is used to store the program and data. All hardware processors are generated using the NISC tools and they interface to the double handshake bus (*DHB*). Since the OPB and DHB protocols are different, a transducer is used to interface between them. The transducer component, therefore, enables communication between MB and the hardware processors.

We start with a software implementation, in which all the MP3 functions are mapped to MB. We will refer to this mapping as *SW+0*. The 0 indicates the lack of any hardware processors. The timed TLM for $SW+0$ was generated by ESE and the frame decoding time was estimated to be $35.66ms$. Based on this estimation, the $SW+0$ design of the MP3 decoder does not meet the decoding time constraint.

As a next step, we decided to add a hardware processor to implement the DCT function. We will refer to the new design as *SW+1*, in which 1 refers to the DCT hardware processor. The DCT hardware is generated using the NISC tools and it uses the (*DHB*) interface protocol, as mentioned earlier. A transducer (*Tx*) was also introduced to connect OPB and DHB. The timed TLM for $SW+1$ was generated by ESE and the frame decoding time was estimated to be $32.89ms$. Based on this estimation, the $SW+1$ design of the MP3 decoder also does not meet the decoding time constraint. The improvement over $SW+0$ was not too large because of the communication overhead caused by Tx.

To further improve the design performance, without much effort, we decided to use two instances of the DCT hardware processor to execute the DCT function

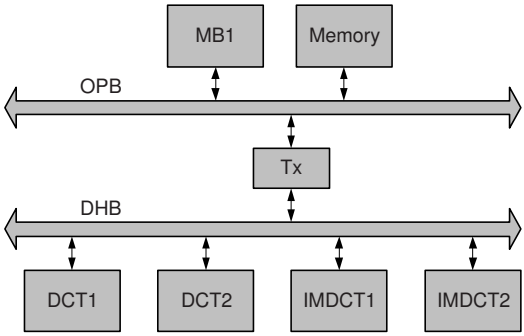

FIGURE 8.13 MP3 decoder platform SW+4

for the two decoding channels in parallel. This design is referred to as *SW+2* because of the two hardware processors. The timed TLM for $SW + 2$ was generated by ESE and the frame decoding time was estimated to be $29.99ms$. Again, the speed up over $SW + 1$ design was only marginal. The $SW + 2$ design of the MP3 decoder also did not meet the decoding time constraint.

As a next step, we created a *SW+4* design that included two instances each of DCT and IMDCT hardware processors. Therefore, the IMDCTs were also accelerated using specialized hardware. This platform in shown in Figure 8.13. The timed TLM for $SW + 4$ was generated by ESE and the frame decoding time was estimated to be $15.96ms$. Based on this estimation, $SW + 4$ design of the MP3 decoder met the decoding time constraint of $26.12ms$. As a result, the $SW + 4$ design was selected for implementation.

The above four platforms and mappings were created graphically in ESE and TLMs were automatically generated for evaluation of the respective designs. The TLMs were then used to synthesize software and hardware for the Microblaze soft-core processor and the Xilinx FPGA by the ESE back end. The generated software and hardware were exported to Xilinx Embedded Development Kit (EDK) for bitstream generation and programming of the FPGA. The programmed FPGA was tested with various MP3 sample inputs. In the next section, we will present various results pertaining to design of the MP3 decoder for the four platforms using ESE.

8.4.3 RESULTS

In this section we will discuss the results for system level design of the MP3 decoder with ESE. We will discuss four designs $SW+0$ to $SW+4$ as described above. The results for ESE front end demonstrate the benefits of using TLMs for early design performance estimation. The back end results demonstrate that automatic software and hardware synthesis can lead to design quality that is comparable to manual design. Automatic synthesis naturally leads to huge productivity gain in both design development and validation time. The overall design time is reduced from several months to less than a week as a result of using automatic system level design tools.

TLM ACCURACY

The MP3 design flow is simplified by the interactive graphical design decisions and automatic TLM generation. The design decisions of adding hardware processors were based on the estimation provided by the timed TLMs. Therefore, it is crucial that the TLM estimation is accurate enough for the design decisions to be made reliably. To determine if TLM estimation is accurate, let us compare

the timing estimates provided by TLMs to actual board measurements for the same designs.

Figure 8.14 compares the speed and accuracy of automatically generated TLMs with traditional models. The X-axis shows the execution time of the model and the Y-axis is the relative accuracy of the timing reported by the model. The actual board design is the naturally the reference for measuring accuracy. It can be seen that the CAM provides timing estimation that is identical to the board measurements. Since the CAM is cycle accurate, this is to be expected. However, the simulation time of the CAM is in the order of 15 to 18 hours for each MP3 sample frame. This is inordinately long for any reasonable design space exploration.

Typically, designers use instruction set simulation model (*ISM*) of a processor to speed up simulation. An ISM models the processor micro-architecture in a high level language such as C/C++. The binary of the software is loaded into the ISM memory. During simulation, the ISM interprets the instruction stream and updates the processor state. The hardware peripherals may be modeled in RTL using VHDL or Verilog. The high level processor model is instantiated as a module in Verilog. The ISM is typically faster than the CAM because it does not model the processor at the cycle-accurate level. However, the performance estimation accuracy of the ISM may vary based on the quality of the processor model. In the case of the MP3 designs, the accuracy of the ISM varied from 50% to 80% compared to board measurements. The unpredictable accuracy of

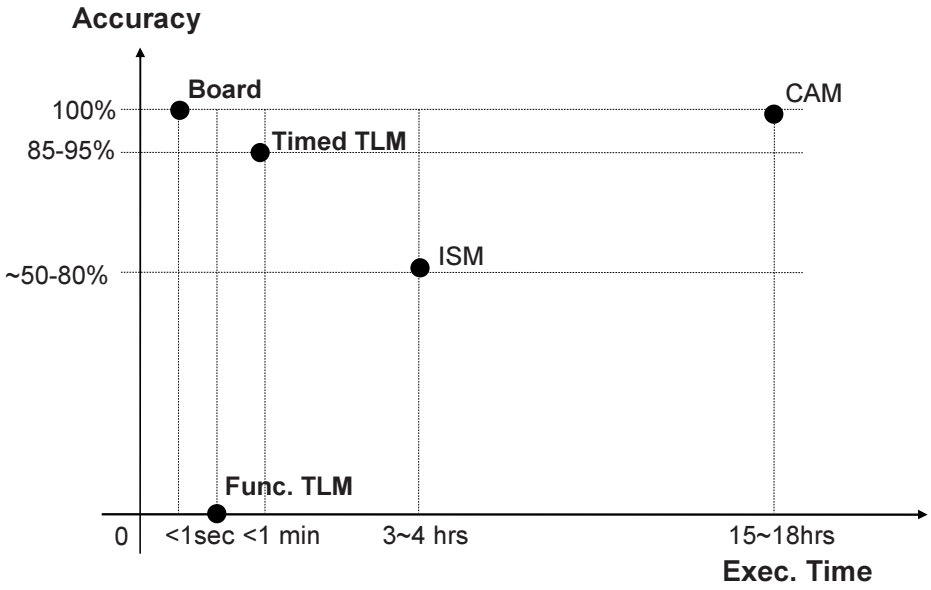

FIGURE 8.14. Execution speed and accuracy trade-offs for embedded system models

ISMs makes them unsuitable for early design space exploration. Furthermore, although the simulation speed of ISMs was about 5 times faster than the CAMs, it was still in the order of few hours.

The TLMs generated automatically by the ESE front end were two orders of magnitude faster than the ISM or the CAM. The timed TLMs were generated for all the design in under a minute and simulated under a minute as well. In contrast with the ISM, the timed TLMs were consistently accurate for all the platforms. A marginal error of under 15% was found in the TLM performance estimation. Therefore, designers can use timed TLMs for early estimation with a high degree of confidence.

In Figure 8.14, we distinguish between timed and untimed (or functional) TLMs. While the timed TLMs are used for performance estimation, the high simulation speed of functional TLMs makes them ideal for software development. It must be noted that functional TLMs may be generated even for a partial or test application. The process code may be developed using the functional TLM as a virtual platform. The results therefore demonstrate the efficacy and suitability of TLMs for early application development and reliable performance estimation.

DESIGN QUALITY

One of the primary concerns of automatic synthesis methods is the quality of design. Various metrics for design quality may be used. Some of the most common metrics are performance, silicon footprint and power dissipation. Generally speaking, it is difficult to evaluate the efficiency of a synthesis method by comparing its output to a manual design. The manual design is highly sen-

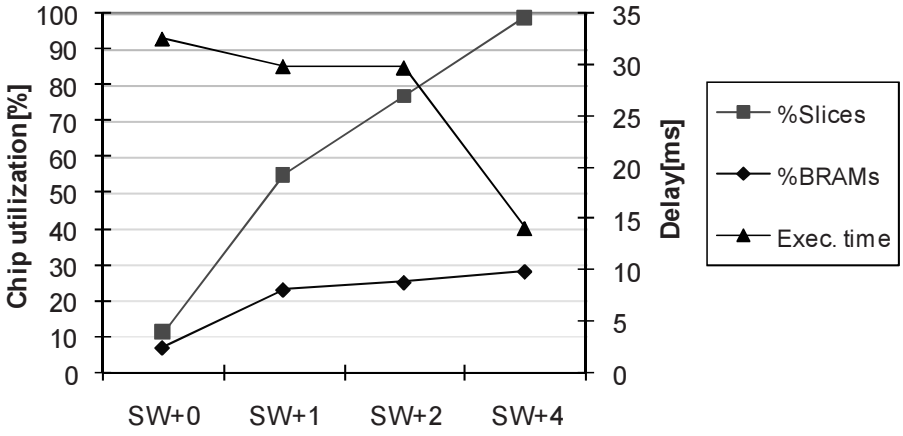

FIGURE 8.15 MP3 manual design quality

the designer. Nonetheless, a comparison of synthesized designs with an expert manual designer may give us better insight into the industrial viability of the synthesis tool.

To evaluate the quality of implementation produced by ESE back end, an expert designer created the software/hardware implementations of the four MP3 decoder designs described earlier. The hardware PEs were designed in RTL and the software was implemented directly on the FPGA using the Xilinx EDK tools. Figure 8.15 shows the performance and area of the manual designs.

In order to evaluate the performance of the designs, a sample MP3 file was loaded on the on-board memory and used as input. The average decoding time for each frame is shown in milliseconds. The pure software design is too slow to meet the 26.12 millisecond decoding time constraint. As predicted by TLM simulation, only the $SW+4$ implementation was able to meet the specified real time constraint. The design area is indicated by the percentage of block RAMs (BRAMs) and FPGA slices used by the implementation. The hardware PEs, namely the DCT and the IMDCT, had a hardwired controller implementation, which justifies the high number of slices used by the $SW+4$ implementation.

Figure 8.16 shows the performance and area of designs generated automatically from ESE. Compared to the corresponding manual designs, the performance of the generated designs was almost identical. In this case too, only the $SW+4$ design was able to meet the real time constraints imposed by the application. The area of the generated designs was different compared the manual designs. Notably, fewer slices were used in the generated design but

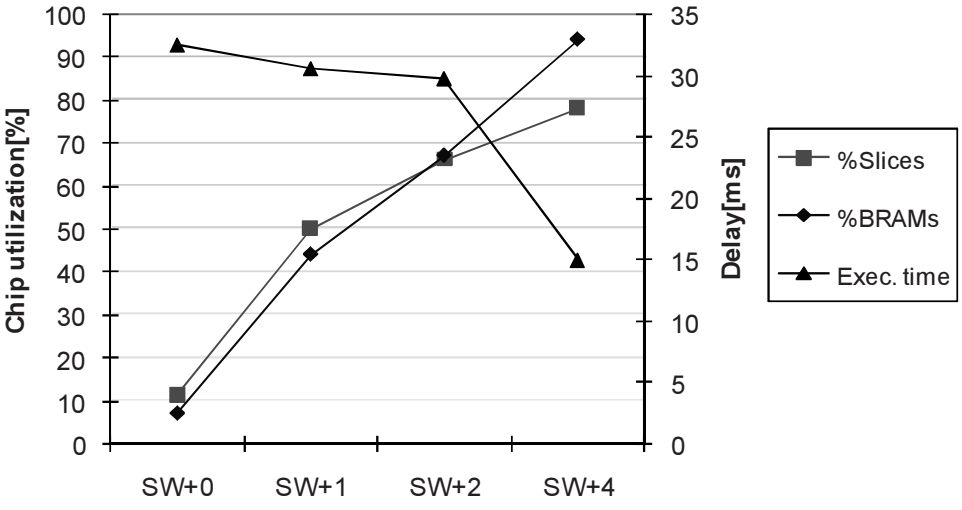

FIGURE 8.16 Automatically generated MP3 design quality

template was used for the hardware PEs in automatically generated designs. In contrast to the manual designed hardware PEs, NISC uses control words in memory to drive the data path. Therefore, NISC implementations are generally memory intensive. However, all the design could still fit on the target Virtex-II FPGA. The total number of FPGA resources used by automatically generated designs was comparable to the manual designs. Therefore, we can conclude that automatically generated designs are comparable to manual designs in terms of quality metrics of performance and area. This is a significant argument in favor of using automatic system level design tools.

PRODUCTIVITY GAINS

The single most important factor that drives the rise in design abstraction level is productivity gain. Typically, designs descriptions at higher abstraction levels are more compact, understandable and easily modified. Therefore, greater optimization opportunities are available at higher abstraction level. The two key productivity metrics we consider here are the design development time and validation time. Development time directly translates to design cost and time to market. Naturally, reducing the development time is always desirable. Similarly, design validation time directly impacts quality of design which is an important factor is product success.

Figure 8.17 illustrates the productivity gain in development time as a result of using ESE. Traditional design practice starts with RTL and embedded SW coding for selected platforms. The reference C specification model is used for developing test bench to verify the cycle accurate models. For MP3 platforms

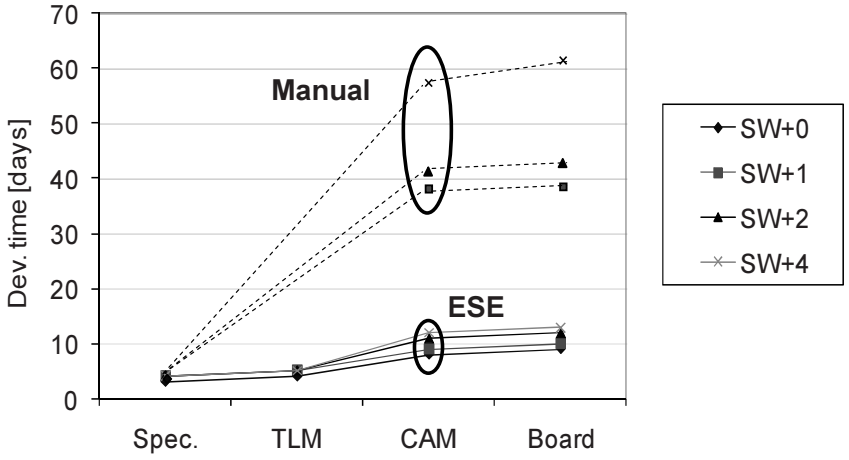

FIGURE 8.17 Development productivity gains from model automation

with HW components, the RTL development time was in the order of months. As a result, board prototypes for these designs took between 40 to 60 days. ESE drastically cuts prototype development time by automatically generating TLM and RTL models. With ESE, the final board prototypes for MP3 designs were available in less than a week after the specification model was finalized. Consequently, ESE results in significant savings in design cost and shorter development cycles.

Figure 8.18 illustrates the productivity gain resulting from a TLM based design methodology supported by ESE. As a consequence of traditional cycle accurate modeling, designers must make design optimizations and changes on RTL and low level SW code. Each change needs to be verified using time consuming cycle accurate simulations. Each CAM simulation of the MP3 designs took 15 to 18 hours for MP3 designs. This is a significant component of design time. Although at speed on-board verification is faster than even reference application C model simulation, bugs found in on-board testing are difficult to trace back to the CAM.

TLMs remove the burden of cycle accurate simulations by moving the design abstraction to a higher level. ESE generated TLMs execute at the same speed as reference C simulation. Design changes can made at the transaction level and can hence be verified and debugged using the automatically generated high speed TLMs. TLMs are easier to debug and maintain because their code size is at least an order of magnitude less than corresponding CAM code size.

Automatic CAM generation from TLM is also less likely to introduce bugs in the design compared to manual CAM optimizations. This has been true in

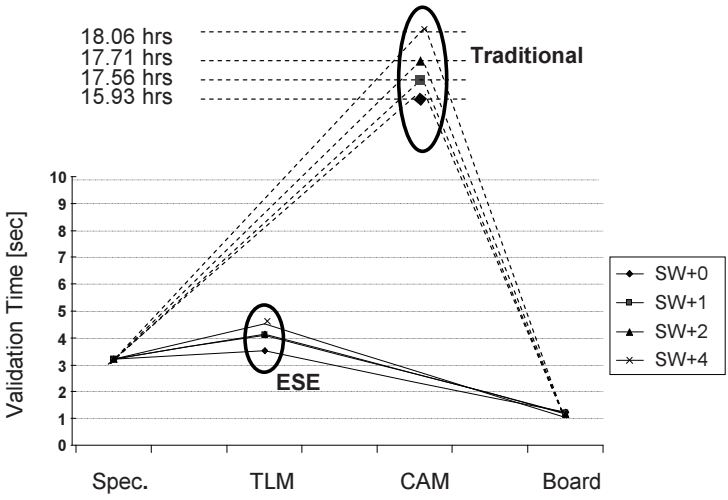

FIGURE 8.18 Validation productivity gain from using TLM vs. CAM

the past when the modeling abstraction moved from gate level to RTL with the use of logic synthesis tools. Therefore, ESE reduces validation time from an order of several hours or even days to a few seconds. As a results, designers can use ESE to make platform and application optimizations at a higher level, automatically generate TLMs and verify the optimizations in a few seconds.

8.5 SUMMARY

We discussed several academic and commercial tools for various aspects of embedded system design. These range from system level modeling and simulation to automatic synthesis of software and hardware. We also presented a case study for design of MP3 decoder on a heterogeneous platform. The results show that the design methods presented in this book can work for practical embedded system design. The automatic design tools provide fast and accurate models, design quality comparable to manual and huge productivity gains. These results point to the significant advantages and benefits of using embedded system design methods described in this book.

References

[1] Samar Abdi and Daniel Gajski. Functional validation of system level model transformations. *International Journal of Parallel Programming*, 34(1):29–59, February 2006.

[2] Samar Abdi, Dongwan Shin, and Daniel Gajski. Automatic communication refinement for system level design. In *Design Automation Conference*, pages 300–305, 2003.

[3] Accellera. *RTL Semantics: Draft Specification, Version 0.8*. Working Group of the Architectural Language Committee, February 2001.

[4] Advanced RISC Machines Ltd. (ARM). *AMBA Specification (Revision 2.0)*, 1999. ARM IHI 0011A.

[5] Perry Alexander. *System Level Design with Rosetta*. Morgan Kaufmann, 2006.

[6] Charles André. Representation and analysis of reactive behaviors: A synchronous approach. In *Computational Engineering in System Applications (CESA)*, Lille, France, July 1996.

[7] Peter J. Ashenden. *The Designer's Guide to VHDL*. Morgan Kaufmann, December 1995.

[8] Motor Industry Research Association. *MISRA-C 2004: Guidelines for the Use of the C Language in Critical Systems*. 2004.

[9] AUTOSAR Partnership. Autosar: Automotive open system architecture. http://www.autosar.org/.

[10] M. Balakrishnan and P. Marwedel. Integrated scheduling and binding: A synthesis approach for design space exploration. In *Design Automation Conference*, pages 68–74, Las Vegas, NV, June 1989.

[11] Felice Balarin, Massimiliano Chiodo, Paolo Giusto, Harry Hsieh, Attila Jurecska, Luciano Lavagno, Claudio Passerone, Alberto Sangiovanni-Vincentelli, Ellen Sentovich, Kei Suzuki, and Bassam Tabbara. *Hardware-Software Co-Design of Embedded Systems: The POLIS Approach*. Kluwer Academic Publishers, 1997.

[12] Felice Balarin, Harry Hsieh, Luciano Lavagno, Claudio Passerone, Alessandro Pinto, Alberto Sangiovanni-Vincentelli, Yosinori Watanabe, and Guang Yang. Metropolis: A

design environment for heterogeneous systems. In Wayne Wolf and Ahmed Jerraya, editors, *Multiprocessor Systems-on-Chips*. Morgan Kaufmann, 2004.

[13] M. Barbacci. Instruction set processor specification (isps): The notation and its application. *IEEE Transactions on Computers*, C-30(1):24–40, January 1981.

[14] M. R. Barbacci. A comparison of register transfer level languages for describing computers and other digital systems. *IEEE Transactions on Computers*, C-24(2), February 1975.

[15] M. R. Barbacci. Instruction set processor specifications for simulation, evaluation, and synthesis. In *Design Automation Conference*, pages 64–72, San Diego, CA, United States, 1979.

[16] C. G. Bell and A. Newell. *Computer Structures: Readings and Examples*. McGraw-Hill, 1971.

[17] Rudy Belliardi, Ben Brosgol, Peter Dibble, David Holmes, and Andy Wellings. *The Real-Time Specification for Java*, 2006.

[18] Albert Beneviste, Paul Caspi, Stephen A. Edwards, Nicolas Halbwachs, Paul Le Guernic, and Robdert de Simone. The synchronous languages twelve years later. *Proceedings of the IEEE*, 91(1):64–83, January 2003.

[19] L. Bening and H. Foster. *Principles of Verifiable RTL Design*. Kluwer Academic Publishers, 2000.

[20] Luca Benini, Davide Bertozzi, Alessandro Bogliolo, Francsco Menichelli, and Mauro Olivieri. MPARM: Exploring the multi-processor SoC design space with SystemC. *Journal of VLSI Signal Processing*, 41(2):169–182, September 2005.

[21] Gerard Berry. The foundations of Esterel. In Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction: Essays in Honor of Robin Milner*. MIT Press, 2000.

[22] Greet Bilsen, Marc Engels, Rudy Lauwereins, and Jean Peperstraete. Cyclo-static dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, February 1996.

[23] Grady Booch, Ivar Jacobson, and James Rumbaugh. *Unified Modeling Language (UML) Specification, Version 1.5*. Object Management Group (OMG), March 2003.

[24] Aimen Bouchhima, Iuliana Bacivarov, Wassim Youssef, Marius Bonaciu, and Ahmed A. Jerraya. Using abstract CPU subsystem simulation model for high level HW/SW architecture exploration. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, Shanghai, China, January 2005.

[25] R. K. Brayton, R. Camposano, G. De Micheli, R.H.J.M. Otten, and J. Van Eijndhoven. The yorktown silicon compiler system. In Daniel D. Gajski, editor, *Silicon Compilation*. Addison-Wesley, 1988.

[26] Forrest D. Brewer and Daniel D. Gajski. An expert-system paradigm for design. In *Design Automation Conference*, pages 203–509, Las Vegas, NV, June 1986.

[27] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computer*, C-35(8):677–691, August 1986.

[28] Joseph Buck, Soonhoi Ha, Edward A. Lee, and David G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *International Journal of Computer Simulation, Special Issue on Simulation Software Development*, 4:155–182, April 1994.

[29] David R. Butenhof. *Programming with POSIX Threads*. Addison-Wesley, 1997.

[30] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems*. Kluwer Academic Publishers, 1999.

[31] Lukai Cai and Daniel Gajski. Transaction level modeling: An overview. In *International Symposium on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Newport Beach, CA, USA, October 2003.

[32] Lukai Cai, Andreas Gerstlauer, and Daniel Gajski. Retargetable profiling for rapid, early system-level design space exploration. In *Design Automation Conference*, San Diego, CA, USA, June 2004.

[33] Jean-Paul Calvez. *Embedded Real-Time Systems: A Specification and Design Methodology*. John Wiley and Sons, 1993.

[34] Raul Camposano. Path-based scheduling for synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(1):85–93, January 1991.

[35] Raul Camposano and Wayne Wolf (editors). *High-Level VLSI Synthesis*. Kluwer Academic Publishers, 1991.

[36] Raul Camposano and Wolfgang Rosenstiel. A design environment for the synthesis of integrated circuits. In *EUROMICRO Symposium on Microprocessing and Microprogramming*, Brussels, Belgium, September 1985.

[37] Carbon Design Systems. Carbon SoC Designer. http://www.carbondesignsystems.com/.

[38] Celoxica Ltd. *Handel-C Language Reference Manual*, 2005.

[39] Center for Embedded Computer Systems (CECS). Embedded System Environment, Center for Embedded Computer Systems, University of California, Irvine. http://www.cecs.uci.edu/~ese, 2008.

[40] Center for Embedded Computer Systems (CECS). NISC Technology. http://www.cecs.uci.edu/~nisc/, 2008.

[41] D. Chen, J. Cong, Y. Fan, G. Han, W. Jiang, and Z. Zhang. xpilot: A platform-based behavioral synthesis system. In *SRC Techcon Conference*, October 2005.

[42] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, January 2000.

[43] CoFluent Design. CoFluent Studio. http://www.cofluentdesign.com/.

[44] Lockheed Martin Corporation. *JSF Air Vehicle C++ Coding Standards for the System Development and Demonstration Program*, 2005.

[45] P. Coussy and A. Morawiec, editors. *High-Level Synthesis: from Algorithm to Digital Circuit*. Springer, 2008.

[46] CoWare. http://www.coware.com/.

[47] G. de Jong. A uml-based design methodology for real-time and embedded systems. In *IEEE International Conference Design and Test in Europe (DATE)*, pages 776–779, Paris, France, March 2002.

[48] S. Devadas, H.K. T. Ma, and A. R. Newton. On the verification of sequential machines at different levels of abstraction. In *Design Automation Conference*, pages 271–276, Miami Beach, FL, USA, June 1987.

[49] Srinivas Devadas and A. Richard Newton. Algorithm for allocation in data path synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(7):768–781, July 1989.

[50] S. W. Director, A. C. Parker, D. P. Siewiorek, and D. E. Thomas. A design methodology and computer design aids for digital vlsi systems. *IEEE Transactions on Circuits and Systems*, 28(7):634–645, July 1981.

[51] Rainer Dömer, Andreas Gerstlauer, and Daniel Gajski. *SpecC Language Reference Manual, Version 2.0*. SpecC Technology Open Consortium (STOC), 2002.

[52] Rainer Dömer, Andreas Gerstlauer, Junyu Peng, Dongwan Shin, Lukai Cai, Haobo Yu, Samar Abdi, and Daniel Gajski. System-on-Chip Environment: A SpecC-based Framework for Heterogeneous MPSoC Design. *EURASIP Journal on Embedded Systems (JES)*, 2008(647953):13, 2008.

[53] dSPACE (Digital Signal Processing And Control Engineering). TargetLink. http://www.dspace.com/.

[54] Bruce Eckel. *Thinking in Java*. Prentice-Hall, Upper Saddle River, N.J., 2003.

[55] Stephen A. Edwards. *Languages for Digital Embedded Systems*. Kluwer Academic Publishers, 2000.

[56] J. P. Elliot. *Understanding Behavioral Synthesis: A Practical Guide to High-Level Design*. Kluwer Academic Publishers, 1999.

[57] Esterel Technologies. Scade suite. http://www.esterel-technologies.com/.

[58] Forte Design Systems. Cynthesizer. http://www.forteds.com/, 2008.

[59] Eclipse Foundation. Eclipse. http://www.eclipse.org/.

[60] D. Gajski and R. Kuhn. New vlsi tools. *Computer Magazine*, pages 11–14, December 1983.

[61] Daniel Gajski, Nikil Dutt, Allan Wu, and Steve Lin. *High-Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic Publishers, 1992.

[62] Daniel D. Gajski. *Principles of Digital Design*. Prentice-Hall, September 1996.

[63] Daniel D. Gajski, Frank Vahid, Sanjiv Narayan, and Jie Gong. *Specification and Design of Embedded Systems*. Prentice-Hall, July 1994.

[64] Daniel D. Gajski, Frank Vahid, Sanjiv Narayan, and Jie Gong. SpecSyn: An environment supporting the specify-explore-refine paradigm for hardware/software system design. *IEEE Transactions on Very Large Scale Integrated Systems (TVLSI)*, 6(1):84–100, March 1998.

[65] Daniel D. Gajski, Jianwen Zhu, Rainer Doemer, Andreas Gerstlauer, and Shuqing Zhao. *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, March 2000.

[66] Lovic Gauthier, Sungjoo Yo, and Ahmed A. Jerraya. Automatic Generation and Targeting of Application-Specific Operating Systems and Embedded Systems Software. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(11), November 2001.

[67] Geensys. Autosar builder. http://www.geensys.com/.

[68] Marc Geilen and Twan Basten. Requirements on the execution of Kahn process networks. In *European Symposium on Programming (ESOP)*, pages 319–334, Warsaw, Poland, April 2003.

[69] Gentleware. Poseidon for uml. http://www.gentleware.com/.

[70] Patrice Gerin, Sungjoo Yoo, Gabriela Nicolescu, and Ahmed A. Jerraya. Scalable and flexible cosimulation of SoC designs with heterogeneous multiprocessor target architectures. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, Yokohama, Japan, January 2001.

[71] Andreas Gerstlauer. *Modeling Flow for Automated System Design and Exploration*. PhD thesis, Information and Computer Science, University of California, Irvine, May 2004.

[72] Andreas Gerstlauer, Rainer Dömer, Junyu Peng, and Daniel D. Gajski. *System Design: A Practical Guide with SpecC*. Kluwer Academic Publishers, 2001.

[73] Andreas Gerstlauer, Junyu Peng, Dongwan Shin, Daniel Gajski, Atsushi Nakamura, Dai Araki, and Yuuji Nishihara. Specify-Explore-Refine (SER): From specification to implementation. In *Proceedings of the Design Automation Conference (DAC)*, pages 586–591, Anaheim, CA, USA, June 2008.

[74] Andreas Gerstlauer, Dongwan Shin, Junyu Peng, Rainer Doemer, and Daniel Gajski. Automatic, layer-based generation of system-on-chip bus communication models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(9):1676–1687, September 2007.

[75] Andreas Gerstlauer, Haobo Yu, and Daniel D. Gajski. RTOS modeling for system level design. In Ahmed A. Jerraya, Sungjoo Yu, Norbert Wehn, and Diedrik Verkest, editors, *Embedded Software for SoC*. Springer, September 2003.

[76] Andreas Gerstlauer, Shuqing Zhao, Daniel Gajski, and Arkady Horak. Specc system-level design methodology applied to the design of a gsm vocoder. In *SASIMI*, 2000.

[77] Frank Ghenassia, editor. *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Springer, November 2005.

[78] Gordon. Specification and verification of hardware, October 1992.

[79] James Gosling, Bill Joy, Guy L. Steele Jr., and Gilad Bracha. *The Java Language Specification*. Addison-Wesley, third edition, 2005.

[80] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, second edition, 1999.

[81] Torsten Grötker, Stan Liao, Grant Martin, and Stuart Swan. *System Design with SystemC*. Springer, 2002.

[82] Yuri Gurevich. Evolving algebras 1993: Lipari guide. In Egon Börger, editor, *Specification and Validation Methods*. Oxford University Press, 1995.

[83] Soonhoi Ha, Sungchan Kim, Choonseung Lee, Youngmin Yi, Seongnam Kwon, and Young-Pyo Joo. PeaCE: A hardware-software codesign environment of multimedia embedded systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 12(3):1–25, 2007.

[84] L. Hafer and A. C. Parker. Register transfer level automatic digital design: The allocation process. In *Design Automation Conference*, Las Vegas, NV, United States, June 1978.

[85] Nicolas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous dataflow programming language Lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[86] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[87] David Harel and Amnon Naamad. The STATEMATE semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(4):293–333, October 1996.

[88] Graham Hellestrand. The engineering of supersystems. *IEEE Computer*, 38(1):103–105, January 2005.

[89] F. Herrera, H. Posadas, P. Sanchez, and E. Villar. Systematic Embedded Software Generation from SystemC. In *Proceedings of the Design Automation and Test Conference in Europe*, Munich, Germany, March 2003.

[90] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[91] Andreas Hoffmann, Heinrich Meyr, and Rainer Leupers. *Architecture Exploration for Embedded Processors with LISA*. Kluwer Academic Publishers, 2003.

[92] Matthias Homann. *OSEK: Betriebssystem-Standard für Automotive und Embedded Systems*. mitp-Verlag, 2 edition, 2005.

[93] Yonghyun Hwang, Samar Abdi, and Daniel Gajski. Cycle-approximate retargetable performance estimation at the transaction level. In *IEEE International Conference Design and Test in Europe (DATE)*, pages 3–8, Munich, Germany, March 2008.

[94] IBM. Telelogic rhapsody. http://www.ibm.com/.

[95] MathWorks Inc. *MATLAB and Simulink Student Edition*. Pearson, 2008.

[96] National Instruments Inc. and Robert H. Bishop. *LabVIEW Student Edition*. Prentice-Hall, 2007.

[97] Tensilica Inc. Xtensa xplorer design environment. http://tensilica.com/.

[98] International Organization for Standardization. *Reference Model of Open System Interconnection (OSI)*, second edition, 1994. ISO/IEC 7498 Standard.

[99] International Technology Roadmap for Semiconductors (ITRS). ITRS Home. http://www.itrs.net/, 2008.

[100] R. S. Janka. *Specification and Design Methodology for Real-Time Embedded Systems*. Kluwer Academic Publishers, 2004.

[101] Axel Jantsch. *Modeling Embedded Systems and SoCs: Concurrency and Time in Models of Computation*. Morgan Kaufmann, 2004.

[102] A. A. Jerraya, H. Ding, P. Kission, and M. Rahmouni. *Behavioral Synthesis and Component Reuse with VHDL*. Kluwer Academic Publishers, 1997.

[103] Ahmed A. Jerraya. Long term trends for embedded system design. In *EUROMICRO Symposium on Microprocessing and Microprogramming*, pages 20–26, Rennes, France, September 2004.

[104] Gilles Kahn. The semantics of a simple language for parallel programming. In *Information Processing*, pages 471–475, Stockholm, Sweden, August 1974.

[105] Joachim Keinert, Martin Streubühr, Thomas Schlichter, Joachim Falk, Jens Gladigau, Christian Haubelt, Jürgen Teich, and Mike Meredith. SystemCoDesigner - an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 14(1):1–23, 2009.

[106] Brian Kernighan and Dennis Ritchie. *The C programming language*. Prentice-Hall, Englewood Cliffs, NJ, 1988.

[107] Kurt Keutzer, Sharad Malik, Richard A. Newton, Jan M. Rabaey, and Alberto Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1523–1543, December 2000.

[108] A. A. Khan, Carolyn McCreary, and M. S. Jones. A comparison of multiprocessor scheduling heuristics. In *International Conference on Parallel Processing*, pages 243–250, 1994.

[109] Wolfgang Klingauf, Robert Günzel, Oliver Bringmann, Pavel Parfuntesu, and Mark Burton. GreenBus: A generic interconnect fabric for transaction-level modeling. In *Design Automation Conference*, San Francisco, CA, USA, July 2006.

[110] D. W. Knapp. *Behavioral Synthesis: Digital System Design Using the Synopsys Behavioral Compiler*. Prentice-Hall, 1996.

[111] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Applications*. Kluwer Academic Publishers, 1997.

[112] Matthias Krause, Oliver Bringmann, and Wolfgang Rosenstiel. Target software generation: An approach for automatic mapping of SystemC specifications onto real-time operating systems. 10(4):229–251, December 2005.

[113] T. Kropf. *Introduction to Formal Hardware Verification*. Springer, 1999.

[114] D. Ku and G. De Micheli. *High Level Synthesis of ASICs under Timing and Synchronization Constraints*. Kluwer Academic Publishers, 1992.

[115] Seongnam Kwon, Yongjoo Kim, Woo-Chul Jeun, Soonhoi Ha, and Yunheung Paek. A retargetable parallel programming framework for MPSoC. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 13(3), 2008.

[116] Luciano Lavagno, Grant Martin, and Bran Selic, editors. *UML for Real: Design of Embedded Real-Time Systems*. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[117] Luciano Lavagno, Alberto Sangiovanni-Vincentelli, and Ellen Sentovich. Models of computation for embedded system design. In Ahmed Jerraya and Jean Mermet, editors, *System-Level Synthesis*. Kluwer Academic Publishers, 1999.

[118] Edward A. Lee. Consistency in dataflow graphs. *IEEE Transactions on Parallel and Distributed Systems*, 2(2):223–235, April 1991.

[119] Edward A. Lee. The problem with threads. *IEEE Computer*, 39(5):33–42, May 2006.

[120] Edward A. Lee and David G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, September 1987.

[121] INMOS Limited. *Occam 2 Reference Manual*. Prentice-Hall, 1988.

[122] Joe S. Lis and Daniel D. Gajski. Synthesis from vhdl. In *IEEE International Conference on Computer Design*, 1988.

[123] Lucky Lo Chi Yu Lo and Samar Abdi. Automatic systemc tlm generation for custom communication platforms. In *International Conference on Computer Design*, pages 41–46, 2007.

[124] H. De Man, J. Rabaey, P. Six, and L. Claesen. Cathedral-II: A Silicon Compiler for Digital Signal Processing. *IEEE Design and Test of Computers*, 3(6):13–25, November 1986.

[125] Florence Maraninch. The Argos language: Graphical representation of automata and description of reactive systems. In *International Conference on Visual Languages*, Kobe, Japan, October 1991.

[126] Grant Martin and Wolfgang Müller, editors. *UML for SOC Design*. Springer, 2005.

[127] Peter Marwedel. The MIMOLA design system: Detailed description of the software system. In *Design Automation Conference*, pages 59–63, San Diego, CA, United States, June 1979.

[128] Peter Marwedel. A new synthesis algorithm for mimola software system. In *Design Automation Conference*, pages 131–137, Las Vegas, NV, June 1986.

[129] Peter Marwedel. *Embedded Systems Design*. Kluwer Academic Publishers, 2003.

[130] Peter Marwedel. *Embedded System Design*. Springer, 2006.

[131] MathWorks Inc. Real-Time Workshop. http://www.mathworks.com/.

[132] MathWorks Inc. Simulink - Simulation and Model-Based Design. http://www.mathworks.com/.

[133] M. C. McFarland. The value trace: A database for automated digital design. Master's thesis, Carnegie-Mellon University, December 1978.

[134] M. C. McFarland. Using bottom-up design technique in the synthesis of digital hardware from abstract behavioral descriptions. In *Design Automation Conference*, Las Vegas, NV, June 1986.

[135] M.C. McFarland. Formal verification of sequential hardware: A tutorial. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(5):633–653, May 1993.

[136] K.L. McMillan. *Symbolic Model Checking: An approach to the State Explosion Problem*. Kluwer Academic Publishers, 1993.

[137] Mentor Graphics. The EDA Technology Leader - Mentor Graphics. http://www.mentor.com/, 2008.

[138] P. Michel, U. Lauther, and P. Duzy, editors. *Synthesis Approach to Digital System Design*. Kluwer Academic Publishers, 1992.

[139] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[140] Microelectronic Embedded Systems Laboratory. SPARK: High-Level Synthesis using Parallelizing Compiler Techniques. http://mesl.ucsd.edu/spark/, 2008.

[141] Robin Milner. *A Calculus of Communicating Systems*. Springer, 1980.

[142] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[143] Andre Nacul and Tony Givargis. Synthesis of Time-Constrained Multitasking Embedded Software. volume 11, pages 822–847, October 2006.

[144] NEC System Technologies Ltd. CyberWorkBench - System LSI Design Environment to implement All-in-C Concept. http://www.necst.co.jp/, 2008.

[145] H. Nikolov, M. Thompson, T. Stefanov, A. D. Pimentel, S. Polstra, R. Bose, C. Zissulescu, and E. F. Deprettere. Daedalus: Toward composable multimedia MP-SoC design. In *Proc. of the ACM/IEEE Int. Design Automation Conference (DAC '08)*, pages 574–579, June 2008.

[146] Achim Nohl, Gunnar Braun, Oliver Schliebusch, Rainer Leupers, Heinrich Meyr, and Andreas Hoffmann. A universal technique for fast and flexible instruction-set architecture simulation. In *Design Automation Conference*, New Orleans, LA, USA, June 2002.

[147] Object Management Group (OMG). Unified modeling language (UML). http://www.uml.org/.

[148] Object Management Group (OMG). *Common Object Request Broker Architecture: Core Specification, Version 3.0.3*, 2004.

[149] Object Management Group (OMG). *OMG Systems Modeling Language (OMG SysML), Version 1.1*, 2008.

[150] Open SystemC Initiative (OSCI). http://www.systemc.org/, 2008.

[151] Alex Orailoglu and Daniel D. Gajski. Flow graph representation. In *Design Automation Conference*, pages 503–509, Las Vegas, NV, June 1986.

[152] Barry M. Pangrle and Daniel D. Gajski. Design tools for intelligent silicon compilation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(6):1098–1112, November 1987.

[153] Barry M. Pangrle and Daniel D. Gajski. Slicer: A state synthesizer for intelligent silicon compilation. In *IEEE International Conference on Computer Design*, Rye Brook, NY, October 1987.

[154] Thomas M. Parks. *Bounded Scheduling of Process Networks*. PhD thesis, Electrical Engineering and Computer Science, University of California, Berkeley, December 1995.

[155] P. Paulin and J. P. Knight. Algorithms for high-level synthesis. *IEEE Computer*, 6(6):18–31, November 1989.

[156] P. G. Paulin and J. P. Knight. Force-directed scheduling for behavioral synthesis of asic's. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(6):661–679, June 1989.

[157] Philippe Coussy and Dominique Heller. GAUT WEB SITE. http://web.univ-ubs.fr/gaut, 2008.

[158] Gordon D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60(61):17–139, July 2004.

[159] Chris Porthouse. Jazelle for execution environments. http://www.arm.com/pdfs/JazelleRCTWhitePaper_final1-0_pdf, May 2005.

[160] J. Rabaey, H. De Man, J. Vanhoof, G. Goossens, and F. Catthoor. Cathedral-II: A Synthesis System for Multiprocessor DSP Systems. In Daniel D. Gajski, editor, *Silicon Compilation*. Addison-Wesley, 1988.

[161] Mehrdad Reshadi and Daniel Gajski. A cycle-accurate compilation algorithm for custom pipelined datapaths. In *International Symposium on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2005.

[162] Sebastian Ritz, Matthias Pankert, Vojin Zivojnvic, and Heinrich Meyr. High-Level Software Synthesis for the Design of Communication Systems. *IEEE Journal on Selected Areas in Communications*, April 1993.

[163] Stewart Robinson. *Simulation: The Practice of Model Development and Use*. John Wiley and Sons, March 2004.

[164] Alberto Sangiovanni-Vincentelli. Quo Vadis SLD: Reasoning about the Trends and Challenges of System Level Design. *Proceedings of the IEEE*, 95(3):467–506, March 2007.

[165] Alberto Sangiovanni-Vincentelli and Grant Martin. The platform-based design and software design methodology for embedded systems. *IEEE Design and Test of Computers*, 18(6):23–33, November 2001.

[166] Gunar Schirner, Andreas Gerstlauer, and Rainer Doemer. Abstract, multifaceted modeling of embedded processors for system level design. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, Yokohama, Japan, January 2007.

[167] Dana Scott and Christopher Strachey. Toward a mathematical semantics for computer languages. Technical Report PRG-6, Oxford Programming Research Group, 1971.

[168] D. P. Siewiorek and M. R. Barbacci. The cmu rt-cad system: An innovative approach to computer aided design. In *AFIPS National Computer Conference*, pages 643–655, New York, NY, United States, June 1976.

[169] Artisan Software. Artisan studio. http://www.artisansoftwaretools.com/.

[170] Space Codesign Systems. http://www.spacecodesign.com/.

[171] SpecC Technology Open Consortium Office. SpecC Technology Open Consortium. http://www.specc.gr.jp/, 2008.

[172] The SPIRIT Consortium. *IP-XACT, Release 1.4*, March 2008.

[173] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, MA, 1997.

[174] Stuart Sutherland, Simon Davidmann, and Peter Flake. *SystemVerilog For Design: A Guide to Using SystemVerilog for Hardware Design and Modeling*. Springer, June 2003.

[175] Spark Systems. Enterprise architect. http://www.sparxsystems.com.au/.

[176] D. E. Thomas, E. M. Dirkes, R. A. Walker, J. V. Rajan, J. A. Nestor, and R. L. Blackburn. The system architect's workbench. In *Design Automation Conference*, pages 337–343, Anaheim, CA, June 1988.

[177] D. E. Thomas, C. Y. Hitchcock, T. J. Kowalski, J. V. Rajan, and R. A. Walker. Method of automatic data path synthesis. *IEEE Computer*, 16(12):59–70, December 1983.

[178] D. E. Thomas, E. D. Lagnese, R. A. Walker, J. A. Nestor, J.V Rajan, and R.L. Blackburn. *Algorithmic and Register-Transfer Level Synthesis: The System Architect's Workbench*. Kluwer Academic Publishers, 1990.

[179] Donald E. Thomas. *The Design and Analysis of an Automated Design Style Selector*. PhD thesis, Department of Electrical Engineering, Carnegie-Mellon University, 1977.

[180] Donald E. Thomas and Philip R. Moorby. *The Verilog Hardware Description Language*. Kluwer Academic Publishers, June 2002.

[181] C. J. Tseng and D. P. Siewiorek. Automated synthesis of data paths on digital systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 5(3):379–395, July 1986.

[182] Underbit Technologies Inc. MAD: MPEG audio decoder. http://www.underbit.com/, 2008.

[183] University of California, Los Angeles (UCLA). The xPilot System. http://cadlab.cs.ucla.edu/soc/, 2008.

[184] Frank Vahid and Tony Givargis. *Embedded System Design: A Unified Hardware/Software Introduction*. John Wiley and Sons, October 2001.

[185] Frank Vahid, Sanjiv Narayan, and Daniel D. Gajski. SpecCharts: A VHDL front-end for embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(6):694–706, June 1995.

[186] J. Vanhoof, K. V. Rompaey, I. Bolsens, G. Goossens, and H. DeMan. *High-Level Synthesis for Real-Time Digital Signal Processing*. Kluwer Academic Publishers, 1993.

[187] VaST Systems Technology Corporation. http://www.vastsystems.com/.

[188] Diederik Verkest, Karl Van Rompaey, Ivo Bolsens, and Hugo De Man. CoWare: A Design Environment for Heterogeneous Hardware/Software Systems. *Design Automation for Embedded Systems*, 1(4):357–386, October 1996.

[189] Virtutech. Virtutech Simics. http://www.virtutech.com/.

[190] K. Wakabayashi and T. Yoshimura. A resource sharing and control synthesis method for conditional branches. In *International Conference on Computer Aided Design*, pages 62–65, November 1989.

[191] Kazutoshi Wakabayashi. Cyber: High level synthesis system from software into asic. In Camposano and Wolf, editors, *High-Level Synthesis*. Kluwer Academic Publishers, 1991.

[192] John Waldron. *Introduction to RISC Assembly Language Programming*. Addison-Wesley, 1998.

[193] Andy Wellings. *Concurrent and Real-Time Programming in Java*. Wiley, 2004.

[194] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution time problem: Overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):1–53, April 2008.

[195] Wayne Wolf. *Computers as Components*. Morgan Kaufmann, 2001.

[196] Xilinx Inc. FPGA and CPLD Solutions from Xilinx, Inc. http://www.xilinx.com/, 2008.

[197] Haobo Yu. *Software Synthesis for System-on-Chip*. PhD thesis, Information and Computer Science, University of California, Irvine, June 2005.

[198] Henning Zabel, Wolfgang Mueller, and Andreas Gerstlauer. Accurate RTOS modeling and analysis with SystemC. In Wolfgang Ecker, Wolfgang Mueller, and Rainer Doemer, editors, *Hardware Dependent Software: Principles and Practice*. Springer, 2009.

[199] Gerhard Zimmermann. The MIMOLA design system: A computer aided digital processor design method. In *Design Automation Conference*, pages 53–58, San Diego, CA, United States, June 1979.