A modular design flow for very large design space explorations

Thorsten Jungeblut [1], Sven Lütkemeier [1], Gregor Sievers [1], Mario Porrmann [1], Ulrich Rückert [2]
[1] Heinz Nixdorf Institute, University of Paderborn, Germany
[2] Cognitive Interaction Technology - Center of Excellence, Bielefeld University, Germany

Introduction

The design of application specific and resource efficient digital circuits, like complex multiprocessor system on chips, often requires to choose among multiple possible configurations affecting both performance and resource consumption. A design space exploration (DSE) of the different architecture configurations and multiple target libraries usually implies hundreds of syntheses. In sub-90nm-technologies place and route (P&R) has to be performed to derive realistic results. This complexity makes the design space exploration interminable. To compare different implementations within a company sets of generally accepted constraints are required.

The product family of Cadence® Design Systems Inc. offers a large variety of tools for the design of microelectronic circuits to perform, for example, RTL synthesis, power estimation, place and route or verification. To speed up the iteration steps of the DSE, we have developed a semi-automatic tool flow, using the EDA environment of Cadence®. This tool flow is used to perform the exploration of very large design spaces with, for example hundreds of RTL synthesis, only limited by the computational power available at our group. To minimize the iteration time and to maximize the efficiency of our EDA hardware, we developed a load balancing system to distribute jobs to different computers.

Two-Staged Design Flow Based on the UPSLA Language

For development of processor architectures and the corresponding tool chains we use a dual design flow based on the Unified Processor Specification Language (UPSLA) as a reference description (cf. Figure 1, [1][4]). Based on this specification, a complete tool chain can be generated automatically. The tool chain consists of a C-compiler, an assembler, a linker, an instruction set simulator (ISS) and various debugging and profiling tools. The automatic generation of the tool chain allows the developer to rapidly evaluate the performance of an application on a very large set of hardware configurations, which is essential for a comprehensive design space exploration of both software and hardware. For the automatic evaluation of the resource efficiency of the hardware we use an automated RTL based design flow starting from the HDL description up to the final ASIC implementation. The choice of the tools and of the target libraries is based on the *"Environment Modules Project"* on Linux systems [2].

The developer can specify a custom selection of modules to adapt the environment to his needs. These modules include, for example, the configuration of license servers, target technologies with specific pre-defined corner set, or EDA front end and back end tools. The modules are hierarchical structured to easily maintain the setup, simplify consecutive use of different versions of design tools or target libraries, and control the availability of specific tools to different groups in a company (e.g. due to license policies). Default versions can be specified to guarantee the user to profit from software updates and bug fixes. Some tools may require others to be fully functional, or do not work in some combinations. The module environments allow the specification of dependencies and conflicts to minimize misconfiguration. For example, synthesis tools require the availability of a target technology and different versions of the same application must not be used simultaneously.

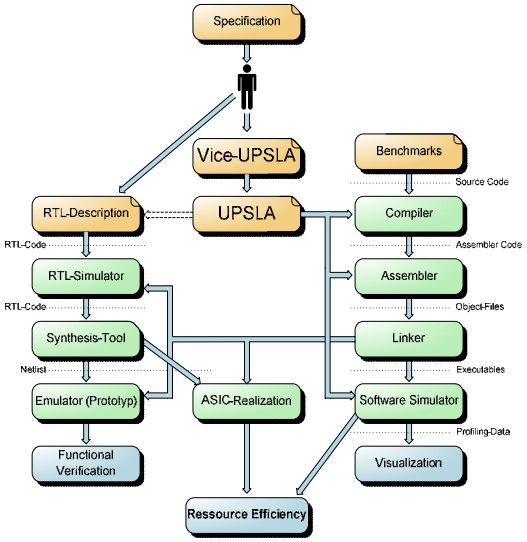

Figure 1: Dual design flow based on a reference processor description

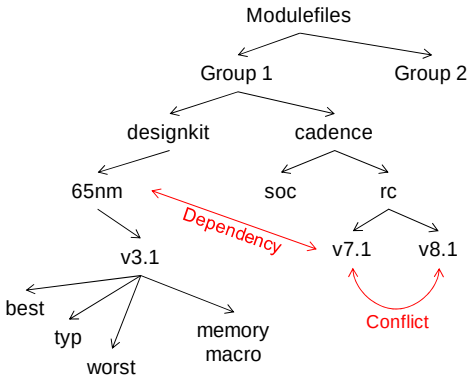

Figure 2: Hierarchical module environment to adapt the users design environment

Figure 2 shows an example of a simple module hierarchy. The modules, provided with our design flow include custom made tools based on the TCL language, to perform, inter alia, automatic code quality checks, RTL synthesis, power analysis, place and route or simulation/verification steps. These tools provide reference procedures and properties which can be extended by custom constraints. This allows also developers, which are no backend engineers, to easily obtain comparable results about the quality of their design up to highest level of detail even in very early design stages. For the evaluation of different target technologies, the modules can define different process corners, e.g., supply voltages or temperature. Using custom properties it is even possible to extend the reference flow for the implementation of final ASIC implementations to be manufactured.

Automated Hardware Tool Flow

Figure 3 depicts an overview of the tools used in our holistic design flow for the design of digital circuits starting from the initial RTL description up to the final ASIC implementation. All tools are based on defined interfaces and allow an automatic design flow from scratch to enable fast design space explorations with only few interactions required by the engineer. Not all steps are mandatory, for example formal verification

does not always have to be performed in intermediate design steps with only slight changes in the architecture. The outputs of the tools can be evaluated by automated tools to quickly review the results.

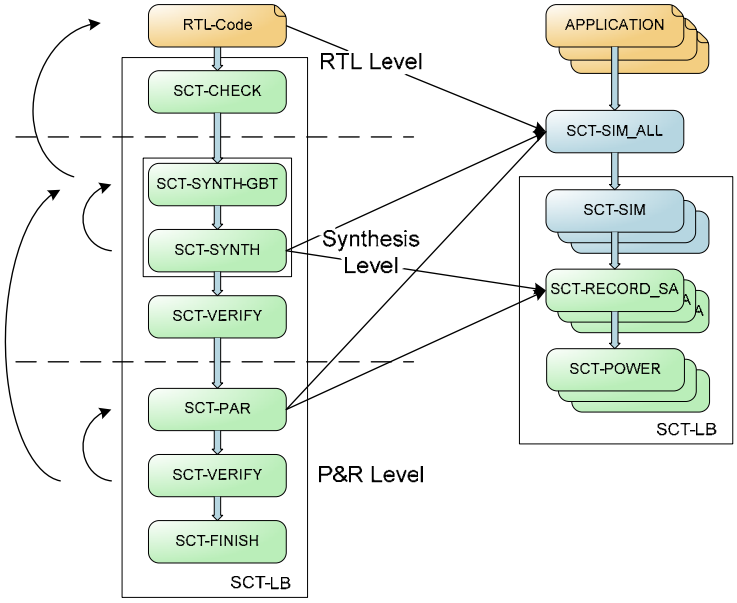

Figure 3: Automated hardware design flow

Starting from the RTL description, *sct-check* (based on Cadence Conformal®) allows an early evaluation of the quality of the code, e.g. FSM checks for unreachable states, unintentional latches, and unused signals. On synthesis level *sct-synth* performs the mapping (Cadence® RTL compiler) of the RTL design to a standard cell library specified by a target technology module (cf. Figure 4). For synthesis, the RTL sources, period and/or I/O-constraints can be specified. The synthesis flow integrates the common design steps as RTL code analysis, elaboration, compilation and reporting. After compilation, timing constraints are reviewed. In case of timing violations, the timing is automatically relaxed and an incremental synthesis is performed. This leads to more accurate power and area reports related to a design with length-adjusted paths. However reports for both steps are generated to allow the analysis of the critical paths. In addition, a slightly over-constrained timing may lead to better results in terms of maximum clock frequency. s*ct-synth* supports the Cadence® PLE(physical layout estimation) and CPF(common power format) flows. Usually, in very early design steps the maximum frequency is not known. *sct-synth-gbt* (gbt=get-best-timing) provides an iterative procedure for the automatic derivation of the best timing. Starting from initial timing constraints, *sct-synth-gbt* performs multiple syntheses by executing *sct-synth* and modifying the constraints after each iteration step, until the worst negative slack is minimized. A threshold can be specified to allow over-constraining of the timing. The module environment allows the transparent usage of *sct-synth* with tools of different vendors. The output of *sct-synth* (i.e., netlist, constraints) can directly be passed to the *sct-par* tool for backend place and route (Cadence® SoC Encounter™). Figure 4 shows the design flow of the place and route. It encompasses floorplan creation/loading, placement, clock tree synthesis, power routing, clock routing, detailed routing, DRC(design rule check)/SI(signal integrity) fixes and pattern fill. All intermediate steps are saved and can automatically be restored, to reduce run-time of the design iterations. Place and route can be performed in two modes: floorplanless mode and floorplan

mode. For early estimations of the resource consumption of a design and for implementing IP blocks in a hierarchical approach, the floorplanless mode is used. A floorplan is generated automatically (including I/O placement, power plan, …) depending on a given target utilization. Starting from this design step, be loaded, modified and then re-used in the floorplan mode. To maximize the quality of the results, an effort of the number of design optimization cycles can be specified. By user-scripts, custom constraints and design flow extensions can be used to have full access to all tool-specific features (e.g., well tap cell insertion, custom power routing, metal fill, etc.). *sct-par* supports CPF (common power format) for, e.g., multi voltage designs. The intermediate netlists (RTL/synthesis level/P&R level) can be automatically verified formally (*sct-verify*, Cadence Conformal®) and functionally using simulation. For the functional verification of processor cores, we apply a validation by simulation method proposed in [3]. Large sets of test cases are executed on the hardware, processor state traces are captured and compared to the execution of the automatically generated instruction set simulator (*sct_sim_all, sct-sim*). The tool *sct-record_sa* allows the automated capturing of switching activity during the simulation, which can be used for an accurate estimation of the power consumption at each level of hardware detail (*sct-power*).

Considering a very large design space and iterating through all described steps in the design flow, very high processing power is required. Many steps (e.g. syntheses for different process corners, simulation of large sets of test cases, …) can be executed consecutively. For an efficient load balancing of the jobs on a large number of EDA hardware systems, *sct-lb* can be used. It implements a simple, but efficient method to equalize the load on the used machines. When issuing a job with *sct-lb*, the tool checks the current load on all machines and selects that with the lowest load. The processing power of the machines is taken into account. An overload of the hardware is prevented by postponing new jobs during very high load. *sct-lb* is not limited to certain applications but can be used with any command.

A Case Study: Exploration of the design space of a configurable VLIW processor

To prove the usability of this approach it was used for the exploration of a very large design space of the CoreVA VLIW architecture [4]. The RTL description of the CoreVA architecture is modular to specify, e.g., the number of VLIW slots, ALUs, MAC units, DIV units, or LD/ST units. The configuration of the number of consecutive functional units does not only affect the runtime in terms of clock cycles (dependent on the portion of inter-level parallelism extracted from the code by the VLIW compiler), but also critical path, area and power consumption of the hardware. This leads to a very large design space of more than 300 architecture variants to be evaluated. Considering five target technologies, about 1500 design iterations had to be analyzed. Using the proposed automated design flow (combining automatic generation of the tool chain and the automated hardware design), the evaluation of these design variants were only limited by the computational power of our synthesis hardware and was performed in less than one week. In addition the tool flow presented in this work was practically used for the design of several ASIC tape outs (e.g. the realization of the VLIW processor in a 65nm low power standard cell technology and a 32-bit RISC core in a custom designed ultra low power standard cell library).

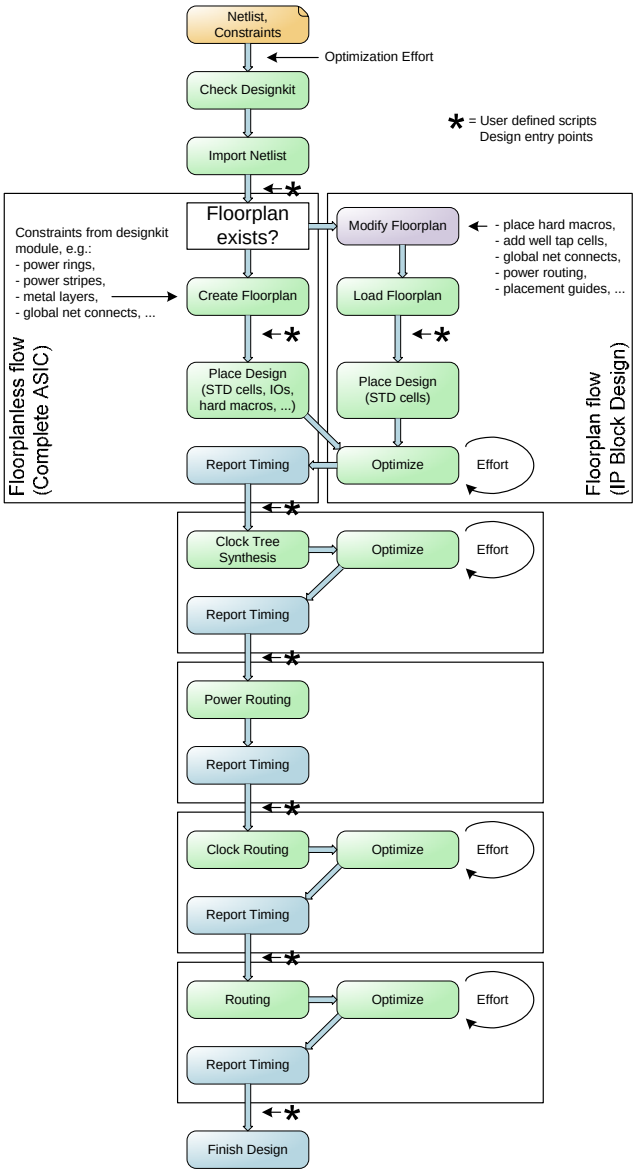

Figure 4: Automated synthesis and P&R design flow

[1] Uwe Kastens, Dinh Khoi Le, Adrian Slowik, and Michael Thies. **Feedback driven instruction-set extension.** *Proceedings of ACM SIGPLAN/SIGBED 2004 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'04)*, Washington, D.C., USA, June 2004.

[2] John L. Furlani, **Modules: Providing a Flexible User Environment**, *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)*, pp. 141-152, San Diego, CA, September 30 - October 3, 1991.

[3] Dreesen, Ralf; Jungeblut, Thorsten; Thies, Michael; Porrmann, Mario; Rückert, Ulrich; Kastens, Uwe: **A Synchronization Method for Register Traces of Pipelined Processors.** *Proceedings of the International Embedded Systems Symposium 2009 (IESS '09)*, S. 207-217, Schloss Langenargen, Germany, 14. - 16. Sep. 2009.

[4] Thorsten Jungeblut, Ralf Dreesen, Mario Porrmann, Ulrich Rückert, and Ulrich Hachmann. **Design Space Exploration for Resource Efficient VLIW-Processors,** *University Booth of the Design, Automation and Test in Europe (DATE) conference*, Munich. 2008.