Automatic Design Techniques for Embedded Systems

M. Holzer, P. Belanović, B. Knerr, and M. Rupp

Vienna University of Technology
Institute for Communications and RF Engineering
Gusshausstr. 25/389, 1040 Vienna, Austria

Abstract

The complexity of modern communication systems, particularly in the wireless domain, grows at an astounding rate, which causes increasing time-to-market. In order to provide a shorter time-to-market, mostly manually performed design tasks are to be overcome and an automation of the design process is needed. This paper presents approaches for automated design steps starting with high level algorithmic descriptions. An environment for automatic floating-point to fixed-point conversion is presented. Based on the generation of evaluation metrics, automatic HW/SW partitioning is performed. Furthermore, the integration of those tools as part of a consistent development environment is described.

1 Introduction

Complexity of modern communication systems, particularly in the wireless domain, grows at an astounding rate. This rate is so high that the demand of algorithmic complexity now significantly outpaces the growth in available complexity of underlying silicon implementations, which proceeds according to the famous Moore's Law [1], i.e., available complexity doubles every 16 to 18 months. Furthermore, algorithmic complexity even more rapidly outpaces design productivity, expressed as the average number of transistors designed per staff/month [2]. This problem first introduced as *design productivity crisis* in 1999 by Sematech (www.sematech.org) is nowadays well known under the name the productivity gap or *design gap*.

A major problem causing design delays is the fragmentation of the design process. Due to the large scope and extremely heterogeneous nature of modern wireless communication designs, their development suffers from incompatible system descriptions. Along this design path many descriptions are required, rewritten and reformulated mostly manually by corresponding experts converting them into other, more suitable description forms. A major problem of so many conversions are serious communication obstacles between design teams due to different approaches

This work has been funded by the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms.

and languages. Also inconsistent verification, lack of design tools for supporting necessary design steps, and difficulties in the discovery and fixing of errors are evident.

These problems arise from the *fragmentation of the design process* into domain-specific efforts. A formal methodology to keep the design integrated and progress consistent through the development process is missing. In [3] such an integrated environment named POLIS has been reported for typical control-oriented, reactive systems based on high level languages supported by a PTOLEMY [4] environment and based on extended finite state machines (EFSM).

Although translational tools may convert one design description into another, information may get lost by this process. A single repository or single system description (SSD) keeping all the information in a data base can prevent such problem. The translational tools thus need to be linked to such SSD in order to preserve all relevant information independent of the required transformation. An SSD can guarantee the consistency in the design which is especially important when a bug appears at a different design level than it was produced.

In Section 2 the framework of an integrated development environment is presented. The problem of floating-point to fixed-point conversion is addressed in Section 3. Furthermore, Section 4 presents analysis methods of algorithms which build the basis for metrics used in Section 5 for automated HW/SW partitioning.

2 Integrated Design Methodology

An integrated design methodology is required to work with existing EDA tools. Since the information required for various tools is never identical, a consistent design methodology needs to utilise a so called Single System Description (SSD), containing all design information. An implementation of an SSD in the form of an Design Database (DDB) is presented in detail in [5]. This database representation is not bound to specific language constraints and thus offers great flexibility in capturing the miscellaneous aspects of a design. Additional advantages of the DDB approach are fast access, data security by the capability to grant permissions to the developers, a high popularity as well as compatibility with major DataBase Management Systems (DBMS) from Microsoft, IBM, Oracle and the open source DBMS MySQL.

A framework reflecting these obligations is shown in Figure 1a. It depicts the SSD surrounded by the required tools each with dedicated interfaces to incorporate the various EDA tools and stays open for incorporating other tools as the empty tool box in the figure indicates. The different design teams provide inputs, such as desired system behaviour and structure, constraints, and tool options. Also, the designers receive outputs, like status of the system description, results of simulations, estimates of hardware costs, timing and similar information. Typically, the outputs of the database are handed to the tools which presents them in form of their GUIs to the designer. Some of the tools supported by the SSD are commercially available, favoured by the various design teams, while others are specially written to perform tasks, usually performed manually by designers in the past like for example HW/SW partitioning, floating-point to fixed-point conversion, as will be explained in the following sections. Still manual interaction with the designer is provided by a database modification tool, simply allowing the designer to enter manually derived values. The database is thus enriched and the system description is refined on its way to implementation. The database system does not require a specific order of which various tools need to be performed. For example, some designers prefer to perform floating-point to fixed-point conversion after the

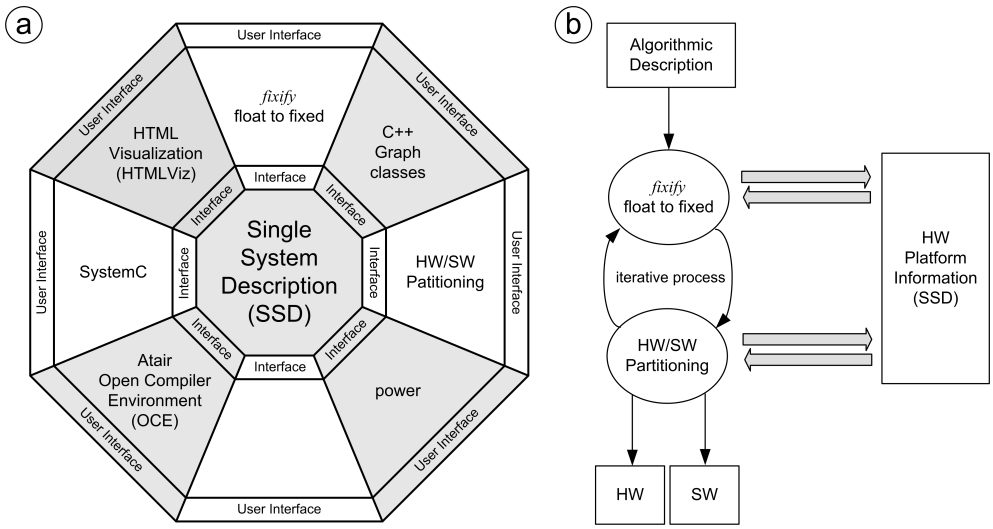

Figure 1. *a) Interaction of the Single System Description with different design tools via interfaces providing a consistent design flow. b) Iterative optimization process utilizing floating-point to fixed-point conversion and HW/SW partitioning.*

HW/SW partitioning. As long as the succeeding tool is provided with sufficient information, it can be started. Such an open environment has not only the advantage that new commercial tools can be incorporated but it also provides a realistic platform to investigate the performance of new research tools.

Figure 1b depicts a design flow example utilizing the tools for floating-point to fixed-point conversion and HW/SW partitioning. A high level C-code is provided to describe the DSP algorithm. Examples in GenericC for COSSAP are presented in [6]. In an iterative process bit-width optimization and HW/SW partitioning can be performed on this high level algorithmic description until the design requirements are fulfilled. Both tools rely on information of the HW platform like available bit-widths and communication structures, which are stored inside the SSD. Such platform individual architectural information helps restricting the huge search space. Also optimization results are written back to the SSD, enriching the database with more detailed architectural information that can be used in the next iteration cycle. The final outcome of this iterative process is an optimal decision of which part is more suitable for software and which part is more suitable for a hardware implementation taking all communication overhead into account. This process can also deliver design parameters of a parametric DSP platform, for example an embedded system consisting of one or more DSPs and several HW accelerators (see Figure 4 further ahead).

3 Floating-point to Fixed-point Conversion

Design flows of communication systems typically start at the algorithmic level, where the system model contains no implementation details and the attention of the designer is focused on the functioning of the algorithm itself. At this abstraction level, all numeric formats in the system model are floating-point formats, because they allow the designer to modify the algorithm, with-

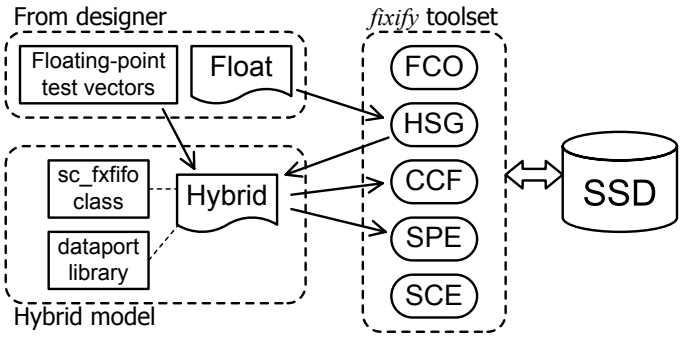

Figure 2. *Structure of the fixify environment*

out having to pay attention to the numeric effects such as overflow or quantization error. Also, floating-point numeric formats are natively supported on the computing platforms such as PCs and workstations, where the designer typically performs algorithmic modeling.

However, at the end of the design flow lies the implementation level, where the system model contains much specific detail on the actual hardware and/or software implementation of the system. At this abstraction level fixed-point numeric formats are used almost exclusively, because they offer significant savings in power consumption and silicon area cost of the final implementation, as well as superior throughput and latency performance.

Hence, conversion from floating-point to fixed-point numeric formats is a necessary step in the design flow. This task is traditionally performed by the designer manually. The designer considers ranges and precision of each data channel in the design and the overflow and quantization errors in each, introduced by use of fixed-point formats. Thus, a trade-off between numeric performance and implementation cost exists and is optimized by the designer by finding the lowest implementation cost which gives an acceptable level of numeric performance.

Several approaches to automating this conversion process have been proposed. In general, all of these can be classified into either analytical (or static) or statistical (or dynamic) approaches. Analytical, or static, methods [7] use code analyses techniques to propagate bitwidth requirements of data channels through the control and dataflow graph representations of the system model. These approaches have the advantages of short optimization runtimes and no dependency on input data, since no system simulations are required. However, they have a number of serious drawbacks, including highly conservative optimization results compared to manual optimization by the designer as well as no consideration of the performance/cost trade-off.

On the other hand, dynamic methods [8, 9] make use of automated system simulations, thus exploring directly the performance/cost trade-off in the same way the designer does manually. The obvious drawbacks of these approaches are the long runtimes required to performs numerous system simulations and the results of the optimization being influenced at least to some extent by the test patterns used during simulation.

In this work, we present an environment called *fixify*, specifically targeted towards automatically converting a system model from floating-point to appropriate fixed-point formats. The *fixify* environment uses system simulations (i.e. employs a dynamic conversion method) and is based around the SSD as shown in Figure 2.

The Hybrid System Generator (HSG) tool is responsible for generating the hybrid description

of the system from the pure floating-point one supplied by the designer. The hybrid system description used in the *fixify* environment is capable of simulating the system description in any combination of floating and fixed-point formats. Furthermore, the hybrid description is compiled just once by the HSG tool and is capable of changing the numeric format of any data channel in the system at run time. This control over the hybrid description is exercised by the other tools in the *fixify* environment through environment variables.

Each conversion process starts from the so-called corner case, or the set of minimal fixed-point formats which for the system in question deliver numeric performance equal to floating-point formats. Hence at this corner case, no numeric degradation is experienced, and the cost of this system implementation is thus maximal. All other fixed-point configurations which will be considered during the optimization will have both lower implementation cost and worse numeric performance. The Corner Case Finder (CCF) tool is responsible for finding the corner case for the design under consideration and thus enabling the beginning of the optimization process.

Since the optimization process involves a trade-off between system implementation cost and its numeric performance, the *fixify* environment includes estimation tools for both. The System Performance Estimator (SPE) tool evaluates the numeric performance of the system in terms of the overflow and quantization error experienced under the particular fixed-point configuration. The System Cost Estimator (SCE) tool provides an estimate of the implementation cost of the system.

Both of these estimates are used in the automatic trade-off optimization, performed by the Fixed-point Configuration Optimizer (FCO) tool. Since the FCO tool is a generalized framework for implementing optimization algorithms, the *fixify* environment offers to the designer a palette of optimization techniques, including restricted-set full search, greedy search and branch-and- bound algorithms.

It has been shown [10] that each of these algorithms is capable of tackling the conversion process automatically, but also that each best suits a different implementation scenario, making the *fixify* environment highly versatile. The restricted-set full search optimization is highly suited to processing algorithms which will be implemented on DSP platforms, being able to guarantee optimal mapping onto the native set of fixed-point formats for the given DSP architecture (e.g. $\{16, 32, 40\}$ on the TI TMS320C62x architecture). Branch-and-bound and greedy algorithms optimize the design in the continuous search space and are hence more suitable to system implementations in custom hardware. For such systems, the branch-and-bound optimization offers guaranteed optimal results. However, for systems with very high complexity, where optimization runtimes with the branch-and-bound algorithm may be impracticably high, greedy search optimization offers ten-fold reduction of runtime, at the expense of 4% to 6% lower quality of results.

4 Static Code Analysis

During the design process of a digital system it is of paramount importance to assure quality of the written code and to base design decisions on reliable characteristics. Those characteristics of the code usually called metrics (for example: lines of code, longest path) can be used to find early estimates of the properties of the final implementation. Up to now many design decisions are taken on the basis of the experience of the designers and not based on well defined metrics. Collecting such metrics can help to produce more reliable and traceable decisions. Tasks like partitioning of a system into hardware and software, static analysis of bit widths, and mapping of algorithmic

descriptions onto architectures are prominent candidates for the usage of metrics.

Evaluating the effect of a decision by a complete run through all the different design tools is extremely time consuming and cannot be used for high level design decisions. Also simulation based approaches heavily depend on the used test vectors and are rather time consuming. Whereas static analysis of the structure of an algorithm achieves the needed performance in order to be applied for large projects.

The analysis of the algorithm is based on the decomposition of a written code into its basic graphs. In general an algorithm in form of sequential code can be decomposed into its control flow graph (CFG), built up of interconnected basic blocks (BB). Each basic block contains a sequence of data operations ended by a control flow statement as last instruction. This sequence of data operations forms itself an expression tree. Figure 3 shows an example of a function and its graphical descriptions.

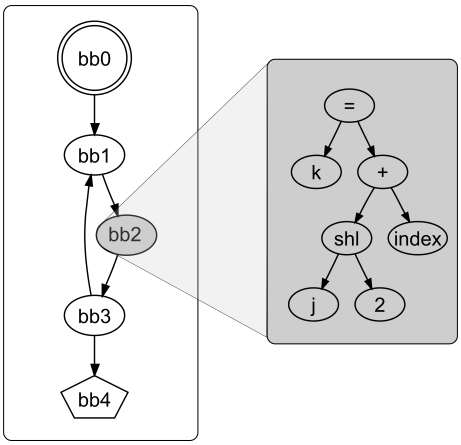

Figure 3. *Control Flow Graph (CFG) and the corresponding expression tree of the basic block bb2.*

For the design of a signal processing system, which consists of hardware and software many different programming languages have been introduced like VHDL, Verilog, C/C++ up to one of the last major contributions SystemC (www.systemc.org). The analysis of the source code and the generation of the above mentioned graphs and the underlaying data structure is embedded into the SSD as described in Section 1. The tool chain allows for reading in a algorithmic description in SystemC and to store its graph representation into the DDB. A couple of further analysis tools uses those representations in order to derive metrics for the needed cycle count of a probable hardware or software implementation. Additionally estimation of the needed gate count are computed. Those informations are stored in the DDB in property tables, which can be accessed by other optimisation tools.

For estimating the gate count of a hardware implementation, first approaches started by simple counting the number of all operations inside a function [11] but neglecting the distribution of the operators over the algorithm or the longest path. Other techniques like [12] analyse the descriptions on register transfer level (RTL) by investigating the graph of boolean networks (BN). For those graphs, structural metrics like number of nodes, edges, fan-in, and fan-out are derived. The

application of this approach is limited to pure data flow representations.

The high-level metric for the gate count is built up by the operational units (ADD, MAC, etc.), and the registers, which store the intermediate results of the operations. For the number and size of the required operational units, the longest path with respect to every single type of operation is investigated. In order to get the HW effort for the required registers the number of edges between operations in the expression trees are taken into account.

For the cycle count of the algorithm implemented in HW the depth of the expression trees associated to every BB is measured. The depth serves as another weight variable, and a longest path considering this weight can be determined. The result is the maximum cycle count for a path in the selected process.

In [13] for the cycle count of a software implementation, of the algorithm on a processor the MIPS value is applied. As only a relative value for the comparison to the cycle count of the hardware realisation is needed, only cycle counts for the SW realisation instead of real time units is considered. In comparison to the HW cycle count not the depth of the expression trees for every BB, but the total number of instructions within the BB is used. In other words there is no exploitation of parallelism within a BB. Using this weight a longest path is computed resulting in a metric for the SW cycle count.

5 HW/SW Partitioning

Optimal HW/SW Co-design has been a research effort over many years. Heuristics like simulated annealing (SA) used in the COSYMA system [14], genetic algorithm (GA), and tabu search are usually used to tackle the NP complete problem of HW/SW partitioning. A comparison of those algorithms is given in [15]. A decade ago partitioning heuristics arose e.g minimising hardware size but not execution time by Vahid [16], or considering both but excluding the degree of freedom brought in by the used communication model (bus system) by Kalavade [17]. Chatha and Vemuri published an approach combining task partitioning and scheduling on a fixed architecture [18]. Even evolutionary algorithms are applied to that problem, which is shown in [19]. Despite of those efforts, such solutions are not being found in commercial EDA tools. The reason for this seems to be that their assumptions are not being sufficiently practical, i.e., the required information cannot easily be extracted from the high-level design. Typically, a complete metric for a HW and SW solution is required a-priori in order to decide how to optimally partition the design. However, such knowledge is not known before-hand. The implementation effort in 3G wireless is so high that an a-priori realisation of the system would be too expensive.

In general the functionality of a system on high level is described with a set of interconnected functional objects, often represented as a block based model (MatLab/SimuLink, COSSAP, Co-Centric System Studio). Such functional objects can be implemented in dedicated hardware (microcontroller, processor, FIR, etc.) or software running on pre-defined programmable hardware. Software development is less expensive, more flexible and faster designable than an equivalent hardware implementation. However, hardware can offer better performance.

The decision of whether to implement a functional object in hardware or in software, the partitioning, is currently done manually by experienced designers. The goal is to provide the designer with accurate information about the internal characteristics of his system and to generate automatically partitioning solutions satisfying all the constraints. The challenge lies in the combination of

the development of accurate metrics on the highest abstraction level, the definition of appropriate cost functions, an optimal heuristic to move through the partitioning search space and a model of the underlying hardware platform, which is general enough to cover different system designs and specific enough to irradiate the area, communication and processing time hotspots of the given design.

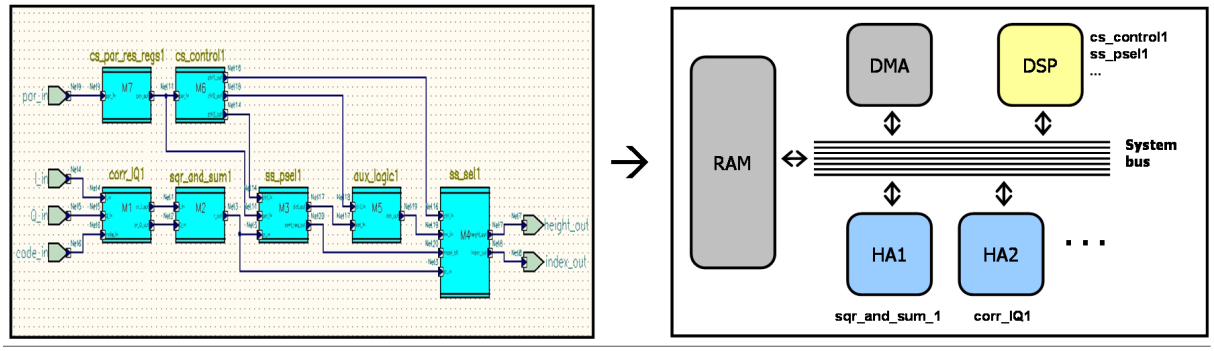

Figure 4. *HW/SW Partitioning: Mapping a block based algorithmic description onto a general target architecture.*

Figure 4 depicts a screenshot of a typical block based algorithmic description in a commercial EDA Tool (CoCentric System Studio from Synopsys) on the left side. In particular a simplified modular design is assumed consisting of interconnected components which contain at least one sequential process. Each of these processes can be written in C or some similar procedural language. The processes communicate via the connections between the components.

The general target architecture, on which the blocks' processes are to be mapped, completes the figure on the right side. The architecture is composed of a standard processor with memory (the software part) and several hardware accelerators (HA1, HA2, ...) possibly with their own memory (the hardware part). The standard processor may use a system bus, and/or may have several bidirectional data ports (such as commonly found on microcontrollers). The hardware accelerators can access the software memory, as well as its own local memory. Our problem is to assign every piece of the program to either the software or hardware part, such that the execution time is minimised while satisfying any size and I/O constraints.

Therefore the HW/SW partitioning process needs a sophisticated format of the system description, which will be provided by the SSD framework. Static Code Analysis (see Section 4) delivers the estimates for area, software and hardware execution times, and later on for power consumption for every process or every code snippet respectively. A modified version of the min-cut heuristic of Kernighan and Lin [20] has been implemented and applied to partition a industry-designed UMTS baseband processing unit based on the estimates of the static code analysis and the target architecture mentioned above [21]. It was possible to capture the constraints and design parameters within the cost functions in such a way that the heuristic reached the same partitioning result as it has been done by the design group manually, however in matters of seconds.

Hence the potential of the SSD framework to solve another thronging issue within the design flow has been established. However the current work concentrates on accuracy improvements

within the computation model of the target architecture. More precisely the integration of different system buses and thus different interprocess communication times is currently under investigation. Further research activities comprise the extension to multi-core systems (i.e. more than one standard processor for the software part) and, with respect to multirate systems represented as synchronous data flow graphs (SDFG), an integrated optimisation engine for the process scheduling on the target architecture that accompanies the partitioning heuristic.

6 Conclusions

This paper presents an integrated design environment in form of an SSD, which is capable of integrating existing EDA tools as well as special optimization tools. With the increasing complexity of systems the need for automating tasks, up to now performed manually by an experienced designer, raises. The task of floating-point to fixed-point conversion is achieved by the *fixify* environment. Also a way of automatically partitioning a system into HW and SW components based on metrics found by static code analysis has been shown.

References

[1] G. Moore, "Cramming more components onto integrated circuits," *Electronics Magazine*, vol. 38 (8), pp. 114–117, April 1965.

[2] R. Subramanian, "Shannon vs. Moore: Driving the Evolution of Signal Processing Platforms in Wireless Communications," in *IEEE Workshop on Signal Processing Systems SIPS'02*, Oct. 2002.

[3] F. Balarin, E. Sentovich, M. Chiodo, P. Giusto, H. Hsieh, B. Tabbara, A. Jurecska, L. Lavagno, C. Passerone, K. Suzuki, and A. Sangiovanni-Vincentelli, *Hardware-Software Co-design of Embedded Systems: The POLIS approach*. Kluwer Academic Publishers, 1997.

[4] J. Buck, S. Ha, E. Lee, and D. Masserschmitt, "Ptolemy: a framework for simulating and prototyping heterogeneous systems," *International Journal of Computer Simulation, special issue on Simulation Software Development*, vol. 4, pp. 155–182, April 1994.

[5] P. Belanović, M. Holzer, D. Mičušík, and M. Rupp, "Design Methodology of Signal Processing Algorithms in Wireless Systems," in *International Conference on Computer, Communication and Control Technologies CCCT'03*, pp. 288–291, July 2003.

[6] P. Belanović, M. Holzer, B. Knerr, M. Rupp, and G. Sauzon, "Automatic Generation of Virtual Prototypes," in *International Workshop on Rapid System Prototyping*, (Geneva, Switzerland), pp. 114–118, June 2004.

[7] H. Keding, M. Willems, M. Coors, and H. Meyr, "FRIDGE: A Fixed-Point Design and Simulation Environment," in *Design, Automation and Test In Europe DATE'98*, February 1998.

[8] S. Kim, K. Kum, and W. Sung, "Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs," *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 45, pp. 1455–1464, November 1998.

[9] C. Shi and R. W. Brodersen, "An Automated Floating-point to Fixed-point Conversion Methodology," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 529–532, April 2003.

[10] P. Belanović and M. Rupp, "Automated Floating-point to Fixed-point Conversion with the *fixify* Environment," in *International Workshop on Rapid System Prototyping RSP'05*, June 2005.

[11] J. Rabaey and M. Potkonjak, "Estimating Implementation Bounds for Real Time DSP Application Specific Circuits," in *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13 of *6*, pp. 669–683, June 1994.

[12] K. Büyüksahin and F. Najm, "High-Level Area Estimation," in *ISLPED'02*, pp. 271–274, August 2002.

[13] J. D'Ambrosio and X. S. Hu, "Configuration-level hardware/software partitioning for real-time embedded systems," in *Proceedings of the 3rd international workshop on Hardware/software co-design*, pp. 34–41, 1994.

[14] R. Ernst, J. Henkel, and T. Benner, "Hardware-software Cosynthesis for Microcontrollers," *IEEE Design & Test*, vol. 12, pp. 64–75, 1993.

[15] T. Wiangtong, P. Cheung, and W. Luk, "Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware-Software Codesign," *Design Automation for Embedded Systems*, vol. 6, pp. 425–449, 2002.

[16] F. Vahid, J. Gong, and D. Gajski, "A Binary-Constraint Search Algorithm for Minimizing Hardware during Hardware/Software Partitioning," in *European Design Automation Conference (EURO-DAC)*, pp. 214–219, 1994.

[17] A. Kalavade and E. Lee, "A Global Critically/Local Phase Driven Algorithm for the Constrained Hardware/Software Partitioning Problem," in *Third International Workshop on Rapid Systems Prototyping*, pp. 42–48, 1994.

[18] K. Chatha and R. Vemuri, "An Iterative Algorithm for Hardware-Software Partitioning, Hardware Design Space Exploration and Scheduling," *Design Automation for Embedded Systems Journal*, no. 5, pp. 281–293, 2000.

[19] T. Blickle, J. Teich, and L. Thiele, "System-Level Synthesis Using Evolutionary Algorithms," *Design Automation for Embedded Systems*, vol. 1, pp. 1–40, 1998.

[20] B. Kernighan and S. Lin, "An efficient heuristic procedure in partitioning graphs," *Bell System Technical Journal*, February 1970.

[21] B. Knerr and M. Holzer and M. Rupp, "HW/SW Partitioning Using High Level Metrics," in *International Conference on Computing, Communications and Control Technologies (CCCT)*, pp. 33–38, June 2004.