Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 64913, Pages 1–18
DOI 10.1155/ES/2006/64913

Efficient Design Methods for Embedded Communication Systems

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

Institute for Communications and Radio Frequency Engineering, Vienna University of Technology, Gußhausstraße 25/389, 1040 Vienna, Austria

Received 1 December 2005; Revised 11 April 2006; Accepted 24 April 2006

Nowadays, design of embedded systems is confronted with complex signal processing algorithms and a multitude of computational intensive multimedia applications, while time to product launch has been extremely reduced. Especially in the wireless domain, those challenges are stacked with tough requirements on power consumption and chip size. Unfortunately, design productivity did not undergo a similar progression, and therefore fails to cope with the heterogeneity of modern architectures. Electronic design automation tools exhibit deep gaps in the design flow like high-level characterization of algorithms, floating-point to fixed-point conversion, hardware/software partitioning, and virtual prototyping. This tutorial paper surveys several promising approaches to solve the widespread design problems in this field. An overview over consistent design methodologies that establish a framework for connecting the different design tasks is given. This is followed by a discussion of solutions for the integrated automation of specific design tasks.

1. INTRODUCTION

Over the past 25 years, the field of wireless communications has experienced a rampant growth, in both popularity and complexity. It is expected that the global number of mobile subscribers will reach more than three billion in the year 2008 [1]. Also, the complexity of the modern communication systems is growing so rapidly, that the next generation of mobile devices for 3G UMTS systems is expected to be based on processors containing more than 40 million transistors [2]. Hence, during this relatively short period of time, a staggering increase in complexity of more than six orders of magnitude has taken place [3].

In comparison to this extremely fast-paced growth in algorithmic complexity, the concurrent increase in the complexity of silicon-integrated circuits proceeds according to the well-known Moore law [4], famously predicting the doubling of the number of transistors integrated onto a single integrated circuit every 18 months. Hence, it can be concluded that the growth in silicon complexity lags behind the extreme growth in the algorithmic complexity of wireless communication systems. This is also known as the *algorithmic complexity gap*.

At the same time, the International Technology Roadmap for Semiconductors [5] reported a growth in design productivity, expressed in terms of designed transistors per staff-month, of approximately 21% compounded annual growth rate (CAGR), which lags behind the growth in silicon complexity. This is known as the *design gap* or *productivity gap*.

The existence of both the algorithmic and the productivity gaps points to inefficiencies in the design process. At various stages in the process, these inefficiencies form bottlenecks, impeding increased productivity which is needed to keep up with the mentioned algorithmic demand.

In order to clearly identify these bottlenecks in the design process, we classify them into *internal* and *external* barriers.

Many potential barriers to design productivity arise from the design teams themselves, their organisation, and interaction. The traditional team structure [6] consists of the *research* (or algorithmic), the *architectural*, and the *implementation* teams. Hence, it is clear that the efficiency of the design process, in terms of both time and cost, depends not only on the forward communication structures between teams, but also on the feedback structures (i.e., bug reporting) in the design process. Furthermore, the design teams use separate system descriptions. Additionally, these descriptions are very likely written in different design languages.

In addition to these internal barriers, there exist several external factors which negatively affect the efficiency of the design process. Firstly, the work of separate design teams is

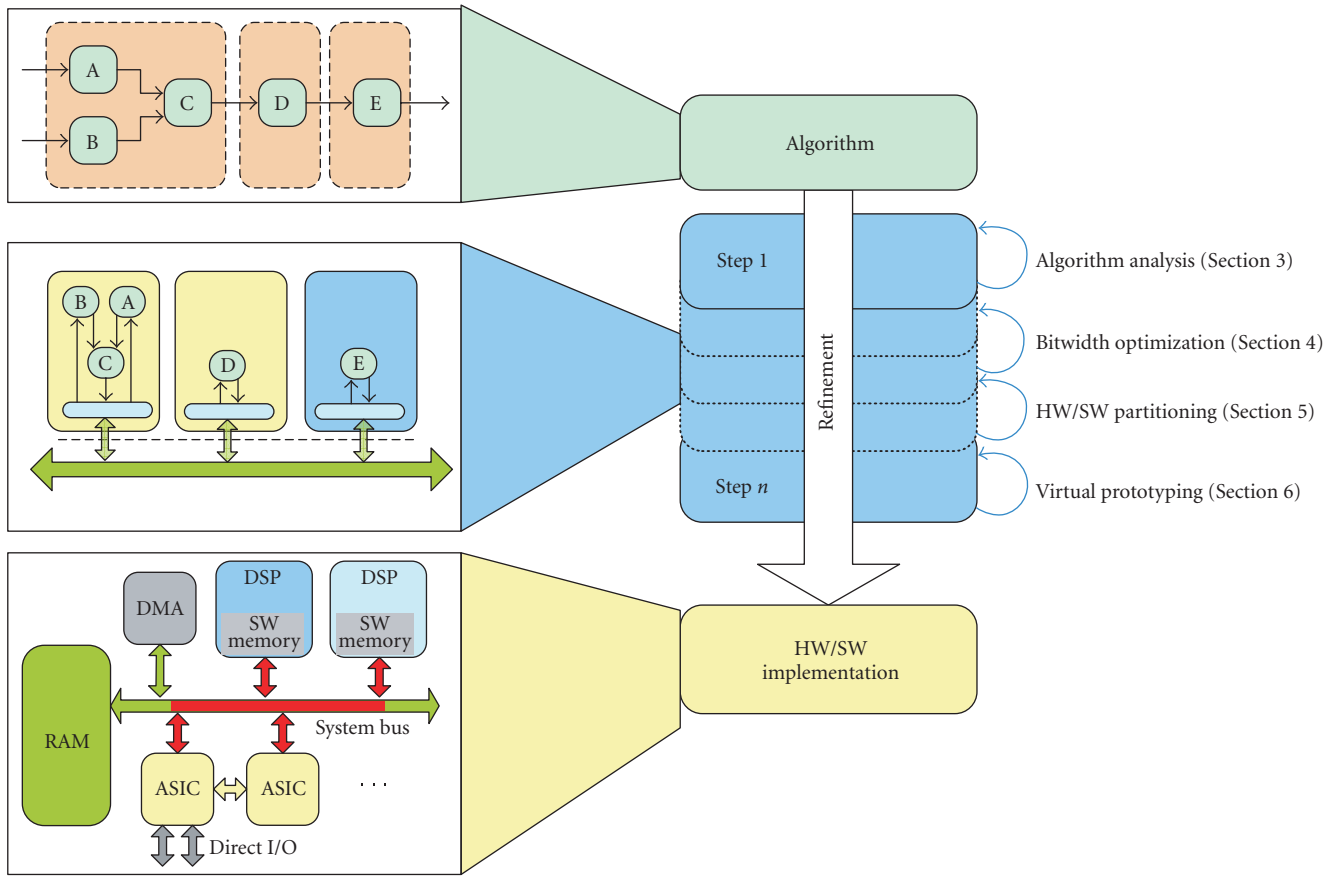


Figure 1: Design flow with several automated design steps.

supported by a wide array of different EDA software tools. Thus, each team uses a completely separate set of tools to any other team in the design process. Moreover, these tools are almost always incompatible, preventing any direct and/or automated cooperation between teams.

Also, EDA tool support exhibits several "gaps," that is, parts of the design process which are critical, yet for which no automated tools are available. Although they have high impact on the rest of the design process, these steps typically have to be performed manually, due to their relatively large complexity, thus requiring designer intervention and effort. Designers typically leverage their previous experience to a large extent when dealing with these complex issues.

In Figure 1 a design flow is shown, which identifies several intermediate design steps (abstraction levels) that have to be covered during the refinement process. This starts with an algorithm that is described and verified, for example, in a graphical environment with SystemC [7]. Usually in the wireless domain algorithms are described by a synchronous data flow graph (SDFG), where functions (A, B, C, D, E) communicate with fixed data rates to each other. An intermediate design step is shown, where already hardware/software partitioning has been accomplished, but the high abstraction of the signal processing functions is still preserved. Finally the algorithm is implemented utilising a heterogenous architecture that consists of processing elements (DSPs, ASICs), memory, and a bus system.

Also some design tasks are mentioned, which promise high potential for decreasing design time by its automation. This paper discusses the requirements and solutions for an integrated design methodology in Section 2. Section 3 reports on high-level characterisation techniques in order to have early estimations of the final system properties and allows to make first design decisions. Section 4 presents environments for the conversion of data from floating-point to fixed-point representation. Approaches for automated hardware/software partitioning are shown in Section 5. The decrease of design time by virtual prototyping is presented in Section 6. Finally, conclusions end the paper.

2. CONSISTENT DESIGN FLOW

2.1. Solution requirements

In the previous section, a number of acute bottlenecks in the design process have been identified. In essence, an environment is needed, which transcends the interoperability problems of modern EDA tools. To achieve this, the environment has to be flexible in several key aspects.

Firstly, the environment has to be modular in nature. This is required to allow expansion to include new tools as

they become available, as well as to enable the designer to build a custom design flow only from those tools which are needed.

Also, the environment has to be independent from any particular vendor's tools or formats. Hence, the environment will be able to integrate tools from various vendors, as well as academic/research projects, and any in-house developed automation, such as scripts, templates, or similar.

To allow unobstructed communication between teams, the environment should eliminate the need for separate system descriptions. Hence, the single system description, used by *all* the teams simultaneously, would provide the ultimate means of cooperative refinement of a design, from the initial concept to the final implementation. Such a single system description should also be flexible through having a modular structure, accommodating equally all the teams. Thus, the structure of the single system description is a superset of all the constructs required by all the teams, and the contents of the single system description is a superset of all the separate system descriptions used by the teams currently.

2.2. Survey of industrial and university approaches

Several research initiatives, both in the commercial and academic arenas, are currently striving to close the design and productivity gaps. This section presents a comparative survey of these efforts.

A notable approach to EDA tool integration is provided by the model integrated computing (MIC) community [8]. This academic concept of model development gave rise to an environment for tool integration [9]. In this environment, the need for centering the design process on a single description of the system is also identified, and the authors present an implementation in the form of an integrated model server (IMS), based on a database system. The structure of the entire environment is expandable and modular in structure, with each new tool introduced into the environment requiring a new interface. The major shortcoming of this environment is its dedication to development of software components only. As such, this approach addresses solely the algorithmic modelling of the system, resulting in software at the application level. Thus, this environment does not support architectural and implementation levels of the design process.

Synopsys is one of the major EDA tool vendors offering automated support for many parts of the design process. Recognising the increasing need for efficiency in the design process and integration of various EDA tools, Synopsys developed a commercial environment for tool integration, the Galaxy Design Platform [10]. This environment is also based on a single description of the system, implemented as a database and referred to as the open Milkyway database. Thus, this environment eliminates the need for rewriting system descriptions at various stages of the design process. It also covers both the design and the verification processes and is capable of integrating a wide range of Synopsys commercial EDA tools. An added bonus of this approach is the open nature of the interface format to the Milkyway database, allowing third-party EDA tools to be integrated into the tool chain, if these adhere to the interface standard. However, this environment is essentially a proprietary scheme for integrating existing Synopsys products, and as such lacks any support from other parties.

The SPIRIT consortium [11] acknowledges the inherent inefficiency of interfacing incompatible EDA tools from various vendors. The work of this international body focuses on creating interoperability between different EDA tool vendors from the point of view of their customers, the product developers. Hence, the solution offered by the SPIRIT consortium [12] is a standard for packaging and interfacing of IP blocks used during system development. The existence and adoption of this standard ensures interoperability between EDA tools of various vendors as well as the possibility for integration of IP blocks which conform to the standard. However, this approach requires widest possible support from the EDA industry, which is currently lacking. Also, even the full adoption of this IP interchange format does not eliminate the need for multiple system descriptions over the entire design process. Finally, the most serious shortcoming of this methodology is that it provides support only for the lower levels of the design process, namely, the lower part of the architecture level (component assembly) and the implementation level.

In the paper of Posadas et al. [13] a single source design environment based on SystemC is proposed. Within this environment analysis tools are provided for time estimations for either hardware or software implementations. After this performance evaluation, it is possible to insert hardware/software partitioning information directly in the SystemC source code. Further, the generation of software for real-time application is addressed by a SystemC-to-eCos library, which replaces the SystemC kernel by real-time operating system functions. Despite being capable of describing a system consistently on different abstraction levels based on a single SystemC description, this does not offer a concrete and general basis for integration of design tools at all abstraction levels.

Raulet et al. [14] present a rapid prototyping environment based on a single tool called SynDex. Within this environment the user starts by defining an algorithm graph, an architecture graph, and constraints. Further executables for special kernels are automatically generated, while heuristics are used to minimize the total execution time of the algorithm. Those kernels provide the functionality of implementations in software and hardware, as well as models for communication.

The open tool integration environment (OTIE) [15] is a consistent design environment, aimed at fulfilling the requirements set out in Section 2.1. This environment is based on the single system description (SSD), a central repository for all the refinement information during the entire design process. As such, the SSD is used simultaneously by all the design teams. In the OTIE, each tool in the design process still performs its customary function, as in the traditional tool chain, but the design refinements from all the tools are now stored in just one system descriptions (the SSD) and thus no longer subject to constant rewriting. Hence, the SSD is a

superset of all the system descriptions present in the traditional tool chain.

The SSD is implemented as a MySQL [16] database, which brings several benefits. Firstly, the database implementation of the SSD supports virtually unlimited expandability, in terms of both structure and volume. As new refinement information arrives to be stored in the SSD, either it can be stored within the existing structure, or it may require an extension to the entity-relationship structure of the SSD, which can easily be achieved through addition of new tables or links between tables. Also, the database, on which this implementation of the SSD is based, is inherently a multiuser system, allowing transparent and uninterrupted access to the contents of the SSD to all the designers simultaneously. Furthermore, the security of the database implementation of the SSD is assured through detailed setting of access privileges of each team member and integrated EDA design tool to each part of the SSD, as well as the seamless integration of a version control system, to automatically maintain revision history of all the information in the SSD. Finally, accessing the refinement information (both manually and through automated tools) is greatly simplified in the database implementation of the SSD by its structured query language (SQL) interface.

Several EDA tool chains have been integrated into the OTIE, including environments for virtual prototyping [17, 18], hardware/software partitioning [19], high-level system characterisation [20], and floating-point to fixed-point conversion [21]. The deployment of these environments has shown the ability of the OTIE concept to reduce the design effort drastically through increased automation, as well as close the existing gaps in the automation coverage, by integrating novel EDA tools as they become available.

3. SYSTEM ANALYSIS

For the design of a signal processing system consisting of hardware and software many different programming languages have been introduced like VHDL, Verilog, or SystemC. During the refinement process it is of paramount importance to assure the quality of the written code and to base the design decisions on reliable characteristics. Those characteristics of the code are called metrics and can be identified on the different levels of abstraction.

The terms metric and measure are used as synonyms in literature, whereas a metric is in general a measurement, which maps an empirical object to a numerical object. This function should preserve all relations and structures. In other words, a quality characteristic should be linearly related to a measure, which is a basic concept of measurement at all. Those metrics can be software related or hardware related.

3.1. Software-related metrics

In the area of software engineering the interest in the measurement of software properties is ongoing since the first programming languages appeared [22]. One of the earliest software measures is the lines of code [23], which is still used today.

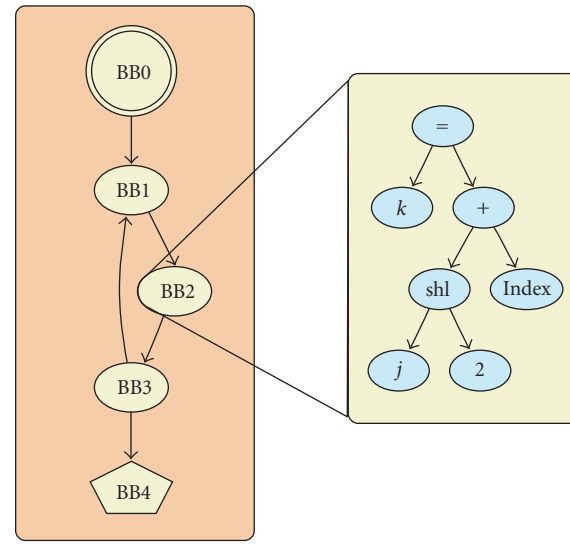


Figure 2: Control flow graph (CFG) and expression tree of one basic block.

In general the algorithm inside a function, written in the 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 ending in a control flow statement as a last instruction. A control flow graph is a directed graph with only one *root* and one *exit*. A *root* defines a vertex with no incoming edge and the *exit* defines a vertex with no outgoing edge. Due to programming constructs like loops those graphs are not cycle-free. The sequence of data operations inside of one BB forms itself a data flow graph (DFG) or equivalently one or more expression trees. Figure 2 shows an example of a function and its graph descriptions.

For the generation of DFG and CFG a parsing procedure of the source code has to be accomplished. This task is usually performed by a compiler. The step of compilation is separated into two steps, firstly, a front end transforms the source code into an intermediate representation (abstract syntax tree). At this step target independent optimizations are already applied, like dead code elimination or constant propagation. In a second step, the internal representation is mapped to a target architecture.

The analysis of a CFG can have different scopes: a small number of adjacent instructions, a single basic block, across several basic blocks (intraprocedural), across procedures (interprocedural), or a complete program.

For the CFG and DFG some common basic properties can be identified as follows.

(i) For each graph type G, a set of vertices V, and edges E can be defined, where the value $|V|$ denotes the number of vertices and $|E|$ denotes the number of edges.

(ii) A path of G is defined as an ordered sequence $S = (v_{\text{root}}\ v_x\ v_y\ \cdots\ v_{\text{exit}})$ of vertices starting at the *root* and ending at the *exit* vertex.

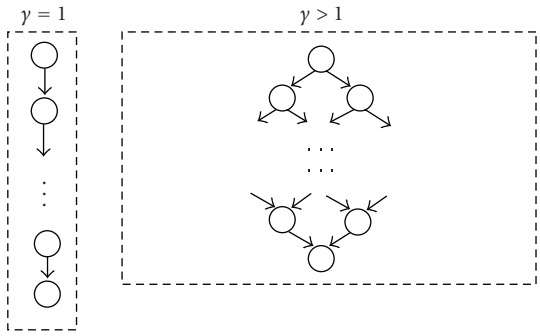


Figure 3: Degree of parallelism for $\gamma = 1$ and $\gamma > 1$.

(iii) The path with the maximum number of vertices is called the longest path or critical path and consists of $|V_{\mathrm{LP}}|$ vertices.

(iv) The degree of parallelism γ [24] can be defined as the number of all vertices $|V|$ divided by the number of vertices in the longest path $|V_{\mathrm{LP}}|$ of the algorithm

$$\gamma = \frac{|V|}{|V_{\mathrm{LP}}|}. \tag{1}$$

In Figure 3 it can be seen that for a γ value of 1, the graph is sequential and for $\gamma > 1$ the graph has many vertices in parallel, which offers possibilities for the reuse of resources.

In order to render the CFG context more precisely, we can apply these properties and define some important metrics to characterise the algorithm.

Definition 1 (longest path weight for operation j). Every vertex of a CFG can be annotated with a set of different weights $\mathbf{w}(v_i) = (w_1^i, w_2^i, \ldots, w_m^i)^T$, $i = 1 \cdots |V|$, that describes the occurrences of its internal operations (e.g., w_1^i = number of ADD operations in vertex v_i). Accordingly, a specific longest path with respect to the jth distinct weight, S_{LP}^j, can be defined as the sequence of vertices $(v_{\mathrm{root}}\ v_l\ \cdots\ v_{\mathrm{exit}})$, which yields a maximum path weight PW_j by summing up all the weights $w_j^{\mathrm{root}}, w_j^l, \ldots, w_j^{\mathrm{exit}}$ of the vertices that belong to this path as in

$$\mathrm{PW}_j = \sum_{v_i \in S_{\mathrm{LP}}^j} \mathbf{w}(v_i)\mathbf{d}_j. \tag{2}$$

Here the selection of the weight with the type j is accomplished by multiplication with a vector $\mathbf{d}_j = (\delta_{0j}, \ldots, \delta_{mj})^T$ defined with the Kronecker-delta δ_{ij}.

Definition 2 (degree of parallelism for operation j). Similar to the path weight PW_j, a global weight GW_j can be defined as

$$\mathrm{GW}_j = \sum_{v_i \in V} \mathbf{w}(v_i)\mathbf{d}_j, \tag{3}$$

which represents the operation-specific weight of the whole CFG. Accordingly an operation-specific γ_j is defined as follows:

$$\gamma_j = \frac{\mathrm{GW}_j}{\mathrm{PW}_j} \tag{4}$$

to reflect the reuse capabilities of each operation unit for operation j.

Definition 3 (cyclomatic complexity). The cyclomatic complexity, as defined by McCabe [25], states the theoretical number (see (5)) of required test cases in order to achieve the structural testing criteria of a full path coverage:

$$V(G) = |E| - |V| + 2. \tag{5}$$

The generation of the verification paths is presented by Poole [26] based on a modified depth-first search through the CFG.

Definition 4 (control orientation metrics). The control orientation metrics (COM) identifies whether a function is dominated by control operations,

$$\mathrm{COM} = \frac{N_{\mathrm{cop}}}{N_{\mathrm{op}} + N_{\mathrm{cop}} + N_{\mathrm{mac}}}. \tag{6}$$

Here N_{cop} defines the number of control statements (if, for, while), N_{op} defines the number of arithmetic and logic operations, and N_{mac} the number of memory accesses. When the COM value tends to be 1 the function is dominated by control operations. This is usually an indicator that an implementation of a control-oriented algorithm is more suited for running on a controller than to be implemented as dedicated hardware.

3.2. Hardware-related metrics

Early estimates of area, execution time, and power consumption of a specific algorithm implemented in hardware are crucial for design decisions like hardware/software partitioning (Section 5) and architecture exploration (Section 6.1). The effort of elaborating different implementations is usually not feasible in order to find optimal solutions. Therefore, only critical parts are modelled (rapid prototyping [6]) in order to measure worst-case scenarios, with the disadvantage that side effects on the rest of the system are neglected. According to Gajski et al. [27] those estimates must satisfy three criteria: accuracy, fidelity, and simplicity.

The estimation of area is based on an area characterization of the available operations and on an estimation of the needed number of operations (e.g., ADD, MUL). The area consumption of an operation is usually estimated by a function dependent on the number of inputs/outputs and their bit widths [28]. Further, the number of operations, for example, in Boolean expressions can be estimated by the number of nodes in the corresponding Boolean network [29]. Area estimation for design descriptions higher than register transfer level, like SystemC, try to identify a simple model for the high-level synthesis process [30].

The estimation of execution time of a hardware implementation requires the estimation of scheduling and resource allocation, which are two interdependent tasks. Path-based techniques transform an algorithm description from its CFG and DFG representation into a directed acyclic graph. Within this acyclic graph worst-case paths can be investigated by static analysis [31]. In simulation-based approaches the algorithm is enriched with functionality for tracing the execution paths during the simulation. This technique is, for example, described for SystemC [32] and MATLAB [33]. Additionally a characterization of the operations regarding their timing (delay) has to be performed.

Power dissipation in CMOS is separated into two components, the static and the dominant dynamic parts. Static power dissipation is mainly caused by leakage currents, whereas the dynamic part is caused by charging/discharging capacitances and the short circuit during the switching. Charging accounts for over 90% of the overall power dissipation [34]. Assuming that capacitance is related to area, area estimation techniques, as discussed before, have to be applied. Fornaciari et al. [35] present power models for different functional units like registers and multiplexers. Several techniques for predicting the switching activity of a circuit are presented by Landman [36].

3.3. Cost function and affinity

Usually the design target is the minimization of a cost or objective function with inequality constraints [37]. This cost function c depends on $\mathbf{x} = (x_1, \ldots, x_n)^T$, where the elements x_i represent normalized and weighted values of timing, area, and power but also economical aspects (e.g., cyclomatic complexity relates to verification effort) could be addressed. This leads to the minimization problem

$$\min c(\mathbf{x}). \tag{7}$$

Additionally those metrics have a set of constraints b_i like maximum area, maximum response time, or maximum power consumption given by the requirements of the system. Those constraints, which can be grouped to a vector $\mathbf{b} = (b_1, \ldots, b_n)^T$ define a set of inequalities,

$$\mathbf{x} \leq \mathbf{b}. \tag{8}$$

A further application of the presented metrics is its usage for the hardware/software partitioning process. Here a huge search space demands for heuristics that allows for partitioning within reasonable time. Nevertheless, a reduction of the search space can be achieved by assigning certain functions to hardware or software beforehand. This can be accomplished by an affinity metric [38]. Such an affinity can be expressed in the following way:

$$A = \frac{1}{\mathrm{COM}} + \sum_{j \in J} \gamma_j. \tag{9}$$

A high value A and thus a high affinity of an algorithm to a hardware implementation are caused by less control operations and high parallelism of the operations that are used in the algorithm. Thus an algorithm with an affinity value higher than a certain threshold can be selected directly to be implemented in hardware.

4. FLOATING-POINT TO FIXED-POINT CONVERSION

Design of embedded systems typically starts with the conversion of the initial concept of the system into an executable algorithmic model, on which high-level specifications of the system are verified. At this level of abstraction, models invariably use floating-point formats, for several reasons. Firstly, while the algorithm itself is undergoing changes, it is necessary to disburden the designer from having to take numeric effects into account. Hence, using floating-point formats, the designer is free to modify the algorithm itself, without any consideration of overflow and quantization effects. Also, floating-point formats are highly suitable for algorithmic modeling because they are natively supported on PC or workstation platforms, where algorithmic modeling usually takes place.

However, at the end of the design process lies the implementation stage, where all the hardware and software components of the system are fully implemented in the chosen target technologies. Both the software and hardware components of the system at this stage use only fixed-point numeric formats, because the use of fixed-point formats allows drastic savings in all traditional cost metrics: the required silicon area, power consumption, and latency/throughput (i.e., performance) of the final implementation.

Thus, during the design process it is necessary to perform the conversion from floating-point to suitable fixed-point numeric formats, for all data channels in the system. This transition necessitates careful consideration of the ranges and precision required for each channel, the overflow and quantisation effects created by the introduction of the fixed-point formats, as well as a possible instability which these formats may introduce. A trade-off optimization is hence formed, between minimising introduced quantisation noise and minimising the overall bitwidths in the system, so as to minimise the total system implementation cost. The level of introduced quantisation noise is typically measured in terms of the signal to quantisation noise ratio (SQNR), as defined in (10), where v is the original (floating-point) value of the signal and $\hat{v}$ is the quantized (fixed-point) value of the signal:

$$\mathrm{SQNR} = 20 \times \log \left| \frac{v}{v - \hat{v}} \right|. \tag{10}$$

The performance/cost tradeoff is traditionally performed manually, with the designer estimating the effects of fixed-point formats through system simulation and determining the required bitwidths and rounding/overflow modes through previous experience or given knowledge of the system architecture (such as predetermined bus or memory interface bitwidths). This iterative procedure is very time consuming and can sometimes account for up to 50% of the total design effort [39]. Hence, a number of initiatives to automate the conversion from floating-point to fixed-point formats have been set up.

In general, the problem of automating the conversion from floating-point to fixed-point formats can be based on either an analytical (static) or statistical (dynamic) approach. Each of these approaches has its benefits and drawbacks.

4.1. Analytical approaches

All the analytical approaches to automate the conversion from floating-point to fixed-point numeric formats find their roots in the static analysis of the algorithm in question. The algorithm, represented as a control and data flow graph (CDFG), is statically analysed, propagating the bitwidth requirements through the graph, until the range, precision, and sign mode of each signal are determined.

As such, analytical approaches do not require any simulations of the system to perform the conversion. This typically results in significantly improved runtime performance, which is the main benefit of employing such a scheme. Also, analytical approaches do not make use of any input data for the system. This relieves the designer from having to provide any data sets with the original floating-point model and makes the results of the optimisation dependent only on the algorithm itself and completely independent of any data which may eventually be used in the system.

However, analytical approaches suffer from a number of critical drawbacks in the general case. Firstly, analytical approaches are inherently only suitable for finding the upper bound on the required precision, and are unable to perform the essential trade-off between system performance and implementation cost. Hence, the results of analytical optimisations are excessively conservative, and cannot be used to replace the designer's fine manual control over the trade-off. Furthermore, analytical approaches are not suitable for use on all classes of algorithms. It is in general not possible to process nonlinear, time-variant, or recursive systems with these approaches.

FRIDGE [39] is one of the earliest environments for floating-point to fixed point conversion and is based on an analytical approach. This environment has high runtime performance, due to its analytical nature, and wide applicability, due to the presence of various back-end extensions to the core engine, including the VHDL back end (for hardware component synthesis) and ANSI-C and assembly back ends (for DSP software components). However, the core engine relies fully on the designer to preassign fixed-point formats to a sufficient portion of the signals, so that the optimisation engine may propagate these to the rest of the CDFG structure of the algorithm. This environment is based on fixed-C, a proprietary extension to the ANSI-C core language and is hence not directly compatible with standard design flows. The FRIDGE environment forms the basis of the commercial Synopsys CoCentric Fixed-Point Designer [40] tool.

Another analytical approach, Bitwise [41], implements both forward and backward propagations of bitwidth requirements through the graph representation of the system, thus making more efficient use of the available range and precision information. Furthermore, this environment is capable of tackling complex loop structures in the algorithm by calculating their closed-form solutions and using these to propagate the range and precision requirements. However, this environment, like all analytical approaches, is not capable of carrying out the performance-cost trade-off and results in very conservative fixed-point formats.

An environment for automated floating-point to fixed-point conversion for DSP code generation [42] has also been presented, minimising the execution time of DSP code through the reduction of variable bitwidths. However, this approach is only suitable for software components and disregards the level of introduced quantisation noise as a system-level performance metric in the trade-off.

An analytical approach based on affine arithmetic [43] presents another fast, but conservative, environment for automated floating-point to fixed-point conversion. The unique feature of this approach is the use of *probabilistic* bounds on the distribution of values of a data channel. The authors introduce the probability factor λ, which in a normal hard upper-bound analysis equals 1. Through this probabilistic relaxation scheme, the authors set $\lambda = 0.999999$ and thereby achieve significantly more realistic optimisation results, that is to say, closer to those achievable by the designer through system simulations. While this scheme provides a method of relaxing the conservative nature of its core analytical approach, the mechanism of controlling this separation (namely, the trial-and-error search by varying the λ factor) does not provide a means of controlling the performance-cost tradeoff itself and thus replacing the designer.

4.2. Statistical approaches

The statistical approaches to perform the conversion from floating-point to fixed-point numeric formats are based on system simulations and use the resulting information to carry out the performance-cost tradeoff, much like the designer does during the manual conversion.

Due to the fact that these methods employ system simulations, they may require extended runtimes, especially in the presence of complex systems and large volumes of input data. Hence, care has to be taken in the design of these optimisation schemes to limit the number of required system simulations.

The advantages of employing a statistical approach to automate the floating-point to fixed-point conversion are numerous. Most importantly, statistical algorithms are inherently capable of carrying out the performance-cost trade-off, seamlessly replacing the designer in this design step. Also, all classes of algorithms can be optimised using statistical approaches, including nonlinear, time-variant, or recursive systems.

One of the earliest research efforts to implement a statistical floating-point to fixed-point conversion scheme concentrates on DSP designs represented in C/C++ [44]. This approach shows high flexibility, characteristic to statistical approaches, being applicable to nonlinear, recursive, and time-variant systems.

However, while this environment is able to explore the performance-cost tradeoff, it requires manual intervention

by the designer to do so. The authors employ two optimisation algorithms to perform the trade-off: full search and a heuristic with linear complexity. The high complexity of the full search optimisation is reduced by grouping signals into clusters, and assigning the same fixed-point format to all the signals in one cluster. While this can reduce the search space significantly, it is an unrealistic assumption, especially for custom hardware implementations, where all signals in the system have very different optimal fixed-point formats.

QDDV [45] is an environment for floating-point to fixed-point conversion, aimed specifically at video applications. The unique feature of this approach is the use of two performance metrics. In addition to the widely used *objective* metric, the SQNR, the authors also use a *subjective* metric, the mean opinion score (MOS) taken from ten observers.

While this environment does employ a statistical framework for measuring the cost and performance of a given fixed-point format, no automation is implemented and no optimisation algorithms are presented. Rather, the environment is available as a tool for the designer to perform manual "tuning" of the fixed-point formats to achieve acceptable subjective and objective performance of the video processing algorithm in question. Additionally, this environment is based on Valen-C, a custom extension to the ANSI-C language, thus making it incompatible with other EDA tools.

A further environment for floating-point to fixed-point conversion based on a statistical approach [46] is aimed at optimising models in the MathWorks Simulink [47] environment. This approach derives an optimisation framework for the performance-cost trade-off, but provides no optimisation algorithms to actually carry out the trade-off, thus leaving the conversion to be performed by the designer manually.

A fully automated environment for floating-point to fixed-point conversion called *fixify* [21] has been presented, based on a statistical approach. While this results in fine control over the performance-cost trade-off, *fixify* at the same time dispenses with the need for exhaustive search optimisations and thus drastically reduces the required runtimes. This environment fully replaces the designer in making the performance-cost trade-off by providing a palette of optimisation algorithms for different implementation scenarios.

For designs that are to be mapped to software running on a standard processor core, restricted-set full search is the best choice of optimisation technique, since it offers guaranteed optimal results and optimises the design directly to the set of fixed-point bitwidths that are native to the processor core in question. For custom hardware implementations, the best choice of optimisation option is the branch-and-bound algorithm [48], offering guaranteed optimal results. However, for high-complexity designs with relatively long simulation times, the greedy search algorithm is an excellent alternative, offering significantly reduced optimisation runtimes, with little sacrifice in the quality of results.

Figure 4 shows the results of optimising a multiple-input multiple-output (MIMO) receiver design by all three optimisation algorithms in the *fixify* environment. The results are presented as a trade-off between the implementation cost c (on the vertical axis) and the SQNR, as defined in (10)

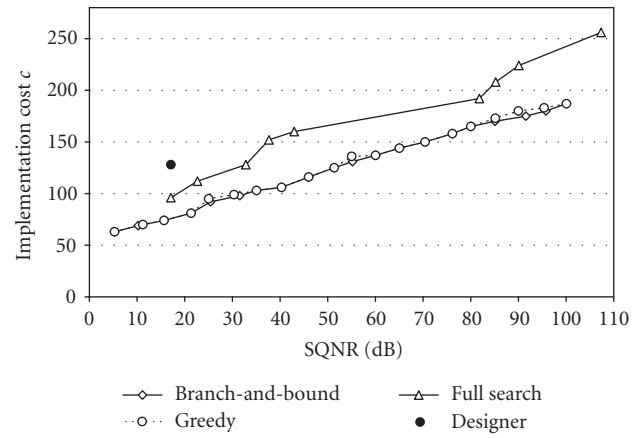


Figure 4: Optimization results for the MIMO receiver design.

(on the horizontal axis). It can immediately be noted from Figure 4 that all three optimisation methods generally require increased implementation cost with increasing SQNR requirements, as is intuitive. In other words, the optimisation algorithms are able to find fixed-point configurations with lower implementation costs when more degradation of numeric performance is allowed.

It can also be noted from Figure 4 that the optimisation results of the restricted-set full search algorithm consistently (i.e., over the entire examined range [5 dB, 100 dB]) require higher implementation costs for the same level of numeric performance then both the greedy and the branch-and-bound optimisation algorithms. The reason for this effect is the restricted set of possible bitwiths that the full search algorithm can assign to each data channel. In this example, the restricted-set full search algorithm uses the word length set of $\{16, 32, 64\}$, corresponding to the available set of fixed-point formats on the TIC6416 DSP which is used in the original implementation [49]. The full search algorithm can only move through the solution space in large quantum steps, thus not being able to fine tune the fixed-point format of each channel. On the other hand, greedy and branch-and-bound algorithms both have full freedom to assign any positive integer (strictly greater than zero) as the word length of the fixed-point format for each channel in the design, thus consistently being able to extract fixed-point configurations with lower implementation costs for the same SQNR levels.

Also, Figure 4 shows that, though the branch-and-bound algorithm consistently finds the fixed-point configuration with the lowest implementation cost for a given level of SQNR, the greedy algorithm performs only slightly worse. In 13 out of the 20 optimizations, the greedy algorithm returned the same fixed-point configuration as the branch-and-bound algorithm. In the other seven cases, the subtree relaxation routine of the branch-and-bound algorithm discovered a superior fixed-point configuration. In these cases, the relative improvement of using the branch-and-bound algorithm ranged between 1.02% and 3.82%.

Furthermore, it can be noted that the fixed-point configuration found by the designer manually can be improved

for both the DSP implementation (i.e., with the restricted-set full search algorithms) and the custom hardware implementation (i.e., with the greedy and/or branch-and-bound algorithms). The designer optimized the design to the fixed-point configuration where all the word lenghts are set to 16 bits by manual trial and error, as is traditionally the case. After confirming that the design has satisfactory performance with all word lengths set to 32 bits, the designer assigned all the word lengths to 16 bits and found that this configuration also performs satisfactorily. However, it is possible to obtain lower implementation cost for the same SQNR level, as well as superior numeric performance (i.e., higher SQNR) for the same implementation cost, as can be seen in Figure 4.

It is important to note that *fixify* is based entirely on the SystemC language, thus making it compatible with other EDA tools and easier to integrate into existing design flows. Also, the *fixify* environment requires no change to the original floating-point code in order to perform the optimisation.

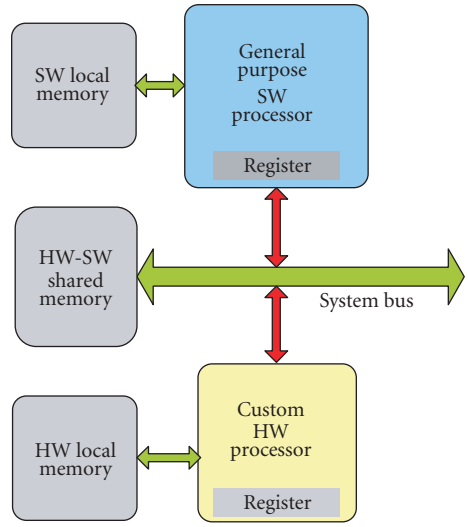


Figure 5: Common implementation architecture.

5. HARDWARE/SOFTWARE PARTITIONING

Hardware/software partitioning can in general be described as the mapping of the interconnected functional objects that constitute the behavioural model of the system onto a chosen architecture model. The task of partitioning has been thoroughly researched and enhanced during the last 15 years and produced a number of feasible solutions, which depend heavily on their prerequisites:

(i) the underlying system description;
(ii) the architecture and communication model;
(iii) the granularity of the functional objects;
(iv) the objective or cost function.

The manifold formulations entail numerous very different approaches to tackle this problem. The following subsection arranges the most fundamental terms and definitions that are common in this field and shall prepare the ground for a more detailed discussion of the sophisticated strategies in use.

5.1. Common terms

The functionality can be implemented with a set of interconnected system components, such as general-purpose CPUs, DSPs, ASICs, ASIPs, memories, and buses. The designer's task is in general twofold: selection of a set of system components or, in other words, the determination of the architecture, and the mapping of the system's functionality among these components. The term *partitioning*, originally describing only the latter, is usually adopted for a combination of both tasks, since these are closely interlocked. The level, on which partitioning is performed, varies from group to group, as well as the expressions to describe these levels. The term *system level* has always been referring to the highest level of abstraction. But in the early nineties the system level identified VHDL designs composed of several functional objects in the size of an FIR or LUT. Nowadays the term system level describes functional objects of the size of a Viterbi or a Huffman decoder. The complexity differs by one order of magnitude. In the following the granularity of the system partitioning is labelled decreasingly as follows: system level (e.g., Viterbi, UMTS Slot Synchronisation, Huffman, Quicksort, etc.), process level (FIR, LUT, Gold code generator, etc.), and operational level (MAC, ADD, NAND, etc.) The final implementation has to satisfy a set of design constraints, such as cost, silicon area, power consumption, and execution time. Measures for these values, obtained by high-level estimation, simulation, or static analysis, which characterize a given solution quantitatively are usually called *metrics*; see Section 3. Depending on the specific problem formulation a selection of metrics composes an *objective function*, which captures the *overall* quality of a certain partitioning as described in detail in Section 3.3.

5.2. Partitioning approaches

Ernst et al. [50] published an early work on the partitioning problem starting from an all-software solution within the COSYMA system. The underlying architecture model is composed of a programmable processor core, memory, and customised hardware (Figure 5).

The general strategy of this approach is the *hardware extraction* of the computational intensive parts of the design, especially loops, on a fine-grained basic block level (CDFG), until all timing constraints are met. These computation intensive parts are identified by simulation and profiling. User interaction is demanded since the system description language is C^x, a superset of ANSI-C. Not all C^x constructs have valid counterparts in a hardware implementation, such as dynamic data structures, and pointers. Internally simulated annealing (SA) [51] is utilized to generate different partitioning solutions. In 1994 the authors introduced an optional programmable coprocessor in case the timing constraints could not be met by hardware extraction [52]. The scheduling of the basic blocks is identified to be *as soon as possible*

(ASAP) driven, in other words, it is the simplest list scheduling technique also known as *earliest task first*. A further improvement of this approach is the usage of a dynamically adjustable granularity [53] which allows for restructuring of the system's functionality on basic block level (see Section 3.1) into larger partitioning objects.

In 1994, the authors Kalavade and Lee [54] published a fast algorithm for the partitioning problem. They addressed the coarse-grained mapping of processes onto an identical architecture (Figure 5) starting from a directed acyclic graph (DAG). The objective function incorporates several constraints on available silicon area (hardware capacity), memory (software capacity), and latency as a timing constraint. The global criticality/local phase (GCLP) algorithm is a greedy approach, which visits every process node once and is directed by a dynamic decision technique considering several cost functions.

The partitioning engine is part of the signal processing work suite Ptolemy [55] firstly distributed in the same year. This algorithm is compared to simulated annealing and a classical Kernighan-Lin implementation [56]. Its tremendous speed with reasonably good results is mentionable but in fact only a single partitioning solution is calculated in a vast search space of often a billion solutions. This work has been improved by the introduction of an embedded implementation bin selection (IBS) [57].

In the paper of Eles et al. [58] a tabu search algorithm is presented and compared to simulated annealing and Kernighan-Lin (KL). The target architecture does not differ from the previous ones. The objective function concentrates more on a trade-off between the communication overhead between processes mapped to different resources and reduction of execution time gained by parallelism. The most important contribution is the preanalysis before the actual partitioning starts. For the first time static code analysis techniques are combined with profiling and simulation to identify the computation intensive parts of the functional code. The static analysis is performed on operation level within the basic blocks. A suitability metric is derived from the occurrence of distinct operation types and their distribution within a process, which is later on used to guide the mapping to a specific implementation technology.

The paper of Vahid and Le [59] opened a different perspective in this research area. With respect to the architecture model a continuity can be stated as it does not deviate from the discussed models. The innovation in this work is the decomposition of the system into an access graph (AG), or call graph. From a software engineering point of view a system's functionality is often described with hierarchical structures, in which every edge corresponds to a function call. This representation is completely different from the block-based diagrams that reflect the data flow through the system in all digital signal processing work suites [47, 55]. The leaves of an access graph correspond to the simplest functions that do not contain further function calls (Figure 6).

The authors extend the Kernighan-Lin heuristic to be applicable to this problem instance and put much effort in the exploitation of the access graph structure to greatly reduce

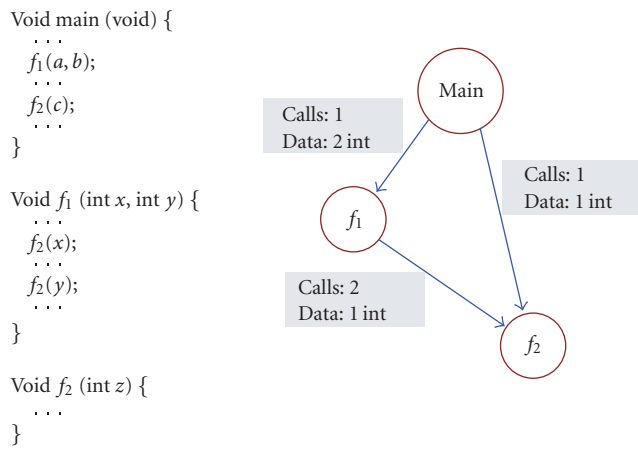


Figure 6: Code segment and corresponding access graph.

the runtime of the algorithm. Indeed their approach yields good results on the examined real and random designs in comparison with other algorithms, like SA, greedy search, hierarchical clustering, and so forth. Nevertheless, the assignment of function nodes to the programmable component lacks a proper scheduling technique, and the decomposition of a usually block-based signal processing system into an access graph representation is in most cases very time consuming.

5.3. Combined partitioning and scheduling approaches

In the later nineties research groups started to put more effort into combined partitioning and scheduling techniques. The first approach of Chatha and Vemuri [60] can be seen as a further development of Kalavade's work. The architecture consists of a programmable processor and a custom hardware unit, for example, an FPGA. The communication model consists of a RAM for hardware-software communication connected by a system bus, and both processors accommodate local memory units for internal communication. Partitioning is performed in an iterative manner on system level with the objective of the minimization of execution time while maintaining the area constraint.

The partitioning algorithm mirrors exactly the control structure of a classical Kernighan-Lin implementation adapted to more than two implementation techniques. Every time a node is tentatively moved to another kind of implementation, the scheduler estimates the change in the overall execution time instead of rescheduling the task subgraph. By this means a low runtime is preserved by paying reliability of their objective function. This work has been further extended for combined retiming, scheduling, and partitioning of transformative applications, that is, JPEG or MPEG decoder [61].

A very mature combined partitioning and scheduling approach for DAGs has been published by Wiangtong et al. [62]. The target architecture, which establishes the fundament of their work, adheres to the concept given in Figure 5.

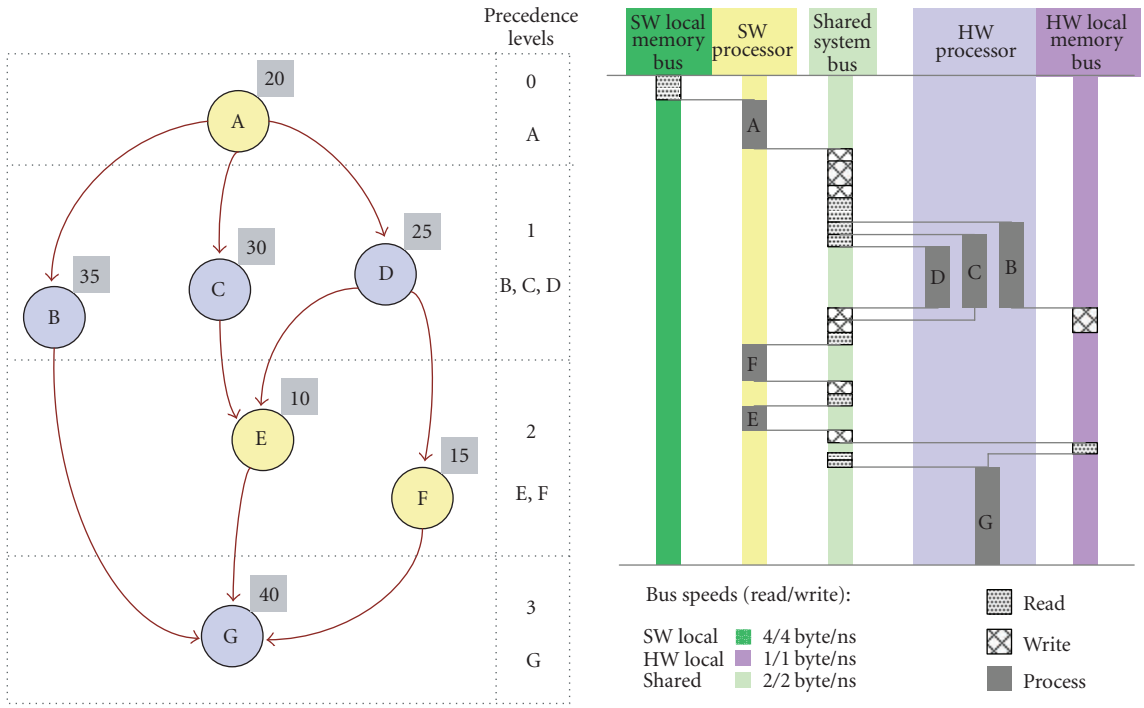


Figure 7: Rank-ordered DAG and its resulting schedule.

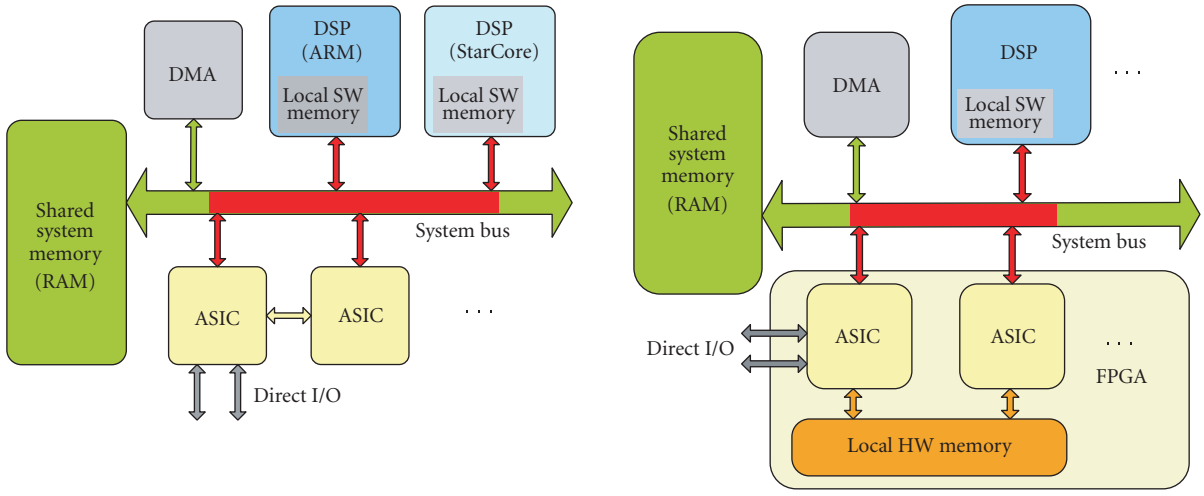
The work compares three heuristic methods to traverse the search space of the partitioning problem: simulated annealing, genetic algorithm, and tabu search. Additionally the most promising technique of this evaluation, tabu search, is further improved by a so-called *penalty reward* mechanism. This mechanism modifies the long-term memory, in which the information about most/least frequently visited neighbourhood solutions is stored. This solution yields the best results in terms of least processing time, shortest runtime of the algorithm, while meeting all the hardware area constraints.

The applied technique that is utilised to schedule a visited partitioning solution avoids any resource conflicts and is very fast. Not surprisingly the technique is essentially a list scheduling technique. The process nodes are grouped together a priori in a so-called precedence list, which is a rank-ordered sequence, where one sequence element, or one precedence level, contains all nodes with the same rank. As the nodes' ranks remain always the same in the DAG, independent from the current partitioning, only the ordering of the processes within *one* precedence level has to be calculated for every new partitioning solution. An example of this approach can be seen in Figure 7. The nodes of the DAG are ordered according to its rank and their different color identifies a certain mapping to either software or hardware. The scheduling of these processes is shown on the right side of Figure 7. Most notably this approach returns an exact system execution time to the partitioning engine in opposition to the estimation-based techniques described before. The scheduling is reasonably fast and collisions are avoided completely. However, the list scheduling fails to recognize situations in which one software process would enable many hardware processes running in parallel, whereas the instead preferred software process with the same rank does not have a single hardware successor, as the decision is based on the larger bus utilization.

The inspiration for the architecture model in the papers of Knerr et al. [18, 63] and the paper [62] originates from an industry-designed UMTS baseband receiver chip. Its abstraction (see Figure 8(a)) has been developed to provide a maximum degree of generality while being along the lines of the industry-designed SoCs in use. It consists of several (here two) DSPs handling the control-oriented functions, for instance, an ARM for the signalling part and a StarCore for the multimedia part, several hardware accelerating units (ASICs), for the data oriented and computation intensive signal processing, one system bus to a shared RAM for mixed resource communication, and optionally direct I/O to peripheral subsystems. In Figure 8(b) the simple modification towards the platform concept with one hardware processing unit (e.g., FPGA) has been established (cf. Figure 5).

Table 1 lists the access times for reading and writing bits via the different resources of the platform in Figure 8(b).

The graph representation of the system, which should be mapped onto this platform, is generally given in the form of a task graph. It is usually assumed to be a DAG describing the dependencies between the components of the system. The authors base their work on a graph representation for multirate systems, also known as synchronous data flow graphs [64]. This representation accomplished the backbone of renowned signal processing work suites, for example, Ptolemy [55] or SPW [65]. In Figure 9, a simple example of an SDF graph $G = (V, E)$ is depicted on the left,



(a) Platform abstraction from realistic UMTS chip

(b) Modification to meet common FPGA and DSP approaches

FIGURE 8: Origin and modification of a general SoC platform abstraction.

TABLE 1: Maximum delays for read/write accesses to the communication resources.

Communication	Read (bits/cycle)	Write (bits/cycle)
Local software memory	128	256
Local hardware memory	64	128
Shared system bus	256	512
Direct I/O	1024	1024

showing four vertices $V = \{v_1, \ldots, v_4\}$ connected by four edges $E = \{e_1, \ldots, e_4\}$. The numbers on the tail of each edge represent the number of bits produced per invocation. The numbers on the head of each edge indicate the number of bits consumed per invocation. On the right the decomposition of the SDF graph has been performed. In the single activation graph (SAG) the input/output rate dependencies have been solved and every *process invocation* is transformed into one vertex. The vertices v_1 and v_2 are doubled according to their distinct invocations that result from the data rate analysis. The solid edges indicate precedence as well as data transfers from one vertex to another, whereas the dashed edges just indicate precedence. The data rates at the edges, that is, interprocess communication, have been omitted for brevity in this figure.

Note that all of the known approaches discuss task graph sets, which are *homogeneous* SDF graphs. This assumption leads to the very convenient situation of *single* invocations of every task. In this work, general SDF graphs with different input and output rates (see edge e_2 in Figure 9) are considered. A mapping of a task in the SDF graph from hardware to software causes a more complex situation since certainly *all* invocations of this task have to be moved to the DSP.

It has to be stated that the hardware/software partitioning process itself relies heavily on the metrics and measurements generated with methods in Section 3. Only in combination

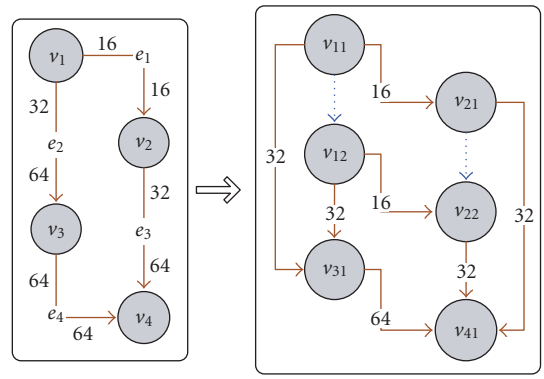


FIGURE 9: Simple SDFG (left) and the decomposition into its SAG (right).

with a high fidelity of values like execution time, power consumption, and chip area, the partitioning algorithm is capable to return useful decisions very early in the design process.

In 2004 the first work of this group has been published solely regarding the high-level metrics generation and the partitioning problem [19]. The objective function for the hardware/software partitioning included estimations of execution time for software and hardware implementation and gate counts for the hardware implementation. The core algorithm to examine the search space was an adaptation of the Kernighan-Lin min-cut heuristic [56] for process graphs. Mainly because of the advancements of the underlying architecture abstraction and the more and more realistic communication model, scheduling issues came to the fore. The generalization to SDF graph systems with multiple invocation of a single process has been shown [63]. This work focused on a fast rescheduling method, which returns exact execution times for every single partitioning solution, which a partitioning algorithm evaluates during its run. The performance

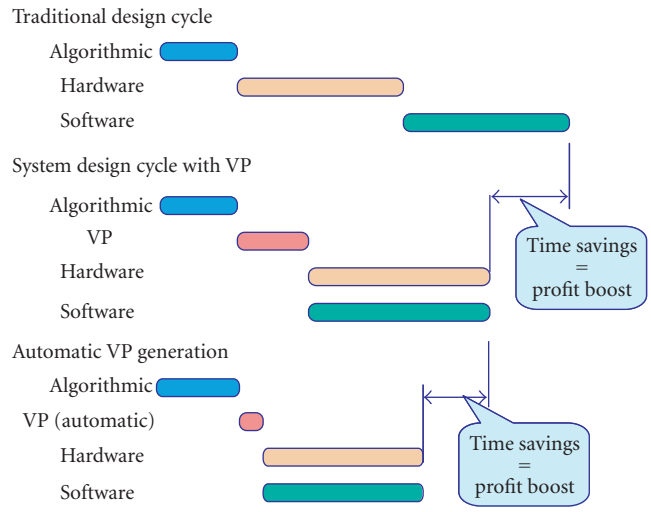


Figure 10: Decrease of design time by virtual prototyping and automatic generation of virtual prototypes.

of the so-called *local exploitation of parallelism (LEP)* algorithm has shown to be better than the aforementioned popular list scheduling techniques, like Hu-level scheduling [66] and earliest task first [67]. Most importantly LEP has been developed to preserve linear complexity, since it is aimed to be applied within partitioning algorithms that move incrementally through the search space (direct neighbourhood searches). This rescheduling method has been enhanced towards multicore architectures with many ASICs and *several* DSPs [63].

6. VIRTUAL PROTOTYPING

One of the main difficulties in the design of an embedded system, which consists of software and hardware parts, is that usually the design and testing of the strong related software parts have to wait until the hardware has been manufactured. Whereas hardware development, and especially its testing, is rather independent from the software development. Thus the design of an embedded system is a sequential process (Figure 10).

The application of a so-called virtual prototype (VP), which is a software model of the hardware, allows for earlier start of the software development process and provides a platform for hardware architecture evaluation. In this technique, software reflects the behavior of the hardware and implements the software interface to the software, as it will be realized later in hardware. Such a VP can be implemented faster than the hardware itself, because all the hardware implementation details specific to the chosen technology can be neglected and high-level description languages can be used instead of hardware description languages (HDLs).

Generally, a complex SoC reflects a platform-based design (PBD), typically one or more DSPs surrounded by multiple hardware accelerators (HA). Those HAs are called VP components if they are used inside a VP simulation. The hardware/software partitioning process transforms a system-level specification into a heterogeneous architecture composed of hardware and software modules. This partitioning can be performed by a tool supported way (Section 5) or manually based on the experience of the designer.

Additionally different abstraction levels of a VP support a refinement process, which is especially needed for systems with high complexity being too large to support a consistent refinement in one major step. Several properties of abstraction layers are proposed for a VP, as they can be time related (e.g., untimed, timed functional, bus cycle accurate, and cycle true), data related (e.g., floating-point and fixed-point representation), and communication related (e.g., synchronous data flow, transaction level modeling (TLM) [68], and open core protocol international partnership OCP [69]). In Figure 11 three different abstraction levels for a VP are shown: one VP (Figure 11(a)) for a first architecture evaluation, which is characterized by its properties (e.g., data rates, execution time, and power consumption); another one (Figure 11(b)) for software development, which achieves fast simulation performance by using a synchronous data flow description; and a third one (Figure 11(c)) for the cycle true cosimulation on register transfer level (RTL). The following sections explain those VP models in more detail.

6.1. *Virtual prototype for architecture exploration*

A first evaluation of the system performance is achieved by a high-level characterisation of a VP component regarding only its features like, for example, input/output data rates, worst-case execution time (WCET), best-case execution time (BCET), and power consumption. Those properties can be combined to a cost function as shown in Section 3.3. Such a model provides a base for investigating communication bottle necks on the bus system, power consumption of the system, and the meeting of real-time constraints.

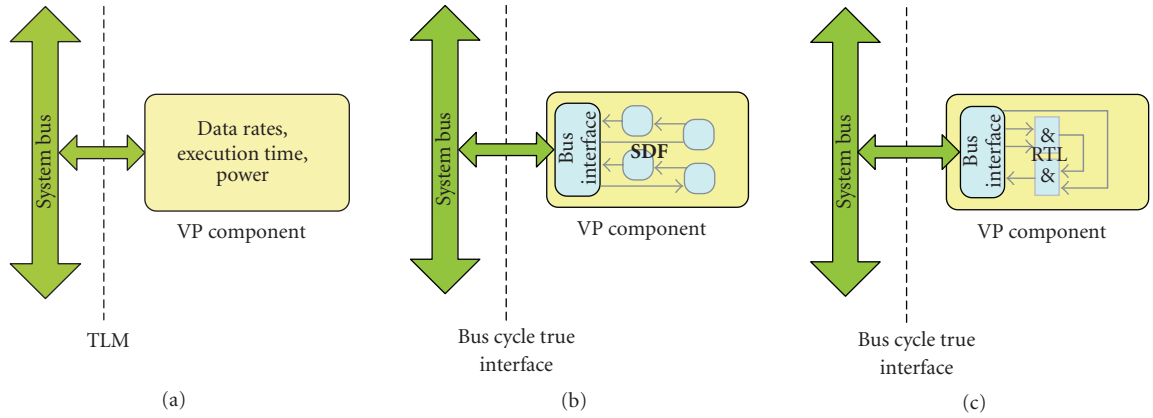


Figure 11: Different abstraction levels of a VP component.

6.2. Virtual prototyping for software development

In order to have high simulation speed together with a certain accuracy of the model, a VP component can have a cycle true interface to the bus system, whereas the implementation of the VP component is a high-level description (e.g., synchronous data flow description in COSSAP). An automatic generation method for a VP tailored for platform-based designs allows for a further decrease of development time [17, Figure 10]. Within such a method the algorithmic description is reused for the VP component (Figure 12).

Usually at algorithmic level the design information is free of communication details. Thus, in order to achieve communication of the VP components via the chosen platform, an object-oriented environment in C++ provides the functionality of functional blocks, ports, FIFOs, and scheduling. While this implementation implies a certain hardware platform, much emphasis is put on the fact that this platform is very general, a DSP with a common bus structure for its hardware accelerator units. The automatizm is implemented for COSSAP designs based on GenericC descriptions only. However, the methodology is left open for supporting other descriptions, like SystemC.

The implementation of such a VP representation needs a simulation environment that allows for simulation of parallel processes. This is provided by a simulation interface, which is proposed by the virtual socket interface association (VSIA) [70]. In this approach a static scheduling is used, achieving faster simulation compared to the event-based simulation of SystemC and VHDL. Even compared to a plain C++ implementation, the VSIA implementation introduces negligible overhead.

The evaluation of a hardware-software system in real-time constraints additionally needs for an accurate description environment of software and hardware. Software and hardware need to be annotated with execution time estimations. Especially the software parts need to take into account the effects of interrupts, which can be modelled with TIPSY (TImed Parallel SYstem Modeling) C++ [71].

6.3. Virtual prototype for hardware development

As a last step, the internal behavior of the hardware accelerators has to be transformed to a cycle true model. This step is usually called high-level synthesis, investigated by many research projects [72], and also adopted to commercially available tools like CatapultC [73]. In that sense VP also supports a refinement-step-based design, which allows a much more consistent forgoing than switching between description languages.

A semiautomatic synthesis is achieved by the MASIC (MATH to ASIC) environment allowing for describing the control part of a system with the global control, configuration, and timing language (GLOCCT). Within this language the FIFOs have to be defined and connected manually. Functions, which are described in C, are used for a bit true and cycle true implementation. Afterwards the RTL code is generated automatically. A speedup in the order of 5 to 8 times compared to manually creation is achieved [13].

The paper of Valderrama [74] describes communication structures that are provided in a library. Such communication libraries implement simple handshake mechanisms up to layered networks. Nevertheless, a focus is needed on the hardware/software cosimulation process in order to increase efficiency and quality of the design process.

7. CONCLUSIONS

This paper presents an overview of modern techniques and methodologies for increasing the efficiency of the design process of embedded systems, especially in the wireless communications domain. The key factor influencing efficiency is the organization and structure of the overall design process. In an effort to increase efficiency, modern design methodologies tend towards unified descriptions of the system, with flexible and generalized tool integration schemes. Such environments can save most of the design effort invested in rewriting system descriptions, thus resulting in a streamlined design process. However, the most substantial increase in

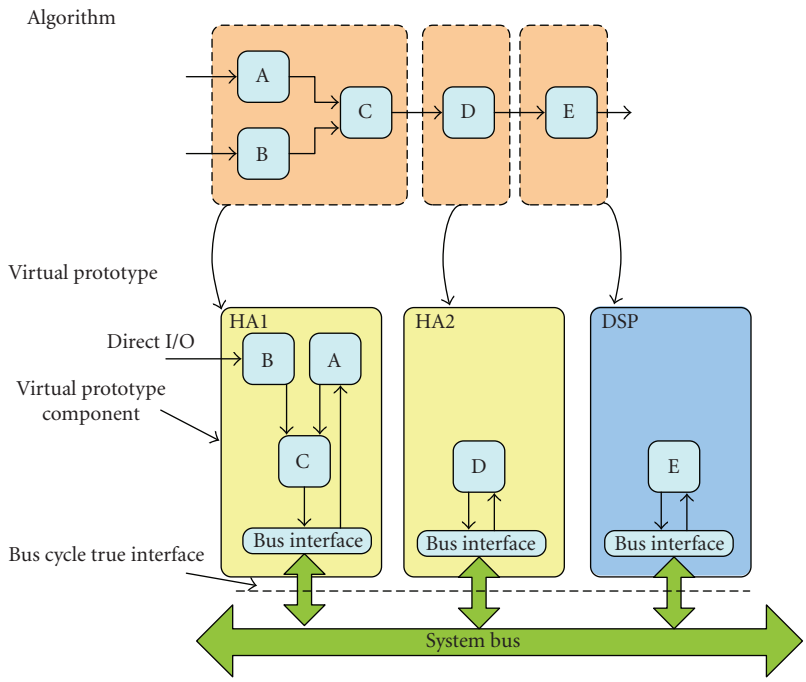


Figure 12: Reuse of algorithmic description for virtual prototype generation.

efficiency comes from the automation of all individual steps in the design process through dedicated tools which are integrated into the design methodologies.

Firstly, design decisions at all levels of the design process are based on characteristics of the system, also called metrics. Hence, reliable, fast, and accurate analysis of the system is of paramount importance. The required metrics are efficiently obtained through static code analysis techniques, which offer increased speed by avoiding lengthy simulations, as well as the capability to estimate a wide range of required system properties.

Floating-point to fixed-point conversion is a critical step in the design process whose automation offers significant savings in design effort. Automation through dynamic (data-driven) techniques is most promising, allowing for complete replacement of the designer's manual effort, while achieving the same quality of conversion results. Modern automation techniques offer optimization algorithms specifically suited for the conversion towards a particular implementation option, such as DSP code or custom hardware.

Hardware/software partitioning is another key step in the design process, for which a variety of automated techniques exists. The practical use and applicability of these implementations to industrial projects hinges heavily on the strength of the underlying algorithms, the degree to which the environment is tailored to the application domain, as well as the integration of the environment into an overall design methodology covering the entire design process.

Finally, virtual prototyping is a promising design technique for speeding up the design process, by allowing parallel development of both hardware and software components in the system. Modern design techniques for automated generation of virtual prototypes also exist, thus boosting the design productivity substantially.

ACKNOWLEDGMENT

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

REFERENCES

[1] Y. Neuvo, "Cellular phones as embedded systems," in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC '04)*, vol. 1, pp. 32–37, San Francisco, Calif, USA, February 2004.

[2] J. Hausner and R. Denk, "Implementation of signal processing algorithms for 3G and beyond," *IEEE Microwave and Wireless Components Letters*, vol. 13, no. 8, pp. 302–304, 2003.

[3] R. Subramanian, "Shannon vs. Moore: driving the evolution of signal processing platforms in wireless communications," in *Proceedings of IEEE Workshop on Signal Processing Systems (SIPS '02)*, p. 2, San Diego, Calif, USA, October 2002.

[4] G. Moore, "Cramming more components onto integrated circuits," *Electronics Magazine*, vol. 38, no. 8, pp. 114–117, 1965.

[5] International SEMATECH, "International Technology Roadmap for Semiconductors," 1999, http://www.sematech.org.

[6] M. Rupp, A. Burg, and E. Beck, "Rapid prototyping for wireless designs: the five-ones approach," *Signal Processing*, vol. 83, no. 7, pp. 1427–1444, 2003.

[7] R. L. Moigne, O. Pasquier, and J.-P. Calvez, "A graphical tool for system-level modeling and simulation with systemC," in *Proceedings of the Forum on Specification & Design Languages (FDL '03)*, Frankfurt, Germany, September 2003.

[8] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Model-integrated development of embedded software," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 145–164, 2003.

[9] G. Karsai, "Design tool integration: an exercise in semantic interoperability," in *Proceedings of the 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS '00)*, pp. 272–278, Edinburgh, UK, April 2000.

[10] Synopsys Inc., "Galaxy Design Platform," http://www.synopsys.com/products/solutions/galaxy_platform.html.

[11] SPIRIT Consortium, http://www.spiritconsortium.com.

[12] SPIRIT SchemaWorking Group Membership, "SPIRIT-User Guide v1.1," Tech. Rep., SPIRIT Consortium, San Diego, Calif, USA, June 2005.

[13] H. Posadas, F. Herrera, V. Fernández, P. Sánchez, E. Villar, and F. Blasco, "Single source design environment for embedded systems based on SystemC," *Design Automation for Embedded Systems*, vol. 9, no. 4, pp. 293–312, 2004.

[14] M. Raulet, F. Urban, J.-F. Nezan, C. Moy, O. Deforges, and Y. Sorel, "Rapid prototyping for heterogeneous multicomponent systems: an MPEG-4 stream over a UMTS communication link," *EURASIP Journal on Applied Signal Processing*, vol. 2006, Article ID 64369, 1–13, 2006, special issue on design methods for DSP systems.

[15] P. Belanović, B. Knerr, M. Holzer, G. Sauzon, and M. Rupp, "A consistent design methodology for wireless embedded systems," *EURASIP Journal on Applied Signal Processing*, vol. 2005, no. 16, pp. 2598–2612, 2005, special issue on DSP enabled radio.

[16] MySQL Database Products, http://www.mysql.com/products/database.

[17] B. Knerr, P. Belanović, M. Holzer, G. Sauzon, and M. Rupp, "Design flow improvements for embedded wireless receivers," in *Proceedings of the 12th European Signal Processing Conference (EUSIPCO '04)*, pp. 2015–2018, Vienna, Austria, September 2004.

[18] P. Belanović, B. Knerr, M. Holzer, and M. Rupp, "A fully automated environment for verification of virtual prototypes," *EURASIP Journal on Applied Signal Processing*, vol. 2006, Article ID 32408, 2006, special issue on design methods for DSP systems.

[19] B. Knerr, M. Holzer, and M. Rupp, "HW/SW partitioning using high level metrics," in *Proceedings of International Conference on Computer, Communication and Control Technologies (CCCT '04)*, vol. 8, pp. 33–38, Austin, Tex, USA, August 2004.

[20] M. Holzer and M. Rupp, "Static code analysis of functional descriptions in systemC," in *Proceedings of the 3rd IEEE International Workshop on Electronic Design, Test and Applications (DELTA '06)*, pp. 243–248, Kuala Lumpur, Malaysia, January 2006.

[21] P. Belanović and M. Rupp, "Automated floating-point to fixed-point conversion with the fixify environment," in *Proceedings of the 16th International Workshop on Rapid System Prototyping (RSP '05)*, pp. 172–178, Montreal, Canada, June 2005.

[22] M. Sheppered and D. Ince, *Derivation and Validation of Software Metrics*, Oxford University Press, New York, NY, USA, 1993.

[23] B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.

[24] Y. L. Moullec, P. Koch, J.-P. Diguet, and J.-L. Philippe, "Design trotter: building and selecting architectures for embedded multimedia applications," in *Proceedings of IEEE International Symposium on Consumer Electronics (ISCE '03)*, Sydney, Australia, December 2003.

[25] T. McCabe, "A complexity measure," *IEEE Transaction of Software Engineering*, vol. 2, no. 4, pp. 308–320, 1976.

[26] J. Poole, "A method to determine a basis set of paths to perform program testing," Report 5737, U.S. Department of Commerce/National Institute of Standards and Technology, Gaithersburg, Md, USA, November 1995.

[27] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic, Norwell, Mass, USA, 1992.

[28] J. Pal Singh, A. Kumar, and S. Kumar, "A multiplier generator for Xilinx FPGA's," in *Proceedings of the 9th IEEE International Conference on VLSI Design*, pp. 322–323, Bangalore, India, January 1996.

[29] K. M. Büyüksahin and F. N. Najm, "High-level area estimation," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED '02)*, pp. 271–274, Monterey, Calif, USA, August 2002.

[30] C. Brandolese, W. Fornaciari, and F. Salice, "An area estimation methodology for FPGA based designs at SystemC-level," in *Proceedings of the 41st Design Automation Conference (DAC '04)*, pp. 129–132, San Diego, Calif, USA, June 2004.

[31] M. Holzer and M. Rupp, "Static estimation of the execution time for hardware accelerators in system-on-chips," in *Proceedings of International Symposium on System-on-Chip (SoC '05)*, pp. 62–65, Tampere, Finland, November 2005.

[32] H. Posadas, F. Herrera, P. Sánchez, E. Villar, and F. Blasco, "System-level performance analysis in SystemC," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '04)*, vol. 1, pp. 378–383, Paris, France, February 2004.

[33] P. Bjureus, M. Millberg, and A. Jantsch, "FPGA resource and timing estimation from Matlab execution traces," in *Proceedings of the 10th International Symposium on Workshop on Hardware/Software Codesign*, pp. 31–36, Estes Park, Colo, USA, May 2002.

[34] S. Devadas and S. Malik, "A survey of optimization techniques targeting low power VLSI circuits," in *Proceedings of the 32nd ACM/IEEE Conference on Design Automation (DAC '95)*, pp. 242–247, San Francisco, Calif, USA, June 1995.

[35] W. Fornaciari, P. Gubian, D. Sciuto, and C. Silvano, "Power estimation of embedded systems: a hardware/software codesign approach," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 2, pp. 266–275, 1998.

[36] P. Landman, "High-level power estimation," in *Proceedings of International Symposium on Low Power Electronics and Design*, pp. 29–35, Monterey, Calif, USA, August 1996.

[37] T. K. Moon and W. C. Stirling, *Mathematical Methods and Algorithms for Signal Processing*, Prentice-Hall, Upper Saddle River, NJ, USA, 2000.

[38] D. Sciuto, F. Salice, L. Pomante, and W. Fornaciari, "Metrics for design space exploration of heterogeneous multiprocessor embedded systems," in *Proceedings of International Workshop on Hardware/Software Codesign*, pp. 55–60, Estes Park, Colo, USA, May 2002.

[39] H. Keding, M. Willems, M. Coors, and H. Meyr, "FRIDGE: a fixed-point design and simulation environment," in *Proceedings of Design, Automation and Test In Europe (DATE '98)*, pp. 429–435, Paris, France, February 1998.

[40] Synopsys, "Converting ANSI-C into fixed-point using Cocentric fixed-point designer," Tech. Rep., Synopsys, Mountain View, Calif, USA, April 2000.

[41] M. Stephenson, J. Babb, and S. Amarasinghe, "Bitwidth analysis with application to silicon compilation," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '00)*, pp. 108–120, Vancouver, BC, Canada, June 2000.

[42] D. Menard, D. Chillet, F. Charot, and O. Sentieys, "Automatic floating-point to fixed-point conversion for DSP code generation," in *Proceedings of International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '02)*, pp. 270–276, Grenoble, France, October 2002.

[43] C. F. Fang, R. A. Rutenbar, and T. Chen, "Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pp. 275–282, San Jose, Calif, USA, November 2003.

[44] S. Kim, K.-I. 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, no. 11, pp. 1455–1464, 1998.

[45] Y. Cao and H. Yasuura, "Quality-driven design by bitwidth optimization for video applications," in *Proceedings of IEEE/ACM Asia and South Pacific Design Automation Conference*, pp. 532–537, Kitakyushu, Japan, January 2003.

[46] C. Shi and R. W. Brodersen, "An automated floating-point to fixed-point conversion methodology," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '03)*, vol. 2, pp. 529–532, Hong Kong, April 2003.

[47] MathWorks Simulink, http://www.mathworks.com/products/simulink.

[48] J. Hromkovič, *Algorithmics for Hard Problems*, Springer, New York, NY, USA, 2nd edition, 2003.

[49] C. Mehlführer, F. Kaltenberger, M. Rupp, and G. Humer, "A scalable rapid prototyping system for real-time MIMO OFDM transmission," in *Proceedings of the 2nd IEE/EURASIP Conference on DSP Enabled Radio*, Southampton, UK, September 2005.

[50] R. Ernst, J. Henkel, and T. Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Design & Test*, vol. 10, no. 4, pp. 64–75, 1993.

[51] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[52] D. Henkel, J. Herrman, and R. Ernst, "An approach to the adaption of estimated cost parameters in the COSYMA system," in *Proceedings of the 3rd International Workshop on Hardware/Software Codesign (CODES '94)*, pp. 100–107, Grenoble, France, September 1994.

[53] J. Henkel and R. Ernst, "Hardware/software partitioner using a dynamically determined granularity," in *Proceedings of the 34th Annual Conference on Design Automation (DAC '97)*, pp. 691–696, Anaheim, Calif, USA, June 1997.

[54] A. Kalavade and E. A. Lee, "Global criticality/local phase driven algorithm for the constrained hardware/software partitioning problem," in *Proceedings of the 3rd International Workshop on Hardware/Software Codesign (CODES '94)*, pp. 42–48, Grenoble, France, September 1994.

[55] E. A. Lee, "Overview of the ptolemy project," Tech. Rep., University of Berkeley, Berkeley, Calif, USA, March 2001. http://ptolemy.eecs.berkeley.edu.

[56] B. Kernighan and S. Lin, "An efficient heuristic procedure in partitioning graphs," *Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970.

[57] A. Kalavade and E. A. Lee, "Extended partitioning problem: hardware/software mapping and implementation-bin selection," in *Proceedings of the 6th IEEE International Workshop on Rapid System Prototyping*, pp. 12–18, Chapel Hill, NC, USA, June 1995.

[58] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli, "System level hardware/software partitioning based on simulated annealing and tabu search," *Design Automation for Embedded Systems*, vol. 2, no. 1, pp. 5–32, 1997.

[59] F. Vahid and T. D. Le, "Extending the kernighan/lin heuristic for hardware and software functional partitioning," *Design Automation for Embedded Systems*, vol. 2, no. 2, pp. 237–261, 1997.

[60] K. S. Chatha and R. Vemuri, "Iterative algorithm for hardware-software partitioning, hardware design space exploration and scheduling," *Design Automation for Embedded Systems*, vol. 5, no. 3, pp. 281–293, 2000.

[61] K. S. Chatha and R. Vemuri, "Hardware-software partitioning and pipelined scheduling of transformative applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 3, pp. 193–208, 2002.

[62] T. Wiangtong, P. Y. K. Cheung, and W. Luk, "Comparing three heuristic search methods for functional partitioning in hardware-software codesign," *Design Automation for Embedded Systems*, vol. 6, no. 4, pp. 425–449, 2002.

[63] B. Knerr, M. Holzer, and M. Rupp, "Fast rescheduling of multi-rate systems for HW/SW partitioning algorithms," in *Proceedings of the 39th Annual Asilomar Conference on Signals, Systems, and Computers*, Monterey, Calif, USA, October 2005.

[64] B. Knerr, M. Holzer, and M. Rupp, "A fast rescheduling heuristic of SDF graphs for HW/SW partitioning algorithms," in *Proceedings of the 1st International Conference on Communication System Software and Middleware (COMSWARE '06)*, New Delhi, India, January 2006.

[65] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[66] T. C. Hu, "Parallel sequencing and assembly line problems," Tech. Rep. 6, Operations Research Center, Cambridge, Mass, USA, 1961.

[67] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee, "Scheduling precedence graphs in systems with interprocessor communication times," *SIAM Journal on Computing*, vol. 18, no. 2, pp. 244–257, 1989.

[68] L. Cai and D. Gajski, "Transaction level modeling in system level design," Tech. Rep., Center for Embedded Computer Systems, Irvine, Calif, USA, 2003.

[69] A. Haverinnen, M. Leclercq, N. Weyrich, and D. Wingard, "SystemC based SoC Communication Modeling for the OCP Protocol," Whitepaper, October 2002.

[70] U. Bortfeld and C. Mielenz, "C++ System Simulation Interfaces," Whitepaper, July 2000.

[71] J. Cockx, "Efficient modelling of preemption in a virtual prototype," in *Proceedings of International Workshop on Rapid System Prototyping (RSP '00)*, pp. 14–19, Paris, France, June 2000.

[72] S. Gupta, N. Dutt, R. Gupta, and A. Nciolau, "SPARK: a high-level synthesis framework for applying parallelizing compiler transformations," in *Proceedings of the 16th International Conference on VLSI Design*, pp. 461–466, New Delhi, India, January 2003.

[73] Y. Guo, D. McChain, and J. R. Cavallaro, "Rapid industrial prototyping and scheduling of 3G/4G SoC architectures with HLS methodology," *EURASIP Journal on Embedded Systems*, vol. 2006, Article ID 14952, 2006.

[74] C. A. Valderrama, A. Changuel, and A. A. Jerraya, "Virtual prototyping for modular and flexible hardware-software systems," *Design Automation for Embedded Systems*, vol. 2, no. 3-4, pp. 267–282, 1997.

M. Holzer received his Dipl.-Ing. degree in electrical engineering from the Vienna University of Technology, Austria, in 1999. During his diploma studies, he worked on the hardware implementation of the LonTalk protocol for Motorola. From 1999 to 2001, he worked at Frequentis in the area of automated testing of TETRA systems and afterwards until 2002 at Infineon Technologies on ASIC design for UMTS mobiles. Since 2002, he has a research position at the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms at the Vienna University of Technology.

B. Knerr studied communications engineering at the Saarland University in Saarbrücken and at the University of Technology in Hamburg, respectively. He finished the Diploma thesis about OFDM communication systems and graduated with honours in 2002. He worked for one year as a Software Engineer at the UZR GmbH & Co KG, Hamburg, on image processing and 3D computer vision. In June 2003, he joined the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms at the Vienna University of Technology as a Ph.D. candidate. His research interests are HW/SW partitioning, multicore task scheduling, static code analysis, and platform-based design.

P. Belanović received his Dr. Tech. degree in 2006 from the Vienna University of Technology, Austria, where his research focused on the design methodologies for embedded systems in wireless communications, virtual prototyping, and automated floating-point to fixed-point conversion. He received his M.S. and B.E. degrees from Northeastern University, Boston, and the University of Auckland, New Zealand, in 2002 and 2000, respectively. His research focused on the acceleration of image processing algorithms with reconfigurable platforms, both in remote sensing and biomedical domains, as well as custom-format floating-point arithmetic.

M. Rupp received his Dipl.-Ing. degree in 1988 at the University of Saarbrücken, Germany, and his Dr. Ing. degree in 1993 at the Technische Universität Darmstadt, Germany, where he worked with Eberhardt Häusler on designing new algorithms for acoustical and electrical echo compensation. From November 1993 until July 1995, he had a postdoctoral position at the University of Santa Barbara, California with Sanjit Mitra, where he worked with Ali H. Sayed on a robustness description of adaptive filters with impacts on neural networks and active noise control. From October 1995 until August 2001, he has been a member of the technical staff in the Wireless Technology Research Department of Bell Labs where he has been working on various topics related to adaptive equalization and rapid implementation for IS-136, 802.11, and UMTS. He is presently a Full Professor for digital signal processing in mobile communications at the Technical University of Vienna. He was an Associate Editor of IEEE Transactions on Signal Processing from 2002 to 2005, he is currently an Associate Editor of JASP EURASIP Journal of Applied Signal Processing, and of JES EURASIP Journal on Embedded Systems, and he is elected as AdCom Member of EURASIP. He authored and coauthored more than 200 papers and patents on adaptive filtering, wireless communications and rapid prototyping, as well as automatic design methods.