Synthesis of Embedded Image Processing Applications from SIMULINK Models

Andreas Doblander[1], Dietmar Gösseringer[1], Bernhard Rinner[1], and Helmut Schwabach[2]

[1]Institute for Technical Informatics,
Graz University of Technology, Graz, Austria
{doblander,goesseringer,rinner}@iti.tugraz.at

[2]Video & Safety Technologies,
ARC seibersdorf research GmbH, Seibersdorf, Austria
helmut.schwabach@arcs.ac.at

Abstract — *In next generation video surveillance systems there is a trend towards embedded solutions. A significant amount of processing power is needed for such complex applications. Therefore, digital signal processors (DSP) are often used to provide the necessary computational capabilities. But resources in embedded DSP systems are typically very limited which imposes significant challenges for software development. Resource constraints must be met while facing increasing application complexity and pressing time-to-market demands.*
Recent advances in synthesis tools for SIMULINK *suggest a feasible high-level approach to algorithm implementation for embedded DSP systems. The model-based visual development process of* SIMULINK *facilitates host-side simulation and validation, as well as synthesis of target specific code. Furthermore, legacy code written in* MATLAB *or ANSI C can be reused in custom blocks. However, the code generated for DSP platforms is often not very efficient.*
In this work the modeling and code generation capabilities within SIMULINK *are evaluated with respect to image processing algorithms. A motion detection algorithm is synthesized from a corresponding model. The resulting code targeted at a Texas Instruments TMS320C6416 DSP is compared to a hand-optimized reference implementation for the same processor. Results show that an ad hoc approach to synthesize complex image processing algorithms hardly yields optimal code for DSPs. However, several methods are presented that can improve performance of synthesized code.*

1 Introduction

In modern application domains digital signal processors (DSP) play an increasingly important role. This is due to a significant increase in DSP processing power which allows for increasingly complex applications. Video surveillance applications, e.g., show a trend

of integrating image acquisition and analysis with compression and network communication functionality into a single embedded device [1, 2]. High performance DSPs are often used to provide the needed processing power. Such complex configurations impose significant challenges on software development. Tight resource constraints have to be met while facing increasing application complexity and pressing time-to-market demands.

Most image analysis algorithms in video surveillance systems are very resource intensive. Additionally, several of such algorithms have to be integrated into one embedded device. Although modern DSPs offer substantial computational resources code optimization is mostly crucial for media applications [3].

Currently, image analysis algorithms are mostly explored using MATLAB. After that they are manually recoded in C or assembly language to meet performance goals on the target DSP. This is an error prone and cumbersome iterative process. Being rather time-consuming this recoding approach also directly conflicts with tight time-to-market objectives.

General purpose computing may give some hints to overcome most of these hindering matters. In hardware design as well as in software development commercial off-the-shelf (COTS) components are extensively used. This approach substantially improves reuse and decreases application integration time. Furthermore, sophisticated tools support developers in routine tasks or even automate entire development steps. Reuse, validation, and verification have to be facilitated also in embedded software development. A holistic approach based on validated and certified software components is promoted by [4]. The major challenge is to increase the level of abstraction in software development while meeting tight resource constraints.

Recently added support for DSP targets in the synthesis tools for SIMULINK simplifies high-level development for embedded DSP systems. The block-oriented modeling in SIMULINK is some kind of a light-weight component framework. It supports hierarchical modeling and, therefore, promotes reuse and addresses algorithmic complexity. Early validation of designs can be achieved by the simulation capabilities of SIMULINK. Additionally, the visual development environment boosts comprehensibility of large designs. To fully exploit model-based development synthesis tools are used to translate models to efficient code for the target platform. Special target specific blocks promise to synthesize rather efficient production code.

In this paper model-based development and synthesis using the SIMULINK environment are explored. The main focus of this investigation is on the development of image processing algorithms for embedded DSP applications. An evaluation of the quality of DSP code synthesized using the REAL-TIME WORKSHOP EMBEDDED CODER for SIMULINK is the main contribution of this work. It is shown that high-level modeling in SIMULINK is a feasible approach to software development for embedded DSP applications. Nevertheless, complex image processing algorithms still pose performance problems. This is because maximum control and flexibility of low-level programming languages are traded for an abstract system modeling approach. Furthermore, decisions concerning the implementation format (i.e. generic programming ensuring portability vs. target specific implementations yielding maximum efficiency) or modularization granularity also have a significant influence on performance. It is also shown that the lack of optimized block libraries most compromises performance. Furthermore, modularization and integration of

algorithms written in C or M-script (language of MATLAB) as custom SIMULINK blocks (i.e. as S-functions) is introduced as an interesting possibility to reuse legacy code.

The presented evaluation is limited to image processing applications on COTS DSP chips. Only single-processor settings are considered. To achieve comparable setups this work considers only a single algorithm system. Multitasking and multi-DSP systems are left out for future exploration.

The remainder of this paper is organized as follows. Section 2 presents related work concerning model-based development, component-based approaches, and code generation for embedded systems. In Section 3 the model-based development process is introduced . Basic development steps and their mutual interactions are sketched. Specifics of modeling in SIMULINK are illustrated. An evaluation of the development process for embedded image processing applications in SIMULINK is presented in Section 4. Modeling capabilities for image processing and the quality of the generated code are scrutinized. For the evaluation we used a simple motion detection algorithm to be run on a commercial DSP board. The code generated by SIMULINK was compared to a manually optimized C implementation of the same algorithm. Performance and memory usage of the two implementations are presented. A discussion of the experimental results and an analysis of possible improvements ends the section. A brief summary of this work and some concluding remarks conlude the paper in Section 5.

2 Related Work

Modeling and synthesis of digital signal processing algorithms and underlying models of computation (MoC) have been investigated extensively (e.g. in [5]). Data flow models are considered to best match the needs for digital signal processing applications.

In order to ease embedded software development the notion of *platform-based development* has been introduced [6]. This approach suggests a uniform development process for embedded software. Every system layer is built upon other grounding layers. All such layers can be seen as platforms for the layers using them (thus the term platform-based). Hierarchical modeling using abstract blocks as in SIMULINK is to some degree similar to this concept.

Component-based principles for embedded systems are developed by Schmidt et al. [7] in their CIAO framework. They present a RT-CORBA implementation for distributed real time and embedded (DRE) systems. The focus of their work is on aiding the programming of distributed systems and automatic component interface generation. In contrast to that the approach presented in this paper concentrates on code reuse and synthesis of algorithmic code. But both concepts provide some kind of a component framework.

Another component framework for real-time systems is suggested in [8]. They focus on reuse of legacy code which is also a major issue in this work. However, their approach also supports code generation only for component interfaces.

A rapid application development approach in the automotive domain is presented in [9]. There the authors describe the modeling of *Time Triggered Architecture* (TTA) using SIMULINK. In [10] the authors evaluate the synthesis capabilities of SIMULINK for automotive powertrain control. Although in automotive applications they have to meet hard real-time deadlines their overall performance requirements are far less than in image processing algorithms. An integrated modeling approach for audio signal processing using

SIMULINK and TI DSPs is discussed in [11].

Semiconductor vendors are also pressed to provide a significant software basis to ease application development for their processor chips. Texas Instruments, for example, introduced the *eXpressDSP Software Initiative* with a standard for algorithm development (XDAIS) [12] . XDAIS defines a framework which enables easy integration of compliant COTS algorithms from third parties. Such XDAIS algorithms are optimal candidates for use as custom blocks in SIMULINK.

Besides the synthesis tools related to SIMULINK (i.e. REAL-TIME WORKSHOP) there is, among others, another commercial product that supports code synthesis for TI DSPs. *Hypersignal RIDE* by Hyperception, Inc. is a visual block-oriented environment, too. It synthesizes code by linking optimized and pre-compiled target specific object code to an executable program [13]. In SIMULINK ANSI C is used as an intermediate language in the synthesis process.

3 High-Level Development for Embedded Image Processing

New development approaches are needed to tackle increasing application complexity while following pressing time-to-market requirements. Software reuse and easy integration of complex applications have to be encouraged. In this section model-based development in general and its application using the MATLAB/SIMULINK environment are presented. The applicability for image processing algorithms and suggested optimization techniques are illustrated.

3.1 The Model-Based Design Process

Generally, the model-based design approach is a generic development paradigm for embedded systems. It addresses system specification, synthesis of implementation, model analysis, validation and simulation, test and design evolution. Facing the various fields of applications of embedded systems—e.g. decision control systems, physical processes, signal processing—a single modeling language cannot be suitable for all systems. Rather, different more specific modeling approaches offering methods and syntaxes that are close enough to the particular application domain are employed [14, 15]. This heterogeneity of applications brought about the notion of domain-specific languages (DSLs).

For the domain of signal processing dataflow models are well suited and commonly used as the DSL. Modeling systems typically use a visual design approach such that modeling can be understood as visual programming where computational units are represented as blocks and data flow interconnections are represented as edges.

Figure 1 depicts the sequence of basic development phases of a design process for signal processing applications. Algorithmic design corresponds to the research phase and is usually done using a development environment for technical computing such as MATLAB that provides a high-level language for algorithm and data exploration. System level design starts with a well-defined and parameterized algorithm and transfers the specification into the domain of the modeling-language. This step corresponds to a generic implementation for embedded systems and replaces the coding phase of conventional software development process. Since the model is not only descriptive but also executable, continuous simulation helps to attain a validated reference. The aspect of having the semantics

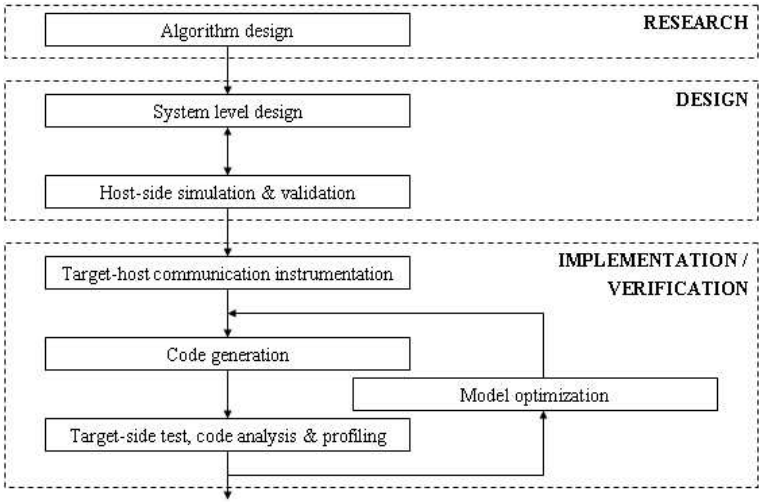

Figure 1: Development steps of a model-based design process

captured for both, the host and the target, is fundamental to model based-design. In the next phase the model can optionally be instrumented for target-side testing. Finally, the automatically generated program can be run on the target from within the model development environment. Test data can be parameterized and output data can be visualized on the fly. Criteria such as profiling results, CPU load, and code size determine if further model optimization steps are required. (For details on optimization techniques refer to Section 3.4.) This would entail the last design process steps to be performed iteratively until performance requirements are met.

3.2 Algorithm Development and Code Generation in SIMULINK

This section gives an insight of how the previously described development process is performed using MATLAB/SIMULINK and related tools by The Mathworks. Basically, all development phases as depicted in Figure 1, are performed using MATLAB, SIMULINK and the REAL-TIME WORKSHOP EMBEDDED CODER and EMBEDDED TARGET, respectively.

Image and video processing algorithms are elaborated using MATLAB as it is considered industry standard for this field of research. Further, environment variables containing simulation and test data such as images or videos which were prepared during algorithm development are directly accessible from SIMULINK. This fact eases the start into system-level design where SIMULINK serves as modeling and simulation environment. Models are composed of blocks that have inputs and outputs, parameters and states. Inputs and outputs can be interconnected if data format and bus width match. At any time during model-composition the model can be validated by simulation. A validation step preceding the simulation checks the model for syntactical correctness and logical integrity. During simulation the model's semantic behavior is tested on the host. Code generation for target-side testing is the main step of the implementation / verification phase. The synthesis is performed by the REAL-TIME WORKSHOP EMBEDDED CODER and EMBEDDED TARGET. If testing from within the MATLAB/SIMULINK environment is desired, the model can optionally be instrumented for target-to-host communication. This is simply done by

replacing data sources and sinks in the model by inputs and outputs provided by a target specific block library. Thereby testing is strongly simplified because test data can be uploaded onto the target, parameterization can be performed on the fly, and the calculated results can be visualized in MATLAB/SIMULINK. Model optimizations are realized via integration of optimized C code using the S-function interface. S-functions are a description of Simulink blocks. Their defined interface permits implementation in various programming languages such as C [16]. Generic ANSI C code optimizations require a wrapper S-function as an interface to SIMULINK. Target-specific embedded C optimizations need to be fully integrated into the generated code via the inlining mechanism provided by the Target Language Compiler (TLC). Its input files (TLC files) contain instructions that control code generation with the REAL-TIME WORKSHOP. Two implementations are necessary, one for simulation in SIMULINK as S-function and one for code generation by the REAL-TIME WORKSHOP as TLC file [17].

3.3 Applicability to Image Processing

Component-based development in embedded video and image processing is in its early stages. Taking into account the naturally high system resource requirements of algorithms in that domain, highly efficient implementations are needed to meet timing constraints of real-time systems. Unfortunately, the current state of modeling systems is that such highly efficient basic image processing functions are not or only sparsely provided. Due to the sequentiality of the data flow driven modeling approach, necessary functionality can hardly be efficiently modeled out of existing blocks. Lack of flexibility in modeling control flow and insufficient expression folding capabilities impose substantial limitations on the modeler. Regarding performance aspects loops are considered the most critical code segments. Most basic image processing operations are composed of loops or even nested loop constructs. Therefore dramatic performance losses can be expected from modeling those missing functions. Typically, powerful control structures and advanced memory indexing inside nested loop kernels would be needed to attain the desired performance.

An example for the deficiency of image processing support of the system design language is data processing on multiple image segments. That is that operations are not performed globally in one step on the whole image but individually on the tiled image segments. Those non-overlapping equally-sized regions of the image are called pixel blocks in the following. Simple operations such as average pixel value calculation — which corresponds to image downsampling by factors of powers of 2 —require modeling work-arounds.

Proposed work-around strategies for implementing repeated loop-like invocation of functional kernels include:

- Employment of control structure blocks realizing loops (if available).
- Implementation of a multi-rate system with the usage of helper blocks bearing the functional kernel, executing at subdivisions of the model's base rate.
- Matrix re-arrangement and re-composition with subsequent row- or column-wise data processing (if applicable).

All work-around variants cause significant performance losses in comparison to an implementation written in a traditional procedural programming language.

3.4 Model Optimization Techniques

The previous section illustrates the need to optimize models and to avoid work-around implementations in the domain of image and video processing. Optimizations performed directly on the output code are not a satisfactory choice since that would break the coherent mapping between model and generated code. All convenient development tools would not be applicable anymore and the model-based design process would be corrupted. Thus, output code modifications are not an option. As an alternative modeling systems like SIMULINK allow the system designer to add custom blocks to extend the built-in functionality. These blocks can be written in another programming language. This degree of freedom respects the rules of the development paradigm that need to be adhered.

When implementing custom modules in programming languages such as C, some considerations have to be taken in advance:

- *Function granularity*. Fine-grained modules with very basic functionality ensure high flexibility and reusability. Coarser-grained modules with naturally more specific functionality give the developer more flexibility for optimizations since high-level dependencies can be exploited. Techniques such as folding operations within loop kernels cannot be applied in the model due to the inherent sequentiality of dataflow representations.
- *Optimization level*.
  - A *generic* implementation in platform independent ANSI C ensures platform portability, flexibility in control flow, usage of efficiently nested loops and operations folding guarantee to attain significant performance gains.
  - Implementations in *target specific* embedded C making use of proprietary intrinsics and compiler directives permit to obtain maximum performance gains. This format requires a second implementation, though, for the runtime environment of the host since it cannot interpret the target specific commands.

4 Experimental Evaluation of the SIMULINK Development Tools

4.1 Experimental Setup

The practical experiments on which this section is based were performed using THE MATHWORKS MATLAB/SIMULINK Release 13 modeling environment. As a code generator the REAL-TIME WORKSHOP EMBEDDED TARGET was used. A Texas Instruments C6416 DSP Starter Kit was used as the target hardware platform.

The comparative profiling results presented in Section 4.2 were performed on different implementations of a motion detection algorithm based on frame differencing. For the evaluation we compared four different cases:

(i) *Reference.* A hand-optimized C implementation exploiting the TI 'C64x DSP's special capabilities. This implementation was taken as performance reference. Optimization was achieved by directly addressing the DSP's instruction set using intrinsic commands, its architecture using data packing for maximizing bus width usage and data throughput, and its software pipeline by enabling compiler loop optimizations through specifying loop properties.

(ii) *Unoptimized model.* The synthesized implementation of the ad hoc algorithm modeling, corresponding to the modeling results after the first iteration of the development process presented in Section 3.1. No optimizations were applied.

(iii) *Generic optimizations.* The synthesized implementation from the model where generic optimizations were applied. Target independent ANSI C S-functions were integrated.

(iv) *Target-specific optimizations.* The synthesized implementation of the model where target specific blocks were used. Adapted code segments from the reference *(i)* written in C were integrated.

In order to sketch the algorithm used for evaluation, the major building blocks can be identified as:

(a) Image *downsampling* by calculating the average of pixel blocks (i.e. regions of an image).

(b) *Buffering* the current and *unbuffering* the previous downsampled image for comparison of consecutive video frames.

(c) Detecting blocks where *motion* has occurred, i.e., blocks where the sum of absolute differences (SADs) is greater than a pre-defined threshold.

(d) Determining if a *pre-alarm* has to be set. That is, summing up the number of motion blocks, considering only those blocks which are set active by a parameter mask, and comparing the result with a threshold.

4.2 Code Profiling Results

Profiling was performed using a video format with a resolution of 368 x 272 pixels and a block size of 8 x 8 pixels. Results are presented in Table 1 and Table 2. The overall consumed CPU cycles are broken down into functional modules for a more detailed view.

We measured a tremendous performance loss of factor 44 for the generated code of the initial unoptimized model *(ii)* compared to the reference *(i)*. This made clear that further optimization was necessary. With the first level of optimization employing ANSI C S-functions *(iii)* the execution time could be reduced to 18% in comparison to the unoptimized model *(ii)*. Still, a performance degradation of about a factor of 8 remains as compared to the reference *(i)*. With the second level of optimization utilizing embedded C segments from the reference we obtained performance results close to those of the reference *(i)*. We measured a 2% overhead between these two implementations. The performance speedup towards the generic optimization *(iii)* amounts to a factor of 7,88.

The reasons for the poor performance of the generated implementation from the unoptimized model *(ii)* are mainly due to the limitations of the modeling language. Operations are performed sequentially in multiple loops, whereas they are folded within one loop in the optimized version. Expression folding within loops enables compilers of single instruction multiple data (SIMD) processors such as the TI C6416 DSP to parallelize instructions resulting in further performance gain. Output code of an unoptimized model is not suitable for compiler optimizations. Additionally, blocks that are only used to interface special function blocks in the model often result in unnecessary code in the synthesis process. For example, a matrix reshape block that is only needed to couple blocks with

Module	*reference implementation (i)*	*non-optimized model (ii)*	*generically optimized model (iii)*	*target specific optimized model (iv)*
downsampling (a)	628,752	27,665,088	5,030,016	639,014
buffer/unbuffer (b)	1,408	37,876	1,592	1,592
detectMotionBlocks (c)	1,520	85,024	26,832	1,536
preAlarm (d)	432	44,116	3,296	448
Overall	632,112	27,832,104	5,061,736	642,590

Table 1: Profiling results in CPU cycles

	reference implementation (i)	*non-optimized model (ii)*	*generically optimized model (iii)*	*target specific optimized model (iv)*
reference implementation (i)	1.00	0.02	0.12	0.98
non-optimized model (ii)	44.03	1.00	5.50	43.35
generically optimized model (iii)	8.01	0.18	1.00	7.88
target specific optimized model (iv)	1.02	0.02	0.13	1.00

Table 2: Comparison of profiling results (ratios of CPU cycles)

different data dimensions results in creation of a replicate. For the algorithm this operation is redundant and can impose significant performance overhead. Another reason for the observed inefficiency is that memory copying is performed in an element-by-element manner. Therefore, even simple functional blocks such as buffering and unbuffering—although not relevant for the overall performance—result in an enormous performance loss of about factor 24.

Data memory consumption of the different examined cases differs only slightly. In all implementations memory for the input video frames makes up about 85% of the total data memory consumption. The rest is divided among several buffers holding downsampled video frames for performing the frame differencing. Contrary to data memory, program code size differs significantly between the reference implementation and the automatically generated versions. The generated model executables require more than two times the amount of program memory needed by the reference implementation.

4.3 Discussion

The profiling results of non-optimized models illustrate the weakness of today's modeling systems in the field of image processing. As already stated previously, the lack of domain specific function libraries and the limitations in modeling low-level functions efficiently necessitate extensibility of the modeling system by integrating custom code. We

Module	*reference implementation (i)*	*non-optimized model (ii)*	*generically optimized model (iii)*	*target specific optimized model (iv)*
Code size	148	345	346	346
Data memory	111	115	116	116

Table 3: Memory consumption in KB

find that the performance differences between the reference and the optimized models are acceptable in favor of the advantages of the system-level design-flow that provides a good framework for modular design. Further it ensures reusability, serves as test and simulation environment and furnishes executable specifications. The component-based development paradigm also takes development of embedded systems one step further as it meets increasingly pressing time-to-market demands. Still many constructs are missing that possibly have the potential to improve performance of image and video processing on embedded DSP platforms. Nevertheless, we believe that the current deficiencies together with the market's needs will boost developments in this field.

5 Conclusion

Increasing application complexity and time-to-market requirements in the domain of embedded image processing demand for efficient software development strategies. Recently, the synthesis tools for SIMULINK were extended to support code generation for Texas Instruments DSPs. Motivated by their availability an evaluation of these tools for design and implementation of image processing algorithms was conducted.

It was indicated in this paper that model-based design is a feasible approach for embedded DSP applications. Reusability and maintainability of the software are promoted by the high-level design. Combined with automatic synthesis capabilities overall development time can be reduced. However, complex embedded video surveillance applications require rather efficient algorithm implementations. A demand that often cannot be satisfied by automatic synthesis from simple models. To meet tight resource constraints on the target optimization of the model is essential. Expressing special DSP features directly in the model can boost performance dramatically.

Of course, automatically generated code will not yield the same degree of optimization that an experienced developer can reach manually. But establishing a comprehensive library of domain specific functional blocks allows synthesis of code that at least exhibits only little performance degradation.

To retain reusability and portability, however, additional work has to be done. Blocks have to be implemented in both, a generic and a target specific version, respectively. The latter serves for taking full advantage of the DSP's architecture while the former allows simulation and preserves portability between different target platforms. Platform specific details have to be adapted only if a different target hardware is used and performance optimizations are needed. Otherwise no major modifications are necessary.

Code profiling experiments were conducted to evaluate the synthesis capabilities of SIMULINK for Texas Instruments TMS320C64x DSPs (i.e. REAL-TIME WORKSHOP EMBEDDED TARGET). A manually optimized implementation of a simple motion detec-

tion algorithm was compared to code generated from SIMULINK models. Several model variants were examined to illustrate possible optimization techniques in the model that improve the synthesized code.

5.1 Further Work

Future work includes further investigation of code generation capabilities of the REAL-TIME WORKSHOP EMBEDDED CODER. Plans are to implement several higher level image processing algorithms, e.g. a stationary vehicle detection algorithm (SVD), as custom SIMULINK blocks. These blocks can be combined to build more complex image processing applications. A high-level framework for integrating several XDAIS algorithms[1] into a single system is desired.

Furthermore, the implementation of specialized blocks for target resource management (e.g. DMA) are needed to allow better memory and bus transfer management. A significant speed-up in image data transfers is anticipated using such target specific blocks. As a further extension to current modeling capabilities multi-DSP support can be imagined to be incorporated. Therefore, some means for distributing specific algorithms to different processors have to be provided in the modeling environment. A possible approach could again be specialized blocks. In this case they would have to represent linking and loading of target code rather than algorithmic computation.

References

[1] W. Wolf, B. Ozer, and T. Lv. Smart cameras as embedded systems. *IEEE Computer*, 35(9):48–53, September 2002.

[2] C. S. Regazzoni, V. Ramesh, and G. L. Foresti. Introduction of the special issue. *Proceedings of the IEEE*, 89(10), October 2001.

[3] Kerem Karaday, Vishal Markandey, Robert J. Gove, and Yongmin Kim. Strategies for mapping algorithms to mediaprocessors for high performance. *IEEE Micro*, 23(4):58–70, July–August 2003.

[4] Alberto Sangiovanni-Vincentelli and Grant Martin. A vision for embedded software. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2001)*, Atlanta, Georgia, USA, 2001.

[5] Marco Sgroi, Luciano Lavagno, and Alberto Sangiovanni-Vincentelli. Formal models for embedded system design. *IEEE Design & Test of Computers*, 17(2):14–27, April–June 2000.

[6] Alberto Sangiovanni-Vincentelli. Defining platform-based design. EEDesign Magazine of EETimes (EEDesign.com), February 2002.

[7] Krishnakumar Balasubramanian, Nanbor Wang, Chris Gill, and Douglas C. Schmidt. Towards composable distributed real-time and embedded software. In *Proceedings of the 8th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003)*, Guadalajara, Mexico, 2003.

[8] Egon Teiniker, Stefan Mitterdorfer, Christian Kreiner, Zsolt Kovács, and Reinhold Weiss. Local components and reuse of legacy code in the corba component model. In *Proceedings of the 28th Euromicro Conference (EUROMICRO'02)*. Institute for Technical Informatics, Graz University of Technology, IEEE, 2002.

[9] R. Obermaisser and P. Peti. A framework for rapid application development of distributed embedded real-time systems. In *Proceedings of the IEEE Region 8 EUROCON 2003. Computer as a Tool.*, Ljubljana, Slovenia, 2003.

[1] That is, algorithm implementations compliant with the Texas Instruments interface definitions for reusable softtware components. For more details refer to [12].

[10] David Wybo and David Putti. A qualitative analysis of automatic code generation tools for automotive powertrain applications. In *Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design*, Kohala Coast-Island of Hawaii, Hawaii, USA, 1999.

[11] K.H. Hong, W.S. Gan, Y.K. Chong, K.K. Chew, C.M. Lee, and T.Y. Koh. An integrated environment for rapid prototyping of dsp algorithms using matlab and texas instruments' TMS320C30. *Microprocessors and Microsystems*, 24(7):349–363, November 2000.

[12] Texas Instruments. *TMS320 Algorithm Standard—Rules and Guidelines*. Texas Instruments, October 2002. Literature Number: SPRU352E.

[13] Hyperception, Inc. Hypersignal ride product website, 2004. http://www.hyperception.com/RIDE/.

[14] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-integrated development of embedded software. *Proceedings of the IEEE*, 91(1):145–164, January 2003.

[15] S. Sastry, J. Sztipanovits, R. Bajcsy, and H. Gill. Scanning the issue - special issue on modeling and design of embedded software. *Proceedings of the IEEE*, 91(1), January 2003.

[16] The MathWorks. *Simulink—Model-Based and System-Based Design*. The MathWorks, September 2003. Writing S-Functions—Version 5.

[17] The MathWorks. *Target Language Compiler—For Use with Real-Time Workshop*. The MathWorks, July 2002. Reference Guide—Version 5.