International Journal of Software Engineering and Knowledge Engineering
© World Scientific Publishing Company

AN EVALUATION OF MODEL-BASED SOFTWARE SYNTHESIS FROM SIMULINK MODELS FOR EMBEDDED VIDEO APPLICATIONS

Andreas Doblander, Dietmar Gösseringer, Bernhard Rinner

Institute for Technical Informatics, Graz University of Technology,
Inffeldgasse 16/1, 8010 Graz, Austria
{doblander, goesseringer, rinner}@iti.tugraz.at
http://www.iti.tugraz.at

Helmut Schwabach

Video & Security Technologies, ARC Seibersdorf Research,
2444 Seibersdorf, Austria
helmut.schwabach@arcs.ac.at

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

In next generation video surveillance systems there is a trend towards embedded solutions. Digital signal processors (DSP) are often used to provide the necessary computing power. The limited resources impose 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 high-level approach to algorithm implementation for embedded DSP systems. The model-based visual development process of Simulink facilitates simulation as well as synthesis of target specific code. In this work the modeling and code generation capabilities of Simulink are evaluated with respect to video analysis algorithms. Different models of a motion detection algorithm are used to synthesize code. The generated code targeted at a Texas Instruments TMS320C6416 DSP is compared to a hand-optimized reference. Experiments show that an ad hoc approach to synthesize complex image processing algorithms hardly yields optimal code for DSPs. However, several optimizations can be applied to improve performance.

Keywords: model-based design; automatic code generation; embedded video surveillance.

1. Introduction

Video surveillance systems 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 (digital signal processors) are often used to provide the needed processing power. Such complex configurations impose significant challenges on the software development. Tight resource constraints have to be met while facing increasing application complexity and pressing time-to-market demands. Although modern DSPs offer substantial computational resources code optimization is mostly crucial for media applications[3,4].

It is a major challenge in embedded software development to increase the level of abstraction while meeting tight resource constraints[5]. Recently added support for DSP targets in the synthesis tools for Simulink[6] simplifies high-level development for such platforms. Block-oriented modeling also supports hierarchical designs that promote reuse and address algorithmic complexity. Validation by simulation is already available in early development stages. Synthesis tools in combination with target specific components are used to generate production code for the embedded platform.

In this paper model-based development and synthesis using the Simulink environment are explored. We intend to integrate synthesized algorithmic code into an intelligent embedded multi-DSP camera for video surveillance. An evaluation of the quality of DSP code synthesized using the Real-Time Workshop Embedded Coder (RTW-EC) for Simulink is the main contribution of this work.

2. Related Work

There are a number of approaches to high-level or model-based embedded software development. Platform-based development[7], e.g., suggests a layered design with well defined interfaces. Other popular methodologies are model-driven architecture (MDA)[8] and UML-based methods[9]. All of these approaches raise the level of abstraction and employ code synthesis which is also a key benefit of modeling in Simulink.

Component-based principles for distributed embedded systems are widely discussed in the literature (e.g., Schmidt *et al.*[10] in their CIAO framework). The focus of their work is on supporting distributed application programming and automatic component interface generation. In contrast, the approach used in this paper concentrates on code reuse and synthesis of algorithmic code.

In Wybo and Putti[11] the synthesis capabilities of Simulink are evaluated for automotive powertrain control. Although most automotive applications have to meet hard real-time deadlines their overall performance requirements are significantly smaller than in video analysis. Similarly, an integrated modeling approach for audio signal processing using Simulink and Texas Instruments DSPs is discussed by Hong *et al.*[12].

Whalen and Heimdahl[13] discuss requirements and problems of automatic code generators for safety-critical systems. A special code generation environment for such systems is described in Kim and Lee[14].

3. High-Level Development for Embedded Video Surveillance Applications

Model-based design is a generic development paradigm that addresses system specification, validation by simulation, model analysis, synthesis, and test. The key idea is to build a model that satisfies the requirements and to use this model (or automatically transformed models) for all further development steps including code generation.

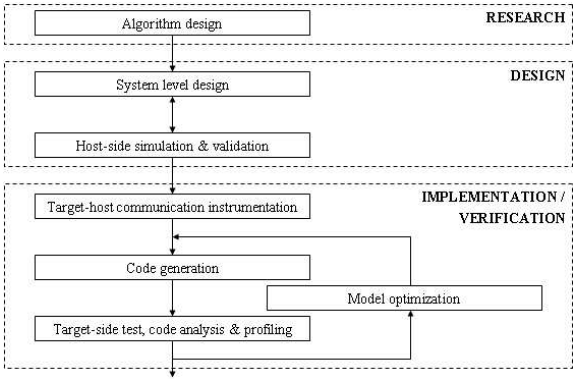

Fig. 1. Generic model-based development process[6,15].

In Fig. 1 the basic steps of a model-based development process for video analysis applications are illustrated. Basically, all development phases depicted in Fig. 1 are supported by Matlab/Simulink together with the Real-Time Workshop Embedded Coder (RTW-EC) and the Embedded Target for Texas Instruments C64x DSPs (ET). Algorithmic design (research phase) is usually performed using a high-level development environment such as Matlab or C/C++ libraries. System level design translates the well-defined algorithms into the domain of the modeling-language. Simulation is employed to maintain a validated reference. Optionally, the model can be instrumented for target-side testing. Finally, code is synthesized and can be executed on the target.

Taking into account timing constraints and resource requirements of algorithms in the video surveillance domain, highly efficient code is needed. Unfortunately, current modeling systems do not provide special video analysis function blocks that yield efficient DSP code. Algorithms have to be modeled by intricate compositions of simple blocks that are often transformed to suboptimal code. Therefore, optimizations are needed to generate efficient code. Modifications on synthesized code are not an option. They would break up the mapping between model and generated code such that the model-based design process would be corrupted. The only choice is to optimize the model.

For that purpose tools like Simulink provide mechanisms to extend their built-in functionality. Such blocks can be written in a traditional programming language (e.g. C/C++). When implementing custom modules to improve efficiency of synthesized code one has to consider two important issues.

(i) Function granularity. Fine-grained modules with very basic functionality ensure high flexibility and reusability. Coarser-grained modules, however, offer better opportunities for optimizations by the compiler[4].

(ii) Optimization level. Generic implementations in ANSI C ensures platform portability. Target specific C implementation, i.e., C plus *intrinsic* instructions and compiler directives, on the other hand, make use of proprietary hardware features, e.g., direct memory access (DMA) and, therefore, yield performance gains.

4. Experimental Evaluation

Experiments were conducted using the Matlab/Simulink R13 / R14 modeling environment. A Texas Instruments C6416 DSP Starter Kit together with the Code Composer Studio v2.21 IDE (CCS) was used as the development environment. The following four different implementations of a motion detection algorithm (MD) with different levels of optimization were profiled and compared:

(i) Reference. A manually coded, hand-optimized C implementation exploiting the hardware capabilities of the TI C64x DSP.

(ii) Unoptimized model. The implementation synthesized from an ad hoc model without special optimization.

(iii) Generic optimizations. An implementation synthesized from a model where generic optimizations were applied. Target independent ANSI-C constructs were integrated. Substantial improvements are achieved by extracting iterative parts of the model and put them into custom blocks implemented in C. Nesting loops to support compiler optimizations is a very promising approach here.

(iv) Target-specific optimizations. An implementation synthesized from a model where target specific custom blocks were used. Adapted code segments from the manually coded reference (i) written in C were integrated.

For the tests a video format with a resolution of 368 x 272 pixels was used. A model was created for each case listed above. From each model code was generated and imported to CCS for profiling. Profiling is also supported in the Simulink environment but CCS offers more control over the profiling process. Results from CPU load profiling are summarized in Tab. 1.

Tab. 1. Profiling results in CPU cycles split among different parts of the algorithm.

Module	Reference implementation (i)	Non-optimized model (ii)	Generically optimized model (iii)	Target-specific optimized model (iv)
Downsampling	628752	27665088	5030016	639014
Buffering/Unbuffering	1408	37876	1592	1592
Sum of absolute differences	1520	85024	26832	1536
Generate pre-alarm	432	44116	3296	448
Overall	632112	27832104	5061736	642590

Tab. 2. Memory consumption in KB.

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

Data memory consumption of the different examined cases differs only slightly (cf. Tab. 2). Input video frame buffers make up about 85% of the total data memory consumption. The rest is used for past frames for frame differencing. The synthesized

executables require more than twice the program memory of the reference implementation.

5. Discussion

Code generated from the initial unoptimized model (ii) was 44 times slower than the reference (i). With generic optimizations (iii) the execution time could be reduced to 18% compared to the unoptimized model (ii). But this code was still eight times slower than the reference (i). The second level of optimization utilized embedded C segments from the reference as a simple target specific optimization. Now the obtained performance was similar to that of the reference (i). Only an overhead of about two percent is still imposed by the modeling environment. With generic optimization (iii) the code was about 7.88 times slower than the code with target specific optimizations (iv).

An important reason for the poor performance of the code generated from the unoptimized model (ii) is that generated code often contains multiple sequential loops. In the manually optimized version operations are compacted to a single loop. Compilers are then able to better parallelize instructions resulting in a performance gain. Additionally, blocks that are only used to interface special function blocks in the model often result in redundant code in the synthesis process.

Another reason for performance losses is the inefficient use of the memory subsystem. Even simple functional blocks such as buffering and unbuffering can lead to degraded performance of synthesized code. Using DMA features of the target processor could substantially increase data transfer performance. Especially, video analysis algorithms benefit from DMA because a lot of data has to be transferred from and to memory. It is almost always possible to fully load the CPU while DMA transfers are performed in the background.

It is also shown that the use of specialized DSP instructions improves performance. In Simulink such specialized code is incorporated into the model via custom blocks that directly make use of target specific instructions or call into optimized (assembly) libraries. Of course, there is a one-time overhead for implementing such custom blocks. However, they can then be easily reused in similar projects without substantial effort.

6. Conclusion

Code profiling experiments of a motion detection algorithm indicate that model-based design is a promising approach for embedded video surveillance applications. Reusability and maintainability of the software are promoted by the high-level design. However, complex embedded video surveillance applications with their resource limitations require rather efficient algorithm implementations that often cannot be satisfied by code generation from simple models. Therefore, model optimization is necessary because optimizing the generated code would corrupt the model-based development flow.

In Simulink custom blocks can be used for model optimizations. Generally, there are two major optimization levels. First, generic optimization concentrates on hardware

independent model modifications. Target specific modifications, on the other hand, are not easily portable to different hardware any more. Nevertheless, they yield the most performance improvements in the synthesized code. To maximize the use of custom optimizations a domain specific library of custom blocks can be created and reused for similar projects.

Future work concentrates on the use of the code generation framework of Simulink to synthesize algorithmic code for an embedded smart traffic surveillance camera[4]. The code generation templates have to be adapted so that generated code can be used as dynamically loadable modules in the software framework that is currently developed.

References

1. W.Wolf, B. Ozer and T. Lv. Smart cameras as embedded systems. *IEEE Computer* 35:9(2002) 48—53.
2. C. S. Regazzoni, V. Ramesh and G. L. Foresti. Introduction of the special issue. *Proceedings of the IEEE* 89:10(2001) 1355—1539.
3. K. Karaday, V. Markandey, R. J. Gove and Y. Kim. Strategies for mapping algorithms to mediaprocessors for high performance. *IEEE Micro* 23:4(2003) 58—70.
4. M. Bramberger, J. Brunner, B. Rinner, and H. Schwabach. Real-time video analysis on an embedded smart camera for traffic surveillance. In *Proc. 10th IEEE Real-Time and Embedded Technology and Applications Symposium* (2004), pp. 174—181.
5. A. Sangiovanni-Vincentelli and G. Martin. A vision for embedded software. In *Proc. Int. Conf. on Compilers, Architecture, and Synthesis for Embedded Systems* (2001), pp. 1—7.
6. The MathWorks, Inc. The mathworks website (2005), http://www.mathworks.com.
7. A. Sangiovanni-Vincentelli. Defining platform-based design. *EEDesign Magazine* (Feb. 2002), http://EEDesign.com.
8. Object Management Group, Model Driven Architecture (2005), http://www.omg.org.
9. T. Schattkowsky and W. Mueller. Model-Based Specification and Execution of Embedded Real-Time Systems, In *Proc. Of the Design, Automation and Test in Europe Conference and Exhibition* (2004), pp. 1392—1393 Vol. 2.
10. K. Balasubramanian, N. Wang, C. Gill and D. C. Schmidt. Towards composable distributed real-time and embedded software. In *Proc. 8th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems* (2003), pp. 226—233.
11. D. Wybo and D. Putti. A qualitative analysis of automatic code generation tools for automotive powertrain applications. In *Proc. 1999 IEEE Int. Symp. on Computer Aided Control System Design* (1999), pp. 225—230.
12. 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. *J. Microprocessors and Microsystems* 24(2000) 349—363.
13. M. W. Whalen and M. P.E. Heimdahl. On the Requirements of High-Integrity Code Generation. In *Proc. of the 4th IEEE International Symposium on High-Assurance Systems Engineering* (1999), pp. 217—224.
14. J. Kim and I. Lee. Modular Code Generation from Hybrid Automata based on Data Dependency. In *Proc. 9th IEEE Real-Time and Embedded Technology and Applications Symposium* (2003), pp. 160—168.
15. G. Karsai, J. Sztipanovits, A. Ledeczi and T. Bapty. Model-integrated development of embedded software. *Proceedings of the IEEE* 91:1(2003) 145—164.