AN INTERACTIVE MODEL RE-CODER FOR EFFICIENT SOC SPECIFICATION*

Pramod Chandraiah and Rainer Dömer
Center for Embedded Computer Systems
University of California Irvine
pramodc@uci.edu, doemer@uci.edu

Abstract To overcome the complexity in System-on-Chip (SoC) design, researchers have developed sophisticated design flows that significantly reduce the development time through automation. However, while much work has focused on synthesis and exploration tools, little has been done to support the designer in writing and rewriting SoC models. In fact, our studies on industrial size examples have shown that about 90% of the system design time is spent on coding and re-coding of SLDL models, even in the presence of algorithms given in the form of C code. Since the quality of the design model has tremendous impact on the cost and quality of the resulting system implementation, creating and optimizing the model is a critical task toward successful SoC design.
In this paper, we present an interactive source re-coder which integrates static analysis and code transformation tools into an editor to assist the designer in tedious modeling and optimization tasks. This novel approach allows the designer to use her/his limited modeling time efficiently, and thus yields significant gains in productivity.

Keywords: System-level Design, Specification Modeling, Embedded systems, System-on-Chip

1. Introduction

In the past, the system-level design community has focused on solving various problems of system synthesis. Researchers have been working towards design automation at various abstraction levels with the goal to automate steps in the design process and reduce the design time. Motivated by the need to meet the time to market and aggressive design goals like low power, high performance, and low cost, researchers have

*This work was supported in part by Nicholas Endowment through the Henry T. Nicholas III Research Fellowship.

proposed various design methodologies for effective design development, including top-down and bottom-up approaches. All these technological advances have significantly reduced the development time of embedded systems. However, design time is still a bottleneck in the production of systems, and further reduction through automation is necessary. One critical aspect neglected in optimization efforts so far is the design specification phase, where the intended design is captured and modeled for use in the design flow.
Each design methodology expects a specific type of input model and most methodologies depend on intermediate design models for interaction between tools and the designer. The specification needs to be either hand-written from scratch, or modified from a reference model. While much of the research has focused on SoC synthesis and refinement tools, little has been done to support the designer in forming these models.

1.1 Motivation

In order to study the intricacies and complications involved in writing a system specification, we have applied a top-down design methodology, as shown in Figure 1, to the example of a multimedia application, a MP3 audio decoder. Here, the design process starts with an abstract specification model which is then refined to create models at lower abstraction levels, including transaction-level, bus-functional and implementation models. After a series of refinement steps, an actual implementation model is finally derived. Each of the refinement steps in the design flow is automated to the extent that model generation is fully automatic, and the designer has to only make the design decisions such as component allocation, mapping and scheduling. Due to this automation, we were able to implement our MP3 decoder model, an industry-size application, in less than a week [1]. In contrast, manually re-coding the reference implementation into a specification model took 12-14 weeks. Writing and re-writing this model was

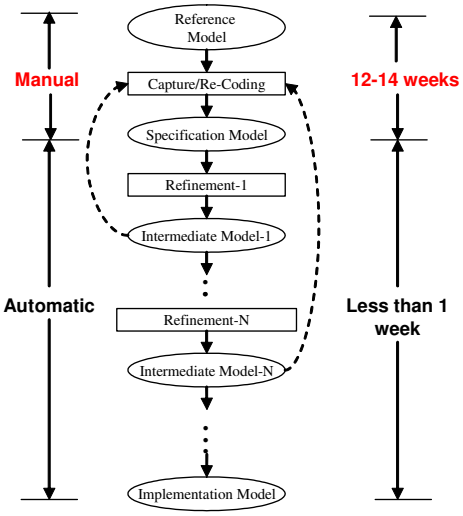

Figure 1. Motivation: Design time of MP3 decoder in a refinement-based design flow.

the main bottleneck of the whole process. More than 90% of the overall design time was spent in creating the specification model.
Also, we need to emphasize that specification capturing is not a one time task. Every time a change in the design is required for a successful refinement step, it is necessary to re-code/change the input specification (as shown by back arrows in Figure 1), making the whole task of specification writing iterative. Such interruptions in the design flow cause costly delays. The problem of lengthy re-coding of models is not a problem specific to a top-down methodology. Its importance is also emphasized in [12, 6].
In conclusion, any step towards automation of model coding and re-coding is highly desirable and will likely improve overall design time significantly.
In Section 2, we discuss model re-coding and related work. In Section 3, we present our solution to the modeling problem, an interactive source re-coder. Section 4 lists our experimental results. Finally, we will draw conclusions and outline future work in Section 5.

2. Model Re-Coding

Reference models of the algorithm obtained from software vendors, standardizing committees (eg. ISO/IEC) or similar sources act as a good starting point for creating the SoC specification. Depending on the design flow, these reference models must be recoded in System Level Description Languages (SLDLs) such as SystemC [7], SpecC [4] or SystemVerilog [18]. Apart from the recoding of the C model into a SLDL model, various modeling guidelines recommended by the design flow must also be incorporated into the model. Typical modeling guidelines [5] include, clear separation of communication and computation, sufficient computational granularity, and exposing concurrency.

2.1 Automated Re-Coding

Apart from the time consuming textual operations, re-coding a system model involves a lot of decision making. Many of these decisions can only be taken by the designer. For example, if the designer decides to map a C function onto a separate hardware component, the model needs to be re-coded to encapsulate this function in a separate block (behavior/module). In the absence of efficient hardware/software partitioning tools, this decision needs to be taken by the designer. However, the tedious recoding of the model can be automated.
To leverage the idea of re-coding automation, we need to distinguish re-coding tasks that can be automated from decision tasks that require

designer's experience. Textual re-coding operations such as introducing blocks, changing scope of variables, grouping functions, etc. can be performed automatically, if the decision is made by the designer. By such automation, the designer is relieved from mundane text editing tasks and can focus on actual modeling decisions. To address such issues, it is efficient to have a re-coder that is interactive and applies the changes on the fly. Since the model being derived is under full control of the designer, the output of such a tool can suit many design flows based on similar C-like languages.

2.2 Related Work

In this section, we will briefly present some work related to re-coding. First, we will look at the general area of program transformations and then focus on interactive ones.

2.2.1 Program transformations. In the past, researchers have developed program transformations for many different areas, including to improve aesthetics, to parallelize applications, and to perform high level synthesis (HLS). For example, the SUIF compiler [9] identifies loop level parallelism in a program and transforms a sequential program into a single-program, multiple data program. The Spark HLS framework [8] applies source and low level parallelizing transformations to a design to improve the quality of the target hardware. Unlike SUIF and Spark, our re-coder specifically aims at re-coding a C reference implementation into models in SLDLs suitable for design space exploration and system synthesis. The SpecSyn [3] synthesis system expects input specification in the SpecCharts language, but provides no facilities to create the initial SpecChart model. Our transformations are interactive and give the designer complete control ("designer-in-the-loop") to code/re-code the C reference model in order to arrive at the most suitable design implementation.

2.2.2 Interactive program transformations. To compare existing interactive coding environments and their capabilities, we distinguish textual abilities, syntax awareness, semantics awareness, analysis and program transformation capabilities. Figure 2 shows a set of tools and the extent to which they meet these capabilities. A simple *text-aware* editor, such as Textpad, supports plain textual entry and basic formatting. Better editors are syntax aware. By *syntax awareness* we mean the ability to recognize syntactical elements of the language. For example, highlighting of keywords and matching of braces requires syntax awareness. A *semantics- aware* editor provides advanced features

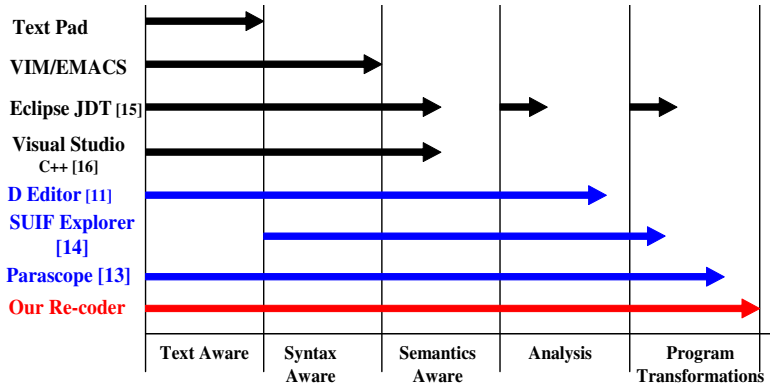

Figure 2. Capabilities of Interactive coding environments.

like context-assist, auto-complete, error indications, etc. Examples include Eclipse Java Development Tool (JDT) [15], and Microsoft Visual studio [16]. In addition to syntax and semantics, the Fortran D editor [11] is equipped to provide dependency analysis and other information about parallelism and communication.
The ParaScope editor [13] for Fortran and SUIF explorer [14] for C/Fortran provide program transformations to parallelize a program and relieve the programmer from tedious manual typing. SUIF explorer provides graphical means of setting compiler directives, but does not support editing. ParaScope provides the user with powerful interactive program transformations and reconstructs the dependency information, incrementally, while editing. Of all, ParaScope combines the most features. Our goal is to build similar and more advanced capabilities into our source re-coder, aiming specifically at analysis and transformations necessary to re-code a C reference source into a SLDL model. It is also necessary to mention that our transformations are generic and will not address specific transformations such as reimplementing data structures, replacing a slow algorithm with a faster implementation, and so on. However, such complex transformations can be realized by the designer using the set of generic transformations provided by our re-coder.

3. Interactive Source Re-Coder

To aid the designer in coding and re-coding, we propose a source *re-coder*. Our source re-coder is a controlled, interactive approach to implement analysis and refinement tasks. In other words, it is an intelligent union of editor, compiler, and powerful transformation and analysis tools.

Unlike other program transformation tools, our re-coder keeps the designer in the loop and provides complete control to generate and modify a model suitable for her/his design flow. By making the re-coding process interactive, we rely on the designer to concur, augment or overrule the analysis results of the tool, and use the combined intelligence of the designer and the re-coder for the modeling tasks.

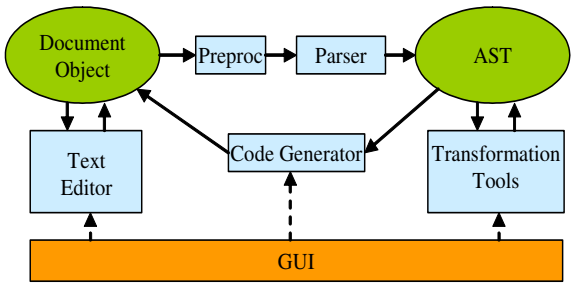

Figure 3. Conceptual Structure of the Source Re-Coder.

Our re-coder supports re-modeling of SLDL models at all levels of abstraction. It can be used to re-code intermediate design models as well as the reference C implementation to generate the initial specification model. The conceptual structure of our source re-coder is shown in Figure 3. It consists of 5 main components:

- Textual editor maintaining the textual document object
- Abstract Syntax Tree (AST) of the design model
- Preprocessor and Parser to convert the document object into AST
- Transformation and analysis tool set
- Code generator to apply changes in the AST to the document object

3.1 Editor

We have chosen a QT [19] and Scintilla [17] based textual editor as the front-end of our source re-coder. The basic document object is based on the data structures in the Andrew text editor [10]. This editor has built-in support for features like syntax highlighting, auto-completion, search, ctags, text folding, bookmarks, undo-redo, and more, for programming languages including C and C++. As an initial step, we have adapted and extended these features for support of SystemC and SpecC SLDLs. The designer makes all the re-coding decisions through the Graphical User Interface (GUI) provided by the editor, for example, through extended context-menus.

3.2 Abstract Syntax Tree

For the editor, the source code being edited is a mere text. In order to analyze and transform the code, the source re-coder needs to recognize the complete structure of the program. When the source code of the design is loaded from a file, a scanner/parser is invoked which creates an object-oriented data structure, AST [21]. internal representation. Figure 4 gives an overview of the amount and the kind of information embedded in the AST object-oriented data structure. The AST preserves all structural information, block, channel, port, and interface, along with C constructs and file information. The AST also provides a set of operations on each object.

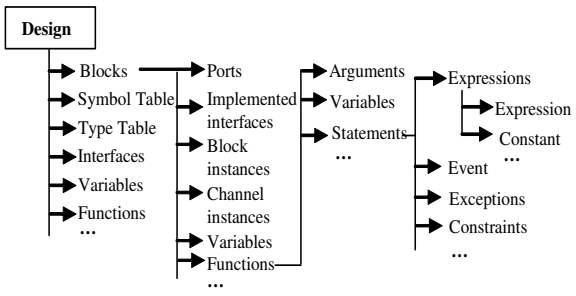

Figure 4. Information in the Abstract Syntax Tree.

The completeness of the AST makes the correspondence between the data structure and the text in the editor possible. Locating an object in the AST for a given line and column in the source window, and vice-versa, is achieved by maintaining line and column information in the AST.

3.3 Preprocessor and Parser

When the design is initially loaded, the preprocessor is run on the source to process file inclusions and macros. The lexer and the parser are extended to provide advanced syntax highlighting in the editor. In addition, the parser creates the AST.

Since we use two separate data structures to maintain text data and syntax tree, we need to synchronize between the two. When the designer modifies the text in the document object, these changes need to be reflected in the AST. Our parser provides this synchronization by applying the changes to the AST immediately while editing [1].

[1] If the text added results in an invalid source code (syntax error), which naturally happens when the designer is typing in the code, the parser parses the program until the point of error and creates only a partial AST. Transformations cannot be invoked during such transient phases, but as soon as the typing is complete and the code is parse'able again, the AST is updated immediately and transformations are available again.

3.4 Analysis and Transformation Tools

The transformation and analysis tool set is the heart of our source re-coder. All re-coding tasks invoked by the user are implemented by these refinement tools. When the designer points to an object in the source window, a node corresponding to the pointed co-ordinates is located in the AST, and a list of available and possible tasks are provided in a context menu. We categorize our re-coding operations into 3 classes.

Structural transformations change the structure of the program by introducing/removing computational blocks, channels and functions. In this category, we further distinguish

- Granulating transformations
- Composing transformations
- Re-organizing transformations

Functional transformations modify computational blocks and functions. For example, the designer may change the interface of blocks by introducing or removing ports. This category is further subdivided into:

- Transformations to contain communication
- Transformation to break dependencies
- Pruning transformations

Analysis functions provide dependency information of an object without introducing any changes to it. Analysis functions also provide information to the designer about potential parallelism at function and block level. In addition, analysis to deal with pointers is also provided.

The description of these transformations is beyond the scope of this paper.

3.5 Code Generator

Any modification in the AST introduced by the transformation tools needs to be reflected in the text in the editor. This synchronization is provided by the code generator, which generates SLDL source code from the modified AST. As such, the code generator is the corresponding tool to the preprocessor and parser which update the AST for text changes.

Table 1. Time to create AST [secs]

Operation	**Simple**	**JPEG**	**MP3**	**GSM**
Lines of code	**174**	**1642**	**7086**	**7492**
Size in bytes	**2757**	**32863**	**223464**	**217414**
Read file, create doc.obj.	0.099	0.104	0.220	0.208
Preprocess	0.027 (0.001)	0.026 (0.001)	0.012 (0.014)	0.028 (0.013)
Parse/Build AST	0.023 (0.034)	0.055 (0.046)	0.431 (0.151)	0.299 (0.147)
Update editor view	0.005	0.005	0.011	0.010
Total time	0.155 (0.035)	0.190 (0.047)	0.675 (0.165)	0.546 (0.160)

4. Experiments and Results

To demonstrate the feasibility and benefits of our re-coding approach, we have implemented the software infrastructure of the proposed re-coder and some of the analysis and transformations to recode a C-based SLDL model. Figure 5 shows the source re-coder at run time. Implemented transformations include

- Localizing global variables
- Re-scoping variables across structural hierarchy
- Synchronizing variable access by introducing channels
- Introducing/Deleting ports in behaviors

Besides, other basic transformations such as semantic-sensitive renaming, deletion, and dependency-analysis functions are also incorporated into the recoder The details of the above transformations are discussed in [2].
To prove the concept of instant refinement in the editor window, we will focus on the responsiveness of the recoder. Following this, we quantify the productivity gains achieved by using our source re-coder.

4.1 Results for Responsiveness

One concern with any interactive environment is its responsiveness. Since our re-coder, in addition to textual editing, builds and maintains a complete AST, and performs complex analysis and transformations, we have measured the time consumed by various operations. Table 1 shows the time it takes to load and build a design from a file. For 4 different designs of varying sizes, the table summarizes timing results for

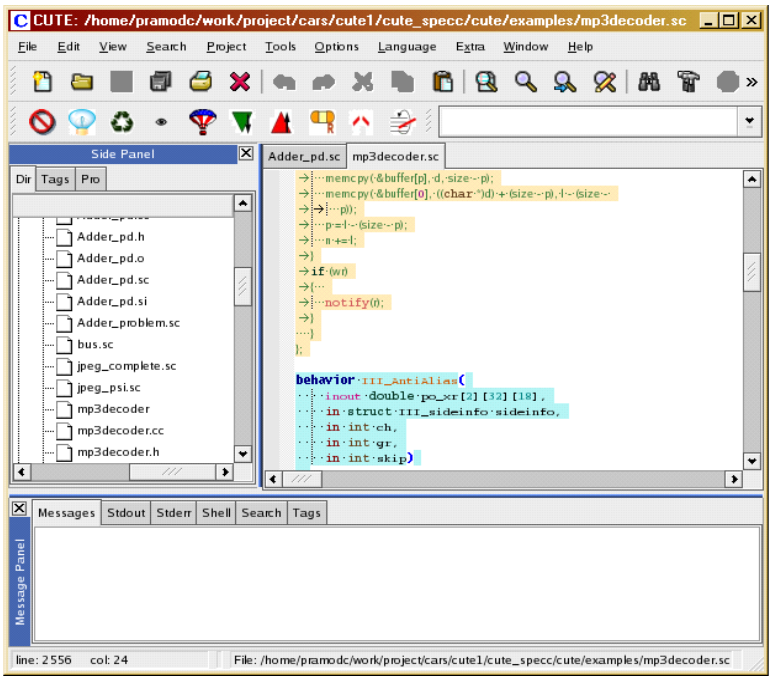

Figure 5. Screen shot of Source Re-Coder.

each sub-operation. The measurements were performed on a Pentium-4 3GHz Linux PC. Even for the designs with more than 7000 lines of code, the responsiveness of the re-coder is satisfactory. Clearly, the creation of the editor data structure and the AST are the most time intensive operations. In the current implementation of our re-coder, synchronization between the editor document object and the AST are implemented using file I/O. These file I/O overheads are indicated in parenthesis. Clearly, our re-coder is sufficiently responsive, even in the presence of the file I/O overhead. In the future, if the performance of the source re-coder becomes critical, some of the overhead will be eliminated. We will also consider incremental updates of the data structures, which promise to cut down the processing times even further.

Table 2 shows the timing of different transformations as experienced by the designer while using the source re-coder on the MP3 design example. The table lists 3 transformations used to create explicit communication structure [2] in the SoC models, the number of times they were invoked for the MP3 example, and the resulting number of lines affected. The interactive recoding time in the table is the wall clock time as experienced by the designer. These otherwise time consuming transformations can be implemented in seconds.

Table 2. Re-coding time in MP3 example[secs] [2]

Operation	**No. of times applied**	**Lines changed**	**Interactive Re-coding time (secs)**
Localizing variables	26	330	90
Re-scope	38	839	126
Synchronize with channels	6	172	30
Total	70	1341	246

Table 3. Productivity gains [2]

Transformations	**JPEG**	**MP3**	**GSM**
Global Variables localized	8	70	83
New Ports added	2	146	163
New Channels added	1	6	2
Re-coding time (secs)	27	246	260
Estimated Manual time (mins)	53	497	585
Productivity factor	117	121	135

4.2 Productivity gain

We will now assess the productivity gains resulting from our source re-coder. We have applied the above mentioned transformations on different design examples and compared the design time taken to implement these manually over using our source recoder. Table 3 shows the productivity gain for different transformations. The manual time is obtained by actually realizing the transformations manually for a set of 10 variables and extrapolating the results for all the variables. The manual editing of the code was performed using Vim [20], an advanced text editor with block editing capability. Re-coding time is the time taken to implement the same transformations using the source re-coder by the same designer. Given the decision information, our source re-coder needs less than a second to implement these transformations. On the contrary, it will take fractions of an hour (instead of fraction of a second) to manually implement these transformations.

In general, measuring productivity is a difficult task. Factors such as designer's experience and tools used must be considered for accurate measurement of productivity gains. However, in our experiments, since productivity gains are in the order of hundreds, any increased accuracy in the measurements will not significantly influence the end conclusions. In other words, since our improvements are by multiple orders of magnitudes, small adjustments to measurement accuracy will not make any significant difference.

5. Summary and Conclusions

In model-based design flows using automated synthesis tools, the creation and maintenance of the design model is often the main bottleneck towards further reducing of the design time. Little or no tool support is typically available for specification coding and re-coding. Usually, designers have to edit the design model manually, using simple text-based editors, requiring significant effort and time in tedious coding. At the same time, modeling is a critical task in SoC design, as the quality of the resulting implementation directly depends on the quality of the input model.

In this paper, we have proposed a novel approach to design specification and modeling, that is based on interactive decision making by the designer ("designer-in-the-loop"), and automation of model analysis and transformation tasks. Eliminating mundane and error-prone text editing tasks, our recoding approach utilizes the precious time and effort of the system designer efficiently.

In particular, we have introduced an interactive source re-coder which integrates compilation, analysis and transformation tools into a text-based editor, to assist the designer in modeling and re-modeling of SoC designs. Our source re-coder is fully text-, syntax-, and semantics-aware, enabling powerful model analysis and transformation operations.

Our approach is especially useful in re-coding of reference models into SoC specification models, which is often the case for new designs. Our source re-coder aids the designer in the analysis and comprehension of the reference model, as well as in restructuring of the model towards a well-specified input model for the system design flow.

For initial analysis and transformation tasks, our experimental results clearly demonstrate that an interactive approach is not only feasible, but also effective. Analysis results or transformed code are presented to the user instantaneously, relieving the designer from tedious coding. Moreover, we have demonstrated tremendous productivity gains through the reduction of modeling time.

For the future, we will extend our approach to include further analysis and transformation tasks, including coupling it with system profiling and estimation tools.

References

[1] P. Chandraiah and R. Dömer. Specification and design of an mp3 audio decoder. Technical report, Center for Embedded Computer Systems, May 2005.

[2] P. Chandraiah, J. Peng, and R. Dömer. Creating explicit communication in soc models using interactive re-coding. In *Proceedings of the Asia and South Pacific*

Design Automation Conference (ASPDAC), Yokohama, Japan, January 2007.

[3] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and Design of Embedded Systems*. Prentice Hall, 1994.

[4] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao. *SpecC: Specification Language and Design Methodology*. Kluwer Academic Publishers, 2000.

[5] A. Gerstlauer, R. Dömer, J. Peng, and D. D. Gajski. *System Design: A Practical Guide with SpecC*. Kluwer Academic Publishers, 2001.

[6] A. Gerstlauer, S. Zhao, D. D. Gajski, and A. M. Horak. SpecC system-level design methodology applied to the design of a GSM vocoder. In *Proceedings of the Workshop of Synthesis and System Integration of Mixed Information Technologies*, Kyoto, Japan, April 2000.

[7] T. Grötker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[8] S. Gupta, R. K. Gupta, N. D. Dutt, and A. Nicolau. Coordinated parallelizing compiler optimizations and high-level synthesis. *ACM Trans. Des. Autom. Electron. Syst.*, 9(4):441–470, 2004.

[9] M. W. Hall, J.-A. M. Anderson, S. P. Amarasinghe, B. R. Murphy, S.-W. Liao, E. Bugnion, and M. S. Lam. Maximizing multiprocessor performance with the SUIF compiler. *IEEE Computer*, 29(12):84–89, 1996.

[10] W. J. Hansen. Data structures in a bit-mapped text editor. *Byte Magazine*, 12(1):183–189, January 1987.

[11] S. Hiranandani, K. Kennedy, C.-W. Tseng, and S. K. Warren. The D editor: a new interactive parallel programming tool. In *Supercomputing*, 1994.

[12] A. Jerraya, H. Tenhunen, and W. Wolf. Guest editors' introduction: Multiprocessor systems-on-chips. *Computer*, 38(7):36–40, 2005.

[13] K. Kennedy, K. S. McKinley, and C.-W. Tseng. Analysis and transformation in the ParaScope Editor. In *ACM International Conference on Supercomputing*, Cologne, Germany, 1991.

[14] S.-W. Liao, A. Diwan, R. P. B. Jr., A. M. Ghuloum, and M. S. Lam. SUIF explorer: An interactive and interprocedural parallelizer. In *Principles Practice of Parallel Programming*, 1999.

[15] Eclipse java development tool-kit. http://eclipse.org/jdt/index.html.

[16] Microsoft visual studio. http://msdn.microsoft.com/vstudio/.

[17] Scintilla source code editing component. http://www.scintilla.org.

[18] S. Sutherland, S. Davidmann, P. Flake, and P. Moorby. *System Verilog for Design: A Guide to Using System Verilog for Hardware Design and Modeling*. Kluwer Academic Publishers, 2004.

[19] Trolltech Inc. Qt application development framework. http://www.trolltech.com/products/qt/.

[20] Vim, advanced text editor. http://www.vim.org/index.php.

[21] I. Viskic and R. Dömer. A flexible, syntax independent representation (SIR) for system level design models. In *Proceedings of EuroMicro Conference on Digital System Design*, August 2006.