WWW Based Structuring of Codesigns

P. G. Plöger[+], *J. Wilberg*[+], *M. Langevin*[+], *R. Camposano**

[+] GMD-SET, Schloß Birlinghoven, D-53757 Sankt Augustin, Germany, {wilberg, ploeger}@gmd.de

* Synopsys Inc., 700 East Middlefield Road, Mt. View, CA 94043-4033, USA, raul@synopsys.com

Abstract.

This paper describes a codesign environment based on the WWW (World Wide Web) and its implementation. Tool invocations and their respective results are linked using hypertext documents. We show how to configure a WWW browser for spawning design tools and how frequent tasks like documentation generation and retrieval are facilitated. The design flow can be adopted to the given application very easily. In addition we introduce the concept of a work flow called 'design by documentation'. A WWW link to the results is given and experience using it in a codesign project is described.

1. Introduction

A design flow for complex embedded systems depends strongly on the application domain [3] [19] [20] [22] [28]. Each domain requires a particular design strategy with emphasis on different, often contradicting, design aspects. Examples of such codesigns are described in [6] [8] [11] [13] [26]. This diversity is difficult to support with a unique rigid design flow. The problem is aggravated if the design flow is based on one large monolithic tool such as synthesis. An alternative is a codesign workbench [3] with a number of small tools. Each of these tools has a limited functionality. They serve as building blocks for configuring complex, application specific design flows. This allows to combine general-purpose tools like editors with highly specialized tools like dedicated filter synthesis systems for a particular design.

An approach to implement design flows were so called 'design frameworks', e.g., CFI [4]. Some of them relied on strong coupling of tools and design data, which makes them complex and not very flexible. This is evidenced by the change of focus in CFI. A framework [9] [14] [15] [16] controls the design tools, encapsulates tool operations, manages access to design databases, etc. Examples range from "traditional" version management tools like RCS, CVS, to tools for design information management, like Nelsis [10], JCF [17], etc. Performing complex codesigns adds a number of new facets to the already rich spectrum of framework services. One example is the cosimulation of complex systems such as in Ptolemy [14].

Especially relevant to complex designs is information management and the documentation of the design. Information is shared by groups of designers from different disciplines. Consider for example the design of a video compression system [1][3][27]. Typically, the best system performance is achieved by selecting the most suitable combination of algorithms, software implementation, and hardware design. This requires close cooperation among engineers of different domains [19]. The hardware designer needs to know about the results of an evaluation of different compression algorithms, since different software implementations will impose different requirements on the datapath. E.g. multiply operations may be favorably replaced by shifts in some implementations, while others are tailored to have exactly one multiplication in each basic block. Providing access to these results is not enough, since the data must be explained and relevant background material must be provided. In other words, the raw data will be useful only to an expert on that particular topic, while other members of the design team need annotated data. Sophisticated codesign systems typically aggravate this situation since they allow to explore many different alternative solutions in a short time. Suboptimal system solutions arise since each designer focuses on his/her particular design part, while a multi-disciplinary approach is difficult due to the lack of information.

One solution to this problem is the tighter coupling of design process and documentation [24], i.e., a *designing by documentation*. Hypertext documents are ideal for this purpose since they allow to select background material on demand by following hyperlinks. Furthermore, tool invocation can be coupled to a link, and design data can be connected in almost arbitrary ways reflecting the complex interdependencies between the design data. In [24] a framework was proposed which integrates design and documentation by means of a commercial text processing system and a special navigator designed for a special 'Active Document Description Language'.

In this paper, we propose an implementation of a similar design by documentation approach but based on the WWW (World-Wide Web) [2] and its powerful HTML language [12]. This offers a number of significant advantages. First of all, widely used WWW browsers, like NCSA's Mosaic [21], are available. The designer does not need to learn a new tool. Hyperlinks can be resolved on a global scale. This allows to integrate material from other WWW sites. The system is scalable, starting from a small implementation mainly based on native WWW features up to a distributed design environment that manages design teams at remote locations. Due to the client/server approach of WWW, the database management of the design can be directly integrated into a server. For example, tool invocation and access privileges can be controlled by using a WWW server even on a global scale! Finally, the design documentation or parts thereof can be published immediately in WWW, allowing the sharing of information among different design groups. Our approach is completely informal, i.e. it is not covered by a formal model or formalism.

This paper is organized as follows. The next section outlines the idea of an HTML based codesign framework based on the problems encountered during the design of a video compression system and gives a brief description of some WWW features. Section 3. describes the basic steps for setting up the framework. Section 4. discusses an example implementation and section 5. concludes the paper with a discussion of the main results.

2. Organizing complex codesign tasks

First, we describe the design of a video compression system with a group of designers. In this example a number of different implementations were analyzed to determine the most suitable system solution. A large amount of design data was produced, which initiated the development of an HTML based codesign framework for organizing it. After outlining the main design flow problems encountered, we summarize some important mechanisms of the WWW, give an overview on these features and how they can be exploited to organize the codesign.

Our sample codesign task is the development of a PC board employing a special processor, which is able to decompress an input stream of compressed motion pictures in real time. The board should be able to handle different compression methods like MPEG [7] and h.261 [18] sequences or JPEG [23] pictures (first level in Fig. 1). We can choose among different implementations of critical routines (second level Fig. 1). In our proposed design flow [3] [27] we experiment with a specification under various operating conditions. First, performance for several characteristic input data streams is simulated with a fixed initial implementation of the application program (third level, upper half). In an analysis phase, the processor description is varied and parallel code and execution profiles are produced. Finally,

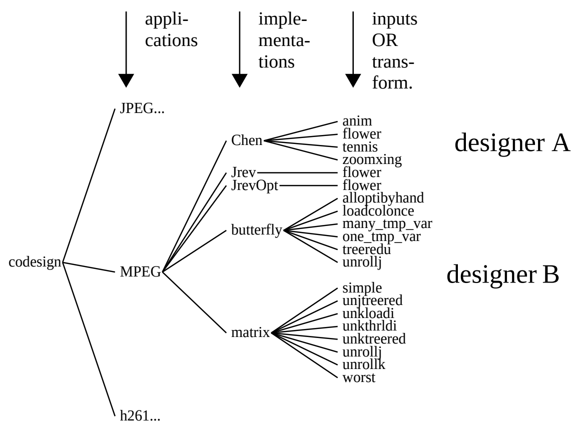


Fig. 1 A sample directory hierarchy to keep intermediate results during the design flow; all nodes are directories.

transformations will change the C source code of some critical modules, thus yielding new executables (third level, lower half). To make a systematic investigation of the expected performance, one usually has to vary a multitude of different parameters. In the discussed example, these are *applications*, *implementations* and *input data*, symbolized by the three levels. Designer A has the task to play with different input sequences using an implementation suggested by Chen [5], which is symbolized by the path **codesign->MPEG->Chen->{anim, flower, tennis, zoomxing}** (directory nodes in bold font) in Fig. 1. After that she checks the **Jrev** and **JrevOpt** implementations. Simultaneously a second designer B starts to work on different implementations, namely **matrix** and **butterfly**, to evaluate the influence of code transformation techniques on the performance of the implementation, like tree height reduction, unrolling nested loops on different levels and memory disambiguation. She places the results of the experiments at the same level where designer A has placed her results. Both initial directory trees, except for the personalized third levels, are private copies and generated by a checkout of the CVS version control system. This ensures, that only local copies are altered.

Now we want to compare some results from different implementations to draw some conclusion w.r.t. the datapath. E.g. a diagram like Fig. 2 comparing the cycle counts of different implementations using 1 to 6 adders results. B cannot be sure which results of designer A might be useful for her since this will depend on the fact that all experiments were conducted, e.g., on the same input sequence. Before any conclusions, like discard butterfly altogether or use more then 4 adders in the datapath, can be drawn, one has to check on the validity of the results. What is needed is a glimpse at the most recent results of each designer, i.e. the current state of the design. Quick access to different loca-

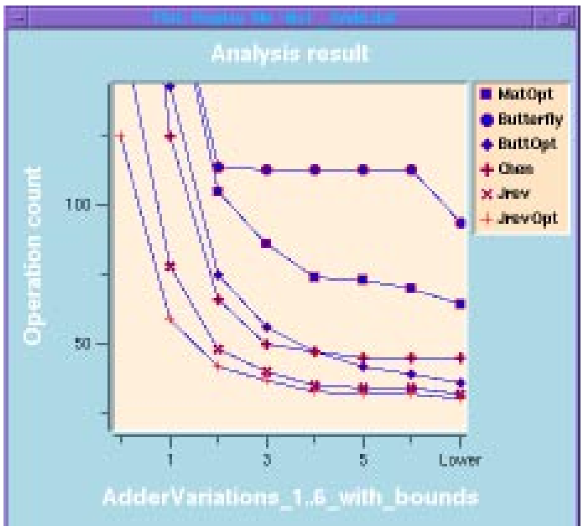


Fig. 2 Cycle counts of implementations for different number of adders

tions and transparent information retrieval is required.

From this example it can be seen that the design flow entangles the directory hierarchy and spreads results all over. These problems are aggravated by consistency assurance for collaborative work or history management. A practical solution for the problem of reproducible deposition of files and information about them in a complex design context is to place meta information in the directories and/or adjacent to the files in question. However, reliable information recovery depends decisively on the documentation discipline of all participants which is usually rather low. In summary, the designer needs to know where to find valid result files of coworkers, documentation about them, and which tools need to be used at what step of the design flow. In this situation our work contributes a solution.

2.1 HTML, WWW and Mosaic

Before describing the technical details of setting up a personalized server daemon and browser for the codesign tasks, we briefly review the basic WWW [2] concepts. The underlying communication pattern for all WWW related communication is based on a client and a server exchanging mail (Fig. 3). A WWW browser, like Mosaic [21] is a client. It is a text viewer displaying text files written in a special language called ‘Hypertext Markup Language’ (HTML)[12]. Two characteristics make this the cornerstone of multimedia communication. First of all, it can follow *hyperlinks*. These use a range of addressing modes and protocols, from addressing a local text file or directory to local HTML files to HTML files offered by an http server daemon on a foreign internet host. Other protocols like ftp or gopher are also integrated. The host serves document requests by running an http daemon process. Secondly, a WWW browser allows the invocation of an *externally spawned viewer*. The viewer to be spawned depends on the context type of the document and a mapping of this type to a certain program. This ability turns a WWW browser into a multimedia environment, since picture viewers, motion picture playing programs or audio players can all be spawned from a single environment. A viewer may also be a shell. In this case ‘viewing’ a document means to execute it.

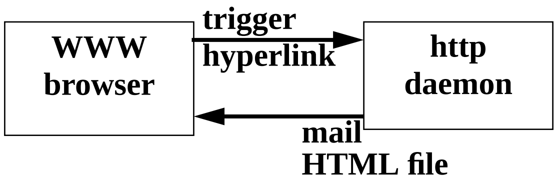


Fig. 3 Communication flow when executing a hyperlink

2.2 A codesign framework using HTML

To solve the problems mentioned at the end of section 2, we enhanced our example hierarchy of Fig. 1 with HTML files documenting the contents of each directory and/or some of the contained files. The hypertext links in the HTML files

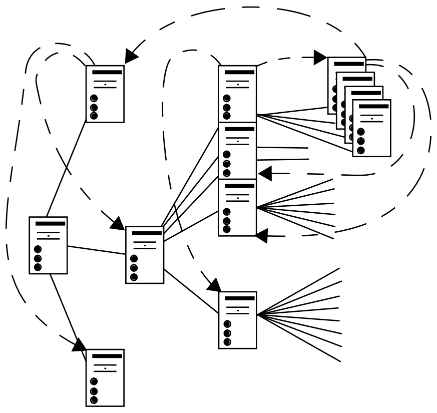


Fig. 4 HTML files meshing directory nodes in an arbitrary way, dashed lines denote ‘links to’

superimpose an arbitrarily meshed structure (Fig. 4) on the hierarchy. This decouples the original location semantics from the design flow. E.g. when the comparison of two results at different directory nodes is needed, two links may be placed on one HTML page, which was written specially for this comparison purpose. Browsing different results for comparison purposes is very easy with the appropriate links. To integrate tools WWW viewers can enable external tool execution. To facilitate matching of documentation and validity of results, an HTML sensitive editor and a special makefile target were integrated into the HTML files. Assuming that the designer always accesses her HTML linked files using the proposed browser, she is reminded to wrong or missing links whenever she tries to follow them.

Three additional advantages of this HTML based approach are worth mentioning. First of all, work may be exported to the research community via a single push button: publish the address of your master HTML page on the net, and they have instantaneous access to your current results and projects. Secondly, it is very easy to implement a running demonstration of a particular design: replace the tool instantiations on each individual HTML page by screen shots of the invoked tools and your demo is ready. Lastly the approach scales up very easily. In principle one can start out to document a single directory, integrating more at demand. Ultimately the underlying directory structure is completely hidden from the designer, who only navigates along those links needed.

3. Implementation

While navigation features are built into every WWW browser, tool invocation must be specially enabled. Furthermore, some mechanism to pass arguments to the invoked tools must be provided. Finally, the setup and configuration files for the WWW codesign environment itself need to be organized.

3.1 Starting tools

The browser must be enabled for tool execution. The Mosaic manual [21] explains how this can be achieved. This can be done for other browsers in a similar way. Every WWW mail has a contents type. Depending on this contents type, the appropriate viewer for displaying the message is chosen. In some cases, the HTML browser spawns an external viewer. There is a special context type, where a c-shell is the viewer. The browser will spawn a subshell executing the text file sent to it, which must be a valid shell script in this case. Therefore, each contents type can be associated with the appropriate viewer for displaying the contents.The mapping of context types to viewers is specified in a parameter file, which can be personalized to special needs

A second parameter file maps file extensions to viewers. Each tool used during the codesign task has to be wrapped into a tool execution script with the correct extension. Then the browser needs the two personalized parameter files to map the WWW mails received to the right tool instantiation. We embedded the original Mosaic into a shell script, which automatically supplies the adapted configuration files. Global environment variables like library paths are set in this shell file also. The technical details of this setup process may be found in [25].

Tool execution is possible now but no argument can be passed to the execution shell in this way and no checking of permissible tool invocations is performed. The global environment variables are fixed when starting the browser but the tool invocation arguments usually depend on the last recently displayed HTML file which triggered the hyperlink.

3.2 Passing arguments to tools

To allow passing arguments to tools we set up our own http daemon. Any such server process is able to resolve a valid WWW hyperlink into a pathname and a filename. It usually mails back the requested file. In addition, a data base query may be performed on a WWW server. In this case, the server generates an HTML document on the fly which contains the query result.

This facility can be used for creating start up scripts containing the tool call *and* the tool arguments. The server mails this script back to the browser which executes it to start the design tool with the appropriate arguments. For example, when the user selects the hyperlink with address 'http:// alcatraz.gmd.de: 9422 /designbin/call? $DESIGN + display.csh + simSpeedUp.tab', the server executes the program 'call' with the string found after the after the question mark, the query string. The 'call' program forms a c-shell script which will invoke the design tool 'display.csh' in directory '$DESIGN' with the argument 'simSpeedUp.tab'.

This script is mailed back to the browser which executes it. The 'call' program also checks if the requested tool is permitted. Only start up scripts for allowed tools are created, otherwise an error message is returned. To summarize, only one setup step is necessary for the daemon: supply a program for creating start up scripts from query strings and let the http daemon execute it.

It is interesting to note that this simple strategy may be used as an entry point to form full-fledged design data base management. In this case the small 'call' program is replaced by a system for design information management [15] [10]. In the hyperlink example above take '$DESIGN' as design object name, 'display.csh' as desired tool and 'simSpeedUp.tab' as design data. In addition, the users address can be used to determine access privileges. After finishing the transaction, the http daemon can take the responsibility to update an underlying design data base. This concept works even on a global scale due to the powerful features of WWW.

3.3 Putting it all together: organization of the codesign framework and HTML files

On the server side, we set up a directory with the perl program 'call' and a number of template cshell files that encapsulate the invocation of the codesign tools. On the client side, we use an HTML browser that is enabled for executing c-shell scripts. The underlying directory hierarchy of the original design was not changed. Instead, we followed some conventions for adding HTML files which overlay the directories with the desired meshing of related results. For each directory node, there is a 'README.html' file containing a brief description of the contents of each file. This is meant as a fast entry point for an expertuser. In addition, there are documents that explain the data in a more verbose way. These files can be used by other designers who are not completely familiar with the specific data. These documents also provide links to background material which allows a novice designer to inform herself about general concepts.

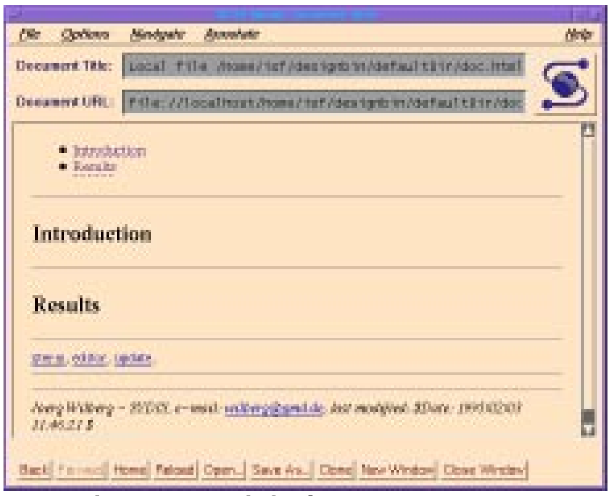
Templates are used for the HTML pages. This contributes to uniform appearance of the design documents and links to frequently used tools are directly included into this template. A single root directory is used as anchor for the design directory. Within this tree, relative addressing is used for the hyperlinks. This supports a reuse of design documents since the directory can be moved to another location without destroying the internal linking.

4. Results

Since this article deals with HTML pages the results are best described by a hyperlink. We offer two anchors, namely 'http://alcatraz.gmd.de:9422/sydisdoc/' and 'http://alcatraz.gmd.de:9422/designenv/'. While the first is very much a traditional tree-like hierarchy of user and reference documentation, the second has a link 'video compression system' in 'Design example'. This link is the entry point for the desired mesh of HTML files described in Fig. 4 for the example of Fig. 1. There is also a link to a demonstration which displays screen shots of our tools. If you like to see the proposed HTML environment at work with real tool invocations, you need to get the CASTLE tools first and install them on your local machine. You find a link how to get these tools in the pages above.

We have used the HTML based system for three months, and it has proven to be an invaluable help. It greatly facilitates visualization tutorials and demonstrations.

Since our WWW setup interweaves documentation and design activities closely, it also influences the work flow. When a designer starts with a codesign task, she will write an initial HTML master page first, describe the goals on it and collect links to frequently needed tools. Other tools can be added on demand, exactly at the time needed. Since all pages are in use from the very beginning, the meshing is permanently being verified. We call this *design by documentation*. To minimize the overhead and the learning curve for the designer to document her design activities on HTML pages, we use an EMACS editor initialized in HTML mode [12]. We supplied a default template HTML reference page shown in Fig. 5. This page contains links to the most frequent activities, namely visiting the source directory via 'xterm', spawning the editor via 'editor', and updating the current directory via a special makefile target 'update'. Furthermore, standard sections like 'Introduction' or 'Results' are included.



Document Title: Local File /home/tsf/designbin/defaultDir/doc.html

Document URL: file://localhost/home/tsf/designbin/defaultDir/doc

- Introduction
- Results

Introduction

Results

xterm, editor, update

Joerg Wilberg – SYDIS, e-mail: wilberg@gmd.de, last modified: $Date: 1995/02/01 11:46:21 $

Fig. 5 HTML default page structure

Several other aspects of designing in a group were not described, e.g., revision management, reusable makefiles and overall context setup for a follow-up design project. These aspects have been successfully integrated into our design environment and are described elsewhere [25]. Although our WWW based approach cannot compete with a full-fledged framework, it will help the designer to achieve reproducible, documented results in a shorter time. Furthermore the HTML based solution is fully scalable, since one can start out with as little as some readme files and end up with the integration of other frameworks into the HTML pages. The professional looking interface in a modern, up-to-date tool will inspires the designer during her work.

5. Conclusions

An environment for complex codesign tasks was proposed. It is based on HTML, supports parametrized tool instantiation and simultaneous documentation of design steps. All designers have common access to the most recent results which are embedded into HTML files. Information flow in a group context is facilitated by merging of production of results and their simultaneous documentation. Finally, the design data can be made available to the world simply by posting an address in the WWW.

Acknowledgments

This work would not have been possible without the CASTLE design team, H. Veit, M. Theissinger and P. Stravers.

Clarifying discussions with H. U. Kobialka about frameworks are also acknowledged. This work was sponsored by the German administration, ‘Bundesministerium für Forschung und Wissenschaft’ under contract number 01M2897A SYDIS.

References

[1] B. Ackland: “The Role of VLSI in Multimedia”, IEEE J. Solid-State Circuits, vol. 29, no. 4, pp. 381-388, April 1994.

[2] T. Berners-Lee, et al.: “The World-Wide Web”, Comm. ACM, vol. 37, no. 8, pp. 76-82, Aug. 1994.

[3] R. Camposano, J. Wilberg: “Embedded System Design”, to be published.

[4] “CFI Home Page”, http://www.cfi.org

[5] W.H. Chen, C.H. Smith, S.C. Fralick: “A Fast Computational Algorithm for the Discrete Cosine Transform”, IEEE Trans. Commun., vol. 25, pp. 1004-1009, 1977.

[6] R. Ernst, J. Henkel, Th. Benner: “HW/SW cosynthesis for microcontrollers”, IEEE Design & Test, pp. 64-75, Dec. 1993.

[7] D.J. Le Gall: “MPEG: A Video Compression Standard for Multimedia Applications”, Comm. ACM, vol. 34, no. 4, pp 46-58, 1991.

[8] D. D. Gajski, F. Vahid, S. Narayan: “System-Level Methodology and Technology”, E-DAC Tutorial, 1994.

[9] D.S. Harrison, A.R. Newton, R.L. Spickelmier, T.J. Barnes: “Electronic CAD Frameworks”, Proc. IEEE, vol. 78, no. 2, pp. 393-417, 1990.

[10] A. van der Hoeven, O. ten Bosch, R. van Leuken, P. van der Wolf: “A Flexible Access Control Mechanism for CAD Frameworks”, EuroDAC’94, pp. 188-193, 1994.

[11] X. Hu, et al.: “Codesign of Architectures for Automotive Powertrain Modules”, IEEE Micro, vol. 14, no. 4, pp. 17-25, August 1994.

[12] “HyperText Markup Language (HTML)”, HyperText Markup Language (HTML): Working and Background Materials, http://www.w3.org/hypertext/WWW/MarkUp/MarkUp.html.

[13] A. A. Jerraya, et al.: “Linking System Design Tools and Hardware Design Tools”, in D. Agnew et al.(eds.): “Computer Hardware Description Languages”, IFIP Trans., vol. A-32, 1993.

[14] A. Kalavade, and E. A. Lee, “A Hardware/Software Codesign Methodology for DSP Applications,” IEEE Design and Test, vol. 10, no. 3, pp. 16-28, September 1993.

[15] R.H. Katz, M. Anwarrudin, E. Chang: “A Version Server for Computer-Aided Design Data”, 23rd DAC, pp. 27-33, 1986.

[16] H.-U. Kobialka, C. Meyke: “View on an Object-Oriented Software Engineering Environment”, Proc. 6th Int. Workshop on Computer-Aided Software Engineering, CASE’93, 1993.

[17] D.C. Liebisch, A. Jain: “JESSI Common Framework Design Management - The Means to Configuration and Execution of the Design Process”, EuroDAC’92, 1992.

[18] M. Liou: “Overview of the px64 kbps video coding standard”, Comm. ACM, vol. 34, no. 4, pp. 59-63, April 1991.

[19] H. De Man: “Design Technology Research for the Ninties: More of the Same?”, EuroDAC’92, pp. 592-596, 1992.

[20] G. De Micheli: “Computer-Aided Hardware-Software Codesign”, IEEE Micro, vol. 14, no. 4, pp. 10-16, August 1994.

[21] “NCSA Mosaic FAQ: Other Mosaic/WWW Software”, http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/faq-software.html

[22] P.G. Paulin, C. Liem, T.C. May, S. Sutarwala: “DSP Tool Requirements for Embedded Systems: A Telecommunications Industrial Perspective”, J. VLSI Signal Processing, vol. 9, pp. 23-47, 1995.

[23] W. B. Pennebaker, J. L. Mitchell: “JPEG Still Image Data Compression Standard”, Van Nostrand Reinhold, New York, 1993.

[24] M.J. Silva, R.H. Katz: “Active Documentation: ANew Interface for VLSI Design”, 30th DAC, pp. 654-660, 1993.

[25] “The CASTLE Analysis Environment”, CASTLE Analysis Home Page, http://alcatraz.gmd.de:9422/designenv/hello.html, in preparation.

[26] D. E. Thomas, J.K. Adams, H. Schmit: “A Model and Methodology for Hardware-Software Codesign”, IEEE Design & Test, vol. 10, no. 3, pp. 6-15, Sept 1993.

[27] J. Wilberg, R. Camposano, W. Rosenstiel: “Design Flow for Hardware/Software Cosynthesis of a Video Compression System”, 3rd Int. Workshop on Hardware/Software Codesign, Sept. 22-23, Grenoble, pp. 73-80, 1994.

[28] W. Wolf: “Hardware-Software Co-Design of Embedded Systems”, Proc. IEEE, vol. 82, no. 7, pp. 967-989, July 1994.