Integration of VHDL into a System Design Environment

Ludwig Schwoerer, Matthias Lück, Hartmut Schröder
University of Dortmund
Working Group on Circuits of Information Processing
44221 Dortmund, Germany
Tel.: (+49) ..231/97506717, Fax: (+49) ..231/7553196
E-Mail: LS@NT.E-Technik.Uni-Dortmund.de

Abstract

Verification of image processing systems is mainly done on the basis of image sequence simulations. To achieve high simulation efficiency, our compiled code simulator MSIPC offers a high performance clock period precision simulation, according to the SDF simulation paradigm. Furthermore it supports mixed mode (e.g. VHDL) simulations via coupling to external simulators, and via cross-compiling.

1. Introduction

The design of complex image processing systems demands for computer aided design procedures. Normally, a design is done in several steps of refinement, starting at the system level with the algorithm design and ending e.g. at the gate level with an ASIC netlist. At all these stages, the proceeding design has to be verified by extensive image or even image sequence simulations. These simulations can take up to 90 % of the whole design time. Therefore, powerful simulation tools are necessary, that enable fast simulations even for these high quantities of test data [1]. To ensure an uninterrupted design over all design stages, the system has to be multi-level and mixed-mode capable, i.e. to offer simulation facilities for all applied implementation styles.

In this paper we present the new compiled code simulator MSIPC, whose main part is a fast multi-level simulator, which can be coupled to several external simulators (e.g. VHDL). Furthermore, optimizing cross-compilers enable very efficient simulations of processor-based modules. Related approaches [2], [3], [4] do not offer this functionality, since they are more focused on synthesis. One of the main topics in current MSIPC development is to extract as much functionality from external descriptions as possible and to simulate them functionally identical within MSIPC, in benefit of simulation efficiency.

2. Typical design flow for image processing systems

Typically, when a new image processing system is being designed, first the underlying algorithms are investigated and developed by means of image sequence simulations, i.e. well chosen testsequences are grabbed into an image sequence store (or already available as files) and the algorithms under test are performed offline with these data. The results are then written back into the image sequence store and visualized on connected monitors. Since the amount of data to be processed is enormous (typically about 400 kB per frame and 25 frames per second, yielding 10 MB per second of image sequence), these simulations are very sensitive to simulation efficiency. These simulations are normally implemented on the basis of ordinary high-level languages, in most cases ´C´, not regarding any impacts on further hardware realization [5]. For example, it is quite common to separate major blocks of the algorithm by frame buffers, so that unlimited random access to twodimensional frame data is possible - which is not the case for real hardware environments.

After the succesful verification of the algorithm at system level, the system is split up into several modules, for each of which an appropriate design methodology (ASIC, signal processor based, etc.) has to be chosen. Normally, here the design flow is broken up, i.e. for each module the design now starts just with the specifications derived from the system simulation, but without any mutual interference between the different modules. This is even more forced due to the different design methodologies, resulting e.g. in a VHDL - description for the ASIC parts, whereas the signal processor based parts result in the corresponding assembler code. Also, possible feed-

backs to the system level resulting e.g. from optimization on register transfer level, are very indirect. All resulting changes have to be induced manually to the system simulation and checked again by image sequence simulations.

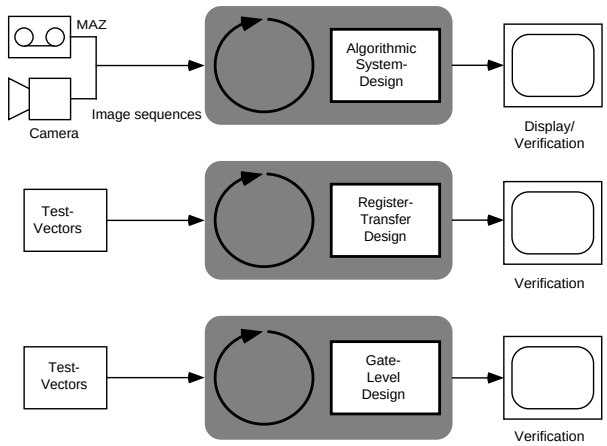

Figure 1: Typical design flow for image processing systems

It was our goal to overcome this gap during the design flow and to enable mixed mode image sequence simulations for the whole system, when we started to work on MSIPC.

3. The concept of MSIPC

The system design environment MSIPC (**M**ixed mode **S**ystem simulation for **I**mage **P**rocessing **C**ircuits) enables a continuous design of image processing systems from system level down to implementation level. Through all stages of the design, the system as a whole with all the interaction between different modules can be evaluated by means of image sequence simulations,.

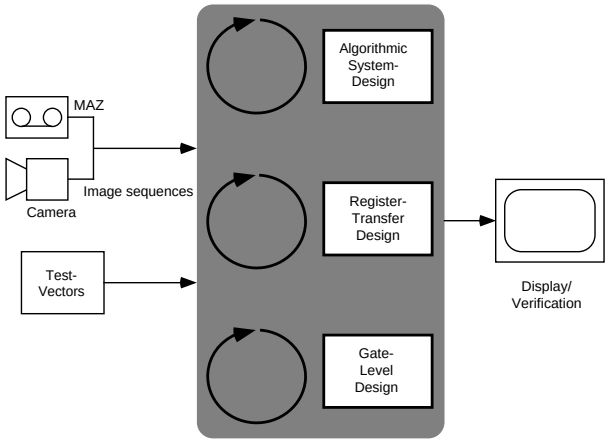

Figure 2: Improved design flow using MSIPC

The main part of MSIPC is a compiled code simulator (sometimes also classified as Streamline Code Simulation [6]). It generates optimized C-Code to perform a clock period precision simulation from system to register transfer level, according to the Synchronous Data Flow (SDF) [7] paradigm. Since typically 80 to 90 % of all signals change every clock cycle, a compiled code simulation with static scheduling is more efficient than an event-driven simulation technique, which suffers from the event-management overhead because of this high change-percentage [6]. Under these circumstances, a simulation in the SDF domain is by 1-2 orders of magnitude more efficient compared to discrete-event simulation [8]. Therefore MSIPC directly yields an adequate and fast simulation for all synchronous parts of the system.

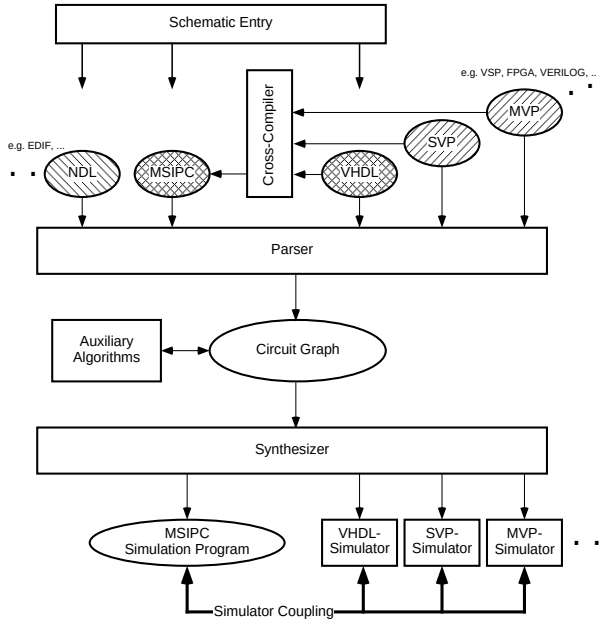

Figure 3: MSIPC system structure

The structural description of the circuit is done by a hierarchical netlist. The description language can either be a pure architectural description language (e.g.: NDL [9]) or a versatile hardware description language as e.g. VHDL. Of course, a description in native MSIPC - code is possible, too. To have a user-friendly front-end, these netlists can be generated by an arbitrary schematic editor.

The functional description of the circuit modules can be done in several ways:

First, the description can be given directly in MSIPC syntax. Since MSIPC is based on ´C´, system designers immediately feel familiar, when using MSIPC. In benefit of simulation speed, the natural data type in a first step is restricted to a bi-valued logic, so that all signals (busses) can be calculated directly with their full wordlengths [10] using standard ´C´algebraic expressions.

On the other hand, functional descriptions can be given in several other formats. The integration of these external formats, e.g. VHDL, is discussed later on.

From these input sources, MSIPC automatically generates an equivalent ´C´ program by taking the MSIPC sources for all modules and arranging them according to the netlist given. In total, a fast clock period precision simulation is achieved, according to the SDF paradigm.

The flexibility of the system concerning different abstraction levels and description styles is achieved by connection to other, already existing simulators. By giving functional descriptions in an external format to MSIPC, the user specifies which parts of the system shall be simulated by which external simulator. To communicate with these simulators, MSIPC automatically inserts appropriate interface modules within the simulation program. Hence it is possible to simulate single modules or blocks of modules with e.g. a VHDL simulator, whereas the rest of the circuit is simulated on system level (mixed mode simulation). This is even more important, when parts of the system are currently refined down to gate level, and have to be tested within the whole system, still specified on system level. But one of the most important advantages of mixed mode simulation certainly is the resulting simulation speed, because most parts of the system are efficiently simulated by MSIPC on system level. After the gate level description has been verified, the whole module can be substituted again by its system level description, and another module can be further refined.

Another way of integrating external functional descriptions is to find a functional (within the SDF domain) identical MSIPC description. Here, all optimizing techniques known from compiler construction can be applied to achieve very high simulation efficiency. This way of cross-compiling is possible for every processor-based module, since they are synchronous by default. As an example, we have implemented such a cross-compiler for the SVP [11], which enables MSIPC to perform most SVP algorithms at 1 frame per second on a Sparc 20. But also a quite large subset of VHDL [12] can be used to derive a functionally identical MSIPC description.

Because of the resulting high simulation speed, now image sequence simulations are possible for the whole system during all stages of design progress.

4. Internal modeling of synchronous circuits and systems

The underlying circuit model of MSIPC consists of modules, whose inputs and outputs communicate via unidirectional signal nets. Modules can be placed in arbitrary hierarchy levels, i.e. that modules can consist of sub-modules, which themselves are constructed by sub-modules. Such modules are called macros. Modules, that have no further sub-modules, are primitives. The behaviour of primitives is given directly as a functional description. An example of such a system is sketched in figure 4.

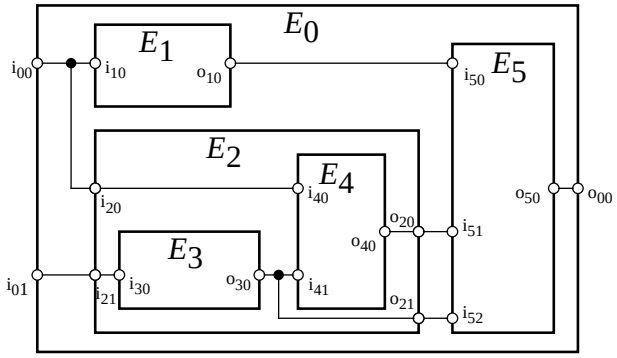

Figure 4: Example of hierarchical system

So, the whole system is defined as a set of modules E_n, n ∈ {0,1, .., N}, each characterized by an according triple $\{I_n, A_n, O_n\}$,where:

I_n is a set of all signal nets connected to the inputs i_{nj}, j ∈ {0,1, .. , J} of module E_n,

A_n is either a structural (for macro modules) or functional (for primitives) description $O_n = A_n(I_n)$,

O_n is a set of all signal nets connected to the outputs o_{nk}, k ∈ {0,1, .. , K} of module E_n.

Inputs and outputs at the top level, i.e. of the total circuit, are called primary inputs and outputs.

The temporal modelling of the system is done according to the applied SDF paradigm with clock period precision, so that MSIPC is dedicated to synchronous circuits. Therefore the delay of a module´s output is not determined by the physical delay time of actual circuitry, but due to internal buffers and registers. So all delays are specified in multiples of clock cycles, and assigned independently to each output pin o_{nk} as

$Del\ (o_{nk})$ = Max (delay (i_{nj} -> o_{nk}), for all j,k).

In total, our module model consists of a pure combinatoric description A_n with concentrated delays at the module´s outputs, as sketched in figure 5.

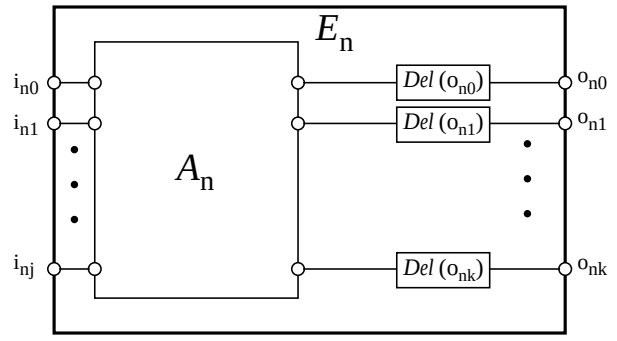

Figure 5: Delays assigned to output pins

The hierarchy of the system and the interconnection of the modules can be described by graphs. The circuit el-

ements (pin and module) are represented by knots, signals and hierarchical corespondencies by edges. One can distinguish two types of graphs:

a) The module-related circuit graph, which is a hierarchical graph, showing on the one side the hierarchical relation between all modules E_n, and on the other side the interconnection between all modules. Figure 6 shows such a graph for the example system of figure 4.

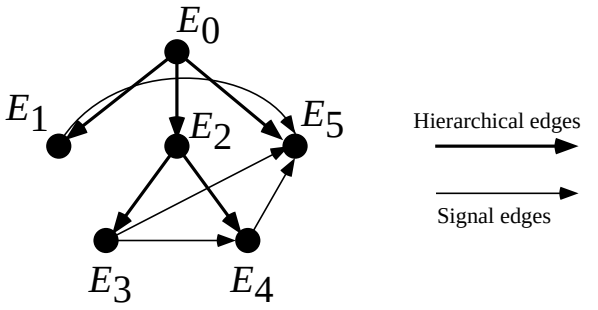

Figure 6: Module-related circuit graph

b) The pin-related circuit graph, where all connections (i.e. signal nets) are represented by bundles of edges. As an eample, the corresponding graph for the example system of figure 4 is sketched in figure 7.

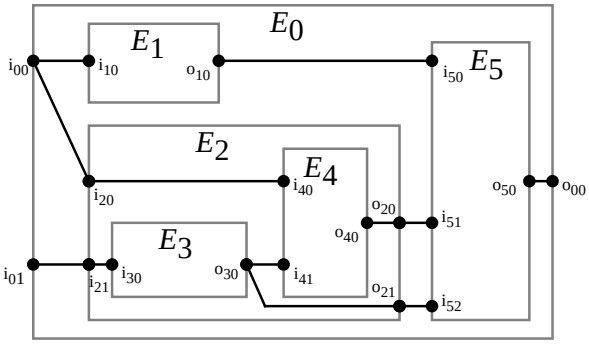

Figure 7: Pin-related circuit graph

To handle these graphs within a software system like MSIPC a suitable data structure has to be implemented to store all circuit information. Since all graphs used by MSIPC are unweighted digraphs, they are stored as double-pointered adjacency list. This allows a fast and flexible search of the circuit architecture [13].

5. Resulting Compiled Code Simulation

After parsing all circuit information to the MSIPC internal representation, the synthesizer part of MSIPC generates a corresponding hierarchical ´C´ program, in which every module is mapped onto a function. That is, a depth-first-search (DFS) algorithm [13] is used to transform the hierarchy graph to ´C´ source code for all macro modules. Within these macro modules, all sub-modules are arranged according to the data flow. The sequence is determined within the socalled ´levelizing´ process. Every module of the module related circuit graph is assigned to a level, which increases along the data flow. Modules without inputs or connected only to primary inputs are assigned to Level(E_p) = 0, all other levels are computed within a breadth-first-search (BFS) algorithm [13] by

Level(E_n) = (1 + Max(Level(E_v) $|I_n \cap O_v \neq \emptyset$))

In cycle-free circuit graphs (i.e. circuits without feedbacks, resulting in directed acyclic graphs, DAG´s) all module levels can be determined this way.

But if the graph contains cycles, this procedure fails. Obviously, the levelizing process has to be cut within circuit cycles. To detect cycles within the system, we use an algorithm to determine all strongly coupled components of a digraph. Now a simple heuristic is applied to choose one module of the cycle, at which the levelizing is cut [14]. Beginning with this module the BFS can be continued. This process is repeated until all cycles are resolved. To illustrate the process, figure 8 shows the determined levels for an example system.

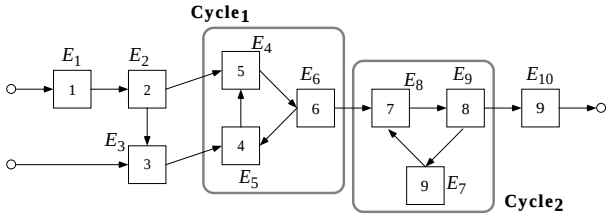

Figure 8: Levelizing within cycles

Since all modules have been arranged according to the data flow, even modules with a specified zero delay can be handled. All signals are evaluated in the correct sequence. Only within cycles at least one module must have a delay ≥ 1 clock cycle, since then calculations are temporally decoupled.

Finally, the automatically generated macro modules together with the user defined primitives (which of course can be predefined in a library) form an autonomous ´C´ program, which is a direct representation of the System Under Test (SUT). All primary inputs and outputs are connected to (image sequence) files, defined during runtime. For the user´s convenience, appropriate MSIPC function skeletons are generated for all modules for which no functional description can be found. So the user has to fill in only the functional description of that primitive, whereas the module´s I/O description is automatically derived to fit into the system.

Because of the very efficient modelling of all synchronous parts of the SUT and simulation as SDF system, a remarkable acceleration of simulation speed results, by this allowing image sequence simulations within reasonable time.

	MSIPC	Behavioural VHDL *	Logic Simulation
temporal resolution	clock period	1 ns	0.1 ns
processing time **	1	200	1500

*) Synopsys VHDL - simulator
**) relative data

Figure 9: Comparison of simulation duration

6. Mixed Mode Simulation within MSIPC

MSIPC offers the possibility of mixed mode simulation. In this case, the generated ´C´ program interacts with external simulators in the sense of synchronous simulator coupling [15]. To prepare for such a mixed mode simulation, the whole circuit graph is searched for modules, which the user has specified for external simulation. To minimize the amount of data exchange with external simulators, all neighboured modules assigned to one specific external simulator, are grouped and form a socalled Simulation Group (SG). This SG is treated as a whole by the external simulator.

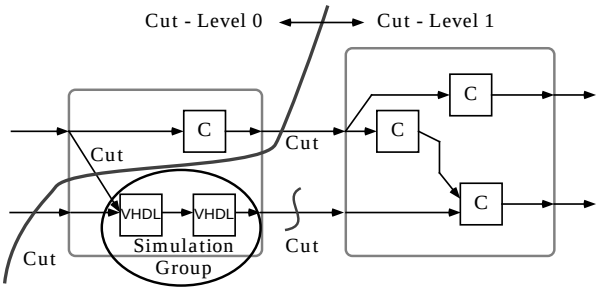

Figure 10: Building of a Simulation Group and introduction of Cuts

For the actual data exchange, two concepts are provided: the serial and the parallel mixed mode simulation.

With the serial mixed mode simulation, the circuit graph is cut at connections between MSIPC modules and external modules. Because parallel paths have to be cut, too, a BFS algorithm is applied. All cuts are done on the highest possible hierarchy level, which guarantees a minimum number of cuts (cf. fig.: 10). As a result, the whole system is split up into several sequential subsystems, ordered by the socalled cutlevel. During simulation, first all data are passed through the subsystem with cutlevel 0, and the results are stored in files. These files are read by the external simulator specific input interface modules, which together with the output interface modules form a testbench for the SG under test (SGUT). The results are again stored as files, which now form the input stimuli for the subsystem with cutlevel 1. This procedure is repeated until the whole system is evaluated.

This method can easily be implemented with every external simulator, even if the simulators run on different workstations, but has the disadvantage, that mixed mode simulations are impossible within circuit cycles.

Therefore a parallel mixed mode simulation method was also implemented, to overcome these restrictions. Here, the generated ´C´ program and the external simulators work as parallel processes and data are exchanged every clock cycle via standard unix pipes, as shown in figure 11. Restrictions on the circuit graph no longer hold.

The automatically generated testbench for the SGUT maps the bi-valued logic of MSIPC to the simulator specific data representation, and vice versa. Furthermore, the testbench is used to convert the synchronous dataflow of MSIPC into temporally equidistant input events by generating appropriate clock and control signals for the SGUT, so that signals are temporally expanded, i.e. held for one clock period, when given to the external simulator, and subsampled when returned to MSIPC. Figure 12 illustrates this process by showing the temporal sequence of samples resp. events at the locations indicated in figure 11.

To verify the functional description given to the external simulator, the results returned can automatically be checked by running the available MSIPC description in parallel and comparing the results. There are three possi-

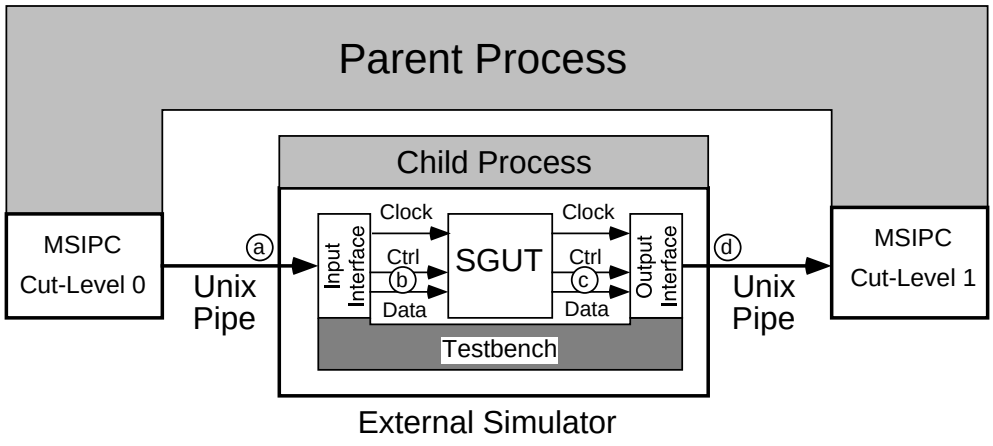

Figure 11: Parallel Mixed-Mode simulation

ble indicators, MSIPC can give to the user: identity, temporal displacement, or non-identity.

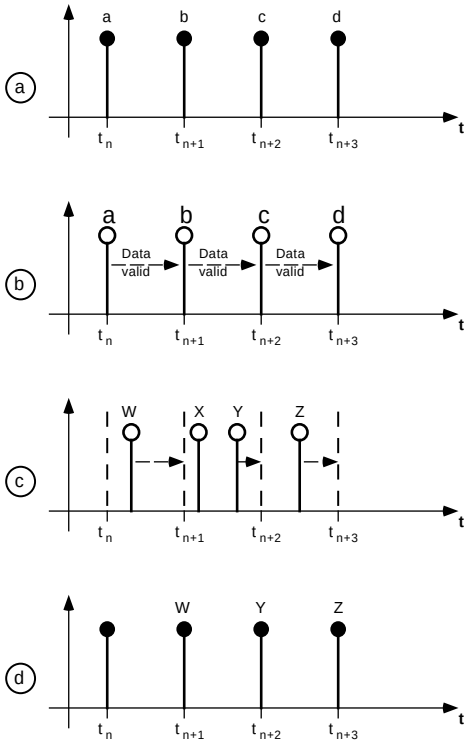

Figure 12: Temporal sequence of samples and events

7. Conclusion

For the design of video signal processing systems efficient simulation concepts are necessary to enable image sequence simulations during all stages of design flow. Since a synchronous design is commonly used for such video signal processing systems, an according simulator can benefit from an appropriate circuit model and simulation paradigm. In this paper we present the compiled code simulator MSIPC, which offers the following features:

- Simulation with clock period precision according to the SDF model, resulting in very efficient simulation of synchronous systems
- Architectural system description via netlists, which can easily be generated by schematic entry
- Mixed mode capability via coupling to external simulators is supported, allowing e.g. VHDL - simulation for single modules
- Efficient simulation of processor-based modules by optimizing cross-compiling

Several designs, most of them in cooperation with industrial partners, have been done using MSIPC, mainly in the area of format conversion [16]. The experiences confirm fast design times and easy application of MSIPC.

Currently work is done to implement an optimizing cross-compiler for VHDL, allowing a functionally identical simulation in the SDF domain, yielding a rapid increase in simulation efficiency.

References

1 Schwoerer, L.; Schröder, H.: "System Design for Digital Video Signal Processing - Relations between Algorithm, Architecture and CAD-Tools", ECCTD'93 - Circuit Theory and Design, Elsevier Science Publishers B.V.

2 CADIS GmbH, Herzogenrath, Germany, COSSAP User´s manual

3 Grötker, Th.; Zepter, P.; Meyr, H.: "ADEN: an Environment for Digital Receiver ASIC Design", ICASSP-95

4 Buck, J.; et. al. : "Ptolemy: A Mixed-Paradigm Simulation/Prototyping Platform in C++", Proc. C++ At Work Conference, Santa Clara, CA, November 1991

5 Konstantinides; Rasure: "The Khoros Software Development Environment for Image and Signal Processing", IEEE Trans. on Image Processing, VOL.3, No. 3, May 1994, pp. 243-252

6 Rammig, F.J.: "Systematischer Entwurf digitaler Systeme" S. 255 - 315, B.G. Teubner - Verlag, Stuttgart 1989.

7 Lee, E.A.; Messerschmidt, D.G.: "Synchronous Data Flow", IEEE Proceedings, September, 1987

8 Zepter, P.: "Kopplung eines VHDL Simulators an einen Simulator für Signalverarbeitungs- und Kommunikations–systeme", GME Fachberichte 11 Mikroelektronik (D. Seitzer, ed.), pp. 127-132, VDE Verlag, March 1993

9 "Network Description Language Reference", LSI-Logic Corporation.

10 Marwedel, P.: "Synthese und Simulation von VLSI-Systemen" S.30 - 32, Hanser Studien Bücher, München Wien 1993.

11 "SVP User Manual", Texas Instruments, Inc.

12 Baker, W.: "An Application of a Synchronous/Reactive Semantics to the VHDL Language", Technical Report UCB/ERL M93/10, U.C. Berkeley, December 1992

13 Sedgewick, R.: "Algorithms", Addison-Wesley, London 1983

14 Krodel, H.T.: "Verfahren zur Logiksimulation komplexer digitaler Schaltungen mit flexibler Modellierung", Dissertation 1989, Lehrstuhl für Rechnergestütztes Entwerfen der Technischen Universität München.

15 Bechtold, M., et. al: "Das Simulatorkopplungsprojekt", GME/GI/ITG - Fachtagung "Rechnergestützter Entwurf und Architektur mikroelektronischer Systeme", (Dortmund, 1990), Springer - Verlag

16 Blume,H; Schwoerer,L.; Zygis, K.: "Subband Based Upconversion Using Complementary Median Filters", 7.th Int. Workshop on HDTV and Beyond, Torino, Italy, October, 1994