1

EDGAR: A PLATFORM FOR HARDWARE/SOFTWARE CODESIGN

António J. Esteves, João M. Fernandes and Alberto J. Proença

Departamento de Informática, Escola de Engenharia
Universidade do Minho
4709 Braga codex, Portugal

Abstract

Codesign is a unified methodology to develop complex systems with hardware and software components. EDgAR, a platform for hardware/software codesign is described, which is intended to prototype complex digital systems. It employs programmable logic devices (MACHs and FPGAs) and a transputer-based parallel architecture. This platform and its associated methodology reduce the systems production cost, decreasing the time for the design and the test of the prototypes. The EDgAR supporting tools are introduced, which were conceived to specify systems at an high-level of abstraction, with a standard language and to allow a high degree of automation on the synthesis process. This platform was used to emulate an integrated circuit for image processing purposes.

Keywords: codesign, rapid system prototyping, FPLDs, transputer.

1 INTRODUCTION

All the platforms used in codesign are not universal, in the sense that not all the systems can be implemented in a straightforward way. Additionally, those platforms are generally too expensive, since they have a large number of hardware resources. If these resources are not completely used for a significant number of systems, the ratio performance/cost is extremely low.

The EDgAR (*Emulador Digital Altamente Reprogramvel*) platform was designed to achieve a high performance/cost ratio and to implement complex systems with critical time constraints, used in real-time applications (especially computer vision systems). However, the platform design was not significantly constrained by the particular aspects of these systems.

EDgAR is a FPGA-based platform that includes a transputer that can be linked to a parallel architecture. With the EDgAR platform, prototypes of complex digital systems can be obtained in a short period of time.

The recent development on the area of re-programmable components (FPLDs - Field Programmable Logic Devices) made them attractive to fast and efficiently create prototypes, because their complexity can achieve tens of thousands of equivalent logic gates, and the manufactures provide electronic CAD tools to support those components. Since the time of design and the production cost were reduced, and the FPLDs need no longer to be removed for programming, they can be used with success in codesign platforms.

The transputer is a microprocessor with communication and processing power and a simple interface. It allows the scale of parallelism, due to its capacity to be interconnected with other identical microprocessors.

Codesign is closely related to the design of systems with unreachable performance in software implementations, and systems with higher complexity than those implemented in hardware (ASICs) [1, 2].

This article is organised as follows. In section 2, the architecture of the EDgAR platform is described. The synthesis of digital systems with EDgAR is analysed in section 3, with comments to the different phases of the process: the system specification, the hardware/software partitioning, the allocation of platform resources to partitions, and the validation of the prototype obtained. In section 4 the emulation of a VLSI circuit, the GLiTCH, on EDgAR is presented.

2 THE ARCHITECTURE OF THE EDGAR PLATFORM

The structure of the EDgAR platform (figure 1) is supported by two major blocks:

i) a digital information processing unit (UPDI), that implements a parallel computation node, with communication and scalar processing power, and where the digital signals processing speed is not crucial;

ii) a programmable logic unit (ULP), containing a great amount of reconfigurable resources and whose operation speed is close to that of the circuits with fast technologies available on the market, allowing better performances than those obtained with traditional simulators.

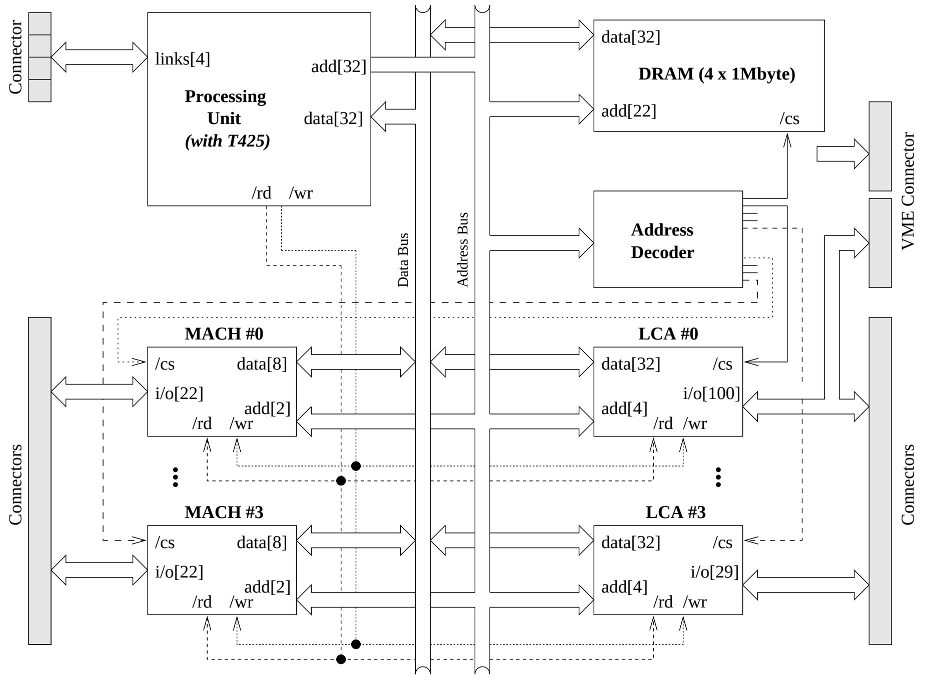

Figure 1 The architecture of the EDgAR platform.

To carry out the UPDI, the transputer (a microprocessor with communication and processing power) was selected. It allows the scale of parallelism, due to its capacity to be interconnected with other identical microprocessors, building up a network with a variable topology. This processor is also responsible for the interface with the prototype development system and for the initial configuration of the ULP components [3]. On the debugging phase, the user's interface with the platform was developed on a unit containing several TRAMs (TRAnsputer Modules) installed on a PC and using a C compiler. The connection between the unit of TRAMs and EDgAR is done by one (or more) transputer link(s), which are asynchronous. The tools available to work with the TRAMs allow to monitor the transputers of the TRAMs and EDgAR, to compile the programs and to load them to the transputers.

The ULP provides a large quantity of resources, without significantly compromising the speed of the systems being implemented. The ULP structure is based on two types of PLDs: one appropriated to implement circuits containing logic at two levels (MACHs - Macro Array CMOS High-density), while the other owning a structure organised like a matrix, suitable to implement circuits containing multi-level logic (FPGAs - Field Programmable Gate Arrays).

The present EDgAR platform version (figure 1) is implemented with a T425 transputer (a T805 could also be used), 4 Mbytes of DRAM, 4 MACHs and 4 FPGAs. The MACHs belong to the 2x0 AMD family, containing 44 pins, 64 macrocells and 32 I/O cells. The FPGAs are Xilinx LCAs that belong to the 3090A family: two FPGAs have 84 pins and the others have 175 pins. All FPGAs have 320 macrocells and 139 I/O cells.

All components are connected to common buses, using different addresses for the transputer internal and external memories, and for each of the FPGAs and MACHs. To emulate distinct digital systems on the platform, and to keep the possibility of reconfiguration by software, each MACH is connected to the buses by 2 address lines and 8 data lines, while each LCA uses 4 lines to connect to the address bus and 32 lines to the data bus. The remaining I/O pins of the MACHs and LCAs are available in connectors, allowing to emulate systems with different number of I/O signals and different size of hardware components. To scale the processing power, the transputer communication lines (links) are available outside the board. To scale the hardware resources, the VME connector can be used to link the FPGAs on EDgAR with other platforms that also have a VME bus.

3 DIGITAL SYSTEMS SYNTHESIS WITH EDGAR

The development process with the present platform runs through several phases, from the specification to the implementation, going through the simulation and test (figure 2). Next, it is explained how these phases are being incorporated on the development environment that will support EDgAR.

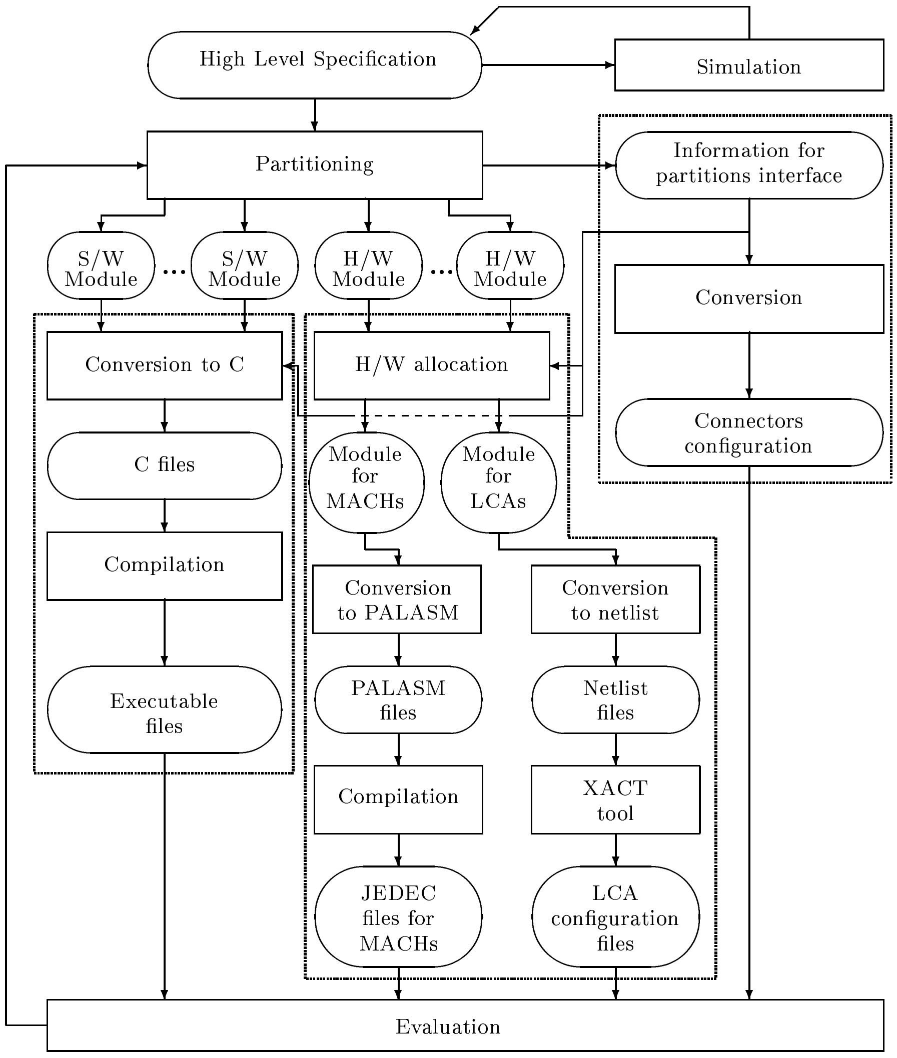

Figure 2 Methodology used for system development on the EDgAR platform.

3.1 Specification

On the codesign context, the selection of a high-level environment for system specification is being considered, which will be the basis of the specification model to be followed. The hypothesis under consideration include an FSM-based representation, the OCCAM language, a representation using Petri Nets (PNs) or the VHDL language. A high level formal representation is used to prove the specification correctness and to guarantee that this correctness is preserved in the next design phases.

The modelling of systems with FSMs has two disadvantages: (i) as a high-level notation, FSMs are not so abstract as desired, and (ii) FSMs are not appropriate to represent systems with high algorithmic complexity [4].

The OCCAM language presents the advantages of being simple, suitable for real-time representation, having potential for parallelism, a well defined semantics (based on CSP [5]) and the adequacy to represent components to be implemented on the transputer [1]. OCCAM is not a good solution, because it is not a widely used language (this is reflected in the reduced number of available synthesis tools) and it has a strong binding to the transputer processors' family, which means that it is not an implementation independent language.

PNs are a mathematical formalism used to model systems that include concurrent activities and its graphical representation can be used to animate the modelled systems. The formalism associated with PNs allows the systems validation in relation to a set of properties: determinism, deadlock freedom, conflict freedom, liveness and boundedness [6].

VHDL is a standard hardware description language used to design digital systems, allowing the model to be clearly specified, simulated and synthesised. The specifications of the systems designed with VHDL can be hierarchically structured and properly represented [7].

The joining between VHDL and PNs is considered to be an acceptable solution. This was studied and applied with success in the specification of parallel controllers [8]. An identical evaluation is being carried out on the EDgAR platform, to implement systems that are more complex than those already tested.

The specification model is influenced by the fact that the EDgAR platform implements systems asynchronously, since a completely synchronous specification model is less suitable to represent the aspects related to implementations in hardware and software, which are asynchronous by nature. Although an independent implementation specification is a goal, this is not commonly achieved.

3.2 Hardware/Software Partitioning

The hardware/software partitioning, considered to be the most complex phase on the codesign context, is a hard task to be fully accomplished by an automatic process. Usually the partitioning algorithm is fed with inputs (supplied by the designer) to assist the process. The partitioning task comprises the phases of assignment and scheduling, although some approaches use assignment only [9, 10].

The partitioning applied in EDgAR is behavioural, since it is done on the system specification. The behavioural partitioning has several advantages over the structural partitioning, but the most relevant is the fact that the impact of changes on the system's specification is smaller on the first one [11].

The approach used for partitioning belongs to the software-oriented solutions. This means that the starting point is a complete software implementation, and after parts of the system are moved to hardware based on time criteria.

The software and hardware partitions are intended to have different granularities: task level on software partitions and block level on hardware partitions. Hardware partitions are implemented with the ULP in EDgAR and the software partitions with the UPDI. Among the hardware partitions, those implemented with MACHs must be distinguished from those implemented with FPGAs.

The partitioning comprises the isolation of the parts with critical time constraints, which will result on hardware partitions; the remaining parts may result on software partitions. The definition and implementation of the communication strategies and interface between partitions is an important aspect to be considered on the partitioning phase. On EDgAR, the interface between two software partitions is implemented with memory positions and transputer channels. Virtual channels are used if the partitions are on the same processor, while physical channels are used if the partitions are on different processors. The interface between two hardware partitions uses registers and connectors, and the interface between a hardware and a software partition is implemented with the resources used in the two previously mentioned types of interface.

3.3 Synthesis of Components

The synthesis of components is divided in three main parts: the synthesis of software partitions (left block of figure 2), the synthesis of hardware partitions (central block) and the synthesis of the interface between partitions (right block). Each part can be seen as an allocation of resources that results on a configuration.

The allocation of UPDI resources to software partitions is accomplished in two phases. In the first, the high-level specification of these partitions is converted into modules on an intermediate language (C). This task requires the existence of a converter to C language, and the generated C modules are compiled to the transputer machine code.

The allocation of ULP resources to hardware partitions results in allocating to these partitions resources available in two types of PLDs: MACHs and FPGAs. The decision about which type of PLD to allocate to each module is based on the need of storage elements and the existence of critical time constraints. Partitions that need a number of storage elements higher than a critical value are allocated to FPGAs, while partitions that require a response faster than a critical value are allocated to MACHs. If both conditions arise in the same partition and it can not be partitioned again, several components are allocated to this partition.

To configure the MACH devices, the compilation and the later mapping of their resources are completed with the agreement of the hardware allocation. The result is a JEDEC file for each allocated device. The hardware allocated to the FPGAs determines their configuration. The first step to obtain this configuration is to create an intermediate format file (netlist) that will be used as input to the Xilinx Automatic CAE Tools (XACT). These tools generate the binary configuration file for each allocated FPGA, defining the device operation, but before they map, place, and route the specification.

When the system is powered on, the transputers download the configuration files to the FPGAs and establish their operation. Among the available ways to send the configuration file to the FPGA, the peripheral mode was selected, which sends the configuration on a byte basis. After the start-up, the FPGA can be reprogrammed without a physical reset of the system.

3.4 Components Verification

XACT allows for two types of simulation, in order to verify the parts of the system implemented with FPGAs: functional and timing simulations. The functional simulation detects logical errors, while the time simulation tests the functionality under different conditions, like a higher temperature, a lower power or a slower process.

The obtained prototype can be validated at a higher level of abstraction in a process called co-simulation. The co-simulation is a time consuming task that demands a huge computation power. For these reasons, it was intended to use a simulation model adapted to parallel architectures [12]. This advantage results because the co-simulation process runs on part of the same architecture that is used to implement the simulated prototypes.

4 THE EMULATION OF A VLSI CIRCUIT WITH EDGAR

The emulation of the GLiTCH chip [13], an associative processor array designed for a VLSI circuit to apply on image processing, was used as a case study, to validate the physical structure of the EDgAR platform and to explore the capabilities of the platform for codesign (figure 3).

The GLiTCH is structured on 5 blocks: an array of 64 1-bit processing elements (PEs), each one with 68 bits of associative memory (CAM), a pattern router (PBL), a video shift register (VSR) with 64x8 bits, and an instruction decoder [14].

The specification of this case study was not carried out at an high-level of abstraction: the modules to be implemented with the hardware components (MACHs and FPGAs) were specified using VIEWlogic schematics, while those to be implemented in software (transputer) were specified in C. To specify PLDs, using the ViewPLD tool from VIEWlogic, the JEDEC format and, textual descriptions in ABEL or VHDL could also be used.

Although manually done, the partitioning process used the performance of the system as the main *criterium* for partition definition, but it also used the particular characteristics of each block. Using a large granularity (block level), two candidates emerged to be implemented in hardware: the CAM and the

VSR. Since the VSR operates in two directions (columns rotation and rows shift), one of these operations would have a low performance if implemented in software. This leads to implementing the VSR in hardware. As a first approach, the CAM did not result on a hardware partition, due to its large dimensions (64x68 bits), but the software implementation did not significantly degrade the overall performance of the system. Further hardware partitions were not created as the PBL and the PEs are strongly tied to the CAM. Since the CAM resulted on a software partition, these two blocks are implemented in software too, reducing the communication cost between two partitions.

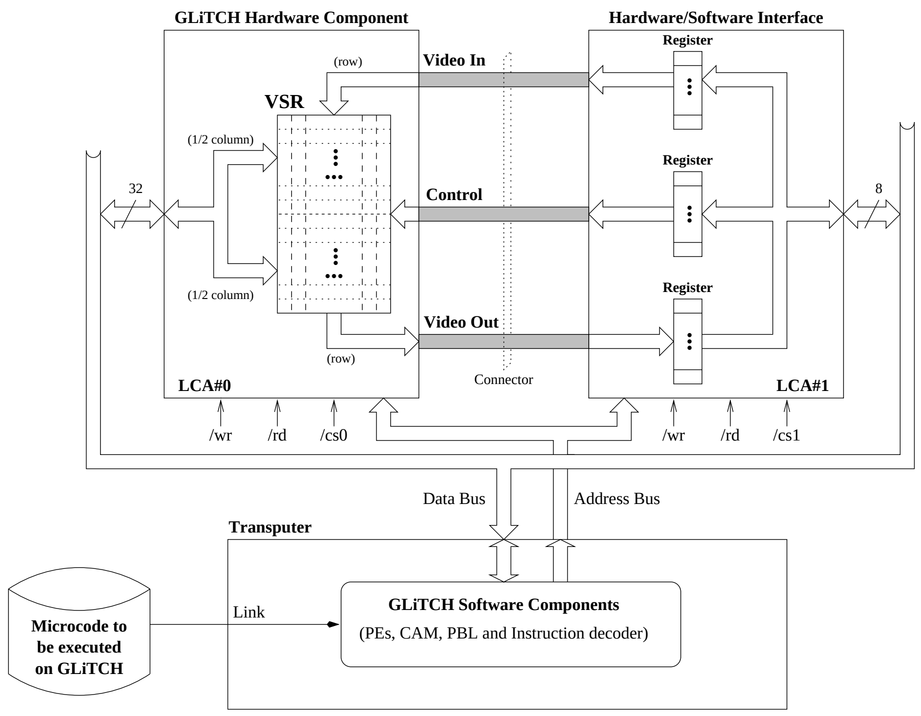

Figure 3 Hardware/software implementation of the GLiTCH on the EDgAR platform.

The VSR is a bi-dimensional shift register organised as a matrix. The GLiTCH uses an 8-bit video bus and includes 64 PEs, resulting on a VSR with 64x8 bits. The VSR functionality is represented by the operations performed on the data it stores. These operations are called SHIFT and SWAP, and correspond to row shift and column rotation, respectively. The SHIFT operation is regulated by the frequency of an external clock. This operation registers the 8 bits of the video input on VSR's row 63, it shifts all rows one position down, and row 0 is sent to the video output. The SWAP operation handles 64-bit columns, but the present implementation of this operation is done in two steps, because the data bus that connects the LCAs with the transputer is 32-bit wide. The SWAP operation reads column 0 to the data bus (parallel read), it registers the content of data bus on column 7, and it simultaneously rotates all the columns one position to the right (parallel write/column rotate). The SWAP operation is used to implement some GLiTCH instructions: rotate_image, extract_image and all others that use IMAGE as a parameter.

The hardware components of the GLiTCH emulator (VSR) was implemented in a 175-pin LCA. Two issues made the VSR implementation difficult: (i) the large percentage of the available storage elements allocated to the VSR (8*64=512), and (ii) the constraints imposed by the fixed position, on the PCB, of some signals (data, address and control). These two aspects result in problems: incomplete automatic routing of the LCA, long accumulated delays and fan-out problems. Some of these problems should be reduced, or even eliminated, if the VSR is implemented with 2 LCAs. However, this option would increase the cost associated with communication between the two VSR halves, and the chosen approach has the advantage of testing the utilisation of the LCAs on the limits (more than 80% of logic used).

To implement the software components of the GLiTCH emulator (PEs, CAM, PBL and instructions decoder blocks), the starting point was their functionality. The functionality of these blocks was described in ANSI C, but the emulator has some minor aspects especially developed for transputers [15]. The software components, running on a single transputer, fully implement the GLiTCH microinstructions, except those microinstructions using the VSR. If better performance is required, the parallel architecture connected to the platform should be used. Each microinstruction has one sub-operation executed by the PBL and one sub-operation executed by the PEs. The PBL sub-operation is executed before the PEs sub-operation (except in microinstructions that write to the CAM).

The interface between the hardware and the software components was implemented with 3 types of EDgAR resources: an 175-pin LCA, the data/address buses and the connectors. The FPGA is used to implement the VSR SHIFT operation, which is not synchronised by the same clock as the other GLiTCH components. The connectors establish the communication between the FPGA used in interface and the FPGA that implements the hardware partition.

The input to the GLiTCH emulator is the microcode of the several microinstructions to execute. For better interface with user, an assembler was developed.

5 CONCLUSIONS AND FUTURE WORK

The GLiTCH emulation led to the conclusion that the performance of the implemented systems strongly depends on the ULP resources allocated. The performance also depends on the hardware/software partitioning procedure. It is not expected that the level of abstraction used to specify the systems will significantly influence the final performance. The case study also demonstrates that EDgAR implements complex systems without scaling the platform, using connections to other platforms or computing nodes. The platform architecture was simplified because the transputer requires a simple interface and it supports the debugging of the architecture where it is included.

With the emulation of the GLiTCH processor using hardware and software components, significant improvements were obtained on the execution time of the instructions that use the VSR. Since the design time was not increased in the same proportion, it is demonstrated that the platform can be used successfully for hardware/software codesign.

The case study results in a hardware implementation without using any MACH, because the MACHs are devoted to implement fast combinational logic blocks, which are not present in the VSR. The validation of the MACHs was verified through other smaller sized systems.

When identical modules were implemented with both types of FPLDs, the delays achieved with FPGAs were bigger than the delays obtained with MACHs. This guarantees that, when both types of FPLDs are included on the platform, better performance is possible, since each device type is adequate to implement distinct parts of the system. This idea is represented by the two criteria used on the hardware partitions generation.

After the promising results obtained with EDgAR, the future work will be directed towards the integration on a more ambitious platform, which will include copies of an updated version of EDgAR, a microprogrammable unit based on a 16-bit sequencer and the MIMD transputer-based architecture. The VHDL language will be used as the unified specification notation, to improve the communication between the different phases of the codesign process: hardware/software partitioning, parallel co-simulation and synthesis.

While several tools for automatic synthesis are available, there is much work to be done for automatic partitioning and co-simulation. Future work includes: (i) the definition of a more complete partitioning strategy that automatically generates representations of the modules being implemented in FPLDs, the microprogrammable unit or the different transputer of the parallel architecture, and (ii) the development of a co-simulator that runs on the parallel architecture, whose main goal is to speed up the simulation, a generally time-consuming process.

REFERENCES

[1] Mike Spivey and Ian Page. *How to Design Hardware with Handel*, Oxford University Computing Laboratory, December 1993.

[2] Rajesh K. Gupta and Giovanni De Micheli. *System-level Synthesis using Re-programmable Components.* In *Proceedings of the European Conference on Design Automation*, pages 2–7, Brussels, Belgium, February 1992.

[3] António Joaquim Esteves. Rapid Prototyping of Digital Systems. Technical report, Dep. Informática, Universidade do Minho, Braga, Portugal, July 1994.

[4] M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, and A. Sangiovanni-Vincentelli. *A Formal Specification Model for Hardware/Software Codesign.* Technical Report ERL-93-48, University of California - Berkeley, June 1993.

[5] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, 1985.

[6] Manuel Silva and Robert Valette. Petri Nets and Flexible Manufacturing. In G. Rozenberg, editor, *Advances in Petri Nets 89*, volume 424 of *Lecture Notes in Computer Science*, pages 376–417. Springer-Verlag, Berlin, Germany, 1990.

[7] Douglas L. Perry. *VHDL.* McGraw-Hill, 1991.

[8] João Miguel Fernandes. Petri Nets and VHDL on the Specification of Parallel Controllers. Master's thesis, Dep. Informática, Universidade do Minho, Braga, Portugal, July 1994.

[9] Rolf Ernst, Jorg Henkel, and Thomas Benner. *Hardware-Software Cosynthesis for Microcontrollers. IEEE Design & Test of Computers*, December 1993.

[10] Asawaree Kalavade and Edward Lee. *A Global Criticality/Local Phase Driven Algorithm for the Hardware/Software Partitioning Problem.* In *Proceedings of the 3rd International Workshop on Hardware/Software Codesign*, pages 42–48, Grenoble, France. IEEE Computer Society Press, September 1994.

[11] Frank Vahid. *A Survey of Behavioral-Level Partitioning Systems.* Technical Report 91-71, Dept. of Information and Computer Science, University Of California, Irvine, October 1991.

[12] W. Billowitch. Simulation Models for Support Hardware/Software Integration. *Computer Design*, 1988.

[13] Henrique D. Santos, José C. Ramalho, João M. Fernandes, and Alberto J. Proença. A heterogeneous computer vision architecture: implementation issues. *Computing System in Enginneering*, 6(4/5):401–8, 1995.

[14] A. W. G. Duller, R. H. Storer, A. R. Thomson, E. L. Dagless, M. R. Pout, and A. P. Marriot. Design of an Associative Processor Array. *IEE Proceedings*, 136, 1989.

[15] António Esteves. Emulation of an Associative Processor Array with EDgaR Platform. Technical Report UMDITR9602, Dep. Informática, Universidade do Minho, Braga, Portugal, May 1996.