A CPU for Educational Applications Designed with VHDL and FPGA

Nelson V. Augusto, Mario L. Côrtes
and Paulo C. Centoducatte
IC/UNICAMP

Relatório Técnico IC–96-10

Outubro de 1996

A CPU for Educational Applications Designed with VHDL and FPGA

Nelson V. Augusto, Mario L. Côrtes
and Paulo C. Centoducatte[*]
IC/UNICAMP[†]

Abstract

This paper presents a CPU designed for educational applications to be used in the Computer Architecture Laboratories, at IC/UNICAMP. The CPU will be used as a basic plataform in order to introduce to the students novel design concepts such as VHDL, Logic Synthesis and FPGAs as well as a means to explore computer architecture characteristics and functionality. The paper also presents a few experiment possibilities, where the students incorporate new functions to the basic CPU using advanced techniques. The experiments complexity, the required design changes and their effects on the FPGA programming are also discussed.

1 Introduction

Before the arrival of large programmable logic devices (PLDs), digital design could be implemented either with off-the-shelf parts (SSI, MSI), or with low and medium size PLDs or even with ASICs (full custom, standard cells or gate arrays). A typical design could use a mix of these types of circuits. The use of ASICs, clearly the most sofisticated among them, was determined by constraints such as cost, turn around time and functionality requirements.

In brazilian university teaching laboratories, the typical digital design was implemented with prototype boards populated with off-the-shelf parts and small PLDs or PALs with wirewrapping interconnections. The design of more complex systems was limited to research projects, often implemented in full custom, with all its problems, high cost, long turn around times, development complexities and lack of proper design automation tools.

With the availability of large PLDs a wide world of possibilities arises. It became possible to design increasingly large systems reaching 100,000 gates on a single chip. The development cycle can be significantly shortened due to the use of powerful electronic design automation (EDA) tools, envolving the use of high level description languages such as VHDL, followed by simulation and logic synthesis. In addition to that, programming can be done by the designer, without the need of long fabrication delays. Other advantage is the availability of programming technologies that allow the repeated use of the same component

[*]{nelsonva,cortes,ducatte}@dcc.unicamp.br

[†]Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

several times and rapid prototype development, with the production of many versions of a design in a short time.

The decision to use custom components or ASICs to implement a design is based on the following cost/benefit criteria:

development time: the development cycle of large PLDs is shorter than that for custom circuits because they require neither complicated physical design nor post design fabrication;

cost: the low non recurring engineering (NRE) costs of large PLDs makes it more attractive for small/medium scale production, even considering the considerably high cost per part;

performance and integration scale: in general, custom circuits present better performance, lower internal delays due to their better integration density and shorter internal interconnection wires;

special structures: some special structures are not widely available in PLDs and can only be implemented in custom circuits.

1.1 The Use of VHDL, Logic Synthesis and FPGAs at IC/UNICAMP

IC/UNICAMP, that has used VHDL and logic synthesis as research tool ([Krüger 93] and [Alexandrino 93]), started to explore FPGAs[1] as target technology to implement systems under investigation [Adário 96]. The next step is the use of these technologies in teaching labs at the university.

In a teaching lab the use of PLDs of FPGAs have several advantages:

- relatively complex designs can be implemented without the inconvenience of high part count, reducing assembly problems with bad interconnection of large number of signals; circuit debugging is also facilitated since the typical difficulty with prototype boards in diagnosing design and assembly errors no longer occurs;
- reduced cost with connectors, board and parts;
- possibility of reuse of expensive components, when FPGA programming technologies such as SRAM, EPROM or EEPROM are used.

The computer architecture laboratory at IC/UNICAMP is equipped with EDA tools that cover the complete development and implementation cycles of FPGAs, provided by the educational programs of Mentor Graphics and Altera Corporation. Undergraduate students are first exposed to these tools in the logic design laboratory where they develop and implement simple designs such as multipliers, dividers, memory testers and frequency meters. In the computer architecture laboratory students will work with the CPU described in this paper

[1]The acronym FPGA is used in this paper to refer to large field programmable logic devices.

in order to practice the basic operation concepts of its internal structures, the data path and control unit. They will also use the presented design as a basis to grow and incorporate new features, expanding the CPU functionality. Possible expansions are: implementation of interruption, design of a register file and DMA interfacing circuitry.

This paper presents the design of a CPU for teaching applications according to the design criteria described in Section 2. The CPU design (Section 3) comprises the definition of a basic instruction set and the design of the data path and control unit. The design, after being described and simulated in VHDL, was synthesized to a FPGA implementation (Section 4). The basic design results are summarized in Section 4.3 and possible expansions and improvements to be conducted by students in the laboratory are described in Section 5.

2 Design Criteria for the CPU

The CPU has to have minimal functionality and yet be simple:

Functionality: The CPU has to be able to execute real programs. It has to have a basic set of functions such as reading instruction and data from external devices, executing arithmetic/logic operations and writing data onto external devices. For simplicity, memory-mapped I/O was chosen as a means for interfacing with external devices. At least some sort of basic branch capability has to be present in the instruction set, both inconditional and conditional.

Simplicity: Data and address busses have the same width. The reading operation of an opcode, a word or an address can be acomplished in a single memory access. That choice simplifies the design of the data path and the control unit.

Having decided that both data and address bus have the same width now one has to decide what is the minimal bus width to allow for the execution of reasonably sized programs. 8 bits bus width was discarded because that would limit the maximum program size in 256 bytes, which is unnacceptably small. Considering that increasing the bus width causes no extra effort in the high level design using VHDL and also that the parts that will be used to map the design have a high pin count packaging it was decided to adopt 16 bit-wide busses. This value offers 64K words addressing space, that in this architecture is enough to acomodate programs larger than 24K instructions long.

Another simplification was restricting the CPU to implicit and direct addressing modes, only. In the arithmetic and logic operations with a single operand, this is implicitly the accumulator. In the operations that require two operands the CPU has a one-address architecture, the first operand being implicit the accumulator and the second one directly addressed.

3 CPU Design

The CPU architecture is shown in figure 1. Two 16–bit wide busses and four signals provide interface with external devices. ABus and DBus are the address and data busses, respectively.

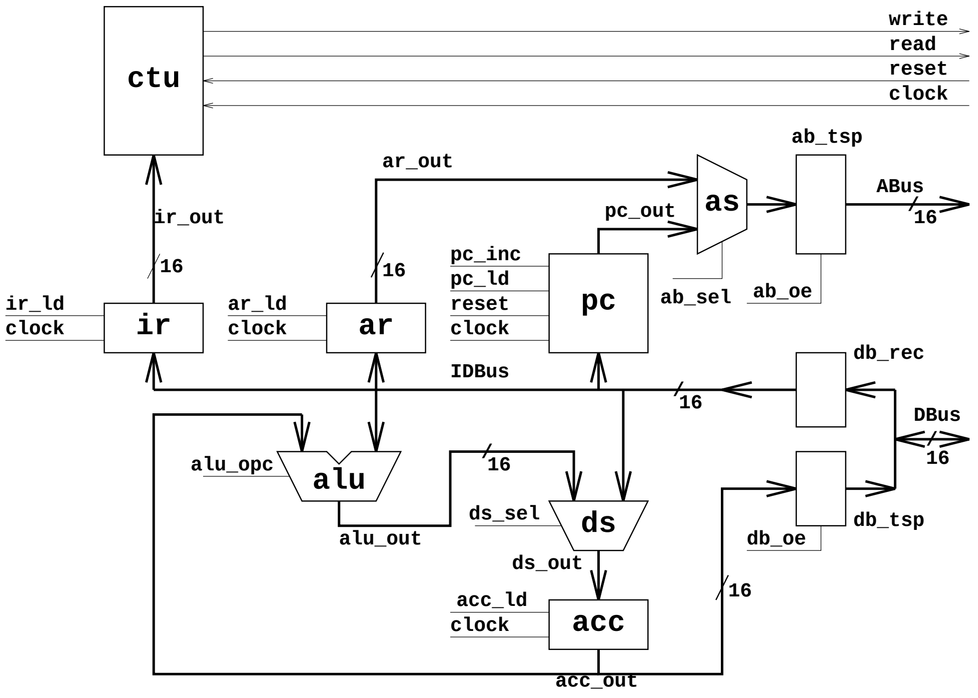

Figure 1: CPU Block Diagram.

The CPU operation is synchronous : clock rising edge determines the beginning of every sequential operation. **Reset** signal provides sequential circuits initialization. Its operation is asynchronous to assure immediate effects and to avoid the undesireble activation of control signals like write during reset period. It is recommended to keep the **reset** signal at high level for a minimum of two consecutive clock cycles at power-on to assure CPU correct initialization.

Read and **write** are output signals acting as interface with external devices, either memory or I/O devices since I/O operatins are implemented using memory–mapped IO technique.

The data path and registers are also 16–bit wide. The CPU has a program counter (**pc**) and the following registers: instruction register (**ir**), address register (**ar**) and an accumulator (**acc**).

3.1 The Basic Instruction Set

Instructions are classified into three groups, according to their function :

- logic-arithmetic;
- data transfer;
- program sequence control;

For logic-arithmetic instructions, the accumulator (**acc**) is simultaneously an operand and the destination register. If a second operand is required, as for in the **add** instruction, it has to be fetched from memory and its address must follow the opcode. For data transfer instructions, the data source or destination address must be provided. Finally, in control instructions, it is necessary to know the target address. In this way, there are two instruction formats one–word and two–word long instructions (2).

Table 1 presents the instruction set for the basic CPU .

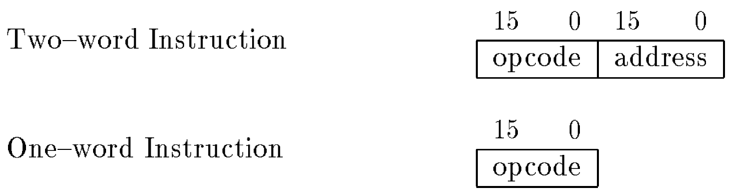

Figure 2: Instruction Formats

Logic–Arithmetic		Opcode	Words
add	$acc \leftarrow acc + Mem[Address]$	F001	2
and	$acc \leftarrow acc\, And\, Mem[Address]$	F002	2
not	$acc \leftarrow Not(acc)$	F003	1
or	$acc \leftarrow acc\, Or\, Mem[Address]$	F004	2
shl	$acc(i) \leftarrow acc(i-1)$, $acc(0) \leftarrow$ "0" $i = 15 \ldots 1$	F005	1
shr	$acc(i-1) \leftarrow acc(i)$, $acc(15) \leftarrow$ "0" $i = 1 \ldots 15$	F006	1
sub	$acc \leftarrow acc - Mem[Address]$	F007	2
Data Transfer			
lda	$acc \leftarrow Mem[Address]$	0001	2
sta	$Mem[Address] \leftarrow acc$	0002	2
Program Control			
jmp	$pc \leftarrow Address$	0003	2
jng	$pc \leftarrow Address\; if\; acc(15) =$ "1"	0004	2
hlt	*Halt*	0005	1

Table 1: Instruction Set[2]

[2]Mem[Address] denote the contents of memory location "address".

3.2 The Control Unit Design

The execution of an instruction consists of a sequence of elementary operations, performed at hardware level, the micro-operations which can be combinational or sequential. Combinational micro–operations define the function of a combinational circuit, such as the operation performed by the **alu** or the input selected by **ds**. Sequential micro-operations define state changes or data updating conditions in sequential circuits, such as Finite State Machines (FSM) or registers. Micro-operations can be executed in parallel during a single clock cycle.

Instruction cycle can be defined as the sequence of micro–operations required to execute one single CPU instruction. It can be decomposed into fetch cycle and execution cycle. Execution cycle can be decomposed into opcode decoding, operand fetch from memory and instruction execution[Hayes 88].

For the purpose of the Control Unit design, instruction cycle was divided in five phases:

1. opcode fetch from memory;
2. opcode decoding;
3. operand address fetch;
4. operand fetch;
5. execution;

Some instructions do not require all five phases in order to be executed and the micro-operations performed at each phase are, in general, diferent from one instruction to another.

Figure 3 shows, for each CPU instruction, the phase sequence needed for its execution. Each rectangle represents one phase that is the result of the simultaneous execution of several micro-operations, described inside the rectangle. Each phase lasts exactelly one clock cycle. It is important to notice that combinational micro-operations are active during the whole clock cycle whereas the sequential ones are active only at clock rising edge, which marks the ending of the current phase and the beginning a the new one.

As an example, the five phases for the **add** instruction are:

1. opcode fetch from memory: a memory read operation is performed. The adress is provided by **pc** and the data fetched via DB**us** is stored into register **ir**;
2. opcode decoding : performed by combinational circuits inside the Control Unit;
3. operand address fetch : a memory read operation is performed. The address is provided by **pc**, and the fetched data is stored into **ar**;
4. operand fetch : memory read operation, addressed by **ar**. Since this memory read is extended until phase 5, no storing is required;
5. **add** execution: the Control Unit generates the operation code for the **alu**, which adds the contents of **acc** and the value present at DB**us** (the operand from memory). The result is stored into **acc** at the end of this phase.

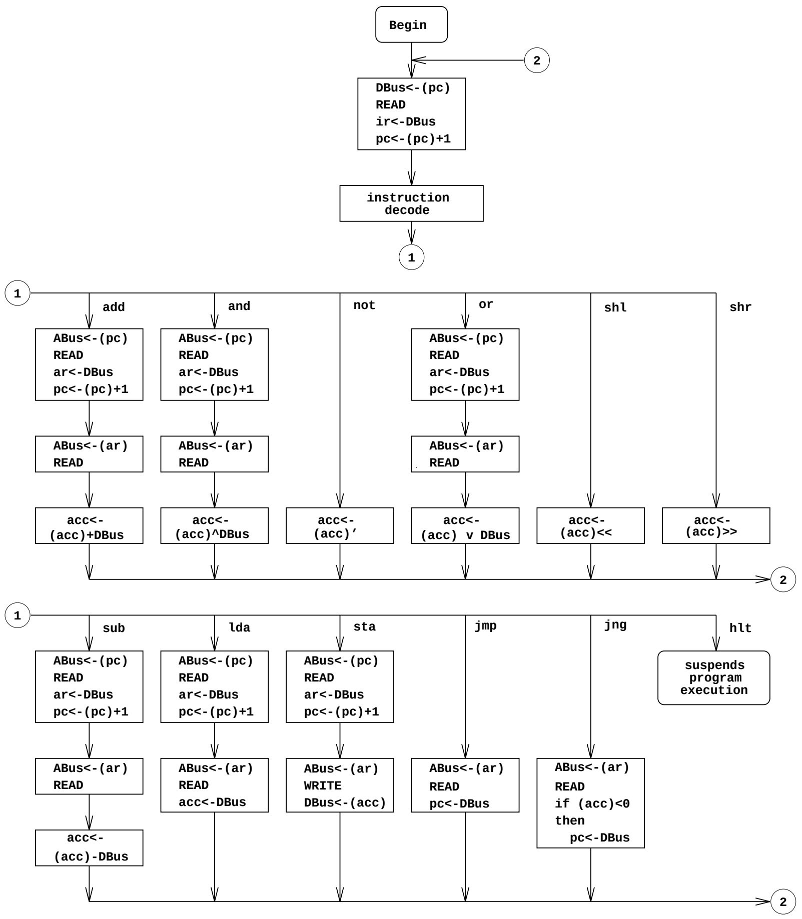

Figure 3: Micro–operations Execution Flowchart.

The design of enhancements to the CPU functionality can begin with their mapping into the execution flowchart. For example, to create a new instructions it is necessary to create a new branch in the execution phase.

The CPU Control Unit was modeled as a FSM. The analysis of the micro-operations flowchart, discussed in begin of the section 3.2, gives elements to define the states and the state transition sequences needed to model the State Machine. The State Transition Diagram for the Control Unit FSM is shown in figure 4. Seven states where defined: R, S0, S1, S2, S3, S4 and H. States S0 to S4 corresponds to the normal CPU operation; state R represents the CPU resetting and H, the hold state, when CPU operation is disabled. Transitions between states are synchronous with respect to the **clock** signal, except for transitions to R state, which is asynchronous for the reasons already discussed in section 3.

Table 2 lists the Control Unit signals and gives the active signal for each state. The column I/O indicates whether a signal is an input or an output; column I/E indicates

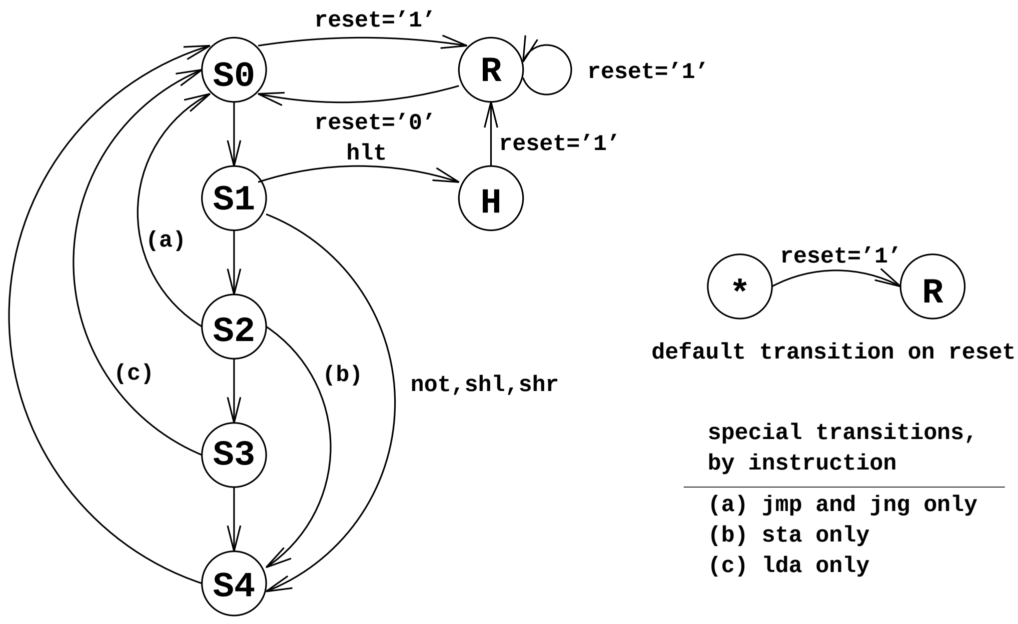

Figure 4: The State Machine Transition Diagram of CPU

whether the signal is internal or external.

4 About the CPU Description Process

4.1 VHDL Description

The CPU was completely described using VHDL. The design has two hierarchical levels: at the bottom level, each logic element presented in figure 1 was modeled using synthesis oriented constructions, either RTL (Register Transfer Level) or behavioral architecture; at the top level, the components were integrated into a structural architecture. As part of the modeling process, each component was individually compiled, simulated at VHDL model, implemented in the target FPGA and post simulated after placement and routing.

4.2 FPGA Selection

The selection of a FPGA is based on an estimate of the required number of I/O pins and logic elements, in terms of for the CPU described here. The figures are : 600 gates for combinational circuits, 67 flip-flops and 38 I/O pins.

Another characteristic to be considered is the technology of memory elements used to store the configuration. SRAM based devices were selected devices due to case of reprogrammability.

The combined result of these factors resulted in selecting Altera's FLEX8000 family, in

Signals	R	S0	S1	S2	S3	S4	H	I/O	I/E	Description
reset								I	E	
opcode								I	I	
acc-msb								I	I	accumulator signal
alu-opc						*		O	I	
read		√		√	√	*		O	E	
write				*				O	E	
pc-inc		√		√				O	I	pc← (pc)+1
pc-ld						*		O	I	pc ← DBus
ar-ld				√				O	I	ar ← DBus
ir-ld		√						O	I	ir← DBus
acc-ld					*	√		O	I	acc← ds output
ds-sel					*			O	I	ds input select
as-sel					√	*		O	I	as input select
ab-oe		√		√	√	*		O	I	as-tsp enable
db-oe				*				O	I	ds-tsp enable

Table 2: Control Unit Signals [3]

particular, the EPF8282 device. The Altera part EPF8282 is a good choice to meet these requirements.

4.3 Results

CPU simulation using Altera's MAX-PLUS II software indicates the maximum clock frequency of 12.5 MHz. An equivalent EPROM based device reaches 25 MHz.

The original circuit used about 85% of the logic blocks and 58% of the I/O pins. Minor changes in the CPU functionality can be implemented on the same device but the more complex modifications will require a larger device, like EPF8452 or EPF8636.

5 Expanding the CPU Functionality

This section presents a list of possible Computer Architecture Laboratory experiments in order to expand the CPU functionality.

The changes can be made with no effects on the opcodes.

E1 : wait-state : use an aditional input signal and modify the VHDL code as shown in figure 5 for one state transition. Change the VHDL code to implement the transitions from states

[3]* It depends on the instruction being executed

```
          :
          :
elsif (current_state=R) then
  next_state <= S0;
elsif (current_state=S0) then

  next_state <= S1;
elsif (current_state=S1) then

          :
          :
          :
          :
          :
          :
```

(a) Original Code

```
                  :
                  :
 elsif (current_state=R) then
   next_state <= S0;
 elsif ((current_state=S0) and
       ( ready='0')) then
   next_state <= S0;
 elsif ((current_state=S0) and
       ( ready='1')) then
   next_state <= S1;
elsif  ((current_state=S1) and
        ( ready='0')) then
                 :
                 :
                 :
```

(b) Modified Code

Figure 5: Adding *Wait-state* feature

E2 : Stack : include a bidiretional counter (stack pointer) and two new instruction (**push** and **pop**);

E3 : Sub-routine call : include two new instructions (**call** and **ret**);

E4 : Interrupt : use an aditional pin for interrupt request and a flip-flop for interruption request memorization. Vectorized interrupt could be the next level of extension.

E5 : Register File : substitute the single accumulator by a register array. Add selection logic and instructions to use the new registers.

E6 : DMA support : add two pins and modify the Control Unit code to support a protocol like hold/holda.

E7 : Microprogrammed Control : this is a more complex design change, consisting in exchanging the Hardwired Control Unit by another, a micro-programmed one. If an external EPROM is used as micro-program memory, about 20 aditional I/O pins are required.

	Structures/Resources	Altered Area	Blocks	Pins
E1	1 state	control unit	1	1
E2	1 counter , 1 select 2 instructions (**push** and **pop**)	data path control unit	100 3	
E3	2 instructions (**call** and **ret**)	unit control	3	
E4		unit control	3	1
E5	N registers	data path control unit	O(N)	
E6	1 state	control unit	1	2
E7	micro–program memory 1 control unit	control unit	O(N)	20

Table 3: Possibles Computer Architecture Laboratory Experiments.

Table 3 summarizes, for each proposed modification, the amount of logic blocks, I/O pins and additional structures required.

6 Conclusions

This article described a simple CPU with enough functionality to be used in Computer Architecture classes. The circuit was synthesized to a FPGA from a VHDL description allowing students to expand its functionality using the same design methodology.

The implementation mapped the CPU VHDL description to an Altera FPGA (EPF8282), with 84-pin PLCC packaging. The FPGA usage percentage was 59%, in terms of pins and 85% in terms of logic blocks. The maximum operating frequency was 12 MHz. The fact that the resulting machine is slow does not have any negative effect on the CPU utilization for educational purposes. The unused resources on the FPGA are sufficient to implement the functionality expansions described in Section 5.

The implementation of the printed circuit board that is going to receive the CPU, local memory and glue logic, is under development. Such system is planned to be connected to a IBM-PC compatible computer bus, which will provide facilities for keyboard, monitor, disk and interface to I/O devices. However, the educational use of this CPU can start immediately, since much can be done by the students in terms of VHDL design and simulation of the introduced functionality expansions. We also have plans to develop a loader for the CPU object programs.

7 Acknowledgements

The authors would like to thank Prof. Nelson Castro Machado [Machado 95], from UNICAMP Computing Institute, for a number of valuable suggestions regarding the CPU architecture, design and its possible expansions; to Prof. Furio Damiani, from Electrical Engineering Institute, UNICAMP, for the suggestions regarding VHDL modeling and to Carlos Kruger, from CPqD-TELEBRAS, for all the help with the use of EDA tools and VHDL modeling. This work would not be possible without the tools and resources provided by Altera Corporation and Mentor Graphics Corporation, through their Educational Programs, and the help of their brazilian representatives, União Digital and Hitech, respectively.

References

[Adário 96] Alexandro M. S. Adário, Mario L. Côrtes e Neucimar J. Leite, *Considerações sobre a Síntese em FPGAs de Células de Processadores Matriciais,* XXIII SEMISH, pp. 369 – 380 — Recife 1996

[Alexandrino 93] Josemir C. Alexandrino e Mario L. Côrtes, *Uma Implementação em VLSI para o Reconhecimento de Imagens,* XX SEMISH, Florianópolis, setembro de 1993

[Altera] Altera Corporation *MAX + PLUS II Gatting Started and Altera 1995 Data Book*

[Hayes 88] John P. Hayes *Computer Architecture and Organization, 2nd Edition,* McGraw-Hill, 1988

[Krüger 93] Carlos G. Krüger e Mario L. Côrtes *Modelamento, Simulação e síntese com VHDL,* Relatório Técnico DCC/UNICAMP - 19/93

[Machado 95] Nelson C. Machado, *Class notes and perssonal communications*

[Mentor] Mentor Graphics Corporation, *Trainning Workbooks in System–1076 and Autologic I,*

[Perry 94] Douglas Perry, *VHDL, 2nd Edition,* McGraw-Hill, 1994

Relatórios Técnicos – 1995

95-01 **Paradigmas de algoritmos na solução de problemas de busca multidimensional,** *Pedro J. de Rezende, Renato Fileto*

95-02 **Adaptive enumeration of implicit surfaces with affine arithmetic,** *Luiz Henrique de Figueiredo, Jorge Stolfi*

95-03 **W3 no Ensino de Graduação?,** *Hans Liesenberg*

95-04 **A greedy method for edge-colouring odd maximum degree doubly chordal graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

95-05 **Protocols for Maintaining Consistency of Replicated Data,** *Ricardo Anido, N. C. Mendonça*

95-06 **Guaranteeing Full Fault Coverage for UIO-Based Methods,** *Ricardo Anido and Ana Cavalli*

95-07 **Xchart-Based Complex Dialogue Development,** *Fábio Nogueira de Lucena, Hans K.E. Liesenberg*

95-08 **A Direct Manipulation User Interface for Querying Geographic Databases,** *Juliano Lopes de Oliveira, Claudia Bauzer Medeiros*

95-09 **Bases for the Matching Lattice of Matching Covered Graphs,** *Cláudio L. Lucchesi, Marcelo H. Carvalho*

95-10 **A Highly Reconfigurable Neighborhood Image Processor based on Functional Programming,** *Neucimar J. Leite, Marcelo A. de Barros*

95-11 **Processador de Vizinhança para Filtragem Morfológica,** *Ilka Marinho Barros, Roberto de Alencar Lotufo, Neucimar Jerônimo Leite*

95-12 **Modelos Computacionais para Processamento Digital de Imagens em Arquiteturas Paralelas,** *Neucimar Jerônimo Leite*

95-13 **Modelos de Computação Paralela e Projeto de Algoritmos,** *Ronaldo Parente de Menezes e João Carlos Setubal*

95-14 **Vertex Splitting and Tension-Free Layout,** *P. Eades, C. F. X. de Mendonça N.*

95-15 **NP-Hardness Results for Tension-Free Layout,** *C. F. X. de Mendonça N., P. Eades, C. L. Lucchesi, J. Meidanis*

95-16 **Agentes Replicantes e Algoritmos de Eco,** *Marcos J. C. Euzébio*

95-17 **Anais da II Oficina Nacional em Problemas Combinatórios: Teoria, Algoritmos e Aplicações,** *Editores: Marcus Vinicius S. Poggi de Aragão, Cid Carvalho de Souza [Not available.]*

95-18 **Asynchronous Teams: A Multi-Algorithm Approach for Solving Combinatorial Multiobjective Optimization Problems,** *Rosiane de Freitas Rodrigues, Pedro Sérgio de Souza*

95-19 **wxWindows: Uma Introdução,** *Carlos Neves Júnior, Tallys Hoover Yunes, Fábio Nogueira de Lucena, Hans Kurt E. Liesenberg*

95-20 **John von Neumann: Suas Contribuições à Computação,** *Tomasz Kowaltowski*

95-21 **A Linear Time Algorithm for Binary Phylogeny using PQ-Trees,** *J. Meidanis and E. G. Munuera*

95-22 **Text Structure Aiming at Machine Translation,** *Horacio Saggion and Ariadne Carvalho*

95-23 **Cálculo de la Estructura de un Texto en un Sistema de Procesamiento de Lenguaje Natural,** *Horacio Saggion and Ariadne Carvalho*

95-24 **ATIFS: Um Ambiente de Testes baseado em Inje,c ao de Falhas por Software,** *Eliane Martins*

95-25 **Multiware Plataform: Some Issues About the Middleware Layer,** *Edmundo Roberto Mauro Madeira*

95-26 **WorkFlow Systems: a few definitions and a few suggestions,** *Paulo Barthelmess and Jacques Wainer*

95-27 **Workflow Modeling,** *Paulo Barthelmess and Jacques Wainer*

Relatórios Técnicos – 1996

96-01 **Construção de Interfaces Homem-Computador: Uma Proposta Revisada de Disciplina de Graduação,** *F'abio Nogueira Lucena and Hans K.E. Liesenberg*

96Abs **DCC-IMECC-UNICAMP Technical Reports 1992–1996 Abstracts,** *C. L. Lucchesi and P. J. de Rezende and J.Stolfi*

96-02 **Automatic visualization of two-dimensional cellular complexes,** *Rober Marcone Rosi and Jorge Stolfi*

Instituto de Computação
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

`reltec@dcc.unicamp.br`