How could we begin NOW to teach codesign ?

Paul AMBLARD, UFR IMA, Université Joseph Fourier

BP 53 X, 38041 GRENOBLE Cedex 9 FRANCE

Paul.Amblard@imag.fr

ABSTRACT

This paper presents a curriculum for the third year of University. It is an introduction to computer architecture with the double point of view of hardware and software. The question "How to introduce codesign ? " is the consequence of this approach.

INTRODUCTION

The general term "designing a digital system" deals with different activities. Building the controler of a bank notes distribution device, designing a signal processing device for satellite images, and many others are examples. Designing a micro-processor or a computer is also a part of the field. But not many people are involved in that job.

From some years it seems that a common point of some of these activities is the fact that designers have to deal with hardware and software implementations and interactions. This field is known as "codesign" [IEEE Computer , Dec. 93, Jan. 94], or "co-design" [Proceedings of the IEEE, Jul. 94]. Introducing codesign in education will be the challenge for the next years in computer science and electrical engineering departments.

Our department of computer science in the University of Grenoble presents pedagogical activities preparing students to work in such designs.

Theses activities and the underlying assumptions are focused in this paper.

These activities represent a "state of the art" at an instant. They are permanently in evolution. They also represent the result of a melting pot of ideas, implementations and willingness from several peoples during several years.

(Thanks to all !!!)

To make the presentation clear, we first briefly describe the global curriculum (part I). Then we give details about the unit where the students first discover hardware and low-level structures in computer organisation (part II). In part III, we state some questions about introduction of codesign. The fourth part (IV) gives the planned answers.

I) The global organization of undergraduate curriculum.

The french academic organization has three systems for the two first years of a scientific university curriculum. Each of them begins when people are around 18 years old. These three systems are IUT, classes préparatoires and university strictly speaking.

- 1- In "IUT" (institut universitaire de technologie) the students specialize in a technical field (computer science, electronics,..) In principle this is a terminal curriculum. Many students have a continued education, just after IUT or during the professional life.
- 2- In "classes préparatoires", the students concentrate their effort on mathematics and physics. Generally no applied sciences are introduced.
- 3- In université, (strictly speaking) the students work in math, physics and, for the case we are concerned in, computer science. Computer science is 150 hours in the year. This "heavy" introduction to writing programs and specifications uses Scheme, SQL and Pascal as programming languages.

The "Ecole Universitaire d'Informatique" of Grenoble begins after these two years. And the students graduate after 3 years. Three main orientations are presented : 1) Management oriented data processing, 2) Design of (hardware and, mainly, software) systems, 3) Preparation to research. It accepts up to 130-150 students per year. (around 30-60 for each orientation). One of the problems we have to deal with is the difference between the backgrounds of the students in the field of computer science.

The step presented here is common to the three orientations. It occurs during the first of the three years, so it is a third year level. This unit is for a majority of students the introduction to the world of gates, registers and hardware. For a lot of them it is also the first introduction to assembly language programming. This unit of "Hardware and software architectures" has 150 hours : lectures (36 h for 130 students) , paper exercises (36 h by groups of 35 students) and lab (72 h by groups of 18-30 students) . Personal activity without teachers is expected to be around 150 hours. It includes reading and preparing labs. To indicate the principles, the companion books are Tanenbaum's one and Hennesy-Patterson's. The students mainly use Tanenbaum's one. (In french translation !!)
Other units are general algorithmics and basic discrete maths. Programming is done in ADA and Prolog.

II) The Unit "Hardware and software architectures"

We present here the basic ideas and how they are implemented.

1) Algorithms, only algorithms...

When teaching hardware design, we try to keep in mind the (original ?) idea that hardware is, like software, a way to put algorithms at work. (Conf Eurochip 91 pp 26-31)

Combinational functions are functions. A finite automaton is the implementation of a kind of eternal loop. The association of a control part and a data-path part is presented as "THE" way to implement an algorithm by hardware. This is now quite common with high level synthesis. It has been taught here for years.

When teaching assembly language programming, we try to concentrate mainly on the systematic ways to transform high level constructs towards instructions and data structures of this level. Control, loops, arrays, stacks, procedures and their parameters are used.

When it is possible, we study the same algorithms implemented in hardware and software. (Syracuse sequence of integers, Bresenham's algorithm to draw a line in a bitmapped plane)

2) Try and experiment...

Architecture needs practizing. Lab represents half of the hours with teachers and the essential personal activities. Lectures always refer to the (past or future..) observations done in lab. The unit contains lab in C programming (writing a link editor for SPARC), assembly language programming, digital circuit design, micro-programming.

The results are actually tried : code generated by the link editor is executed, some VLSI circuits were processed and tested [Euro ASIC conf 91, pp 239-242]

3) Do not forget the links between hardware and software...

The teachers are the same for the different activities, the final written exam integrates the different point of views.

Machine language and interruptions constitute the core of the problems.

This approach has been introduced after a time where the department saw a splitting between hardware and software ! This is impossible at the time of co-design !

4) Some formal tools are useful in any design...

A focus is given on functions and finite automata as paradigms to describe any complex behaviours. Finite automata are studied in detail. We implement some automata by the classical synthesis technique flip-flops + gates or PLA, some others by a registers+ALU approach.

This duality will allow to introduce control part and data-path part in a quite natural way. This will be the ground for "algorithmic machines".

The more theoretical point of view about regular grammars, regular expressions and their relations to finite automata is covered in another unit.

5) Any design must be seen at different abstraction levels.

It does not make sense to describe what the electrons do when a C procedure returns some parameters !!

We present the different layers of a computer, each one being implemented from basic blocks of the underlying level. We try to avoid any gap. The basic concept of "interpreting" a language by some more basic primitives is essential. In this field, we maintain the teaching of micro-programming. It could be considered as an out-of-date implementation technique. It is a basic concept.

These 5 basic ideas constitute, in our mind, a good base to introduce "co-design".

Products

To achieve these goals the mains "objects" produced are the following :

- An environment for SPARC assembling and linking.

The students receive the skeleton of an assembler and of a link editor. They have to complete it. This is their first job with the C language.

- A customization of Sparc Assembly Simulator to simulate input/outputs and interruptions.

A software description of a pseudo input-output controller has been done. It allows to write assembly language programms with interruptions facilities. On a Unix workstation this is of course simulated, we do not trigger IRQ by a grounded wire !

- A micro-program micro-assembler

We have defined a simple computer architecture with a micro-programmed control part. The language of micro-instructions is translated in sequences of micro-commands by a micro-assembler.

- A Solo 1400 description of a micro-processor to support micro-programms.

The full gate and flip-flops description of the same micro-processor is done to allow the simulation of this machine at the gate level. (Solo 1400 is a CAD package allowing to design at the gate level a cell-array based circuit. There is also a "bridge" towards a FPGA implementation technique)

III) Introducing "Co-design".

We present here our questions about this future introduction.

1) A first question is "how to avoid the **Big Salad** ?"

Big Salad is the thing one has in mind when one confuses completely the different levels. It occurs when one beleives that a 74191 counter directly executes the C i++ instruction, or that the fetch sequence of the processor consists in reading the label of assembly language.

The language chosen to describe a codesign must make clear what is the level we are working at.

2) A second question is "how to avoid the **Big Wall** ?"

Big Wall is the thing one has in mind when one considers that the world has two kinds of people : electronicians who wire gates and programmers who write procedures. The intermediate levels between hardware and software are now a common thing. writing VHDL code, implementing functions on FPGA are not so easy to characterize as "Hard" or "Soft".

3) A third question is "how to avoid the **Big Simulator** ?"

Big Simulator is the thing one has in mind when one beleives that everything is simulated. If one uses a simulated environment to program in assembly code with interruptions, logic simulation for circuits, and VHDL simulation for interaction between hardware and software, one could beleive that the "true" world with TTL gates and reset buttons has been replaced by the Big Simulator !

IV) What will be the beginning ?

We are now preparing some new activities based on a PC board extension containing a FPGA. This will interact with programms on the PC. Some typical comparisons are to be done :

- using C or assembly language on the computer,
- communications under scruiting or interruptions between the circuit and the computer,

Up to now we plan to use a "true" Harware Description Language, and a true programming language to exhibit clearly the different functions.

We have to investigate true codesign languages. What will we use to describe the global behavior ? VHDL, Verilog, a dialect of C or C++. We would like to use the real-time Lustre [Proc of the IEEE, Sep. 91] that has a sound semantics. But it has not yet a good compiler allowing to split the global function between several components. The importance of the choice is obvious. Experiments will occur in 95-96.