Hardware and Software : The Closing Gap

C. A. R. Hoare and I. Page

Computing Laboratory, University of Oxford,
Oxford OX1 3QD, U.K.

Abstract

The study of computing science is split at an early stage between the branches dealing separately with hardware and software; and there is a corresponding split in later professional specialisation. This paper explores the essential unity and overlap of the two branches. The basic concepts are those of occam, taken as a simple example of a high-level programming language; its notations may be translated by the laws of programming to the machine code of a conventional machine. Almost identical transformations can produce the networks of gates and flip-flops which constitute a hardware design. These insights are being exploited in hybrid systems, implemented partly in hardware and partly in software. A TRAM-standard printed circuit board called HARP has been constructed for such applications. It links a transputer by shared memory with a commercial Field Programmable Gate Array. Prospects for application are discussed.

1 Correctness of Design

The design of a complex engineering product like a real time process control system is ideally decomposed into a progression of related phases. It starts with an investigation of the properties and behaviour of the process evolving within its environment, and an analysis of requirements for its optimal or satisfactory performance, or at least for its safety. From these is derived a specification of the electronic or program-controlled components of the system. The project then may pass through an appropriate series of design phases, culminating in a program expressed in a high level language. After translation into the machine code of the chosen computer, it is loaded into memory and executed at high speed by electronic circuitry. Additional application-specific hardware may be needed to embed the computer into the system which it controls. Each of these phases presents a conceptual gap, as wide and challenging as that between hardware and software. Reliability of the delivered system requires that all the gaps be closed. It is achieved not just by testing, but by the quality of thought and meticulous care exercised by analysts, designers, programmers and engineers in all phases of the design.

This has been a description of an ideal that is rarely achieved in any field of engineering practice. Nevertheless, an ideal forms the best basis for long-term research into engineering method. The goal of this research is to discover and formalise methods
which reduce the risks and simplify the routines of the design task, and give fuller scope for the exercise of human skill and invention in meeting product requirements at low cost and in good time. The goal of this paper is to convey an impression of the methods and intermediate results of the research. It illustrates them by the techniques of provably correct compilation, either to machine code or to hardware. Finally, it describes a project for mixing hardware and software implementation of programs, particularly for embedded applications.

In principle, the transition between one design phase and the next is marked by delivery of a document, expressed in some more or less formal notation. Each phase starts with study and acceptance of the document produced by the previous phase; and ends with the delivery of another document, usually formulated at a lower level of abstraction, closer to the details of the eventual implementation. Each designer seeks high efficiency at low cost; but is constrained by an absolute obligation that the final document must be totally correct with respect to the initial document for this design phase. Thus the requirements must be faithfully reflected in the specification, the specification must be fully achieved by the design, the design must be correctly implemented by the program, the program must be accurately translated to machine code, which must be reliably executed by the hardware. Although we have used different words in English to describe the correctness relation at each different level of design, we shall show that conceptually it is the same relation in all cases, namely logical implication, denoted by $\subseteq$.

When the system is eventually delivered and put into service, all that really matters is that the actual hardware delivered should meet the overall requirements of the system. This is guaranteed by a simple mathematical property of the implementation relation: it is transitive. If $P$ is implemented by $Q$ and $Q$ is implemented by $R$ then $P$ is implemented by $R$:

$$\text{If } P \subseteq Q \text{ and } Q \subseteq R \text{ then } P \subseteq R.$$  

However long the chain of intermediate documents, if each document correctly implements the previous one, the overall requirements will be correctly implemented by the delivered hardware.

We have given a very simple account of the design process, and the reason why it can validly be split into any number of phases. The account is highly abstract: in concrete reality, complications arise from the fact that each of the design documents is written in a different notation, adapted to a different conceptual framework at a different level of abstraction. For example, a requirements document for a real time system may use timing diagrams or temporal logic, a specification may use set theory ($Z$ or VDM), a design may use flow charts or SSADM, a program may use ADA or C, the machine code may be INTEL 8080 and the hardware may be described in pictures or as a netlist of components and wires. How can we be certain that a document serving as an interface between one design phase and the next has been correctly understood (i.e. with the same meaning) by the specialists who produced it as a design and the different specialists who accepted it as a specification for the next phase? The utmost care and competence in each individual phase of design will be frustrated if bugs are allowed to congregate and breed in the interfaces between one phase and the next.