Workshop Report

VLSI: Machine Architecture and Very High Level Languages

VLSI is changing electronics and computing! If we accept that the problems of 'how' to realize millions of gates on silicon are being solved, we should be thinking now about the problems of 'what' computers we are going to design. With this theme Brian Randell and David Kinniment of the University of Newcastle upon Tyne welcomed 35 leading researchers from the USA and Europe to the VLSI workshop. The aim of this workshop was to bring together the currently largely separate strands of research in IC design, parallel machine architecture and very high level programming languages. The workshop, sponsored by the UK Science Research Council, was held from 14–18 April 1980 in Lumley Castle, which is a magnificent 13th century building now established as a first class hotel. The timetable consisted of invited presentations interspersed with panel discussions dealing with various aspects of the theme.

In the first invited talk Iann Barron, Managing Director of Inmos Limited, took a pessimistic stance when discussing the impact of VLSI. Investment costs for establishing a VLSI processing capability are massive, so it follows that this capability will remain in the hands of a small number of companies with strong vested interests. It would be naive to believe that VLSI will signal the demise of the von Neumann computer. The von Neumann computer has a lot of momentum behind it, and in the world of mass production and mass marketing momentum tends to dominate such factors as elegance of style or conceptual sophistication. It is an undeniable fact that the first product to hit the marketplace corners the lion's share of that market. It is also undeniable that consumers, as well as manufacturers, are notoriously reactionary. Hence the 'market forces' which govern what chips get produced will mitigate strongly against innovative architectures. Only when it can be convincingly argued that some novel design will have universal applicability (or, more significantly, can be packaged to have universal appeal) will a break with current practices be countenanced. Setting the tone for the remainder of the workshop, this talk provoked a vigorous discussion.

The next speaker, Carlo Sequin of the University of California, Berkeley, asked what kind of components we need to order today to make it possible to build interesting machines in the future? He then went on to explore the possibility of a single, general-purpose component for building computers, within the framework of Berkeley's X-tree research project. One specific issue of this project, which started in 1977, was to define a modular component X-node from which general-purpose computing systems of arbitrary size could be built. Each X-node contains a memory, processor, control, I/O switch and buffers. The primary contender for interconnecting X-nodes into truly modular and incrementally expansible networks are tree-structures. X-tree is therefore a binary tree enhanced with additional horizontal links to form a half-ring or full-ring tree. These additional links help to evenly distribute message traffic throughout the tree and provide a degree of fault tolerance. In X-tree the children of X-node n have node addresses $2n$ and $2n + 1$ respectively.

Wayne Wilner of Xerox Corporation's Palo Alto Research Center continued the discussion of general-purpose VLSI building blocks by describing his Recursive Machine Project. Recursive Machines are based on the concept of recursion and order. A Recursive Machine (RM) consists of either a RM element or, recursively, an ordered set of RMs. A RM element has: (1) storage, in the form of variable length registers and through an I/O processor connected to secondary storage; (2) logic, in the form of a general-purpose, microprogrammable processor with writeable control store; and (3) input/output, in the form of a communication processor and bus ports through which it transmits messages on behalf of fields located within its storage. It also has two point-to-point connections through which fields migrate from one element to another. Information is represented in terms of fields, which are recursively defined to be either bracketed strings '(. . .)' of characters or bracketed strings of fields '(( ) ( ) . . .)'. Thus, a machine instruction is recursively defined to be either a string of characters or an ordered set of machine instructions. Instructions which are ordered sets have as their first instruction a string which governs the control and interpretation of the remaining instructions. An objective of this project is to remove hardwired limits on the semantics of programming languages, such as address space size, string length, universe of types, and functional forms.

H. T. Kung of Carnegie-Mellon University then took up the challenge of the design of special-purpose chips which can function as peripheral devices attached to a conventional host computer. VLSI provides opportunity for special-purpose chips. Silicon compilation will make such chip fabrication routine. However, algorithms that perform well on conventional random access computers are not always the best for VLSI implementation. Computation is cheap in VLSI; communication determines performance. Kung then described what he calls 'systolic' algorithms. Such algorithms have 'VLSI properties', namely they are implementable by a few simple cell types, whose data and control flow is simple and regular, utilizing extensive pipelining and multiprocessing. For VLSI algorithms there are a number of different simple and regular strategies for interconnecting processors: (1) 1-dim. linear arrays are suitable for matrix-vector multiplication and recurrence evaluation, (2) 2-dim. square arrays are suitable for pattern matching and relational data base operations, (3) trees are suitable for searching algorithms, and (4) shuffle-exchange networks for FFT. Lastly, H. T. Kung described a specific VLSI chip—one that performs online pattern matching of strings with wild card characters—produced at Carnegie-Mellon.

Gerry Sussman of the Massachusetts Institute of Technology then graphically illustrated the power of Mead and Conway VLSI design techniques by describing how they designed and implemented a LISP chip in just five weeks. The single-chip microcomputer called SCHEME-79 directly interprets a typed-point variant of SCHEME, a dialect of the language LISP. To support this interpreter the chip implements an automatic storage allocation system for heap-allocated data and a user interrupt facility. The processor is divided into two parts: the data path and a controller. The data path consists of a set of special purpose registers, with built-in operators, which are interconnected with a single 32 bit bus. The controller is a finite state machine which sequences through the microcode implementing the interpreter and garbage collector. As a performance estimate they computed (fib20) = 6765 with two different memory loadings, with a clock period of 1595 nanoseconds, and a memory of 32K LISP cells. The SCHEME-79 chip took about 1 minute, with a substantially empty memory, and about 3 minutes with a half-full memory, to execute the program.

The next speaker, Bart Locanthi of the California Institute of Technology, argued that there is increasing agreement that applicative or functional programming techniques are the answer to the problems of discovering and exploiting concurrencies in computations. This point of view appears also to extend to the conclusion that functional programming is the answer to the challenge presented by semiconductor technology. Bart then outlined his Homogeneous Machine, whose key notions are functional programming and a tree-structured organization. Where his work differs from others is in the way a functionally pure subset of LISP interacts with a tree architecture, and in what he calls a hierarchical cache used to produce a large multi-port memory. Each node in the tree comprises a processing element, a cache memory through which access to ancestral data is provided, and a local memory which can be written only by the processing element in the node. The processor is connected to its immediate ancestor and descendants in a tree structure. The sole purpose of the cache is to buffer read accesses to ancestral data.

John Darlington of Imperial College, London presented an abstract scheme for a multiprocessor implementation of applicative languages. The absence of side-effects or shared variables from applicative languages, besides bestowing many benefits for comprehensibility, also makes them ideal vehicles for parallel execution. He then went on to consider a scheme for evaluating such languages in a way that allows many simple processors to be

CCC-0010-4620/82/0025-0153 $02.50

used. The program, or graph, of the expression to be evaluated is represented as packets of work, each containing a packet identifier, an operator, and a list of input and output operands. At the abstract level the architecture consists of a pool of packets accessed by many processors, whose job is to select an active packet, apply one or more reductions to it, and return the resulting packets to the pool. The pool of packets represents the main implementation challenge. Two possible architectures are being investigated: first, a pool implemented as a ring on which packets circulate, and, second, a pool based on a more conventional RAM.

The next speaker, Gyula Mago of the University of North Carolina, described his reduction machine and discussed its efficiency of program execution. The computer architecture has the following properties which are particularly suited to VLSI. It has a cellular construction, i.e. the machine is obtained by interconnecting large numbers of a few kinds of chips in a regular pattern. The machine directly executes Backus' functional programming language FP, and automatically exploits the parallelism present in FP programs thereby, he claimed, achieving execution speeds far in excess of what is deemed merely adequate. The machine structure is a binary tree with two different kinds of cells employed: one kind are leaf cells (called L cells) and the other kind are non-leaf cells (called T cells). The FP expression, considered a linear string of symbols, is mapped onto the L array from left to right, one symbol per L cell, possibly with empty cells interspersed. Both kinds of cell are simple—a network containing a million L cells should be compared with a sequential computer having a million words of main memory. Mago said that a performance comparison of a 220 node reduction machine with a CDC 6600 for a Gaussian elimination with matrix sizes of 60, 130 and 220 indicated 7, 25 and 100 fold speed-ups.

Next, Klaus Berkling of Gesellschaft für Mathematik und Datenverarbeitung, Bonn described the GMD Reduction Machine built and in operation in his laboratory. This machine has created a medium on which an algebra of programs with functions as objects and functional forms as operators can be realized. The string reduction architecture has the following features: expressions (i.e. trees) are kept in stacks as sequences of constructors (i.e. nodes) and atoms (i.e. leaves) in preorder linear form. Atoms represent either abstract entities like numbers, booleans and strings of letters, or lambda variables, or name functions. The hardware consists of a set of seven stacks which are connected by two parallel buses to the processing unit. An expression kept in one (source) stack may be transferred to another (sink) stack by a recursive transfer process which first moves the constructor to a third (auxiliary) stack, then transfers the first subtree, then transfers the second subtree and finally moves the constructor from the auxiliary stack to the sink stack. During such a transfer if a primitive triplet apply-functional-argument combination is found it is replaced by its value. In terms of circuit count (about 1500 TTL chips) the GMD Reduction Machine compares with a von Neumann type mini-computer. Dr Berkling claimed, however, that its functional capabilities exceed those of the mini-computer by a large extent.

Arvind of the Massachusetts Institute of Technology argued that any machine comprising hundreds of processing elements must have a highly distributed and asynchronous control structure. As a solution, Arvind then described a system he is designing based on dataflow principles in which each processing element contains part of the program, and the processors communicate by sending information packets to each other. The machine is programmed in a high-level expression-oriented, single-assignment language called Id, and supports a novel way of interpreting dataflow known as the U-interpreter. The machine consists of N processing elements (PEs) and an $N \times N$ packet communication network. Each PE is essentially a complete computer with an instruction set, and a 16K word memory, etc., and is realizable in nMOS technology using one custom made chip and several standard chips. For the construction, he said that they plan to use several standard memory boards with 16K or 64K dynamic RAM chips and a commercially available ALU such as the Intel 8087. A dataflow computer with 64 PEs is currently being designed at MIT and is expected to be fabricated within two years.

Carl Hewitt of the Massachusetts Institute of Technology then described a multiprocessor computer called APIARY, suitable he claimed for VLSI implementation, which he is designing for use in an Artificial Intelligence environment. Hewitt said the main design problems relate to the required interconnection topology. Unless care is taken the design will produce long wires (connections) and hence unacceptable delays. These problems can be solved by using a Hypertorus system. The advantages of this folded torus system are that all wires are short, it occupies only a small space, and it is homogeneous, isotrophic and physically extensible. In APIARY, each computing element or worker component (actor) is constructed as follows: a work processor, IC memory (1Mb), a communications processor and a communications interface.

Next, John Hennessy of Stanford University discussed representational problems (of programs) in decentralized computer systems design. Representation of a program is the key problem in designing new languages and new computer architectures, since representations must be optimized in terms of certain key measures, e.g. size, execution speed and potential parallelism. These problems become much worse in a VLSI environment because of parallelism. Applicative or functional program representations offer many advantages for VLSI, for instance they are highly parallel, and since functions are dependent only on their inputs, the system lacks a central state and global environment. The Toy language is being developed at Stanford to investigate such approaches. Toy may be described as an intersection of: (1) the data types and syntax of PASCAL, (2) the loops and scope of the data flow languages VAL and Id, and (3) the operators of Backus' FP. There is no implementation or evaluation scheme predefined by the language. Two important evaluation strategies are being explored: data driven and demand driven strategies. The data driven scheme is based on the data flow notion that an operator executes as soon as its inputs are available, whereas the demand driven scheme follows reduction with an operator being executed when its result is needed. A major problem for the Toy language is handling data structures, since the concept of sharing is not present at the user level in a functional programming environment.

Continuing in the same vein, David Turner of the University of Kent discussed the implementation of applicative languages, in the context of his SASL programming language and its graph reduction execution model. This approach to compiling a very high level language, based on combinatory logic, was shown to have certain efficiency advantages over the traditional method of implementing applicative languages via Landin's SECD machine. There are basically two types of parallel reduction regime, namely cautious and rash. Cautious parallelism is a simple generalization of normal-order reduction, leftmost outermost plus the parallel evaluation of arguments of strict operators such as '+'. Rash parallelism evaluates reductions not lying on the normal order path, in the hope that it may require their results. Lastly, Turner outlined the advantages of using combinators: the combinator code is extremely simple, and it is known on theoretical grounds that any applicative language can be compiled to combinators. One of the main benefits of this second point is that we can work on developing the high level language without interfering with the design of the machine.

Sharon Sickel of Logical Paradox Inc. discussed programming for concurrency: how to uncover natural dependencies of the subparts without overspecifying the problem. A distinction was made between the set of descriptive languages (DL) and the set of effective languages (E). DL includes formal notations such as predicate calculus, algebraic formulas, set notation, etc., while E includes notations such as LISP, COBOL and ADA. However, a number of interesting languages, having the advantages of both groups, lie at the intersection of DL and E. These include Prolog, SASL and the dataflow languages. This allows a problem to be specified in a variety of notations (as long as the formal semantics are given) familiar to the requester and then transformed into an intersection language denoting an efficient computation. As an illustration, Dr. Sickel referred to Hoare's program FIND in its first order predicate calculus specification and transformed it into a Prolog program.

Daniel Friedman of Indiana University continued the theme of applicative programming, discussing the encapsulation and control of contending parallel processes within data structures. Programming problems whose solutions require such control include interrupt handlers, merging (of input streams), and the

airline reservation problem. In discussing primitives for such programming problems, Friedman used LISP's cons of a list. Discussion centred on the difference between McCarthy's traditional 'strict' cons whose arguments are call-by-value and his own 'suspending' cons and 'frons' whose arguments are called-by-need or called-by-delayed-value. Friedman demonstrated indeterminate programming using frons by presenting an indeterminate merge operation in an applicative style. The advantage of embedding such contention within a data structure is that the contention, itself, becomes an object which can be handled by the program at a level above the actions of the processes themselves. This means, he claimed, that indeterminate behaviour, never precisely specified by the programmer or by the input, may be shared in the same way that an argument to a procedure is shared by every use of the corresponding parameter, an ability which is of particular importance to applicative-style programming.

Next, Peter Henderson of the University of Newcastle upon Tyne gave a short presentation on the use of a purely functional language (LISPKIT) to describe geometric patterns. This was a preliminary report on some work being done at Newcastle by Henderson and McLauchlan on high level descriptions of the geometry of integrated circuits. Henderson showed how a simple register file could be described using just half a dozen simple functions in the language.

Changing the tone, David Boyd of the SRC Rutherford Laboratory described the UK Science Research Council's support for (VLSI) integrated circuit production. The Rutherford Laboratory is the centre of the SRC Interactive Computing Facility (ICF) network of PRIME, GEC and DEC computers which provide country-wide interactive facilities for a number of application areas including microelectronics design work. At present integrated circuit design is based on the GAELIC suite of programs, commercially marketed by Compeda Ltd. It supports manual mask layout using a text language, interactive CRT or digitizer automatic layout of general cells, design rule checking, logic simulation, and pattern generator output in particular to the mask-making facility at Rutherford. The Electron Beam Lithography Facility (EBLF) at Rutherford is based on a Cambridge Instrument's EBLF-2 currently supplying a service down to 2 micron minimum features on up to 4 inch chrome-on-glass masks. Fabrication facilities are available at Edinburgh and Southampton Universities for *n*-channel silicon gate processing. The SRC is also supporting a number of MSc. courses in Microelectronics beginning in October 1980 with opportunities for design, fabrication and testing of integrated circuits.

The final presentation was by Philip Treleaven (the author) of the University of Newcastle who described a decentralized computer architecture and computational model. The architecture is based on a single chip building block—a computing element—which is plugged together with its duplicates (*cf.* LEGO blocks) as a vector to form a parallel computer. Each computing element contains integral capabilities for memory, processing and communication. Each memory holds a contiguous part of the program structure which is encoded as nested delimited strings using the characters '(', "0", "1" and ')'. A processor executes work in its memory and also services accesses to its part of the program structure. The communication unit handles memory accesses in non-adjacent elements, by establishing a 'route' between a reference to an item of information and the actual item. Such a route is established by an address, which in the machine is a sequence of selectors like a telephone number. Lastly, the computational model represents a synthesis of the concepts underlying data flow, control flow and reduction models, and hence is general-purpose by supporting these models.

One of the doyens of computer architecture, Bob Barton of the Burroughs Corporation was also a participant at the workshop and made a number of stimulating informal presentations. Unfortunately he resisted considerable pressure to give a formal presentation of his VLSI-related work.

Intermixed with the invited presentations were a number of panel discussions covering topics ranging from design of integrated circuits to very high level programming. For each discussion session four to five people were invited to make short position statements after which the topic was thrown open to all participants.

The first discussion addressed the problem of design environments and tools. The panel consisted of David Kinniment (University of Newcastle upon Tyne), Irene Buchanan (Edinburgh University), Steve Hollock (Plessey Research Limited) and Doug Lewin (Brunel University). David Kinniment, setting the theme for the other panelists, urged the adoption of a hierarchical and structural method to overcome problems such as the increasing length of an IC design cycle and management of the layout and other problems which are dominating the chip area. Design and layout of systems with a million transistors and above is a very complex task and cannot be tackled with present day methods which essentially operate at the level of geometrical manipulation using graphics. Continuing the call for structured design in VLSI, Irene Buchanan stated that the principles are analogous to those of structured programming. This is not surprising since both approaches are concerned with the problem of controlling complexity. The design hierarchy of VLSI separates into two different types of cell. Cells at the lowest level contain primitive components and are termed 'leaf' cells. Cells at levels above, contain instances of these lower level cells and are called 'composite' cells. In Steve Hollock's experience customers were more interested in performance than complexity, requiring more customization (i.e. ultra-high speed logic, mixtures of high and low speed, analogue and digital). However, sufficient repetition has led Plessey to develop two major design methodologies: a high speed ECL ULA family and Microcell for low power C-MOS. Doug Lewin highlighted a number of points concerning design. These include: VLSI is only one area in designing digital systems which contain both hardware and software; there are few viable tools for specification testing and synthesis of digital systems; due to complexity, design algorithms with exponentially increasing time are no longer viable; and there is a need to consider the total system including analogue and transducer components.

Attacking the panel, Peter Henderson (University of Newcastle) said that Sussman seems to have been able to design a fairly complex system making minimal use of graphical facilities. This appeared to contrast with the emphasis of some other speakers, who seem to think that graphical tools are more important. Carlo Sequin leapt to the defence of graphic tools, giving as an example that digitizing is the simplest way to do random corrections. Gerry Sussman commented that anything we wanted to tell a program by showing it, we also want to tell by saying it. This is because we may eventually build a second (AI) program which is intelligent enough to instruct the first.

The second discussion session on VLSI building blocks took as its theme: will a 'universal' processor-memory chip solve all our building block problems, or do special-purpose and customizable PLA-type chips have a contribution? The panelists were David Aspinall (UMIST), Mike Rogers (Bristol University), Iann Barron (Inmos Limited) and Brian Warboys (International Computers Limited). Mike Rogers commented that listening to many of the talks about architecture and programming, he was wondering about the relevance of all this to VLSI. Maybe we should be thinking more about the design methodology than the building blocks. Iann Barron then described his ideal building block—a 'Transputer'. Briefly a transputer has a processor, memory and communications on a chip. It is also important that the communication is uniform across the network since people will want to build quite simple systems from this component. Brian Warboys argued against introducing too many dichotomies in VLSI. There is the hardware/software, the processor/memory, and now the general-purpose/customized chip dichotomy. He hoped that we could keep the transparency in large systems we seem to be discussing, in respect of small systems. Small may be beautiful but large has a hell of a lot of inertia. It seems absurd to be fixing now on standard components at a time when this new technology is just coming in. Arvind then asked why do we think a processor/memory pair is a good component? Iann Barron replied that one of the biggest design constraints is the number of pins. You want to keep that number low. Additionally, speed of communication between processor and memory is a major factor. A chip with both processor and memory can be used essentially as a memory chip if you want it that way. The processor/memory chip may not solve all problems (e.g. pattern recognition) but it does help solve some important problems. Supporting Barron, Carlo Sequin added, the way things have

happened is to put the high bandwidth path on the chip. The next path in line for inclusion in the chip is the processor-memory connection. Now you have the problem of getting enough memory onto the chip. Klaus Berkling questioned whether we were merely perpetuating the von Neumann machine by this processor/memory structure? Countering, Gerry Sussman said a network of (say 1000) von Neumann machines is not a von Neumann architecture from the point of view of the programmer who has to program them. David Kinniment interjected that the whole concept of building blocks assumes that we are going to build a very big system. But the major market is for small customized systems on a single chip. Brian Randell added that surely, we progress by accepting a set of standard components and living with them. A great strength of von Neumann has been that, no matter how bad a particular machine, we have been able to program round it.

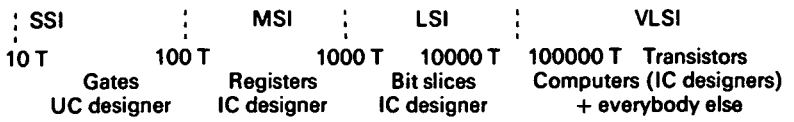
The next discussion on the architecture of ultra concurrent machines, had as panelists Philip Treleaven (University of Newcastle), Robert Milne (Inmos Limited), Bob Barton (Burroughs Corporation) and Wayne Wilner (Xerox Corporation). Philip Treleaven pointed out that whatever our VLSI building blocks if large numbers are to be employed in computer systems then there must be a system-wide architecture and computational model which they all obey. In identifying such a model, he said there seemed to be two basic mechanisms. First, a control mechanism defining how one instruction causes the execution of another and consisting of either 'by availability' of data, or 'by need' for data. Second, a data mechanism defining the way a particular argument is used by a group of instructions and consisting of an argument either being passed directly 'by value' between instructions or being passed indirectly 'by reference' to a shared memory cell. Most computational models regards these mechanisms as two pairs of alternates, for a VLSI architecture to be general-purpose it needs to support both pairs. Taking a different theme, Robert Milne asked what the simplest microcode and protocols should be to enable us to build (software/hardware) systems easily. For describing a decentralized system we want a system describing language which specifies what the allowed sequences of messages between the components are and how these messages are to be understood. In principle this language could be applicative. However, the discussion of applicative architectures should not concern their adequacy in principle but their adequacy in practice. The three drawbacks are: (1) its reliance on kernel programs such as storage managers, (2) its inability to hide information in processes properly, and (3) its omission of a parallel operator for binding messages from source processes to destination processes. At the lowest level of a system we may move away from purely applicative programming. Yet once we do this, things proliferate: we start to have functions, processes, parameters, variables, messages, and indeterminacy (concurrency), or indeterminacy (choice) . . . Not all these things are essential; which ones to combine, while still avoiding the drawbacks of applicative programming, Milne presented as a challenge.

Bob Barton made a number of interesting comments under the title of 'Various'. Barton said that yesterday's problems have been updated. We used to have arguments between the logic designers and the programmers about designing machines. Yesterday's logic designers have been replaced by programmers and the programmers by mathematical types who know a lot about logic but are most at home thinking about applicative languages. It's the same old arguments, just new protagonists. He said he appreciated Turner's approach. Turner has found a practical application for a nice little bit of esoteric mathematics: combinatory logic. It's a completely fresh approach to designing a machine. Barton then asked the following question. What would justify a computer manufacturer in investing the millions of dollars needed to design and manufacture new machinery? In answer, he said that functional programming is the first thing that had interested him in programming for many years. He thought it is really most important. We should be thinking also about using the capacity for logic, allowed by VLSI, to gain other advantages than just performance. We should use that capacity to obtain reliability and fault tolerance for example. Our dependence on networks establishes a very real target for terrorists, as long as fault tolerant issues in network components remains unaddressed. Wayne Wilner said that rather than proposing a specific architecture he would like to present a dozen issues which ultra-concurrent machines must resolve in order to be successful. The first five are areas of opportunity for ultra-concurrent machines: input/output, software system structure, task decomposition, source-level control over the machinery, and interconnection structure. The last seven are potential obstacles: fault tolerance, address mapping mechanisms, flexible hardware/software boundaries, efficient use of common resources, multiple instruction sets, interprocessor interference, and deadlock avoidance. After giving the dozen issues Wayne expanded on each topic.

The discussion on very high level programming languages took as its theme how will we program future concurrent machines, or can we design the languages first and the machines second? The panelists were Peter Henderson (University of Newcastle), Sharon Sickel (Logical Paradox Inc.), Joe Stoy (University of Oxford), David Park (University of Warwick) and David Turner (University of Kent). All of the panelists strongly favoured applicative languages, and the general theme of the presentations may be gained from the following contribution by David Turner. On the software side we have a crisis which can be resolved by moving to a much higher level of programming language, the kind of language he was calling earlier a 'denotational language'. These languages are characterized by a group of properties collectively known as 'referential transparency'. On the hardware side we can see that to realize fully the potentialities of VLSI we have to move to a highly parallel architecture. Parallelism on the necessary scale cannot conceivably be under the conscious control of the programmer—it must arise from an inherent asynchronousness in the language he is using. The remarkable point is this—that the conditions for a language to be asynchronous in the right way turn out to be *precisely the same conditions* as for the language to be referentially transparent.

A number of people disagreed strongly with the panelists. Carlo Sequin's remarks illustrates their view. Applicative language are (1) a bad match to the real world, and (2) a bad match to VLSI technology. The 'real' world has 'things' in it. These things have 'state' which does not change unless there is 'interaction'. Interaction occurs in time and takes time. Von Neumann languages have a messy mixture of state and interaction. Applicative languages leave out state and have implicit and immediate interactions. In a more suitable abstraction things will have implicit state (e.g. processes), all interactions will be explicit and interactions would be needed to know or change state. Applicative languages are a bad match to VLSI technology. Current IC technology produces memory at about 10 times more dense than logic. Note: (1) how can you make effective use of this fact if you throw out states? (2) computation of a function takes much more energy than a storage of a state; (3) state machines and von Neumann processors are existing building blocks suitable for the representation of things (e.g. processes); and (4) extraction of parallelism becomes more natural when tied to partitioning into things. Following Carlo Sequin's remarks, with support from Gerry Sussman and others, a heated discussion developed. The participants were divided into two approximately equal factions on the question of the general-purpose nature of applicative languages. One group championed applicative languages and the other favoured object-oriented languages such as Xerox PARC's Smalltalk.

In the final discussion session, asking the question: are we really ready for VLSI?, the panel were Mike Rogers (Bristol University), Carlo Sequin (University of California, Berkeley), David Park (University of Warwick) and Brian Warboys (International Computers Limited). Mike Rogers opened. VLSI imposes two major constraints and at the same time provides two major opportunities for the system designer. The constraints are: (1) a small number of simple cells repeated many times, and (2) regular interconnections. The opportunities are: (1) high degree of concurrency, and (2) creation of innovation architectures, in particular hierarchical structures. Except for repetitive designs, such as memories, the problems of complexity are increasing rapidly and these are evidenced by the lengthening design time-scales. This situation closely mirrors the history of systems software except that we now have the advantage of hindsight, and can learn from the lessons of the past. Ideas of hierarchical design, with a structured, modular approach to the problem have been adopted generally successfully in this area, and point the way ahead for VLSI designers. However, these require appropriate software



tools and at present we do not have them. So in this sense we are not yet ready for VLSI. To take advantage of VLSI we must also be able to deal with all the problems arising from concurrency and here the major difficulty arises in applying in a systematic way the advances in computer science which have been made over the past few years in our understanding of concurrent processes. In this area we are partially ready for VLSI. Finally, one of the toughest problems to crack, and one in which very little progress has been made is the whole subject of testing and verification. In this area we are certainly not ready for VLSI, and in the long run it is likely that this will have the biggest single effect in limiting the size/complexity of designs.

Carlo Sequin felt we were definitely not ready for VLSI. There are various ways to characterize this type of technology (see diagram).

Therefore VLSI is a *social phenomenon*. There are several fallacies about VLSI. Everybody wants to design their own languages, but now they can design their own machines too. VLSI is cheap, so if something is too complex and expensive to implement in software, do it in hardware instead. One need not even take care over the design of the hardware because fast turn-around times allow faults to be located quickly in the actual chip. The above points indicate why we are not ready for VLSI, in brief we have the wrong attitude, insufficient design and testing tools, and not enough designers to span the range of knowledge required from system down to transistors.

David Park also felt that we were not ready for VLSI. VLSI technology is with us now, but we are not intellectually ready for it, which is going to cause problems. A possible step in the right direction is the development of functional languages since they allow programs to be thought about in a mathematical way. An example of this is a program which describes the transformation that can be applied to a context free grammar. Brian Warboys stated that the breakthrough will occur when, for example, relational data bases meet reduction techniques. VLSI tells us that the building blocks are the system. The important thing is to introduce disciplines and integrated systems. Then Steve Hollock argued that designing VLSI chips is difficult and expensive. Mead and Conway techniques are wasteful and do not use state of the art circuits. Manufacturers will not allow researchers access to an up-to-date production line, the best that can be hoped for is one that is two years old. A number of people disagreed. Arvind said his machine is directed to the solution of a real problem and will be built with the help of industry. Lastly, Gerry Sussman pointed out that MIT is using an up-to-date line.

In summary, I hope that this report gives the flavour of what, most participants agreed, was a very extensive and exhilarating week of discussions held in a most enjoyable environment. The workshop proceedings are published by the Computing Laboratory as Technical Report number 156. Individual copies can be obtained by sending £4 ($10) per copy to: VLSI Workshop Proceedings, Computing Laboratory, University of Newcastle upon Tyne, Newcastle upon Tyne, NE1 7RU, England. Cheques should be made payable to 'University of Newcastle upon Tyne'.

PHILIP C. TRELEAVEN
University of Newcastle upon Tyne,
Claremont Tower,
Claremont Road,
Newcastle upon Tyne NE1 7RU, UK