SINGLE - CHIP COMPUTERS,

THE NEW V L S I BUILDING BLOCKS

Carlo H. Séquin

Computer Science Division
Electrical Engineering and Computer Sciences
University of California
Berkeley CA 94720

ABSTRACT

Current trends in the design of general purpose VLSI chips are analyzed to explore what a truly modular, general-purpose component for digital computing systems might look like in the mid 1980's. It is concluded that such a component would be a complete single-chip computer, in which the hardware for effective interprocessor communication has been designed with the architecture of the overall multiprocessor system in mind. Computation and communication are handled by separate processors in such a manner, that both can be performed simultaneously with full efficiency.

This paper then describes relevant features of **X-TREE**, a research project which addresses the question how the power of VLSI of the next decade can best be used to build general purpose computing systems of arbitrary size. In **X-TREE**, a general VLSI component realizable in the mid 1980's is defined, and its interconnection into a hierarchical tree-structured network is studied. The overall architecture, communications issues and the blockdiagram of the modular component used are discussed.

1. INTRODUCTION

By now it seems to be a well accepted truth that computing demand will always stay ahead of the available computing power. No matter how much computational power can be realized on a single VLSI circuit, there will always be customers that want to have computing power that substantially exceeds the capabilities of a single chip. (Even though the fraction of those customers will decrease dramatically compared to the number of single-chip users.)

If a system is to be extended over several integrated circuits, the question arises how it should be partitioned onto the various chips. This question is crucial since the connections between chips represent an inherent communications bottleneck. The number of interconnection points on a VLSI chip is limited and increases at a much slower rate than the number of gates in the circuit. Furthermore the bandwidth through those pins is limited; with current packaging technology, the physical paths leading from one chip to the next are one or two orders of magnitude longer than on-chip interconnections, and possess correspondingly larger parasitic capacitances.

It is thus important to find the right functional blocks to be packaged onto individual chips. We can state immediately that the partitioning should not cut through the high-bandwidth path between main memory and the actual processing circuits. Thus memory and processing elements should stay tightly coupled on one and the same chip. VLSI technology in the mid 1980's, will make it possible to pack a sophisticated processor and a substantial amount of memory on a single integrated circuit.

2. GENERAL PURPOSE VLSI COMPONENTS

No matter how fast, convenient and sophisticated VLSI design tools will become, standard off-the-shelf parts will always be in strong demand for people who want to put together experimental systems or who want to bring a product to market quickly. They cannot afford to start from scratch and to determine in all details, how the optimum chip for their purpose could be constructed. For small and medium volume production, it is far more cost effective to pick a component from a limited selection of existing parts. If these parts have already been debugged and if applications manuals are available, the user need not worry about all the critical details of internal timing and interconnections. It normally pays to choose a somewhat overdesigned component, using more circuitry, power or silicon area than absolutely necessary, thus guaranteeing that the requirements of the application will be safely met.

The question thus arises what type of circuits can be placed on the shelf and will prove useful in a large number of applications. Within the frame work of digital computing systems, what kind of chip will take on the former role of NAND gates, flop-flops or registers ? Simple extrapolation of existing trends may not be the right answer for general purpose components with a hundredfold greater functional capability.

Memory chips with four times more storage capacity appear on the market about every three years now. Certainly, a 64k-word by 1-bit chip is a nice part to have, but will a 1M-word by 1-bit chip be equally attractive ? - Or would most customers rather use a 64k-word by 16-bit part ? Also, from the point of view of optimal systems partitioning, discussed in the previous section, a simple memory chip is a very poor component. It has virtually no internal interactions, and every time it is used, information has to go in and out through the package pins. These shortcomings should be remedied by adding processing power onto the memory chip itself.

Looking at the field of random logic, the current trends are more obscure. There is a certain evolution from the NAND gate to more complicated Boolean functions such as multi-bit logic functions, parity checkers, arithmetic functions or priority encoders. However there is a limit to the size of useful logic functions that are commonly used. The trend is to add input or output registers to the logic functions to permit easy assem-

bly of synchronous machines. Here again, we find a mixture of processing and storage elements.

The most complicated parts available today are obviously the single-chip microcomputers with a certain amount of on-chip memory. But can these chips really be regarded as general purpose, modular building blocks in the sense of a NAND gate ? While one can prove that all computing systems could be built from nothing but NAND gates, it is rather doubtful that we can interconnect many, say, INTEL 8048 and produce a computing system of superior power ! What is missing is an answer to the question how several such components should be interconnected and how they should interact. While successful TTL parts have been designed so that many of them can be combined to form a system with more capabilities than the sum of its parts, the interconnection issue was definitely not a primary concern in the construction of the first-single chip microcomputers. In the design of a truly modular VLSI component for large computing systems, the communications issue can no longer be neglected.

3. INTER-CHIP COMMUNICATION

Standardizing the communication between computers is a considerably more complicated issue than standardizing interconnections between Boolean logic chips. In addition to voltage levels and load considerations, which apply to logic chips, interconnection of computers involves additional issues such as link allocation, timing, message formats, addressing and routing. All these issues have to be addressed in the design of a truly modular general purpose VLSI component. Hardware for effective support of these features has to be incorporated on the chip. The block diagram of a potential general purpose component, consisting of at least a processor, memory, a communication switch and its controller, is shown in Fig.1 using PMS notation [Bell & Newell 1971]. The case for such "Computer Modules" with proper consideration of the communications issues was first made by Bell *et al.* [1973].

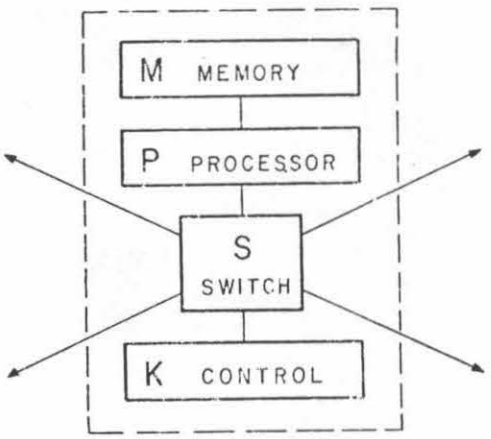

Figure 1. X-NODE, a modular VLSI building block of the mid 1980's, shown in PMS notation.

4. PROJECT X-TREE

In 1977 a research project to investigate these questions was started at the University of California at Berkeley. One specific issue was to define a modular component from which general purpose computing systems of arbitrary size could be built. Since the most advantageous way to interconnect these components turned out to be a tightly coupled, hierarchical, tree-structured network, the project was named X-TREE [Despain & Patterson 1978a]. And the projected single component, from which this computing system would be assembled, is known as X-NODE. The project is still in its early stages of research. First some of the communications issues have been addressed in a top-down manner [Séquin *et al.* 1978]. Discussions of the design of the architecture of X-NODE have recently been started [Patterson *et al.* 1979]. The anticipated timeframe for a potential realization of X-NODE is the mid-1980's. It is assumed that at that time VLSI chips with about 100'000 gates and half a million bits of storage will readily be feasible. Key research issues are the design of a truly modular VLSI component and the organization of many such components into a general purpose computing system.

5. INTERCONNECTION TOPOLOGIES

In an evaluation of different multiprocessor interconnection topologies [Despain & Patterson 1978b], it turned out that most schemes have one of three disadvantages.

1) The interconnecting link constitutes a serious communication bottleneck, such as a common bus which has to be shared by all attached processors.
2) The switching hardware becomes unreasonably expensive for a large number of processors, as in fully interconnected networks or in a full crossbar arrangement.
3) The interconnection scheme is not truly modular, because the requirements for the individual components change as the system grows. This is the case in a hypercube network with the topology of a cube in n dimensions, where the number of ports has to increase with the logarithm of the number of nodes.

Networks that are truly modular and incrementally expansible comprise lattice structures in n dimensions and tree structures. Among those, trees have the further advantage that the average distance between nodes grows only logarithmically with the number of nodes [Despain & Patterson 1978a,b].

The primary contender for the interconnection scheme in X-TREE is therefore a binary tree enhanced with additional links to form a half-ring or full-ring tree (Fig.2). These additional links further shorten the average path length, they distribute message traffic more evenly throughout the tree, and they provide the potential for fault tolerant communication with respect to the removal of a few nodes or links. Various placements for these extra links have been investigated, but most yield only insignificant improvement on the message traffic density and typically require significantly more complicated routing algorithms than the half-ring or full-ring structures [Despain & Patterson 1978b].

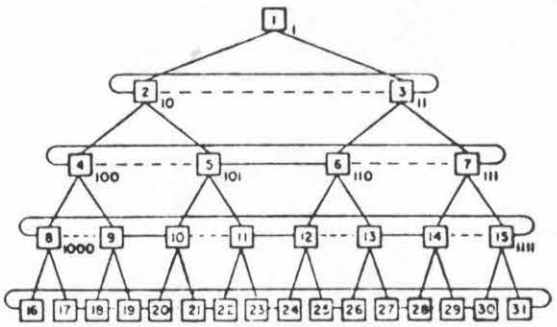

Figure 2. Binary tree with full-ring connections. When the dashed branches are omitted, a half-ring tree is obtained. Notice that the children of node n have node addresses 2n and 2n+1 respectively.

6. ADDRESSING AND ROUTING

X-TREE is designed to be a truly modular, incrementally expansible system. To allow the system to grow to arbitrary size, neither the number of nodes nor the address space should be limited by artificial restrictions. Thus unbounded, variable length addresses are employed throughout [Séquin *et al.* 1978].

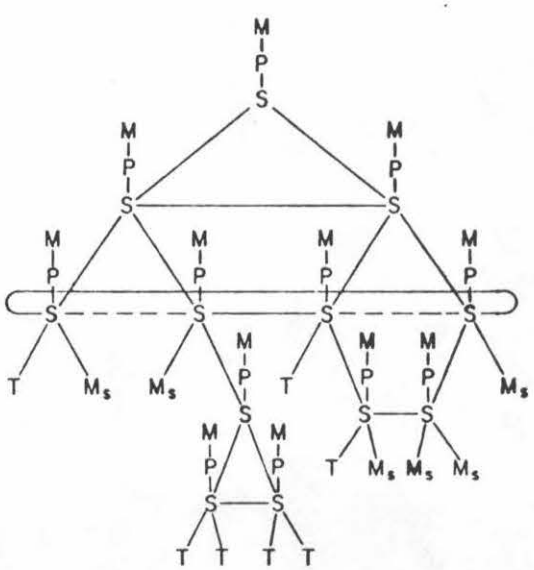

Figure 3. An X-TREE example in PMS notation.

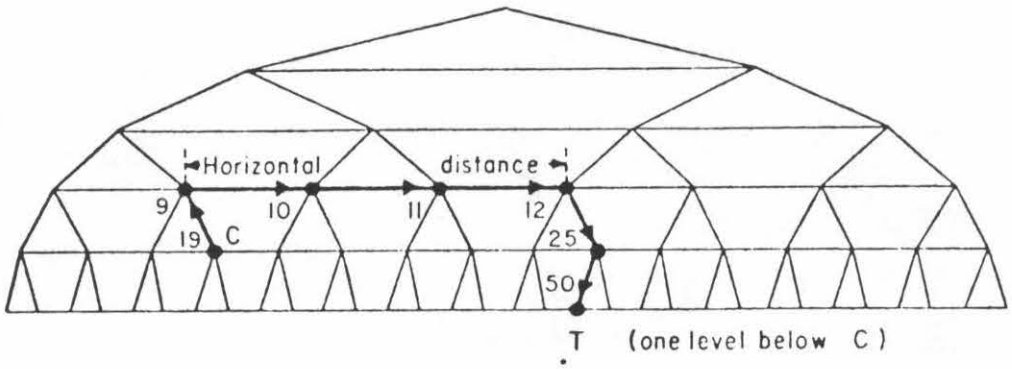

Figure 4. Shortest path in a full-ring tree from current position, C, on node 19 to target, T, on node 50.

All communication throughout the tree is in the form of messages. To enable effective routing of messages, the complete address is subdivided into a node address part and a second part identifying a particular memory location if that node has any memory attached that belongs to the global address space. In X-TREE secondary memory, as well as all input/output, is restricted to the frontier (i.e. the leaves) of the tree. This is to minimize the number of ports per node, and thus to alleviate the restrictions originating from the limited number of pins. A PMS representation of an example of a small X-TREE is shown in Fig.3.

The addressing scheme is designed so that messages can be effectively routed from node to node simply based on local decisions, involving only the current node address and the destination address. In a binary tree this can be achieved in a very simple manner. The root node is assigned node address "1". The node address of a left child is formed by appending a "0", and that of a right child by appending a "1". With that scheme any node can readily be found by starting at the root and using the sequence of bits in the node address to make the routing decision at each node. To go from an arbitrary node to another node, one has to move up in the tree to the common ancestor of the two nodes, i.e. to the node where the address matches all leading bits in the target address. From there one moves down to the destination.

In half-ring and full-ring binary trees the routing algorithm is only marginally more complicated. The horizontal links permit a message to take a shortcut before the common ancestor has been reached. It turns out that in a full-ring tree the optimal level to take the horizontal links is the one where ascending and descending path are separated by less than five link units (Fig.4). Again the routing decision can be based entirely on a comparison of the local node address and the target address.

Choice of directions as a function of relative target position							
choice		far left	near left	left line	right line	near right	far right
above	1	up	up	up	up	up	up
	2	left	left	left	right	right	right
	3	right	right	right	left	left	left
level	1	up	left	proc.	proc.	right	up
	2	left	up	back	back	up	right
	3	right	ldown	back	back	rdown	left
below	1	up	left	ldown	rdown	right	up
	2	left	up	rdown	ldown	up	right
	3	ldown	ldown	left	right	rdown	rdown

Table 1. Decision table for full-ring tree routing algorithm

Some fault tolerance can be achieved by listing secondary and ternary choices for the routing decision, which are taken when

the primary choice cannot be followed. Such a decision table for the case of the full-ring tree is shown in table 1. The proper field is selected from a comparison of the horizontal and vertical distances between current position and target node. Within each field, first, second and third choice for the routing out of the current position are listed.

Of course there are always cases where this fault tolerant routing algorithm cannot be successful - for instance when the target node itself is missing. Special means to safeguard against cluttering the tree with worthless messages are thus required. One possible solution is to accompany each message header with a byte that counts the number of detours experienced, i.e. how often the message could not be routed in the primary direction. When this count reaches a limit, preset by the operating system, a higher level recovery mechanism is invoked. For instance, the message could then be purged and a notice of this fact could be returned to the originator.

7. SYSTEM EXPANSIBILITY

In **X-TREE** all input or output and secondary memory is at the frontiers of the tree. Thus, every time a node is added to the tree, a terminal or a storage device has to move and thereby changes its node address. A mechanism is necessary to route messages automatically to that node even though the message header may still carry an outdated node address. With the following conventions this can be achieved. Messages for leaf nodes are identified as such. Each node knows whether it is a leaf node or not. Messages destined for leaf nodes are routed downwards from their specified target node address along the chain of left children until they reach an actual leaf node. This scheme works, if during the expansion of the tree existing leaf attachments are moved to the left child position, of the newly attached node, and the right child position is used to attach new equipment (Fig.5).

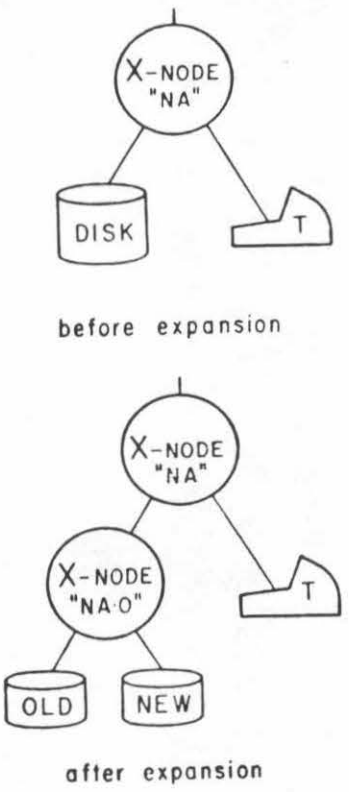

Figure 5. Principle of incremental expansion of **X-TREE**, which permits messages to find the proper leaf node.

8. MESSAGES

If **X-TREE** lives up to its potential of substantial parallel computing, many messages will travel simultaneously through the tree, and many messages may want to use the same link between two nodes. In order not to stall one message until another message has been transmitted completely, messages are time multiplexed on the links on a demand bases. For this purpose each message header carries a unique identifier called a "slot address", so that the various parts belonging to different messages can be sorted out again at the receiving node. In each node all incoming messages are assigned such a "slot address", which is valid within that node and on the subsequent link. A specific slot address precedes any part of a message transmitted over a particular link.

Bytes	Explanation
SA / new	Slot address (SA) of type "new"
Target NA	together with target node address (NA), which may extend over several bytes,
NA cont.	sets up a new message channel through the tree.
Data	Beginning of first submessage.
Data	
:	Here, one or several other messages
:	may be interspersed on the same link.
SA / old	Slot address of type "old" identifies continuation of a previous message.
Data	
Data	Data may consist of one or more submessages which may be individually checked
Data	and acknowledged to the originator.
:	Here, one or several other messages
:	may be interspersed on the same link.
SA / old	Previous message continues.
Data	End of last submessage in this channel,
Data	may include crc check remainder and end of transmission character.
SA / end	Tear-down slot address removes message channel.

Table 2. Message format.

For the purpose of creating and removing message paths, there are different types of slot addresses. With the header of a new message, which carries the target address, a slot address of type "new" is transmitted, to establish a path throughout the tree to the proper destination. All subsequent bytes are considered to be part of the message content and are thus not involved in the routing process. If a message has to be interrupted, then subsequent parts of that message will be identified with the same slot address as before, but of type "old" to indicate that this is a continuation of a previous message. Active transmission is terminated by the originator by sending a slot address of type "end", which removes the established message channel. These conventions result in the message format shown in Table 2.

9. X-NODE ARCHITECTURE

At a first glance, each X-NODE is simply a computer that communicates with 4 or 5 nearest neighbors. However because of the bandwidth requirements discussed above, normal microprocessor input/output techniques are inadequate. It is important that the processor is not involved in the menial task of routing messages through X-TREE. Computation must occur in parallel with communication. Thus each X-NODE contains a self controlled switching network with its own I/O buffers and controllers. In the simplest version of X-NODE, the actual processor is attached to this network in the same manner as the links to the nearest neighbors (Fig.6). This permits an easy separation of the development of the communications hardware and of an advanced X-NODE computer.

10. SWITCHING HARDWARE

The heart of this switching network is a time multiplexed bus. Since this bus is anticipated to be completely contained within one chip, its associated parasitic capacitances will be rather low, and the resulting bandwidth for a given amount of drive power will thus be about an order of magnitude higher than that through the package pins associated with the input / output ports. This bus can thus conveniently serve the attached six to eight ports at their full capacity (Fig.6).

Each port consists of a set of input and output buffers and the necessary finite state machines to control them. Arriving slot addresses, which precede every separate part of a message and are identified by a special tag, switch the input multiplexer to the proper fifo buffer for that particular message (Fig.7). The internal communications bus consists of a data bus and an address bus carrying port numbers as well as slot addresses. The combined bus is allocated in a fixed round-robin manner to all attached input ports, which in turn can transmit one byte to a particular output port or to the routing controller.

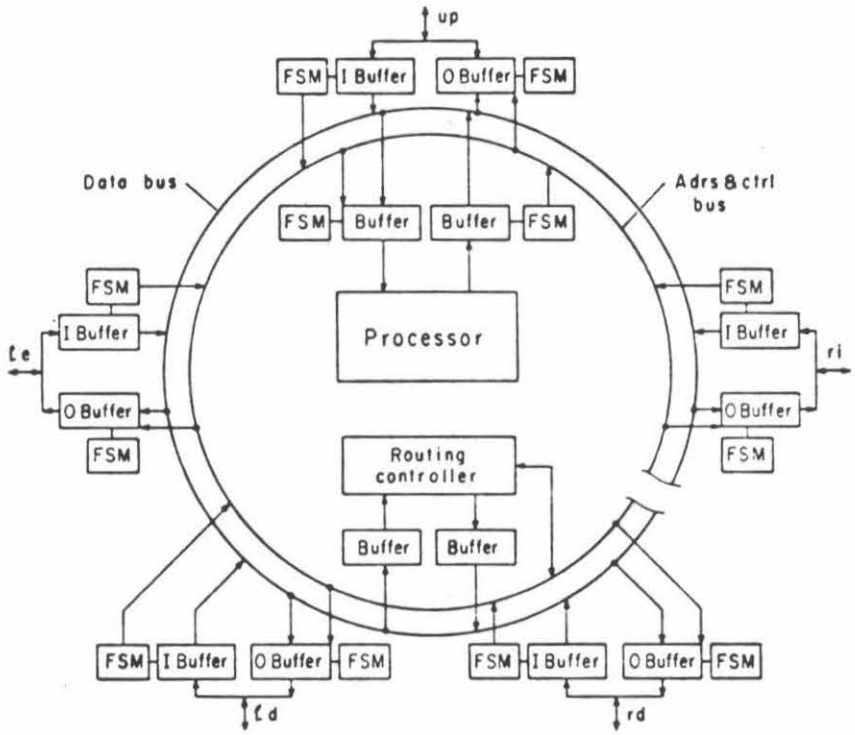

Figure 6. Block diagram of X-NODE, showing the X-NODE processor and the switching network consisting of intra-node bus, routing controller and input/output buffers to the communication links.

Slot addresses of type "new" cause the message header to be sent to the routing controller. There the proper output port is determined from the routing algorithm and a suitable slot address will be assigned to this newly to be established message channel. This information is returned to the port of entry where it is written into a look-up table (Fig.7). From then on, the intra-node communication for this channel can be handled directly by the input port.

Each output port monitors the intra-node bus. If it finds its own port number on the address bus, it will pick up a slot address and the corresponding data byte and enter the later in the proper fifo output buffer (Fig.8). Simultaneously, the finite state machine at the output end selects a channel with valid data for transmission over the link. The data bytes are preceded by the transmission of the corresponding slot address.

11. X-NODE MEMORY

Memory contained within each X-NODE is not part of the global address space. It acts only as a cache to the secondary memory contained entirely at the leaves of X-TREE. To alleviate the paging overhead, this memory should be as large as possible and thus the densest storage technique should be employed. For the mid 1980's it is anticipated that about 64k bytes could be implemented together with the X-NODE processor on the same chip if dynamic RAM or charge coupled devices are used. The contents of this local main memory can be data, program or microcode, and they will thus be used by different functional blocks. Unfortunately, typical high-density memory cannot easily be implemented as a true multiport memory, and thus it is not possible to extract three words from different locations simultaneously. Severe contentions for the memory bus can thus be expected.

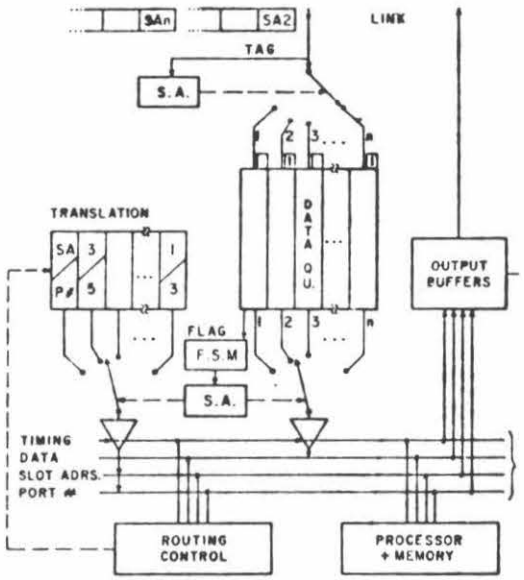

Figure 7. Blockdiagram of input port.

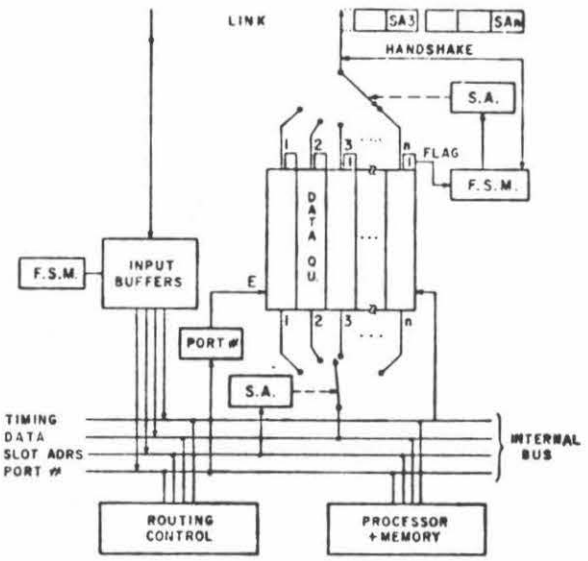

Figure 8. Blockdiagram of output port.

To alleviate this problem, we have decided to build an on-chip memory hierarchy consisting of a high-density RAM and three high-speed static caches dedicated to data, instructions and microcode, and therefore closely tied in with the ALU, the instruction decoder and the microcontroller, respectively. Contentions for the common data path between the main memory and the three caches can be minimized by proper choice of the cache parameters. This approach combines the storage density advantage of dynamic RAM with the higher speed of static caches and yields a lot of flexibility in the allocation of local main memory space to data, programs or microcode.

Swapping of microcode occurs through the same mechanisms as paging of programs or data from secondary memory at the frontier of X-TREE. No special mechanism has to be invoked to change the instruction set of one of the X-NODES, and the latter can thus be tailored dynamically to best solve the current computational problem.

12. OTHER ARCHITECTURAL FEATURES

There are other architectural features which indirectly relate to the communication between processors. Issues of timing, synchronization, data sharing and protection are all crucial in multiprocessor, multi-user systems. The architecture of future chips should thus lend ample support to those features. Specifically, the concepts of processes [Dijkstra 68, Wirth 1971, Brinch Hansen 1972, 1975] and modules [Wirth 1977] are important for a clean and structured approach to parallel computing. In X-TREE we plan to give strong support to the mechanisms that create, terminate or move processes throughout the system [Patterson *et al.* 1979]. A vital part in this context is a mechanism that permits fast process switching without the explicit need to save many special registers. Again the cache based architecture provides a fast, automatic and transparent mechanism for the replacement of the active parts of the memory contents when a process environment is changed. A

small operating systems kernel based on the language Modula [Wirth 1977] is currently being developed and will be used in future extensive studies of the operating issues.

It goes almost without saying, that the customers of the next decade will expect strong support for high level language constructs, data structures, bounds checking and any thing that may help to slow down the increase of the gap between software and hardware costs.

13. SUMMARY

In summary, there will always be a demand for off-the-shelf, general-purpose components, which can be used as truly modular building blocks for the construction of computing systems of any size. One such component has been identified and one way to organize that component into an incrementally expansible computing system has been outlined. Two crucial issues in the design of such a system are interprocessor communication and the operating system. Both issues must be addressed at an early stage of the design so that the necessary hardware support can be provided.

We felt that the issue of interprocessor communication is even more important at an early stage of a VLSI multiprocessor project. The emergence of truly modular, general purpose VLSI components may actually depend on the design of a standard interprocessor communication protocoll and a realization of the required switching hardware on VLSI chips.

14. ACKNOWLEDGMENTS

The author would like to point out that project X-TREE is a "tightly coupled" team effort of our Architecture Group in the Computer Science Division at Berkeley, including Profs. Despain and Patterson, and several graduate students, and that it is almost impossible to determine the specific contributions of each member of the team. Particular thanks go to Al Despain and Dave Patterson for their suggestions, comments and careful review of this manuscript.

This study was sponsored in part by the Joint Services Electronics Program, Contract F44620-76-C-0100.

REFERENCES

Bell,C.G. and Newell,A. (1971):
in *Computer Structures: Readings and Examples,* McGraw-Hill 1971, Chapter 2.

Bell,C.G., Chen,R.C., Fuller,S.H., Grason,J., Rege,S. and Siewiorek,D.P. (1973):
"The Architecture and Applications of Computer Modules: A Set of Components for Digital Design", *IEEE Compcon,* March 1973, Conf. Proc. pp 177-180.

Brinch Hansen,P. (1972):
"Structured Multiprogramming", *Comm. ACM 15,* No 7, July 1972, pp 574-578.

Brinch Hansen,P. (1975):
"The programming language Concurrent Pascal", *IEEE Trans. Software Eng. 1,* No 2, 1975, pp 199-207.

Despain,A.M. and Patterson,D.A. (1978a):
"X-Tree: A Tree Structured Multiprocessor Computer Architecture", *5th Symp. on Comp. Arch.,* Palo Alto, CA, April 3-5, 1978, Conf. Proc. pp 144-151.

Despain,A.M. and Patterson,D.A. (1978b):
"The Computer as a Component: Powerful Computer Systems from Monolithic Microprocessors", submitted to *Comm. of ACM.*

Dijkstra,E.W. (1968):
"Co-operating sequential processes", in *Programming Languages,* ed. F. Genuys, Academic Press, London, 1968.

Haendler,W., Hofman,U. and Schneider,H.J. (1976):
"A General Purpose Array with a Broad Spectrum of Applications", in *Computer architecture workshop of the Gesellschaft fuer Informatik* Erlangen, May 1975, Informatik Fachberichte, Springer, Berlin, 1976.

Patterson,D.A., Fehr,E.S. and Séquin,C.H. (1979):
"Design Considerations for the VLSI Processor of X-TREE", submitted to the 6th Annual Symposium on Computer Architecture, Philadelphia, April 1979.

Séquin,C.H., Despain,A.M. and Patterson,D.A. (1978):
"Communication in X-TREE, a Modular Multiprocessor System", *ACM 78,* Washington D.C., Dec.4, 1978, Proc. pp 194-203.

Wirth,N. (1971):
"The programming language Pascal", *Acta Informatica 1,* 1971, pp 35-63.

Wirth,N. (1977):
"Modula: A language for modular multiprogramming", *Software - Practice and Experience 7,* No 1, 1977, pp 3-35.