MICROMODULES: MICROPROGRAMMABLE BUILDING BLOCKS FOR HARDWARE DEVELOPMENT

Richard G. Cooper
National Security Agency
Fort Meade, Maryland

I. Introduction

The algorithm design phase in the development of special purpose hardware is usually a very small part of the overall effort. A much larger portion - often 90% or more - is expended on logic design, fabrication, and debug. Furthermore, since the pure algorithmic complexity of hardware tends to be small, algorithm design errors typically account for a small part of the total debug time; errors due to electrical effects consume the lion's share. Precautions taken during machine design, fabrication, and debug to minimize reflection, switching noise, and synchronization errors are time consuming and expensive. This situation is at its worst when various types of equipment are to be produced in small quantities. As the use of Schottky-TTL and ECL increases, the problem will become more severe.

The purpose of the Micromodules project is to greatly reduce the amount of effort expended on logic design, fabrication and debug for small quantity developments. Secondly, with this modular approach, a quick reaction capability is sought that would allow a large reduction in the time interval between system specification and the delivery of the finished product. Finally, by simultaneously simplifying and speeding up the development process, we aim to improve the practicability of implementing more complex equipments.

These goals can be achieved by the development of a family of microprogrammable modules. Each module will be architecturally compatible with a small class of common hardware structures with obeisance to a standardized interconnection discipline. The system designer will obtain a collection of modules from inventory and configure them, by means of the interconnection discipline, into a system which is architecturally suited to solve the problem at hand.

It is likely that many systems will require some special hardware development in addition to the standard modules; our intention is to minimize the quantity and complexity of such special equipment. As the project progresses, additional common structures will be identified and the family of micromodules will be expanded to contain them when justified.

Our approach is not without precedent; the Macromodules project [1,2,3] at Washington University has been a fundamental source of inspiration. There, under the direction of W. Clark and C. Molnar, a set of asynchronous building blocks were constructed. These can be interconnected with standard cables. Loading factor allowances, noise attenuation and techniques for synchronization were built into each module. Functionally, their modules are quite simple. Using adders, registers, memories and other modules of similar complexity, they can construct systems of interconnected blocks which are effectively free from electrical errors. System implementation can be accomplished quickly and easily; it is not uncommon for an engineer to design, construct, and debug a significant system in a matter of days.

Due to the functional simplicity of each module, the relative cost of eliminating intramodular electrical errors is high. However, macromodules are intended for the construction of experimental equipment. A number of modules are configured to implement a certain algorithm; the system is used for a short period of time and the modules are then returned to the stockpile for later use. In such an environment, the cost of each module is not very important. It will be used in many different implementations and only a fraction of each module's cost need be attributed to each use. The time and effort required to build each experimental system is the more important consideration.

Our approach has been to apply the macromodular concept to the development of unique operational special purpose equipment. In this environment, the cost of each module is quite important; it will be used in only one machine; therefore, the relative cost per module of eliminating electrical errors must be reduced. To achieve this reduction, we chose to increase the functional power of each module rather than relax the interconnection discipline. A given algorithm would be implemented with fewer, more powerful modules; as a result, the overhead of eliminating electrical errors is reduced.

II. Microprogrammed Machines

Note that a more complex module increases the danger of sacrificing the flexibility required for constructing special purpose hardware of greatly varied designs. If flexibility is to be retained, individual types of modules should be modifiable within the range of their architectures to suit a diversity of applications. For this reason, our modules are often microprogrammed, i.e. designed with alterable control memories. Integrated circuit PROMs (programmable read-only memories) will be used to specify the functions to be performed by each module. When new applications of existing module architectures are required, new PROMs will be designed to tailor the modules to the application. With this approach, we can, in effect, create a wide variety of complex building blocks for a minimum of

developmental effort.

Most projects will still require some ROM design. Although many data routing and formatting functions will be satisfiable with basic designs, it is not likely that needed processing and sequencing functions will have been previously designed. Therefore, compared with the macromodular approach, a system built with micromodules will require more effort - weeks instead of days. Nevertheless, the effort involved in system implementation will be greatly reduced when compared with that of current, traditional hardware development methods.

The total cost of a given implementation will probably also be reduced. Cost reductions will be achieved in three areas. Since effort can be translated into dollars, substantial savings will be gained in reducing total effort. Because the modules will be produced in quantity, the economies of scale create further savings. Finally, the extensive use of MSI and LSI technology, usually unjustifiable in one-of-a-kind equipment development, also contributes to overall economy. Offsetting these reductions, several factors require expenditures not normally accruing to equipment development. The cost of developing the modules, their associated production tooling and inventory maintenance costs must be distributed among the equipments produced. Any portion of each module's capabilities that is not effectively used in a given equipment must still be purchased. Quantitative comparisons of these factors cannot be made at this time, but it appears that the overall cost per equipment will be reduced.

In order to clearly describe the micromodular approach, we must examine those user-microprogrammable machines currently on the commercial market.

The great majority of commercial machines are oriented towards emulation; for this reason, they tend to be complex and expensive. Because the machines will be used in stand-alone configurations, the architectural emphasis tends toward high speed full word arithmetic and logical processing overlapped with random access memory fetch. Very limited Boolean capabilities and almost no multiple Boolean decision and control functions are included. When used in special-purpose equipment, emulation machines require considerable interface logic. Relatively small amounts of local high speed storage are common, because main memory offers large amounts of cheaper, slower storage. Due to the complexity of emulation machines, their cost prohibits multiprocessing systems for many hardware applications. Even when multiprocessing is used, the burden of synchronization falls on the microprogrammer, or hardware synchronization must be provided.

Another large segment of the commercial market is directed towards the implementation of disk and tape controllers. Few of these are truly user microprogrammable and virtually all are fixed architecture machines. Synchronization of multiple machine configurations must be microprogrammed or implemented by means of additional hardware.

Recognizing the limitations of current machines, we decided to use a building block approach with the micromodules. Each module is designed to solve a small class of common hardware problems, without frills. The emphasis is on low cost, high instruction cycle rate, and the possibility of cooperation between modules. Although the class of problems compatible with each module's architecture is small, several modules can be configured to achieve the requirements of a given implementation.

III. Modular Design Considerations

The separation of functions is an important theme in the design considerations. Since microprogramming can be a difficult task, the separation of functions is useful in dividing the problem into subproblems which can more easily be solved. Each subproblem can then be attacked using the most appropriate module. As new classes of subproblems are identified, new modules tuned to these classes can be developed. System debug can also be simplified by the subproblem approach; each subsystem can be debugged individually, postponing debug at the system level until the last subsystems are ready.

Since systems will be constructed from collections of modules, synchronization and buffering are also important considerations. Facilities for synchronization and buffering are built into each module in hardware. In most practical cases, loop-free networks of modules can be constructed, freeing the designer from these problems. Where loops must be constructed, some simple precautions will ensure that no deadlock problems exist. As will be shown later in this paper, a minimal amount of programmed synchronization can greatly improve efficiency for certain kinds of processes.

Connections between modules can be either arithmetic or Boolean. Arithmetic paths are eight bits wide (one byte). Each byte path is constructed by connecting a polarized ten-conductor cable between a byte output port on one module and a byte input port on another. Each port maintains a FULL flip-flop which specifies whether the port contains data. When data is transferred from an output port to an input port, the FULL flip-flop in the output port is cleared and the FULL flip-flop in the input port is set. The transfer of data between ports and control of the FULL flip-flops during transmission are performed completely in hardware. Two wires in the ten-conductor cable are used for handshaking signals. Transfer between ports is accomplished by logic built into each port. Since each port contains its own data buffer register, the interconnected modules can be performing computations while the transfer is taking place.

Synchronization of byte data transfers with processing is accomplished by use of the FULL flip-flops. If a microinstruction attempts to read data from an input port which does not contain data, completion of that instruction is suspended until data is transferred into the port by the handshaking logic. When an input port is read, its FULL flip-flop clears, allowing the handshaking control to transfer in another byte. Thus each access of an input port reads a new byte of data regardless of the input arrival rate. Similarly, a microinstruction that attempts to place data into an output port, which already contains data, is suspended until the

port empties. Since both input and output ports contain data storage registers, all byte transfers between modules are double buffered by the hardware.

Each arithmetically oriented module can contain multiple input and output ports for byte data. Thus data words larger than eight bits can be transferred serially by byte or in parallel along several cables. Parallel transfers occur independently. Since modules can be processing while transfers take place, and since data is double buffered, the duration of data transfer can be several instruction times long without much degradation of performance. This relatively slow data transfer, combined with fixed loading factors and reflection characteristics, reduces electrical interconnection errors to a low level. Resistor terminators are built into the input ports and interconnection cables are shielded.

Boolean interconnections are of two kinds: level signals and pulsed signals. Level signals are useful for connections between the modules and peripheral equipment. Level signals can be used for controlling and sensing Boolean lines, e.g. tape and disk drives. Pulsed signals are useful for synchronization tasks within the network of modules.

Coaxial cables are used for transmitting Boolean signals and each module can contain one or more Boolean input and output ports. Switches are provided on some modules to specify whether a port will be a level or pulsed signal device.

Level signals are strobed into flip-flops at the beginning of each instruction cycle to assure unambiguous operation. Schmidt triggers are used in some modules to perform level conversion and signal conditioning.

Pulsed signals require a rise and fall cycle of operation. A two phase flip-flop configuration is used on the input lines to synchronize pulsed signal transmission. A received pulse is stored in a flip-flop until the receiving module tests that flip-flop. When a pulsed flip-flop is tested, it is automatically reset. Pulsed signals are therefore not acknowledged in hardware by the receiving device. If a given system requires acknowledgement, this task must be performed in firmware.

IV. Design Aids

The design of ROMs, as has been previously stated, is a difficult task. For many projects, ROM design will be the most time consuming part of system implementation. For this reason, numerous ROM design aids are planned. Design aids will be written in time-sharing Fortran IV for the DEC PDP-10.

A basic table-driven assembler will be constructed. Individual symbolic assemblers can then be written for each module by providing the basic assembler with the proper tables.

A single preprocessor program will be used to expand macro routines prior to assembly. Alphanumeric text, consisting solely of macro control statements, will be input to the preprocessor. Expansion will then be independent of the individual assembly languages; thus the macro capability need not be provided for every version of the assembler. ROM designers must expand macro calls individually and then edit the expanded macro text into the body of the program.

A functional simulation of each module will be provided. The ROM designer can then debug his microprogram by repeated cycles of editing, assembly and simulation in a manner similar to the debugging of software.

An interconnection simulation routine will be used to debug configurations of modules. This routine will be an event-table simulator which enables the system designer to observe the interaction of the modules in a system. The degree of overlapped operation can be observed and the effects of alterations to individual modules on the configuration can be ascertained.

When ROM designs are completed, each object program can be dumped to paper tape. A ROM can then be physically constructed by "burning" the pattern specified on the paper tape into PROM integrated circuits.

System implementation would be accomplished by the software simulation process of ROM design, followed by ROM pattern fabrication. The ROMs would then be plugged into the appropriate micromodules obtained from stock. Standard cables, also obtained from stock, would be used to interconnect the modules.

V. Networks of Modules

An important goal of the Micromodules project is to facilitate the construction of more complex equipments than are feasible with traditional methods of constructing hardware. In particular, we wish to encourage the use of large networks of modules. Two adaptations of well known techniques are expected to be of general use in such systems: pipelining and parallel processing.

A. Pipelines

Pipeline structures are particularly appropriate to the micromodules. Let each packet of data be represented as x. Let the function f(x) be computed by a pipeline of n stages, thus

$$f(x) = f_1(\ldots f_{n-1}(f_n(x))\ldots)$$

as illustrated in figure 1.

In designing a pipeline, each processing element should compute the appropriate function in a fixed time period. Thus each packet spends an identical amount of time in each processor. If the subfunctions to be computed do not have identical computation times, synchronization circuitry must be included in the design. If some of the subfunctions require random computation times, the buffering of data packets must also be provided for the sake of efficiency. Furthermore, each processor should be designed so that its average computation time is approximately equal to that of the other processors. Because of these optimization problems, pipeline structures are not often practicable for the implementation of complex functions.

However, a modified version of the structure (figure 2) allows the pipeline concept to be applied to a larger class of practical problems. Let the packet y be defined as the augmented pair of elements

$$y = (i,x)$$

where i is a tag value (initially, i=n) representing the next subfunction to be computed, i.e. $f_i(x)$. Let there be m processing elements g_j, for j=1,...,m. After each processing element, there is a buffer q_j of fixed size. Let b_j be a Boolean feedback signal from q_j to g_j such that

$$b_j = \begin{cases} 1 & \text{iff } q_j \text{ is more than half full} \\ 0 & \text{otherwise.} \end{cases}$$

The b_j signal allows the processing element to determine the state of its output buffer. By considering the tag value i of its current packet and the state b_j of the output buffer, each processor makes the decision to pass the current packet to the buffer or to compute the next subfunction. Thus

$$g_j = \begin{cases} g_j(i-1,f_i(x)) & \text{iff } (b_j=1) \text{ and } (i>0) \\ (i,x) & \text{otherwise} \end{cases}$$

$$\text{for } (0<j<m+1).$$

Note that b_m=1 regardless of the state of q_m. This fact allows each processing element to contain the same microprogram. Furthermore, the microprogram is not dependent on m or n. Thus a pipeline executive microprogram can be written and debugged for arbitrary m and n values. The executive would only be concerned with reading and writing data packets, and with making the decision to process or pass the current packet. System design of a pipeline could be performed by combining the executive with a list of packet sizes and subfunction addresses in a table indexed by i, and the microcode for each subfunction.

Since the pipeline structure does not depend on m, fast failure recovery is facilitated. The faulty module can be quickly removed from the pipeline and the system can be restarted with a structure of size m-1. Performance would be degraded, but the structure could still operate with up to m-1 failures.

It was stated previously in this paper that loop-free interconnect structures could sometimes be implemented for algorithms which contain loops. The simplest method would be to contain the loop within a single module by means of the microprogram. Loops can also be integrated into the pipeline structure by a modification of the definition of g_j; let the subfunction microcode also compute $s_i(i)$, the next value of the tag i. Thus

$$g_j = \begin{cases} g_j(s_i(i),f_i(x)) & \text{iff } (b_j=1) \text{ and } (i>0) \\ (i,x) & \text{otherwise} \end{cases}$$

with each iteration of a loop being considered as a new invocation of the same subfunction.

Either definition of g_j preserves the order of packet throughput. Although one packet can be completely processed in the first element and another packet partially processed by each stage, each packet will leave g_m completely processed and in the original order.

Because of the importance of the pipeline concept, a data flow simulator has been programmed. The system designer can specify the type and shape of computation time distributions (assuming they are independent) and the packet size for each value of i. By selecting values of m and by combining or reducing subfunction definitions, he can determine the most effective implementation of those considered.

B. Parallel Processing

The use of parallel structures like that in figure 3 is also anticipated. An input controller I is used to schedule the flow of input data to each of the m processors g_j, while output controller O merges the result streams. The mode of operation depends upon the characteristics of the function f. If f requires a nearly constant amount of processing time regardless of the data packet values, a phased sequence of processing can be scheduled by I and O. If packet transfer time is t_d and computation requires time t_f for each packet, then for maximum throughput,

$$m > [\ (t_f + 2t_d) / t_d\ ] \quad \text{for } t_d > 0$$

If t_f is random with a significant variation, a more complex structure and scheduling algorithm might be used. Data packets can be buffered as shown in figure 4 with the scheduling controlled by the state of the buffers. For the input controller, let b_j be the Boolean state signal defined previously. Then a good scheduling algorithm might be

$$n = \min(j) \text{ such that } b_j=0$$

where n, if defined, is the subscript of the next buffer q to receive a packet of data. Similarly, if a_j is the Boolean state signal for the jth output buffer, then let

$$k = \min(j) \text{ such that } a_j=1$$

else if no a_j=1,

$$k = \min(j) \text{ such that } r_j \text{ is not empty.}$$

For maximum throughput,

$$m > [\ E((t_f + 2t_d) / t_d)\ ]$$

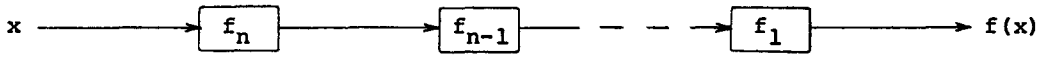
Note that the order of packet input is not preserved at the output for the case of random scheduling. If order must be preserved, then an order index can be attached to each packet at I. This index can be used by O to place the results in order. Let l be the maximum number of packets containable by the system in figure 4. Let u be the maximum possible computation time, and let v be the minimum. Then three buffers of size w are required for reordering where

$$w = [(lu)/v]$$

For the random scheduler, a data flow simulation is planned that will be similar to that for the pipeline structure. Several executive routines for phased and random schedulers, with and without order indexing, will be written.

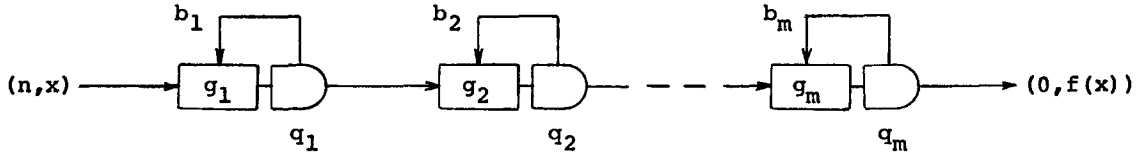
VI. Summary

The Micromodules project is directed towards the simplification of hardware design and implementation. A powerful and flexible



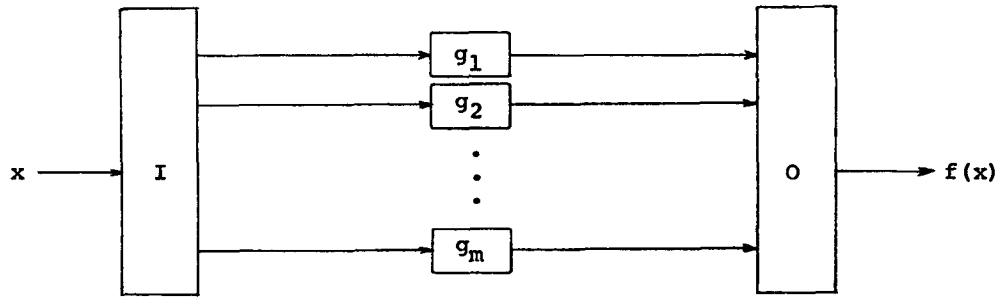
$$f(x) = f_1(\ldots f_{n-1}(f_n(x))\ldots)$$

Figure 1. A Pipeline Structure



$$g_j = \begin{cases} g_j(i-1, f_i(x)) & \text{iff } (b_j=1) \text{ and } (i>0) \\ (i,x) & \text{otherwise} \end{cases}$$

Figure 2. A Self-Optimized Pipeline Structure



$$g_j(x) = f(x) \quad (0<j<m+1)$$

Figure 3. A Parallel Processing Structure

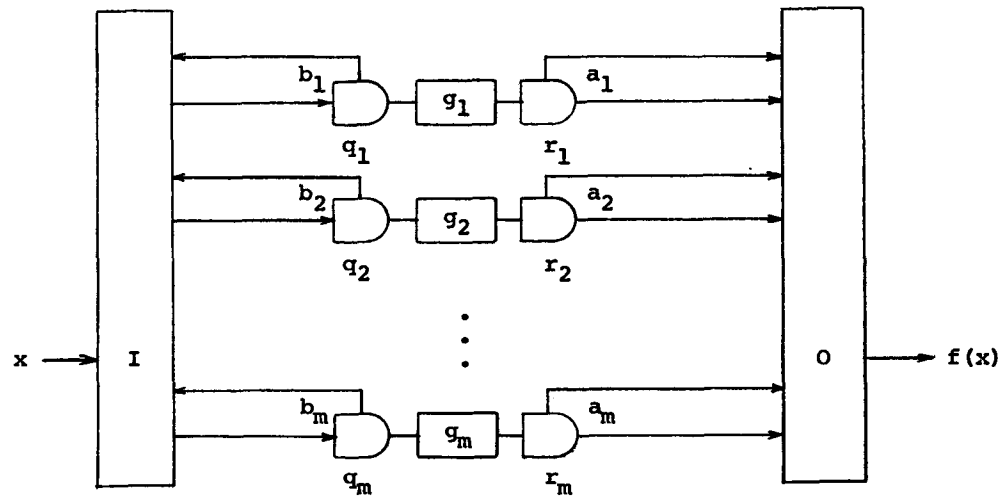


Figure 4. A Self-Optimized Parallel Processing Structure

set of microprogrammed modules is provided. The use of a standardized interconnection discipline, with an emphasis on the elimination of electrical errors, allows the engineer to concentrate on the architectural aspects of his problem.

The system designer will have two powerful structures at hand: the pipeline and parallel schedulers. He can design microprograms for the functions to be computed. Using the computation time characteristics of the function microprograms, he can simulate a data flow model and manipulate the model to achieve the desired throughput. Finally, he can assemble an arbitrary network of modules without bearing the burden of synchronization and buffering design.

A basic family of four micromodules is now in the development stage. Future work will include the identification and realization of other useful structures, whether microprogrammed or hardwired. A continuing effort to construct ROM designs with broad applicability and to further improve design aids is anticipated. Some effort will also be made to discover other basic system structures which would be useful in distributing processing tasks among a collection of modules.

References

[1] W. A. Clark, "Macromodular Computer Systems", 1967 SJCC Proceedings, p335

[2] S. M. Ornstein, M. J. Stucki and W. A. Clark, "A Functional Description of Macromodules", 1967 SJCC Proceedings, p337

[3] C. E. Molnar, S. M. Ornstein and A. Anne, "The CHASM: A Macromodular Computer for Analyzing Neuron Models", 1967 SJCC Proceedings, p393

Bibliography

[4] H. H. Loomis, Jr. and M. R. McCoy, "A Scheme for Synchronizing High Speed Logic: Part I." and "... Part II.", IEEE Trans. Computers, January and February 1970

[5] H. H. Loomis, Jr., "The Maximum Rate Accumulator", IEEE Trans. Computers, August 1966

[6] C. G. Bell and J. Grason, "Register Transfer Modules (RTM) and their Design", Computer Design, May 1971

[7] D. Misunas, "Petri Nets and Speed Independent Design", Comm. ACM, August 1973