FORTH for a Multimicroprocessor Control System

C. A. Myerholtz, A. J. Schubert, M. J. Kristo, and C. G. Enke
Department of Chemistry
Michigan State University
East Lansing, Mich. 48824

In our laboratory, a triple quadrupole mass spectrometer (TQMS) has been placed under complete computer control. A distributed-processing system of several microcomputers were required to control the more than 30 instrumental parameters, acquire data and perform peak-finding in real-time. The hardware, which consists of four 8088-based microcomputers (one master and three slaves) was designed and built using the Newcome-Enke Bus (1). Hardware supporting three modes of interprocessor communication are provided: direct memory transfer (DMT), which allows large blocks of data to be moved from one processor memory to another, command transfer (CT), which allows the master to load a list of routines to be executed into the command buffer provided in each slave, and status check (SC) which allows each processor to check a set of status flags from any other processor. All of these modes can be executed without interrupting or delaying any of the slave's real-time operations (2). The hardware was designed to provide the maximum modularity and flexibility for the changing needs of the scientific research environment.

The software for this system was based on a standard FORTH operating system (3). FORTH was chosen as the programming language for this multimicroprocessor system, because its modularity complemented the modularity of the hardware and because its speed, low memory requirements, and extensibility lead to greater ease in programming, testing and operating the instrument. The basic FORTH system which normally operates in a single processor environment had to be expanded to handle the interprocessor tasks of block data transfer, parameter passing, and task assignment. This involved additions to the FORTH system running on the master processor and modifications to the FORTH system for the slave processors.

The first step in adapting FORTH to a distributed processor environment involved modifications to the target compiler on the master processor. Several interprocessor memory access words were developed that utilize direct memory transfer hardware to transfer large blocks of data between the memories of any two processors in the system. These words first select a slave processor and then transfer data between the master's parameter stack and the selected slave's memory. Two slave control words, RESET and HLD, utilize the control signals provided by the command transfer hardware to reset a selected processor or put a processor in the HOLD state, preventing any program execution. These interprocessor control and access words were used to create a modified version of the target compiler that directly downloads the code into the desired slave's RAM instead of writing the compiled code to the disk.

There are several advantages to having a compiler that directly

downloads code into the target processor. For instance, the compiler can initialize variables and tables in the slave processor during compilation. This eliminates the need to store special slave initialization routines in slave memory space even though they are only used once. The devices interfaced to the slaves are all memory-mapped, so the compiler can also initialize all of them appropriately. Also, since the compiler interprets code from the disk, the entire specialized initialization procedure may reside on the disk rather than occupy any system memory.

For the master processor to instruct a slave processor to execute a selected routine, the master processor should have a list of the routines available in each slave. To accomplish this, a word was added to the compiler which records information about selected slave commands into a Slave Command Access Table (SCAT). This new word, COMMAND, is a FORTH immediate word. When executed, COMMAND records into the slave's SCAT the first three characters of the routine's name, the length of the name, and the code field address of the routine. This information is later used to direct he execution of a task in a slave processor.

The source code of the basic FORTH system implemented in the slave processor had to be modified so that the slave processor can receive instructions from its first-in first-out (FIFO) command buffers, into which the master writes all of its commands to the slave. The code that normally interprets commands and data from a terminal was replaced with new code which performs the same function using the command FIFOs. The result is a slave interpreter that operates in the following manner: using the command buffer hardware, the slave monitors the condition of its command FIFO. When the FIFO becomes not empty, the 24-bit value in the FIFO is read. Eight of the 24 bits either force an immediate control operation or tell the microprocessor whether the remaining 16 bits are data or a command. If an immediate control operation was executed, the lower 16 bits are discarded. If the lower 16 bits contain the code field address of a FORTH word to execute, control is then transferred to the routine at this address. When execution is completed, control will return to the command FIFO interpreter. If the lower 16 bits are data, the value is then transferred to the slave's parameter stack. This method of moving numeric data to the stack and initiating the execution of a word appears to the rest of the FORTH system to be identical to interpreting text from a terminal. This new command interpreter was installed in such a manner as to preserve the multitasking capabilities of the polyFORTH system. Thus, a slave can be running a background task, such as an oscilloscope display, and still be able to act on new commands sent to it through the command buffers.

The slave processors do not possess terminals or disk drives; thus the support code for these devices is not normally down-loaded into a slave. This reduces the basic FORTH slave system to approximately 2 Kbytes. However, for debugging purposes, terminal support software can be loaded into a slave processor to allow a programmer to interact with it directly. Normally the user directly interacts with only the master processor. Since the slave interpreter mimics normal operation from a terminal, a programmer may test a routine on the master and be confident that it will behave in the same manner when it is down-loaded into a slave.

To allow programs running on the master to pass data and commands to the various slave processors, several new words were defined for the master processor FORTH system. The first of these follows the form nPUSH, which pushes the value at the top of the master processor's stack to the top of the nth slave processor's parameter stack. This becomes the primary method of parameter passing from the master to the slave processors. The second type of word was designed to ease access of variables and arrays in a slave processor from the master processor. It follows the form nLABEL and defines a new word on the master processor that when execute selects the appropriate slave and leaves the address of a variable in the slave on the master's stack. The execution of words defined by this command prepares the master processor for use of one of the interprocessor memory access words. The third major type of word added follows the form SLn, and allows commands to be passed to a slave in the form of addresses of FORTH words to execute. When executed, a word of this form looks up the code field address for the word that follows it in the Slave Command Access Table for slave n. If the look-up is successful, the address is transmitted to slave n's FIFO as a command to be executed. The SLn and nPUSH commands use the status hardware, which is provided for each microcomputer, to determine if a slave's FIFO is full or not. If the FIFO is full, the command passes control to the next task in the multitasking loop. When control is returned to the command it checks the FIFO status again. This process is repeated until data can be transferred to the slave's FIFO.

The use of this type of software system is not limited to the dedicated hardware developed in our laboratory. The same approach can be implemented on any microprocessor system with shared memory. Additional software would be required to emulate the various communication modes implemented in hardware in our system. However, the specialized communications hardware developed as a part of this project offloads some of the burden from the software and offers, not only vital time savings, but also the ability to perform command transfer and status check without interference with any slave's real-time tasks. Even direct-memory transfer affects only the processors involved. A software system based on the system described here has been implemented on Intel's Multibus system, for example.

In conclusion, we have developed a programming environment for a distributed processing system that is simple and easy to use. The programmer can interact with the system as a unit, rather than a series of isolated microprocessors. The programmer has access to all words defined for all processors and processor linkage at execution time is transparent to the programmer.

References

1. B. H. Newcome and C. G. Enke, Rev. Sci. Inst., 55, 1984, 2017.
2. B. H. Newcome, C. A. Myerholtz, and C. G. Enke, in preparation.
3. Forth, Inc., Hermosa Beach, California.