The architecture and implementation of the FLEX/32 MultiComputer

by NICHOLAS MATELAN
Flexible Computer Corporation
Dallas, Texas

ABSTRACT

The FLEX/32 MultiComputer is a generic environment for cooperating multiple processors. The FLEX/32 can support a number of different processors, making it heterogeneous in terms of the instruction sets it can support, and homogeneous in its ability to provide consistent storage and input/output facilities to its differing processors. These facilities are accessed through standard 32-bit VMEbus connections.

The FLEX/32 supports the full UNIX System V Operating System and languages associated with it, plus the extended ConCurrent C, ConCurrent FORTRAN 77, and Ada languages that allow programming of concurrent software at a high level. Direct programming support at all levels is provided by the environment hardware for concurrent software execution and optimization, including hardware support for shared resource access arbitration, conditional critical region arbitration, and interprocessor messages.
INTRODUCTION

The past 30 years have seen dramatic improvements in the performance of computers. In general, these improvements have been due to improvements in components. Today, as we approach physical limits in the performance of components, the architectures of computers become more and more important. Though we still see a great deal of improvement to be made in the application of faster component technologies, the effects of these components on conventional architectures using a single central processing unit begin to show diminishing returns. Architectures based on multiple central processing units show promise of providing not only increased power, but also increased flexibility in meeting the varying demands of future computation. The FLEX/32 is such a system (see Figure 1).

The FLEX/32 is a MultiComputing Environment; that is, it is an environment that supports multiple computers working on one or more tasks together or independently under coordinated software. These computers need not be the same. Therefore, the environment is heterogeneous. The computers supported in this environment can differ in power, in the amount of memory supported, and in their basic orientations. Some of these computers could contain processors dedicated to control; others might be used for array processing and floating-point operations, for example.

The addition of a new computer, with its new processor and instruction set, requires only adaptation to the environment. This means that once the adaptation has been made, the generic software and input/output capabilities of the environment are fully available to the new processor. Such software includes the UNIX System V Operating System and other special tools needed for developing concurrent programs. Input/output is performed via a set of standard VME buses. These buses support interfaces to peripheral equipment that may be purchased from any of the 80 to 100 current providers of VMEbus interfaces or from Flexible Computer Corporation, giving a truly open architecture.

A final, but no less significant, feature of the environment is its SelfTest System. Built into the environment and distributed throughout its modules (computers, memory, and peripheral interfaces) are test circuits dedicated to determining the health and performance of the environment as a whole. This system allows not only such features as automatic shutdown and restart in response to power failures, but also fault isolation and repair verification and performance analysis based on information collected during the run-time execution of programs.

LEVELS OF DESCRIPTION

In describing a computer, it is important to distinguish between its architecture and its implementation. An architecture is a description of the fundamental attributes and functionality of a device or program without regard to its detail. An implementation is a description of the collection of details needed to construct a product. It is the product that provides certain levels of performance and power. It is the architecture that provides product line consistency throughout its history.

Both hardware and software have an architecture. A major thrust of the FLEX/32, in addition to its great flexibility of configuration, is to provide an environment for programmers to produce software more cost effectively. The dichotomy of the architecture and implementation is maintained primarily to guarantee that software interfaces to the FLEX/32 MultiComputing Environment will not change throughout the life
of the product line. We all know that as time passes there will be better ways to produce hardware that is faster, more reliable, and cheaper. However, it is the intention of the Flexible Computer Corporation that changes in FLEX132 hardware, as they must come, will not involve changes in existing application software. The architecture defines a set of invariant interfaces for the users and builders of software intended for the FLEX132 product line, even as its hardware is adapted over the years to the use of newer and better techniques. Its software, at every level, can remain invariant because its interface to the hardware remains invariant.

THE FLEX ARCHITECTURE

The FLEX132 is a Multiple Instruction Stream/Multiple Data Stream (MIMD) Multiprocessor System. Its architecture (generically represented in Figure 2) is composed of devices, buses, processes, and topologies.

Devices are either computational or peripheral. Computational devices include processors, memories, bus interfaces, interprocessor signaling mechanisms, and common locks. Peripheral devices are the devices that control and sense the outside environment. They include tapes, disks, printers, terminals, and their controllers.

Devices are connected one to another by buses. Four separate buses are defined in the FLEX132 architecture. These are the common bus, the local bus, the peripheral bus, and the SelfTest Bus. Common buses are those that are intended to allow the sharing of resources, such as shared memory or shared devices. Local buses are intended to provide paths between a processor and its attached local devices, such as local memory. Since each local bus is defined as multimaster, more than one processor can be attached to it.

Peripheral buses are those that allow attachment between devices on local buses to the controllers of peripheral devices and actuators in the outside environment. The SelfTest Bus, which runs throughout the MultiComputing Environment, is that transfer path that allows information about the state of the machine to be received and control of the machine to be sent.

Of course, all this hardware would be useless without a method for applying algorithms to control and direct operations toward a useful result. The FLEX132 supports multiple, truly concurrent processes. True concurrency is the execution of processes on different processors at exactly the same time. This is often called multiprocessing. Apparent concurrency is the execution of processes in a shared fashion on a single processor at a rate fast enough to lead an observer to believe that they execute at the same time. This is usually called multitasking or multiprogramming. The FLEX132 is a parallel processing system and therefore allows both true concurrency and apparent concurrency, depending on the needs of the programmer.

Each of the devices, processes, and buses outlined above may be connected in number of possible configurations. Figure 2 shows the physical topology of the FLEX132 MultiComputing Environment. Any number of local buses may be attached through ports to several common buses. Common buses are fully arbitrated and will allow only one access at a time from each local bus in a fair arbitration scheme. Each local bus is attached to its own common lock device. This device can be programmed to allow processes mutually exclusive access to shared resources for an arbitrarily complex use of shared data. In general, interprocessor communication is maintained through shared memories residing on the common bus. Notice that this topology does not allow processors to reside on the common bus, only resources. Processors are always attached to local buses. Peripheral devices are attached to local buses through bus interfaces.

There is no limit to the number of local buses or the number of processors that may be attached to local buses in the architecture. Implementations, however, must put a bound on the number of buses and processors that can be contained in any particular unit, such as a cabinet. The architecture, however, fully allows multiple cabinets; therefore, even in the implementation there is no inherent bound on the number of processors that can be placed together in a FLEX132 configuration.

The SelfTest Bus connects all devices in the architecture. Processes that actually perform selftest functions reside within one of the processors in the system.

THE FLEX132 HARDWARE IMPLEMENTATION

The FLEX architecture is an abstract representation of the allowable interconnections of devices. As such, it makes no demands for execution speed, word sizes, and so forth. The FLEX132 product line is a 32-bit implementation of the FLEX architecture. It specifies the technology from which the FLEX132 is built, its packaging, and its physical parameters. The FLEX132 hardware implementation is the environment...
used to carry out the commands of the FLEX software described below.

The philosophy of the FLEX/32 implementation is the provision of a flexible, universal hardware and software environment for a number of different instruction sets. Just as software environments have become a way of supporting programs written in different languages, the FLEX/32 hardware is an environment for different processors. These processors are supplied with generic memory, input/output support, and multiprocessor and network interprocess communication mechanisms. This environment is the same for each processor type, but the instruction sets supported are different. This allows not only software written in different languages, but also different machine instruction sets, to execute together.

The components of the FLEX/32 hardware are divided into the Card Level, the Backplane Level, and the Unit Level. Circuit cards define the replaceable module level of the FLEX/32. There are three classes of cards in a FLEX/32 System. These are Universal Cards, Common Communication Cards, and Peripheral Cards.

Universal Cards support local bus activities such as computation, memory storage, and array processing. Common Communication Cards allow access to, arbitration of, and control over the common buses and their shared resources, such as the common memory. Peripheral Interface Cards are standard single, dual, or triple Eurocard interfaces available from commercial manufacturers. There are currently 80 to 100 commercial manufacturers producing cards to control standard peripheral equipment based on the VMEbus Eurocard format. These cards are interconnected via a backplane supporting the local bus, common bus, and SelfTest Buses.

The local bus is a standard, asynchronous VMEbus with extensions for control internal to the FLEX/32 cabinet. The common bus is a synchronous version of the local bus. The Common Communication Cards house a high-speed shared memory. The SelfTest Bus is an RS422 bus supporting the SDLC protocol. All external communications are through bus interfaces on Universal Cards to standard VMEbuses (no extensions). Peripheral buses are attached to standard VMEbus Interface Cards.

The cards connected to the backplanes are supported by card cages and are divided into two types of units. One unit is the MultiComputer Unit, or MU. It is the MU that houses both Universal Cards and Common Communication Cards. All Interface Cards to peripheral equipment are housed in a second type of unit, called a Peripheral Control Unit, or PCU. Cables between the two units allow computers attached to local buses to control their various I/O devices.

The MultiComputer Unit can house up to 10 local buses and 2 common buses. There can be 20 universal cards in a MultiComputer Unit, 2 per local bus. The MultiComputer Unit supports up to 10 communication cards, 1 for each local bus, allowing any processor attached to any local bus to communicate with any other processor, either through a shared memory associated with a Common Communication Card or through the direct interprocessor messaging capability. Furthermore, a common lock capability allows a processor to define a critical region in the shared memory and to own that region for operations without affecting traffic on the common bus.

The initial Universal Cards offered by Flexible Computer Corporation are Computer Cards and the Memory Card. Computer Cards, based on either a 32032 or a 68020 processor, include an attached floating point coprocessor and 128K bytes of ROM, one or four megabytes of memory protected by error correction and detection logic, and a VMEbus port that can be configured to either 32 or 16 bits of data. The processor can access its bus interfaces and its memory on this card without affecting the operation of its local bus. This is important when both slots associated with local buses are used for computer cards.

The Memory Card supports from 1 to 8 megabytes of random access memory protected by error correction and detection logic. It also contains a VMEbus interface.

The MultiComputer Unit can be configured in a number of ways. For example, a unit can be configured with 20 computer cards, giving a machine with 20 processors, 20 to 80 megabytes of memory, and 20 VMEbus interfaces. This system could also be configured to support up to 20 megabytes of fast common memory. Another system could be configured with one computer card, and the remainder of the MU filled with memory. This would give a system with 1 processor and 20 VMEbus interfaces plus 153 megabytes of memory. A more usual card complement would be four or five computer cards with an extra Memory Card, giving processors with 9 megabytes of memory each and perhaps a few single processing cards without extra memory. Other computer card types will include floating-point capability in the 4-to-6-megaflops (millions of floating-point operations per second) range. It should be noted that all FLEX/32 processor types can be mixed and matched in each FLEX/32 MultiComputer Unit.

VMEbus interfaces can be simply cabled together, giving extra shared paths other than those associated with the common buses. Interprocess communications over these paths can be made using read/modify/write interprocess communication instructions between local memories. The same jumpers can be used to connect multiple MultiComputer Units, forming much larger systems. Four of these VMEbuses, for example, could be used to connect to neighbors north and south, and east and west. Such a method could be used to define a plane of MultiComputer Units. Similarly, six interconnections could be used to define hypercubes of MultiComputer Units yielding a large number of computers (dozens to thousands) that could be applied to the same tasks. The possibility of such large multiple-processor systems makes the selection of the algorithms very important in determining the usefulness of any configuration. As is the case with any computer, infinitely expandable may not mean infinitely useful, except for a narrow range of algorithms. It is fortunate, however, that some of these algorithms are very useful indeed.

FLEX/32 SOFTWARE IMPLEMENTATION

For system development, Flexible Computer provides the full UNIX System V Operating System supported on each computer within the MultiComputing Environment.
Flexible Computer has also extended the C and FORTRAN languages to produce the new languages ConCurrent C and ConCurrent FORTRAN. These languages are standard C and FORTRAN with an extra set of statements that allow direct specification of concurrent programs for execution in the FLEX/32 environment.

The FLEX/32 can execute programs directly under ConCurrent C or FORTRAN program control instead of under UNIX. Flexible supplies a set of MultiComputing Multitasking Support Utilities to facilitate such dedicated operation.

The Ada language is also available.

**UNIX System V**

UNIX System V is a true industry standard for software development. It includes support software such as Source Code Control System (SCCS) and its associated editors and language processors, such as FORTRAN 77, which Flexible has extended with the ISA real-time extensions (S61.1), RATFOR, SNOBOL, and Assembly Language. It provides development and debugging tools and file management capabilities within the most portable operating system presently available. In addition, concurrent execution of processes can be simulated, using the shared-memory software capability of UNIX System V, or truly executed simultaneously.

**The ConCurrent C and ConCurrent FORTRAN Programming Language**

The C Programming Language has proved an excellent tool for programming in a sequential processing environment. The ConCurrent C Language is designed to further increase the capabilities of the C language by facilitating direct concurrent and real-time processing for advanced parallel multiprocessor systems while maintaining the original C style and philosophy. C is upward compatible to ConCurrent C, which preserves all of C's definitions and features.

ConCurrent C extended constructs are categorized in two classes: new variable definitions (event variables and shared variables) and new control-flow statements (process interaction, process control, concurrent execution, and event supervision).

ConCurrent C introduces a new type, event variables, to support real-time event handling. All real-time events are either timers or exceptions. The keywords *timer* and *exception* are used to declare and define event variables.

The WHEN statement is used to suspend its enclosing process until a specified event is satisfied, at which time its associated statement list is executed. The WHENEVER statement is provided to support nonprocedural event supervision.

The WHEN statement is also used, in addition to procedural event response, to synchronize access to shared data between processes. The WHEN statement can thus be used to directly implement the Conditional Critical Region technique of sharing data.

The process concept is the basis of true concurrent execution. A process in ConCurrent C is defined and started by a PROCESS statement.

Proper combinations of these statements and other existing C Language statements can define and cause to be executed every known multiple process intercommunication technique, including semaphores, monitors, and messages.

Each of the capabilities of ConCurrent C listed here is also available in Flexible's ConCurrent FORTRAN 77.

**MultiComputing Multitasking Operating System**

The FLEX/32's MultiComputing Multitasking Operating System (MMOS) provides support for real-time, run-time embedded applications.

The MMOS Utilities are resident in the System Library and are included by the loader to resolve all external calls generated by the ConCurrent C preprocessor. The capabilities of the MMOS Utilities include the following:

1. **Priority Oriented Task Management and Multiprogramming**
2. **MultiComputing, by providing interprocessor communication, synchronization, and data protection for concurrent or sequential processing**
3. **Interprocess Communication and Signaling**
4. **Event Management to supervise conventional interrupts, Interprocessor Messages, user-defined exceptions, system defined exceptions, and timers**
5. **Memory Pool Management**

**A METHODOLOGY FOR CONCURRENT PROGRAM DEVELOPMENT**

Of major importance in producing commercial-quality (that is, useful) concurrent programs is the availability of a development methodology fully supported by software tools.

Figure 3 provides a block diagram of the flow of program compilation, loading, and execution within the FLEX/32 MultiComputing Environment. The steps from sequential code development through final concurrent program integration constitute the FLEX/32 development methodology.

At the top of Figure 3, a ConCurrent C program source file is shown. It is first processed via the ConCurrent C Preprocessor, resulting in the output of a C Language Source File containing unresolved MMS system calls. The Preprocessed ConCurrent C Source is next compiled under UNIX System V by the C compiler, resulting in an object code file. The object code file can then run through the system loader and have all system calls resolved by the MMOS Utilities contained in the System Library, resulting in an executable image file.

The right side of Figure 3 depicts the several execution options provided by the FLEX/32 MultiComputing Environment. Three different environments are provided with the FLEX/32 MultiComputer. The first allows execution under a Concurrent Executive and provides for a true parallel-computer, concurrent operating environment. The second is execution in a simulation of a concurrent environment under
the UNIX System V environment. This program is called the FLEX/32 Concurrency Simulator. The third environment is UNIX System V, which allows either nonconcurrent (sequential) operation or distributed operation.

The FLEX/32 methodology for concurrent program development is a phased migration of processes from one environment to another. Separate compilation and testing of sequential programs under UNIX and its tools allow a number of programmers to contribute to the development of large software systems. As these programs are developed and known to be functionally correct, they can be collected under a Concurrent C program as processes. This concurrent program can be fully debugged, using the Concurrency Simulator, still under UNIX and its many support tools. If appropriate, this program can be forever executed in multiple processors under UNIX. If, however, the program was intended to execute directly on the MultiComputer, processes can be moved one at a time, or all at once, into the intended processors under MMOS. This allows incremental use of true concurrency from the shelter of apparent concurrency and UNIX support afforded by the Concurrency Simulator.

The diagram also depicts a number of source processes written in some of the languages that can be compiled via their individual language compilers and combined by the system loader. Again, their system calls are resolved, and images produced that allow them to execute together in different computers under any of the concurrent, simulated concurrent, and nonconcurrent environments provided.

CONCLUSION

Concurrent processing using multiple processors is an advanced technique for achieving more processing power, faster computation, and flexible application of computing hardware to changing requirements.

ConCurrent C and FORTRAN were developed to provide high-level software development tools for the concurrent programming of true multiple instruction stream/multiple data stream (MIMD) computing environments such as the FLEX/32 MultiComputer.

REFERENCES
