COMPARING MICROCOMPUTER DEVELOPMENT SYSTEM CAPABILITIES

Hardware and software integration difficulties during microprocessor based system design mandate use of microcomputer development systems with comparative processor support capabilities and software, and differing control consoles, memories, and in-circuit emulation architectures

Bruce E. Gladstone Futuredata Computer Corporation, Los Angeles, California

Proliferation of microprocessor based products necessitated the design of microcomputer development systems. These tools represent either general purpose or universal investigative aids that support several microprocessor types, or dedicated aids restricted specifically to a single microprocessor type or family. A system assists the designer in evaluating alternative microcomputer hardware or software prototypes because it incorporates all standard computer development tools, such as the central processor, mass storage memory, control console, editor, assembler, and compiler. Also, simultaneously testing hardware and software produces powerful debugging capabilities (Fig 1).

Specialized in-circuit emulator types of microcomputer development systems evolved to handle problems inherent in separating hardware and software defects. These include the Futuredata Microemulator and Tektronix In-Prototype Emulator that support microprocessors from a variety of manufacturers, the Intel ICE and Motorola USE that support a family of microprocessors, and single-microprocessor systems from RCA, Rockwell, and Zilog. The in-circuit emulator provides the most accurate method for testing and checking microcomputer system hardware and software. It replaces the central processing unit (CPU) chip in the system under test, duplicates the chip functionally, and furnishes indications of system operation and control at a console or terminal. The ability to examine, change, or modify CPU registers, storage memory, and program execution permits rapid and easy hardware and software testing.

Hardware Elements

A microcomputer development system consists of a series of hardware and software elements. Major hardware elements are the central processor, memory, mass storage, and control console (Fig 2). The central processor executes the various algorithms involved in the development task and, at times, executes an editor, an assembler, a debugger, and the designer's program. Thus, it performs two major functions: host and designer program execution.

Memory stores both program and data. These programs can be host programs, such as the editor or compiler, or designer programs. Data involve editor workspace, assembler symbol table, input and output (I/O) buffers, and/or data generated by the designer's program.

Mass storage stores operating programs, designer's programs, and either temporary or permanent data. It is sometimes used as an extension of main memory. For example, when editing a large program, the editor workspace may not be large enough to contain the entire

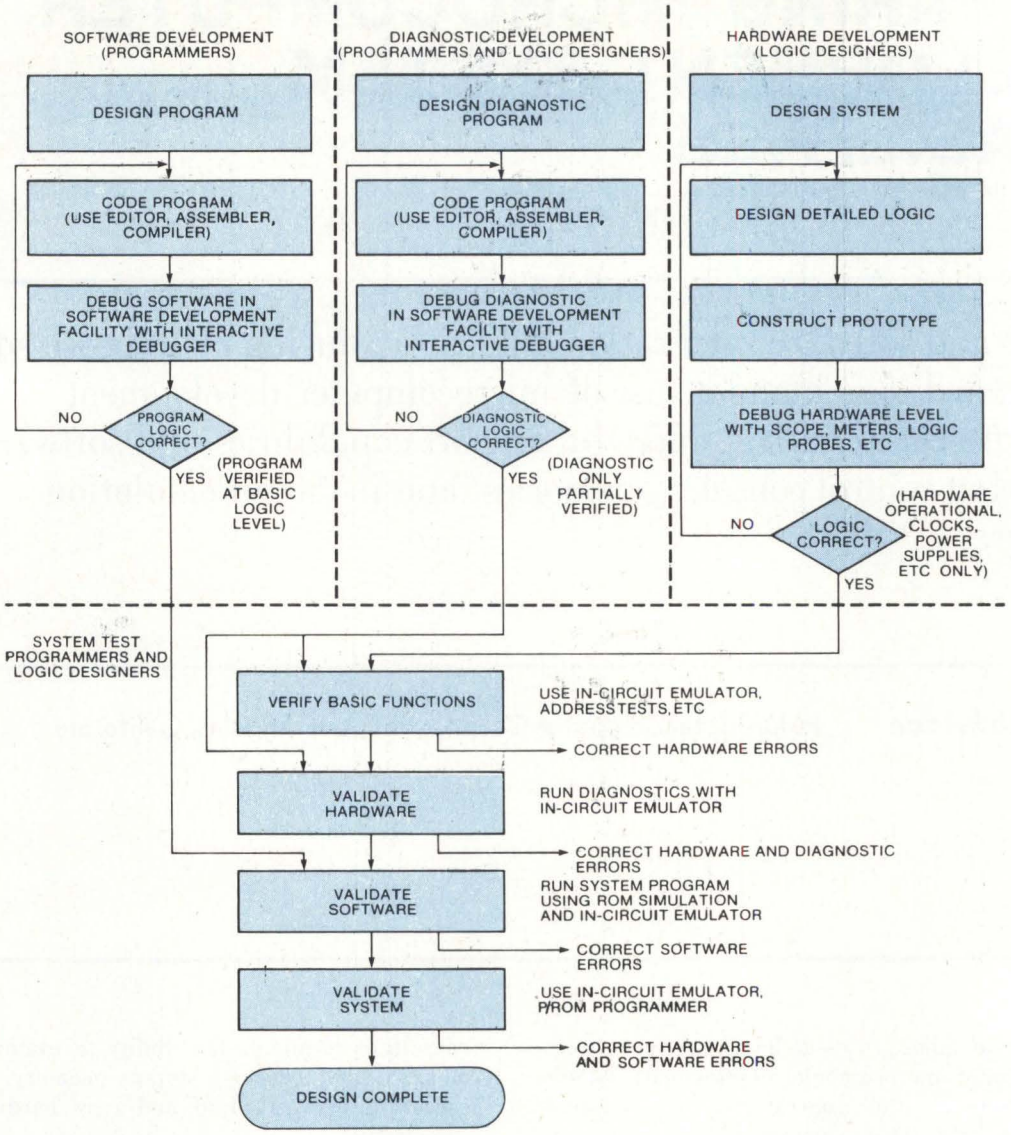

Fig 1 Microprocessor based system design with development system. Development of microcomputer based design involves preparation of application program, hardware design, diagnostic programming, and system test phase. Capability of development system is especially critical in latter, when it is necessary to sort out hardware versus software problems

program during edit. Moving the source program from one file in mass storage through the edit buffer to another file in mass storage accomplishes an edit. In this case, the two files and the editor work area comprise a very large memory space.

The control console provides an interface between the designer and the microcomputer development system. Systems differ greatly in the type of console used. Two primary functions of the console are to accept operator input (source programs, debugging commands, operating system commands, data) and to provide feedback and output to the operator (assembly listings, memory dumps, register displays).

Essentially, these four hardware elements also constitute a software development system, of which the Intellec 8 and Intellec 80 are examples. The next three hardware elements, however, relate to the hardware development aspect of microcomputer design and apply specifically to microcomputer development systems.

An in-circuit emulator provides a direct connection between the microcomputer development system and the prototype system. Read/write memory within the microcomputer development system simulates read-only memory (ROM) or programmable read-only memory (P/ROM) in the prototype system. This greatly reduces the time needed to change and correct designer programs. Some

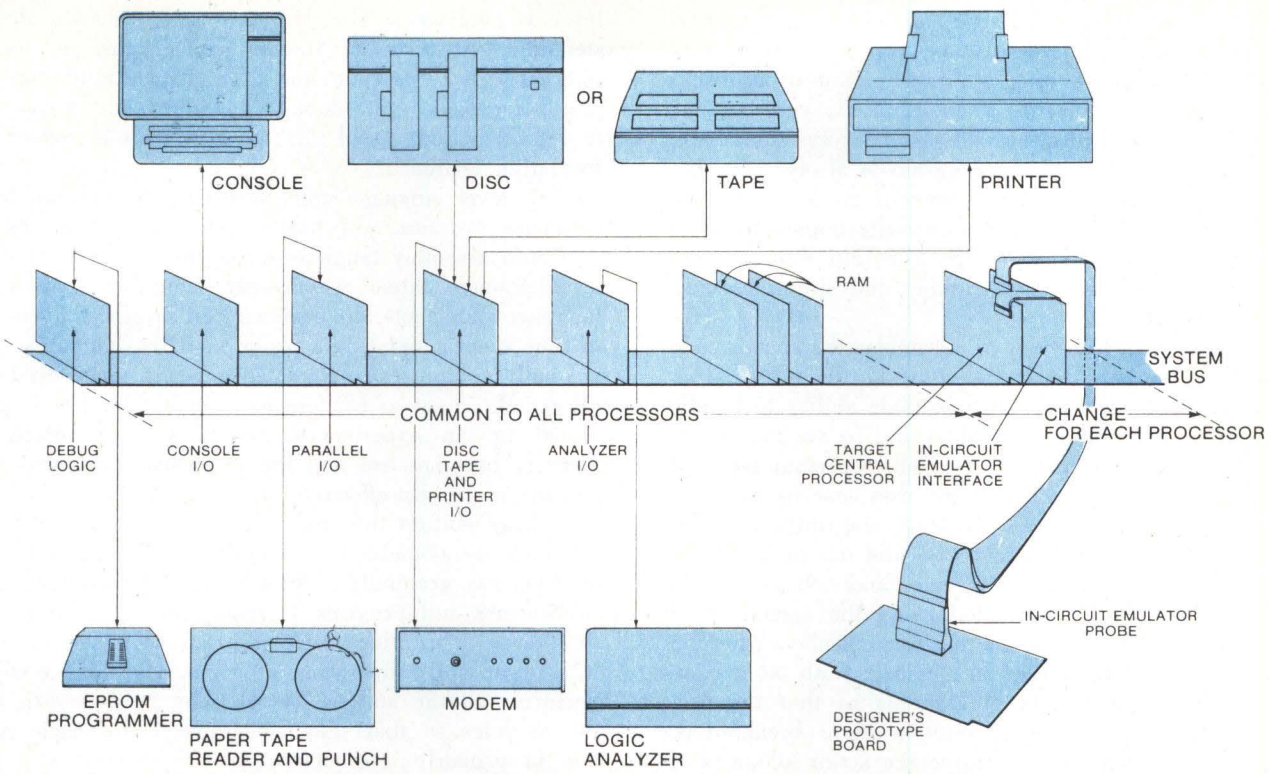

Fig 2 Relationship of universal development system hardware components. Components common to differing microprocessors include all I/O and memory devices. Only target central processor and in-circuit emulator interface are changed to accommodate specific microprocessor

in-circuit emulators allow the designer to gradually switch functions into the prototype system. Thus, the designer separately tests the clock circuits, input control lines, and direct memory access (DMA) control system to evolve the final design in discrete increments.

Final software can be stored in P/ROM using the P/ROM programmer. Programs debugged and running in simulator ROM are permanently burned into an electrically programmable ROM (EPROM) or P/ROM.

Hardware debug aids, used in the debugging process, separate hardware and software problems. These aids include a single-stepping facility, a hardware breakpoint facility, and a logic analyzer. With a single-step facility, the designer steps through the program one instruction at a time. A hardware breakpoint facility sets up a logical condition, usually an address, that halts the program whenever it encounters that address. Then, the designer can examine interim results, determine whether they are correct, and either continue executing the program or go back and modify it. In addition to providing hardware breakpoints, a logic analyzer captures data on the fly and displays it. Thus, the designer views the system's operation as a series of bus transactions. The logic analyzer has a fixed amount of storage that is continually updated by the last bus cycle, while the earliest bus cycle is erased. Therefore, when a hardware breakpoint is encountered, the logic analyzer memory shows events leading up to the breakpoint.

The microcomputer development system contains other I/O hardware elements. The printer provides hardcopy listings of the program, while other I/O devices, such as a paper tape reader, a punch, or a modem, provide a standardized interface into other systems for data interchange. Thus, with a paper tape reader and punch, a designer may write a program on one development system and communicate it to another development system. Likewise, with a modem, a designer connects a microcomputer development system by telephone lines to a large computer which may have other processing facilities available (cross-compilers, cross-assemblers, etc).

Software Elements

Integrated into a development system are an editor, assembler, debugger, high level language compilers, linkage editors, and operating system—all software components. The editor creates and modifies source programs, written in assembly language or in higher level languages, depending on the task. The editor must, of course, have editing commands to change, delete, and insert lines of code; positioning commands to target on a particular location in the program where changes are to be made; and utility commands to read and write edited data.

Editors differ in their capabilities and in their interactions with the control console. Most are written for

teletypewriter compatibility. To meet this design criterion, editors are designed with a small amount of information feedback. More recent designs use ultra high speed cathode-ray tube (CRT) displays to provide context based editing and to offer a large amount of information feedback. A context based editor is very similar to the editors used in CRT word processor systems; here, a significant amount of the program is always "on display." A cursor within the display or context performs the editing; all changes instantaneously update the program as displayed. This feedback within the program context substantially reduces errors and simplifies editing commands.

The assembler translates or assembles the source code (assembly language), generated using the editor, into object code. Assemblers differ in their ability to handle macros, to generate relocatable code, to support a variety of operand formats, and to allow certain types of pseudo-instructions. The designer uses macros to name commonly used sequences of instructions, which then are "called" with a single instruction—the macro name.

A debugger interactively executes and debugs the object program that is generated using the assembler or compiler. Commands handle memory display, program execution, and data storage in memory. With other commands, the designer sets breakpoints so that the program will halt when it encounters these breakpoints. Thus, the designer can determine the action of parts of the program. Additional commands set values in the processor registers, find data in memory, and read and write object program files. With the advent of in-circuit emulator capability, additional debugger commands map memory between the host and prototype systems, and switch the various microcomputer control lines between the host and prototype systems, thereby establishing the level of emulation. The debugger also provides single-stepping and program tracing. Since most debuggers interact with a teletypewriter, they provide minimum operator feedback. New debugger designs take advantage of the ultra high speed CRT displays with increased information feedback.

High level language compilers translate a high level language program (in BASIC, FORTRAN, COBOL, PL/M, etc) into assembly language programs. Since each high level language statement represents a number of assembly language statements, the designer can shorten the amount of time spent generating a program. This time advantage normally contains an associated expense, since the compiled code is not as efficient as assembly level code generated by an experienced programmer; however, as memory becomes less and less expensive, compilers will become more cost-effective.

Linkage editors link individual program modules, all of which are assembled or compiled separately. Thus, the designer can gradually build a library of commonly used subroutines and program segments and, as a last step, link these subroutines and program segments together to form an entire operating program. The linkage editor performs all the address calculations necessary to link the modules so that they can interface to each other and fit properly into memory.

Consisting of an editor, assembler, monitor, debugger, compilers, and linkage editors, the operating system manipulates, stores, retrieves, loads, and executes system programs, and creates and deletes data files. Various utility functions, such as the ability to copy programs

TABLE 1

Operator Console Types and Major Function Times

Operator Console	Application Example	Debug Function Time[1] (Overhead)	Edit Function Time[2]
Binary lamps and switches	Front panel; Imsai and Altair computers; DEC PDP-11	10 min	Not practical
Hexadecimal keyboard and display	Intel SDK-85; KIM-1	180 s	Not practical
Printer terminal with P/ROM debug monitor	Motorola EXORciser with Exbug, and TI Silent 700 terminal	120 s; includes time to write results; printing time is significant	480 s; must re-list to check edit
CRT terminal—120 to 960 char/s—and debug monitor	Intel MDS with Intel CRT terminal; Tektronix 8002 with CRT terminal	80 s	240 s; must re-list to check edit
Memory refreshed CRT display	Futuredata Microsystems 10, 15, 20, and 30	40 s; minimum operator entry	90 s; context editing

[1]Standard debug example: set breakpoint, execute, examine register and 32 bytes of memory, make a 5-instruction patch, re-execute, examine registers, and continue execution.
[2]Standard edit example: change two lines, delete three lines, and insert five lines. Assume that all data are in memory.

either in source or object form from one media to another, also are offered.

The mass storage device attached to the system, such as paper tape, magnetic tape, and disc, principally differentiate the various levels of the overall operating systems.

Operator Console Levels

Microcomputer development systems differ primarily in two hardware areas: the operator control console, and the type and speed of mass storage devices. Table 1 shows five levels of operator control consoles. Early operator consoles contained binary lamps and switches. Even today, most minicomputers have a front panel containing a row of toggle switches and a row of lamps. Manual loading of an initial program is common practice with these front panel switches. A bootstrap program inputs a more sophisticated loader through a paper tape reader and, finally, that loader program loads in the designer or system program, again from the teletypewriter paper tape reader. Interaction with an operating program for debugging can also be done with panel lamps and switches. Obviously, this manual debugging method is tedious, evolving, as a result, many modifications to operator consoles.

The first improvement added a printer terminal with debugger. Debuggers written around this terminal type provided a minimal response to each command since printer terminals were relatively slow (data rates of 10 to 30 char/s). Each command deliberately limits information; otherwise, the designer continually would be waiting for the terminal to finish printing the results of the last command. This in turn led to two further improvements: one in the direction of less cost (hexadecimal keyboard and display) and the other in the direction of higher performance (CRT terminal).

The hexadecimal keyboard and display are inexpensive methods of implementing an operator console. Information previously entered with switches and displayed with binary lamps is now entered with a hexadecimal keypad and shown on 7-segment light emitting diode displays. The keypad approach reduces operator entries from 8 or 16 switches to 2 or 4 keystrokes. This technique clearly minimizes operator errors but still does not provide much additional information. High speed CRT terminals, as implemented on most microcomputer development systems, merely act as high speed versions of a printer terminal. System software has not been updated to take full advantage of the CRT terminal speed; only the process is accelerated by generating output data at a greater rate.

The microcomputer development system designer uses the highest level of operator console, a memory refreshed CRT display, to take into account the ultra high speed capability of the console at design time. Thus, a context based debugger and a context based editor can be provided. In this type of console, the designer sees a continually updated register display, large memory dumps, and whole segments of source code in context, greatly reducing the confusion as to exactly what is happening. This type of interactive software is designed around consoles that operate with data rates between 10k and 50k char/s. The present state-of-the-art and direction of microcomputer development systems indicate that this level of operator console should become more prevalent in the future.

Mass Storage Levels

Small computers use four general levels of system mass storage. The first level is no mass storage facility. The designer keys in the program, usually in binary or hexadecimal, and then executes immediately. The next level is paper tape based systems available on many small computers. The mass storage medium is punched paper tape, and the combined operator console and mass storage device is usually a teletypewriter. This level of storage system was dominant for many years, largely because of excellent cost-performance characteristics. Only recently have CRT displays and magnetic tape or disc systems been able to compete effectively with paper tape systems. The third level of mass storage is low speed magnetic tape (generally, cassette). Data are read and written into cassette tapes somewhere between 30 and 400 char/s. This represents an improvement of 3 to 40 times over the teletypewriter and makes low cost microcomputer development systems practical. This level of storage system is perhaps a good choice for relatively small programs (500 to 1000 lines or less), a limited budget, or an initial implementation of a development system.

The fourth, generally accepted standard for mass storage on development systems is flexible disc, either a 5.25″ (13.34-cm) or an 8″ (20-cm) diameter version. Data transfer rates are greater than 1000 char/s. Access time and throughput for this mass storage device cease to be significant factors in development time. Comparisons of editing overhead times using teletypewriter, paper tape reader and punch, medium speed cassette tape, and fast flexible disc indicate that even for a 100-line program, a teletypewriter is slow. Likewise, for a 1000-line program, the medium speed cassette based system is probably too slow. However, with a flexible disc based system, the designer edits and assembles 10k-line programs in a relatively short time.

Development System Architectures

Architectural considerations involved in comparing microcomputer development systems relate to two basic implementations of in-circuit emulation—a master-slave approach, such as taken by Intel and Tektronix, and a single-processor approach, such as that of Motorola, Futuredata, and Zilog. Table 2 summarizes these architectures and implications.

Master-Slave System

In a typical master-slave system (Fig 3), the master (host) microprocessor runs all system software functions, including editing, assembling, disc file management, and "downline loading" of object programs to be tested in the prototype (target) microprocessor using in-circuit emulation. The target (slave) microprocessor is the designer's microprocessor under investigation. The development system manuafcturer primarily accrues the

TABLE 2

Development System Emulator Architectures and Associated Hardware/Software Devices

Manufacturer	Emulator Architecture	Micro-processors	Memory and Peripherals	Software	CRT Display
Futuredata (Microemulator)	Single processor and common memory; allows full speed emulation and complete control of emulation modes; universal	8080 8085 8086 6800 6802 Z80	Up to 64k Cassette Floppy disc EPROM programmer Logic analyzer Printer	Editor Assembler Debugger Macro Assembler Linker BASIC Interpreter BASIC Compiler	Memory refreshed
Intel (ICE)	Modified master-slave single memory; slows emulation with wait states. Allows control of emulation modes; nonuniversal	8048 8080 8085 8086	Up to 64k Paper tape Floppy disc EPROM programmer Logic analyzer Printer	Editor Assembler Debugger Macro Assembler Linker PL/M Compiler FORTRAN Compiler	RS-232-C Teletype-writer mode*
Motorola (USE)	Single processor with common memory; nonuniversal	6800 6802	Up to 64k Cassette Floppy disc Printer	Editor Assembler Debugger Macro Assembler Linker BASIC Interpreter FORTRAN Compiler	RS-232-C Teletype-writer mode
Tektronix (In-Prototype Emulator)	Master-slave architecture with split memory; permits full speed emulation and control of emulation modes; universal	8080 8085 6800 Z80 9900	16k Host, Up to 64k Target Floppy disc EPROM programmer Logic analyzer Printer	Editor Assembler Debugger Macro Assembler Linker	RS-232-C Teletype-writer mode
Zilog (In-Circuit Emulator)	Single processor with common memory; allows only minimal control of emulation modes; nonuniversal	Z80	Up to 64k Floppy disc Printer	Editor Assembler Debugger Macro Assembler Linker PL/Z Compiler	RS-232-C Teletype-writer mode

*Teletypewriter mode: CRT display is treated as a byte-serial device with data rates of 120 to 960 char/s.

advantages of such an approach. First, using a standardized host microprocessor minimizes software cost for implementing a new microprocessor. The second advantage is that less of the system resources need be reserved for the host system.

A disadvantage of the master-slave system is that it splits memory into two separate spaces. The system programmer encounters more difficulty in dealing with a split memory and its discontinuous address space than with a single large memory. A single memory space more effectively handles host functions that require large amounts of memory (editor work areas, and assembler and compiler symbol tables). Thus, 16k of host memory and 16k of prototype memory provide only a 150-line edit buffer. The same 32k memory in a single large device supports over 1500 lines of editor code (some of this advantage is due to packing the editor data). A second disadvantage is the higher cost that additional memory and microprocessors add to the system.

A third and rather subtle disadvantage relates to modification of the system programs. Most designers of computer systems realize that over a long period of time the supplied operating software will be modified. This modification may add a new I/O peripheral device, for example, printer or paper tape reader or punch, 7-track or 9-track tape drive, hard disc drive, or a number of devices not included in the original system. With the master-slave approach, the designer must modify pro-

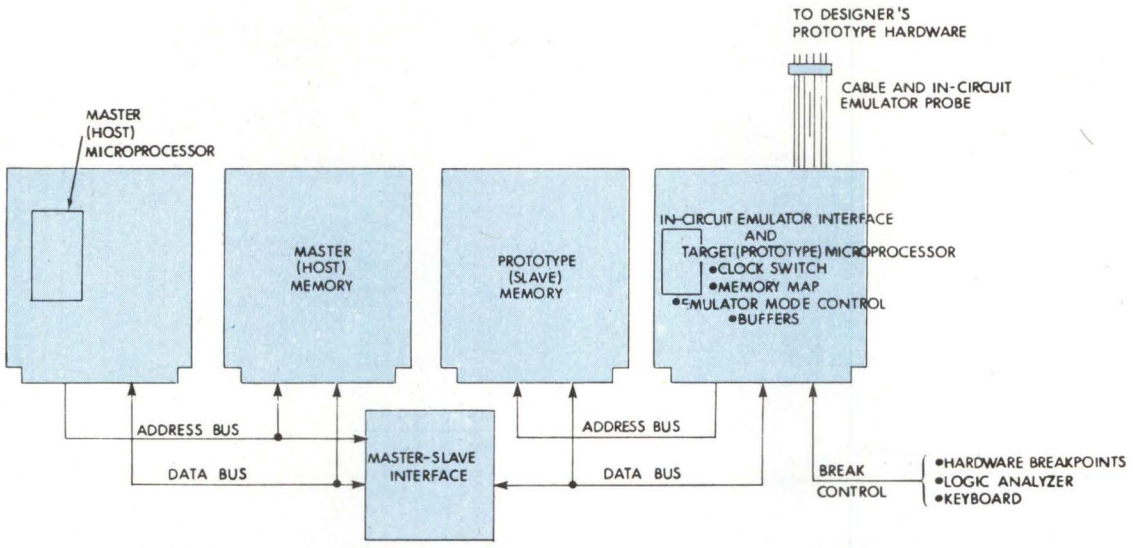

Fig 3 Master-slave in-circuit emulator architecture. System and prototype functions are divided between master (host) and slave (target) microprocessors, respectively. System development functions, such as file management, text editing, host I/O, and debugging, are performed by master microprocessor. Prototype related functions, such as prototype program execution, prototype I/O, and in-prototype testing, are done by slave microprocessor

grams written for an unfamiliar host microprocessor. Thus, an 8080 or 6800 designer might be confronted with the necessity of modifying code written for a 2650 microprocessor.

The Microemulator II systems permit both common and separated memories to be used, depending on designer requirements. A master-slave processor approach is used, but all memory is contiguous in the host system. The slave microprocessor is remote from the system (at the emulator plug), and none of its resources need to be reserved for the system. Software systems written in their native languages support the 8080, Z80, and 6800 families of microprocessors.

Single-Processor System

In a single-processor system (Fig 4), system software usually is written on the target microprocessor. Thus, a 6800 designer would modify 6800 code, an 8080 designer would modify 8080 code, and a Z80 designer would modify Z80 code. This type of development system can become a multipurpose device. After becoming familiar with the system hardware and software, many designers may use that same hardware in the development of test equipment that will follow the design through its production life.

The clear cut advantage of a single-processor system is low cost, since much less hardware is included. A second advantage is that one large memory is furnished; thus, a 32k memory space provides for 26k of object code in the prototype system, and an assembler symbol table with many thousands of symbols. The Intel ICE system, while a master-slave system, does provide for one large memory accessed by both microprocessors. To do this, the system delays the target microprocessor when it accesses development system memory. However, this compromises the ability to totally emulate prototype system operation. Another advantage of the single-processor system is the use of development system hardware and software as part of in-plant test equipment.

A disadvantage of a single-processor system is that the manufacturer must entirely rewrite software packages for each new microprocessor. Standardization trends in the industry make this less of a problem. A second disadvantage, depending on the application, is that emulation is not entirely separate. Thus, most single-processor systems reserve part of the memory space for system programs that must be resident during emulation. They generally have at least one privileged I/O address to switch the system in and out of emulation mode. Depending on the design of the single-processor system, it also may use some DMA capabilities and an interrupt

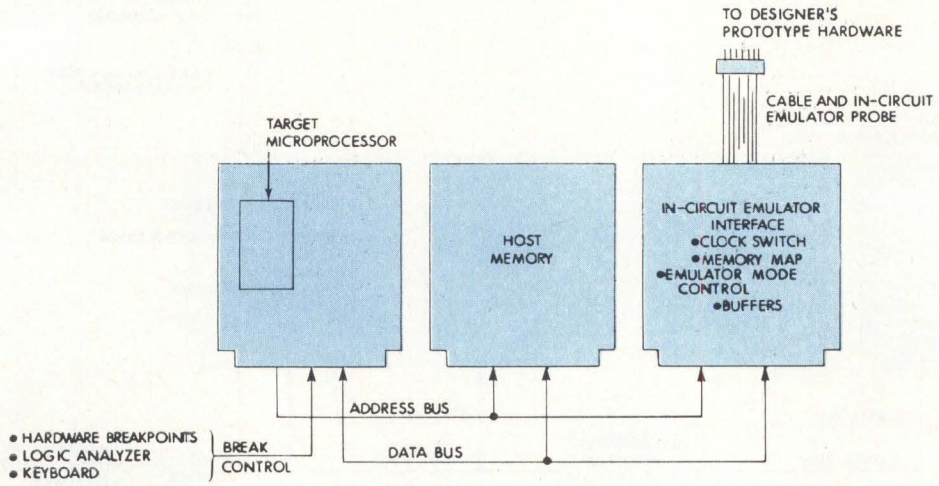

Fig 4 Single-processor in-circuit emulator architecture. Dedicated system development functions are programmed in related instructions for direct execution by target microprocessor. Differing target microprocessors require associated system software modifications

structure. The designer can upgrade the emulation capabilities of the system as needed to support development.

Summary

Microcomputer development systems have common capabilities of processor, memory, console, mass storage, in-circuit emulation, and system software. They differ in type of operator console, mass storage device, in-circuit emulation architecture, high level language support, and whether or not they are universal (support a wide range of microprocessors from a variety of manufacturers).

Development system support for the 8080 and 6800 and their successor microprocessors is necessary. These two microprocessors are standards to some extent, because of significant investment in 8080 and 6800 software. The 8080 standard includes as its successors the 8085, the 8086, and the Z80. The 6800 standard includes as its successors the 6802, 6809, and 6502. In general, these successor microprocessors, with the exception of the 6502, have attempted to remain program compatible with earlier versions. Both microprocessor manufacturers and designers who invested substantial amounts of money in programs to support these microprocessors require this compatibility.

With some 20 manufacturers, most of whom can introduce one or more microprocessors per year, a universal development system becomes an essential tool for designers. Since the 8080 is 10 times more powerful than the 8008, the Z80 is 2 to 3 times more powerful than the 8080, and the Z8000 and 8086 are 5 to 10 times more powerful than the Z80, the pressure to adopt new microprocessors is intense. Thus, the designer must be able to switch the target microprocessor without starting from scratch.

Bibliography

D. A. Cassell and J. E. Cavanaugh, "The Microcomputer Development System," *Mini-Micro Systems*, Aug 1977, pp 34-40

B. Gladstone and P. D. Page, "Programming Hints Ease Use of Familiar Microprocessor," *Computer Design*, Aug 76, pp 77-83

P. Snigier, "Microprocessor development systems—which one is best?" *EDN*, Mar 5, 1977, pp 68-78

Bruce Gladstone, vice president of Futuredata Computer Corp, has expertise in hardware and software aspects of microcomputer systems design, as well as electronic systems design. He has been responsible for Microemulators™, EPROM programmers, and Microanalyzer™ designs, and development system software and debuggers. He holds an MS in engineering from UCLA.