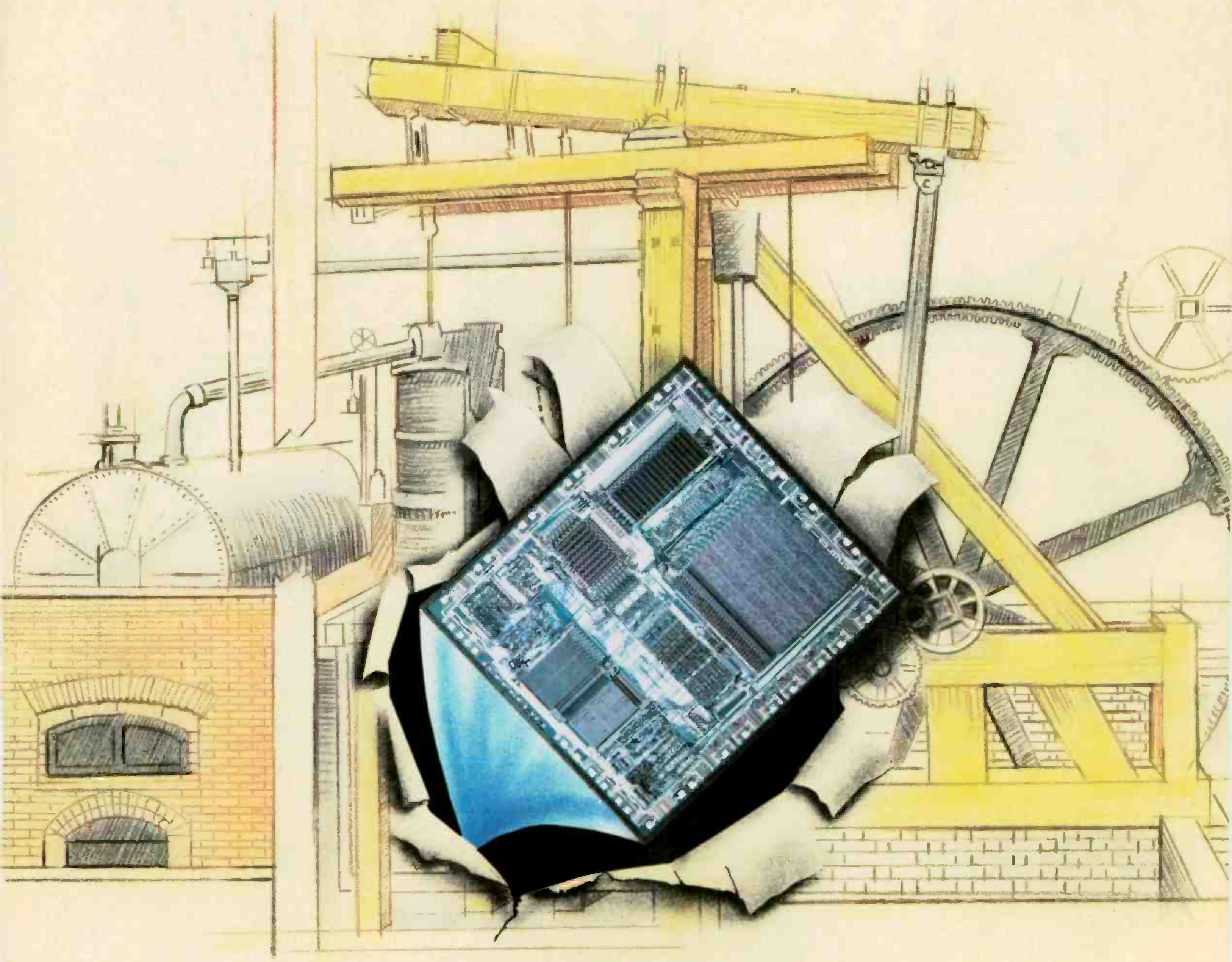
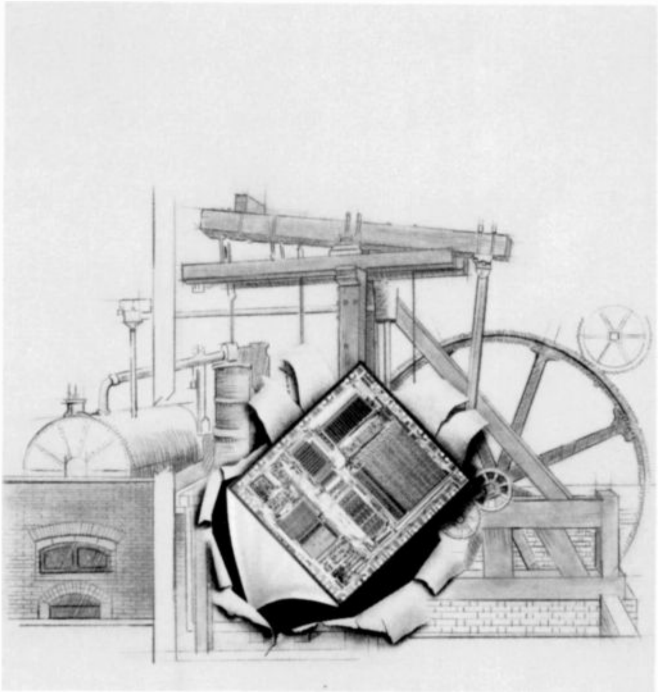
RCA Engineer

Vol. 26 No. 8 Sept./Oct. 1981



Technology Breakthrough: The Microprocessor Revolution

Cover design by Louise Carr and Mike Sweeny
Cover illustration by Michael Katz of NBC News Graphics



Our cover shows the RCA 1804 microprocessor breaking through an old-style rendering of the original Watt steam engine. Because of the power of the engine and the mechanical improvements that accompanied its development, people devised production methods that drastically altered the shape of societies in the 1800s, leading to a cluster of technological breakthroughs and applications known as the Industrial Revolution.

Since that time, many new technologies have helped or affected us in "revolutionary," though somewhat limited, ways. But microelectronics will affect such a broad range of activities that advanced industrial societies will need to organize their production systems around electronics in order to stay competitive.

Already, sales of electronic goods are well over $100 billion a year. By 1990, the market may reach about $400 billion a year, and the economy will be global. Microprocessor technology is so sophisticated and laden with potential applications that it could be characterized as the catalyst for a "Second Industrial Revolution." This technological breakthrough may relegate the idea of mechanical control to little more than a torn page in an old book.

American companies, currently accounting for more than 70 percent of the world's production of integrated circuits, will need to develop and maintain their competitive edge. This issue of the *RCA Engineer* samples a variety of microprocessor applications spanning several RCA businesses.

Our cover design is an extension of an idea by Bob Mausler, the *RCA Engineer*'s Technical Publications Administrator at NBC. He is part of the network of talented, dedicated publications people and engineers throughout RCA who invaluably assist in planning and recruiting articles, and producing the issues accurately and on time.

—MRS

RCA Engineer

A technical journal published by
RCA Research and Engineering
Bldg. 204-2
Cherry Hill, NJ 08358
TACNET: 222-4254 (609-338-4254)

•To disseminate to RCA engineers technical information of professional value •To publish in an appropriate manner important technical developments at RCA, and the role of the engineer •To serve as a medium of interchange of technical information between various groups at RCA •To create a community of engineering interest within the company by stressing the interrelated nature of all technical contributions •To help publicize engineering achievements in a manner that will promote the interests and reputation of RCA in the engineering field •To provide a convenient means by which the RCA engineer may review his professional work before associates and engineering management •To announce outstanding and unusual achievements of RCA engineers in a manner most likely to enhance their prestige and professional status.

C.J. Santoro

Microprocessors: A Revolution Put into Practice

As investment and ingenuity have driven the cost of the microprocessor down from the hundreds of dollars of the mid-1970s to the several dollars of today, this technological breakthrough in computing capability has found itself in the hands of greater numbers of creative engineers and designers. As a result, a multitude of new electronic systems now work for us every day — from uses in entertainment to protection, from control to display functions.

A broad spectrum of architectures and a wide selection of word lengths have made available a once-only-dreamed-of computing capability at a low cost. Microprocessor systems are even being used to generate and test newer microprocessors. Communications, data processing, transportation, and defense systems' features are enhanced by microprocessors. Few days pass when we fail to come in personal contact with a microprocessor of some kind. The externals of these systems are so simple to use and so easy to understand that we tend to forget the complexity of the subcomponents within them.

And yet, we are less than ten years into "the microprocessor revolution." Chips have integration levels of less than 100,000 transistors and we still have not reached an ultimate cost level. We can be assured that the fertile engineering minds that have led us this far will continue to extend the capabilities of these chips.

A key to expansion is the the application of the "micros" — today's and tomorrow's. Most of today's systems use substantially less than the full capabilities of the existing microprocessors, and as we begin to more and more fully use them, we will see an even broader spectrum of electronic services with truly astounding sophistication.

At RCA we have been a leader in this microprocessor revolution and we will continue to establish new directions. We have talent in every aspect of this technology, from chip design to system manufacture. By pooling our creative energies, we can continue to find applications that will transform our world. This issue features a selection of applications for microprocessors. These are but a small sample of the multitude of exciting ideas that clearly show how microprocessors can exert an ever-increasing influence on our lives.

Carm Santoro
Division Vice-President, Integrated Circuits,
Solid State Division

RCA Engineer

Vol. 26|No. 8 Sept.|Oct. 1981

microprocessor applications overview

the 1800-series microprocessors

microprocessor test sets

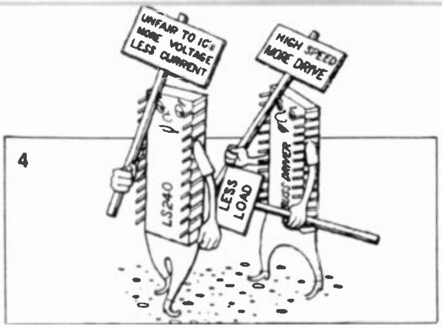
microprocessor control applications

distributed microprocessor systems

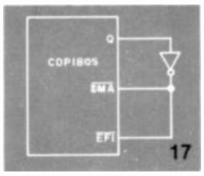
departments

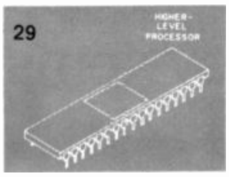
in this issue...

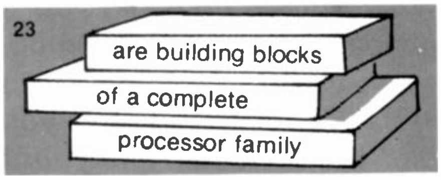
Microprocessor applications and development


4

■ **Caracappa|Ny|Ripley** *"When developing small microcomputer systems, you often do not need the power of the latest development system with all its options."*


17


29

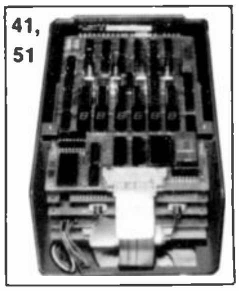

23

■ **Paradise** *"The RCA CDP1800 microprocessor series is a well-tried LSI product line that implements control-oriented microcomputer functions in CMOS."*

■ **Paradise** *"These types of components and their functions more closely match equivalent NMOS offerings found in recent catalogs."*

■ **Wright** *"If a higher-level program language could be designed — easy to learn and not requiring a high-cost development system to use — many new microprocessor applications would open up and others would dramatically drop in cost."*

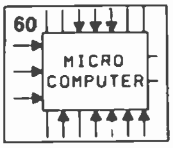
36


41, 51

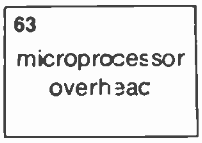
■ **Milley|Resnick** *"EPROM technology is attractive in this application because it is cost competitive with ROM."*

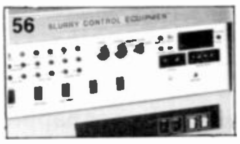
■ **Heppl|Isham** *"A system of this type provides a clean break between the logic that performs the test and the logic that decides which test to perform."*

■ **Schilp** *"The purpose of the project described was to build a piece of equipment to demonstrate the use of RCA Microboards in specialized manufacturing-test equipment."*


60

66


63


56

73

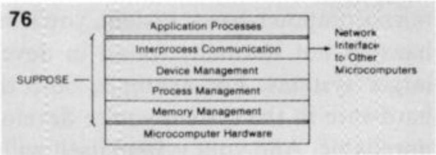
■ **Woestman** *"The quantity of each of these three liquids added to the charge vessel at each dispense is determined and controlled by the microprocessor."*

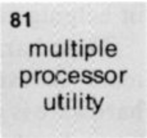
■ **Alvero** *"A new microprocessor-based lighthouse control needed to assure that the panel was exposed equally about the optical axis."*

■ **Malyszka** *"Satcom D will be the first RCA communications satellite to employ a microcomputer for fully automatic thruster control."*

■ **Chin|Kaye** *"As implemented in the REMBASS repeater, the microprocessor provides most, but not all, of the control and processing functions."*

■ **Papke** *"With the subsequent growth of the A/V industry in both size and sophistication, the manufacturers have turned to programmable, non-dedicated microprocessors."*


76


81

■ **Britton|Stickel** *"With SUPPOSE, an application is organized around shared resources."*

■ **Horsley|Clapper** *"Further upgrading of the signal processor was a prominent part of the plan, specifically in terms of application of VLSI technology and distributed microprocessors."*

■ **Ricker** *"The microprocessor was seen as the ideal host computer for the architecture — distributed single process nodes — that has been identified."*

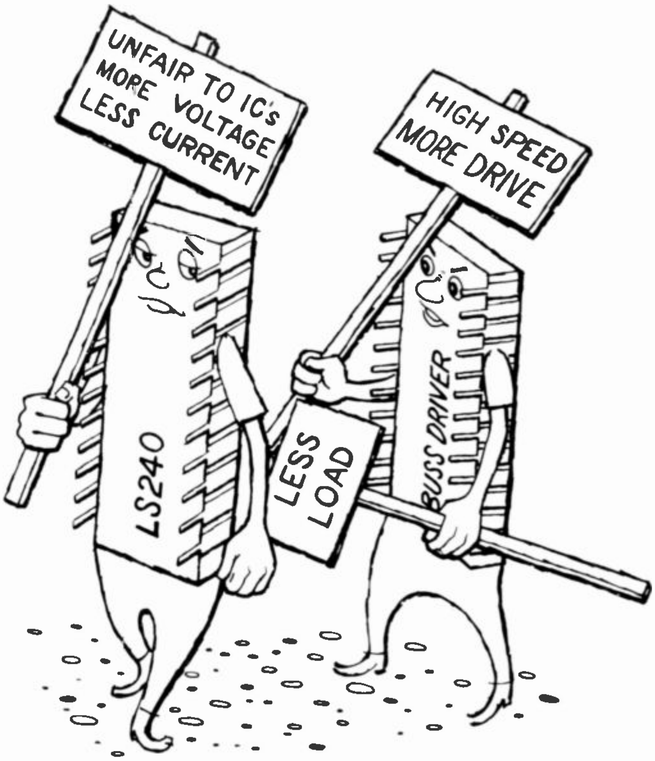
in future issues...

SelectaVision® VideoDisc, manufacturing engineering, electro-optics

D.N. Caracappa|N.O. Ny|G.D. Ripley

Designing and building microcomputer-based systems

What do you do when your program works... almost all the time? How do you handle it when your constants aren't and your variables don't? What do you do when your pull-ups are down and your bus drivers are on strike?



Do you have those days when your bus drivers appear to be on strike and your pull-ups don't?

Abstract: *This tutorial on microprocessor software development begins with a discussion of overall system design concerns, then addresses the special advantages and disadvantages of working in a minimum-resource environment. Next, the authors cover typical medium-resource tools, MDS and other resident systems, and the development environment when you have these tools. Finally, mini-sized systems and non-resident tools provide a large-resource development environment that the authors examine. The conclusion shows future trends in development hardware and software.*

Has this ever happened to you? You've worked all week to finish a phase in the development of a microcomputer-based system. On Monday, after a weekend of celebration, you begin the next step and find that the functions that worked so well on Friday are now totally inoperative. And all you did was turn the system on.

Developing any computer-based system can be frustrating indeed. But if you're considering building a microcomputer-based system, you'll be facing some special hazards not normally found in developing software for larger systems. For example, both the software and the hardware in the system you're developing will initially be unreliable. And your system itself will prove nearly useless in helping you put together the software.

These hazards are not found in developing software for maxi- or even minicomputer-based systems, because the hardware is generally reliable—any bugs are automatically isolated to the software. And the computer itself provides many tools to help develop the software, from language translators to debuggers.

Reprint RE-26-8-1
Final manuscript received July 6, 1981.

We'll give some tips on how you can deal with these and other problems in developing microcomputer-based systems, answering questions such as:

□ What are some good analysis and design tools to help me simplify the task?

□ Suppose I can't afford the best development tools money can buy—what then?

□ How do I decide what to implement in hardware, and what in software?

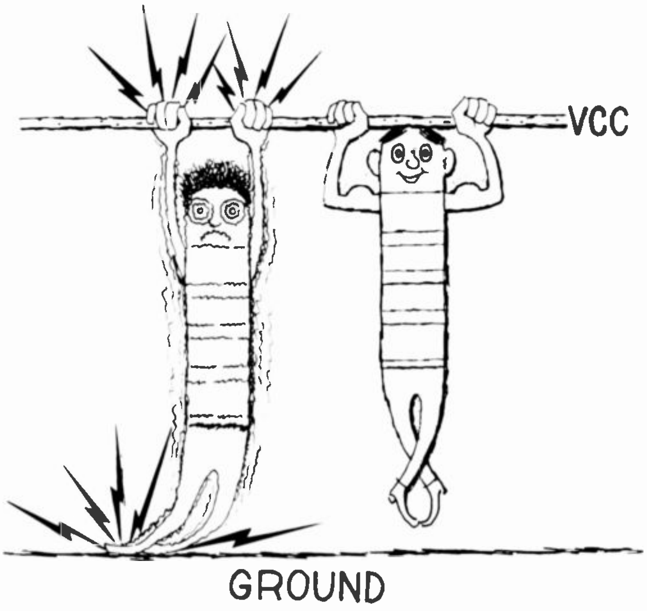
□ You've sold me—but now how do I obtain these wonderful tools and the expertise needed to use them in building my system?

First we'll review the all-important early stages of microcomputer-based system development: analyzing and designing the system. Then, we'll focus on implementation tools for a range of development environments, from "bare bones" to "large resources." We conclude the guide by looking at trade-offs and other pragmatic considerations. You may want to keep the myriad of tools straight in your mind by referring to the glossary on page 15.

Analyzing and designing a microcomputer-based system

Three key phases make up the development of any system: analysis, design, and implementation. Each stage feeds vital information into the succeeding stage, information that bears directly on the ultimate outcome of the project (Fig. 1).

An analysis of the problem identifies the inputs to the system and the outputs the system should generate in response to these inputs. In addition, analysis should resolve such questions as: when must the system be



All cartoons were drawn by Darwood F. Taylor, RCA Labs, Princeton

finished? at what cost? and who is available to work on the project? The information developed during this stage is then spelled out in such documents as data-flow diagrams (Fig. 1 is an example) and data dictionaries. While we'll only mention a number of these documents in passing, Reference 1 describes them in more detail, with examples and annotated references to "how-to" books on the subject. The key point here is that analysis is sometimes not recognized as a formal step in system development. But by paying more attention at this stage, you'll usually save a bundle of trouble later in the project.

A system designer must determine how to apply available resources to the hardware and software requirements in order to arrive at a solution. The choices that face the designer are not always obvious, and making a particular choice sometimes increases the risk of project failure. As an example, at some point the designer must choose to do a particular function in hardware or software. A decision to use off-the-shelf hardware components will mean higher product cost. On the other hand, the decision to develop (custom) software may increase the risk of slipping the schedule. Clearly, the life of a system designer is far from easy!

What aids will help with the design task?

One aid is a system block diagram, which shows the different subfunctions and how they are connected. Another is a diagram showing the interface between the microcomputer and the rest of the system: how the input data arrives at the micro, and what outputs from the micro go where. This interface diagram reflects some of the trade-offs that have been made between hardware and software.

So, now that you've analyzed and designed the system, it's time to start building it, right? Almost, but not quite. It pays to first plan how you'll go about testing the system once it's put together. You need to plan tests for the step-by-step integration of hardware and software, and to develop a final acceptance test that checks that the system satisfies the original requirements. If testing is not planned now, while requirements are still fresh in mind and you're not yet in the heat of the test-debug-modify battle, testing is likely to be inadequate, and reliability of the resulting system will suffer.

On the other hand, with these documents — the test plan and acceptance test — hardware and software development can take place somewhat independently and concurrently, with a fair chance of successful integration of these two portions when the time comes.

Is it really that simple?

Unfortunately, as you've probably suspected, issues involved in analysis, design, and implementation are often complex and interrelated. You'll find that you'll return to the design and even the analysis stages as problems and changes arise. For example, maybe the hardware could be much cheaper if a timing spec is loosened. The system designer's job is to determine the ramifications of such a change to ensure that the system will still work.

Before we move on to the implementation stage, note that tools are no substitute for a common-sense approach to designing a system. For example, in the analysis stage, sit down with the ultimate user to determine the real requirements. A visit to the installation site can sometimes

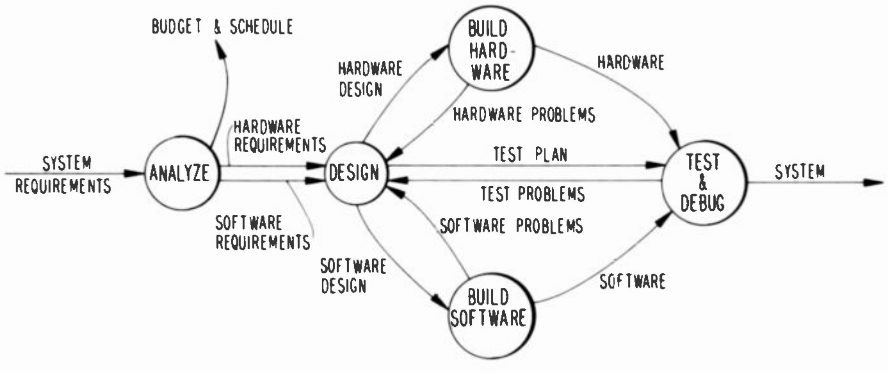
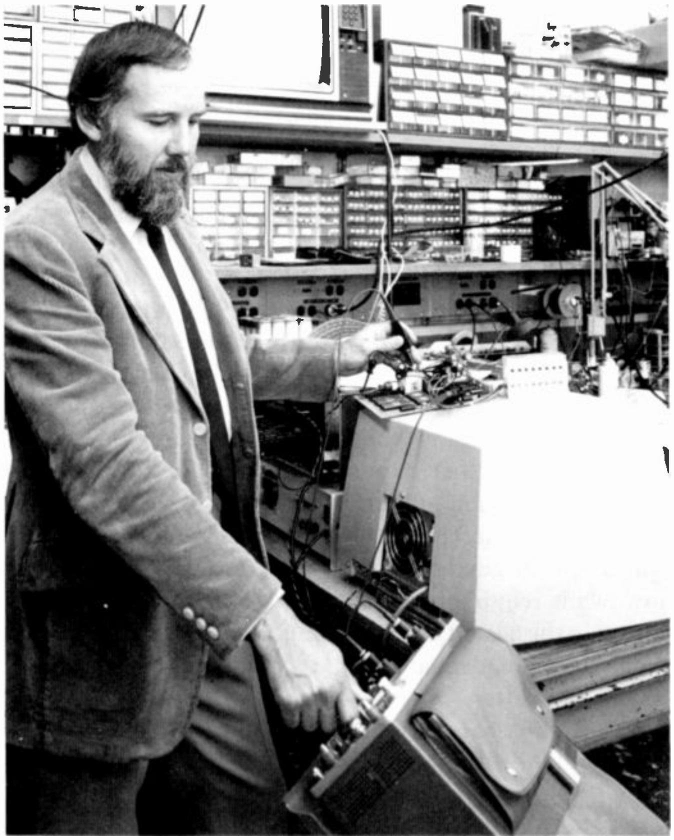


Fig. 1. System development process. This data-flow diagram shows the three key phases in the process — analysis, design, and implementation.



Author Ny debugging a microcomputer-based system in a "bare-bones" environment. Scope is in the foreground.

be helpful. If previous or existing solutions are available, an examination of these can sometimes give further insight into the problem. The best tool available for designing a system is thought. Clear-minded, unbiased, unbridled thought (maybe even blue-sky rambling) can sometimes lead to a unique and clever approach to the design.

Pictures — block diagrams, data-flow diagrams, schematics — will give many insights into the analysis and design process since they usually present a lot of information in small, easily understood packages. In short — think, draw lots of pictures, and communicate with users and other project members.

Implementing in a "bare-bones" environment

You are faced with implementing a small microprocessor-based control system, or you simply want to build a small system to educate yourself about microprocessors in general. You go to your manager and subtly suggest that you need $15,000 for a minimal microcomputer development system. Using that special tact which is bestowed only on managers, he counters with something like, "As you know, this year's budget is very tight. Perhaps we can use the company time-sharing system and if things go well, we might be able to get a new oscilloscope next year." And you hadn't even gotten to the part about needing a $10,000 logic analyzer yet! Does this sound familiar? Do you need to implement a small microprocessor system without the desired resources? Don't despair! This section describes what we've dubbed the "bare-bones" development environment.

First of all, note that we used the word "desired" rather than "necessary" resources. When developing small microcomputer systems, you often do not need the power of the latest development system with all its options. You need not implement your small control programs in Ada. If you are building a small microprocessor-based system for your own education, you can assemble your program by hand. Hand assembly is a means to an end that also gives you valuable insight on the inner workings of an assembler (for example, how it handles the symbol table, or why it needs two passes). You'll be putting the assembled code in an EPROM (or else a large pile of PROMs), so you'll need some means of burning the code into the EPROM. One of the ultraviolet EPROMs, like a 2708 or 2716, will do very nicely. You'll also need access to an EPROM eraser.

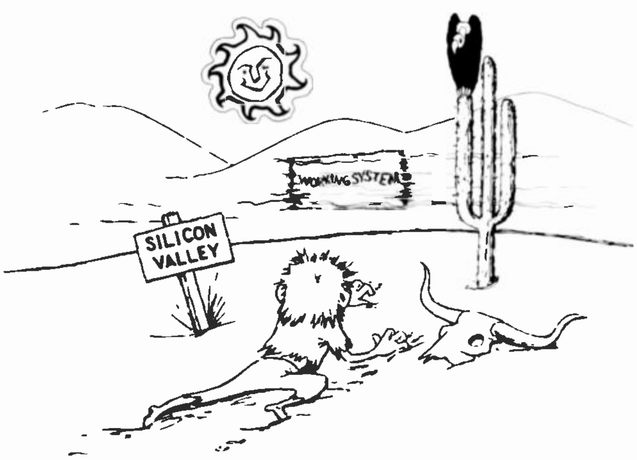
On the other hand, if your task is project-related, you will need some additional support, even to implement only a small system. First, you will need access to an assembler. Here almost anything will do, for example, a resident assembler on someone else's system, or a cross-assembler on a larger time-sharing system. The other piece of equipment needed is a decent oscilloscope. Although additional available resources would be convenient, you can implement even medium-complexity microprocessor systems using nothing more than these tools.

Can I really debug without sophisticated tools?

Yes — by using the system you are building as both your hardware and software debugging tool. Just be sure to avoid building all the hardware and then trying to figure out why it doesn't work. Trying to debug fully-built hardware with a few hundred connections is not a job for a single-trace scope.

Instead, build a small piece at a time. Start with the clock circuitry and convince yourself that it works correctly. Then connect the microprocessor chip and the EPROM chip. This usually requires only that you connect the address and data lines and a few control lines with some small-scale integrated circuitry. "How can I check that this is working since I don't have any input or output?" you ask. The idea here is to keep it simple. At this point, your whole system consists of only three or four chips with perhaps 20 or 30 connections. Keep the test software simple, since this is the only time that you have no working system to help debug the rest of it. A short program that branches someplace and then branches back again is all you need. You can then sync the scope on one address line (unique to one of the locations) and look at the address lines to see if you are branching between the two locations.

After you have this minimal configuration working, you now have a valuable tool to help you debug the rest of the system. You now add the RAM chips and the I/O ports to your system. Again, do it a step at a time. Add just the RAM and write a short test program to see if you can read



Attempting more complex systems design in the "bare-bones" environment may be more than you can handle.

and write to memory before you add the I/O. What is needed in these test programs is pure simplicity to avoid programming errors. In general, you can read from a non-existent memory location in order to provide the scope with a strobe. Then, follow with a simple loop (for RAM, just write a constant to memory followed by a read from the same location, then repeat). Now, if things don't work, it becomes a relatively simple matter to trigger the scope and look, one at a time, at the data lines, the write line, and so on, to see what is not working.

Keep this process going, one block at a time, until you have a good-sized system working (perhaps 20 to 50 chips with many hundreds of connections). If at any time in the process the hardware does not work with the previous test program, you have only the handful of connections to the additional chip to worry about.

How about debugging the software?

At this point you are probably one step ahead of us, and have concluded that you should use a similar step-at-a-time approach to the software development. You are absolutely right! To debug the software, one very useful piece of hardware is a box with eight switches tied to an input port and eight LED lights tied to an output port. Now have your suspect program perform a single task and output some intermediate result to the lights, so that you can see the state of affairs. If things are not what you expected, then you have only a few lines of code to look at carefully. If it does work, you simply move your test down past the next task in the program. As with the hardware, the basic theme is to keep things simple (even though your overall software may be quite complicated).

As an example of a system developed in a "bare-bones" environment, one of the authors recently developed a Z80-based controller for a future consumer product. The Zilog Z80 processor used 4 kbytes of EPROM and 1 kbyte of RAM (1k = 1024). The system included two programmable I/O chips, a programmable counter/timer, interrupt controller, and a handful of miscellaneous low-level chips. The author used a scope, an assembler on an MDS (see the next section), and an EPROM burner. Excluding the (borrowed) MDS, this development environment cost less than $5,000.

Many diehards who have been playing with microprocessors from the beginning have built entire systems without an assembler or even a scope. System development in the "bare-bones" environment, although feasible when you have nothing better to work with, has some disadvantages. It can become extremely frustrating. Furthermore, system implementation can require as much as an order of magnitude more time. It's not that one cannot develop systems this way, but rather that it becomes much more productive — and pleasant — with better tools.

Implementing in a medium-sized environment

As in any other discipline, better tools will let you spend more time fighting the real problem, and less time fighting your equipment. What are these wonderful tools that will free you from a life of drudgery? In this section we'll mention some of the key development aids.

Can existing tools in the Corporation help?

Yes. For example, simulation programs for electronic circuit design are available on RCA's IBM 370 system in Cherry Hill. Two of these programs are RCAP (RCA Circuit Analysis Program), for use in analyzing analog circuits, and MIMIC, a circuit analysis program for analyzing digital circuits. These tools are accessible by merely opening an account with the people in Cherry Hill.

Other valuable sources of development assistance are all those manufacturers out there who are so eager for you to use their products. A well-placed call to the manufacturer of an IC or piece of equipment you plan to use may bring you an applications note, or if you are lucky, access to a good applications engineer.

Several techniques exist for implementing a piece of the hardware system. You can put together an easily modifiable breadboard, with quick-connect tools or even services which can wire-wrap as many boards as you want, given a wire-connect list. Alternatively, a printed circuit board approach can be useful when the system must be replicated many times. There are also computers that aid in laying out PC boards (such as an Applicon system).

To test the hardware once it's built, you'll generally have to rely on your wits. However, a few tools can make the job of testing the digital/microprocessor components a lot easier. These include: a good multitrace, wide-bandwidth oscilloscope; a logic analyzer; and a function generator. As you know, an oscilloscope records one or more analog signals for some period after it receives a trigger signal and displays that information as a graph on a CRT. A logic analyzer is basically an oscilloscope for digital signals. It records digital words (usually 8 to 32 bits) and saves some number of them when it receives a trigger word, and displays them in either numeric or graphical form on a CRT. A function generator might be useful for stimulating the inputs to the system with a known waveform.

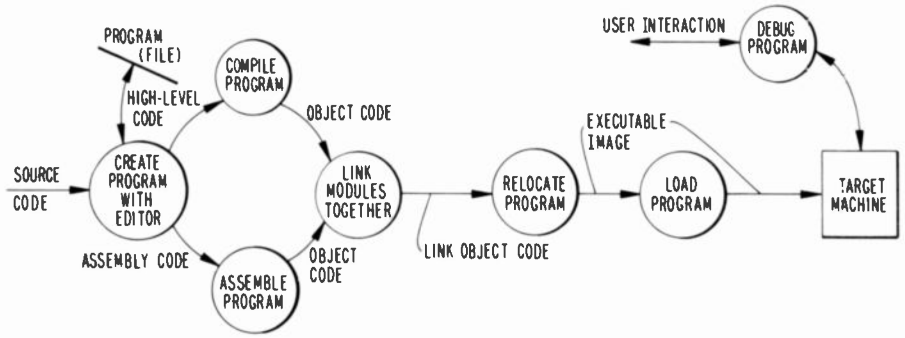
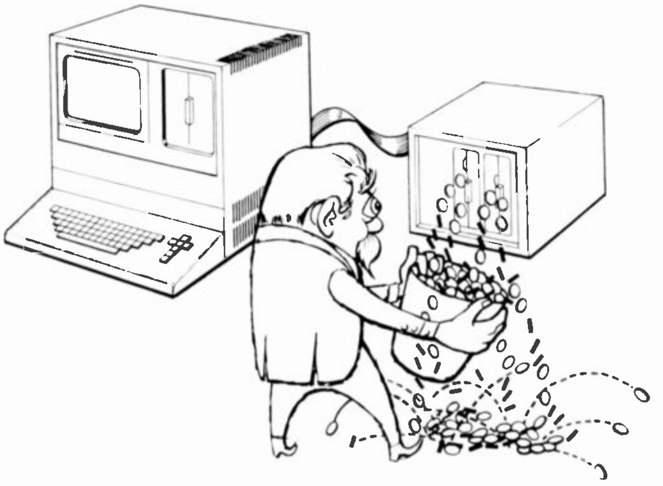


Fig. 2. Program development in a medium- or large-resource environment.

So much for the hardware. But what about the software? In fact, it is software tools that will make the biggest difference to your efficiency. We noted in the "bare-bones" environment the value of an assembler over hand-assembling your program. This is but one of many useful software tools. Figure 2 shows the different steps in developing software in a medium-sized (and large-resource) environment. The details follow.

How can a higher-level programming language help?

Certainly the most important tool in software development is a high-level language such as BASIC, FORTRAN, or Pascal. Programs written in these languages are generally much shorter and easier to understand, debug, and modify than those written in assembly language. A compiler is then used to translate these programs into machine code, corresponding to the assembler used in translating assembler programs. For those programs complex enough to be broken into separate modules, their compiled and assembled code modules must then be linked together into a single module, using a linker, and that module must be relocated to the memory location at which it will reside during execution.



Stand-alone Microprocessor Development Systems may not have adequate resources to accommodate the more complex projects.

Finally a program, whether originating as one module or a collection of modules, is loaded into memory using a "loader" (often a linker also does the job of the relocater and loader, combining the three processes into one program).

Where do these software tools live?

Compilers, assemblers, linkers, relocators, and loaders are all computer programs, and must have a computer on which to run. A computer with these facilities, which can support a single engineer, is called a microcomputer development system, or MDS. If program preparation has all been done on such a separate (host) computer, the process of placing the program into the (target) computer's memory is called downloading. On occasion you'll want to send data from the target to the host, using what is called an uploader.

An MDS also provides an editor for entering your program into a file prior to compilation or assembly. Since much of your time will be spent interacting with an editor, its ease of use and reliability are quite important. If it is hard to use, making even the simplest change can be very frustrating. For example, if it loses track of a file after several hours of creating and correcting it, you may feel like physically abusing what otherwise might have been a fairly good MDS!

OK — but who can help me with the really hard part of finding my program's bugs?

Once a program has been loaded into memory, it is ready to be executed. This is the supreme test and the moment you've been waiting for! Unfortunately, for some strange reason only the simplest of programs run correctly the first time. Most require many, many iterations of testing, locating and fixing bugs, recompiling, and relinking. In fact, the program test phase can easily be the most time-consuming and frustrating phase of all. If there were ever a need for assistance, now's the time!

Enter the debugger, which will help monitor and control execution of the program in memory. A debugger initiates execution of the program, allows you to look at and modify the contents of those memory locations you are interested

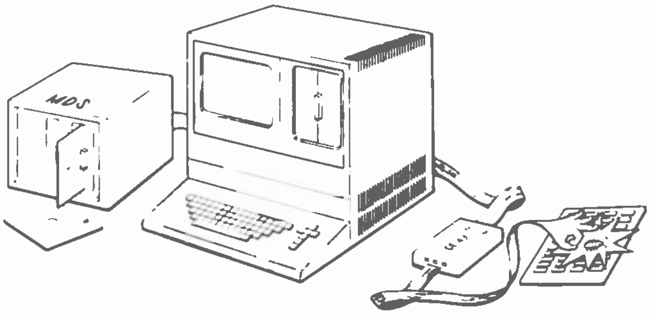
Author Caracappa debugging a system (to his rear and left) in a medium-sized environment. Intel MDS is in the background, on the left.

in, and stops at specified locations to enable you to check values of variables and so on. As with the editor, you'll be spending a good deal of your time interacting with your debugger.

Another important form of assistance at the execution stage is the In-Circuit Emulator (ICE), which physically stands between an MDS and the target system. An ICE module comes with a cable that plugs into the socket in your micro system where the microprocessor normally goes. It takes over the functions of, or emulates, the microprocessor. The software portion of the ICE system resides in the MDS and it allows you to debug as described in the preceding paragraph. The big edge an ICE system gives you over a software debugger is that it also provides a debugger for the hardware. You can check on interrupt operations, clock operation, address lines, and so on, using an ICE.

What's a typical medium-sized system cost?

An example of a system recently developed as a joint project by the Labs and the Manufacturing Technology Center at Indianapolis using medium development resources is a system to aid in color television instrument alignment. The system consisted of an Intel 8086 16-bit processor, two wire-wrapped boards of custom analog and digital logic, 12 kbytes of EPROM and 1 kbyte of RAM. The software was developed on an Intel MDS 230, using the usual steps of editing, compiling, linking, relocating, and loading. The programming language used, Intel's PLM, is roughly a small subset of PL/1. A multitrace oscilloscope and Intel ICE were the hardware development aids. This development environment cost approximately $30,000, although medium-sized systems range from as low as $15,000 to $50,000 or more.



In-circuit emulation hardware can be a valuable tool to help you eliminate the bugs from your design.

What is the common thread running through these medium-sized development tools?

You've guessed it — in building these tools, we've harnessed some of the power of the computer to help us to solve our problem. For assistance in hardware tasks, we have the 370 (or other large computer) to assist with the design, and the (microprocessor-based) logic analyzers and function generators to assist with testing and debugging. For software assistance, the computer has done most of the dirty work for us, from editing our source code to helping debug our object code. In short, many of the things that could be done better by a computer were, indeed, done by a computer.

You've mentioned the good news in using a medium-sized system — how about the bad?

There are still plenty of problems left in developing our microcomputer-based system. The problems lie almost exclusively in the software area, however. Sometimes, for example, you will realize that a version of a program that you've since modified, was correct in the first place. Unless there is a backed-up copy of the original program, your only alternative is to recreate this version manually, and go through the debugging process again. Other times, you'll want to know the value stored in a variable you've called *X*. The debugger in the medium-sized environment may only know numeric storage addresses. So, you have to read through the address table produced by the relocator to find *X*'s address. This activity does not contribute directly to the development of your system and is error prone.

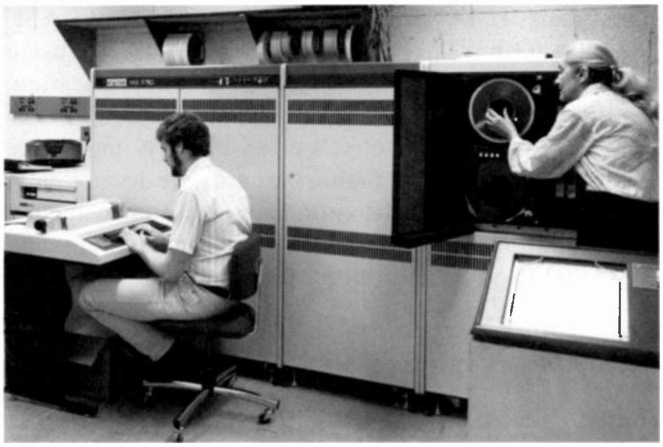
These are but two examples of weaknesses in this collection of tools. Nevertheless, these tools can dramatically increase your productivity, and enjoyment, in developing a microcomputer-based system.

Implementing in a large resource environment

We've seen that a few well-chosen tools, both hardware and software, can do a great deal for us. In this section, we'll see that, when necessary, we can carry this approach even further; the result in doing so is what we've called a "large resource environment (LRE)."

What's different about a Large Resource Environment?

Well, first of all, it has as its core a relatively large computer, such as a Digital Equipment Corporation VAX



VAX 11/780 large-resource environment from Digital Equipment Corporation similar to the DEC20 system mentioned in the article. The system currently supports up to 32 simultaneous users. A high-speed printer/plotter is in the foreground.

11/780 or even the much larger IBM 370 computer system at Cherry Hill, in place of the smaller MDS of the medium-sized environment. It may be dedicated to a particular project or shared with many other users, such as the Cherry Hill machine. An LRE should provide a very convenient cross-development environment with which to prepare and sometimes execute programs that will eventually run on the target microprocessor. Hence, as with the medium-sized environment, we are "cross-developing" programs — partially developing them on one machine, to ultimately run on another.

You might be wondering why in the world we would want to do such a thing? Why fool around developing our microprocessor program on an entirely different machine? After all, now we'll have to learn how to use two machines — not only the micro, but the cross-development machine as well! Well, you have a point, but sometimes — depending on the nature of the project — a cross-development environment can provide enough benefits to make the effort of using it well worthwhile. Let's first look at the tools found in a typical LRE, and then get back to the pros and cons of this approach.

Now, as we've said, an LRE might be thought of as a larger, more powerful version of the MDS used in the medium-sized environment. There are, after all, the usual editors, compilers, and so on, found on an MDS. But there are important differences — some good, a few not so good. The good ones include not only a difference of scale, but also the amount of software tools available and the number of simultaneous users.

An LRE computer is likely to be at least several times faster than most MDSs, typically executing up to a million or more instructions a second. This means that compilations and linkings will be done more quickly, assuming the computer is not heavily loaded by other users. There are generally very large disks (each holding hundreds of millions of characters), tape drives, fast printers, and other convenient peripherals.

Do I get more than speed for my money in an LRE?

A good LRE will also have a wealth of software tools, depending on how long the particular model LRE computer has been in existence and how many other installations use it for applications similar to yours. In fact, so important is the software to the success of an LRE, that the rule of thumb in selecting an LRE should be: choose the richest and most appropriate software environment you can find, then buy the hardware that it runs on.

An LRE is likely to include a greater depth of tools containing, for example, several editors and compilers for several languages. Some of these tools will be more reliable than others, some will have better features than others. Having the freedom to choose between tools with differing features leads to a better fit for a particular project, and may lead to higher individual productivity.

There will also likely be a wider range of tools in an LRE. For example, in addition to source code editors, there may be special document editors (for producing and quickly updating specs, memos, and so on) and graphics editors (for creating graphics scenes to be integrated into and manipulated by graphics programs). There may be automatic backing up and archiving of files. There will likely be cross-compilers and cross-assemblers for a variety of target microcomputers, extending the usefulness of the development environment — and the expertise built up in using it — to other projects. Debuggers may be more powerful, allowing you to work in terms of your high-level language program (for example, variable *X*) instead of in terms of the raw machine (*X*'s object code address).

Unlike its smaller cousin, the MDS, an LRE is a multi-user facility. The computer is time-shared by users, each getting but a fraction of the computer's attention on a millisecond-by-millisecond basis. This means improved communications between several project members developing software; they may "talk" to one another while remaining at their respective terminals by sending messages. The LRE's electronic mail system may also be used for sending notes, to be read and responded to at the receiver's convenience. And last, but far from least, data and programs in a time-sharing system are naturally sharable among its users. Sharing is but a single command away — no computer-to-computer link, no magnetic tape transfer of files from one machine to the other, and no changing of dialects from one machine's high-level language to the other's — simply copy the program with one command and run it.

What's the role of the MDS in an LRE?

The MDS is used in an LRE as a kind of intermediary between the large computer and the micro, rather than as a program development facility as in the medium environment. Downloading often goes through the MDS, on whose disk downloaded programs can be stored before they are passed on into the micro. This allows for generally quicker reloading of the micro's memory after aborts, and is especially useful for those pieces of the software that haven't changed lately.

An essential role of the MDS in an LRE is in debugging using its ICE capability. As we mention later, an LRE can simulate execution only so far, and then an ICE capability is needed.

What does a particular LRE look like?

A recent joint project at the Princeton Labs and the Consumer Electronics Division in Indianapolis, Indiana, employed a Digital Equipment Corporation DEC20-based LRE. This computer supported a dozen or so simultaneous users producing software for an Intel 8086 microcomputer. The software used for this project included two editors, two graphics editors, two cross-assemblers, three cross-compilers, two linker/relocator/loaders, an 8086 simulator/debugger, and a test monitor to assist in systematically applying the tests in the test plan. It also included a source-code-version control program to ensure that changes to a module were recognized by all users of the changed module; a Pascal debugger; a Pascal program-performance monitor to identify program bottlenecks; downloaders to Intel MDSs and to the 8086 directly; document formatters; and a mail system. In addition to the DEC20 itself, there were a graphics terminal, two MDSs, single-board 8086s (the targets), logic analyzers, EPROM burners, and oscilloscopes.

What are the disadvantages of an LRE?

You've probably been wondering about the price tag of such a grandiose system. An LRE can be quite expensive. For example, the above DEC20-based LRE ran well over $300,000 for hardware and software, over $20,000 per year for equipment maintenance, and the equivalent of one full-time person to maintain the system. And these figures don't include the MDSs and associated equipment. In addition, over three person-years were spent tailoring the software tool environment to the project. Of course, the process of customizing an LRE to your needs is an on-going one, and cost must be spread over more than one project. On the other hand, if you plan to use your LRE on another project, you may well have to purchase, modify, and/or build from scratch additional tools needed specifically for the particular micro. This effort can be extensive, and should be considered from the beginning.

A time-shared LRE can result in productivity losses if it becomes heavily loaded — and most of them seem to get that way eventually. Not only does it take longer for the machine to perform each development step (for example, compiling your program) on a loaded system, but the unpredictability of the time that a step will take makes it hard for you to plan your other activities. At least on a single-user MDS, if you've found that a particular compilation will take, say, half an hour, other things can be accomplished during this period.

We've mentioned the ability to simulate execution of a micro on an LRE. Just how accurate is such a simulation? Sometimes not very. For one thing, timing is different. So-called "real-time" programs depend on certain things happening in a certain amount of time — for example, an interrupt should be processed in a known time interval. If these assumptions are violated, the program will likely fail. Furthermore, most simulators don't really simulate the entire microprocessor environment; in particular they don't simulate most peripherals. So when it comes time to exercise these peripherals, the simulator is no longer useful and you must execute on the MDS using ICE or on the target system itself.

Finally, an LRE increases the number of things to be learned (all those handy tools come with manuals), and often (depending on how effective your simulator is) it greatly increases the number of steps you must go through to test and debug a program. If any steps are particularly slow, such as downloading (which is fine at 960 bytes per second but terrible for large programs at 30 bytes per second), the debug cycle can become very time-consuming. Furthermore, additional steps mean more things to go wrong: bugs in the software, hardware problems in the central computer system, terminals, communications equipment, or download link.

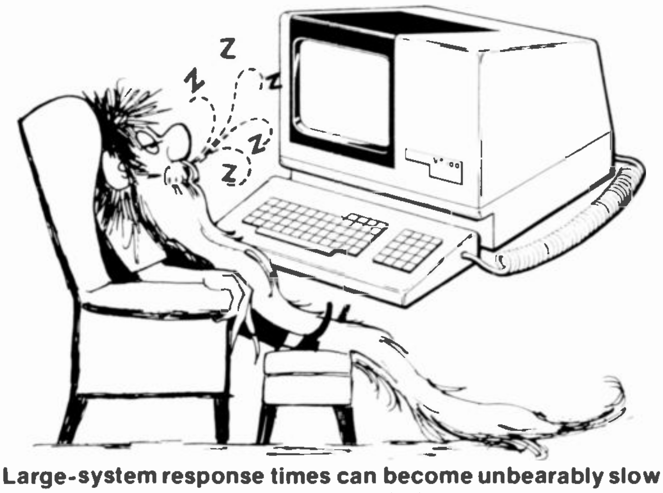
Nevertheless, an LRE is particularly useful in large projects. It also offers the longevity that is often missing in dedicated development systems. Even if the target microprocessor changes from project to project, many tools will remain the same: editors, mail system, document system, and utilities.

Dealing with other system development problems

We've seen three levels of development environments, and some of the pros and cons of tools at each level. But there are other issues and considerations besides tools in developing microcomputer systems.

When can I trust my development tools?

This is a very hard question to answer, but we can offer some guidelines. First, like almost everything else in life,



Large-system response times can become unbearably slow when the systems are heavily loaded.

When you can't find the cause of your problem, let friends look at your program because they can often locate errors which escape you.

you generally get what you pay for. It is often, but not always, true that a $3,000 software debugger will contain more of the desired features and be more reliable than a $300 debugger. At least the higher-priced software usually comes with more maintenance support (that is, assurance of bug fixes).

Secondly, the newer the software package, the higher the risk. Generally, if the version-update number is newer (version 3.37 instead of version 3.26), chances are that you will receive a more bug-free version because this denotes that specific bugs have been found and corrected. On the other hand, if the version number itself has increased (version 4.03 instead of version 3.37), you run the risk of getting an increased number of bugs because a new version number corresponds to major changes (usually new features). Unfortunately, companies who turn out new processor chips are under pressure to deliver development software and provide in-circuit emulation and debugging tools quickly. This leads to hardware and software that has not had the time to be adequately tested.

Even though most tool reliability problems are due to software failures, occasionally hardware tools will fail. We have purchased development tools from a manufacturer (who shall remain nameless!) where we spent several days trying to find out what was wrong with our hardware, only to discover that the hardware emulation and debugging aids were in worse shape than the system we were trying to debug! It is often better to use an existing tool, which someone, who knows where the bugs are, has used before, than to experiment with a new tool.

How do I separate hardware from software problems?

The answer to that question is a rather easy one: always suspect the software. Provided that the hardware was checked by simple-minded test routines, future problems are almost always caused by software errors. This is a concept which can be difficult to put into practice because many of us have a rather large ego to contend with. The typical process goes something like this. We make a change to the software and notice the program no longer works. We note the symptoms and start looking at the appropriate piece of code and sure enough, we find some obvious errors and fix them. We try it again and it sort of works but not all the time, and we can't quite isolate the precise conditions that cause it to fail. At this point, we have carefully looked at every line of code and convinced ourselves that it cannot possibly be in error! And now we make the big mistake and conclude that the only explanation must be that we have a hardware bug, which we now proceed to look for instead of looking at the software again.

What went wrong? We sometimes don't like to suspect ourselves. Often the hardware is designed by us and constructed by someone else. Even when we build it ourselves, we like to put the blame on someone or something physical. After all, when we wrote the program code, we faithfully wrote the instructions which performed just the function we had in mind. Or did we? Well, we did make that obvious error that we found and corrected but now it must be correct... Actually, instead of suspecting the hardware at this point, it's often better to have someone else look at your program. But don't describe each piece of your program and how it works. If you do, that person will go down the same rosy path that you did and reach the same conclusion that you did — that there is nothing wrong with the program. Often, someone unfamiliar with your program will quickly find an error in a piece of code that you must have looked at hundreds of times. The problem is that when you wrote it, you were sure it was correct, and each time thereafter you only further convinced yourself of its correctness. We'll admit that this procedure can be somewhat humiliating, but it's often very worthwhile indeed.

Of course, if you (and your friends) have spent days or weeks looking for a software problem to no avail, then you might begin to suspect the hardware. At this point, you'll drop back to simple hardware test routines to try to isolate the problem.

Which processor should I use?

When it comes to choosing a processor or deciding whether or not to perform a given task in software, many factors should be considered. How much computing power do you really need? How much effort will be required to perform this task in software? How much will the finished system cost?

First, let's start with some realities. It always requires more effort (often ten or more times more effort) to implement a given task in software instead of using a specialty hardware chip. In contrast to the hardware situation, there are presently very few off-the-shelf software building blocks. Software is generally written to exactly fit a situation, and hence is not reusable in a different context.

The hardware/software choice is often dictated by the ultimate use of the system, and other times by the tools and development facilities available. Or, the choice may be based on the experience that the people involved have with a particular processor.

Shouldn't the ultimate usage guide my choices?

Absolutely. If your system is going to be part of a device that will be sold by the tens of thousands, then you will choose the processor with the minimum required capabilities at the lowest possible cost. Almost every function possible will be performed in software to reduce hardware replication costs. For this kind of a situation, the cost of development tools (although they may be very expensive) is insignificant. It is also worthwhile to spend a month to develop a routine in software that eliminates a $2.00 hardware chip. In short, it is almost worth any effort (and a lot of effort will be required since it will be just barely possible to perform the functions with the processor that is chosen) in order to reduce the system replication cost by a few pennies.

On the other hand, suppose you are developing a system that will be one of a kind, or perhaps replicated a few times. Then you should not be afraid to include a liberal sprinkling of sophisticated hardware support chips in your design, as it will reduce the development time and effort required. And if the processor you choose is more expensive and has capabilities that you will not fully use, so what? It is also perfectly reasonable to use a processor that you are familiar with even if it may not be the best choice, particularly if you want something working in a relatively short period of time.

How can I obtain the necessary equipment and expertise?

Obtaining microprocessor development equipment can be an expensive proposition (remember your manager's reaction!). Also, if you don't have some system design experience it is very difficult to know what you should buy. First, you must determine your needs. If you will have to implement a wide variety of systems, then one of the general-purpose systems that support several of the popular processors might be appropriate (such as one from Tektronix or General Radio). On the other hand, if your needs can be met by one family of processors from a single vendor, then you will be better off with a development system from that vendor. The reason for this is that you will be able to get support for the latest product the vendor has to offer almost as soon as it is available. Also, you can usually get help with problems from vendors who are anxious to sell you their processors.

We should point out that a general-purpose development system, although more flexible and adaptable to new processors, does have some drawbacks. Availability in these systems of in-circuit debugging hardware and software for a new processor usually lags behind that available from the manufacturer by many months or even years in some cases. The general-purpose systems are also usually rather slow by comparison and are designed to be a jack-of-all-trades and consequently master of none. Still, if you need to work with a variety of processors that are not the very latest that technology has to offer, then the general-purpose system is your best choice.

Where do you go from here?

A wide range of available tools can greatly improve your productivity in developing a micro-based system. What does this mean for you? Well, productivity is a complex issue, depending on such things as the "friendliness" of the system, the ease of learning the operating system and applications programs, the languages and other software available for the system, and the expandability for future applications.

Common sense tells us that we don't need the power of a time-sharing system to develop a simple controller and likewise we cannot have twenty people working on a complicated project in the "bare-bones" environment. In most cases, however, we are not faced with such an obvious choice. Usually more than one person and less than twenty must be considered. Also, the project is often neither simple nor extremely complicated.

What choice should one make? Part of the answer lies in understanding the minimum resources needed to implement your particular application. There is a problem if you are getting into microprocessor systems for the first time with little awareness of the options available to you. You could listen to the salesmen for the various micro, mini and time-sharing development systems, but when you listen to a salesman you may get the impression that with a few expensive options the system could double as a boat or an airplane for weekend recreational purposes.

If you are inexperienced, try to get some advice on what your minimum resource requirements are. After you have a good feel for the features you need, talk to development system vendors and ask a lot of tough questions such as: how does your option really fit my needs? If the option is new, how soon will it be installed in my plant, and what happens if it doesn't work? If you find that you need a lot of expensive options to the basic system in order to meet your needs, you should probably consider a more powerful basic system. In many cases, a less powerful system with lots of options will be the more expensive choice.

The main point is to let your demonstrated (as opposed to imagined) needs drive your quest for and selection of development tools. This way you're more assured of using what you buy — at least until the next incompatible project comes along!

But how do I learn to use all these tools?

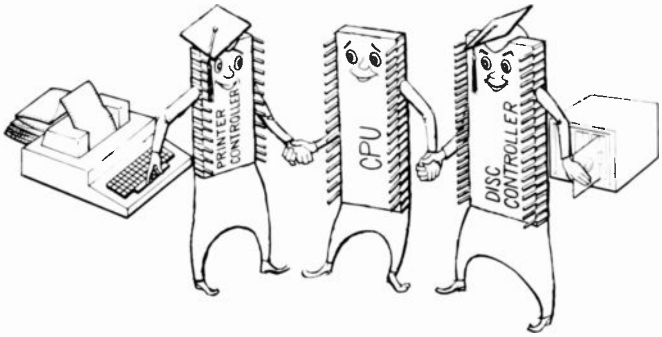
If may not be as hard as it sounds. If you learn well in a classroom environment, then there are several microprocessor courses offered as part of RCA's continuing education program. There are also many good books on microprocessors and system design — some from the manufacturers themselves are free. If you learn best by doing, then there are several microprocessor courses offered that include a small computer system (yours to keep). This is particularly useful since you can continue to play with your small system to gain additional insight and expertise. If you are also an incurable experimenter (once having gained the necessary experience, this person probably makes the best small-system designer), you will probably want to obtain a microprocessor and some support chips and design your own system from the start. You can get by with very little, other than your desire to learn what makes the system tick. You can start off in the "bare-bones" environment, as discussed earlier, and obtain some valuable experience and insight. If you have the desire don't be afraid to dig right in with only a handful of chips. Even if you have no hardware experience, you can usually gain what is needed from a manufacturer's applications note.

Trends in microprocessor system development

It seems hard to imagine that the microprocessor field began only a few short years ago with the lowly 8008 from Intel. Even the term "microprocessor" has lost some of its original meaning. Some of the newer "micros" can legitimately be considered minicomputers or even mainframes. Micros now come in assorted flavors from single-bit controllers up to 32-bit machines with 4-, 8-, 12-, and 16-bit machines in between. The trend is forever upward in size and complexity. Where will it all lead? Will we have handheld 370s before the decade is out? Everyone is squeezing more and more into the same silicon real estate as before; even the limits of optical lithography do not seem to have slowed anyone down.

How will this trend affect my hardware/software system designs?

For the one-of-a-kind system, life will become increasingly easier as you make use of the specialty support chips now becoming widely available. The software task in many cases will center on merely sending data to and from the support chips. For medium size systems, it will be hard to justify anything written in anything other than a high-level language. With the low cost of memory, you can now reasonably implement your software in FORTRAN, BASIC, Pascal or even PL/1 on even the low-cost 8-bit machines. Microprocessor manufacturers are also designing machines for specific languages by gearing instruction sets to particular languages, such as the Pascal Micro Engine from Western Digital or the Ada-oriented iAPX 432 microprocessor from Intel.



Intelligent peripheral control chips essentially reduce the programming burden to one of handshaking.

All these advances lead to reduced effort to develop useful applications software. This is very significant when you consider that software development costs and effort are often an order of magnitude greater than those for hardware development. Even the cost-sensitive applications which cannot use these new wonders will benefit from the lower cost of the simpler chips due to the experience gained in making the more complex ones. Overall,˙ we see the trend continuing with more and more specialty chips to reduce your software effort as well as a steady trend towards high-level language development for small systems implementation.

Will cheaper, more powerful hardware affect software development tools?

Most certainly. The new, much more powerful dedicated computer systems are beginning to blur the distinctions between the LRE and MDS environment. In fact, systems for around $30,000 combine the best of these two environments — the power and peripherals of a minicomputer, plus a high-speed network for tying systems together (providing the vital communications link found in a time-sharing system). Although these systems can be shared by several users, the tendency as they become cheaper and more plentiful, will be to use them as single-user ("personal") systems.

Perhaps the most important software trend is the ongoing implementation (through the combined effort of many companies and universities) of generally useful software tool sets. These tools run on any machine that supports FORTRAN (virtually all machines). This effort was motivated by the work at Bell Labs on a system called Unix™. Unix™ consists of an operating system and a very wide and deep set of software tools, all implemented in the high-level language C. For a while Unix™ only was implemented for PDP/11 computers, but it is now becoming available on a wide variety of computers, including Digital's VAX, IBM's 370, and many microcomputers. This commonality of tool sets across projects and machines will help improve productivity in the software development area.

It sounds exciting, but...

Although all of these developments help reduce the software effort required for a given application, things are becoming more complex with each new processor. Gaining expertise with each of these new developments will become a greater burden. Development tools will also be more sophisticated and costly, and as everything becomes more complex, you will not be able to do without them. For example, programs written for Intel's 8086 processor are nearly impossible to hand-assemble due to the complexity of both the machine and the assembly language.

Microcomputer systems and development terms

acceptance test Final test of a system — tests a system's functionality, reliability, etc.

Ada A powerful, new, U.S.-government-sponsored high-level programming language for embedded computer systems, but generally useful for a wide range of applications. It has its origins in Pascal.

assembler Translates an assembly-language program into object code (see "object code").

BASIC An old but still thriving high-level programming language used for scientific and other applications. It emphasizes interactive computing.

breadboard The initial hardware part of a system used to iron out the flaws (usually wired by hand).

circuit analysis program From a description of an electronic circuit, initial circuit condition, and input (driving) functions, this program simulates the circuit and generates a set of output (response) functions.

compiler Translates high-level language programs on host machine into object code for the host machine.

cross-assembler Translates assembly-language programs on host machine into object code for the target machine.

cross-compiler Translates high-level language programs on the host machine into object code for the target machine.

cross-developing Developing software on the host computer for the target computer.

data-flow diagram Shows flow of data and data transformations in a system.

debugger Provides execution-time help to programmer in finding bugs in his or her program.

downloader See loader.

editor A text editor (or just "editor") is used to compose and modify a program's source code. A document editor formats a document as it is entered

Although each of the new chips tends to make things easier than they were before, each of them promises more processing power and speed. At the same time, we are attempting to implement much larger, more complex applications.

Are things becoming too complex? No! There will always be applications which are just out of reach of the current technology. But more and more applications are coming within reach of the ever-expanding microprocessor frontier. This same technology, coupled with sophisticated software packages, means that we will also have the necessary tools to build tomorrow's systems.

Authors (left to right): Nils O. Ny, G. David Ripley, David N. Caracappa.

Reference

1. Ripley, G.D., "Some Software Engineering Techniques," RCA Technical Report (Oct. 1980).

David Caracappa has been with RCA since 1974, and is currently a Member of Technical Staff at the Laboratories. Originally he was with Astro-Electronics in Hightstown. While there, he worked on the TIROS weather satellite program and did advanced system design work on utilizing microprocessors in command decoders, telemetry processors, and the space shuttle camera controller. Since joining DSRC in 1978, he has been involved in the design and implementation of microprocessor-based systems for new Consumer Electronics products and color TV manufacturing systems.

Contact him at:
RCA Laboratories
Princeton, N.J.
TACNET: 226-3278

Nils Ny originally joined RCA in 1965 at Astro-Electronics in Hightstown and came to the Laboratories a year later. He has been involved with minicomputer and microprocessor systems for a variety of applications including colorimetry analysis, cable TV communications and factory automation. He is currently a Member of Technical Staff with the Microtechnology Research Group where he works with microprocessor hardware and software for future consumer electronic products.

Contact him at:
RCA Laboratories
Princeton, N.J.
TACNET: 226-2209

David Ripley originally joined RCA Laboratories as a Member of Technical Staff in 1970, doing research in programming languages and translators. After spending six years at the University of Arizona on the Computer Science Department faculty doing research in language translators and program performance measurement, Dr. Ripley rejoined RCA Laboratories in 1978. He is presently involved in the software for new consumer electronic products and the design of user-computer interfaces.

Contact him at:
RCA Laboratories
Princeton, N.J.
TACNET: 226-2884

into the machine. A graphics editor is used to compose and modify graphics scenes for presentation on a usually high-resolution, color CRT.

electronic mail Enables users of a time-sharing system (or collection of connected computers) to communicate by sending each other messages, or "mail."

emulator Hardware that mimics a particular processor chip.

EPROM Eraseable Programmable Read-Only Memory; can usually be erased by exposure to ultraviolet light, and reused.

executable image The binary version of a program that can be loaded directly into memory and executed.

FORTRAN An old but active high-level programming language used primarily for science and engineering applications, and available on nearly all machines.

function generator Hardware used to produce electrical signals of desired waveshape. These may be analog or digital functions.

high-level language A programming language that contains terms and constructs that are high-level (or English-like) and applications-oriented.

host system The computer system on which software is cross-developed (see "target system").

ICE (In-Circuit Emulation) Debugging by using an emulator instead of the processor chip in the target system.

interface diagram Shows connections of external signals to the microprocessor in the system.

linker Combines separately compiled or assembled object modules into one program by filling in linkages (calls and data references to the various modules).

loader Loads an executable image into a computer's memory. If there is loading from host machine to target machine, it consists of two parts: the "pitcher" or the host, sending the image to the "catcher" or the target, which then loads into the target memory. combination is called the downloader.

logic analyzer Hardware that displays a state diagram of some of its inputs (usually 8 or 16) upon occurrence of a specified condition of the other inputs (usually 16).

MDS Microcomputer development system.

micro See microcomputer.

microcomputer A single chip containing a processor, RAM, ROM (or EPROM), and I/O.

microcomputer development system A host system consisting of a computer, I/O devices (e.g., CRT, printer), mass storage (e.g., floppy disk), and software for developing software for a target system.

microprocessor A single chip containing only a processor.

module A single main program, subroutine, procedure, or function.

object code The binary representation of a program — translated source code.

oscilloscope Hardware which displays a graphical representation of one or two inputs.

Pascal A modern, high-level programming language, used for a wide range of applications, that emphasizes flexible data types and error checking.

PL/I A large high-level programming language for general-purpose use that has features from FORTRAN, COBOL (a popular business language), and ALSOL (an early scientific language).

port The physical means through which a computer communicates with external devices (there are typically at least several of these per processor).

processor The central processing unit in a computer. It fetches and executes instructions.

RAM (Random Access Memory) A memory device, an arbitrary location of which can be read from or written into.

relocator Relocates object code addresses to enable the code to reside in a specified memory location.

ROM (Read Only Memory) A memory device, an arbitrary location of which can be read from but *not* written into.

simulator Simulates the execution of one computer on another computer.

source code The original, human-readable representation of a program.

strobe An electrically generated pulse that can be used to synchronize other events.

system block diagram Shows basic functional portions of the system and their relationships to each other.

target system A computer system for which software is developed (see "host system").

test plan Describes tests to be made on the target system and specifies intent, expected results, and chronology.

time-sharing Scheme for sharing a computer between a number of simultaneous users, each of which appears to have the computer to himself or herself.

J.P. Paradise

New CDP1805 microprocessor upgrades CDP1800-based systems

A new microprocessor, the CDP1805, speeds throughput and reduces chip count while retaining all the advantages of the register-based CDP1800-series architecture.

Abstract: *The CDP1805, an updated version of the time-tested CDP1802 microprocessor, possesses on-chip hardware and software functions that are similar to those found on many microcomputers. These additional functions, coupled with a higher clock rate than on the CDP1802, can offer significant improvement in both system throughput and integration, as well as widen the field of application of CDP1802-based systems.*

The RCA CDP1800 microprocessor series is a well-tried LSI product line that implements control-oriented microcomputer functions in CMOS technology. The preeminent device of the series, the CDP1802, has been in production since 1977, with yearly output now exceeding one million devices per year. Finding its way into a wide range of applications that require low power for portability, high noise immunity for satisfactory operation in industrial environments, or a wide operating temperature and voltage range to assure optimum performance under a wide variety of ambient conditions, the CDP1802 has established itself as the leading CMOS microprocessor. With the introduction of the CDP1805 by RCA, CDP1800-based designs can be further enhanced, and proposed new or improved system designs whose performance requirements previously exceeded the capability of the CDP1802 can now be implemented.

The CDP1800-series microprocessor architecture

This section will help readers unfamiliar with CDP1800-series architecture to understand its fundamentals and salient features, so that the enhancements achieved in the CDP1805 can be more fully appreciated.

As shown in Fig. 1, the central feature of the CDP1800-series microprocessor is an array of sixteen 16-bit scratchpad registers used to provide control over memory addressing and internal housekeeping. When used to address memory, the registers are selected by software instructions that load 4-bit values into register selectors (P,N,X). These selectors program the scratchpad registers as program counters, memory pointers, and stack pointers. This assignment flexibility allows the use of multiple-pointer and context-switching techniques, and provides quick subroutine-call implementation and efficient stack and interrupt handling in real-time control applications. In addition, these same registers can be used for variable data storage, providing up to 30 bytes of data memory on-chip.

The 16-bit-wide register-array matrix provides two advantages in its dual use in address and data operations. A 2^{16} or 65,536-byte memory address range is possible for RAM/ROM addressing in main program, subroutine, and stack operations. For data storage, the contents of each 16-bit register can be used as a 16-bit data word, with instructions for increment, decrement, register-to-register, and register-to-memory manipulation of 16-bit operands. Thus, 16-bit arithmetic and logic operations can be supported by this internal architectural configuration.

Externally, CDP1800-series devices interface with a variety of bus and I/O pins to external memory and peripheral functions. Memory addresses are generated via a multiplexed address bus; a latching TPA signal is provided for the high-order byte. I/O addressing is accomplished via a separate 3-bit I/O bus with dedicated instructions for data transfer between memory and peripherals.

Other I/O lines are provided for on-board DMA address and control-line generation, for interrupt vectoring, for use as flag lines for polling, and for implementation of a Q-line-output port for control of external devices. The flag and Q lines, in conjunction with a software-driver routine, can also be used as a serial port to external I/O.

Reprint RE-26-8-2 ST-7009
Final manuscript received July 15, 1981.

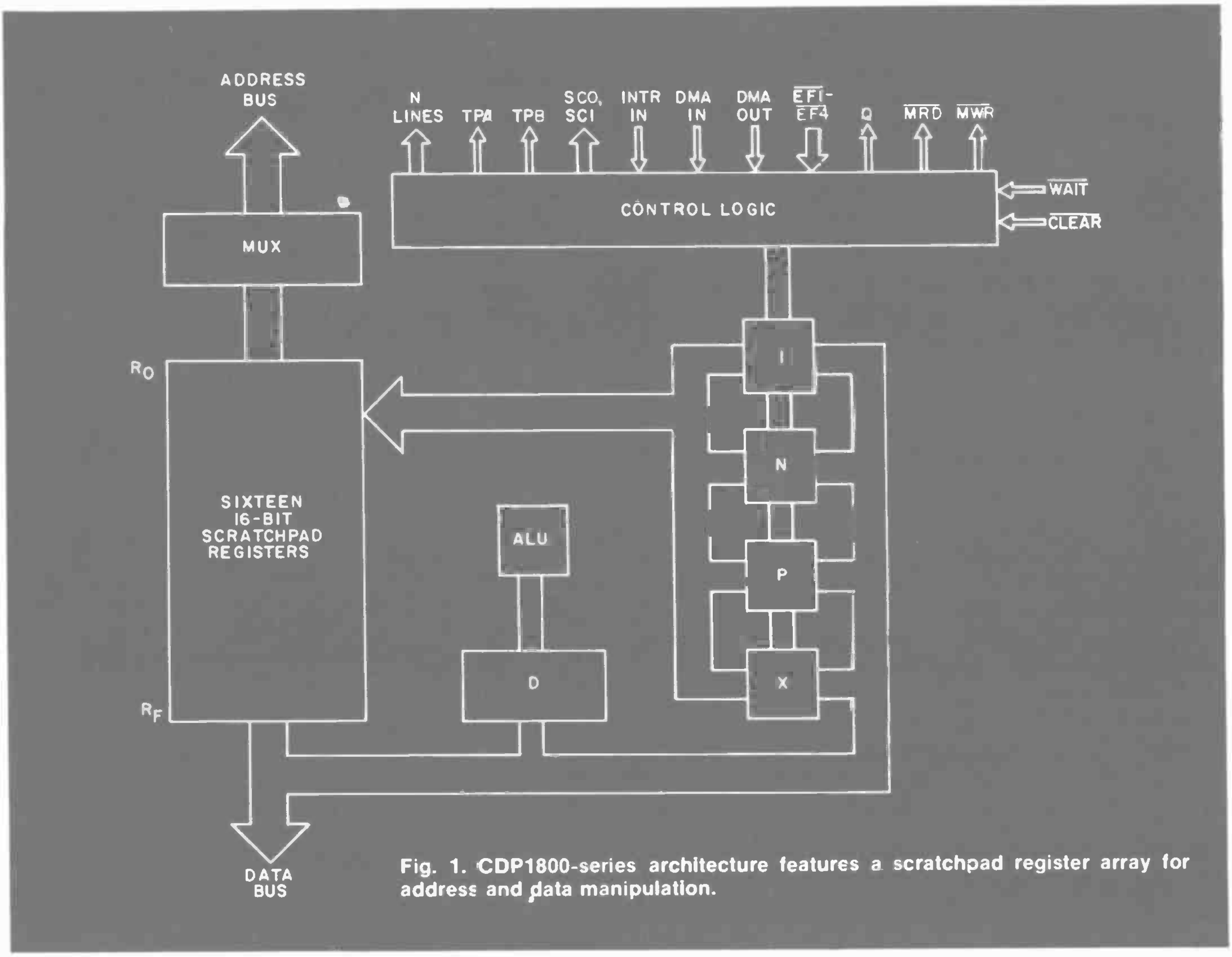


Fig. 1. CDP1800-series architecture features a scratchpad register array for address and data manipulation.

The remainder of this article describes specific CDP1805 features and explains how they are optimally applied in a microcomputer system.

CDP1805 — the specifics

On-board RAM complement

The CDP1805 includes, on-board, a 64-byte RAM array in addition to the 16 x 16 scratchpad register array described above. The RAM's six address lines are internally connected to the CPU section; the eight data lines are part of the CPU's external bus. The chip-select function for the RAM is available through pin 16 (the V_{CC} connection on the CDP1802). Thus, the RAM is essentially configured as independent memory, with its 64-byte block locatable anywhere in memory space, and with its data lines available to external I/O for reading and writing. The 64-byte block is of adequate size for a stack area, being able to handle many levels of subroutine and interrupt nesting, as well as providing sufficient space for a modest data stack for main program or I/O routines. Figure 2 shows a typical connection of the CDP1805 in a CDP1800-based system.

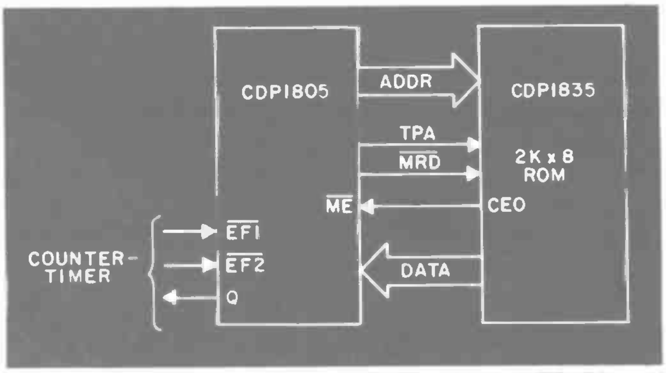


Fig. 2. Minimum connection diagram for a CDP1800-based system that includes CPU, 2-kbytes ROM, 64 bytes of RAM, and a counter-timer.

On-board counter-timer

The CDP1805 contains an 8-bit presettable down-counter on-chip. The counter is configurable in a wide variety of modes through the use of new CDP1805 software instructions (Fig. 3); external input and output lines are available via the flag and Q pins. The counter features an internal counter interrupt, which is activated on counter underflow, and internal execution similar to the external interrupt mechanism found on the CDP1802.

The counter is loaded with a desired count, and preset to its intended operating mode, by means of software instructions. Counter contents may be loaded into the CPU D register and manually decremented, a function that can replace software loops in timing applications. In its dynamic operating modes (of which there are five) the counter-timer may receive its internal decrement-clock signal from TPA divided by a ÷ 32 prescaler, or from flag lines $\overline{\text{EF1}}$ and $\overline{\text{EF2}}$ (Fig. 4).

In the timer mode, the effective clock input is the crystal frequency divided by 256 (XTAL ÷ 8 = TPA, TPA ÷ 32 = prescaler output). Interrupts are generated on each counter underflow; the preset counter value is reloaded into the counter after each interrupt. In the timer mode, the counter-timer can function as an accurate time base in real-time applications. An ETQ instruction causes Q to toggle with each interrupt, creating a programmable-frequency square wave for external-device control applications.

Event counting can be performed in the SCM1 and SCM2 modes (Fig. 3) by bypassing the internal TPA clock and feeding inputs directly from $\overline{\text{EF1}}$ and $\overline{\text{EF2}}$. This arrangement still permits the flag lines to retain their normal functions. Again, the counter generates an interrupt when a preset number of external events have occurred.

$\overline{\text{EF1}}$ and $\overline{\text{EF2}}$ can also be used to make pulse-width measurements in the SPM1 or SPM2 modes (Fig. 3). In either of these modes, $\overline{\text{EF1}}$ and $\overline{\text{EF2}}$ are used as a gate to the TPA clock without the prescale divider. An interrupt is generated, with the remaining count frozen, on the trailing-edge transition of the flag line. Thus, the counter value represents the width of the pulse appearing at $\overline{\text{EF1}}$ or $\overline{\text{EF2}}$. Since the two flag inputs may be used together, these modes are useful for comparison-type measurements.

These generalized explanations give some idea of how the counter can be used in control applications. The specific examples given below employ some of these counter-timer functions and illustrate other powerful features of CDP1800-series I/O.

Enhanced hardware DMA: the counter-timer

On-board DMA control is a useful I/O feature of CDP1800 architecture, especially in data transfer operations where high speed is essential. Instead of requiring an external controller to provide address and memory read/write signals, the CDP1800 processor performs these functions automatically, through register R(O), when DMA-in or DMA-out pins are activated. The counter-timer, along with a single inexpensive CD4011, a COS/MOS NAND gate, keeps track of the desired number of DMA transfers, with the peripheral only required to issue a DMA request in the form of a single short pulse. The modest internal software initialization and maintenance required with this method results in an acceptable trade-off with external hardware techniques.

LDC	— LOAD
GEC	— READ
STPC	— STOP
DTC	— DECREMENT
STM	— TIMER MODE
SCM1	— EVENT COUNTER VIA EF1
SCM2	— EVENT COUNTER VIA EF2
SPM1	— PULSE WIDTH VIA EF1
SPM2	— PULSE WIDTH VIA EF2
ETQ	— COUNTER OUTPUT VIA Q

Fig. 3. CDP1805 counter-timer instructions for manual control and selection of one of five dynamic modes.

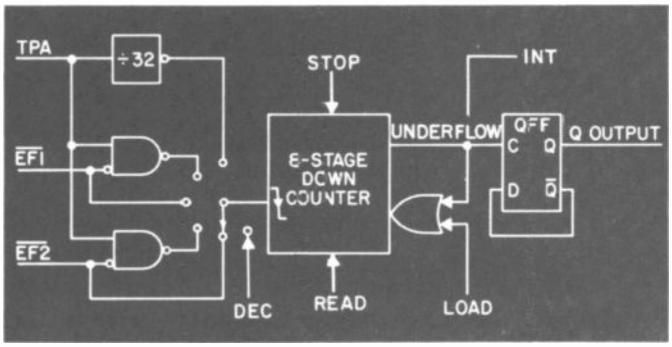


Fig. 4. CDP1805 counter-timer model. The 8-bit down-counter has a variety of possible inputs and an output available from the Q line.

In the configuration shown in Fig. 5, the CDP1805 itself keeps track of DMA transfers. The counter-timer is placed in event-counting-mode SCM1, and counts TPA transitions that occur during the time that DMA is active. When the desired number of transfers is complete, Q toggles, ending the DMA mode.

Software DMA with the CDP1805 counter-timer

When properly initialized, the CDP1805 can be used to generate software-driven DMA (Fig. 6). In this application, the counter-timer is set to pulse-mode SPM1. An SEQ instruction, locatable anywhere in software, forces DMA to begin. The counter counts internal TPA pulses

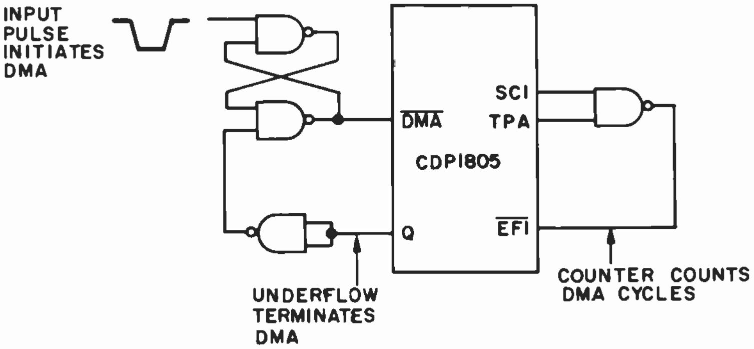


Fig. 5. CDP1805-enhanced DMA operation using on-board counter-timer. The counter-timer can be present to count from 1 to 255 DMA transfers.

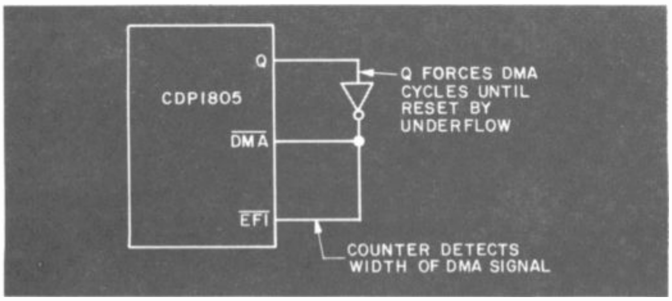

Q
CDP1805
DMA
EFI
Q FORCES DMA CYCLES UNTIL RESET BY UNDERFLOW
COUNTER DETECTS WIDTH OF DMA SIGNAL

Fig. 6. Software-controlled DMA using counter-timer. The user has full control over any DMA transfer.

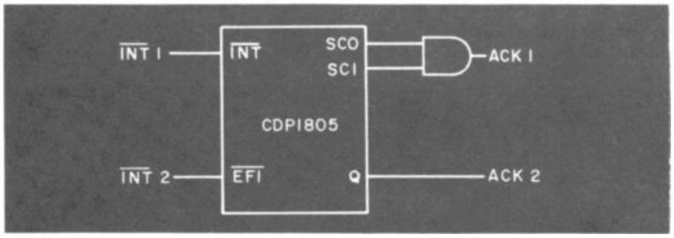


Fig. 7. CDP1805 multiple interrupt capability with handshaking using counter-timer. The additional interrupt is edge-sensitive and has its own handshake line.

until underflow, at which time Q toggles and terminates the DMA mode. The process may be repeated anywhere in the software sequence; the number of transferred bytes is determined by the count loaded into the counter-timer.

Adding a second interrupt to the CDP1805

The internal counter-timer interrupt that is generated within the CDP1805 can be used to advantage as a second external interrupt, complete with a dedicated handshake line. Furthermore, this interrupt is edge rather than level sensitive, and the interrupt-acknowledge signal can be reset manually within the service routine to provide accurate status information to a peripheral.

For this operation, shown in Fig. 7, the counter-timer is set to the counter mode, with a count of 01 preloaded, and with ETQ enabled. The first high-to-low input-signal transition causes a counter underflow, generating an interrupt and creating ACK2. The interrupt source is arbitrated within the interrupt-service routine, during which time ACK2 can be reset.

Enhanced interrupt control

The type of interrupt action described in the counter-timer discussion above is made possible because of the enhanced interrupt-control logic and instructions of the CDP1805. Since two interrupts must be dealt with internally in the CDP1805, six additional linked instructions, shown in Fig. 8, have been provided for arbitration and control. In addition to standard CDP1800-series RET and DIS instructions, which provide master interrupt control, separate XIE, XID, CIE, and CID instructions are provided for independent control of external and counter interrupts. In addition, both external and counter interrupts are pollable by means of BXI and BCI short-branch instructions.

The advantages to the user of this structure are that he can enable or disable a counter or external interrupt request with a simple instruction, and avoid the indirect programming structure of RET and DIS for simple enable or disable functions. Although not latched, the external interrupt is pollable, making it useful as a fifth flag line, if desired. Finally, the latched counter interrupt can be tested as a real-time or polled event by selective use of the CIE, CID, and BCI instructions.

RC-oscillator capability

The addition of a Schmitt trigger in the CDP1805 oscillator section provides crystal or RC-oscillator capability (Fig. 9). An RC oscillator can have several advantages over a crystal in noncritical timing applications. The most obvious advantage is in cost; the crystal typically costs ten times more than the resistor-capacitor combination. A temperature-compensated capacitor can be purchased to improve stability further when required by the application.

The RC structure also permits easier change of frequency. The resistor can be replaced with a simple user-adjustable potentiometer. Further, step changes in frequency for specific applications, time-base changes, or lower power considerations can be accommodated through simple RC-switching techniques. By way of providing basic design guidelines, Fig. 10 matches resistor and capacitor values with frequency.

Higher clock rate

The 4-MHz clock speed of the CDP1805, a 25-percent increase over the 3.2 MHz of the CDP1802A, can give the

XIE	**— ENABLE EXTERNAL INT**
XID	**— DISABLE EXTERNAL INT**
CIE	**— ENABLE COUNTER INT**
CID	**— DISABLE COUNTER INT**
BCI	**— BRANCH ON COUNTER INT**
BXI	**— BRANCH ON EXTERNAL INT**

Fig. 8. CDP1805 interrupt-control instructions. These instructions arbitrate between external and counter interrupts.

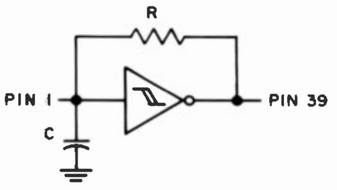


Fig. 9. CDP1805 oscillator configuration using RC time constant. The internal Schmitt trigger allows crystal or RC operation.

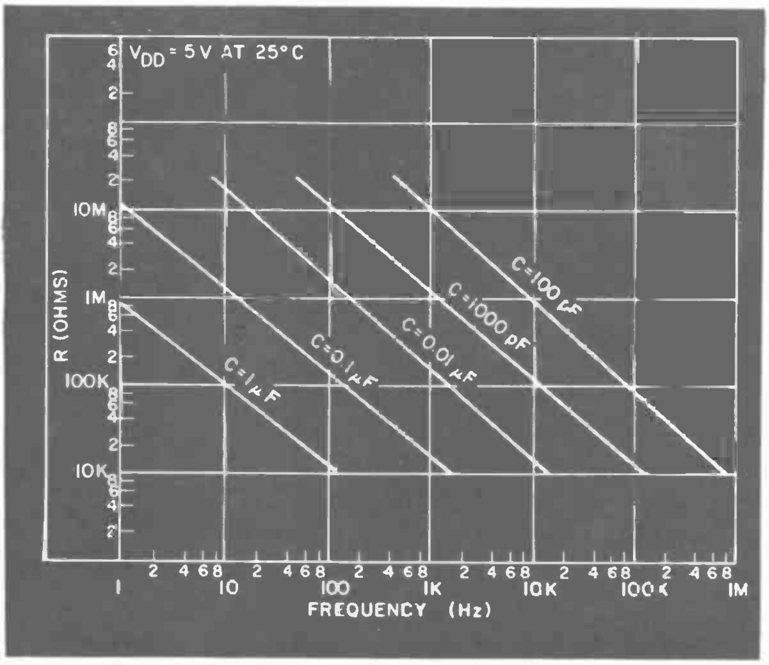


Fig. 10. Component values as a function of frequency for the RC oscillator. The graph shows that frequencies up to 1 MHz are achievable.

CDP1805 the performance edge in certain applications. This performance edge, together with the enhanced data instructions described below, can produce fast throughput in both real-time and response-critical applications.

SCAL and SRET instructions

Of all new CDP1805 instructions, the new call and return instructions, SCAL and SRET, provide the most significant improvement in throughput. These 4-byte instructions perform the same function as the SCRT technique outlined in existing user manuals. However, the savings in code and execution time is significant (Table I), especially where subroutines are frequently used.

Besides providing subroutine call-and-return capability in a single instruction, SCAL and SRET support the additional parameter-passing feature. Parameters are passed from main program to subroutine by saving the main program counter in a scratchpad register, rather than on a stack, and using a memory-load instruction to link data following the SCAL instruction to the subroutine.

16-bit data instructions

Four instructions, RLDI, RLXA, RSXD, and RNX, were added to the CDP1805 to handle enhanced 16-bit operations involving the scratchpad registers. Figures 11 and 12 illustrate the enhancements made possible through these new instructions. For example, they permit 16-bit loads of data from program memory to a designated scratchpad, 16-bit loads and stores between scratchpads and memory, and transfer of the contents of any scratchpad to R(X).

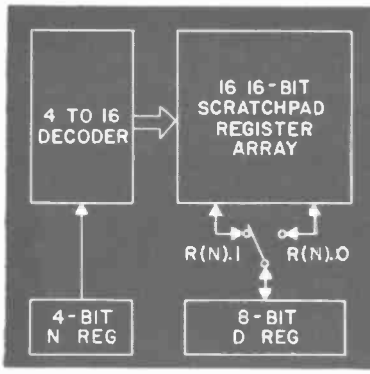


Fig. 11. CDP1802 scratchpad-register data-transfer model. The register array is accessed via the D register.

Table I. Performance Comparisons — subroutine call and routine for CDP1802 and CDP1805 implemented with various techniques.

	1802 SOFTWARE "SCRT" TECHNIQUE	1802 SOFTWARE "SEP" TECHNIQUE	1804/05 SOFTWARE SCAL / SRET INSTRUCTIONS	1804/05 SOFTWARE "SEP" TECHNIQUE
NUMBER OF MACHINE CYCLES - CALL	32	2	10	2
NUMBER OF MACHINE CYCLES - RETURN	24	4	8	4
CALL TIME (1802 @ 3.2 MHz) (1804/05 @ 4 MHz)	80 μs	5 μs	20 μs	4 μs
RETURN TIME (1802 @ 3.2 MHz) (1804/05 @ 4 MHz)	60 μs	10 μs	16 μs	8 μs
NUMBER OF BYTES SOFTWARE CALL + RETURN	45	4	6	4

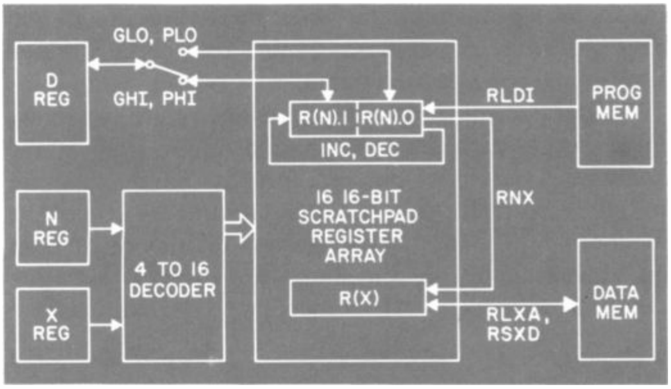


Fig. 12. Enhanced CDP1805 scratchpad-register data-transfer model. The CDP1805 allows direct register-memory transfers.

Several advantages accrue from these enhancements. In the case of RLDI, two bytes are saved for each register load, but more importantly, the contents of the D register are not destroyed in the process. RLXA and RSXD register loads and stores, from or to memory, are useful for stack storage of 16-bit address values, or 16-bit data manipulation in arithmetic operations. Finally, the RNX instruction permits any scratchpad to become a memory pointer for stack and I/O operations.

Conclusion

The architecture and performance improvements afforded by the new CDP1805 microprocessor have been reviewed. For each improvement, an advantage or benefit to the user in a real system application has been pointed out. For customers requiring any or all of these performance improvements, the CDP1805 may prove to be the appropriate choice in a new or modified CMOS system design.

Joe Paradise is currently Leader, LSI Applications, for the CMOS CDP1800-series product line. His responsibilities include customer interface and presentations, preparation of support literature, and definition of new CPU and peripheral products. He has been involved in nearly all phases of CMOS engineering since he joined RCA in 1970, including design engineering of standard and custom CD4000-series logic devices, microprocessor peripheral design, product engineering, high-reliability engineering, and automotive-applications engineering.

Contact him at:
Solid State Division
Somerville, N.J.
TACNET: 325-7352

J.P. Paradise

CDP1800-series peripherals

are building blocks of a complete processor family

The CDP1800 series now offers a full array of auxiliary chips that make available functions, flexibility, and performance levels once achievable only with NMOS technology.

Abstract: *The author discusses the structure, operation and application of a number of peripheral devices that take advantage of the versatility of the CDP1800-series I/O architecture to allow implementation of diverse systems.*

Table I lists all of the currently available and soon-to-be-announced peripheral functions in the CDP1800 series. These functions range from simple buffers, decoders, and latches to complex special and general-purpose devices. See other paper in this issue or the appropriate literature (listed in the references) for more information on CDP1800-series I/O.[1]

This discussion is focused on the more complex general-purpose components (marked with an asterisk in the table), which contain a number of addressable registers and which can operate in a variety of programmable modes. These types of components and their functions more closely match equivalent NMOS offerings found in recent catalogs, and all feature the ability to interface with any general-purpose bus-oriented microprocessor or expandable microcomputer. The diagram in Fig. 1 shows a typical interface between CMOS multiplexed-address/data bus processors and complex CDP1800-series I/O devices. All of these featured devices will operate in a 5-MHz CDP1800 system, with access times on the order of 500 nanoseconds (at 5 volts). Operation is over a voltage range of 4 to 10.5 volts, and over a temperature spread of −40 to +85°C.

CDP1851, programmable I/O expander

The CDP1851 is a general-purpose programmable I/O device that has 20 I/O lines that may be used in several different modes of operation.[2] Two full 8-bit ports, complete with handshaking lines, provide efficient interfacing between a parallel CPU bus and peripheral functions. The CDP1851 is programmed by the CPU — by means of the CDP1851 control register — to define port mode, interrupt enabling, I/O bit assignment, bit masking, and so on.

Table II shows a summary of the CDP1851 programming modes with corresponding pin configurations. Simple input- or output-port functions with "ready" and "strobe" handshaking control lines may be selected, or more complete bidirectional and bit-programmable modes may be used. In the bidirectional mode, the handshake

Table I. Twenty-seven I/O functions support the CDP1800-series product line.

I/O Ports
CDP1851*
CDP1852
CDP1872
CDP1874
CDP1875

Memory/I/O Decoders
CDP1853
CDP1858
CDP1859
CDP1866
CDP1867
CDP1868
CDP1873

UART
CDP1854A*

Multiply/Divide
CDP1855*

Buffers
CDP1856
CDP1857

Video Control
CDP1861
CDP1862
CDP1864
CDP1869
CDP1870
CDP1876

Keyboard Interface
CDP1871

Timer Functions
CDP1863
CDP1878*
CDP1879*

Interrupt Control
CDP1877

*Discussed in detail in this paper.

Reprint RE-26-8-3 ST-7023
Final manuscript received July 15, 1981.

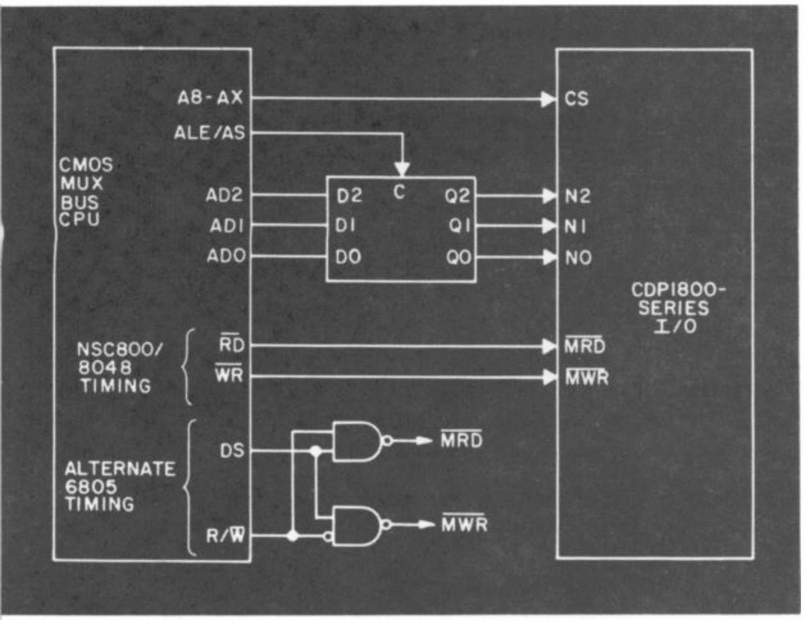


Fig. 1. General interface requirements. CMOS multiplexed-bus CPU is connected to CDP1800-series I/O.

control lines maintain proper bus-flow discipline and allow the CDP1851 to interface to a master or slave bidirectional bus. In the bit-programmable mode, individual lines can be designated as input or output lines, including the handshake pins, which are not used for ready and strobe in this mode.

Separate interrupt lines are available for each port, with OR-tie capability for single interrupt CPU inputs. Interrupts may be enabled or disabled, and can be read from an on-board status register. In the bit-programmable mode, interrupts are generated by logic conditions (AND, OR, NAND, NOR) programmed on bit inputs. Thus, a bit-programmable port can act as an interrupt expander in the OR mode, or can respond to the coincidence of several input conditions in the AND mode.

The CDP1851 interfaces, without additional components, to all CDP1800-series processors in either I/O or memory space. In I/O space, the N lines are connected directly to CDP1851 inputs, with different N-code combinations selecting the CDP1851 registers and ports (see Table III). In memory space, an on-board latch is provided to create a chip-select from any selected high-order address line on the CDP1800 multiplex bus.

Figure 2 illustrates an application that uses the complex capabilities of the CDP1851 in a multiprocessor application. In this configuration, one or more CDP1851 devices can interface to a CPU or shared master memory. Port A is used as a bidirectional port, with handshake lines controlling bus transfer. Port B is configured as a bit-programmable port, and is used to accept interrupt requests from the master controller for proper sequencing and placement of data transferred by the CDP1851. Because master bus interfacing is done under interrupt control, individual slave CPUs can perform independent dedicated tasks and ultimately increase total system throughput and performance.

Table II. CDP1851 programming modes.

Mode	(8) Port A Data Pins	(2) Port A, Handshaking Pins	(8) Port B Data Pins	(2) Port B, Handshaking Pins
Input	Accept input data	Ready, Strobe	Accept input data	Ready, Strobe
Output	Output data	Ready, Strobe	Output data	Ready, Strobe
Bidirectional (Port A only)	Transfer input/ output data	Input handshaking for Port A	Must be previously set to bit-programmable mode	Output handshaking for Port A
Bit-Programmable	Programmed individually as inputs or outputs	Programmed individually as inputs or outputs	Programmed individually as inputs or outputs	Programmed individually as inputs or outputs

Table III. CDP1851 I/O space-register assignment.

N Line Code		Instruction	Action	Register
N0	N1			
1	0	INP 1	READ	STATUS REGISTER
1	0	OUT 1	LOAD	CONTROL REGISTER
0	1	INP 2	READ	PORT A
0	1	OUT 2	LOAD	PORT A
1	1	INP 3	READ	PORT B
1	1	OUT 3	LOAD	PORT B

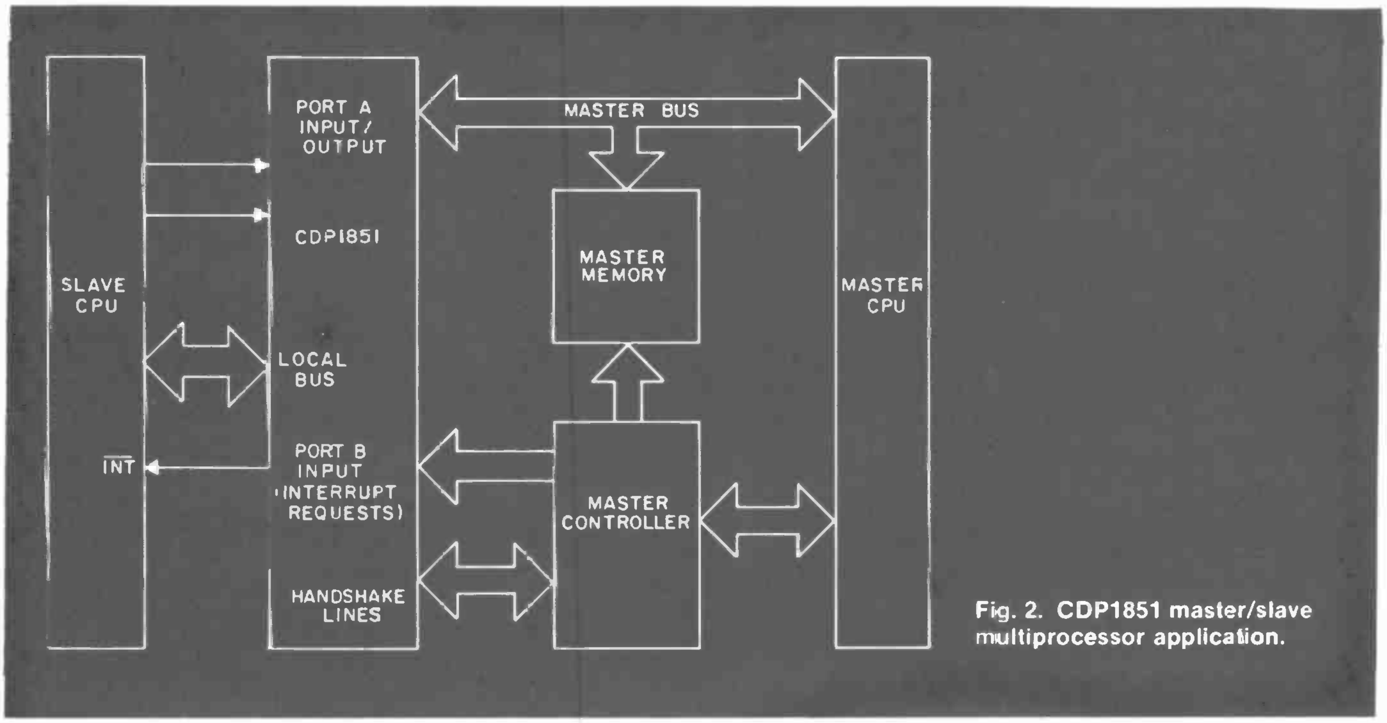


Fig. 2. CDP1851 master/slave multiprocessor application.

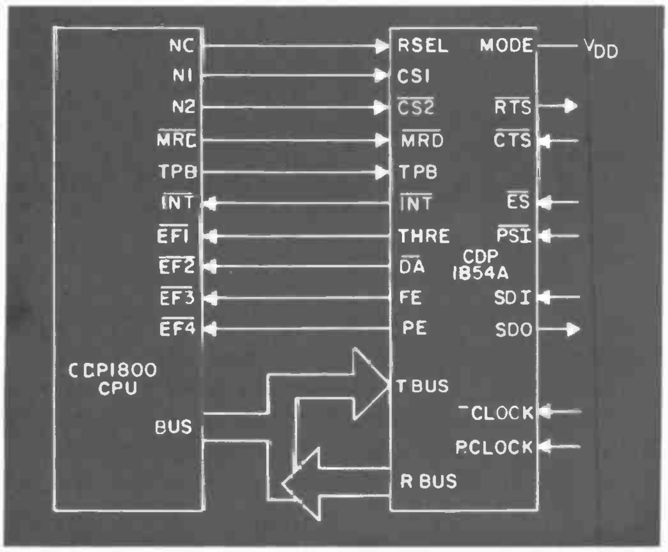


Fig. 3. CDP1854A Mode-1 interface to CDP1800-series CPU.

CDP1854A, UART

The CDP1854A is a CMOS Universal Asynchronous Receiver/Transmitter (UART) circuit.[3] It provides the necessary formatting and control for interfacing between parallel and serial data paths. The circuit operates half or full-duplex, has a double-buffered receiver and transmitter section, and features a programmable data word, parity bit, and stop-bit length. The receiver can check parity and the occurrence of a stop bit with parity, overrun, and framing-error output indication.

The CDP1854A can be programmed to operate in one of two modes through the use of a single control pin. In Mode 1, it is fully CDP1800 compatible, as shown in Fig. 3, and features programmability options made available through a control register that is accessed with N lines. In Mode 0, the CDP1854A is compatible with industry-type 1602 devices, and uses hard-wired external pins for data formatting and control.

The CDP1854A is useful in any serial communication circuit where hardware parallel-to-serial or serial-to-parallel operation is desired. Performance is superior to equivalent bit-banging software techniques, with typical data rates in excess of 250 kbits per second possible.

This UART device can provide an interface in loosely coupled systems to other processors, with transmit and receive UARTs isolating local buses from each other — the diagram in Fig. 4 shows such an application. In the configuration shown, the CDP1802 acts as an intelligent controller in interfacing a master system to a print buffer, and frees the host CPU for other tasks during the spooling operation. In the figure, the CDP1854A is configured in Mode 0, with handshaking to both the CPU and printer for data transfer. As data is received by the UART into its receiver holding register, it flags the CDP1802 for transfer to buffer memory under CPU DMA (direct memory access) control. The CDP1802 also polls the printer for data transfer from buffer memory to printer through the transmit section of the same UART. Since the receiver and transmitter functions are independent, the data exchange rates can match those of both the higher-speed host system and the slower printer.

CDP1855, MDU

The CDP1855 is a Multiply-Divide Unit (MDU) that can be an efficient hardware replacement for the software-only implementation of arithmetic and signal-processing

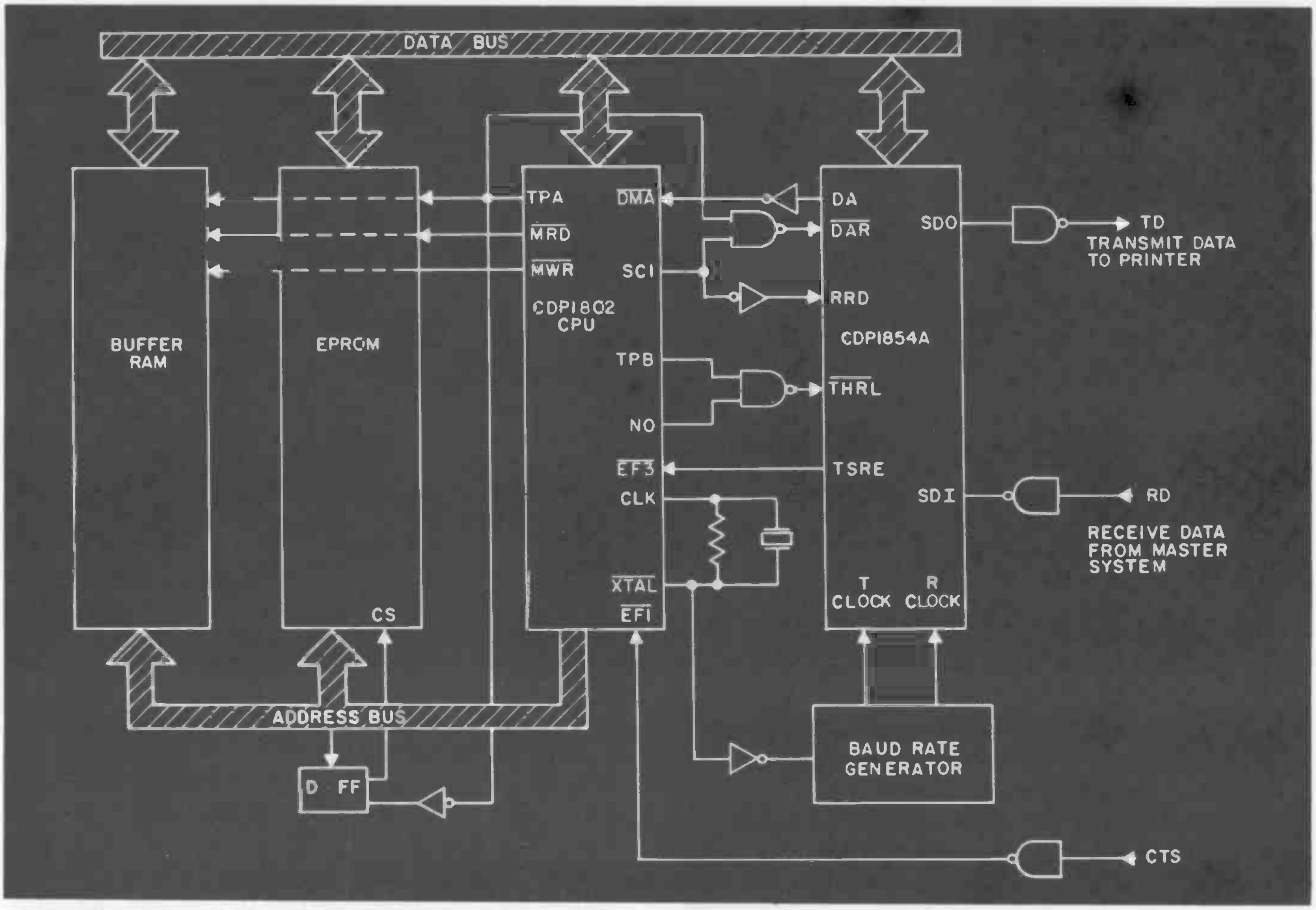


Fig. 4. CDP1802/CDP1854A slave-controller application—printer-buffer interface.

algorithms.[4] It performs multiplication and division operations on unsigned 8-bit data with an add-and-shift hardware implementation, and permits cascading of identical units to handle operands of up to 32 bits.

The MDU has three 8-bit registers — X, Y, and Z — that are loaded by the CPU with operands prior to the arithmetic operation, and which contain a product or quotient when the process is complete. The actual hardware operation typically requires only five microseconds for an 8-bit calculation, with additional software overhead time for loading and unloading registers. An 8-bit control register defines and initiates the operation, with a status register available for overflow indication.

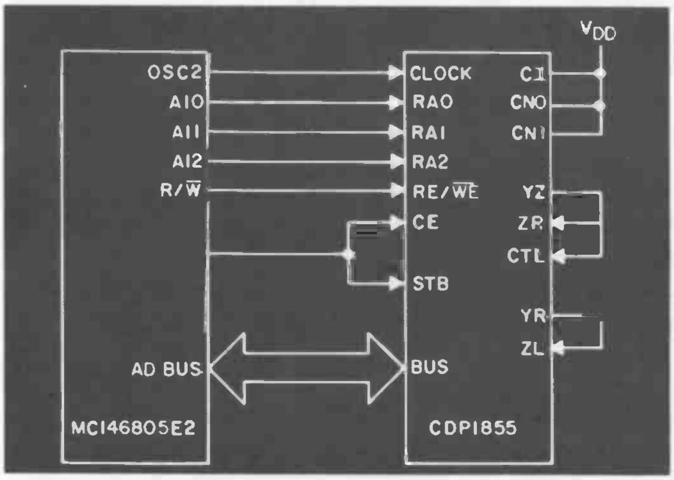


Fig. 5. Minimum 6805/CDP1855 memory-mapped (in 4k) interface.

The MDU can typically be mapped in I/O space, with eight instructions required to address its X, Y, Z, control and status registers. The device also easily maps into the memory space of other processors, as shown in Fig. 5, where an MDU is interfaced to an MC146805E2.

The CDP1855 can be used in any application that requires fast multiply or divide throughput, or where the CPU would be required to perform real-time tasks during a long arithmetic routine. MDU efficiency increases as word size increases, as shown in the chart of Table IV. In addition, in low-frequency signal-processing applications, the MDU is suitable for use as a recursive digital filter in conjunction with A/D and D/A conversion circuitry.

CDP1878, counter-timer

The CDP1878 is a dual 16-bit counter-timer with a variety of operational modes.[5] It is a general-purpose device that

Table IV. Software versus hardware multiply-time comparisons for the CDP1855 MDU.

Word Size	Approximate Time in Machine Cycles	
	Software	Hardware
8 x 8	280	26
16 x 16	1000	41
32 x 32	5000	89

can produce an assortment of output formats usable in real-time or control-oriented applications. It handshakes with the CPU through an interrupt line that is independent of the timer outputs. This mechanism provides efficient operation with a minimum of software overhead, and without compromising the wave shape required by an external device.

Each section of the CDP1878 consists of a 16-bit programmable down counter with a separate control register, programmable-level gate, true and complement outputs, and a maskable interrupt request that can appear on a shared output pin and in a status register.

In operation, the user jams a desired counter value in two 8-bit sequences, and then selects the desired counter mode and initiates timing by writing to the control register. This control register, which programs gate level and interrupt enabling, can also be used to stop the counter at any time, and can assure a stable counter readout by freezing the present count into a separate holding register.

The five counter modes of the CDP1878 are shown in Table V along with typical applications for each. These modes, along with different combinations of gating levels, output polarity, and underflow interrupt indication, provide a complete array of timing, pulse-forming, and event-counting programming for efficient use of the device in system designs.

The CDP1878 can be mapped into CDP1800 I/O space, or into the memory space of CDP1800 or other general-purpose processors by means of an external control pin. Fig. 6 shows a typical application for the device, which, in

Table V. CDP1878 counter-timer's modes of operation.

	Name	Function	Application
1	Timeout	Outputs change when clock decrements counter to zero.	Event counter
2	Timeout strobe	One clock-wide output pulse when clock decrements counter to zero.	Trigger pulse
3	Gate controlled one-shot	Outputs change when clock decrements counter to zero. Retriggerable	Time-delay generation
4	Rate generator	Repetitive clockwide output pulse	Timebase generator
5	Variable duty cycle	Repetitive output with Programmed duty cycle	Motor control

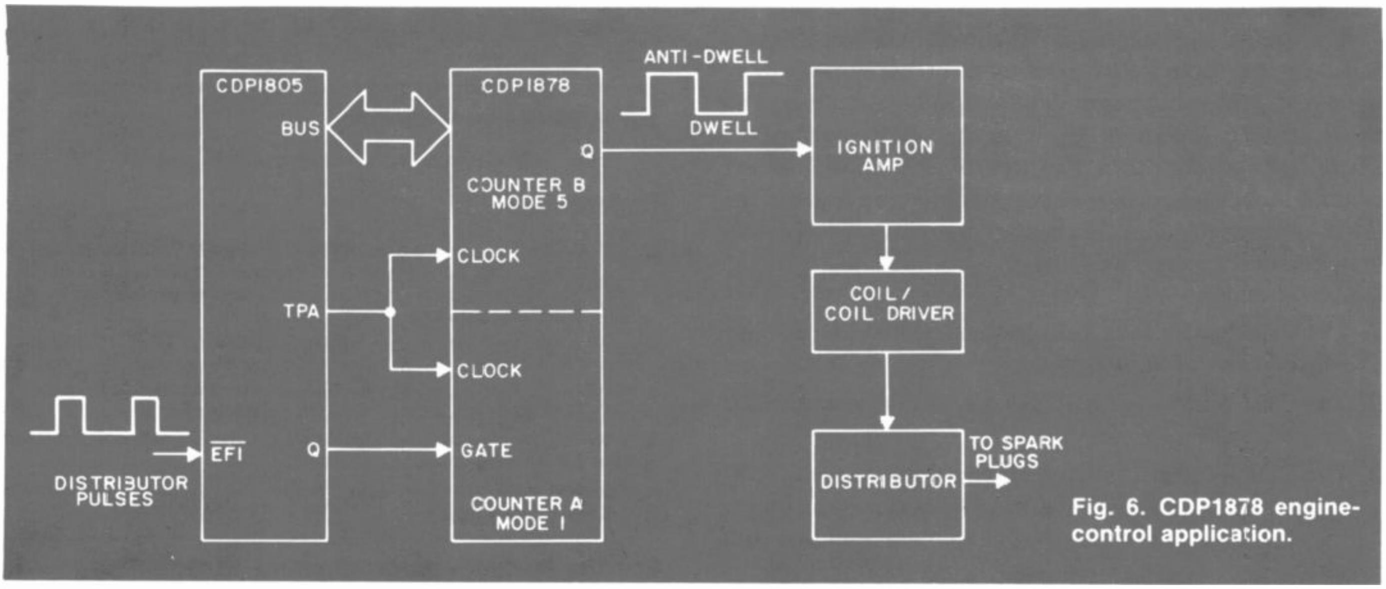


Fig. 6. CDP1878 engine-control application.

the figure, is mapped in the I/O space of the CDP1805. The circuit provides a means for generating spark advance and dwell timing based on engine speed and a lookup table of engine constants for various load conditions. Half of the CDP1878 is used to monitor engine speed as a high-resolution, 16-bit count based on time period, while the other half outputs a variable-duty-cycle phase-shifted pulse to the ignition circuitry, with proper formatting for correct dwell and spark-advance settings.

CDP1879, real-time clock

The CDP1879 is a time-of-day clock/calendar chip useful in a variety of real-time applications.[6] The device counts seconds, minutes, hours, date, and month. It operates with a variety of crystal frequencies, has a separate clock output, and features an interrupt alarm that has a one-second resolution with a 24-hour period.

This real-time clock device can be thought of as a programmable divider chain. One of four crystal frequencies, from 32 kHz to 4 MHz, is selected as the clock source. An on-board control register selects the appropriate prescaling to produce a one-second pulse that is fed to a chain of five programmable counters. The prescaler and divider chain can be tapped to generate 50-percent duty-cycle pulses (subsecond or one per second, minute, hour, or day) at the clock output along with an interrupt request. Counters may be written to or read from in BCD format through individual addresses. Special circuitry allows reading "on-the-fly," even if the counter chain is rippling through a clock pulse at the instant a read attempt is made. Separate second, minute, and hour alarm registers generate an interrupt request when their values match those of the counters. A status register keeps track of alarm and clock status when interrupts are disabled.

The CDP1879, like the CDP1878, interfaces in I/O space to all CDP1800 CPUs, as well as in memory space to CDP1800 or other general-purpose processors, through a single control pin.

A powerful application for this real-time clock is as a wake-up control to a CPU that reduces total system power in intermittent-use systems. A hookup diagram illustrating this feature is shown in Fig. 7. In this configuration, the alarm and power-down features of the CDP1879 are utilized in the control of the sleep and wake-up states of the CPU. A typical shut-down/start-up sequence for this system could proceed as follows:

1. The CPU has finished a current task and will be inactive for the next six hours.
2. The CPU loads the CDP1879 alarm registers with the desired wake-up time.
3. The CDP1800 Q output is set high, which stops the CPU oscillator. As an alternative, in an NMOS system, power to all components except the clock chip could be shut off.

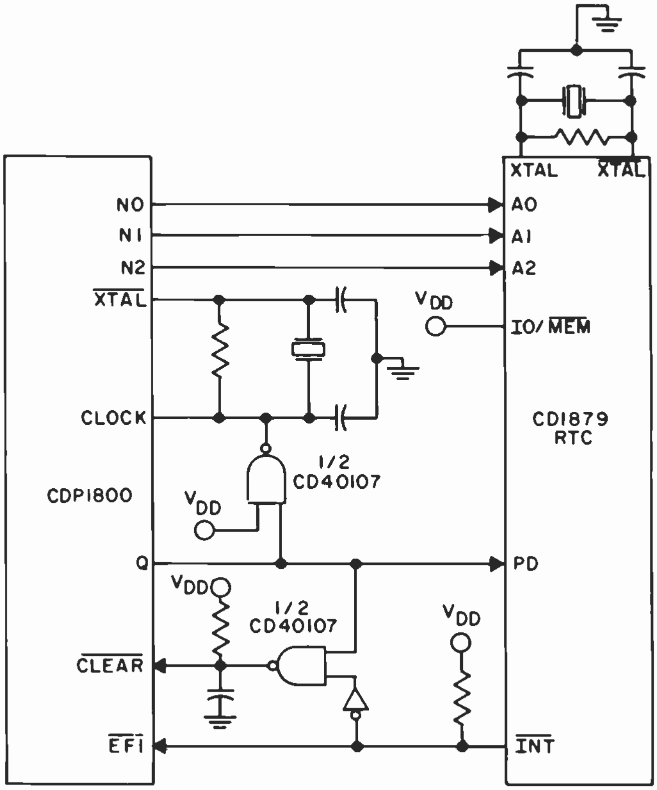


Fig. 7. CPU wake-up circuit using the CDP1879 real-time clock.

4. This Q-output signal is received by the CDP1879 as a power-down signal.
5. The CDP1879 tri-states all pins (to accomodate powered-down chips).
6. The CDP1879 eventually times out, and sets an alarm by driving the $\overline{\text{INT}}$-output low.
7. The alarm signal resets the CPU (to avoid oscillator start-up problems) and flags the processor for a warm-start routine.
8. The CPU, once into its normal software sequence, writes to the CDP1879 control register to reset the interrupt request.

Because of the versatility of the CDP1879, it is not restricted to use with CMOS processors. Any processor capable of writing to and reading from the clock chip can use its low-power capability.

References

1. "CMOS 8-Bit Microprocessor," CDP1802A, CDP1802AC, RCA Solid State File No. 1305. A general discussion of CDP1800-series I/O structure is contained in "CDP1800 Peripherals: Building Blocks of a Complete Processor Family," RCA Solid State Technical Paper ST 7023.
2. "COS/MOS Programmable I/O Interface," CDP1851, CDP1851C, RCA Solid State File No. 1056.
3. "COS/MOS Programmable Universal Asynchronous Receiver/Transmitter (UART)," CDP1854A, CDP1854C, RCA Solid State File No. 1193.
4. "8-Bit Programmable Multiply/Divide Unit," CDP1855, CDP1855C, RCA Solid State File No. 1053.
5. Data sheet in preparation.
6. Data sheet in preparation.

See the author's biography and photo on page 22.

J.K. Wright, Jr.

CLIP-3 is a high-level controller language for 1802 microcomputers

We can expand the area of microprocessor applications by making microprocessor systems easier and less expensive to use. CLIP-3 is another step in that direction.

Abstract: *As software development becomes the predominant cost and time factor in designing microprocessor-based systems, the need for methods to reduce software costs becomes more urgent. High-level languages represent one solution in the effort to overcome this problem, but many small microprocessor systems, such as controllers, cannot easily support a full high-level language due to the larger memory requirements and development system costs. This article describes a high-level hexadecimal controller language — CLIP-3 (Control Language Interpreter Program) — designed to use minimum memory, to be easy to use, and to require a minimum cost development system. The language has been implemented for a specific* 1802*-based controller board having* 16 *ON/OFF inputs,* 16 *ON/OFF outputs, a* 60-*Hz clock on the interrupt line, and* 2 *or* 4 *kbytes of program memory (it should be adaptable to other* 1802 *microcomputers having similar features).*

When microprocessors were first introduced, many engineers were very excited about the new chips' ability to replace random logic hardware with a program written by the designer to perform the same functions. This would mean that, as the system hardware design progressed, the program could be "easily" changed to iron out design bugs or add features. At least, the changes were easier than constantly redesigning the random logic and rewiring the board. However, gaining these advantages meant that one had to learn the machine (or assembly) language of the processor being used, and also had to buy and learn to use a suitable development system. In lower-cost, controller-oriented microprocessor applications, the high cost of the development system and the time required to write and debug the program can make many possible applications difficult to justify. If a higher-level program language could be designed — easy to learn and not requiring a high-cost development system to use — many new microprocessor applications would open up and others would dramatically drop in cost.

Emphasis is on higher-level program languages

High-level languages reduce the amount of time and cost required to generate a given application program. It has been shown that an average programmer can program approximately ten lines of debugged code per day no matter what the language, and a higher-level language results in fewer lines of code (examples of some typical high-level languages are BASIC, FORTRAN, and Pascal). Also, the resulting program is more reliable and easier to debug since it has fewer instructions and is easier to understand. Program maintenance is also less costly for the same reasons. One trade-off in using a higher-level language is increased execution time of the high-level program over an equivalent machine-language version. However, all that is really necessary is that the execution time be adequate for the application being programmed. In I/O-intensive applications such as controllers, speed of execution determines the delay between the last known picture of the system as seen by the controller inputs and the actual picture of the system when control is subsequently applied to it. This delay can be on the order of 2 to 10 milliseconds without causing problems for most applications.

Another trade-off in using a high-level language is that the high-level language system requires more memory. This is becoming less of a problem as memory costs continue to decline. Also, designing a high-level language specifically for control applications reduces the overall memory requirements from that required for a more general language.

Reprint RE-26-8-4
Final manuscript received June 8, 1981.

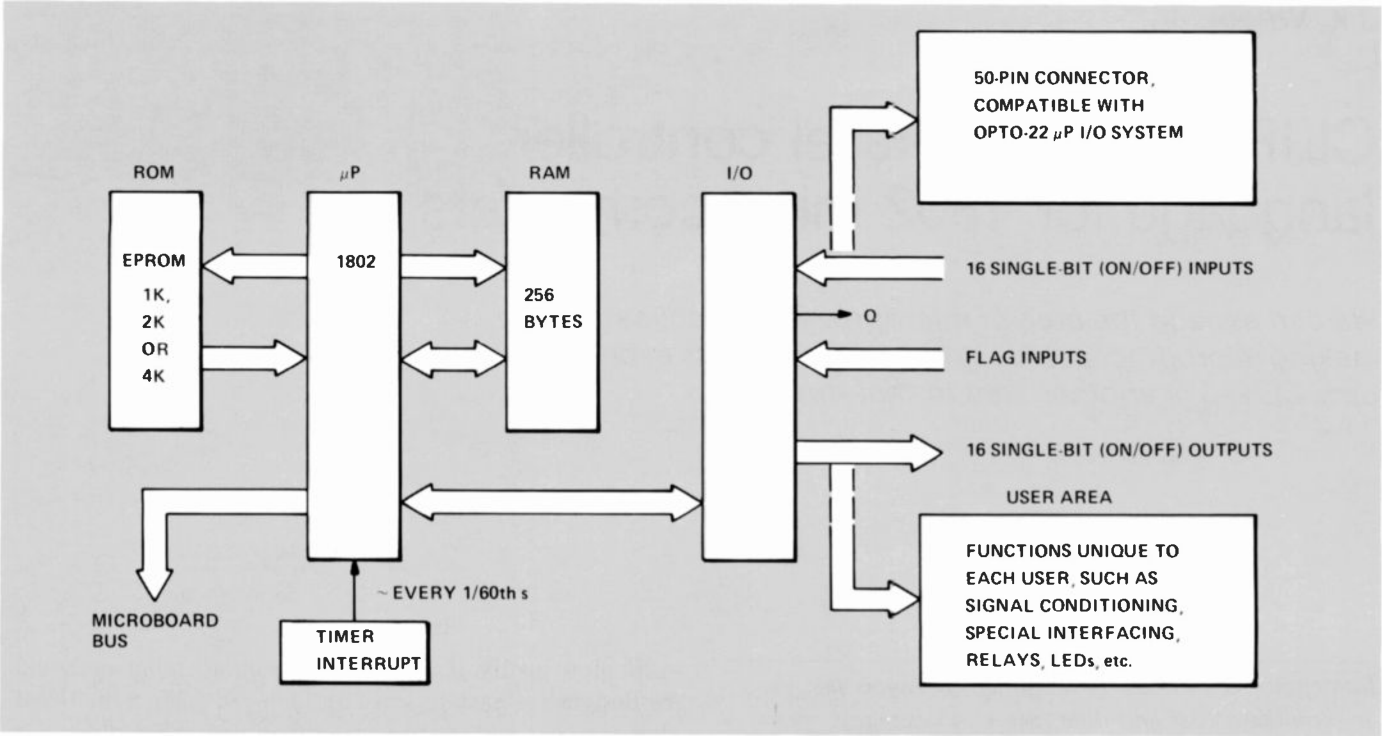


Fig. 1. Block diagram of an 1802-based controller. The control interpreter and the user's program are resident in EPROM, and 16 single-bit inputs and 16 single-bit outputs are available for connection to the outside world.

In most cases, the more general-purpose we make a language or a system, the harder it is to use; and conversely, the more special-purpose we make it, the easier it is to use. We have, therefore, concentrated our language-design efforts in a specific limited applications area; that is, control applications that can be performed by a single-board 1802-based computer with 16 ON/OFF inputs and 16 ON/OFF outputs. These applications do not require extensive mathematical processing of the input data or extensive storage of the input data. Because of the general observation that there are many more applications for lower-cost processor systems than for the sophisticated higher-cost systems, we expect that limiting and focusing our controller language in this way will not excessively limit the number of applications for it. In fact, as mentioned earlier, the lower cost of the processor system and the ease of use of the language should open up even more applications.

Applications

Typical applications of our controller and language include intelligent sequencers of all kinds, automatic sorting machines, component or system testers, conveyor controllers, operation counters, robot controllers, or any control problem that requires logical decision, counting, or timing functions. CLIP-3 would have applications in a wide range of industries, including food and drug processing, film processing, glass works, packaging, paper and cardboard, printing and publishing, plastics, textile, and so on. All these applications involve many decision-oriented tasks that result in simple ON/OFF commands. For example: Is limit switch closed? Has timer interval ended? Has counter reached XXXX? When inputs A, B, and C are on, start motor M3.

A more specific example of a controller application would be a plastic parts sorter that sorts the parts by color. The controller program analyzes color signals from three photodiodes capped with filters to determine each part's color, then it directs each part into the appropriate bin. Another example is the controller's use as a system or component life-tester. Controller output lines turn the system or component on and off and provide input signals to it as necessary, while the controller input lines check it for proper functioning, and count the number of cycles until failure occurs. A third example would be the controlling of a game-token press, where controller outputs would operate the coin-blank feeder, the coin press, and the packaging equipment, while controller inputs would sense if a blank was positioned in the press, and if the boxes and cartons were positioned properly for loading. Other outputs could provide an alarm signal for various equipment failure modes, and could clock a counter display to show how many tokens had been packaged. Yet another example would be the control of an automatic press brake; as the punch penetrates the metal workpiece, the controller senses the force expended during penetration, and tells the machine precisely when to reverse its motion for the best result. We thus get an improvement in both quality and speed as well as reduced waste. All of these areas of application are increasingly important today as companies work to improve their productivity.

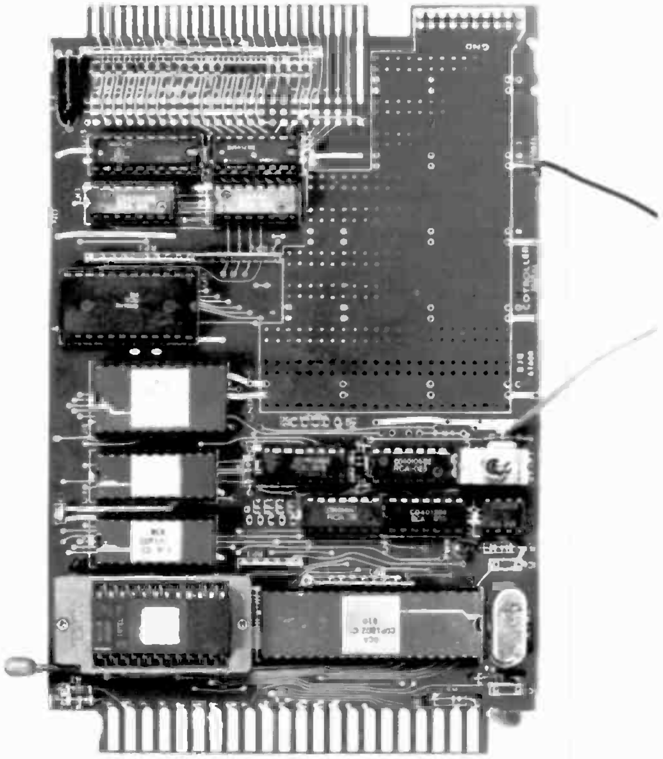


Fig. 2. The entire controller shown in Fig. 1 fits on one small PC board.

An 1802-based controller that would handle applications in these areas was designed, and its general structure is shown in Fig. 1. A photograph of the controller is shown in Fig. 2. It was designed hand-in-hand with the high-level controller language in an attempt to reduce the language's memory requirements and increase the execution speed. The low cost and small size of the board mean that should a failure occur, quick replacement by a duplicate board is a feasible solution. The small size also makes it easier to isolate the board from environmental extremes and from external noise. For a typical controller application, the 1802 controller board, including the user's program in EPROM, should cost less than $100, and the program itself will have required much less time to write, debug, and maintain than if it had been done in machine language.

The 1802 microprocessor was used because it lends itself readily to writing high-level languages in the form of interpreters. An interpreter is a special program in memory that "interprets" or executes the statements of the user's program when the user's program is run. The best-known and most widely used interpreter is the one used to run the BASIC language. If the user typed in the BASIC statement, "LET A = B," for example, the interpreter program would analyze the statement when the program was run and then perform the operation. In this case, it would determine that this is a "LET" command, and would then set A equal to the present value of B. It would then go on to the next statement and interpret it, and so on. In general terms, we can think of an interpreter as actually changing the characteristics of a processor so that it will look like a "higher-level" processor; this then executes commands in the higher-level language rather than in machine language (Fig. 3). The new 1804 microprocessor, with its internal ROM, can be made to look like a higher-level processor by programming this ROM with an interpreter program.

In designing the control language, a limited version of BASIC was considered as a possible approach. This has been useful in several applications (see References 1, 2, 3),

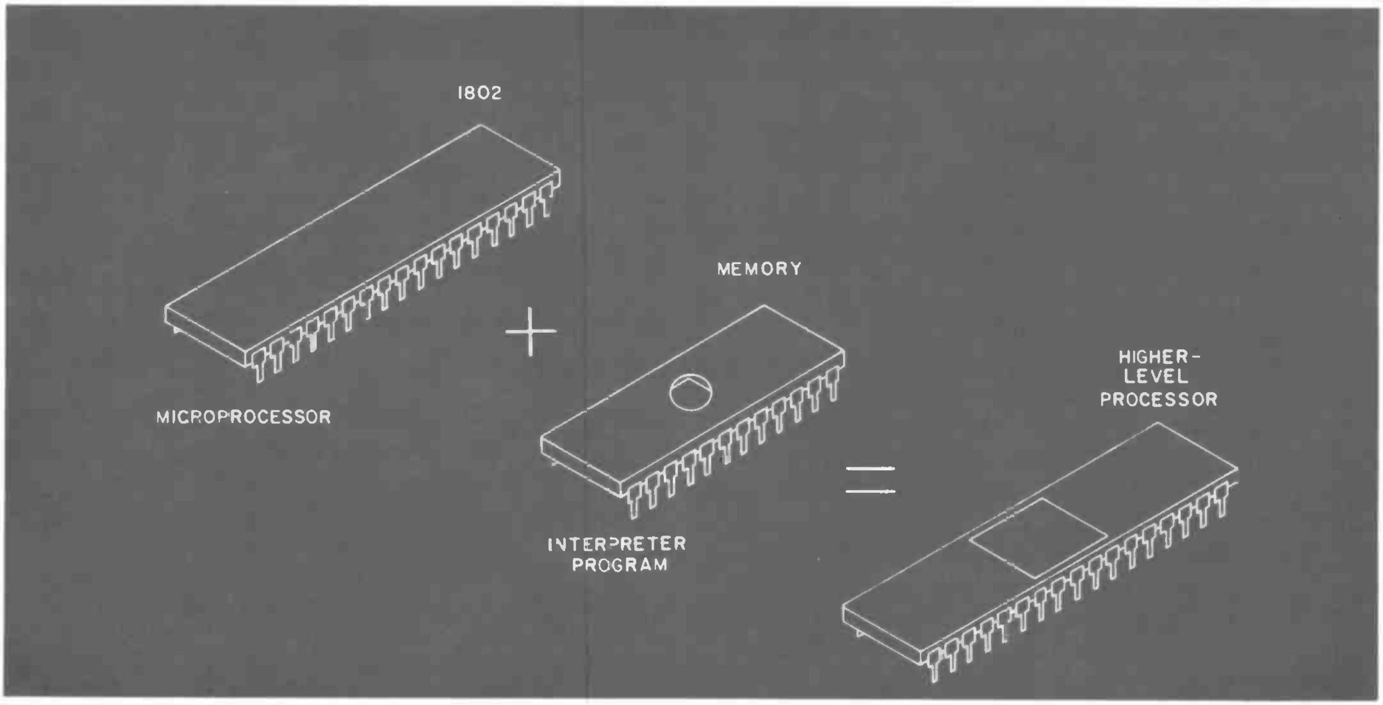


Fig. 3. An interpreter can be thought of as actually changing the characteristics of a microprocessor, producing a new "higher-level" processor.

Table I. General instruction summary for CLIP-3.

1. Turn specified Output Line ON or OFF.
2. Turn specified Output Line ON or OFF until specified Input goes ON or OFF.
3. Turn specified Output Line ON or OFF until specified Delay Expires.
4. Delay by a specified Time (this time may be specified in the instruction or may be specified in a register).
5. Delay until a specified Input goes ON or OFF.
6. Set any VX Register.
7. Add two VX Registers.
8. Subtract two VX Registers.
9. Set any ZX Register.
10. Skip the next N instructions depending on a specified Input or on the value of any ZX Register.
11. Execute a specified Control Language Subroutine (from one to 255 times).
12. Execute a specified Machine Language Subroutine.
13. Set a Time Limit on how long a Control Language Subroutine can be executed before an automatic exit is caused.
14. Let ZX indicate which one of several Control Language Subroutines is to be executed.
15. Stop.

but has disadvantages. It is weak in the I/O (Input/Output) area, requiring the user to write machine-language routines for most of the controlling and sensing functions his program will perform. Also, its execution speed is typically very slow, which eliminates it from several application areas. In an effort to eliminate these disadvantages, a new control language was designed using techniques similar to those used in the RCA VIP's CHIP-8 language, a higher-level language used to write video game programs. The control language was named CLIP-3 (Control Language Interpreter Program), and, like BASIC, is an interpreter-based language. It runs over 10-times faster than BASIC and requires less than half the memory of a typical "Tiny BASIC" interpreter (typical "Tiny BASIC" = 2 kbytes).

Control language description

A general summary of the CLIP-3 instruction set is given in Table I. For low-cost control applications, the most important functions that a language should have are timing, counting, and ON/OFF control and sensing. The CLIP-3 interpreter creates a controller structure that includes sixteen 16-bit registers, identified by the name VX, where X is a hexadecimal number between 0 and F (in other words, the 16 registers are V0, V1, V2,... VF). These registers can be used as timers, counters, or to hold input or output data. There are also sixteen 8-bit registers, called ZX, where X again ranges from 0 to F. These are used chiefly as indicators — they can be set in one part of the program and then checked in another part to determine which of several choices for action should be made. One (Z0) is dedicated as an easy-to-use subroutine loop counter, and another (ZF) is used as an overflow indicator. A typical instruction is composed of two hexadecimal bytes, although several instructions requiring long arguments are longer than two bytes.

Although having the instructions in the form of hexadecimal bytes does reduce the readability of a user's program, this format greatly reduces the program's memory requirements and increases its execution speed over a language such as BASIC. It also allows using an inexpensive VIP as a development system for transferring programs from paper to EPROM. A more sophisticated development system would allow the user to type in alphanumeric commands that would be translated by the development system software into the CLIP-3 hexadecimal instructions.

The CLIP-3 instructions

Setting and manipulating the VX registers

6K8N
DDDD
EEEE
•
•
•
•

Starting with VK, load the next N 16-bit hexadecimal data words (starting with DDDD) into the VXs.

In other words, load DDDD into VK, load EEEE into V(K+1) and so on, until N VXs have been loaded.

6X4Y Add two VXs: VX = VX+VY. ZF = 01 if overflow, else 00. ZF can be checked by a 3N4X or a 3N9X instruction (see page 34).

6X2Y Subtract two VXs: VX = VX−VY. ZF = 00 if result is negative, else 01.

Setting the ZX registers

5XKK Set ZX = KK, where KK can be any number between 00 and FF. Note: Z0 is dedicated as a control language subroutine loop counter (see subroutine instructions), and ZF is used as an overflow indicator in the **6X4Y** and **6X2Y** instructions.

Subroutines

□ *Calling a subroutine*

2MMM Do CLIP-3 subroutine at 0MMM. 0MMM is the hexadecimal address of the first instruction in the subroutine.

1MMM Same as 2MMM except that the subroutine return skips the 2-byte instruction following 1MMM. This instruction saves memory when you want to do either one or the other of two subroutines but not both, based on some condition. The second subroutine call would follow the 1MMM.

The next instruction is used before calling a subroutine to specify how many times the subroutine is to be done. If you only want to do the subroutine once, then it is not necessary to use this instruction at all (that is, default is 01).

50KK Do the next subroutine KK times (KK = 00 through FF). This instruction uses Z0.

Any subroutine can call other subroutines (this is called "nesting"), and subroutines can be nested to ten levels. The 50KK instruction can also be used within subroutines or nested subroutines.

Although seldom necessary, machine language subroutines can be called by using the next instruction:

0MMM Do machine language subroutine at hex address 0MMM.

□ *Writing a subroutine*

Subroutines can be written using any of the normal CLIP-3 instructions, but each subroutine *must* have one of the following exit or looping instructions:

00AA Exit the subroutine (return to the calling program).

00AE Check if the subroutine has been done the number of times specified by the 50KK instruction. If it has, exit the subroutine. If it has not, go back and repeat the subroutine again.

00A6 Go back to the first statement of the subroutine.

D4 Exit the subroutine — used only by machine language subroutines.

Timing

FFFF
DDDD Delay by DDDD.

FF8X Delay by contents of VX.

The hexadecimal number DDDD or the contents of VX can range between 0001 and FFFF, which corresponds to delays between about 1/60th of a second and 18.2 minutes. Longer delays can be realized by stacking instructions.

FF7Y Delay until Input Y = OFF (Logic 0). Y designates one of the 16 single-bit inputs.

FF3Y Delay until Input Y = ON (Logic 1). Y designates one of the 16 single-bit inputs.

In the above instructions, the processor can do nothing while the delay is being counted down. There are many cases where you would like to set a timer and then have the processor continue to execute part of the program while the timer is running. When the timer expires, the processor will be notified to stop the present program and go on to the next one. For instance, you may want the processor to watch a two-state sensor or a process for a fixed time, and then do one of two things, depending on the sensor output or process result. This can be done with the SET TIMER instruction:

0054 Set special timer to TTTT.

FMMM Then immediately do this subroutine at 0MMM. If this subroutine is exited *before* the timer expires, skip the next 2-byte line. If the timer expires before the processor exits the subroutine, the processor is forced to immediately exit the subroutine and to do the next statement.

As in the previous timing instructions, TTTT can vary between 0001 and FFFF (1/60th second to 18.2 minutes). Timers can be nested (that is, the subroutine at 0MMM can also use this instruction) to a maximum of four levels. The 50KK instruction can be used first to set the number of times the subroutine is to be repeated during the timer interval.

This instruction also allows putting a maximum time limit on all "until" instructions to keep the processor from waiting too long for the specified condition to occur. For example, when using the instruction "Delay until Input Y = ON," if Input Y never changed to ON, the processor would "hang up" in that instruction indefinitely. By setting the timer first, the processor would be forced out of this instruction when the timer expired. *The statement following FMMM (done after a forced exit) should turn off any output lines (see ON/OFF control instructions) turned on by the FMMM subroutine if that subroutine would have turned them off before a normal exit*. A normal exit is one in which the subroutine is exited *before* the timer expires.

A requirement to toggle an output or perform some sequence repetitively for a fixed time can also be handled by this instruction. A subroutine to perform the required sequence is first written using the subroutine looping instruction 00A6. The subroutine is then called as follows:

0054 Set Timer = TTTT.
TTTT

FMMM Call subroutine.

XXXX Next instruction in main program.

•
• Program continues.
•

The subroutine will now be repetitively executed until the timer interval TTTT has expired and then the instruction XXXX will be executed.

ON/OFF control

In these instructions, Q designates one of the 16 single-bit outputs of the controller card. At program reset, all 16 output bits are automatically zeroed by the hardware. ON = Logic 1, and OFF = Logic 0 (ground).

9Q00 Set Output bit Q to ON.

9Q7Y Set Output bit Q to ON until Input Y = OFF.*

9Q3Y Set Output bit Q to ON until Input Y = ON.*

9Q8X Set Output bit Q to ON until Delay VX has expired.

9QFF DDDD Set Output bit Q to ON until Delay DDDD has expired.

8Q00 Set Output bit Q to OFF.

8Q7Y Set Output bit Q to OFF until Input Y = OFF.*

8Q3Y Set Output bit Q to OFF until Input Y = ON.*

8Q8X Set Output bit Q to OFF until Delay VX has expired.

8QFF DDDD Set Output bit Q to OFF until Delay DDDD has expired.

0123 Set all Output bits to OFF.

* Input Y is checked first and if it is already in the specified state, Output bit Q is not changed.

Conditionals

In these instruction descriptions, one line = 2 bytes or 4 hex digits (usually one instruction).

3N1Y If Input Y = OFF, skip next N lines.

3N2Y If Input Y = ON, skip next N lines.

3N4X If ZX = 0, skip next N lines.

3N9X If ZX ≠ 0, skip next N lines.

3N90 Skip next N lines (since Z0 is always ≠ 0).

For all the above instructions, if N = 0, it is equivalent to a CONTINUE statement.

Since the first two instructions above each take about 220 microseconds or more to execute, the maximum "sampling frequency" of checking inputs is about once every 220 microseconds (with a 2-MHz microprocessor clock).

"CASE" instruction

CX00 If ZX = J, do

JMMM CLIP-3 subroutine at 0MMM, else if ZX = K, do

KNNN CLIP-3 subroutine at 0NNN, etc.

•   •

•   •

•   •

0PPP else if ZX = none of the above, do CLIP-3 subroutine at 0PPP. If this line is 0000, it means, "end of instruction, continue."

This instruction is similar to the Pascal "CASE" instruction; it decides which of several subroutines to do based on the value of ZX. The X in ZX is specified by the first part of the instruction, the CX00. The next line of the instruction, JMMM, says that if ZX = J, do the subroutine at address 0MMM. Appropriate subroutines to be executed for other values of ZX (1 through F) can be added next in the same format (the ZX value is the first digit; the subroutine address is the last three digits). The last line of this instruction *must* have 0 as the first digit. If the remaining three digits are also 0, it means "end of instruction, continue." If the remaining digits are not 0, they are taken as the address of a CLIP-3 subroutine to be executed for any other values of ZX not specified in the earlier part of the instruction.

Miscellaneous

011E STOP
The last instruction in the main program must be a "STOP" instruction.

EMMM Go to Address 0MMM.
If you want to write "STRUCTURED" programs that are easier to debug and to understand later, use this instruction as little as possible.

The procedure for using CLIP-3 in a typical application is as follows:

□ Decide on the input and output lines to be used on the controller board. Interface these to the system to be controlled.

□ Write the control program in CLIP-3.

□ Load the CLIP-3 interpreter and your program into the development system memory and transfer to a standard 2716 or 2732 EPROM. The memory map in Fig. 4 shows the memory map of the EPROM. For faster program development, the Intel 2816 EEPROM can be used (it doesn't have to be UV-erased every time a program change is to be made).

□ Insert EPROM in controller board and turn on power.

To show how the control language would be used, let's write a short program to transfer the register V2 to a LED display in decimal. The contents of V2 might represent a count of the number of items rejected or sorted by the controller, for example. First, we connect a counter-display chip, such as the Intersil ICM7217, to a controller output line, say Output #1. This single inexpensive chip will count the pulses on the output line in decimal, and will display the result on an attached LED display. Now we write a subroutine to count down V2 to 0, turning output line #1 ON and OFF at each count (Set V1 = 0001 first).

Subroutine:

6221: V2 = V2 − V1.

319F: If V2 is positive, skip next instruction.

00AA: Exit subroutine.

9100: Turn Output #1 ON.

8100: Turn Output #1 OFF.

00A6: Return to first statement above, that is do this subroutine again.

When this subroutine is called, it will transfer the count in V2 to the LED display.

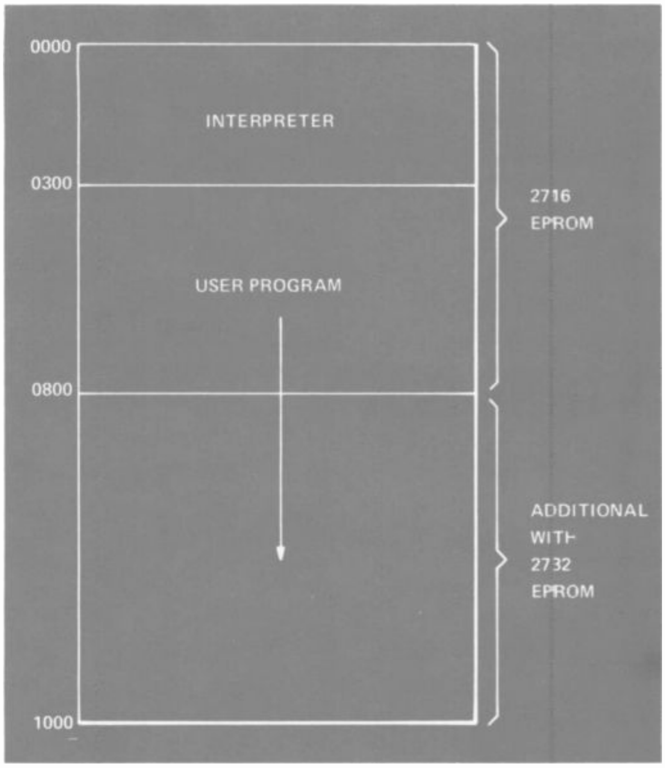


Fig. 4. The memory map of the controller's EPROM, which includes the CLIP-3 interpreter and the user's control program, written in CLIP-3.

If the CLIP-3 language is to be used with other 1802 microcomputers, all except the I/O instructions can be used unchanged (the microcomputer must have a 60-Hz interrupt generator and one page of RAM). Generally, the changes required to rewrite the I/O instructions will result in a larger interpreter and slower execution of the I/O instructions, since the controller board that CLIP-3 was written for was designed specifically for use with an interpreter, to maximize execution speed and minimize interpreter size.

CLIP-3 was also designed to allow the user to add additional instructions if required for his application. For example, if an integer multiply instruction were needed, there is room in the language framework to add this. Machine language subroutines can also be added, if necessary, to increase execution speed in time-critical parts of a user's program, although this is seldom necessary. Analog inputs or outputs can be handled by interfacing A/D or D/A chips to the normal controller inputs or outputs.

Conclusion

The CLIP-3 language has been used to control an industrial ball sorter (automatically separating steel, aluminum, and plastic balls) and a toy robot tank. The robot tank system (a modified Milton Bradley "BIG TRAK") was fitted with an 1802 controller board to illustrate the power of the CLIP-3 language and the compactness of the controller hardware. It was programmed to sense its environment, and with no human intervention, search a room for a way out without getting boxed into a corner or trapped under a chair. The control programs for both these systems were written by relatively inexperienced users who were able to learn and to use the language much faster than they would have been able to learn assembly language. The compactness of the language allowed both the CLIP-3 interpreter and the user program to fit easily into a 2-kbyte EPROM in both of these applications. Execution speed was fast enough so that no special machine language subroutines were needed in any part of the control program.

Acknowledgment

I would like to thank Joe Weisbecker, whose CHIP-8 interpreter provided the inspiration for this interpreter, and Phil Baltzer, whose support and encouragement made it possible. Thanks, also, to B.J. Call, who built the hardware and kept it running, to K. Apperson, J. Higham, M. Hyczka, and P. Madden for making the language appear easy to use, and to L. Stintnicolaas and J. Winsor for writing most of the software for the program development system.

References

1. Cushman, R.H., "1-Chip μCs, High Level Languages Combine for Fast Prototyping," *EDN* (Aug. 5, 1980).
2. Cushman, R.H., "Bare-bones Languages Serve Small Systems with Ease," *EDN* (Sept. 5, 1980).
3. "Using NSC Tiny BASIC," *National Semiconductor*. (Manual for INS8073 Tiny BASIC Microinterpreter.)

Jack Wright, a Member of Technical Staff at RCA Laboratories, joined RCA in 1974 and has since been involved in all aspects of microprocessor systems work, from design of both hardware and software to manufacturing. As a member of the LSI Systems Design and Applications Group, his major interest has been in making microprocessor hardware easier and less expensive to use. He is currently investigating ways to fully exploit the potential of the next generation of VLSI technology in system design.

Contact him at:
RCA Laboratories
Princeton, N.J.
TACNET: 226-2575

D.A. Milley|H.L. Resnick

For Army vehicles:

Memory design in a microprocessor-based test set

Automated Systems engineers ingeniously used 1802 microprocessor software and hardware to expand the Simplified Test Equipment applications data base to include a few new Army vehicles.

Abstract: *The authors describe methods for using bank-switching and logical addressing to allow an* 1802 *micro-processor to use several hundred kilobytes of EPROM. This application expands the Simplified Test Equipment (STE) applications data base to include a number of new Army vehicles. STE minimizes the difficulties of vehicle testing faced by the Army mechanic. Now STE can be used on more vehicles, because engineers at Automated Systems developed this* 1802*-based test set with over* 3/4*-million bytes of memory.*

In order for repair and support capabilities to keep pace with the increasing complexity of Army combat vehicles, RCA has developed a family of microprocessor-based field test sets. Designed with the goal of minimizing the technical difficulties of vehicle testing faced by the Army mechanic, this family has been designated as Simplified Test Equipment (STE).

In expanding this product line to capture the encoded diagnostic test procedures for a number of vehicles, including the ABRAMS M1 Tank, the Infantry/Cavalry-carrier Fighting Vehicle System (FVS), and a number of others planned in the future, a variety of strains were created on state-of-the-art hardware and software capabilities. One of these strains involved the difficulty of providing enough nonvolatile memory, on the order of 400 kbytes (kilobytes), to store the applications data base. This problem was solved with a combination of bank switching and logical addressing that allows an 1802 microprocessor to use several hundred kbytes of Erasable Programmable Read-Only Memory (EPROM).

Software system based on EPROMs

EPROM technology is attractive in this application because it is cost competitive with ROM. Also, with such a large volume of application programs and changing baseline for the supported Army vehicles, periodic software updates must be expected. EPROMs, because they are erasable, allow for convenient updating.

The software system, based on a design that uses up to 60 EPROM chips, uses an addressing scheme whereby code anywhere within the system can be referenced by a 20-bit logical address, and data local to a test being executed is referenced by a 16-bit offset pointer, relative to a logical address. This 20-bit addressing scheme provides potential access to over one-million bytes of memory and allows addresses to be independent of the physical location of memory within the system.

The hardware for this system has been designed with expansion in mind, providing capacity for up to five EPROM-bearing boards per test set. Two of these boards are known as "computer" boards, with six EPROMs each. These EPROMs may be either 4- or 8-kbyte types, are always addressable, and are thus referred to as "unswitched" memory. The other three boards, known as "memory boards," contain up to 16 EPROMs each. This memory is addressed in 4-kbyte blocks regardless of EPROM size. Since bank switching is used, these blocks

Reprint RE-26-8-5
Final manuscript received June 25, 1981.

are referred to as "switched" memory. EPROM chips on these boards can be in the form of 4-, 8-, or when available, 16-kbyte chips. In a full 60-chip system, this can provide 240, 480, or 864-kbytes of storage.

STE/M1 field test system

The field test system pictured in Fig. 1 is designated STE/M1. It was designed to provide on-vehicle maintenance support for the ABRAMS M1 tank. The development of software application programs for the STE/M1 created a demand for an extraordinarily large stored-program memory capacity. Applications-program development is based on analysis of each vehicle subsystem and subsequent structuring of test flowcharts indicating detailed test procedures.

A portion of a flowchart page is shown in Fig. 2. Flowcharts contain: messages as displayed to the operator on the hand-held communicator (in parallelograms), measurements made by the test set (hexagons), comparison of measurement results to limits by the test set (diamonds), and diagnostic conclusions (parallelograms with double

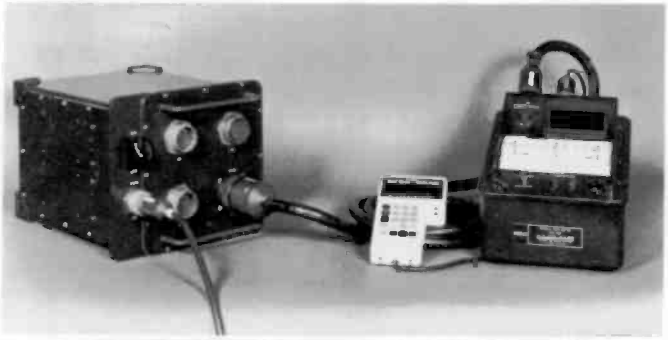
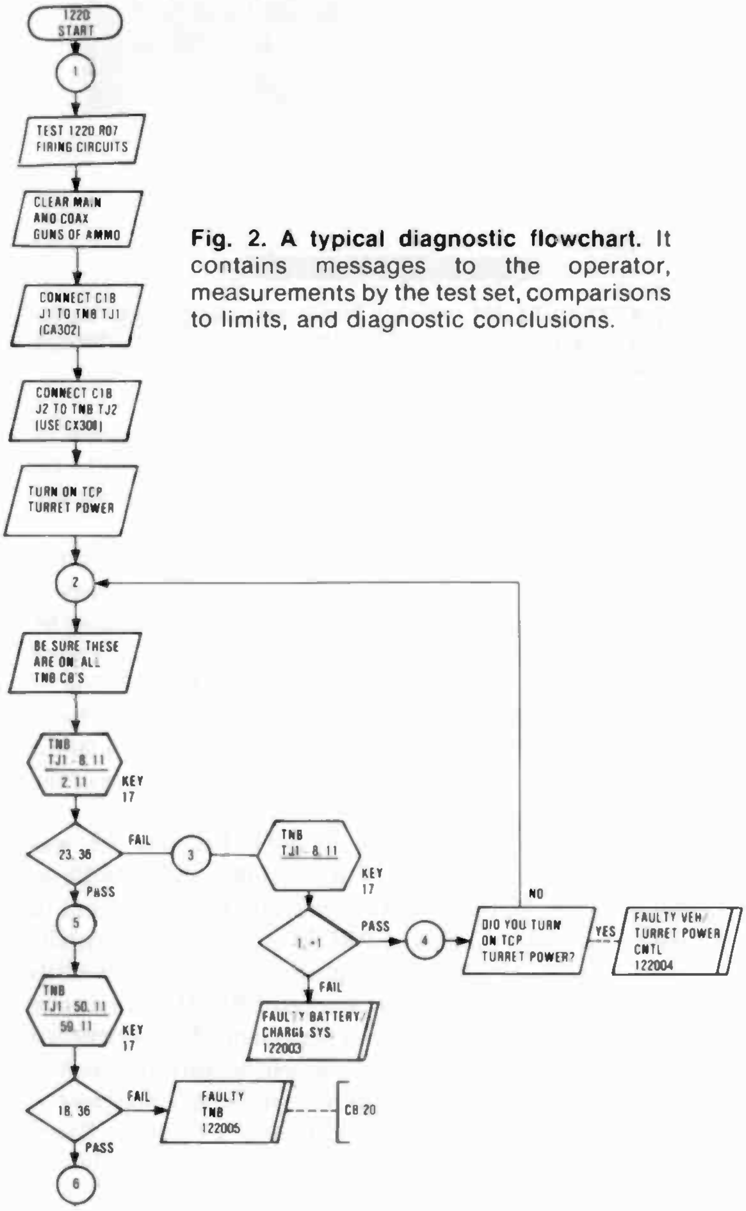


Fig. 1. The STE-M1 test system. Containing two 1802 microprocessors and 192 kbytes of EPROM memory, it was designed to provide support for the ABRAMS M1 tank with minimal use of manuals.



1220 START
1
TEST 1220 R07 FIRING CIRCUITS
CLEAR MAIN AND COAX GUNS OF AMMO
CONNECT C1B J1 TO TNB TJ1 (CA302)
CONNECT C1B J2 TO TNB TJ2 (USE CX300)
TURN ON TCP TURRET POWER
2
BE SURE THESE ARE ON ALL TNB CB'S
TNB TJ1 - 8. 11 / 2. 11
KEY 17
23. 36
FAIL
PASS
3
TNB TJ1 - 8. 11
KEY 17
1. +1
PASS
FAIL
4
DID YOU TURN ON TCP TURRET POWER?
NO
YES
FAULTY VEH/TURRET POWER CNTL 122004
FAULTY BATTERY/CHARGE SYS 122003
5
TNB TJ1 - 50. 11 / 59. 11
KEY 17
18. 36
FAIL
FAULTY TNB 122005
CB 20
PASS
6

Fig. 2. A typical diagnostic flowchart. It contains messages to the operator, measurements by the test set, comparisons to limits, and diagnostic conclusions.

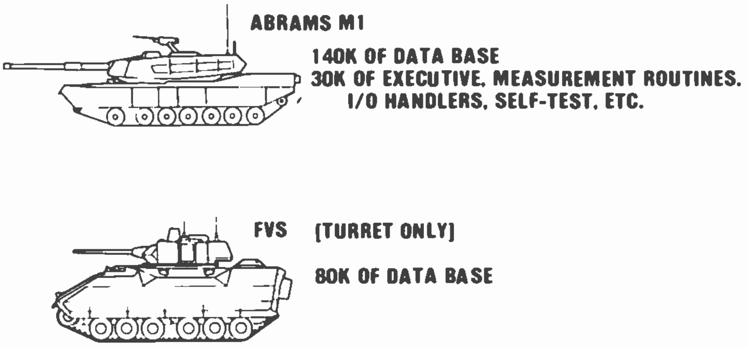

ABRAMS M1
140K OF DATA BASE
30K OF EXECUTIVE, MEASUREMENT ROUTINES, I/O HANDLERS, SELF-TEST, ETC.
FVS
(TURRET ONLY)
80K OF DATA BASE

Fig. 3. Memory requirements for supporting ABRAMS M1 and FVS turret exceed 200 kbytes.

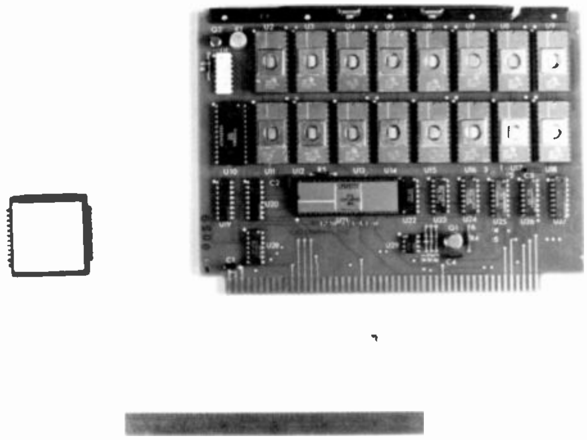


Fig. 4. The STE-M1 memory board was designed to bank-switch sixteen 4-kbyte EPROMs, providing a storage capacity of 64 kbytes.

lines). The entire M1 diagnostic package is composed of approximately 800 pages of flowcharts. Memory requirements for holding such a large diagnostic package are unusually high.

Figure 3 summarizes the memory requirements for the M1-tank support package. Since the addressing capacity of the 1802 microprocessor employed by STE/ICE is only 64 kbytes, it is obvious that some special techniques were required to handle the amount of memory shown in Fig. 3. This memory problem was solved by two techniques: bank switching of EPROM memory chips, and using a 20-bit logical addressing mode to extend the address space of the 1802.

When the hardware for the STE/M1 test set was designed, the state-of-the-art in EPROMs was the 4-kbyte device. Pictured in Fig. 4 is the memory board designed to bank-switch 16 of these EPROMs. The test set initially contained two such boards plus two computer boards containing 16 additional unswitched EPROMs for a total memory capacity of 192 kbytes per system. The unswitched EPROMs contain the operating system software, such as power-up initialization routines, self-test, interpreters, measurement utilities, and the test-set executive. Bank-switched EPROMs, residing on memory boards, hold the applications program data base.

Software fills memory

Software for supporting the ABRAMS M1 vehicle filled virtually all 192 kbytes of memory in the STE/M1 test set. In order to accommodate new vehicle applications, more memory had to be added, creating an update to the STE/M1 system. This new version, known as STE-M1/FVS, had two modifications for memory expansion. First, the memory board was redesigned to hold any of three PROM types — the original 4-kbyte chip, the present state-of-the-art 8-kbyte chip, or a projected near-term 16-kbyte chip. The computer board was redesigned to use either 4- or 8-kbyte PROMs. Second, the executive software was modified to allow use of a spare board slot for an additional memory board.

Bank-switching employed

A block diagram of the bank-switching scheme employed in STE-M1/FVS is shown in Fig. 5. As stated before, the 1802 microprocessor can directly address 64 kbytes of memory via its 16 address lines. There are two microprocessors in the system. Each 64-kbyte memory space is divided into sixteen 4-kbyte blocks. Twelve of these blocks (or 48k) are used for the unswitched EPROMs, and three blocks (12k) are used for RAM. This leaves one 4-kbyte block for the switched EPROMs.

A memory board loaded with 16 chips must be divided into 4-kbyte blocks regardless of the size of the device employed. To accomplish this, all eight bits of the microprocessor data bus are used. Four bus bits are decoded to select one of sixteen EPROMs on the board. Two bits act as the upper-two address bits for the 8- or 16-kbyte EPROMs. This allows a 4-kbyte portion of the device to be selected. The last two bits are used to distinguish between the existing memory board and the board added to the spare slot. Thus, each memory board is treated as if it were

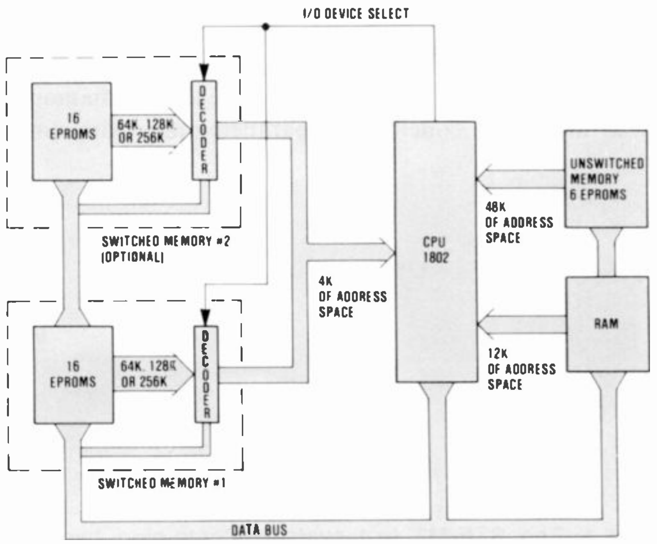


Fig. 5. The bank-switching technique allows up to 512 kbytes of EPROM memory to occupy 4 kbytes of address space.

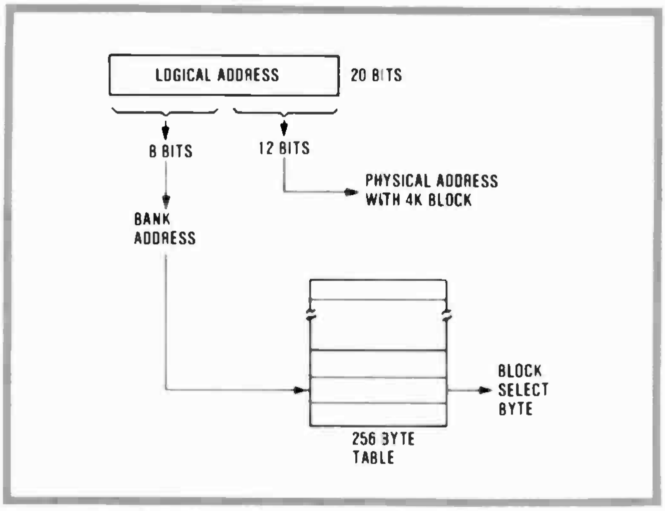


Fig. 6. The 20-bit logical address is divided into an 8-bit bank address and a 12-bit physical address.

an I/O device. It must be selected by an output instruction from the microprocessor, and it receives parameters via the data bus. If both boards are installed, their outputs are tied together to occupy the same 4-kbyte block, since only one board may be on at any given time.

The bank-switching scheme described ultimately results in a data base on the order of hundreds of kbytes of memory. In order to address a data base of that size, it was necessary to develop a special addressing scheme tailored to the EPROM bank-switching technique. A 20-bit logical addressing mode, capable of accessing approximately one-million bytes of memory, was chosen.

Figure 6 shows the bit utilization of the 20-bit address as it relates to hardware bank-switching. The first byte, known as the bank address, is mapped through a table that is used to select a 4-kbyte page within the data-base memory. This is accomplished by energizing the correct memory board and selecting the correct chip or portion of a chip within that board. The final 12 bits of the logical address are used by the 1802 microprocessor as an offset within the selected 4-kbyte page.

STE/ICE	14K
STE-M1	192K
STE-M1/FVS (CURRENT)	480K
STE-M1/FVS (POTENTIAL)	864K

Fig. 8. Sophisticated memory manipulation and improving technolgy have allowed an evolution to over ¾-million bytes of EPROM memory in an 1802 microprocessor-based system.

The mapping table

The mapping table used by the system for translating the bank address portion of a logical address is filled within RAM during the test set's power-up initialization sequence. This table has 256 slots, allowing one slot for each possible 8-bit bank address.

Figure 7 illustrates the use of the mapping table. The EPROM chip in board #2, slot #9, assumed to be a 4-kbyte chip, is shown identifying itself as bank-address 1A during a poll taken by the executive system at power-up time. The executive would then fill table-slot 1A with information that corresponds to board #2, slot #9. During subsequent execution of application software, any logical address that

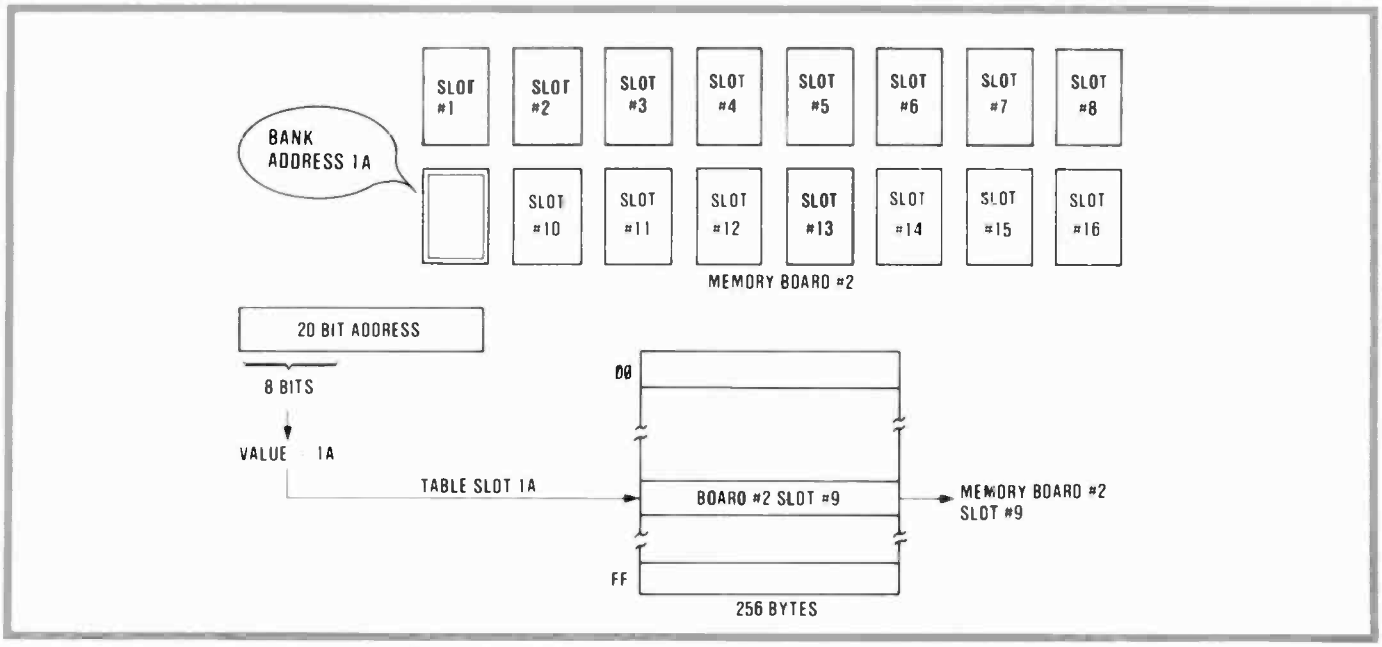


Fig. 7. The bank address is mapped through a table in order to energize the correct board and slot.

begins with 1A will result in board #2, slot #9, being enabled. By taking the memory page corresponding to switched memory and adding the remaining 12-bit offset from the logical address, the 1802 can then address the data-base memory in its normal way.

Summary

Figure 8 summarizes the memory capacities of the STE/ICE test set and the STE-M1/FVS test system based on 4-, 8-, and 16-kbyte EPROMs. The evolution of an 1802-based test set to a system with over 3/4-million bytes of memory was made possible by the innovative hardware and software memory manipulation techniques engineered within the STE projects.

Herb Resnick, Senior Design Engineer (standing), and Dave Milley (seated) Member Technical Staff, Automated Systems, Burlington, Massachusetts.

Dave Milley is a Member of Technical Staff of Non-Electronic Test Engineering, Automated Systems, Burlington, Massachusetts. Since joining RCA in 1978, he has been involved with hardware and software designs of microprocessor-based test equipment. He is a graduate of the University of New Hampshire where he received a BSEE.

Contact him at:
Automated Systems
Burlington, Mass.
TACNET: 326-2963

Herb Resnick is Senior Design Engineer at Automated Systems, Burlington, Massachusetts. Since joining RCA in 1974, he has been involved in hardware and software design of automatic test equipment. He received a BSEE and MEE from Cornell University.

Contact him at:
Automated Systems
Burlington, Mass.
TACNET: 326-2872

W.J. Hepp|R.H. Isham

Multimicroprocessor-based transistor test equipment

A multimicroprocessor-based test set employs several levels of intelligence in a test console that provides many new test-system features.

Abstract: *A state-of-the-art transistor test set that performs all commonly required tests and that is adaptable to almost any custom-test requirement is described. The set features complete testing with one insertion and includes self-calibration and self-diagnosis capability. Its in-house fabrication cost can be as little as one-fifth the cost of a commercially available test unit.*

As the capability of power devices has improved, it has become increasingly important to accurately test not only for the traditional parameters, but for characteristics—such as thermal resistance and forward-biased second breakdown—directly related to the application in which the device is used. In addition, some customers require special-use tests. Switching tests and high-temperature parameter verification are also being required in an increasing number of applications. Traditionally, these testing needs have been satisfied by multiplexed computer-controlled test sets that measure standard parameters, and a multiplicity of special-purpose test sets that measure those characteristics that must be controlled in specific applications or that cannot be measured effectively in conventional test sets. The need to reduce as much as possible the number of insertions, that is, the number of different types of test sets that a device must pass through, has also increased in importance. The passage of a device through a multitude of test sets increases its cost by adding handling costs, and increases the possibility that it will be mis-segregated at least once. These factors are particularly important in the testing of automotive ignition transistors, where very high quality levels are demanded, and where pricing is very competitive. This paper discusses a modern test system that meets these demands through its ability to perform multiple-temperature testing of traditional and custom parameters in one insertion.

History

Computer-controlled test equipment was first introduced to the RCA Solid State Division (SSD) power-manufacturing facility in the form of SCOPE test systems.[1] More than 35 of these hardware-multiplexed mini-computer systems were built between 1969 and 1976, and 25 are still being used in various SSD locations throughout the world for routine device testing.

The marketing of high-voltage Darlington transistors for use in automotive ignition systems required testing to quality levels unprecedented for commercial devices. Some customers also insisted that tests be performed to simulate distributor misfiring, reversed battery polarity, battery disconnect transients, and fouled spark-plug operation. One-hundred-percent testing of both switching time at room temperature and dc parameters at several high temperatures was also required. Traditional testing methods would have involved at least seven insertions, making the achievement of the required quality levels difficult.

A method of internally heating a power transistor by dissipating energy had been demonstrated through the use of an engineering version of a SCOPE test system. The method worked well, but required more test time on a multiplexed, multisocket SCOPE system than the maximum 200 to 300 milliseconds that was economically feasible. It was recognized that a nonmultiplexed SCOPE system would not suffer the same restriction since all of the hardware would be dedicated to a single socket. In addition, removal of the multiplex circuitry, with its long leads and high stray capacitance, would allow a primitive

Reprint RE-26-8-6 ST-7020
Final manuscript received July 30, 1981.

switching-time circuit and the special customer tests to be added.

A prototype of a nonmultiplexed system, the first GALT test station,[2] was developed and placed on line in 1975. It used the basic SCOPE concepts, implemented in state-of-the-art hardware, and was optimized for single-insertion multiple-temperature testing of automotive Darlington transistors. It was connected to the minicomputer in an existing SCOPE test system and operated by "stealing" CPU time. Calculations had shown that more than 20 GALT stations could be added to the system with no degradation in the performance of the SCOPE system. This prototype station is still in use in SSD's plant in Malaysia.

Introduction of the SWITCHMAX high-speed transistor families in 1977 brought a need for accurate testing of switching time at high temperatures. Using experience gained with the prototype GALT system, a new station optimized for high-speed switching time was designed. In this station, the minicomputer was replaced with a COSMAC development system as the station controller. This replacement was possible because the minicomputer in a typical SCOPE station is idle about 99 percent of the time during testing, and has a machine-cycle time only ten-times faster than a COSMAC CPU. To keep the cost down, the only peripheral included with the GALT station was an RCA Microterminal, which was to be used primarily for maintenance purposes. A central computer system was used to load test programs and to log results over a serial data link.

A second COSMAC development system with disk-operating software and four test-station interfaces was originally used for the central computer. More recent GALT sets have tied to existing Hewlett-Packard minicomputer-based systems, and have employed RCA Microboard prototyping systems as the station controllers.

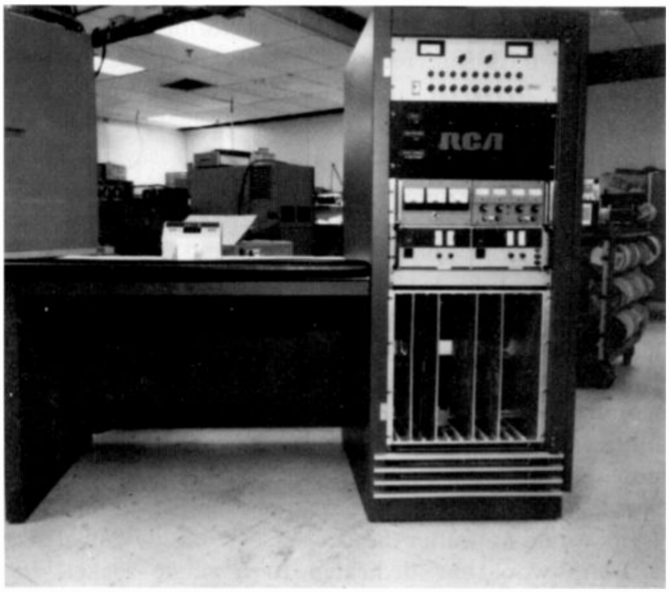


Fig. 1. GALT V switching-time console (with door removed).

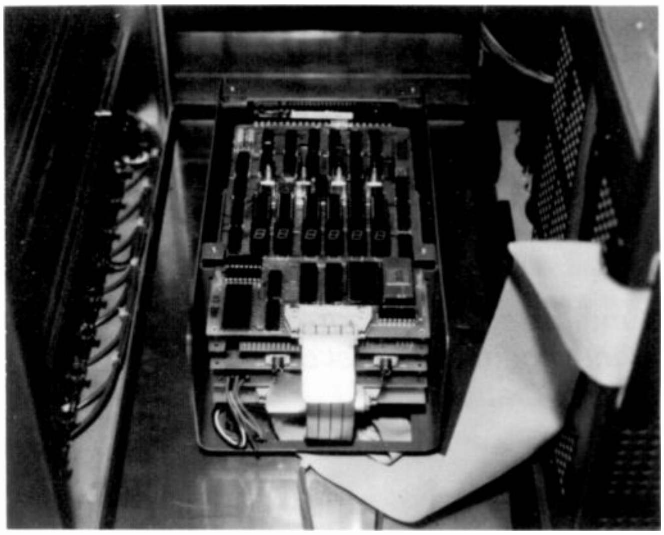


Fig. 2. Microboard station controller (with cover removed).

A total of three of these high-temperature-switching-time test stations (Fig. 1) have been built to date. One is in operation in SSD's Mountaintop location and the other two are operating in Malaysia. In addition to testing switching time, the stations are configured to test forward-biased second breakdown and several conventional parameters.

System description

Unlike most other commercial test-system manufacturers, RCA has generally grouped test hardware functions on boards; one board contains all of the drivers needed for gain measurements, another those needed for leakage measurements, and so on. This practice allows customizing of the response of each driver to suit the task, and allows modification or addition of one class of test without affecting others.

Until recently, there had been little control at the board level. All information needed to run a test was placed in parallel on the test backplane, timing signals were generated, and results read under control of the central computer. More complex tests, such as switching time, required more information than could be placed on a test backplane or transmitted over a parallel data link. This predicament suggested a board-level controller that could accept data in serial/parallel fashion, run the test, and return results. A system of this type provides a clean break between the logic that performs the test and the logic that decides which test to perform. Thus, the testing hardware developed can be used in different systems with minimum trouble, and can be easily expanded.

The heart of the test station developed to satisfy the needs just described is an RCA Microboard computer system (Figs. 2, 3, and 4). Figure 2 is a photograph of the Microboard station controller itself. Figure 3 depicts the system Microboard complement, and Fig. 4, the block diagram of the switching-time console. As shown in Fig. 2,

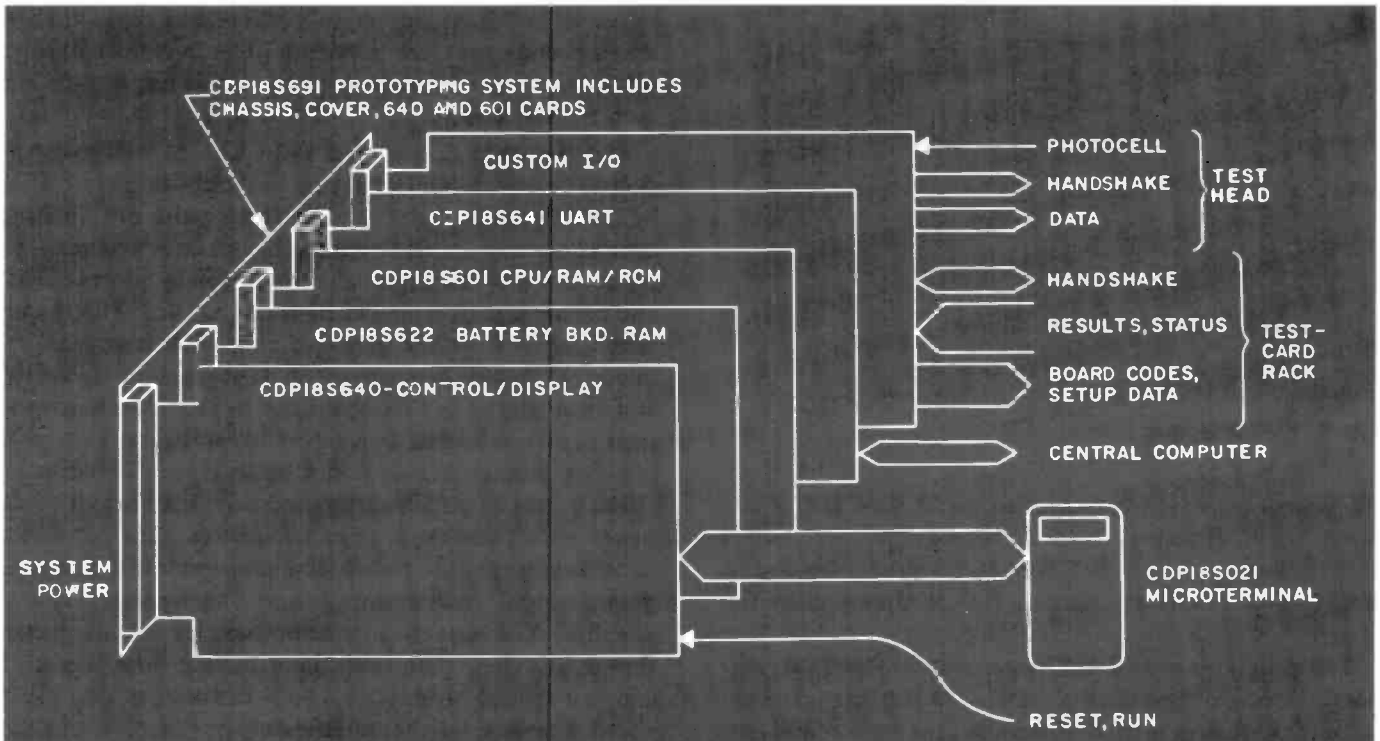


Fig. 3. The Microboard system.

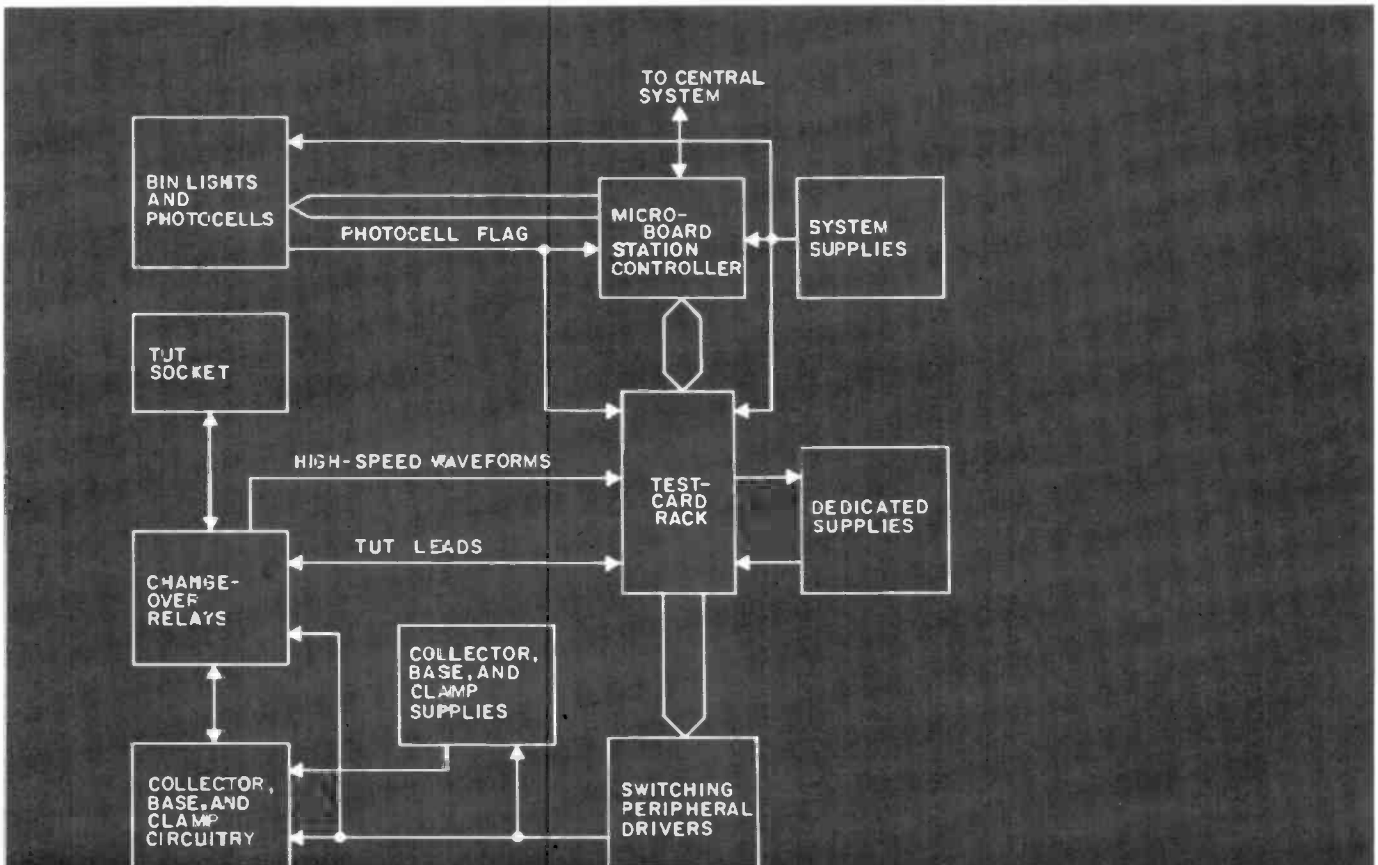

TO CENTRAL SYSTEM
BIN LIGHTS AND PHOTOCELLS
PHOTOCELL FLAG
MICRO-BOARD STATION CONTROLLER
SYSTEM SUPPLIES
TUT SOCKET
HIGH-SPEED WAVEFORMS
TEST-CARD RACK
DEDICATED SUPPLIES
TUT LEADS
CHANGE-OVER RELAYS
COLLECTOR, BASE, AND CLAMP SUPPLIES
COLLECTOR, BASE, AND CLAMP CIRCUITRY
SWITCHING PERIPHERAL DRIVERS

Fig. 4. Block diagram of the switching-time console.

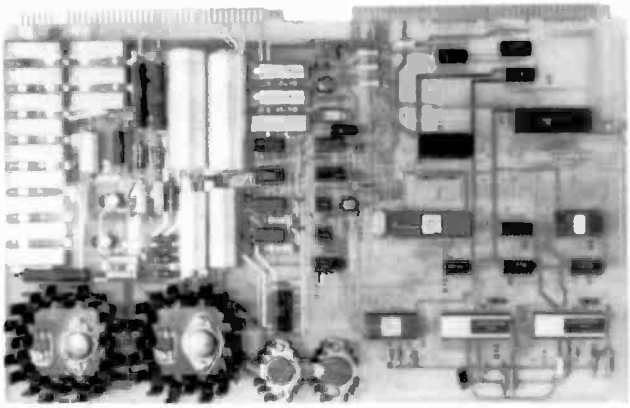


Fig. 5. Forward-drop board.

the system employs four standard cards: the CDP18S601 CPU, the CDP18S641 UART, the CDP18S640 control and display board, and the CDP 18S622 battery-backed 8k RAM card. A custom parallel I/O card completes the system.

Test programs are loaded from a central station at 2400 baud through the UART card. No further external communication is needed unless data taking is requested. The battery-backed RAM provides protection from power outages; the control and display card provides the interface to an RCA Microterminal (a pocket-sized keyboard and display unit) used for servicing. The custom I/O card provides communications with the set of test boards and with the test head.

The test head houses pass/fail LEDs as well as hexadecimal readouts for displaying bin selection. As in RCA SCOPE systems, light and photocell pairs are grouped around the test socket to detect the operator's hands. The readouts are latched internally and driven by address, data, and timing signals from the custom I/O card. The "hand-clear" signal from the test-head photocells is critical; the operator's safety depends on it. It is used to signal the system to start a test sequence and to provide a hardware reset to all test boards, overriding any software. The signal is also generated during power up to clear all boards.

Each class of tests is accommodated on a 11- by 17-inch board with two 43-pin edge connectors. One connector handles large signals: transistor-under-test leads, dedicated power-supply connections, and high-power system supplies. The second connector handles all low-power system supplies, data, and handshaking lines. Figure 5 shows a typical board with its two connectors, and Fig. 6 shows a typical test-board assembly or "cage."

Each test board contains a small CDP1802-based system (Fig. 7) that controls the hardware on the board and

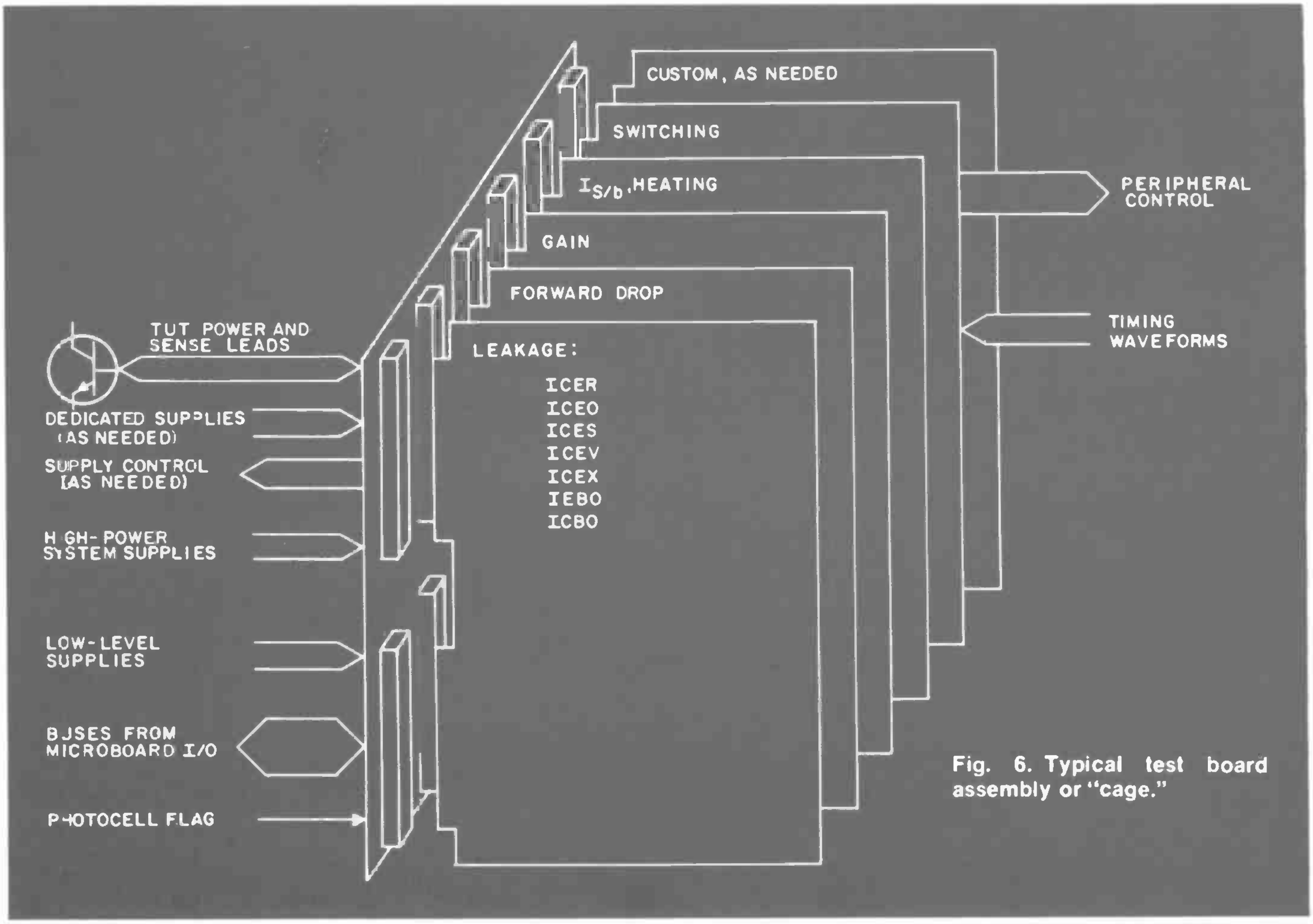


Fig. 6. Typical test board assembly or "cage."

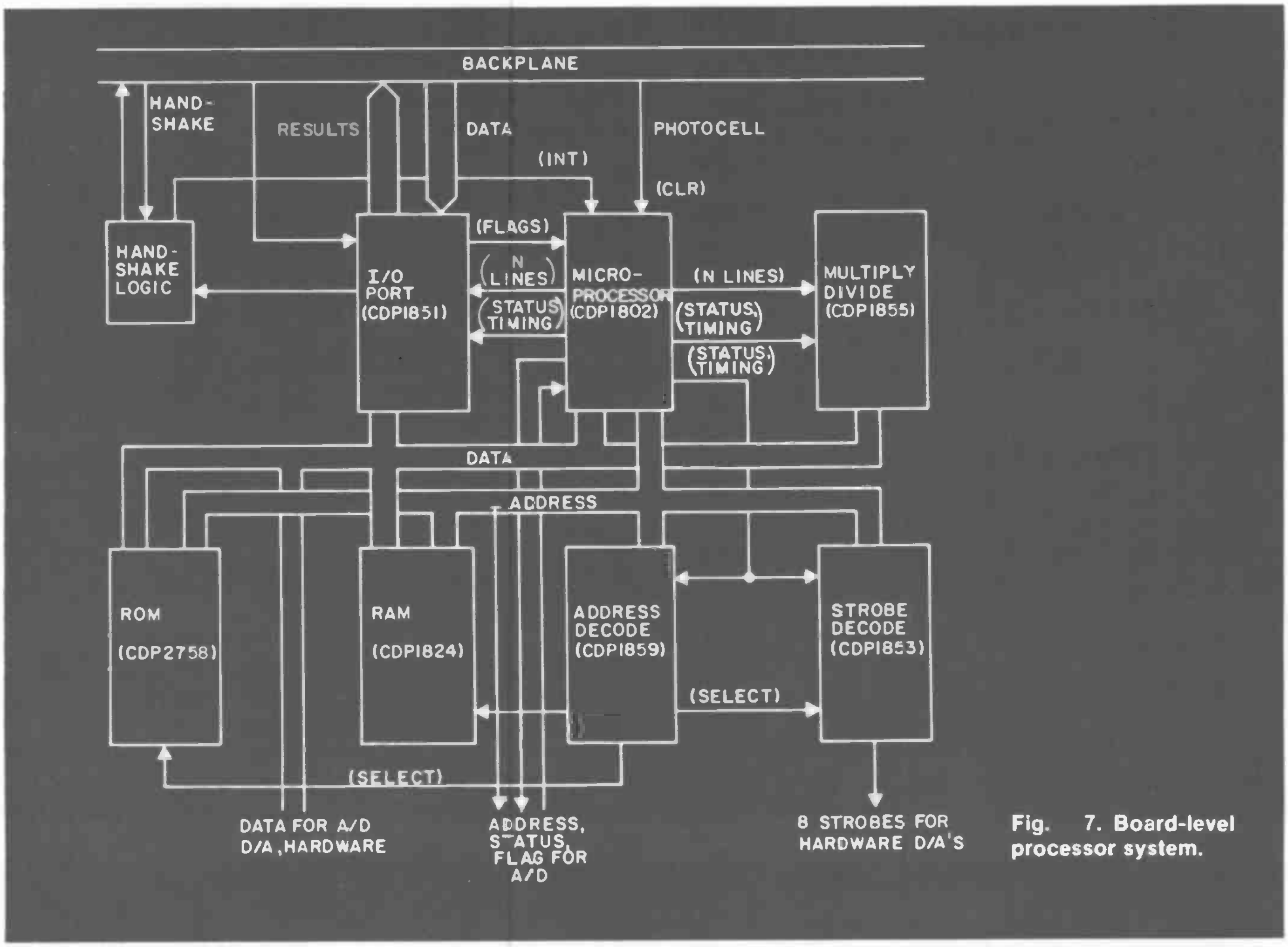


Fig. 7. Board-level processor system.

communicates with the test-station controller. The system includes 1 kilobyte of ROM, 32 bytes of RAM, a CDP1851 programmable input/output port, and some handshaking and address-decoding logic. A self-calibration feature added to some boards requires the addition of a CDP1855 multiply-divide chip. Because of the limited I/O decoding capability of the CDP1802, eight memory addresses are decoded for the on-board hardware interface.

The CDP1802 was chosen as the main component of the board-level processor system for several reasons, Designer familiarity and its low cost were important factors, but CMOS noise immunity was the deciding one. There has, indeed, been very little trouble with "glitches," despite the electrically noisy environment of the test stations.

Two data buses connect the test boards to the system controller: one to pass "board codes" or test set-up data to the boards, and one to pass back results and status. The "hand-clear" signal resets each board at the start of a test sequence. All output ports to the backplane are three-stated; the boards initialize (and self-calibrate) and wait in an idle loop. The station controller places the address of the desired test (board code) on the backplane and generates a signal to interrupt all boards. All boards compare the "board code," and the addressed board responds with a flag. Sufficient data to run the test is then passed over the bus using a simple handshaking scheme. The board compiles the set-up data, runs the test, and signals the system controller to retrieve the results. Again, using a simple handshaking scheme, two bytes of results and status are passed back. The board then idles, waiting for another interrupt.

The leakage board (Fig. 8) typifies recent hardware in that it uses no off-board components and is self-calibrating. The microprocessor front-end interfaces to the hardware through relay drivers, a data latch, and A/D and D/A converters. The D/A converter is time multiplexed to set both base-bias conditions and collector or emitter voltage.

When the board has received all data needed to specify a test, the relay-driver bits are determined and outputted. During the relay-settling time, the base and then the collector D/A-converter values are calculated and loaded (the D/A converter has two internal registers). The base bias is isolated from the D/A converter, the collector voltage is "rippled through," and the collector-supply inverter is activated. After a further settling time, which depends on current measuring range, the A/D converter is strobed. Supplies are shut off in reverse order, and the results are returned.

As an example of a noise source present in the test set, the

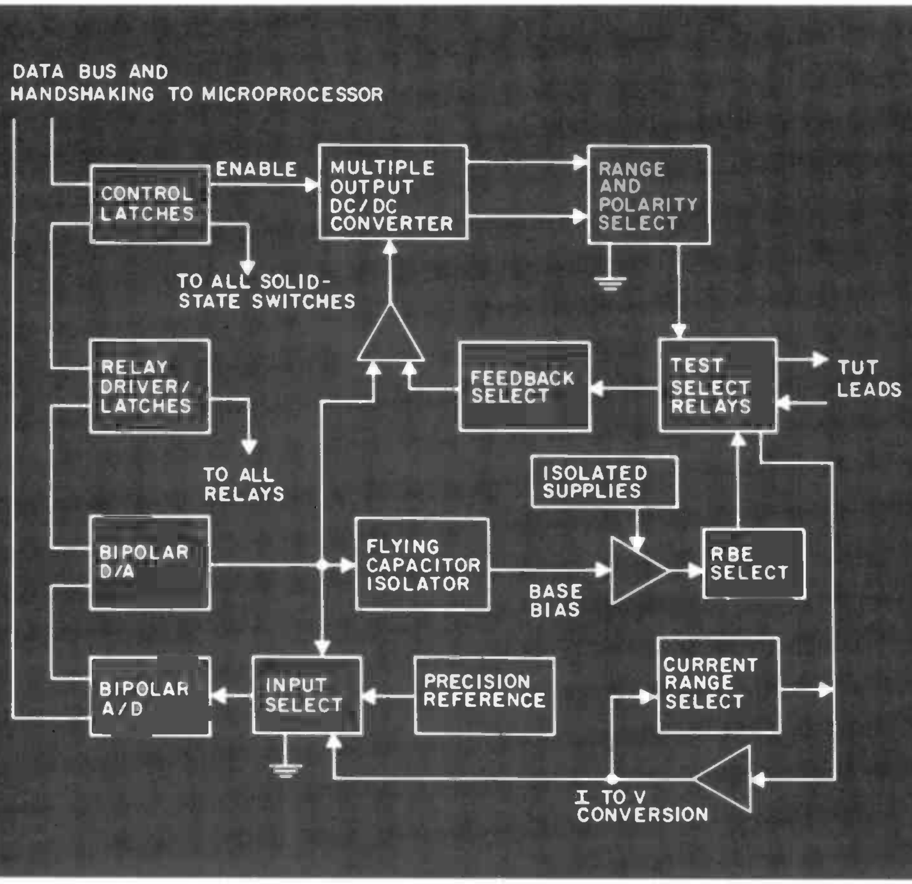
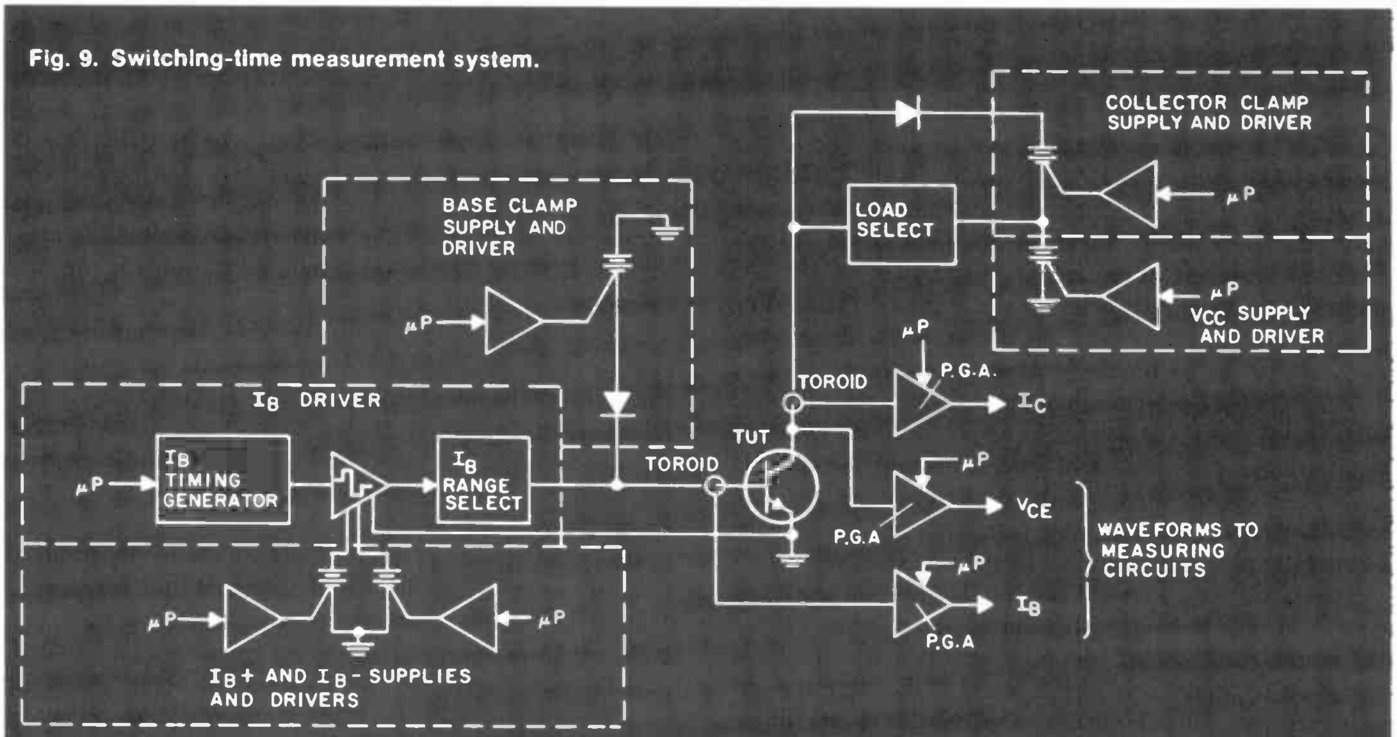


Fig. 8. Leakage-board output section.

Fig. 9. Switching-time measurement system.



P.G.A = PROGRAMABLE GAIN AMPLIFIER

RELAYS THAT DISCONNECT TUT FROM MAIN BACKPLANE NOT SHOWN

REVERSE ALL POLARITIES FOR PNP's

leakage supply is a 125-kHz push-pull inverter with a 1-kilovolt output. The inverter is located within several inches of the microprocessor "front end."

Switching-time measurements

The measurement of power-transistor switching parameters over a wide range of conditions requires more than one board of hardware; Fig. 9 shows the set-up used. The transistor under test must be disconnected from the normal test-board backplane and connected in a low-inductance loop with a collector load and a current-sensing transformer. A second low-inductance loop is composed of the base-drive circuitry and a second current transformer. A third low-inductance loop contains high-speed switching diodes and bypass capacitors that clamp collector voltage when inductive loads are switched.

Base- and collector-current and collector-voltage waveforms (Fig. 10) are fed back to the switching-time board. These signals then pass through programmable gain amplifiers and comparators into the measurement circuitry. Measurement is made by time-interval averaging,[3] which produces a resolution of five nanoseconds. All comparator levels are set under program control, an arrangement that allows the measurement points to vary.

The processor system is somewhat larger than on other test boards, with more RAM and 32 decoded output lines. A separate rack is provided for all supply drivers, relay drivers, the base-drive timing driver, and so on.

The most difficult hardware functions to implement in the multimicroprocessor test set included the base-drive circuitry and the programmable amplifiers. The base drive must output a reversing current pulse of 5- to 50-microseconds duration and of 1-milliampere to 6-amperes magnitude. Rise and fall times of less than 20 nanoseconds are produced at low currents, with socket and wiring inductance limiting the highest current pulses to 50 nanoseconds. The drive is entirely bipolar; both *n-p-n* and *p-n-p* transistors can be accommodated. The programmable amplifiers must operate over a gain range of 1 to 2000, with 1, 2, 5 sequence. Rise and settling times total 25 nanoseconds, accuracy is one percent, and inverting or noninverting gain is provided.

The main collector and collector-clamp supplies were also somewhat difficult to realize. They must sink or source currents as high as one-half ampere at voltages up to 450 volts, also under program control, and can be stacked to a maximum of 900-volts clamp voltage.

The appearance of the switching-time circuitry to the test-console controller is similar to any other test board, but approximately 20 bytes are required to specify the test. Nine different parameters can be measured in one test, but as described earlier, only one result of up to 16 bits is returned per test. To avoid unnecessary test repetition, the board first compares all set-up conditions to those of the previous test. If the conditions compare exactly (except for the result requested) and no other board has been called since the last test, the board will go directly to returning results.

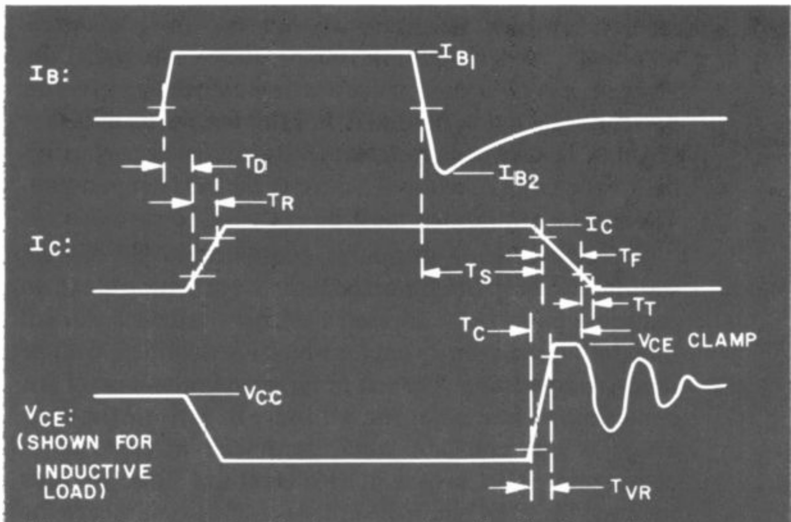


Fig. 10. Base- and collector-current and collector-voltage waveforms supplied to the switching-time board during switching-time measurements.

If a new test is required, the first function of the system is to decode and energize the required relays. The starting values for all supplies are then calculated, and after a waiting period for relay settling, the supplies are programmed up. Reference values for all D/A converters are outputted along with other set-up conditions, such as base-drive pulse width. After the waiting period for supply ramp-up, the base drive is activated at a 2-millisecond repetition rate. At this time a supply-adjustment procedure begins.

Switching parameters can be very sensitive to small changes in set-up conditions; therefore, these conditions must be maintained as accurately as possible. Some parameters, such as reverse base current during turn-off, maintain their peak value for less than 50 nanoseconds. These constraints make conventional regulation techniques difficult, and it is for this reason that the processor has been made part of the feedback loop. Comparators are set with the desired peak values of collector current, forward- and reverse-base current, and collector-clamp voltage. During a switching cycle, latches are set if any value exceeds that desired. The processor examines the latches after each cycle, and increases or decreases each supply as needed. This adjustment is made after every cycle for 100 cycles, and then reduced to one adjustment on every fourth cycle. The measurement circuitry is then enabled, and the switching, adjusting, and measuring continue for another 100 cycles. Supplies are then disabled, results calculated, possible errors checked, and the result phase entered.

Operating-system software

The operating-system software in any test system should provide the means for testing devices in the minimum amount of time necessary to bin (or sort) a device accurately. When multiple temperatures are used, the problem is complicated, since it is time consuming to cool a device once it has been heated. For this reason, all tests that might be needed at a given temperature must be performed before the unit under test is heated to the next temperature. Those tests not needed must be excluded to save time.

The Microboard computer contained in the GALT stations has been programmed to perform testing along an optimum path. Units are tested so that the device will fall into the highest priority bin for which it is qualified, with no unnecessary testing. The test program contains a list of test conditions to be used by the hardware to perform the tests. There is one entry for each parameter at each set of conditions. The program also contains lists of tests required for each bin. Each entry contains a pointer to the proper entry in the test-condition list and the limits to which the results of the test are to be compared. Limit comparison is done in software, since each test returns an actual result. This is a departure from almost all commercial test systems, which make only analog hardware comparisons. A unit is assumed to be eligible for all bins before testing begins. An eligibility flag for each bin is maintained in memory during testing, and is used by the steering logic to determine which test to perform next.

Testing begins when a device is plugged into the test head and the "hand-clear" flag is set by the photocell hardware. The first test performed is a continuity test to assure that the unit is making proper contact in the socket. If this test fails, the continuity-fail code is displayed on the bin lights, and testing is complete. If this test passes, all of the eligibility flags are set to "eligible," and all of the test-result locations are set to indicate that no tests have been performed. The pointer for the first test of the highest-priority bin (the one which must be filled first) is then retrieved from the required test list, and the test is performed. The results of the test are stored in the memory location reserved for that test, overwriting the test-not-performed flag. The results are then compared to the specifications for the highest-priority bin. If the test has failed, the eligibility flag for that bin is set to "reject," and no further tests are performed for that bin. If the test passes, the eligibility flag remains unchanged, and the pointer for the next test in that bin is retrieved.

Testing continues, as long as a fail does not occur, until the last room-temperature test for the highest-priority bin is performed. At this point, the program looks ahead to see if any tests are required at higher temperatures. If they are, the eligibility code remains unchanged and testing continues with the next bin. If no tests are required at higher temperature, the eligibility code is set to "qualified."

Before the room temperature tests for the next bin are started, the list of tests for that bin is scanned to see if any of the tests have already been performed. The previously performed tests are compared to the required limits, and a pass/fail decision is made for each. If a failure occurs, the eligibility code is set to "reject" (as above) for that bin, and no further tests are performed for that bin. If no failures are found, and there are remaining room-temperature tests, they are performed. This look-ahead feature prevents further testing of a device that has already failed an identical test, and prevents repeating tests already performed to the same set of conditions.

Room-temperature testing proceeds for the remaining bins or until a bin is found for which the device is fully qualified. The program then checks to see if the device is eligible for higher-temperature testing for a bin of a higher priority. If it is not, the number of the bin for which the device qualifies is displayed and testing is complete. If the device is eligible for higher-priority bin testing at a higher temperature, the proper amount of energy is dissipated in the device to heat it to the next temperature. The test-result flags are reset to show no tests done, and testing continues. The process is repeated for each temperature. If, at any point, all eligibility codes are set to "fail," the reject code is displayed on the bin lights and testing is complete.

The method of heating devices through the use of internal heating is a time-consuming one. Therefore, there is a tendency on the part of the test programmer to heat the device as fast as possible by setting the dissipation as close to the capability of the device as he can. When this is done, however, some of the tested devices may experience forward-biased second breakdown. If this condition occurs, the device does not reach the proper temperature and further testing is inaccurate. The control program handles this problem by displaying a failed heating code on the bin lights and by considering testing complete. Since this condition is not a true failure, the device can be placed in a bin for retesting with lower-energy heating pulses.

Some of the tests used to classify power devices require the dissipation of energy sufficient to damage a device and make the previous test results invalid. A feature has been included in the subject test-system control program to code such tests. If at any time the device fails one of these potentially destructive tests, a special code is displayed on the bin lights, and testing is complete.

Some of the microprocessor-controlled test boards have provisions for self-diagnostic checks. When one of these checks fails, a code is included in the test results sent to the Microboard computer. When the control program senses this code, it immediately stops the testing process and displays a flashing code on the bin lights to indicate a board failure. The display includes both the board number and type of failure. The test station is locked up at this time. After the board failure is noted, the station can be reset to allow testing of the next device, although board failure will be detected if it occurs again.

When one test board is switched off and a different one called, there is the possibility that a relay may fail to open. Since this situation is potentially destructive, the software calls in a "hung-relay" check each time the boards are switched. If a hung relay is found, the system locks-up in a manner similar to that for a failed board.

Utility software

The control software contains two methods of testing in addition to the optimum path described above. A user can define a data-test program that contains a list of tests in a specified sequence. This method is used in conjunction with a data-analysis program in the central computer to record actual test results. In this mode, the testing begins when the serial number of the device to be tested is sent from the central computer to the test station. The serial number is displayed on the bin lights and the test station is enabled. When a "hand-clear" flag is received, the tests are performed as specified. The results are sent to the central computer, which processes them and, when ready, sends back another serial number. The process continues until the file of serial numbers is depleted and the central computer sends a test-complete code in place of a serial number. This signal causes the test-complete code to be displayed on the bin lights, and testing is complete.

The second method of testing is a very simple test mode designed for maintenance purposes. This mode employs the station's Microterminal for I/O. The necessary control words for a single test are loaded into a scratchpad area of memory. When operating in this mode, the control program repeats the test defined in the scratchpad continuously and displays the results on the Microterminal. Repetition times can vary from 12 milliseconds to 30 seconds. This mode can be used to perform a test once only by setting the repetition factor to zero. Utility software that checks the operation of the bin lights, photocells, and communications link has been included.

The software also contains routines for the loading of test programs from the central computer. While communication is proceeding, the photocell flag is ignored. This arrangement prevents an operator from testing while a program is being changed.

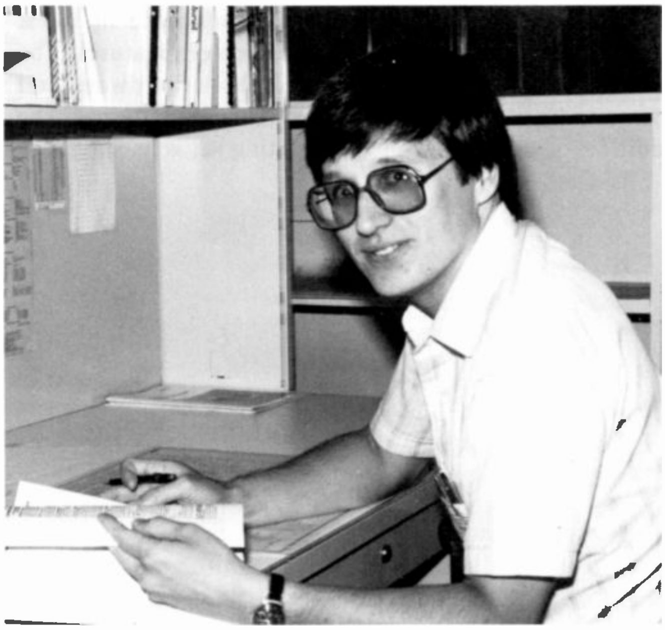
Maintainability

The most useful tool in facilitating repair of the subject transistor test system has been the RCA Microterminal used with the utility software described above. Because of the intelligence available at the board level, the words needed to run a test are simple to compile by hand. By manually loading a single test and using the appropriate control software, a board can easily be observed in operation.

Because it is known that the setting up of A/D and D/A converters is time consuming (and often ignored when one of these functions is being replaced), a way was sought to bypass these operations without compromising testing accuracy. The method decided on involves the placement of an accurate voltage reference on each board along with facilities to connect the A/D converter to the reference and to the D/A converters. Immediately after the "hand-clear" flag is set by the photocell hardware, each board reads ground and the reference voltage through its A/D con-

Bill Hepp joined the Solid State Division of RCA in 1958, as an applications engineer in the Micro Module Department in Somerville, New Jersey. He subsequently held applications engineer positions in the Commercial and Power Transistor Departments. In the early 1970s, he became interested in the use of computers for test and analysis of transistor parameters. This led to a position in the Power Test Engineering group, and the development of the GALT test systems. In 1978, he transferred to Mountaintop where he is a Systems Engineer in the Test Development group.

Contact him at:
Solid State Division
Mountaintop, Pa.
TACNET: 327-1381



Bob Isham joined RCA in 1970 on the Engineering Rotational Program. He was employed in Power Transistor/Thyristor, and Power-Hybrid Applications before entering Test Engineering for power devices in 1976. He is presently working in Microsystems Products in Somerville.

Contact him at:
Solid State Division
Somerville, N.J.
TACNET: 325-6291

verter. Gain and offset factors are calculated and stored for use in correcting the A/D-converter readings. Each D/A converter is then driven to ground and to a value near the end of the scale. Using the corrected A/D converter, gain and offset factors for the D/A converters can be calculated. During a board's operation, the factors are recalled and used to correct the D/A-converter outputs. Calculations are done during the three-millisecond relay-settling time, so that no test time is added. The result is corrected at the end of the test, which adds several hundred microseconds to the test time.

The test-board circuitry is designed to be minimally affected by offsets or other op-amp parameters. This fact, together with the self-calibration feature, allows the boards to meet one-percent accuracy specifications with no set-up at all, thus reducing labor content considerably.

Some limited self-diagnosis is starting to be included on test boards, with status bits being returned along with test results. A simple diagnostic test, such as for forward-biased second breakdown (IS/b) may report that a supply had run out-of-compliance, thus negating the result. A complex board, such as the switching-time board, may report such problems as counter overflow, supplies adjusted out of expected range, or improper base-drive response. It must be stressed that several problems may have identical symptoms; the diagnostics can only suggest the general location of the trouble.

Summary

The placement of several levels of intelligence in a test console has provided many new test-system features. The extreme modularity of the set, in both hardware and software aspects, allows a very flexible system design. The addition of a capability to an existing set is as simple as plugging-in a board. The system software is independent of the boards used.

The optimum-path software provides a powerful and time-saving determination of test sequence. The multiple-temperature, multiple-binning capability in one insertion provides a cost-effective way of meeting ever more stringent custom-testing requirements.

Utility software, Microterminal access, self-calibration, and self-diagnosis speed the understanding and solution of problems. Self-calibration and self-diagnosis decrease the possibility of mistesting the product. The construction and check-out time of boards is also greatly reduced, and the need for minor adjustments ("tweaking") has been eliminated.

The multimicroprocessor-based transistor test system is a flexible system that is adaptable to almost any custom-test requirement. As a bonus, the cost to fabricate the system internally can be as little as one-fifth of the purchase price of commercially available test systems.

Acknowledgments

The authors thank Don Burke for his support during the development of the GALT stations, Wally Williams for clearing administrative requirements, Clyde Van Horn for his contributions to the control and communications software, and Jim Hoshowsky for constructing the hardware from prints that no one else could read.

References

1. Cutshaw, E., "SCOPE—The Power Testing Paladin," *RCA Engineer*, Vol. 21, No. 5 (Feb./Mar. 1976).
2. Rand, Ayn, *Atlas Shrugged*, Random House, New York (1957). The term, GALT, was taken from this book.
3. "Time Interval Averaging," Hewlett Packard, Application Note 162 - 1.

W.H. Schilp, Jr.

Microboard equipment control

RCA Microboards and higher-level languages speed design of specialized microprocessor-controlled manufacturing equipment.

Abstract: *An approach to the design of microprocessor-control equipment through the use of RCA Microboards is discussed. The use of standard board products greatly simplifies the hardware design. As an example, a demonstration unit of a portion of the transistor test set described by Hepp and Isham in this issue was designed and built completely with Microboards. To ease the programming, two of RCA's higher-level languages were used in the project: Basic3 and PLM 1800.*

The purpose of the project described was to build a piece of equipment to demonstrate the use of RCA Microboards in specialized manufacturing-test equipment. The specific example selected was the testing of some active parameters of power transistors. The equipment was required to give a pass/fail response to preprogrammed limits for one type of transistor and to optionally enter a diagnostic mode and record statistics. The system was designed to be user interactive and to be flexible and expandable, so that additional tests, limits, or transistor types could be handled. All information is displayed by the system on a color monitor; the program comes-up running in the pass/fail mode. Since portability is a requirement, a 10-milliampere limit has been placed on current testing to keep the power requirement low.

System configuration

In the typical factory power-transistor test set there are four major blocks: central computer, station controller, board-level controller, and transistor under test. In the demonstration system, the central computer was replaced by a video terminal and keyboard because only one transistor type was to be tested and there was no need for mass storage of test programs. Figure 1 is a block diagram of the demonstration system. It shows a multiprocessor design with the processor in the system controller handling the keyboard inputs, video output, pass/fail criteria and diagnostics, while the test-subsystem processor controls the test conditions for the transistor under test and the reading of the test results. The system is designed so that additional test subsystems for tests such as leakage current, breakdown voltage, switching parameters, and energy capability can easily be added.

Reprint RE-26-8-7 ST-7026
Final manuscript received Aug. 4, 1981.

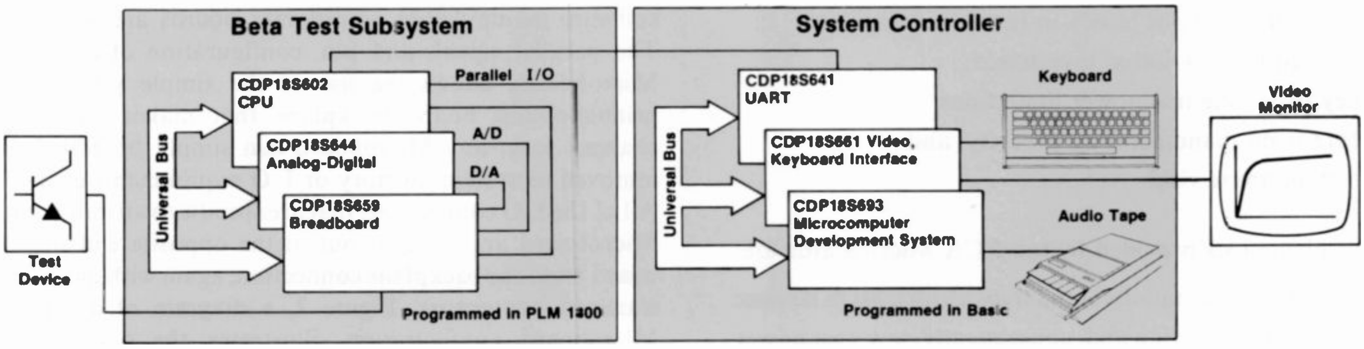


Fig. 1. Block diagram of the test system.

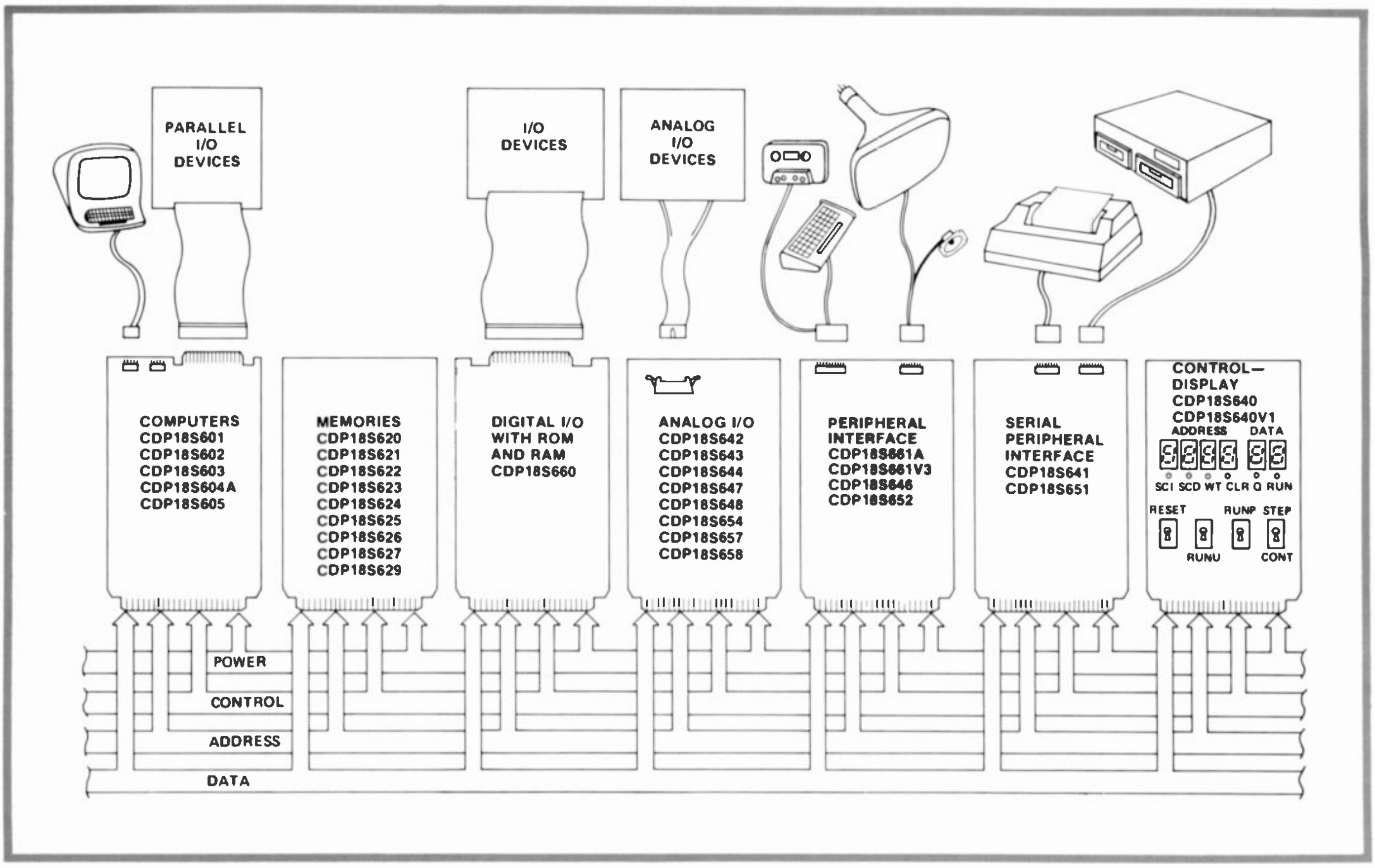

PARALLEL I/O DEVICES
I/O DEVICES
ANALOG I/O DEVICES
COMPUTERS CDP18S601 CDP18S602 CDP18S603 CDP18S604A CDP18S605
MEMORIES CDP18S620 CDP18S621 CDP18S622 CDP18S623 CDP18S624 CDP18S625 CDP18S626 CDP18S627 CDP18S629
DIGITAL I/O WITH ROM AND RAM CDP18S660
ANALOG I/O CDP18S642 CDP18S643 CDP18S644 CDP18S647 CDP18S648 CDP18S654 CDP18S657 CDP18S658
PERIPHERAL INTERFACE CDP18S661A CDP18S661V3 CDP18S646 CDP18S652
SERIAL PERIPHERAL INTERFACE CDP18S641 CDP18S651
CONTROL— DISPLAY CDP18S640 CDP18S640V1
ADDRESS
DATA
SCI SCD WT CLR Q RUN
RESET
RUNP
STEP
RUNU
CONT
POWER
CONTROL
ADDRESS
DATA

Fig. 2. Diagram of typical Microboard-system complement.

Why use Microboards?

Only a few units of a piece of equipment like the subject tester are usually built, and those, in many cases, by people familiar with the product to be tested rather than microprocessor circuit design. One way of assuring a more successful product under these conditions is the use of standard board-level products, in this case the RCA Microboards.

Some of the more common reasons given for using Microboards include:

- they can be used by those who lack microprocessor circuit-design expertise;
- they give fast turnaround;
- low volume cannot justify in-house development;
- they require low initial investment;
- they overcome manpower limitations;
- there is no manufacturing capacity; and
- they minimize risk.

The advantages provided by the RCA Microboards are:

- the high noise immunity standard with all CMOS devices;
- low-power operation and, consequently, low-cost power-supply requirements;
- the compactness of the 4.5-by-7.5-inch boards;
- easy system modification through the use of the universal backplane (described below); and
- good support from a full line of compatible hardware and software products.

The Universal Backplane

Since the requirements of the test system include flexibility and expandability, the board interconnections must exhibit the same properties. The RCA Universal Backplane is of an industry-standard, 44-pin, card-edge design that accommodates all pins on all boards in a standard configuration. All control signals of the CDP1802 are available on the backplane, an advantage that minimizes software manipulation as different boards are accessed. The parallel signal, and pin, configuration of all RCA Microboards allows the use of the simple yet rugged printed-circuit board backplane that makes hardware changes easy; any Microboard can simply be added or removed as system memory or I/O requirements change. All of the I/O connections that are specific to an individual Microboard are brought out at the opposite end of the board from the backplane connection, again with industry-standard connectors. Figure 2, a diagram of a typical Microboard configuration, illustrates the board connections.

Circuit operation

On power up, both the system controller and test-subsystem processors are reset, and an initialization program for each is stored in PROM. The operator is prompted to insert a transistor in the test socket and key an input on the keyboard to start testing. The system-controller calculates test words, which are sent over a serial line to the test subsystem. The test-subsystem microprocessor decodes these words, sets up the proper conditions on the transistor and causes a measurement (reading) to be taken. Based on the reading, the test-subsystem processor may change some of the conditions and take additional readings. If a proper reading is achieved, data words are sent back; if not, an error code is returned. The system-controller processor then stores the reading and sends another set of data words. This action continues until all of the tests are performed. The system-controller calculates the beta, compares it to preprogrammed limits, and outputs a pass or fail message to the video display. The operator can then have the equipment enter a diagnostic mode, can retest the same device, or can insert a new one.

System controller

Hardware

The system controller is designed to calculate and transmit the data words to the test subsystem, receive and decode the data words, calculate pass/fail information, receive inputs from the keyboard, and update the video monitor as new information becomes available. Because the controller must be interactive with the user and easily expanded or changed, an interpretive language was chosen as its means of communications with the system. A requirement of this language was that it have the floating-point arithmetic capability required to perform arithmetic calculations. RCA's Basic3 meets these criteria and, in addition, is available in the Microcomputer Development System Microboard, CDP18S693. A CDP18S641 UART (universal asynchronous receiver/transmitter) Microboard is used to provide communication (it transmits test and data words) over the RS232 link with the remainder of the system. A CDP18S661A Audio, Video, Keyboard-Interface Microboard is used for keyboard and color-video-monitor control; all of the video is in color. These boards, together with those in the MCDS (microcomputer development system), the CDP18S601 CPU Microboard and the CDP18S652 ROM, RAM, Cassette-Interface Microboard, complete the board complement of the system controller.

All system Microboards are housed in a CDP18S676 chassis with cover; the dc voltages — +15, −15 and +5 volts — are brought into the rear of the chassis. The RS232 link cable plugs into the CDP18S602 board of the test-subsystem.

Software

During the initial software development for the system controller, the developmental version of Basic3 was used, so that it was possible to encode and run the program in real-time. Once the code was completed, it was burned into EPROM and the Run Time Basic PROMs were substituted for the developmental version. The Interpreter PROMs occupy 12 kilobytes (kbytes) of memory, the Basic program occupies 2 kbytes, and the text for the video output takes 4 kbytes. Approximately 1 kbyte of RAM is used for stack, variable storage and work area.

On reset, the program initializes the UART board and the video board, outputs a message on the video monitor and waits for command input from the operator to start testing. On command, the system controller calculates and sends test words to the test subsystem and receives and stores the data words. After the testing is complete, the beta of the transistor is calculated for each test point and compared to a limit. A pass or fail message is output to the video, and the system waits for an input telling it to retest or enter the diagnostic mode. Figure 3 is a portion of the Basic3 program, specifically, the single-step diagnostic mode.

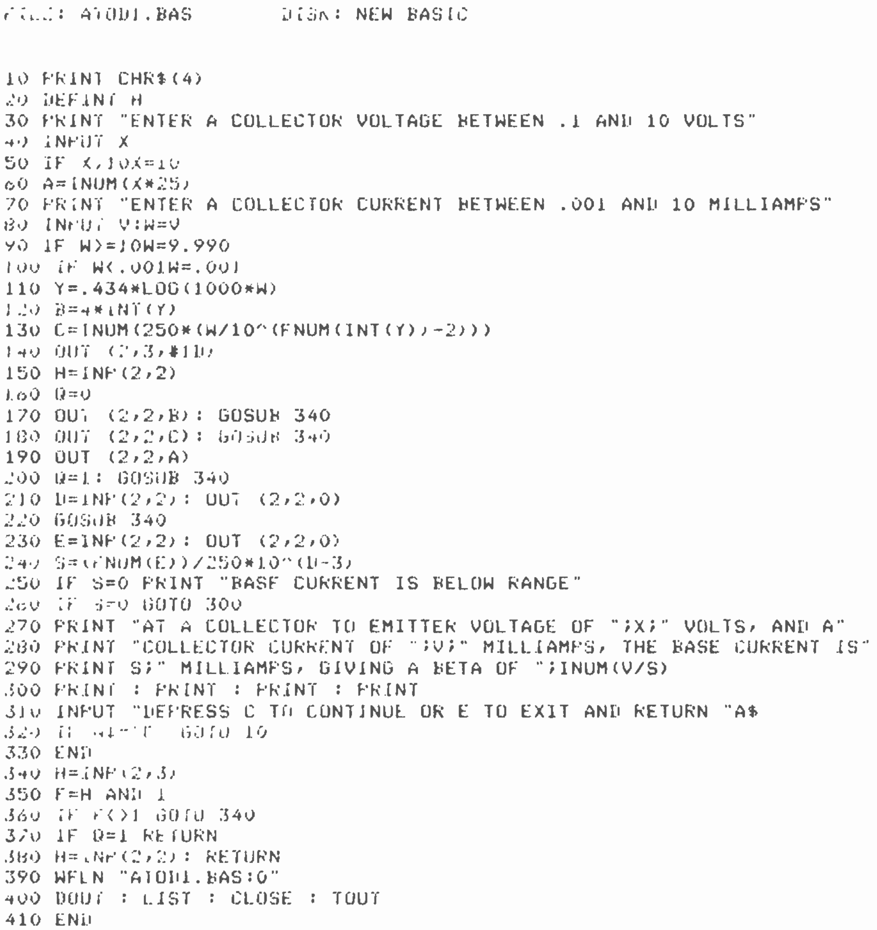
Diagnostics

Three choices of diagnostic mode are available: single step, table of values, and graph of beta-versus-collector current. In the single-step mode, the operator is prompted to input a collector-emitter voltage and collector current. From these values, the system controller calculates the test words and transmits them to the test subsystem, which performs the test and returns the data words. The data is decoded and beta is calculated; the base current and beta are displayed for the operator. The second option will display a table of the value of collector current, base current, and beta as found during the pass/fail testing. The final option presently uses the graphics capability of the CDP18S661A Video Microboard. A graph of beta-versus-collector current is calculated from the pass/fail test data and displayed one video dot at a time by this board, a practice that produces a smooth and continuous curve.

Beta-test subsystem

Hardware

The beta-test portion of the test system is required to accept and decode test words, set up the test parameters for the transistor under test, read the results, reset the test parameters, if necessary, and finally transmit the results. A latch was needed to store the emitter-current-range bits for the test. A power-on reset assures that the system comes-up running, and ROM and RAM are used to store the program and provide for stack area. All of these capabilities are available on the CDP18S602 CPU Microboard. The transistor under test is connected into the system through a CDP18S659 Microboard breadboard.



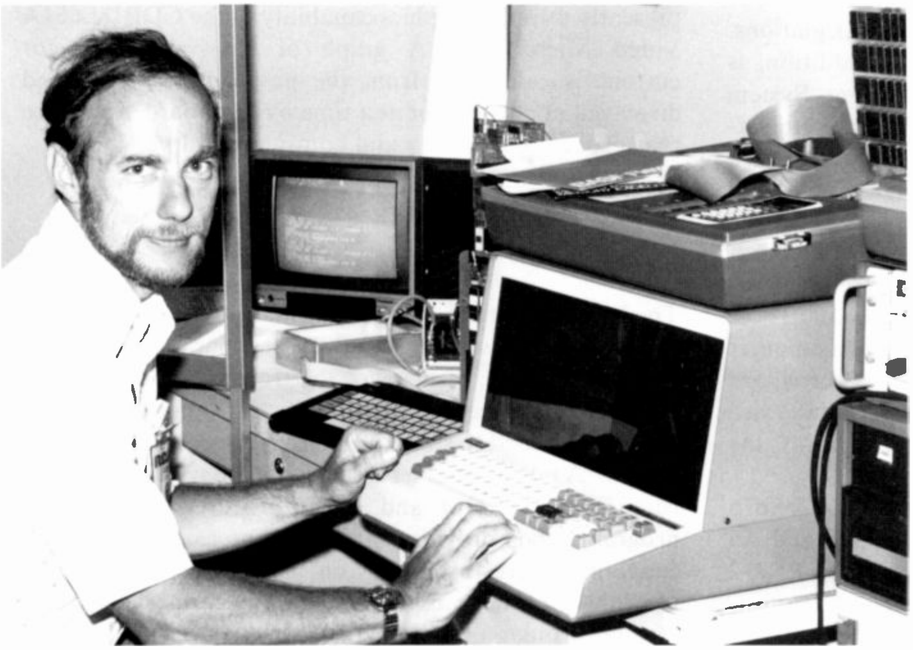
```
FILE: ATODI.BAS        DISK: NEW BASIC

10 PRINT CHR$(4)
20 DEFINT H
30 PRINT "ENTER A COLLECTOR VOLTAGE BETWEEN .1 AND 10 VOLTS"
40 INPUT X
50 IF X>10X=10
60 A=INUM(X*25)
70 PRINT "ENTER A COLLECTOR CURRENT BETWEEN .001 AND 10 MILLIAMPS"
80 INPUT V:W=V
90 IF W>=10W=9.990
100 IF W<.001W=.001
110 Y=.434*LOG(1000*W)
120 B=4*INT(Y)
130 C=INUM(250*(W/10^(FNUM(INT(Y))-2)))
140 OUT (2,3,#1D)
150 H=INP(2,2)
160 Q=0
170 OUT (2,2,B): GOSUB 340
180 OUT (2,2,C): GOSUB 340
190 OUT (2,2,A)
200 Q=1: GOSUB 340
210 D=INP(2,2): OUT (2,2,0)
220 GOSUB 340
230 E=INP(2,2): OUT (2,2,0)
240 S=(FNUM(E))/250*10^(D-3)
250 IF S=0 PRINT "BASE CURRENT IS BELOW RANGE"
260 IF S=0 GOTO 300
270 PRINT "AT A COLLECTOR TO EMITTER VOLTAGE OF ";X;" VOLTS, AND A"
280 PRINT "COLLECTOR CURRENT OF ";V;" MILLIAMPS, THE BASE CURRENT IS"
290 PRINT S;" MILLIAMPS, GIVING A BETA OF ";INUM(V/S)
300 PRINT : PRINT : PRINT : PRINT
310 INPUT "DEPRESS C TO CONTINUE OR E TO EXIT AND RETURN "A$
320 IF A$="C" GOTO 10
330 END
340 H=INP(2,3)
350 F=H AND 1
360 IF F<>1 GOTO 340
370 IF Q=1 RETURN
380 H=INP(2,2): RETURN
390 WFLN "ATODI.BAS:0"
400 DOUT : LIST : CLOSE : TOUT
410 END
```

Fig. 3. The single-step diagnostic-mode portion of the Basic3.

The V_{CE} voltage-test word and the emitter-current word must be converted into analog signals before they can be applied to the transistor. In the subject system, eight bits of data are sufficient for the desired accuracy. The two D/A channels in the CDP18S644 A/D and D/A converter Microboard handle this conversion. The CDP18S644 also has sixteen 8-bit A/D channels, one of which is used to read the voltage across the base sense resistors to provide a base-current multiplier. The values of the multiplier and the base-current range are transmitted through the UART to the system controller, which outputs the value of the base current.

Software

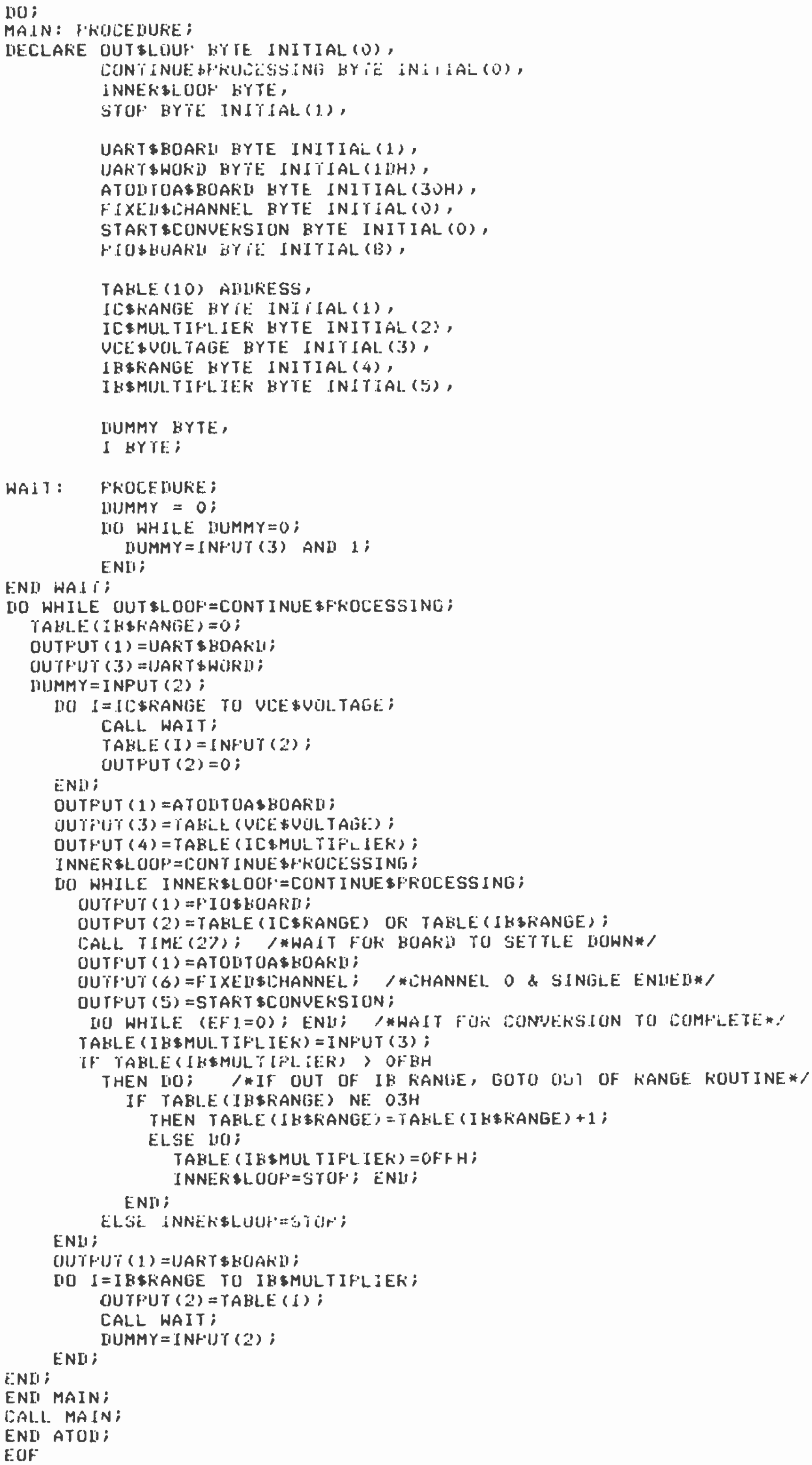
The software program for this part of the system was written on a CDP18S008 CDSIV Development System in PLM 1800. PLM was chosen because of the ease of coding in the higher-level language, and particularly because of the I/O constructs in this language that permit direct communication with the I/O Microboards. On reset the PLM program starts automatically, initializes the hardware, and waits for a signal from the UART to the effect that it



William Schilp is a Member, Technical Staff, in the Microsystems Engineering Department of the Solid State Division at Somerville. He joined RCA in 1964 at the RCA Laboratories in Princeton in advanced silicon-device process research. In 1969, he moved to the Solid State Division and became involved in power-transistor design engineering. Mr. Schilp was responsible for the design of a number of power transistors including most of the radiation-hardened power devices. In March 1980 he transferred to the applications area of the microsystems group.

Contact him at:
Solid State Division
Somerville, N.J.
TACNET: 325-6369

has received data. After receiving the third data word, all three words are sent to the A/D and D/A converter board and the parallel I/O port of the CDP18S602 board, which sets up the voltage and current for the transistor. The A/D converter is instructed to read the sense resistor and convert the data, which is then transmitted to the system controller via the UART. The program then loops back and waits for another set of inputs. Figure 4 is the PLM source code of the program. The assembled code occupies approximately 3 kbytes of PROM and uses less than one page of RAM area.



```
DO;
MAIN: PROCEDURE;
DECLARE OUT$LOUP BYTE INITIAL(0),
        CONTINUE$PROCESSING BYTE INITIAL(0),
        INNER$LOOP BYTE,
        STOP BYTE INITIAL(1),

        UART$BOARD BYTE INITIAL(1),
        UART$WORD BYTE INITIAL(1DH),
        ATODTOA$BOARD BYTE INITIAL(30H),
        FIXED$CHANNEL BYTE INITIAL(0),
        START$CONVERSION BYTE INITIAL(0),
        PIO$BOARD BYTE INITIAL(8),

        TABLE(10) ADDRESS,
        IC$RANGE BYTE INITIAL(1),
        IC$MULTIPLIER BYTE INITIAL(2),
        VCE$VOLTAGE BYTE INITIAL(3),
        IB$RANGE BYTE INITIAL(4),
        IB$MULTIPLIER BYTE INITIAL(5),

        DUMMY BYTE,
        I BYTE;

WAIT:   PROCEDURE;
        DUMMY = 0;
        DO WHILE DUMMY=0;
          DUMMY=INPUT(3) AND 1;
        END;
END WAIT;
DO WHILE OUT$LOOP=CONTINUE$PROCESSING;
  TABLE(IB$RANGE)=0;
  OUTPUT(1)=UART$BOARD;
  OUTPUT(3)=UART$WORD;
  DUMMY=INPUT(2);
    DO I=IC$RANGE TO VCE$VOLTAGE;
        CALL WAIT;
        TABLE(I)=INPUT(2);
        OUTPUT(2)=0;
    END;
    OUTPUT(1)=ATODTOA$BOARD;
    OUTPUT(3)=TABLE(VCE$VOLTAGE);
    OUTPUT(4)=TABLE(IC$MULTIPLIER);
    INNER$LOOP=CONTINUE$PROCESSING;
    DO WHILE INNER$LOOP=CONTINUE$PROCESSING;
      OUTPUT(1)=PIO$BOARD;
      OUTPUT(2)=TABLE(IC$RANGE) OR TABLE(IB$RANGE);
      CALL TIME(27);  /*WAIT FOR BOARD TO SETTLE DOWN*/
      OUTPUT(1)=ATODTOA$BOARD;
      OUTPUT(6)=FIXED$CHANNEL;  /*CHANNEL 0 & SINGLE ENDED*/
      OUTPUT(5)=START$CONVERSION;
       DO WHILE (EF1=0); END;  /*WAIT FOR CONVERSION TO COMPLETE*/
      TABLE(IB$MULTIPLIER)=INPUT(3);
      IF TABLE(IB$MULTIPLIER) > 0FBH
        THEN DO;   /*IF OUT OF IB RANGE, GOTO OUT OF RANGE ROUTINE*/
          IF TABLE(IB$RANGE) NE 03H
            THEN TABLE(IB$RANGE)=TABLE(IB$RANGE)+1;
            ELSE DO;
              TABLE(IB$MULTIPLIER)=0FFH;
              INNER$LOOP=STOP; END;
          END;
        ELSE INNER$LOOP=STOP;
    END;
    OUTPUT(1)=UART$BOARD;
    DO I=IB$RANGE TO IB$MULTIPLIER;
        OUTPUT(2)=TABLE(I);
        CALL WAIT;
        DUMMY=INPUT(2);
    END;
END;
END MAIN;
CALL MAIN;
END ATOD;
EOF
```

Fig. 4. PLM source code of the program.

System expansion

With additional software only, the beta-test subsystem can be made to perform base-emitter voltage (V_{BE} active) testing. By adding additional CDP18S641 UART Microboards to the system controller, additional subsystems can be designed and added to measure other transistor parameters, such as emitter-base and emitter-collector saturation voltages, junction breakdown voltages, and junction leakages. Each subsystem can be designed and built as a module and, since each has its own microprocessor, can, with the overall supervision of the system controller, test, make some first-level decisions, and pass back only final data, thus speeding up the overall process of testing.

Acknowledgments

The author appreciates the assistance of R. Isham in the form of discussions of general design concepts and, in particular, for his design of the test-transistor interface board.

J.W. Woestman

Slurry-control equipment uses microprocessor

A microprocessor gives dynamic adjustment of the slurry composition used on color picture tubes at every dispense so that specific gravity and viscosity are maintained continuously.

Abstract: *The microprocessor system controls the specific gravity and viscosity of the phosphor slurry that is deposited on the top of the picture tube. The program repeatedly calculates errors or imbalances in the slurry, water, and PVA makeup.*

In the manufacture of the cap, or panel, on the front of a color picture tube, the three color phosphors are deposited individually on the inner surface in a process known as screening. The phosphor crystals of a single color are combined in a batch of slurry together with water, polyvinyl alcohol (PVA), and other stabilizing and wetting agents.

The established method of screening requires that the slurry be generously dispensed into the cap and distributed all across its inside frontal surface. Excess slurry is salvaged and returned to the mixture both because a greater number of caps can be screened per batch and because the phosphor crystals are too costly to scrap.

When the salvaged slurry is combined with the main slurry stock, the specific gravity and viscosity of.the. batch is modified because the diminished phosphor content lowers the specific gravity and viscosity. For optimum screening conditions, specific gravity and viscosity should remain constant.

The practical compromise has been to maintain the slurry level in the six-liter charge vessel by replenishing its contents from a filler vessel. The specific gravity and viscosity of the slurry in the filler vessel are above bogie, so that, combined with the salvage, the slurry composition remains within an acceptable range until exhausted.

The introduction of a microprocessor to screening provides for dynamic adjustment of the slurry composition at every dispense so that the specific gravity and viscosity is maintained continuously very near to bogie values.

Control implementation

In order to provide for continual adjustment of the slurry composition in the charge vessel, three stocks of liquid are located close-by. The filler vessel contains slurry having very high density and slightly high viscosity. Another vessel contains a highly viscous PVA mixture, and the third vessel contains water. The quantity of each of these three liquids added to the charge vessel at each dispense is determined and controlled by the microprocessor. Figure 1 depicts the arrangement of the slurry equipment.

Solenoid valves at each of the three slurry vessels control the quantity of make-up slurry added to the charge vessel after every dispense to a cap. The exact quantity of a liquid is determined by the length of time its valve is open. The flow rate from any vessel is a function of the viscosity, density, and head of the liquid (all valves are the same size). This flow information is stored in the microprocessor program. It is fixed for the water and PVA liquids, but is operator-adjustable for the filler slurry (described later). The quantity of liquid from each vessel is that required to restore or maintain the charge slurry at bogie specific gravity and viscosity, and to maintain the level of slurry in the charge vessel at a predetermined height. The latter is important so that the calculation of quantities necessary to reduce errors to zero can be based on a (relatively) fixed quantity of slurry in the charge vessel. The system operates in a controlled environment and corrections for temperature and humidity are not necessary. The quantity of make-up slurry is also adjustable by the operator to match different size caps (13- to 25-inch color picture tubes) Dynatrol™ specific gravity and viscosity sensors provide variable input signals.

The control electronics are housed in a metal cabinet shown in Fig. 2. The front panel has controls and indicators for putting the system in operation and for monitoring it. The system normally operates unattended once the set points have been entered. Alarms are provided to signal the operator if the slurry specific gravity, viscosity, or level exceed predetermined limits. Indicator LEDs tell when valves are open,

Reprint RE-26-8-8
Final manuscript received June 19, 1981.

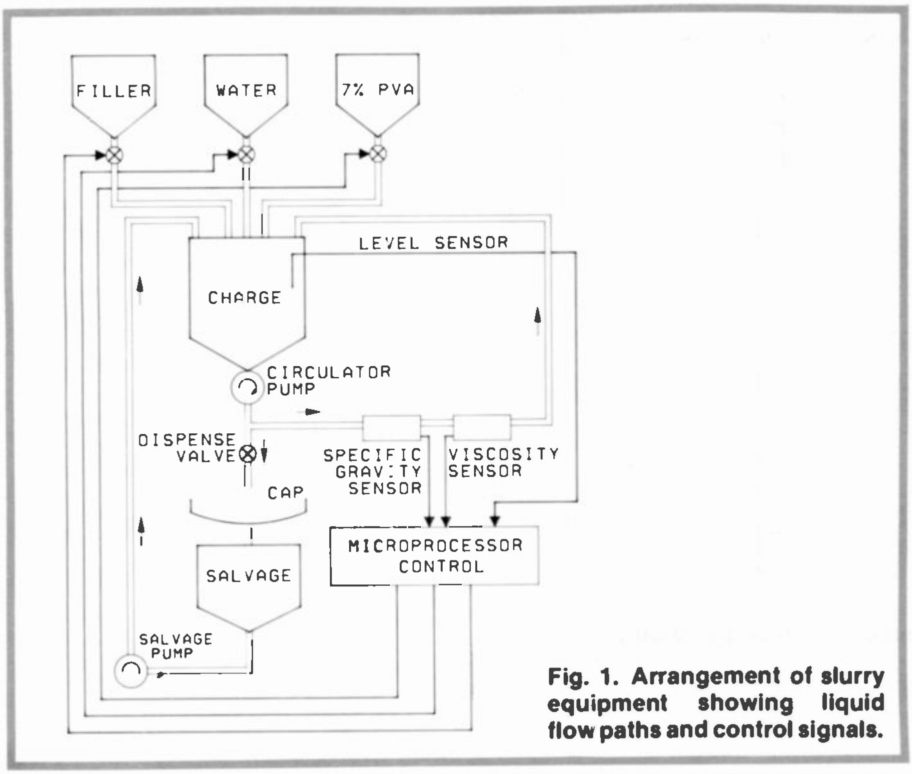


Fig. 1. Arrangement of slurry equipment showing liquid flow paths and control signals.

when high and low variable limits are exceeded, when set points are illegal, and whether set points or sensor readings are being presented on the display. Specific gravity and viscosity set points are entered by thumbwheel switches. A 3½-digit display indicates specific gravity reading or set point, viscosity reading or set point, filler slurry flow-rate setting, and dispense-volume setting.

Electronic components are an Intel iSBC 80/20-4 microprocessor board, an eight-channel A/D converter board having a two-channel D/A converter as well (Burr Brown MP8413), a multi-voltage regulated power supply (Intel iSBC 655) and optically isolated solid-state relays for operating the valves.

The slurry-control equipment uses 23 input and 19 output ports of the available 48 I/O ports. One-hundred-and-eighty bytes of RAM and over 5200 bytes of ROM are used in the program. Only four of the eight analog input channels are used and one of the two D/A converter outputs is used for display on a digital panel meter. Although this method of display is somewhat unusual, it is economical. The D/A converter channel was provided anyway, as part of the analog board; the digital panel meter, being a high volume component, is inexpensive; and output is provided with only a few lines of code in the program. The program contains four sections, initialization, valve timing calculations, indicator and alarm servicing, and thumbwheel and display servicing. A modified form of BASIC named SLAM is the programming language used.

In the initialization section, software programming of the Intel microprocessor board is done and initial values are assigned to particular variables. Also, default set points for specific gravity and viscosity are set.

In the section for valve timing calculations, the four analog input channels are read and the time for each valve to be open is calculated. The algorithm for this calculation first determines the specific gravity and viscosity error. It then determines the volume of water, PVA, and filler to correct the error. These volumes are compensatory. For example, if X ml of extra filler slurry are needed to correct a specific gravity error, then X ml of water will be subtracted from the total water added. The total of all three of the liquids contributing to a dispense is determined by the size of cap being screened. These volumes range from 50 to 150 ml. In addition, the total added volume is increased or decreased to maintain the prescribed level in the charge vessel. Constants and parameters for the algorithm were determined first by calculation. They were then refined by extensive testing in a mock set-up. Operators began inputting the filler flow rate when it was found that filler flow rate varied from batch to batch.

Valve timing calcuations are made repeatedly by the program. But, when a dispense to the cap is made, the calculations are suspended until all of the valves have closed. That occurs when the replenishing of the slurry dispensed from the charge vessel has been completed.

Timing for the valves is done by

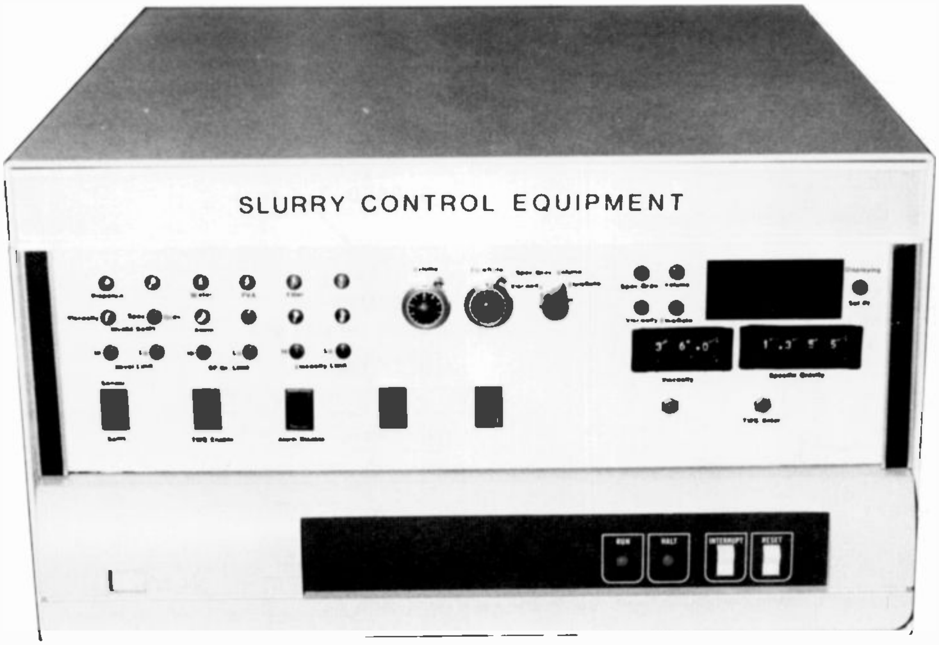


Fig. 2. Control cabinet showing location of front panel.

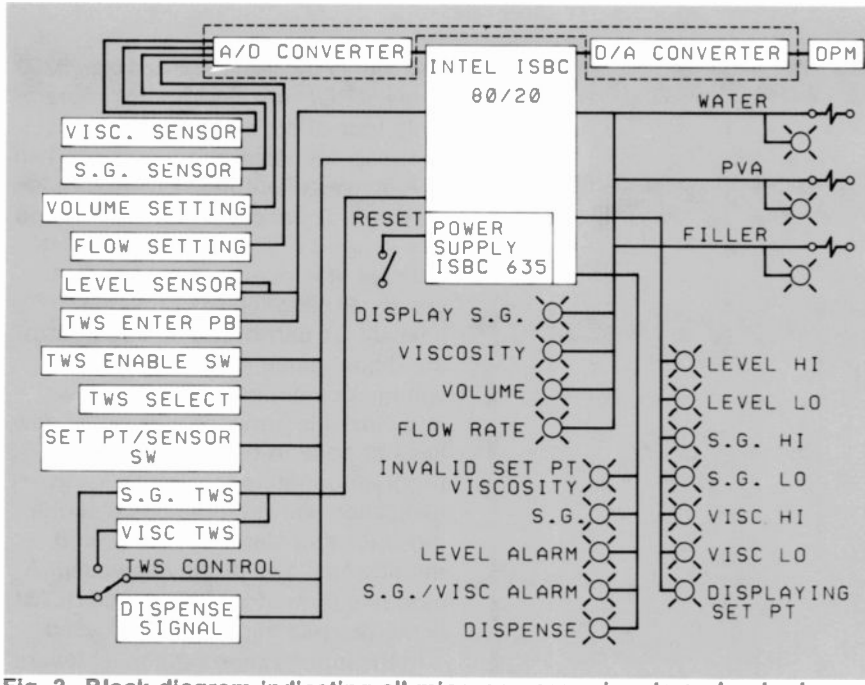
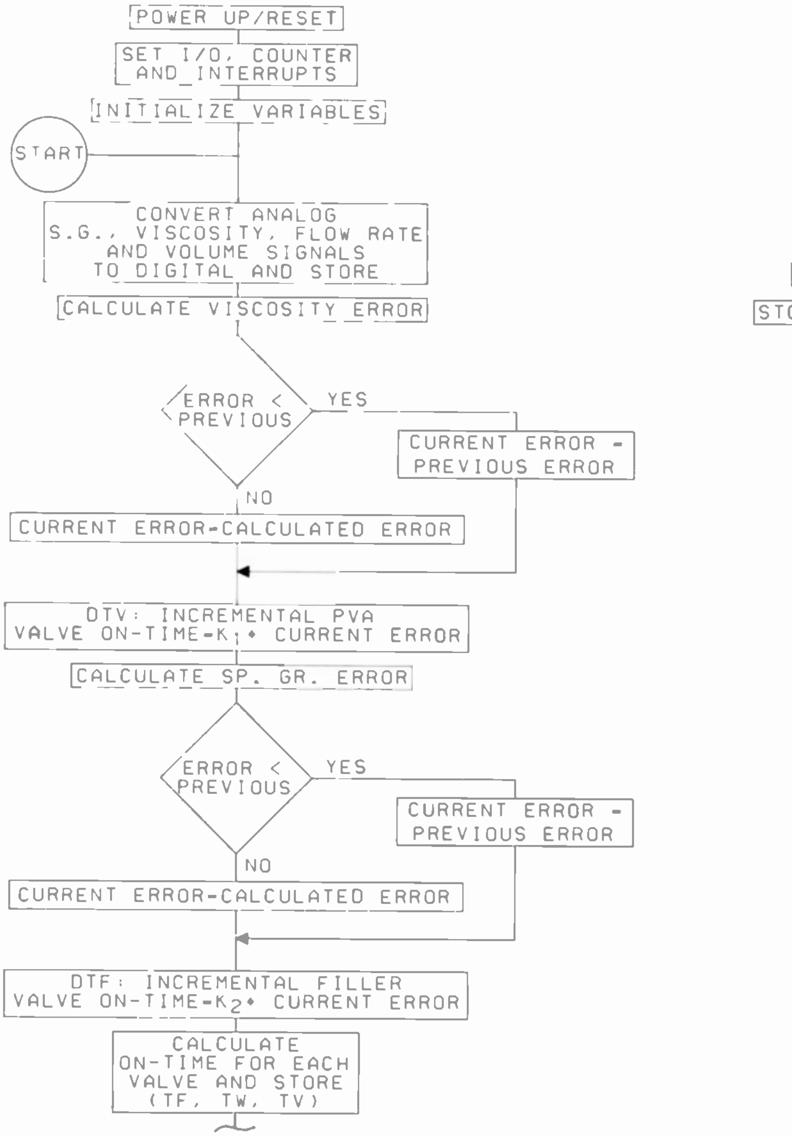


Fig. 3. Block diagram indicating all microprocessor inputs and outputs.


POWER UP/RESET
SET I/O, COUNTER AND INTERRUPTS
INITIALIZE VARIABLES
START
CONVERT ANALOG S.G., VISCOSITY, FLOW RATE AND VOLUME SIGNALS TO DIGITAL AND STORE
CALCULATE VISCOSITY ERROR
ERROR < PREVIOUS
YES
CURRENT ERROR - PREVIOUS ERROR
NO
CURRENT ERROR-CALCULATED ERROR
DTV: INCREMENTAL PVA VALVE ON-TIME-K1* CURRENT ERROR
CALCULATE SP. GR. ERROR
ERROR < PREVIOUS
YES
CURRENT ERROR - PREVIOUS ERROR
NO
CURRENT ERROR-CALCULATED ERROR
DTF: INCREMENTAL FILLER VALVE ON-TIME-K2* CURRENT ERROR
CALCULATE ON-TIME FOR EACH VALVE AND STORE (TF, TW, TV)

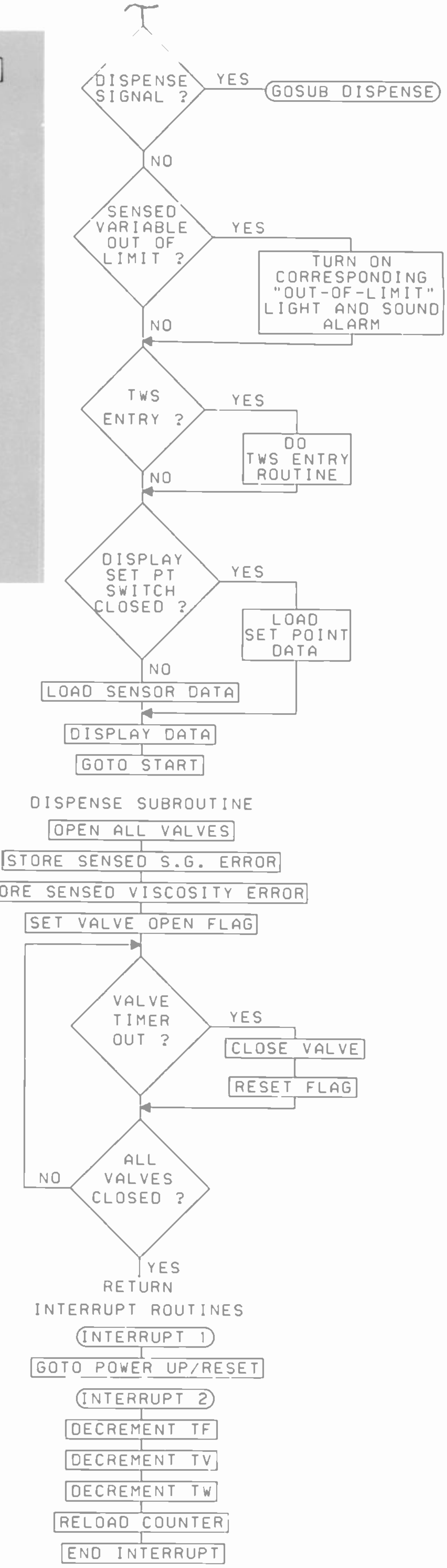


Fig. 4. Generalized flowchart with principal decision paths of the program

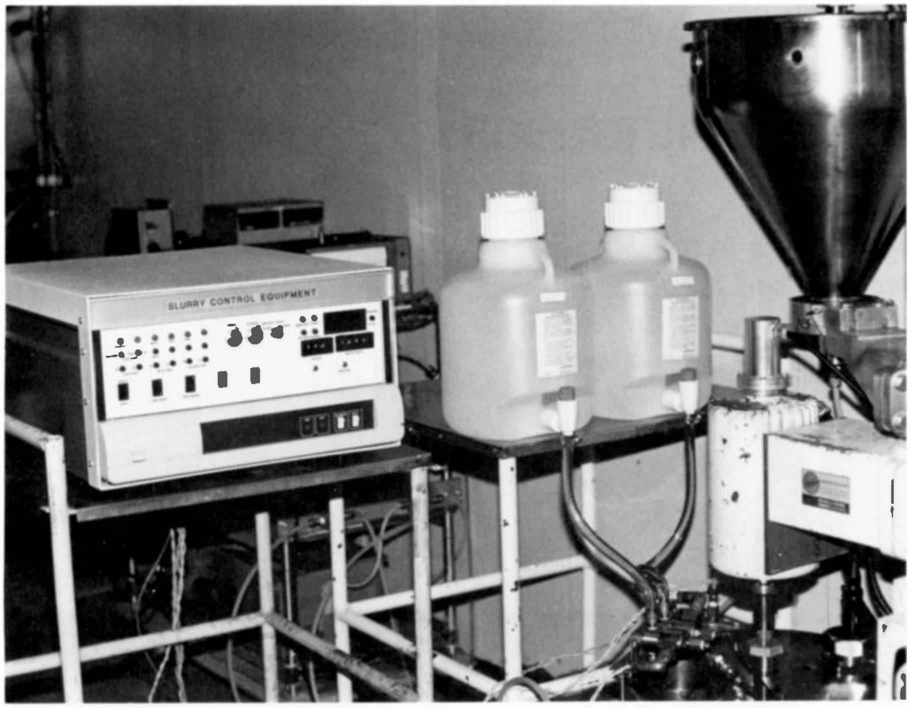


Fig. 5. Test setup used for laboratory simulation. Main charge vessel is at lower right.

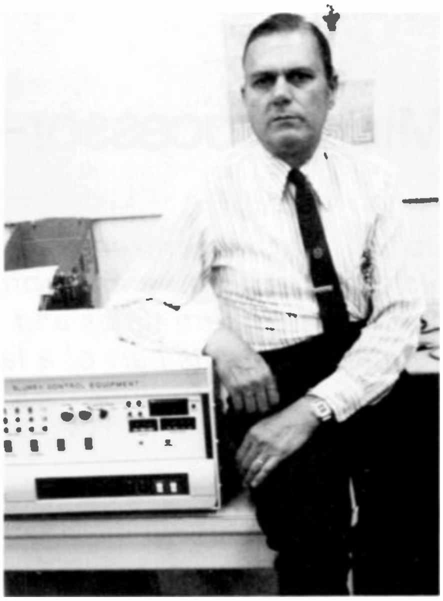
using one of the four counters on the Intel 80/20 board. During initialization, counter #1 is loaded with a value that provides a countdown to zero in 0.01 second. When the counter reaches zero, it generates an interrupt. Upon interrupt, the variables representing the valve on-times are each decremented. Then the counter is reloaded, and the interrupt ended. Thus, to open a valve for one second, the program will load 100 into the variable for that valve.

For indicator and alarm servicing, the sensed values of specific gravity and viscosity are compared to high and low limits. If any limit is exceeded, the appropriate indicator light on the panel is lit, and an alarm is sounded. Level is also sensed, but this is not an analog signal. Level is sensed as too high, too low or neither of these. The level sensor is essentially two switches. The alarm for level-limits exceeded has a different sound from the other limit alarm.

If panel switches for activating the thumbwheel switches and display are appropriately set, the program enters the routines to service them. Set points entered on the thumbwheel switches are entered into the program when the "enter" pushbutton is depressed. The value entered is echoed by the display unless it is an out-of-limit value. If an out-of-limit value is put on the thumbwheel switch, a red "set-point out-of-limit" light comes on, the value is not entered, and the previous set point is retained. Display of sensed or set-point values is obtained by front panel switches. Set points for cap size and filler flow rate are entered by potentiometer settings. Values entered are displayed.

Figure 3 is a block diagram of the control and Fig. 4 is a generalized flow chart.

The equipment has been extensively tested in the laboratory where the algorithm was developed and refined. Operation simulating production conditions demonstrated potential to control specific gravity to ±0.05 and viscosity to ±2 centipoise. A prototype unit is being readied for test at the Marion, Indiana tube plant. A photograph of the test setup is shown in Fig. 5.



Joh- Woestman spent three years in the development of commercial equipment using microprocessors prior to joining the Picture Tube Division as Member Technical Staff in August, 1978. He received his BSEE from the University of Cincinnati in 1949. He has had design experience in transistor circuits, and in infrared and optical sensors for military and spacecraft applications.

Contact him at:
Picture Tube Division
Lancaster, Pa.
TACNET: 227-2435

E.J. Alvero

Microprocessor-based lighthouses

During the mechanized screening process in the Marion picture tube plant, a microprocessor-based machine controls exposure times and motor movements on the lighthouse, and is part of a larger, coordinated system control.

Abstract: *Automated sliptop lighthouse equipment is part of the mechanized, picture-tube screening process at Marion. During the time that the panel is on the lighthouse for exposure to ultraviolet light, several local inputs determine the exposure time. In addition, the panel is moving with respect to the light source while the panel is being exposed. A microprocessor was selected to control the exposure time and to control motor movement. In addition, the automated lighthouse is a part of larger system control. The microprocessor communicates to a programmable controller.*

Manufacturers use "lighthouses" to make screens for color-TV picture tubes. Lighthouses are used to expose the photoresist coatings on the inside of the glass-panel screen with ultraviolet light. RCA is now using a microprocessor-based machine in the mechanized screening room in the picture-tube manufacturing plant at Marion, Indiana.

Normal procedures go like this. Before placing the panel on the lighthouse, the phosphor and photoresist coating is applied on a screening machine. After further processing, the mask-frame assembly, which has a graded aperture mask, is inserted into the panel. The panel is then unloaded from the screening machine and placed on the lighthouse. After exposure, the mask-frame assembly is removed. The unexposed photoresist is washed away, leaving a phosphor pattern on the screen. This procedure is repeated for green, blue, and red phosphors.

Motor movement is required

The light source is positioned mechanically on the lighthouse with respect to the panel in the x-axis and z-axis. The x-axis (S) mechanical placement is for the separation between colors for the particular tube. The z-axis (P) is set up for the distance between the light source and the seal end of the panel for the particular tube type. Previously, lighthouses were set up with the y-axis at optical center. No motor movement occurred during exposure. Thus, lighthouse control was only exposure-time control, achieved with electromechanical timers.

However, with the precision in-line matrix tube types, motor movement in the y-axis during exposure is desirable. There are several reasons for this. The first reason is due to the artwork of the mask. It is rectangularly slotted, and moving the light source during exposure eliminates "necking" caused by the vertical distance between mask slots (Fig. 1). An elongated light source and motor movement during exposure greatly reduces this problem. Secondly, motor movement during exposure minimizes any errors due to dirt spots on the optical lens during exposure. Finally, the tolerance of the mask assembly does not have to be as tight to produce good screens.

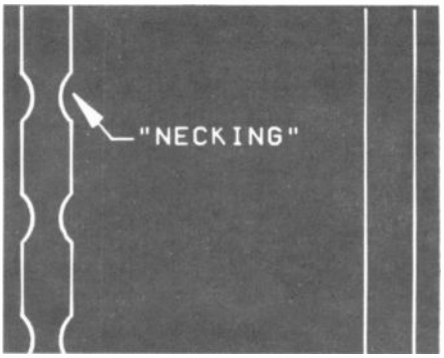


Fig. 1. Phosphor line with no movement (left), and phosphor line with motor movement on exposure (right).

Requirements

A new microprocessor-based lighthouse control needed to assure that the panel was exposed equally about the optical axis. The distance the lighthouse moved during exposure was to be variable with tube type, and externally adjustable. If the tube type was non-matrix, the lighthouse control had to provide exposure at optical center with no motor movement. Tighter control on exposure time was also desired. Exposure of the panel based on cell-size information was to be provided as in the old lighthouse control.

Based on the mask transmission of the mask-frame assembly, panels are divided into one of four cells. Each cell requires a different exposure. A panel with a mask assembly having a low mask transmission is exposed longer. Later refinements of the system included compensating for reduced lamp intensity by increasing exposure

Reprint RE-26-8-9
Final manuscript received June 19, 1981.

time and putting the lighthouse under a larger automated system control. Because the requirements seemed to involve much input data processing to produce an accurate distance movement while also maintaining an accurate exposure, a microprocessor-based system was selected. We chose an Intel 80/20 single-board computer, which has the 8080 microprocessor.

Figure 2 shows the block diagram of the lighthouse-control system. The system inputs fall into two categories: local input data from the lighthouse-control panel or lighthouse-machine base, and signals provided by external programmable controllers.

One external controller is the Modicon 484 PC used by a transfer unit. The transfer unit takes a panel off the screening machine and places it on an available lighthouse. The 484 passes a signal that tells the lighthouse to begin processing. It also provides 8-bit data that is either actual mask transmission for that panel, or cell-size information. A supervisory programmable controller provides the signal as to whether the data is mask transmission or cell-size data.

The main local inputs are: five 3-digit thumbwheels that give exposure and excursion distance data for the tube type being processed on the lighthouse; inputs from an incremental rotary encoder and a home limit switch, which does the positioning of the y-axis; and inputs used to provide exposure control based on the intensity of the mercury arc lamp used in exposing the panel. A toggle switch to determine whether to move during exposure is another important input. The five thumbwheel switches are multiplexed. Five lines are used to select the appropriate thumbwheel, and twelve lines are input to the computer. The program selects the thumbwheels to be read and performs checks to see if the inputs are within limits. Functions are as follows: (a) a switch that sets the minimum exposure preset time for the tube type being processed, (b) a switch that specifies the difference between the base exposure and the maximum exposure for the tube type, (c) two switches that specify the minimum and maximum transmission value for the tube type and that are used only if the data passed by the transfer unit is actual mask transmission, and (d) a switch that specifies the distance to be moved while exposing the panel.

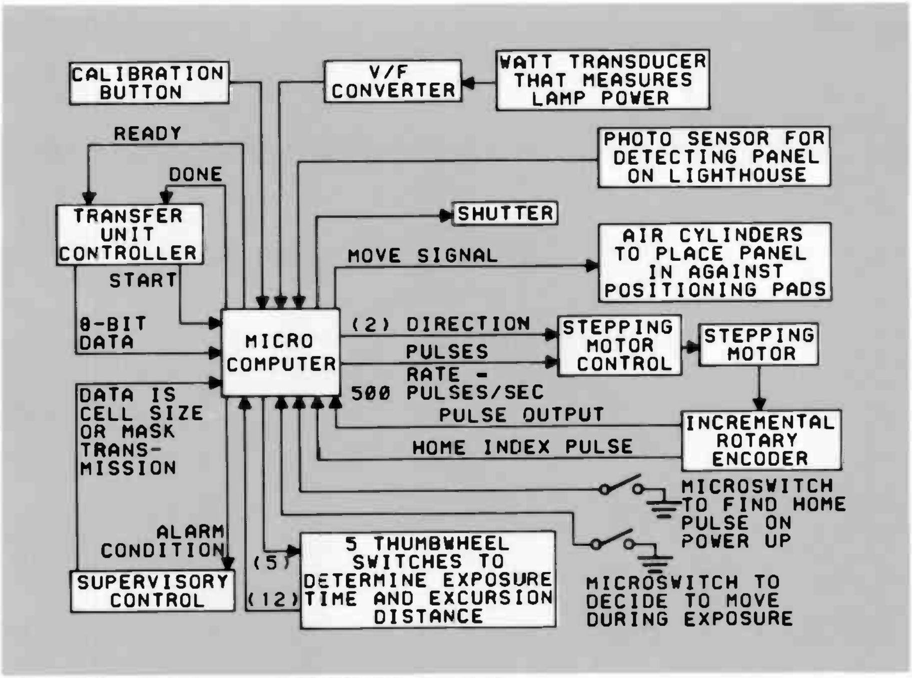

CALIBRATION BUTTON
V/F CONVERTER
WATT TRANSDUCER THAT MEASURES LAMP POWER
READY
DONE
PHOTO SENSOR FOR DETECTING PANEL ON LIGHTHOUSE
SHUTTER
TRANSFER UNIT CONTROLLER
MOVE SIGNAL
AIR CYLINDERS TO PLACE PANEL IN AGAINST POSITIONING PADS
START
8-BIT DATA
MICRO COMPUTER
(2) DIRECTION
PULSES
RATE = 500 PULSES/SEC
STEPPING MOTOR CONTROL
STEPPING MOTOR
DATA IS CELL SIZE OR MASK TRANS-MISSION
PULSE OUTPUT
INCREMENTAL ROTARY ENCODER
HOME INDEX PULSE
MICROSWITCH TO FIND HOME PULSE ON POWER UP
ALARM CONDITION
(5)
(12)
5 THUMBWHEEL SWITCHES TO DETERMINE EXPOSURE TIME AND EXCURSION DISTANCE
SUPERVISORY CONTROL
MICROSWITCH TO DECIDE TO MOVE DURING EXPOSURE

Fig. 2. Block diagram of lighthouse control.

A stepping motor creates movement on the y-axis. A closed-loop system ensures that the transfer unit always places the panel properly on the lighthouse. Signals provided by the encoder are the DATA signal and the "home" index pulse. The "home" index pulse occurs every revolution of the stepping motor shaft. There are 200 steps per revolution of the motor shaft. Each step moves the y-axis one mil (0.001 in.). Pulses provided to the stepping motor controller occur at a fixed rate of 500 pulses/second. Thus, the DATA signal of the encoder going to the computer is at that rate. A microswitch, positioned 0.775 in. from optical center, is used to find the home pulse on power-up. The computer turns the motor counterclockwise until the home index pulse from the encoder is found. That pulse is set up to be 0.675 in. from optical center.

The final important local input is the data to provide compensation for changes in lamp intensity. A watt transducer monitors the input power to the lamp. The lamp intensity is directly proportional to changes in lamp power. The analog voltage output from the transducer is converted to a frequency and input to the computer. That frequency (pulse train) is the clock to a 16-bit counter on the computer. To generate an exposure from a calculated preset time, the computer calculates how many pulses it must count from the V/F converter to get the desired exposure. It is performing this calculation based on what the frequency was when the lighthouse was calibrated. Since the lamp power needed to produce a given light-intensity output varies from lighthouse to lighthouse, a calibration feature was designed. The computer calculates the average frequency of the V/F converter over a certain time window when a "calibrate" pushbutton is pressed. That pushbutton is an external interrupt to the 80/20 computer. The principle behind the exposure compensation is fairly simple. As lamp power decreases, the frequency is reduced by the same amount. Since that frequency is clocking the counter that controls exposure, the exposure time is increased proportionally.

The outputs from the computer to the transfer-unit controller are two handshaking signals. One tells the transfer unit that the lighthouse is ready to accept another panel. It signals this if there is no panel on the lighthouse (sensed by photo sensors mounted just above where the panel is placed). The other signal calls for the panel's removal. An alarm signal is passed to the supervisory controller if any problems occur while the lighthouse is processing the panel. The local outputs are: the signals to open the shutter, the direction signals (clockwise and counterclockwise), and an output signal with a rate of 500 pulse/second that goes to the stepper

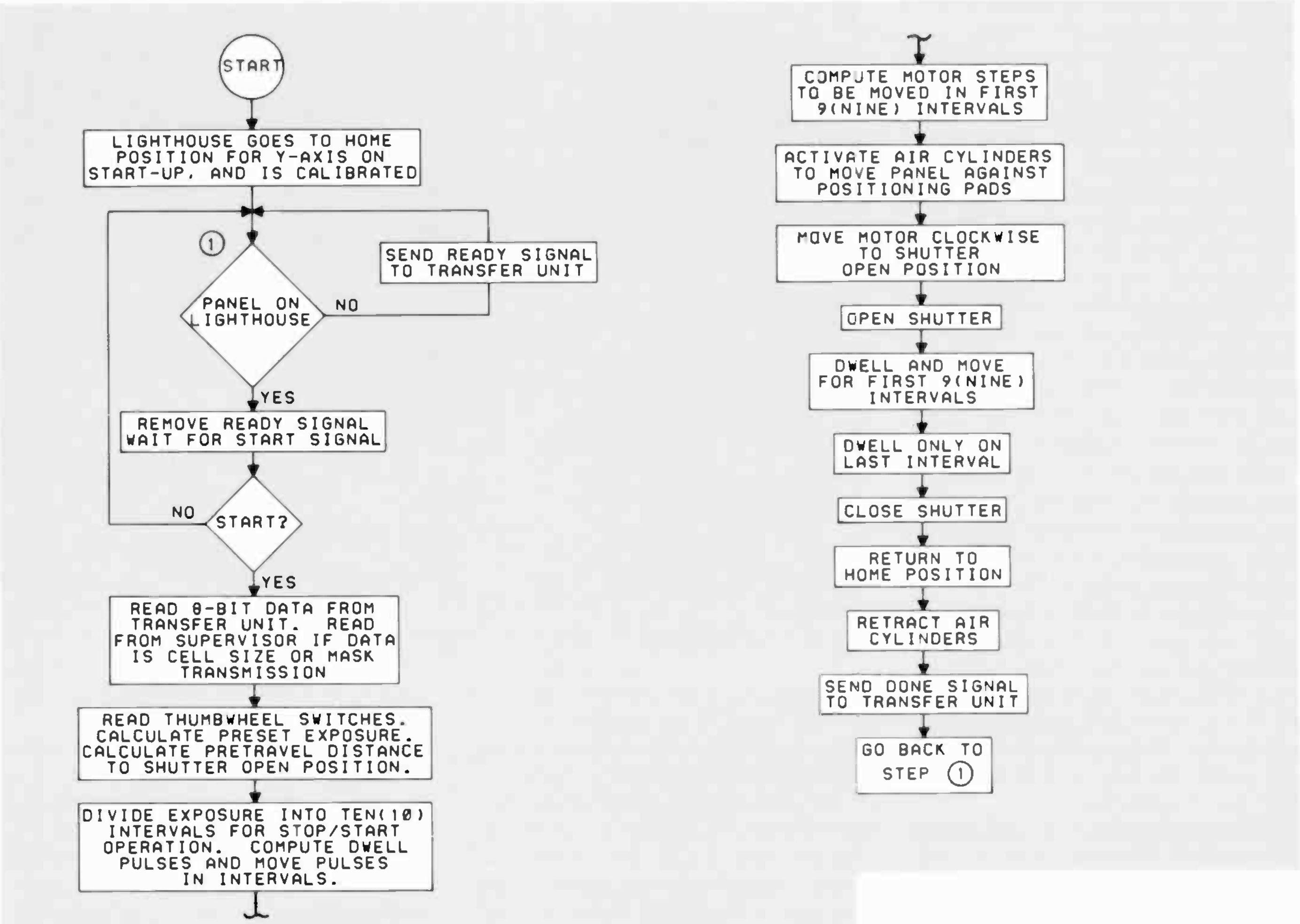

START
LIGHTHOUSE GOES TO HOME POSITION FOR Y-AXIS ON START-UP, AND IS CALIBRATED
1
PANEL ON LIGHTHOUSE
NO
SEND READY SIGNAL TO TRANSFER UNIT
YES
REMOVE READY SIGNAL WAIT FOR START SIGNAL
NO
START?
YES
READ 8-BIT DATA FROM TRANSFER UNIT. READ FROM SUPERVISOR IF DATA IS CELL SIZE OR MASK TRANSMISSION
READ THUMBWHEEL SWITCHES. CALCULATE PRESET EXPOSURE. CALCULATE PRETRAVEL DISTANCE TO SHUTTER OPEN POSITION.
DIVIDE EXPOSURE INTO TEN(10) INTERVALS FOR STOP/START OPERATION. COMPUTE DWELL PULSES AND MOVE PULSES IN INTERVALS.
COMPUTE MOTOR STEPS TO BE MOVED IN FIRST 9(NINE) INTERVALS
ACTIVATE AIR CYLINDERS TO MOVE PANEL AGAINST POSITIONING PADS
MOVE MOTOR CLOCKWISE TO SHUTTER OPEN POSITION
OPEN SHUTTER
DWELL AND MOVE FOR FIRST 9(NINE) INTERVALS
DWELL ONLY ON LAST INTERVAL
CLOSE SHUTTER
RETURN TO HOME POSITION
RETRACT AIR CYLINDERS
SEND DONE SIGNAL TO TRANSFER UNIT
GO BACK TO STEP 1

Fig. 3. Normal lighthouse operation flowchart.

control, and the signal to place the panel against positioning pads by activating air cylinders.

Figure 3 shows the basic flowchart for normal operating lighthouse control. A start-stop operation of the stepping motor is used because traveling a short distance continuously over the normal exposure range introduces a vibration on the mask. The exposure is divided into ten equal intervals. In the first nine intervals, there is a certain dwell time, then there is a move time in which the stepper moves one-ninth of the total excursion distance. The final interval is used to dwell. This scheme ensures equal exposure about the optical axis. The calculation to compute the number of steps it has to travel before opening the shutter is as follows: pretravel equals 675, minus the excursion distance divided by two. If the excursion is 1.00 in., then the excursion distance equals 1000 steps.

The calculation to compute exposure time from manual cell-size data is as follows: (1) the exposure range value is divided by four to set up an equal time range for each cell, (2) the cell-size 8-bit data is converted to a cell code (either 0, 1, 2 or 3), (3) the cell code is added to 0.5 to get the midrange of the cell and multiplied by the exposure range, and (4) that result is added to the base exposure.

After doing the data processing, the lighthouse exposes the panel and moves the panel with respect to the source. After finishing this task, the lighthouse sends a signal for the robot to place the panel back on the screening machine. These lighthouses, a part of the mechanized screening room in the RCA picture-tube manufacturing plant in Marion, Indiana, have shown satisfactory results. The time to produce a working system was short due to the use of the microcomputer.

Ernie Alvero is Member Technical Staff, Picture Tube Division, in Lancaster. He joined RCA in August 1976 in the Equipment Development section where he is currently employed. His technical activities include microprocessor-based lighthouse design, and he is currently developing computer-control system software for the horizontal mask-etch line. His BSEE is from City College of New York.

Contact him at:
Picture Tube Division
Lancaster, Pa.
TACNET: 227-3438

S. Malyszka

Microcomputers in space: Automatic thruster control for Satcom

Once Satcom D is in orbit, microcomputer architecture based on the RCA CDP1802 will help it keep its station.

The geostationary Satcom orbit degrades with time and must be periodically adjusted to maintain the satellite to within 0.1 degree of its assigned station. Orbit degradation is caused by the gravitational effects of the sun, earth, moon, and solar-radiation pressure. Orbit adjustment (called stationkeeping) is accomplished with 12 hydrazine thrusters. These thrusters are also used, during stationkeeping maneuvers, for satellite attitude control.

On earlier Satcoms, thruster control was semiautomatic, performed by ground commands in conjunction with on-board sequential logic. Satcom D will be the first RCA communications satellite to employ a microcomputer for fully automatic thruster control.

Abstract: *The attitude logic processor (ALP) is the microcomputer that will provide automatic thruster control on Satcom D. The author briefly describes the capabilities of the firmware, including interfaces and safety features.*

Automatic control minimizes ground intervention during stationkeeping maneuvers. These factors provide for more reliable service to customers and

Reprint RE-26-8-10
Final manuscript received July 28, 1981.

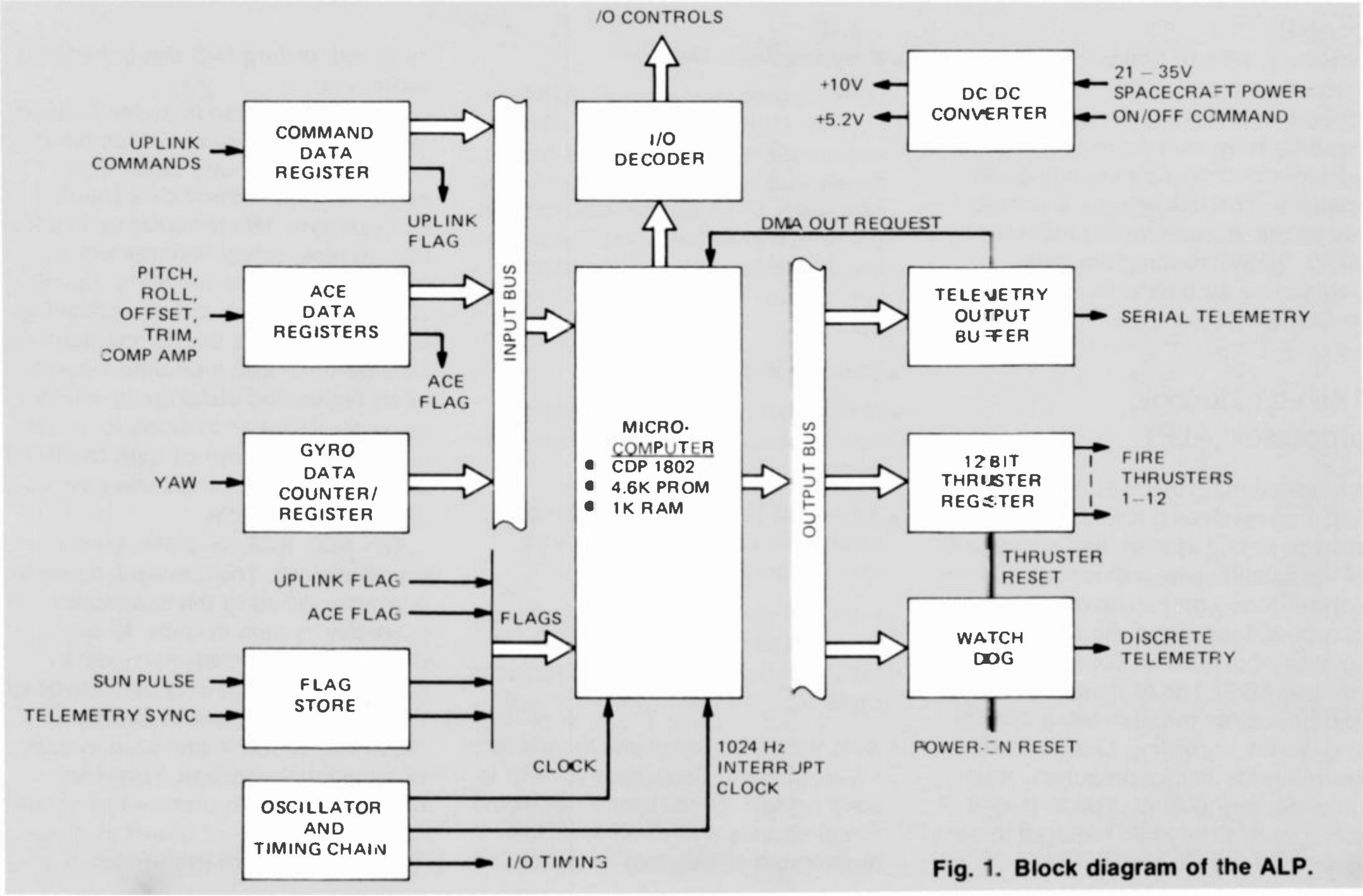


Fig. 1. Block diagram of the ALP.

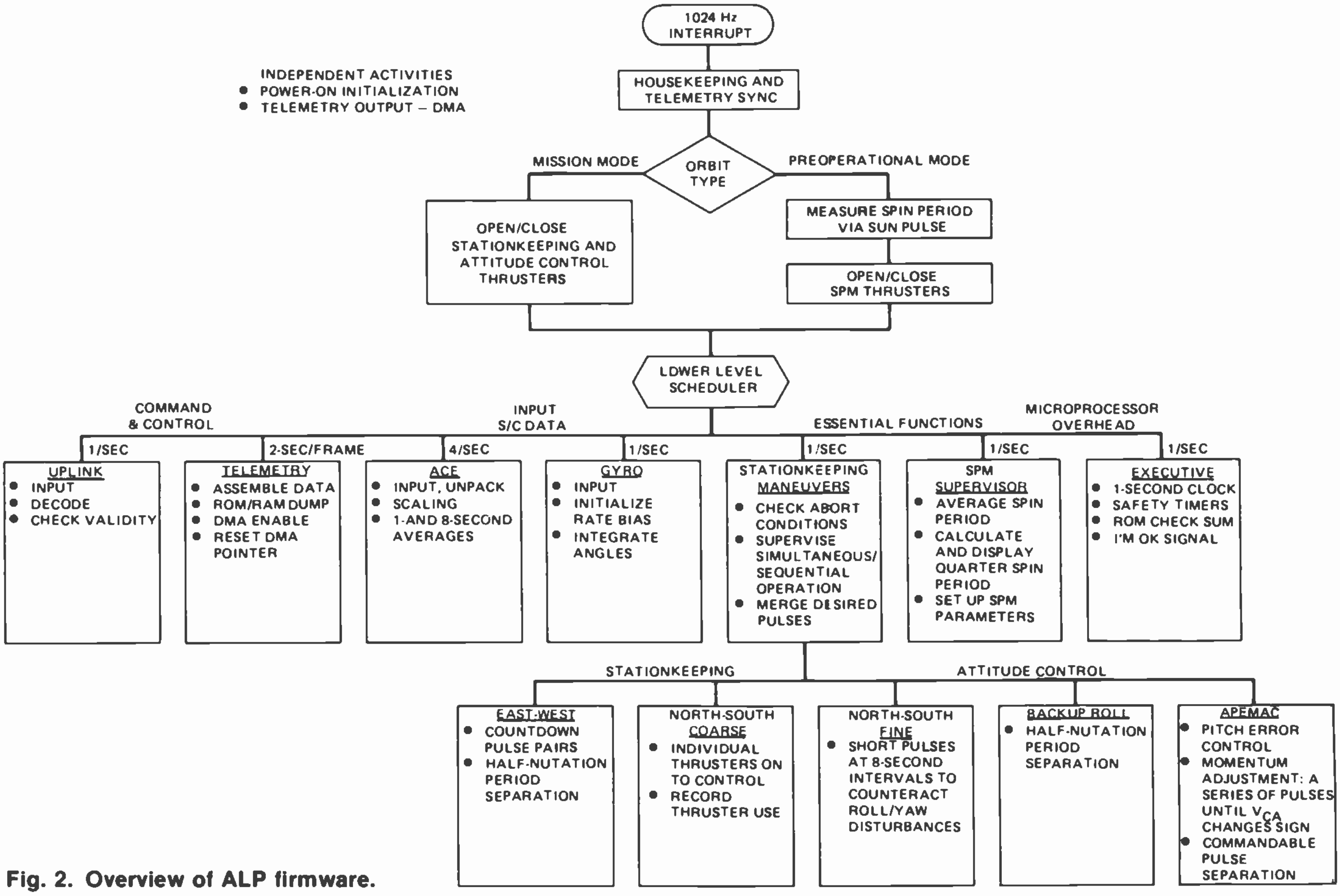


Fig. 2. Overview of ALP firmware.

increased ease of operations. Microcomputer control offers flexibility in accommodating logic changes resulting from varying mission requirements simply by changing the firmware. The following is a description of the attitude logic processor (ALP), the microcomputer that provides the automatic thruster control on Satcom D.

The attitude logic processor (ALP)

A block diagram of the ALP is shown in Fig. 1. It receives pitch, roll, and yaw attitude data, together with a measure of the satellite bias momentum (pitch-control-loop compensation-amplifier output voltage) from the satellite sensors and the attitude control electronics (ACE). The ALP then computes and generates thruster-firing control sequences according to algorithms stored in the microcomputer's read-only memory (ROM). The firmware incorporates the logic required to perform the following tasks:

▪ **Preoperational Mode**

- Spin precession maneuver (SPM) execution. During the transfer orbit, the spacecraft is spinning about the thrust axis at a nominal 60 rpm. It is necessary to place the spacecraft in the proper orientation with respect to the orbit prior to firing the apogee kick motor. This maneuver is called the SPM.

▪ **Mission Mode**

- North-south (N-S) stationkeeping (latitude orbit correction) and attitude control.
- East-west (E-W) stationkeeping (longitude orbit correction) and attitude control.
- Back roll control. Propulsive roll control is used as a backup in case of failure of both roll magnetic torquing coils.
- Automatic pitch error and momentum adjust control. Propulsive control is used to stabilize the spacecraft about the pitch axis and to off-load the momentum wheel, that is, to reduce its speed, during N-S stationkeeping maneuvers.

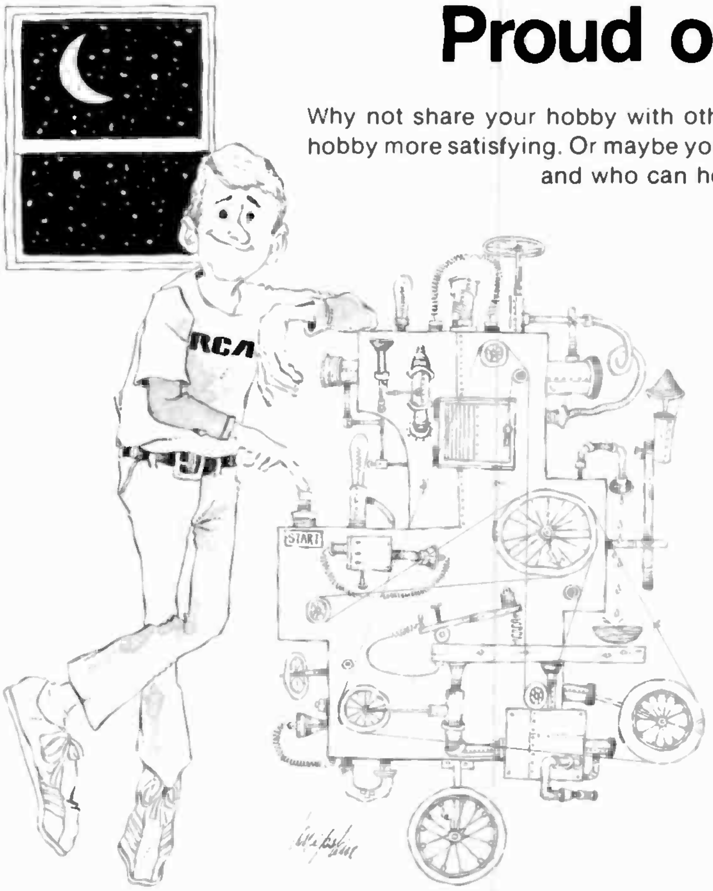
The firmware also includes code for required interfaces: uplink-command processing, telemetry collection, earth- and sun-sensor data inputs, and yaw gyro data processing. In addition, several safety features are an integral part of the firmware: uplink command and data validity checking, maneuver backup timers, maneuver-attitude-error and momentum-level-abort limits, and continuous monitoring of the ROM. Provisions for a complete memory dump of both the ROM and the random-access memory (RAM) are available.

The ALP, ACE, and yaw gyro are asynchronous. The firmware, however, is synchronized to the spacecraft telemetry system in order to synchronize the ALP telemetry data readout with the timing operations of the telemetry system. A backup "flywheel" mode is provided in case of telemetry sync loss. Firmware routines requiring precise timing are driven by a 1024-Hz interrupt clock. Other routines are run in the background, according to a 2-second schedule that activates them often enough to detect changes in their input. An overview of this operation is given in Fig. 2. The RAM is initialized at power turn-on, and the actual telemetry output is by way of direct-memory access (DMA). The ALP also incorporates a hardware safety feature. A "watch-dog" circuit is provided to periodically monitor the status of the microcomputer. A firmware routine, executed once every two seconds, provides a reset pulse to the watch-dog timer. Should the microcomputer fail to execute two or more timer reset pulses, the watch-dog circuit will automatically disable all thruster-firing outputs.

The ALP microcomputer architecture is based on the RCA CDP1802 microprocessor, with 4.6 kbytes of bipolar ROM for storing the control programs and 1 kbyte of SOS (silicon-on-sapphire) RAM for data storage. Programmable ROMs are used to accommodate program changes as mission requirements vary. The PROMs are power switched in the ALP to conserve power. The use of CMOS integrated circuits also contributes to low-power operation.

Stephen Malyszka joined RCA as a design engineer in 1967 and currently is the Electrical Design Manager in the Electronic Systems Department at Astro-Electronics, Princeton, New Jersey. His group is responsible for the design and test of electronic subsystems for satellite applications.

Contact him at:
Astro-Electronics
Government Systems Division
Princeton, N.J.
TACNET: 229-2825



Proud of your hobby?

Why not share your hobby with others? Perhaps their interest will make your hobby more satisfying. Or maybe you'll find others who already share your hobby and who can help make your own efforts more rewarding.

The *RCA Engineer* likes to give credit to engineers who use their technical knowledge away from the job. We've published articles about subjects as diverse as a satellite weather station, model aircraft and railroading, solar heating, and an electronic fish finder.

For more information on how you can participate in this feature of the *RCA Engineer*, call your local EdRep (listed on the inside back cover of the *Engineer*) or contact Frank Strobl (222-4220).

G.D. Chin | L.C. Kaye

REMBASS THE Repeater

The 1802 microprocessor, applied to a VHF repeater for message handling and data verification, can provide real-time operation to relay message-data packets reliably.

Abstract: *The 1802 microprocessor is applied to a VHF repeater to perform message handling and data verification. The microprocessor examines and analyzes all incoming data bits; selects and stores the valid message data; and enables the transmitter to rebroadcast any stacked messages. One novel feature is that the system can process high-density signals in real-time. The repeater is small, uses low power, and can be adapted easily to accommodate different message structures.*

For an unmanned radio repeater to reliably discriminate and relay messages or data packets that occur randomly and have several possible formats, a high level of built-in intelligence is essential. As used in the Army's REMBASS (Remote Battlefield Sensor System) communication link operating in a warfare environment, the repeater normally maintains radio silence to evade electronic detection by the enemy, and only when a properly signatured message is received does it respond immediately with a short message-retransmission burst. To be very selective in accepting messages and achieve a high rate of correct message retransmissions, the required processing involves not only accurately recognizing a message and orderly managing its content, but also verifying the integrity of the received data. The repeater must also be able to handle peak message traffic, while maintaining a low probability of missed messages. Thus, whenever the received-message rate is greater than the retransmission rate, the excess or backed-up messages need to be temporarily stacked in a data buffer and released on a first-in, first-out basis. These operational requirements, coupled with the control functions for system initialization, message transmission and various scheduled events, constitute a controller/processor work load that is well suited for a microprocessor.

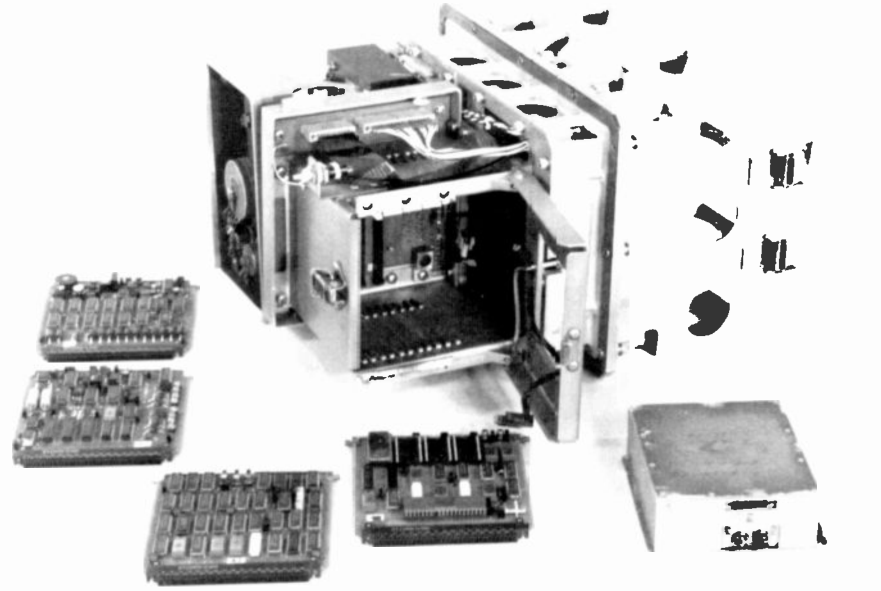
As implemented in the REMBASS repeater, the microprocessor provides most, but not all, of the control and processing functions—a compromise situation created by timing constraints due partly to the data rate in a message and partly to the operating limits of the selected microprocessor itself. However, the advantages of the microprocessor-based design—even with some auxiliary hardwired functions—still outweighed that of an all-random-logic approach. Besides providing a lower-cost system with less hardware complexity and attendant power consumption, the microprocessor approach resulted in a more flexible system that can be easily modified via software to accommodate future operational requirements. Also, by software control, the power drain of high-load circuits, particularly in the transmitter, can be duty-cycle managed, producing a more efficient system and prolonging battery life. Another attractive benefit was the realization of a data buffer with adequate capacity in the RAM and its access by software, making for a very inexpensive message-storage medium.

REMBASS overview

Developed for the U.S. Army by RCA,* REMBASS is a family of equipment that is used to gather activity information in forward battle areas or in rear protection zones. This ruggedized surveillance system employs unattended electronic sensors that are hand-emplaced, air dropped or ballistically delivered in potential avenues of enemy approach. A variety of available sensors respond to seismic disturbances, acoustic energy, magnetic field changes and object heat produced by enemy personnel and vehicle movements. Detection of activity is reported by a sensor in a

*Under Contract DAAB07-77-C-3298, sponsored by ERADCOM, U.S. Army, Fort Monmouth, New Jersey.

Reprint RE-26-8-11
Final manuscript received June 25, 1981.



The main electronics chassis contains a six-bay battery compartment at one end, a diplexer at the midsection, and all remaining electronics assemblies at the other end. The card cage holds four printed circuit boards and the transmitter assembly, which is constructed as a plug-in unit. The receiver and control panel are located behind the card cage.

short digital message to a remote data-monitoring unit via an FM-radio communication link. The remote monitoring unit may receive activity messages directly from the field-deployed sensors or through repeaters, as illustrated in the operational deployment configuration (Fig. 1). The sensor message arriving at the monitor is demodulated, decoded, displayed and permanently recorded — the record, with timing reference, being provided for post-evaluation of the activity in the monitored areas.

System equipment

The repeater-equipment complement for ground deployment is comprised of an omnidirectional monopole antenna, a tripod-based mast and an electronics unit (Fig. 2). In a field installation, the antenna is press-fitted on top of the mast and the telescopic mast is extended and locked to the desired antenna height. Connecting the free end of the antenna cable to the electronics unit, completes the system. For operation in a helicopter, the mast and antenna are not used. Instead, the electronics unit, mounted on a vibration-isolation tray, connects to an existing VHF antenna on the aircraft using a supplied extender cable.

Electrical description

The REMBASS repeater operates as a simplex communication unit, providing 320 digitally selectable channels in the VHF band from 138 MHz to 153 MHz. In any given operational set-up, a pair of separate channels, one for receiving and one for transmitting, are used to permit simultaneous reception and rebroadcast of real-time (analog) as well as time-delayed (digital) messages.

A block diagram of the major functional modules in the repeater is shown in Fig. 3. The antenna is shared by the receiver and transmitter through the diplexer, which contains two band-pass filters that separate the total operating frequency band into a low-band and a high-band region, each 4-MHz wide. With the receiver tuned to operate in one of the bands and the transmitter working in the other band, the diplexer provides the necessary signal isolation to allow both the receiver and transmitter to function at the same time without mutual interference.

The low-band and high-band ports of the diplexer connect to the receiver-RF input and the transmitter-RF output by way of a coaxial relay-switching network (not shown in Fig. 3). Conditioned during system initialization, these relays select the required RF paths according to the transmitter's assigned frequency channel. The receiver is automatically connected to the diplexer port opposite that for the transmitter.

Both the receiver and transmitter are modular assemblies that have common use in the REMBASS family of equipment. Consuming less than 250 milliwatts, the receiver accepts FM signals in one of 600 programmable channels spaced 25 kHz apart over the operating frequency band. It has an input sensitivity of -114 dBm, exclusive of the diplexer and line losses. For the digitally modulated RF signal, the nominal peak-to-peak frequency deviation is 6 kHz.

The receiver is channel-designated by an 11-bit select word provided from one of two sources — a memory on the Power and R/T Control module that has been preloaded by an external code-programming unit or a data register on the Analog Conditioner that is software controlled. The output lines

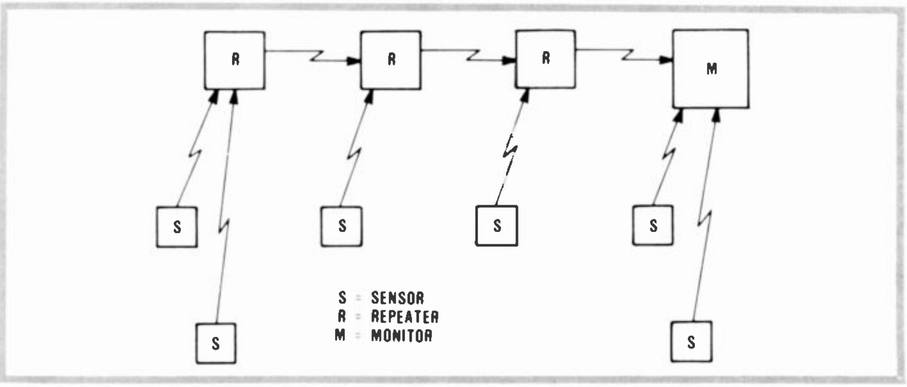


Fig. 1. Depending on the terrain features and distance to the monitoring unit, one or more repeaters are used in an operational deployment to extend the broadcast reach of sensor messages. A repeater in a relay chain and the monitor can service locally-emplaced sensors within a 15-kilometer radius.

from the receiver consist of an analog signal that is further amplified in the Analog Conditioner and three digital signals that are translated and formatted in the RCVR-XMTR Interface module for use by the microprocessor.

Like the receiver, the transmitter is also programmable to select 600 frequency channels over the same band. Its carrier frequency is generated by a voltage-controlled oscillator (VCO) operating in a digitally controlled phase-lock loop that is referenced to a highly stable local frequency source. To produce the frequency-modulated carrier signal, the message information is modified in a shaping modulation filter and inserted at the summing mode of the phase-lock loop. The resulting modulated carrier from the VCO is boosted by a power amplifier to provide a minimum RF output of 2 watts. The synthesized carrier is high in purity with residual sidebands down -50 dBc and spurious signals down -65 dBc.

The four modules that communicate with the 8-bit data bus represent the circuit partitioning of the remaining electronics for the repeater. On these modules, low-power analog circuits and CMOS logic operating at 5 volts are used throughout to conserve battery power.

General system control and data processing is managed by software resident in a 2-kbyte ROM device on the Microprocessor-Memory module. In addition to an 1802 CMOS microprocessor element and the 2-kbyte ROM, this module contains a 2-MHz crystal, 1 kbyte of RAM, an address decoder for the first 16 kbytes of memory address space, bus separators for the on-board memory, associated memory-timing circuits and pre-wired sockets for three more ROM devices for future software growth. For the software development phase, ultraviolet PROM-type 2716, which is pin compatible with the 2-kbyte CMOS ROM (Hughes type-1836), was used.

On the Receiver/Transmitter Interface module, the three digital output signals from the receiver are preprocessed to a more acceptable form for microprocessor analysis. The data clock is recovered from the incoming biphase data to set the interrupt-

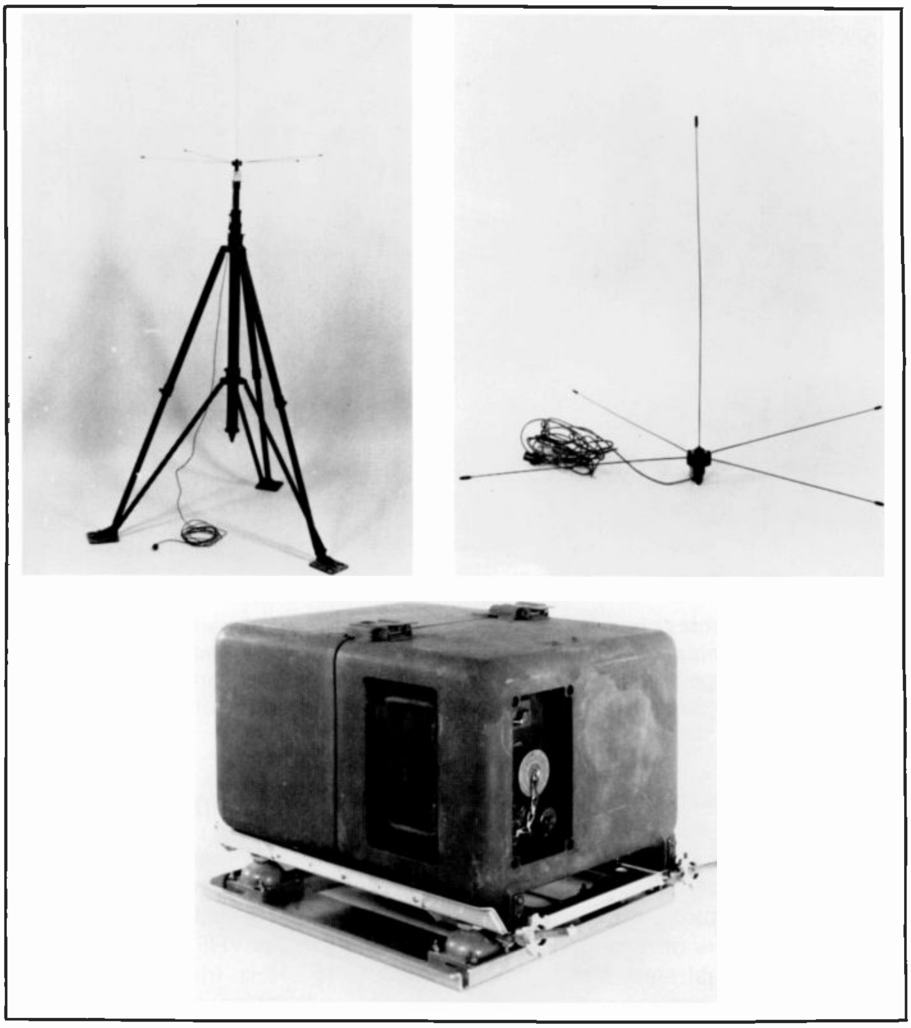


Fig. 2. The REMBASS repeater consists of an omnidirectional radial antenna with a pre-attached RF coaxial cable (top right), a tripod-based mast used to raise the antenna up to a maximum height of 14 feet (top left), and an electronics unit containing batteries to sustain a 30-day mission (bottom).

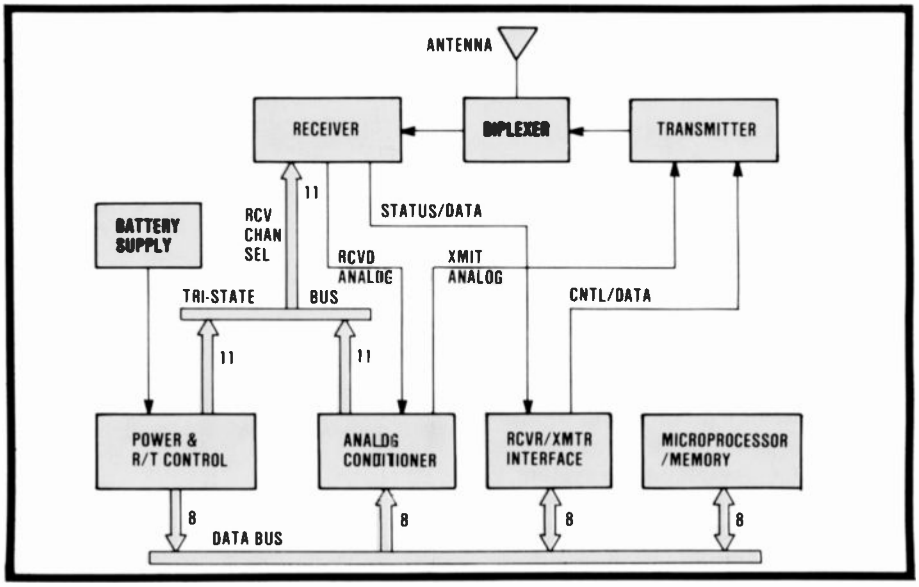


Fig. 3. Block Diagram—the major functional modules consist of the receiver, diplexer and transmitter subassemblies, and four printed circuit boards which communicate among themselves over the data bus.

request flag, identifying the mid-bit clock transition. The extracted clock is monitored to detect "out-of-tolerance" clock periods and the two received signal-status lines are combined by logic to sense a signal fade during a message. Fault signals for a detected erroneous clock period and a message fade are provided to the microprocessor for use in data-bit acceptance decisions.

Once initiated, a message transmission is completely controlled by dedicated logic on the Receiver/Transmitter Interface module. The transmission-control logic consists of a data-clock generator, an 8-bit parallel-to-serial converter, a biphase encoder, a 16-bit message length counter and an 8-bit system control register.

The Analog Conditioner module contains a programmable gain amplifier to adjust the analog path gain, a 2-byte register to provide a channel-control word to redesignate the receiver and four coaxial switch drivers during system initialization. To complete the gain-calibration circuitry, a 600-Hz sinewave reference generator, two peak-amplitude detectors and a voltage comparator are included on this module. During system initialization, the analog path gain is set in an end-to-end RF self-test loop, with a 600-Hz modulation tone, normalizing the transmit-to-receive modulation ratio for a factor of one.

A 5-volt switching regulator and the battery-protection diodes are located on the Power R/T Control module. Also included are the tamper-monitoring circuitry, the mission-life timers, the battery-voltage monitor, and a 32-bit data register that stores the operating receive-transmit channel words.

The battery complement for 30 days of digital-messages-only operation is three 12-volt, 20-AH, lithium organic batteries; for 30 days of combined digital and analog operation, six batteries.

Message characteristics

The repeater will respond only to the REMBASS-defined messages and other like-formatted messages that use the same data-structuring rules. There are two types of REMBASS messages — a 29-bit basic digital type and a hybrid or "analog" type consisting of a 29-bit basic message as the sensor type identifier followed by 15 seconds of analog (audio) signal. A totally digital message is generally processed on a store and forward basis; that is, the entire message is received, stored and verified before it is transmitted. In the case of a hybrid message, the digital portion is similarly processed, but after sending the 29th bit, the analog portion is switched in for real-time retransmission. The two message formats used in REMBASS are shown in Fig. 4.

There are four principal sections in a 29-bit basic message. The preamble/sync section consists of a fixed pattern of eight "0s" followed by a "1" bit, the sync bit. The next four bits provide the message type code, which usually identifies the user system or subsystem. The remaining two sections are the data fields, each with its own parity bit. For REMBASS, field one contains 7 bits and field two has 9 bits. As can be seen, an analog message has the analog signal appended to the 29-bit basic message that serves as the header. At a nominal data rate of 1200 bits per second, the "on-air" time for a 29-bit message is 24.2 milliseconds.

Two other message formats can be serviced by the repeater — a long message format where a long data field is attached to a basic message and a modified 29-bit message with a single data field of 16 bits.

Message processing

On a continuous basis, the receiver monitors the designated listening channel for the occurrence of a sensor message, while the processor examines the data output from the receiver for a particular starting bit pattern or preamble. Before the received data is presented to the microprocessor for analysis, it is preprocessed to extract the data clock contained in the Manchester biphase-encoded signal. The mid-bit transition of the data clock is used to generate an interrupt request to prompt the microprocessor to accept the data bit. The period of the data clock is also prescreened to determine whether the clock rate is within the accepted tolerance window. When a clock period is too long, an "out-of-window" signal is provided to the microprocessor to reject the current message.

As the signal strength of a message exceeds the detection threshold, the receiver sends a "Message-Ready" signal to indicate that the incoming data is conditionally valid. Since the indication signal of a detection will tend to lag the leading message bits, the incoming data bits will initially be examined regardless of RF-signal level — the initial data-acceptance criteria being that the data values are zeros and the data-bit periods are within the tolerance window.

After a starting pattern of a minimum

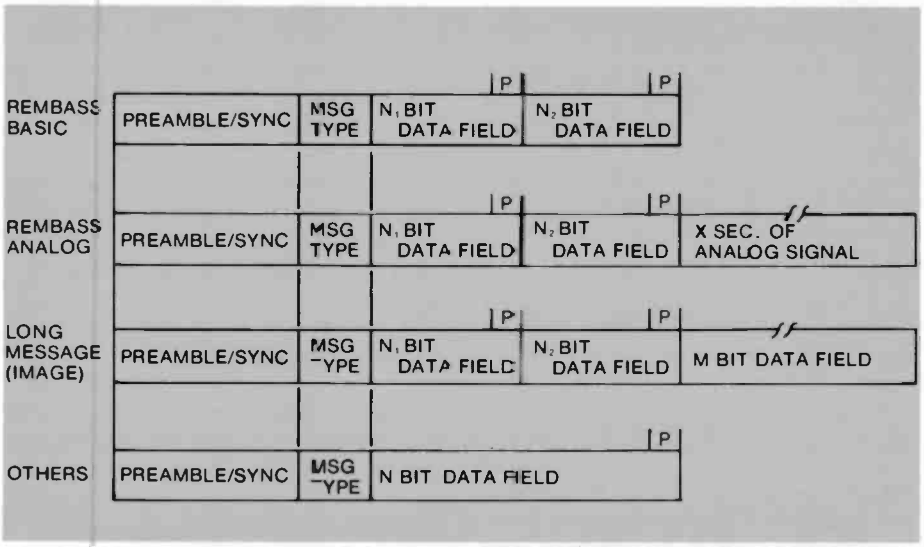


Fig. 4. The message formats to be handled by the repeater fall into four categories — the basic message, the analog message, the long message, and the modified basic message (others). In processing data, a characteristic table is used to select and define the bit and field handling routines for a particular message (MSG) type.

of five zeros has been collected and "Message Ready" is true, the transmitter — except for its final output stage — is commanded to power-up to allow its frequency synthesizer to stabilize in anticipation of a possible message retransmission. The processor proceeds to collect the next expected message portion of a sync bit (one), followed by the four bits defining the message type. If the message type code is pre-defined in the message-characteristic table, the processor will store the preamble of zeros (8), the sync bit, and the message type code in a revolving data buffer in RAM. The remaining incoming data bits are collected, formatted in byte form, and stored in successive byte locations. As the parity bit appears in the data stream, it is checked for correctness. If the parity bit indicates an error, the message-handling activity is terminated and the processing returns to seek a new message. There are two parity checks for a basic 29-bit REMBASS message.

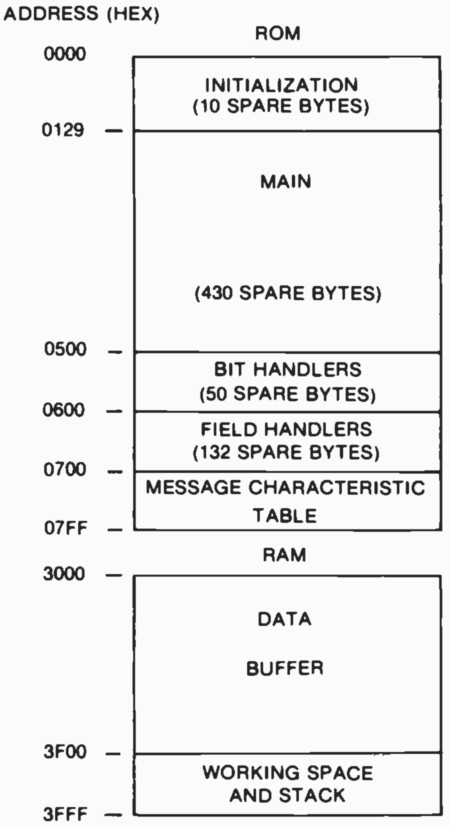


Fig. 5. The 2-kilobyte ROM contains the firmware for system initialization, main system control, bit handlers, field handlers, and message-characteristics table. Up to 109 basic REMBASS messages can be stored before the data buffer pointers return to the starting address.

After a complete message has been stored, the processor will initiate a message-transmission cycle. A message-length word is transferred to a message-bit counter in the RCVR/XMTR interface module and a transmit command is issued to energize the output stage of the transmitter. By direct memory access (DMA) into the data buffer, the message data is obtained, a byte at a time, controlled by dedicated output logic. The parallel data is converted to a serial, Manchester biphase format for driving the transmitter modulator. When the last message bit has been sent, the message-bit counter stops the transmission event and the transmitter's output stage is de-energized. After 5 seconds of delay following the last transmitted bit, the transmitter's frequency synthesizer is turned off. For a programmed mission life of 7.5 or 15 days, this frequency synthesizer is kept powered.

The handling routine for a hybrid message, one consisting of a 29-bit header and 15 seconds of analog signal, is automatically called when the analog message type code is recognized. The entire message header will be treated like a basic 29-bit message in the "store-and-forward" manner. At the end of transmitting the 29th bit, the modulator input is switched from the digital message source to the amplified analog signal provided by the Analog Conditioner. The processor will maintain the analog transmission for a minimum of 14 seconds and allow the end of the analog signal to terminate the transmission in the next final second. If the duration of the analog signal is longer than 15 seconds, the transmission is automatically truncated at 15 seconds.

Memory map

The on-board address decoder selects operational memory in 2-kbyte blocks for the first 16 kbytes of memory address. Shown in Fig. 5, the memory map identifies the various program segments resident in the current 2-kbyte ROM (address 0000 to 07FF) and the location of the data buffer, working space and stack in the 1-kbyte RAM (address 3C00 to 3FFF). To cover future design modifications and performance growth, 6 kbytes of additional ROM space has been prewired.

Located in the first 768 bytes of RAM is the revolving data buffer, which is managed by three register pointers — two for the input activity and one for the output activity. Because the input pointer must be recoverable to restart a new message storage in case of an input message abort, a first-byte-input-reference pointer is needed in addition to a normal incrementing input pointer. When an input message fails a verification test, any prior stored data is rejected by overwriting it with the next message, relying on the first-byte-input pointer to restore the incrementing pointer to the starting data bin.

All messages are stored with a 2-byte prefix that defines the message-bit length. For the 29-bit REMBASS message, seven bytes of buffer space is required — two for message length, four for actual data, and one for output-timing adjustment.

The data-buffer pointers are continuously monitored by software to sense the approach of the upper-buffer address or the end of the buffer. When the input or output pointer indicates that less than seven bytes are left, the pointer is automatically returned to the beginning of the buffer. The "end-of-the-buffer" test and pointer readjustment is a regularly scheduled main program event.

The 2-kbyte ROM is partitioned into five principal subprogram sections consisting of the initialization program, the executive program, the bit handlers, the field handlers, and the message characteristic table. Certain subprograms are started at 256-byte page boundaries to take advantage of the short branch instructions. Spare bytes are purposely interspersed to facilitate design changes or expansion without disrupting the overall subprogram-addressing scheme.

Design approach

The primary design goal was to accept input data at its peak rate, process each data bit, assemble the message and retransmit it utilizing the resources of the 1802 microprocessor to reduce the hardware requirements.

The main task was to monitor the receiver output for the starting preamble of a message and, noting its arrival, collect and analyze the remaining message data. At an input data rate of 1200 baud, a new bit occurs every 833 μs, during which about 104 instructions can be executed if a 2-MHz microprocessor clock is used. Because of this low number of available instructions and a worst-case input of contiguous incoming messages, it was decided that the output-control functions be implemented with hardwired logic to minimize the required output-control software. Data for the output message would be extracted from the data buffer by DMA. The required output software merely presets the DMA pointer and hardwired output-timing logic to initiate the transmission event. The hardwired logic will convert the parallel data byte from the buffer to serial Manchester encoded data for driving the modulator in the transmitter. A new DMA request for a data byte will occur after every eight shifted bits, continuing until the message length counter that monitors the number of transmitted bits terminates the output activity.

The software was also assigned the tasks of managing the system-control flags between the input-bit interrupts, supervising the data buffer and monitoring for RF interference by other communication or countermeasure systems.

Projected performance growth involves the handling of new message types and different data formats that may be employed by other potential repeater users. Based on the current tri-service interoperability data-link concept, the first 13 bits of the message header is common for all message types with the remainder of the message to be defined by the service user. The most practical approach to easily accommodate the new message types was the use of a message-characteristic table that would contain a specific processing menu or schedule for each message type. When a new message type is to be handled by the repeater, the required processing menu consisting of subroutine addresses and appropriate constants is simply added to the ROM containing the table. A 256-byte table was selected to cover all 16 message types (4-bit code) allotting a 16-byte menu per message. Currently, the table is programmed to recognize five message types for rebroadcasting.

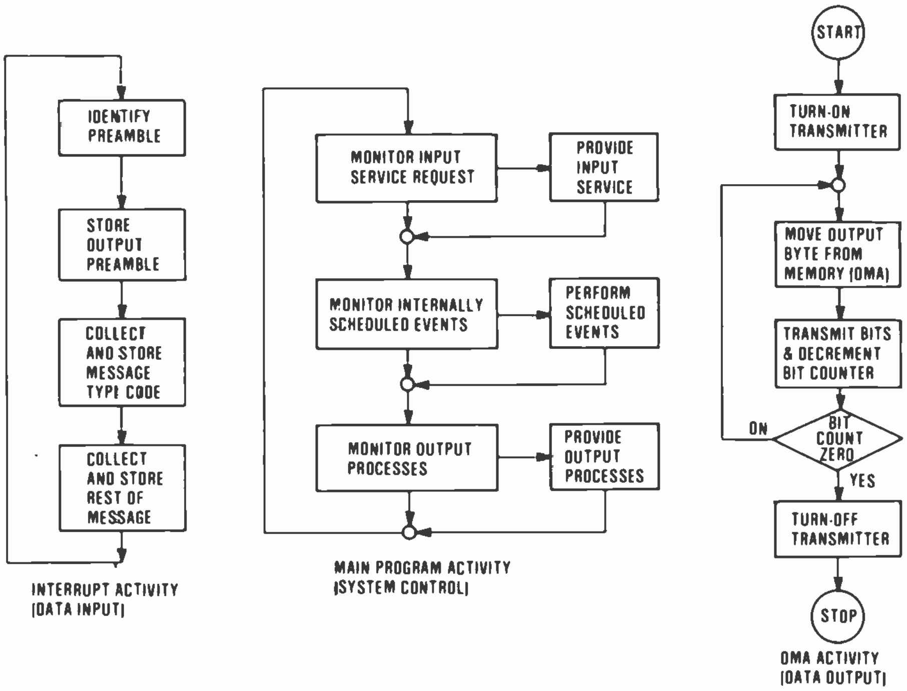

IDENTIFY PREAMBLE
STORE OUTPUT PREAMBLE
COLLECT AND STORE MESSAGE TYPE CODE
COLLECT AND STORE REST OF MESSAGE
INTERRUPT ACTIVITY (DATA INPUT)
MONITOR INPUT SERVICE REQUEST
PROVIDE INPUT SERVICE
MONITOR INTERNALLY SCHEDULED EVENTS
PERFORM SCHEDULED EVENTS
MONITOR OUTPUT PROCESSES
PROVIDE OUTPUT PROCESSES
MAIN PROGRAM ACTIVITY (SYSTEM CONTROL)
START
TURN-ON TRANSMITTER
MOVE OUTPUT BYTE FROM MEMORY (DMA)
TRANSMIT BITS & DECREMENT BIT COUNTER
ON
BIT COUNT ZERO
YES
TURN-OFF TRANSMITTER
STOP
DMA ACTIVITY (DATA OUTPUT)

Fig. 6. The primary software activities involve the system control events, the acceptance and handling of the input data, and the transmission control of the output data.

Software design

The software is concerned with three major activities, the interrupt event for handling the input data, the main program for controlling the system functions and the DMA activity for providing output data. Depicted in Fig. 6, these activities are implemented in a multiprogramming approach to obtain apparent concurrent operations of both systems control and data processing.

The main or executive program performs operational systems control and schedule housekeeping functions on a

Table I. Software flag definitions. Software flags 1 through 14 and 18 are used to request a particular service or to indicate an event status.

Flag No.	*Function*
1	7.5/15-day mission
2	Synthesizer enable request
3	Synthesizer on and stable
4	Synthesizer warm-up delay (33 ms)
5	Abort store and forward message
6	Synthesizer turn-off delay (5 sec)
7	S & F message transmit request
8	Long message transmit request
9	End of long message
10	Long message abort request
11	Transmitter busy
12	Synthesizer on and stable (also flag 3)
13	Analog message transmit request
14	Interference test request
18	Transmitter on during last received bit

flag-request basis. These flags are software generated by the input/output activities in handling a sensor message and by the internally scheduled events such as transmitter-frequency synthesizer control, maintenance of buffer pointers and RF-interference monitoring. Routinely scanned in a loop are operational flags, listed in Table I, to indicate a specific control state or a service request. As each raised flag is encountered, a service or function is performed, following which the program returns to the flag-testing loop. The main program is interrupted any time a data-clock transition has occurred whether it is due to real-data or open-receiver noise.

The role of the interrupt activity is to collect, analyze and store a packet of data in a revolving file or buffer, arranging the data in groups of eight bits and storing the groups in the proper sequence for later data retrieval by DMA. The header of all messages inclusive of the message type code is processed in an identical manner.

The output activity is automatically instituted whenever there are untransmitted messages remaining in the data buffer. A transmission cycle is initiated by the main program with a very minimal subprogram and essentially off-loads the output task to the hardwired output logic, which then manages a message transmission to its conclusion including the transfer of data from the RAM buffer by DMA.

The required software functions included system initialization and some coarse software timers.

Summary

A microprocessor-based repeater can provide real-time operation to relay message-data packets in a reliable manner. This system implementation has greater flexibility for incorporating new performance requirements or modifying current operations or functions. Particularly for handling new message types that meet the general message structure being processed, the repeater can be updated simply by reprogramming one ROM device. Because of timing limitations in this application, however, inevitable trade-offs were made to remove some counting-intensive operations from the software load. The sizeable data buffer — employing RAM, of course — was obtained at very little expense.

Acknowledgments

The successful design of the REMBASS repeater is the result of a team effort of several people who contributed their time and experience in their own field of expertise. The authors would like to acknowledge the valuable contribution of R. Elder (REMBASS Engineering Manager), E. Fisher (Receiver), R. Bosselaers (Transmitter), M. Kurina (Mechanical Design Manager), and J. McNamee (Mechanical Design) for their participation on the project.

Left to Right: Leo Kaye, Engineering Scientist, Gep Chin, Engineering Scientist, Automated Systems, Burlington, Massachusetts.

Gep Chin is a member of the Radiation Systems Engineering section at RCA Automated Systems, Burlington, Massachusetts. From 1955 to 1964 he worked at RCA Missile and Surface Radar, Moorestown, New Jersey, where he was a digital and analog circuits designer on the land-based Talos project, the AN/FPS-16 and AN/FPQ-6 radar programs and BMEWS. Transferred to RCA Burlington in 1965, he has been involved in the Lunar Module Rendezvous Radar program, the AN/ALQ-127 Tail Warning Radar, the Electronic Solid State Wide Angle Camera System (ESSWACS), REMBASS and various IR&D projects. Since 1975, he has had hardware design responsibility on three microprocessor-based systems.

Contact him at:
Automated Systems
Burlington, Mass.
TACNET: 326-2044

Leo Kaye, a member of the software engineering section, has been with the Burlington facility since joining RCA in 1962. He has contributed to the design of computers, controllers and special digital devices. Microcoding he did for the computers and TIPI controllers provided a basis for programming computer and microcomputer applications for REMBASS, CAC and other projects.

Contact him at:
Automated Systems
Burlington, Mass.
TACNET: 326-2338

F.E. Papke

Multi-image programming: Using microprocessors to "move" people

Computerized programming has brought a new level of professionalism to slide presentations. Government Systems Division is using the programmed presentation of visual images to communicate more effectively to today's sophisticated, demanding audiences.

Photo courtesy Lyons Studios Incorporated

In addition to its economic and industrial impact, the computer is directly affecting all phases of our society. Computer technology is being harnessed to induce massive changes in the cultural and artistic realms. Nowhere is this more evident than in the sweeping changes taking place in audiovisual (A/V) presentations, in particular those using banks of projectors on a number of screens.

Such multi-image presentations, as they are popularly called, are not new; they were used at world's fairs and industrial expositions before the turn of the century and are currently well-established in museums, planetariums, and exhibits scattered throughout the country. But for the most part, while computer technology was evolving in the '50s and '60s, presentations produced for industry, education, and government were limited to either large-scale temporary extravaganzas, simple filmstrips or one-projector slide shows. These were time-consuming to produce and suffered from equipment unreliability.

In the early '70s, automated multi-image equipment emerged based on the new digital computer technology. By the mid-70s, a major breakthrough occurred in the form of electronic programming equipment that freed the show producer from tedious housekeeping functions and allowed him to be more innovative — enabling him to create sophisticated effects on the screen in a fraction of the time normally required for production.

These early programmers used hard-wired microprocessors which the manufacturers saw as a means of reducing their product costs to meet the low-volume A/V market. But with the subsequent growth of the A/V industry in both size and sophistication, the manufacturers have turned to programmable, non-dedicated microprocessors. These units enable meeting the performance demands of multi-image through software updating, and at the same time add a capability to meet business applications such as word processing, accounting, inventory and other database management functions.

Reprint RE-26-8-12
Final manuscript received June 22, 1981.

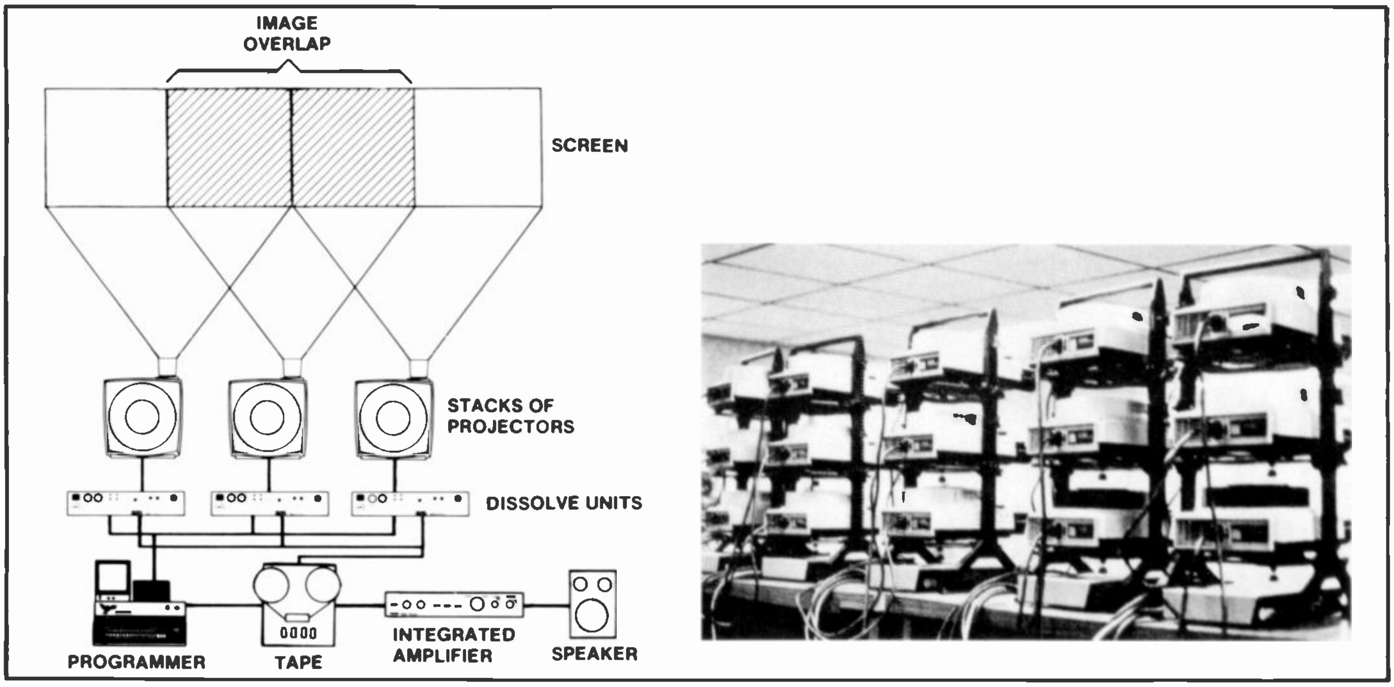


Fig. 1. Multi-image equipment in a typical show configuration. Dissolve units control the action of slide projectors in response to digital commands from a programmer or magnetic tape.

Programming *systems* are now available or in development that use either the rapidly proliferating business computers, such as the Apple II and the Bell & Howell Model 078401, or similar specially designed units such as the AVL Eagle or the Apple II-modified Clear Light Superstar. And some manufacturers are going one-step further in applying their systems to the generation of simple graphics for producing low-cost slides, and in adding microprocessor control to slide/animation cameras and compound tables.

This multi-image equipment development over the last decade has coincided with a real explosion in slide usage in the business world. In terms of dollars expended, slides are the leading visual communcations medium, nearly generating as much total annual spending as video and motion pictures combined.

Multi-image equipment operation

A typical multi-image projection setup is shown in Fig. 1. Dissolve units control the action of slide projectors in response to digital commands from a programmer memory or from a magnetic tape track. Each dissolve unit commonly controls a stack of two, three, or four slide projectors (depending on the manufacturer's design) with each stack illuminating a separate screen area. Screen areas can be butted to achieve a desired overall configuration; and screens can be overlapped to increase projector capacity and enable greater animation or special effects in the overlap area. Once the programming is completed, the commands can be recorded in sync with the audio, and show playback can be accomplished directly from the tape through the dissolve units.

What and why multi-image

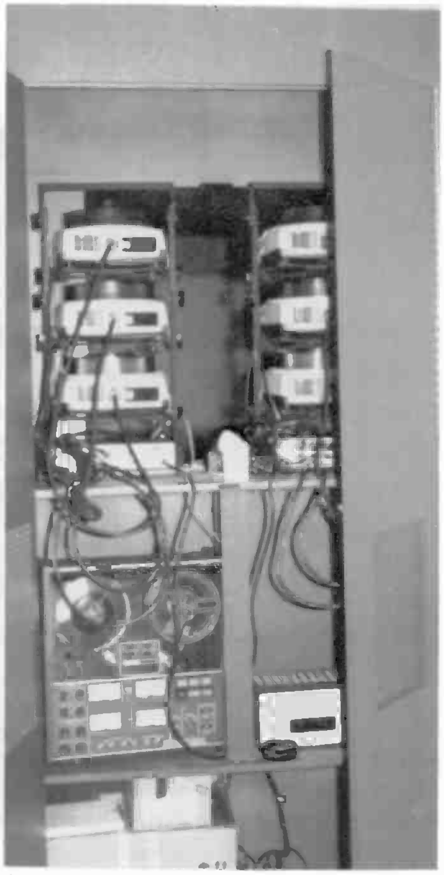
Multi-image deals with the programmed presentation of visual images to convey information and create an impression (see lead illustration). Multi-image is a dense communications medium. Programming is the key. It enables you to manipulate images; to dissolve from one image to the next at any of a number of rates; to animate; and to integrate slides, motion pictures, light displays, and other special effects into an automatically controlled, repeatable production.

Programmed presentations offer a flexibility in communication that is not available with most single-image media such as slides, films, or video. Multi-image presentations can project multiple-viewpoints of the same subject—and can do it simultaneously. This enables the viewers to see objects, events, or settings in perspectives impossible through normal human perception. Viewers can see the whole as well as the parts; the top, bottom, and the sides; the beginning, middle, and the end—and all at the same time.

And through this enhanced perception and organization, multi-image can compress the time needed to convey the information and to create the desired impression. It is a dense communications medium.

With multi-image, the visual format can be selected to suit the audience, and to enhance the message or impression you wish to convey. Screen format is almost unlimited. Multi-screen and wide panoramic presentations create an excitement for the viewer who is all-too-conditioned to watching a 21-inch image on his home TV.

And, perhaps best of all, multi-image production is economical. The use of slides provides a modularity not available in video and motion-picture productions. Manipulation of images and development of special effects is easily accommodated in programming, and changes at any time—even



GSD uses the Multi-Image Program Development Center to produce multi-image presentations for Industry trade shows. Here, a typical GSD booth (top), some of the recording equipment used (far right), and a multi-image presentation (right) are shown.

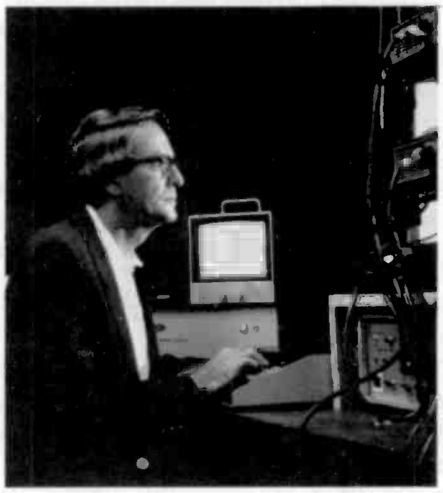
after show completion — can be rapidly and inexpensively accomplished.

GSD efforts in multi-image

Early in 1980, the Multi-Image Program Development Center was established — a pilot project to produce multi-image presentations for RCA Government Systems Division. Several six- and nine-projector shows have been produced for industry trade shows and employee-motivation presentations. Except for slide preparation, the Center is fully equipped for all phases of multi-image production.

Our programming equipment has almost unlimited capability. The AVL Eagle programmer, with its 32-kbyte memory and more than 130 programming controls, can drive up to 120 slide projectors. It programs in English and offers continual capabilities-updating through software modifications. Twelve slide projectors with dissolve units and stacks enable us to take a six-projector show on the road while simultaneously developing another six-projector show in-house. An assortment of lenses can accommodate a range of projection requirements, from a tight booth space to a large auditorium. And our audio equipment can fill that auditorium with sound. An audio mixer and four-track magnetic tape recorders enable us to prepare quality stereo sound tracks that combine live narration with production music and sound effects from our record library. Other equipment includes a recorder designed for continuous play of short shows, a two-projector dissolve unit for control of simpler presentations, and light tables that facilitate slide viewing and show assembly.

Shows are produced in a dedicated area in building 204-1 in Cherry Hill. It includes both a production room and a combination screening/conference room that can accommodate both front and rear projection.



Fred Papke is Manager, Proposal Media and Presentations Concepts in the Government Systems Division Staff Marketing organization. He has 23 years of experience in technical communications in a variety of media. In 1980, he established the Multi-Image Program Development Center and has since directed its operations.

Contact him at:
Government Systems Division
Cherry Hill, N.J.
TACNET: 222-5804

D.E. Britton|M.E. Stickel

SUPPOSE: A microcomputer operating system for distributed applications

SUPPOSE, an operating system for dedicated networks of microcomputers, provides a run-time environment for distributed applications as well as a conceptual framework for programming them. SUPPOSE is an acronym for Server Uniform Process Protocol Operating System Environment.

Abstract: *SUPPOSE is an operating system for dedicated networks of microcomputers that run distributed applications. A distributed application running under SUPPOSE consists of a number of cooperating concurrent processes, with one or more processes executing on each microcomputer in the network. SUPPOSE provides facilities for memory, process, and device management. In addition, SUPPOSE provides a conceptual framework for programming distributed applications. The services and resources provided and used by an application determine how the application can be decomposed into a set of cooperating processes. Processes may be servers, which provide a service or control a resource, or requestors, which use a service or resource. SUPPOSE supports this conceptual framework by providing a set of highly disciplined interprocess communication operations. As a result, SUPPOSE facilitates the design and implementation of distributed applications and makes them comprehensible.*

The SUPPOSE operating system provides a conceptual framework for structuring distributed applications and is, in addition, the operating system on which a distributed application executes.

Distributing processing among several computers is sometimes more advantageous than using a single computer. For instance, distributing the processing among several computers allows each computer a lighter processing load, and consequently lighter memory and speed requirements, than if a single computer were used. Thus, the processing requirements of an application may be able to be met more economically with a number of computers than with a single, more powerful computer—indeed, a single computer that is powerful enough may not even exist. Applications that use several peripheral devices each with substantial processing and real-time response requirements—examples include message switching, signal processing, and device monitoring—are good candidates for distributed processing (Fig. 1).

Another motivation for using distributed processing is that it is sometimes advantageous to place processing power close to where it is needed, for example, in each of several microprocessors attached to devices in a device-monitoring system. Distributed processing permits this.

Yet another reason for using distributed processing is for system reliability. When an application is appropriately decomposed and distributed over several computers, failure of one computer in the network need not result in failure of the entire application. The combined use of redundancy and geographic distribution of processing and data can avoid loss that would otherwise result from a single accident.

Unfortunately, effective program design techniques for

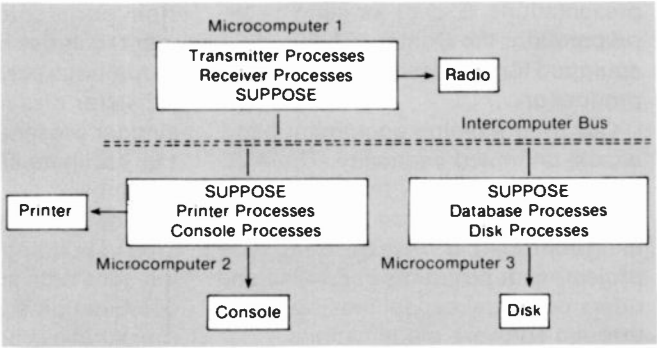


Fig. 1. SUPPOSE radio-message-switching application.

Reprint RE-26-8-13
Final manuscript received July 17, 1981.

distributed applications are not easy to discover. Often the designer of an application may have few guidelines for structuring the application so that it can be effectively distributed. SUPPOSE corrects this situation by providing both an operating system for distributed microprocessor applications and a conceptual framework for their design.

The SUPPOSE conceptual framework

In the SUPPOSE conceptual framework, an application consists of a set of cooperating concurrent processes. Each process can act as a requestor process, a server process, or both. The distinction between requestor and server processes provides a basis for decomposing an application into modular units that can be distributed over a network.

A server process provides a service, such as controlling a device or providing access to a database. A requestor process uses the services of one or more server processes. A server process may itself require the services of other processes, so that a process may be both a server and a requestor. Processes communicate entirely with messages that are passed from process to process. Thus, a requestor process requests a service by sending a message to the appropriate server and the server responds by sending a message back.

With SUPPOSE, an application is organized around shared resources. Each shared resource, such as a device or database, is controlled by a server process. In order to access a shared resource, requestor processes make requests on the server process controlling the resource. The server handles the service requests one at a time as they are received, assuring mutually exclusive access to the resource for all requestors.

After an application has been decomposed in this manner, it can be fitted to a network of microcomputers. SUPPOSE permits one or more processes to execute on each microcomputer in the network. The application designer assigns processes to microcomputers, taking into account such factors as the aggregate processing load on each microcomputer, the need for device driver processes to be located on the microcomputer to which the device is attached, and the reduced cost of communication between processes that are on the same microcomputer.

Processes communicate exclusively via messages regardless of whether or not they reside on the same microcomputer, the only difference being that it takes longer for a message to be sent between processes on different microcomputers. Consequently, where a process is to reside does not affect the way the process is programmed. Application decomposition and programming can be done before it is known where each process will reside or how many microcomputers will be in the network.

The SUPPOSE operating system

SUPPOSE provides those facilities that are needed by every microcomputer in a network dedicated to a distributed application: memory management, process management, device management, and interprocess communication (Fig. 2).

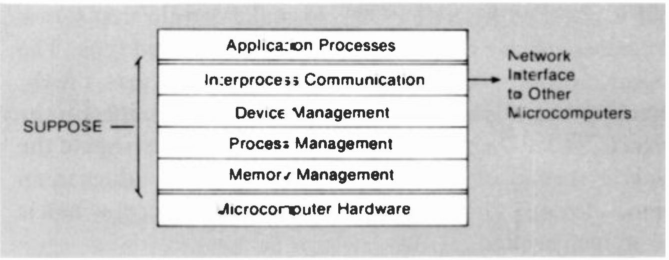


Fig. 2. SUPPOSE hierarchy of services.

SUPPOSE does not provide facilities such as file systems, databases, or support for particular hardware devices. These would all be programmed as SUPPOSE server processes in an application and would probably reside on only one node. Any process needing access to such a facility would make requests on the appropriate server process.

SUPPOSE is an operating system strictly for the execution, not the development, of distributed applications. A SUPPOSE program development environment could, however, be written as a SUPPOSE application.

Memory management

SUPPOSE memory management provides shareable process code, non-shareable process data, and dynamic objects (Table I).

Any process can create a dynamic object by using a SUPPOSE operation. A SUPPOSE object is always owned by and accessible to a single process, initially the object's creator. A SUPPOSE object can be used as a message to be sent from one process to another or can be pointed to as a subobject of a message. The message recipient becomes the new owner of the object.

The creator of an object specifies its type and format. The object type can be examined by application processes

Table I. SUPPOSE memory management operations.

new	create object
dispose	destroy object
copy	copy object
typeof	return type of object
nobjs, nints, nreals, nchars	return number of subobjects, integers, reals, or characters in object
getobj, getint, getreal, getchar	get subobject, integer, real, or character from object
putobj, putint, putreal, putchar	put subobject, integer, real, or character into object
mgetint, mgetreal, mgetchar	get multiple integers, reals, or characters from object
mputint, mputreal, mputchar	put multiple integers, reals, or characters into object

and is checked by SUPPOSE to make certain that server processes receive only messages of the expected type. The object format determines how many integers, reals, characters, and pointers to subobjects are contained in the object. SUPPOSE provides operations to interrogate the type or format of an object, to fetch or store values in an object, to copy an object, and to destroy an object which is no longer needed.

Process management

SUPPOSE processes are programmed as procedures in a sequential programming language such as Pascal. The SUPPOSE operating system schedules these procedures and runs them as independent processes. Processes communicate with each other through message operations provided by SUPPOSE.

Each process executes on a single microcomputer, and more than one process can execute on each microcomputer. Processes have execution priorities: on each microcomputer, the highest-priority "runnable" process runs until it blocks itself or until a higher-priority process becomes runnable. Processes block themselves by waiting for a message or an interrupt, and become runnable again when the waited-for message or interrupt arrives.

SUPPOSE provides an operation that creates a process from a programmer-specified procedure (Table II). The programmer specifies the name of the procedure, the name of the process, its execution priority, its data space requirements, and an initialization object. The initialization object is a SUPPOSE dynamic object that provides initialization information to the process. Ownership of the object passes from the creating process to the created process. Exiting the procedure causes termination of the process.

Device management

Devices are resources in applications using SUPPOSE. They are owned and controlled by single processes, the device driver processes.

Devices can be divided into two categories: those for which interrupts are solicited, and those for which interrupts are unsolicited. Solicited interrupts are expected in response to a requested device control operation having been performed, for example, completion of a disk read. Unsolicited interrupts are caused by external events, for example, a character being typed on keyboard. Driver processes for devices in the first category behave like server processes. They receive requests, perform device control operations, wait for interrupts, and return responses. Driver processes for devices in the second category behave like requestor processes. They perform device control operations, wait for interrupts, and send request messages (Table III).

Table II. SUPPOSE process management operations.

create	create process
mypid	return name of current process

Table III. SUPPOSE device management operations.

initio	initialize I/O device
waitio	wait for I/O interrupt
pool	allocate messages for device driver

Each device driver process is usually used by a single server process, referred to as a device handler process, which makes the device available to the rest of the processes in the application. For example, the device handler process for a keyboard would queue input lines or characters and dispense them on request to other processes as needed. The device handler process for a printer would prevent interleaved printer listings by providing processes with mutually exclusive access to the printer for the duration of as many print requests as needed for a single listing.

Interprocess communication

What makes SUPPOSE especially well-suited for distributed applications—and indeed is the basis of the SUPPOSE conceptual framework—is its interprocess communication facility (Table IV). It is only through the SUPPOSE interprocess communication facility that processes on the same or different microcomputers are able to communicate and thus cooperate in performing a task. The SUPPOSE interprocess communication facility consists of a set of operations on messages, divided into operations for requestors and operations for servers.

A requestor obtains services from servers by sending request messages. For each request it sends, a requestor can assume that a response message will be returned, which indicates completion of the service and may include return values.

Usually, a requestor immediately waits for the response to a request. However, a requestor may increase the concurrency of requests by having several done in parallel. Thus, the requestor may send requests to several servers and then wait for the responses. When several requests are

Table IV. SUPPOSE interprocess communication operations.

Requestor Operations	
send	send request to server
response	receive response to particular request
anyresponse	receive response to any request
disregard	disregard response to request
request	send + response
sendoff	send + disregard
Server Operations	
receive	receive request from requestor
respond	respond to request
forward	forward request to another server
defer	defer request until later
continue	cause deferred request to be re-received
length	return length of deferral queue

outstanding, the requestor uses sequence numbers generated by SUPPOSE to keep track of which responses belong to which requests.

Although a requestor may assume that there will be a response to every request, a requestor may not always care about the response. In such situations, a requestor may choose, either at the time the request is issued or afterwards, never to see the response.

A server process receives requests, one at a time, from requestors in first-come-first-served order. Every server has a mailbox, from which it receives all requests, Since requestors can assume that a response will be returned for each request, a server has the responsibility for responding to each request it receives or seeing to it that some other server responds to the request. Moreover, having received a request, a server is required to meet this responsibility before receiving another request.

The most direct way in which a server can meet its responsibility for responding to a request is to perform the requested service and return a response. The response to a request is the request message itself, the server having perhaps modified the contents of the message before returning it to the server.

Another way in which a server can meet its responsibility for responding to a request is to forward it to some other server process, transferring the responsibility to the new server. The forward operation allows a server to process several requests concurrently by delegating the requests to concurrently executing "subordinate" servers.

Sometimes it is not possible for a server process to honor a request immediately. For example, the resource controlled by a server may already be allocated to another requestor. In such situations, a server may use a deferral queue to hold the request until a more suitable time. Deferral of a request temporarily discharges the server's responsibility for responding to the request, so that other requests may be received. Later, when the server is able to handle the request, the server moves the request from the deferral queue to the server's mailbox. A server may have any number of deferral queues, each of which may hold as many messages as necessary.

The constraint that a server must respond to, forward, or defer a received request before receiving the next request implies that server processes are programmed most naturally as initialization code followed by an infinite loop. The loop consists of the only receive operation in the server followed by code that determines which service is being requested and then responds to, forwards, or defers the request (Fig. 3).

SUPPOSE interprocess communication is a message-based adaptation of the monitor concept[1] to a distributed environment. Because requests can be forwarded and multiple requests can be made in parallel, SUPPOSE interprocess communication permits greater concurrency than monitors do. A more detailed description of SUPPOSE interprocess communication is given in the references.

The clockserver process offers three services that provide access to a clock: time, delay, and tick. When the process is created, it is passed to an object that is used to set the clock.

The time service returns the current value of the clock in the response to the request.

The delay service causes the request to be responded to immediately if the clock time is as great as the time argument in the request. Otherwise, the request is deferred until a tick request, updating the clock, is received.

The tick service increments the clock. It would normally be requested by the clock device driver to update the clock. Tick causes the continuation of all previously deferred delay requests which may now be able to be acted upon since the delay may have expired.

```
procedure clockserver (init: object);
  var msg, waitq: object;
    pid: processid;
    i, clock: integer;

  begin
  msg:= nilobject; waitq:= nilobject;
  clock:= getint(init, timefield);
  dispose(init);
  repeat
    receive(msg, pid);
    case getint(msg, servicefield) of
      time: begin
        putint(msg, timefield, clock);
        respond(msg) end;
      delay:
        if (getint(msg, timefield) > clock) then
          respond(msg)
        else
          defer(msg, waitq);
      tick: begin
        clock := clock + 1
        for i := 1 to length(waitq) do
          continue(waitq);
        respond(msg) end end;
  until false
  end;
```

Fig. 3. Clockserver process example.

SUPPOSE implementation

A prototype implementation of SUPPOSE has been developed for the DEC LSI-11 microcomputer. It is written mostly in Pascal,[3] a high-level programming language, which facilitates its reimplementation on other microcomputers. Assembly language is used for programming concepts inexpressible in Pascal, such as processes and interrupts, and sometimes for efficiency.

SUPPOSE executes on a configuration of up to four LSI-11 microcomputers connected by a custom-built intercomputer bus developed by William G. Wong at RCA Laboratories.[4] SUPPOSE was programmed and developed by the authors on the LSI-11 computers using the RT-11 operating system.

An early version of SUPPOSE was converted by Howard C. Edinger, Jr., at RCA Laboratories to run on a configuration of three Intel 8086 microcomputers connected by shared memory simulating an intercomputer bus. The 8086 version of SUPPOSE was developed on a DECsystem-20 computer using cross-development software for the 8086.

References

1. Brinch Hansen, P., *The Architecture of Concurrent Programs*, Prentice-Hall, Englewood Cliffs, N.J. (1977).
2. Britton, D.E. and Stickel, M.E., "An Interprocess Communication Facility for Distributed Applications," *Proceedings of IEEE COMPCON Fall 80*, Washington, D.C., pp. 590-595 (Sept. 1980).
3. Jensen, K. and Wirth, N., *PASCAL User Manual and Report*, Second Edition, Springer-Verlag, Berlin (1974).
4. Wong, W.G. and Levy, W., "A Master-Master Party-Line Bus in a Microcomputer Network," *Proceedings of Electro* 1978, New York, Session 27 2, pp. 1-5 (Apr. 1979).

Dianne Britton is a Member of Technical Staff in the Advanced Systems Research Laboratory at RCA Laboratories. Since joining RCA in 1978, she has been doing IR&D work in distributed processing in the areas of operating systems, design of distributed programs, and database management for distributed data.

Contact her at:
RCA Laboratories
Princeton, N.J.
TACNET: 226-2198

Mark Stickel was a Member of Technical Staff in the Advanced Systems Research Laboratory at RCA Laboratories. He joined RCA in 1978 and was involved in the design, implementation, and analysis of distributed computing systems, especially in the areas of operating systems and databases. He left the company in June 1981 to take a position with SRI International, Menlo Park, Calif.

At your RCA Library

Bibliography Covers Microprocessors, Programming and Programming Languages

As the use of microprocessors accelerates and the art of programming improves, the literature published about these subjects, and acquired by RCA's technical libraries, has grown steadily. In 1978, a bibliography entitled *Books on Microprocessors in RCA Libraries* was compiled. A second bibliography, *Books on Programming and Programming Languages in RCA Libraries,* was issued in October 1979.

With the cooperation of RCA's technical librarians, Technical Information Systems, Corporate Engineering, has now come out with a new bibliography which combines the two listed above and adds to it all of the books which have been acquired by RCA libraries in the last three years on those subjects. The new bibliography, entitled *Books on Microprocessors, Programming and Programming Languages,* includes 700 books, each indexed by author, title, and subject. The selection of titles for inclusion was made from Library of Congress subject headings in the fields of microcomputers, microprocessors, microprogramming, computer programs, programming, programming languages, programming of specific computers, acronyms for computer languages, and other closely related topics. Each book is identified by the libraries which hold it.

For a copy of the bibliography, see the librarian at your location. If you do not have access to a library, a copy can be obtained from:

Doris Hutchison
Bldg. 204-2
Cherry Hill, N.J.
TACNET: 222-5412

J.O. Horsley|S.L. Clapper

A multiple-processor solution for the advanced AEGIS signal processor

This system enhances internal computational capability, minimizes custom equipment design and allows flexible implementation of future designs.

Abstract: *Modern microprocessor technology offers a combination of rapid cycle times and memory capacity that is well suited to the placement of computers near the equipment they support. This paper describes the application of microprocessors in a signal processor design for the AEGIS AN/SPY-1 B Radar System, in which 11 microprocessors satisfy the requirements for equipment timing and control. The microprocessors are arranged in two parallel, sequential configurations driven by a control computer. Thus, system control is centralized, but execution is decentralized.*

The original signal processor for the AEGIS AN/SPY-1 radar was designed in the early 1970s, before microprocessor technology had matured to its present levels. The design, although functionally suitable to its purpose, was relatively large, heavy, and expensive; it also presented difficulties in manufacture and test.

As part of an ongoing Navy program of system simplification and cost reduction, detailed system design trade-offs were conducted during 1973 and early 1974, resulting in part in a modified radar system, designated AN/SPY-1A. Included in these modifications was a packaging simplification of the signal processor, involving incorporation of LSI technology and resulting in a reduction in equipment cabinets from 15 to 11.

Further study efforts in the area of system simplification and cost reduction during 1977 and 1978 addressed application of new technology to AEGIS. One result of this work was a plan for additional performance increase and cost reduction in the AN/SPY-1A Radar System, to be completed in 1983 and redesignated AN/SPY-1B. Further upgrading of the signal processor was a prominent part of the plan, specifically in terms of application of VLSI technology and distributed microprocessors.

Studies and subsequent concept design proved the feasibility of the approach to the signal processor, and the program has moved forward into final design and initial fabrication. Application of current VLSI technology, as anticipated, makes the AN/SPY-1B signal processor significantly lighter, smaller (5 equipment cabinets), and less costly. Important additional benefits, attributable in part to the application of microprocessors, include increases in performance capability, equipment reliability, and ease of manufacture and test.

This paper traces the evolution of the AN/SPY-1B signal processor, beginning with a general description of multiple processor applications, followed by a brief discussion of the signal processor requirements of the radar, the design drivers involved, and the methodology used to achieve a multiple-processor solution for the advanced signal processor.

Multiple-processor utility

Because of their modular nature, multiple-processor systems generally afford higher system throughput rates than do single-processor systems. As throughput rates increase, single-processor systems soon reach technological limits. The conclusion should not be drawn, however, that N processors with the equivalent speed of a single processor can yield the same performance (number of operations in a specified time). Performance in multiple-processor systems is diluted by the increased flow of input/output data and by the sequential nature of the computations. Sequential (or pipeline) processing, however, is generally acceptable in radar applications.

System planners prefer multiple processors for systems that require functional modularity and incremental growth. Such arrangements permit the tailoring of a base system to satisfy different requirements of various users without the need to completely redesign the system for each user.

Multiple-processor systems are also regarded as more fault-tolerant than single-processor systems. When a

Reprint RE-26-8-14
Final manuscript received Aug. 14, 1981.

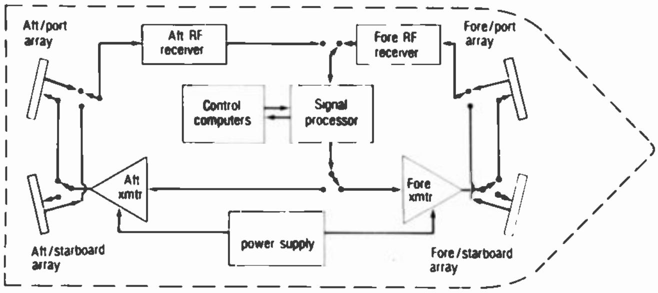


Fig. 1. AN/SPY-1A radar system block diagram imposed on an outline of the CG 47 cruiser in which the radar will be installed. Note position of arrays to provide 360° coverage.

failure occurs, automatic load sharing and proper network selection provide for graceful degradation in performance. When a failure occurs in a single-processor system, however, the complete system fails.

Advances in integrated circuit technology and packaging techniques have enhanced the effectiveness of multiple-processor systems through the contiguous placement of microprocessors with the equipment they support. Such placement simplifies system interfaces because these devices are controlled by standard bus structures. Integration of these microprocessors with the equipment leads to a network eminently suited to radar system applications, as described in the following paragraphs.

AEGIS AN/SPY-1A radar system

The AEGIS AN/SPY-1A radar system, shown in Fig. 1, is a shipboard phased array system that includes four arrays, two RF transmitters and receivers, a control computer, and a signal processor. The radar provides 360 degrees of coverage and can search for, acquire, and track large numbers of targets. The control computer schedules all radar activity, issuing a set of high-level commands to the signal processor for each dwell to be executed. (A dwell is the interval during which the radar beam is in a fixed position.) This command set is translated by the signal processor into a series of controls that configures the system to transmit the specified waveform and to receive the resultant radar returns. From these returns, the signal processor measures target detection ranges in search modes and monopulse angle errors in track modes.

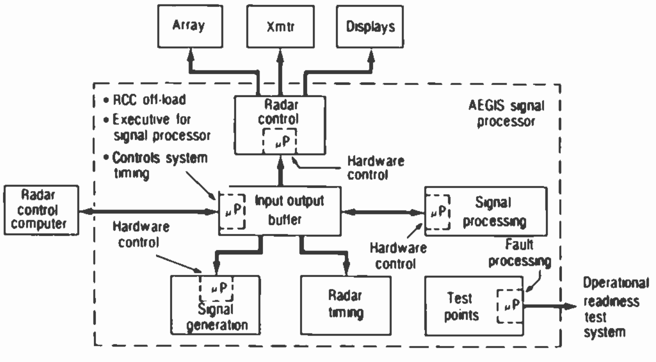


Fig. 2. AN/SPY-1B signal processor block diagram. The addition of microprocessors (dashed blocks) is a major change which, along with other modifications, results in the advanced-design signal processor.

AN/SPY-1A signal processor

The AN/SPY-1A signal processor, shown in Fig. 2, (which becomes the AN/SPY-1B signal processor with the addition of the microprocessors in the dashed blocks) has an input-output buffer for communication with the control computer. It also includes signal generation, signal processing, and radar timing and control equipment. This equipment enables the signal processor to perform the following basic functions:

- Signal generation — includes frequency sources for transmission and reception of all specified waveforms. When ordered, the signal generator also generates simulated target returns at specified ranges for operator training and test purposes.
- Signal return processing — performs detection functions and measures range and angle errors for tracking purposes. It utilizes sensitivity time control and sidelobe blanking, and has a moving-target indicator system for clutter rejection.
- Computer buffering — serves as the primary interface for messages that are interchanged between the control computer and the signal processor.
- System synchronization and mode control — establishes the radar pulse repetition frequency and configures the system to the mode commanded by the control computer. It generates all system timing and control signals to ensure synchronization between transmit and receive cycles.

Evolution of the AN/SPY-1B radar signal processor

Processing requirements

All signal processing is bounded by radar dwells; the required processing is accomplished within a dwell. The control computer, however, can link these dwells to perform a specific function. The AN/SPY-1B signal processor's requirements may be expressed in terms of message handling, processing to be performed, and control needed to properly set up the equipment for data collection.

- Message Handling — On each radar dwell the signal processor must accept a serial block of command data of up to 64 32-bit words from the control computer. Concurrent within the dwell, the signal processor must furnish the control computer with a report that can also be up to 64 32-bit words.
- Processing — Input processing requirements include message decomposition for application to the equipment

and the translation of range-related command words that are expressed in nautical miles by the control computer to radar range bins usable by the equipment. Output data processing requirements include automatic alignment of the system to account for system biases caused by variations in signal paths from the four arrays, and provisions to adjust for long-term drifts of equipment components.

- Equipment Control — The signal generator and transmitter must be set up to transmit one of a set of waveforms for each radar dwell. The return signal processing equipment must be set up for all-range matched-filter processing, either before or after clutter rejection by the moving-target indicator system. Other processing requirements include the setup of timing generators for system synchronization to control the transmission, reception, and processing of radar return signals.

Considering all requirements, the total load translates into a throughput requirement of approximately 16 megabits per second.

Other requirements and design drivers

Like previous designs, the AN/SPY-1B signal processor must interface with the control computer that is tasked to provide overall system command and control. Provisions must also be made to off-load certain functions from the control computer that are best performed at the local level. In addition to these requirements, the signal processor must be highly reliable and provide internal fault processing as part of the AEGIS Weapon System's Operational Readiness Test System.

The selected design must be adaptable to functional changes with minimal impact on equipment design, must allow for incremental growth, and must provide a soft-failure capability.

Network methodology

Extensive treatment of network structure and methodology exists in the literature. The particular application and system performance requirements, however, usually dictate network structure.

In choosing the network for the AN/SPY-1B signal processor, we first analyzed the overall system structure and performance requirements and identified those areas that could benefit from the use of microprocessors. Next, we selected the family of available microprocessors that could effectively do the job and then integrated them into the overall system architecture.

System structure analysis

From the signal processor overview, shown in Fig. 2, the input-output buffer, signal generation, radar control, signal processing, and test-point equipment were identified as areas that could benefit from the use of microprocessors.

A microprocessor in the I/O buffer equipment off-loads certain functions from the control computer and serves as an overall executive for the signal processor system. The functional tasks of an executive microprocessor include:

- System synchronization control;
- Radar control computer communication;
- Radar stimulus command decoding and distribution to local microprocessors;
- Radar target report collection and formulation; and
- Self-test.

Microprocessors, placed at the local equipment levels, are embedded into the equipment they support, and translate high-level orders from the control computer into a detailed set of commands directly usable by the equipment. A functional task matrix is given in Table I.

A microprocessor in the signal processing equipment performs such post-processing functions as computing monopulse errors for tracking purposes. These error estimates are then sent to the control computer for further processing; track files on selected targets are established and maintained. Post-processor functional tasking includes:

- Monopulse angle error estimate computation;
- Auto-alignment;
- Processing of radar signals to assess the environment;
- Target data report formulation; and
- Self-test.

Operational readiness and test system functional interfaces extend throughout the signal processor group. A test microprocessor monitors the performance of all other microprocessors and includes:

- Test stimulus setup;
- Test results collection and evaluation;
- Fault isolation to the lowest replaceable unit;
- Fault report development; and
- Self-test.

Processor selection criteria

To meet our performance requirements, the selected processor must have an average instruction time execution rate of at least 1 MHz. Since the equipment is distributed (imposing a large amount of input/output data at each microprocessor), an input/output throughput rate of about 1 MHz (500 ns per read or write) is also required. In addition to meeting military specifications for equipment, the configuration must allow for future expansion and performance upgrading. Finally, the system must efficiently emulate a military standard (MIL-STD-1750A) instruction set.

Table I. Local microprocessor functional tasking.

Task	*Signal Processing Microprocessor*	*Signal Generation Microprocessor*	*Radar Control Microprocessor*
Accepts selected stimulus command words from executive microprocessor.	X	X	X
Decodes and distributes data to the IF processors, detection processors, and search-track processors for thresholding, etc.	X		
Decodes and distributes data to the waveform generator and auxiliary waveform generator.		X	
Decodes and distributes data to the transmitters, RF receivers, displays, and environment analyzer equipments.			X
Decodes and distributes command words to the post processor.	X		
Sets up hardware counters and buffers for timing and control.	X	X	X
Self-test	X	X	X

Selected microprocessor

The microprocessor selected, shown in Fig. 3, uses the AMD 2900 bit-slice family of integrated circuits. It offers a solution to high precision, high data rate, and special equipment control applications; is configurable as a 16-bit system; and efficiently supports MIL-STD-1750A. It is fast (0.6 μs per instruction), constructed with all Mil-Spec parts, and runs off a single 5-V power supply.

Network selection

The last selection methodology step is integration of the identified microprocessors into an acceptable network. Many networks are possible: stars, rings/loops, broadcast, irregulars, fully coupled, etc. The network selected, based on the stipulated requirements and design drivers, is a symmetrical arrangement of two irregular sides, each of which is the mirror image of the other. This network satisfies the single-failure criterion — system availability with the failure of a single microprocessor — because either side of the network can function independently of the other.

AN/SPY-1B signal processor

The selected network, when superimposed over the equipment, translates into the system shown in Fig. 4. Two of the processors shown perform executive processing, six are used to control local hardware, and two others perform post-processing functions. The eleventh unit, as part of the operational and readiness test system, assesses system operability and is used to detect and locate faults.

The two executive microprocessors process messages

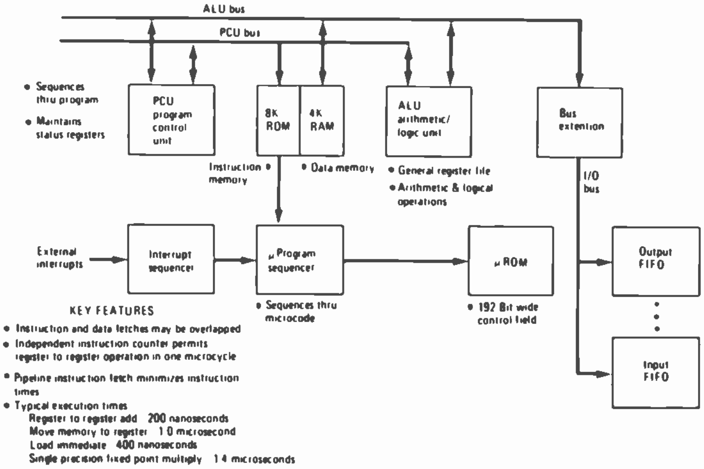

ALU bus
PCU bus
• Sequences thru program
• Maintains status registers
PCU program control unit
8K ROM
4K RAM
ALU arithmetic/logic unit
Bus extention
Instruction • memory
• Data memory
• General register file
• Arithmetic & logical operations
I/O bus
External interrupts
Interrupt sequencer
μ Program sequencer
μ ROM
Output FIFO
Input FIFO
• Sequences thru microcode
• 192 Bit wide control field
KEY FEATURES
• Instruction and data fetches may be overlapped
• Independent instruction counter permits register to register operation in one microcycle
• Pipeline instruction fetch minimizes instruction times
• Typical execution times
Register to register add 200 nanoseconds
Move memory to register 1.0 microsecond
Load immediate 400 nanoseconds
Single precision fixed point multiply 1.4 microseconds

Fig. 3. Block diagram of the selected microprocessor configured in a 16-bit system.

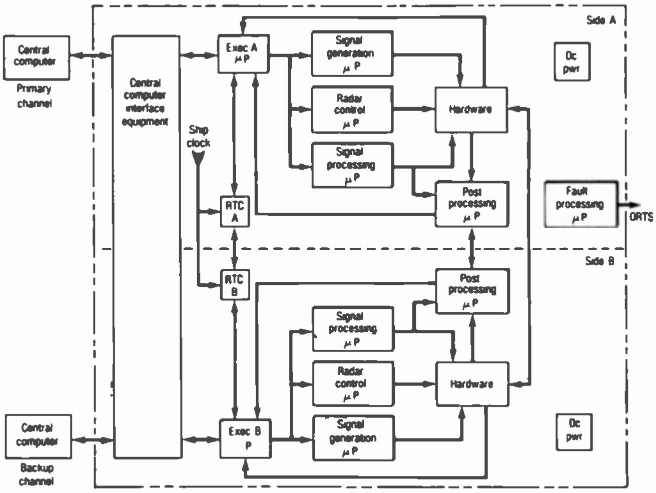


Fig. 4. Selected network translates to functional AN/SPY-1B signal processor.

that are interchanges between the control computer and the signal processor. They accept a command block of data from the computer for each radar dwell and distribute selected subsets of this data to microprocessors at the local level for execution. When radar data is available from the equipment and from the post processor, these microprocessors collect and format a report block for transfer to the control computer where further processing occurs. The executive microprocessors also control real-time clocks that generate system synchronization of timing pulses.

The six microprocessors at the local hardware levels control signal processing equipments, signal generation equipments, transmitters, antenna arrays, and the display equipments. They accept commands from the executive microprocessors and translate them into a detailed set of control signals for their respective equipments.

The two post processors accept radar data from the signal processing equipment and form error estimates for tracking purposes. These units also perform computation to analyze the environment for the presence of electronic countermeasures.

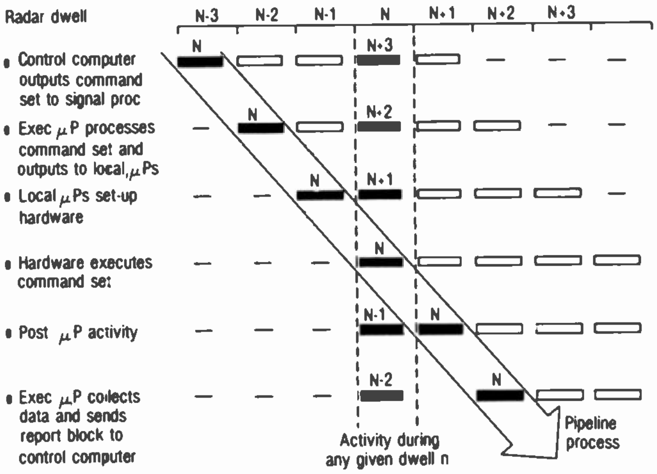
The three-cabinet AN/SPY-1B signal processor includes signal processing equipment comprising four IF processors, four search-track processors, two detection processors, and two environment analyzers. By channelizing this equipment into equal halves, each of which is assigned a portion of the AEGIS waveform to process, the single-failure criterion can be satisfied with no more than a 3-dB loss in radar detection performance. This signal processing equipment is supported by two signal processing microprocessors.

In addition to the signal processing equipment are two signal generators and input-output buffer equipment for communication with the control computer. Each signal generator is assigned a microprocessor as is each half of the input-output buffer. The two microprocessors in the input-output buffer control the transmitters, arrays, and displays.

The design approach in the signal processor structure is duality rather than redundancy. All elements of the system are required for normal operations. If a failure should occur, performance degrades gracefully. One example of duality is that both executive microprocessors are loaded with identical programs. During each radar dwell, they process the same command block and form identical reports for the control computer. Similarly, corresponding local microprocessors execute identical programs in each functional half.

Processing flow

If all processing were accomplished within the same radar dwell, the pulse repetition interval would be too long for this application; it would influence sector search times and the number of targets that could be revisited for tracking purposes. To avoid these long radar dwell intervals, a pipeline process flow was established, as illustrated in Fig. 5. This procedure allocates a radar dwell for processing at each microprocessor level.



Radar dwell
N-3
N-2
N-1
N
N+1
N+2
N+3
• Control computer outputs command set to signal proc
• Exec μP processes command set and outputs to local μPs
• Local μPs set-up hardware
• Hardware executes command set
• Post μP activity
• Exec μP collects data and sends report block to control computer
N
N+3
N
N+2
N
N+1
N
N-1
N
N-2
N
Activity during any given dwell n
Pipeline process

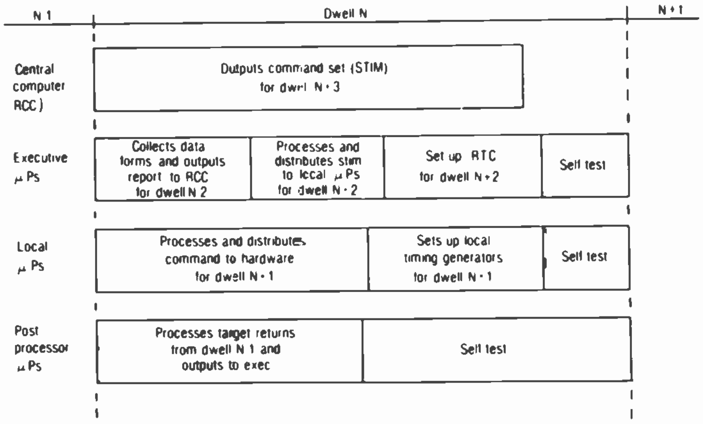
Fig. 5. Processing flow activity designed to perform all functions over six dwells.

As illustrated (diagonal arrow), the central computer sends command data for dwell N during the execution of dwell N-3. This command set is received, distributed, used for equipment setup, and executed during dwell N. Resultant data is available from the post processor during dwell N+1. Thus, the report data from dwell N is formed during the execution of dwell N+2.

This throughput delay meets the response criterion for loop closures by the radar control computer. During any given dwell N (dashed segment of figure), the signal processor receives a command set from the control computer for dwell N+3 and furnishes a report based on data collected during dwell N−2. The flow can be related to intra-dwell microprocessor activity, as shown in Fig. 6.

System bus structure

Each microprocessor has a dedicated data bus, as shown in Fig. 7. During any given dwell, data does not flow between buses (which might create contention), but is interchanged through first-in, first-out (FIFO) buffer memories. This



N 1
Dwell N
N+1
Central computer RCC)
Outputs command set (STIM) for dwel N+3
Executive μPs
Collects data forms and outputs report to RCC for dwell N 2
Processes and distributes stim to local μPs for dwell N+2
Set up RTC for dwell N+2
Self test
Local μPs
Processes and distributes command to hardware for dwell N+1
Sets up local timing generators for dwell N+1
Self test
Post processor μPs
Processes target returns from dwell N 1 and outputs to exec
Self test

Fig. 6. Intra-dwell microprocessor activity, showing actions of radar control computer and ten microprocessors during a single radar dwell.

permits each microprocessor to control all activities on its own bus. Each microprocessor can fetch data from these buffers connected to its bus and output data to FIFO buffers connected to other microprocessor buses. Each buffer stack was made deep enough to hold two dwells of data. In that way, each microprocessor can unload the stack at any time within the dwell interval without placing timing restrictions on any other microprocessor.

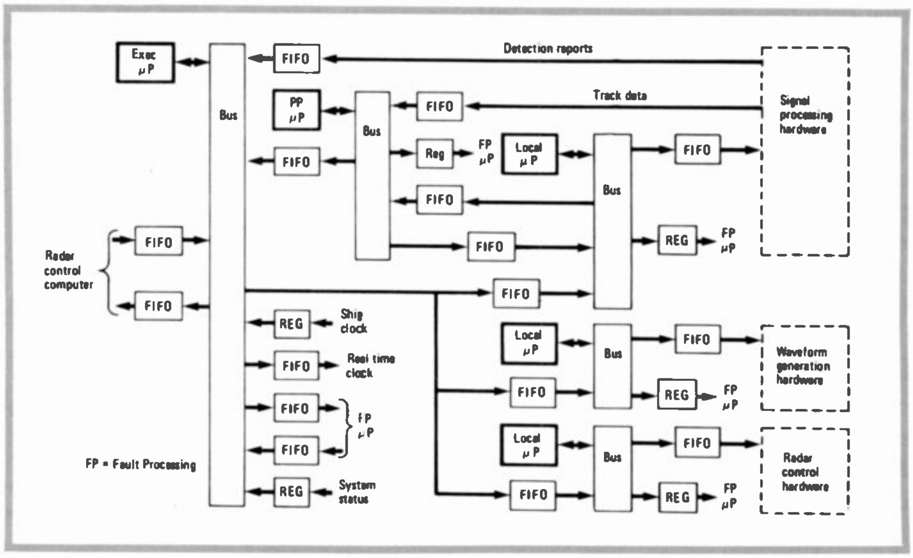


Fig. 7. Bus structure for the AN/SPY-1B signal processor.

Summary

As seen in the case of the AN/SPY-1B signal processor, microprocessors distributed throughout the signal processor have helped to satisfy the overall system objectives. A methodology was followed that identified hardware areas that could benefit from having a microprocessor. Tasks were allocated to each microprocessor and the correct microprocessor selected to meet those tasks. The microprocessors were then integrated into a network that would satisfy the requirements and design drivers.

This multiple-processor solution greatly enhances the internal computational capability of the AN/SPY-1B signal processor. The multiple-processor approach minimizes the design of custom equipment and offers flexibility in the implementation of designs that may appear in the future — including the microprocessor itself.

Conclusions

Multiple processors distributed throughout a system offer the distinct advantages of:

- Modularity;
- A soft-failure capability if appropriately networked; and
- Increased system throughput.

Distributed processor systems, however, do create situations that do not exist in systems with a single processor. While the microprocessor may prove the *sine qua non* for today's innovative designer, certain factors must be addressed and overcome before committing a system to a multiple-processor solution:

- Added I/O;
- Inherent sequential nature of computing; and
- System synchronization requirements.

Multiple microprocessors have been used in the design of an improved signal processor for the AEGIS AN/SPY-1B Radar System. Although this application is unique, these same principles can be applied to other military, industrial, and commercial system solutions.

Steve Clapper (left) enters a line of code while Jim Horsley watches.

Jim Horsley is Principal Member of the Engineering Staff in the Systems and Advanced Technology Department at RCA Missile and Surface Radar. Jim joined RCA in 1958. His areas of expertise include the design of signal processors and of radar timing and control systems. Recently, his assignments have concentrated on the application of microprocessors to radar systems.

Contact him at:
Missile and Sur'ace Radar
Moorestowr, N.J.
TACNET: 224-2055

Steve Clapper is Senior Member of the Engineering Staff in the Systems and Advanced Technology Department at RCA Missile and Surface Radar. Since 1975, when he joined RCA, Steve has designed microprocessors and developed supporting software. He is now designing and fabricating microprocessors for incorporation into high-speed signal processors.

Contact him at:
Missile and Sur'ace Radar
Moorestown, N.J.
TACNET: 224-2401

C.L. Ricker|R.J. Moran

Microprocessor network in operation

RCA Government Communications Systems shows that a less complex software architecture pays off.

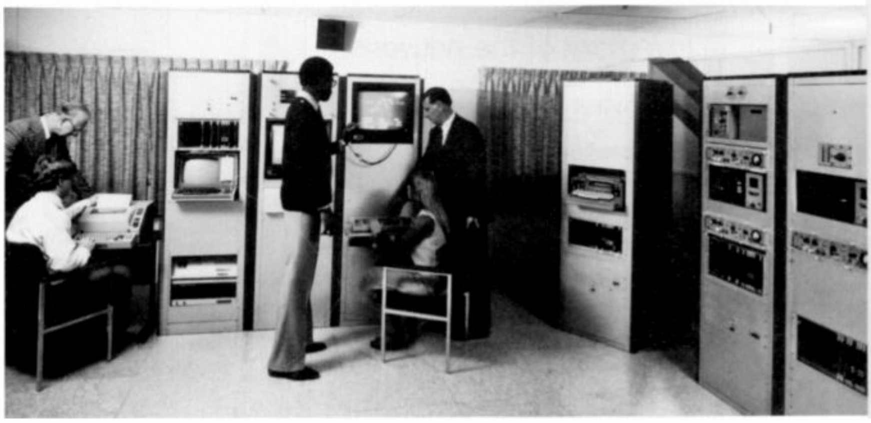


Fig. 2. RCA distributed processor network hardware.

Abstract: *A distributed processing approach to a GCS communications application was undertaken to improve upon expensive software maintenance, system availability and survivability. A description of the system is followed by a section on enhancements of the man/machine interface including touch panels and voice input/output.*

In January of 1980, an extensive IR&D program was instituted for a GCS application that used a network of microprocessors. At that time, only a plan existed — today an operational system exists including 40,000 lines of code, an architecture that weaves together software, firmware and hardware, a functional flow methodology and a substantial array of hardware (Figs. 1 and 2).

This paper will concentrate on architecture and functional flow implementation, but the objectives of the IR&D program will be reviewed first. The IR&D objectives are to conduct studies and develop concepts for:

- □ Man/machine interface
- □ On-line training during network operation
- □ Operational tolerance of hardware and software faults
- □ On-line fault isolation and repair during network operation
- □ Automated start-up and restart

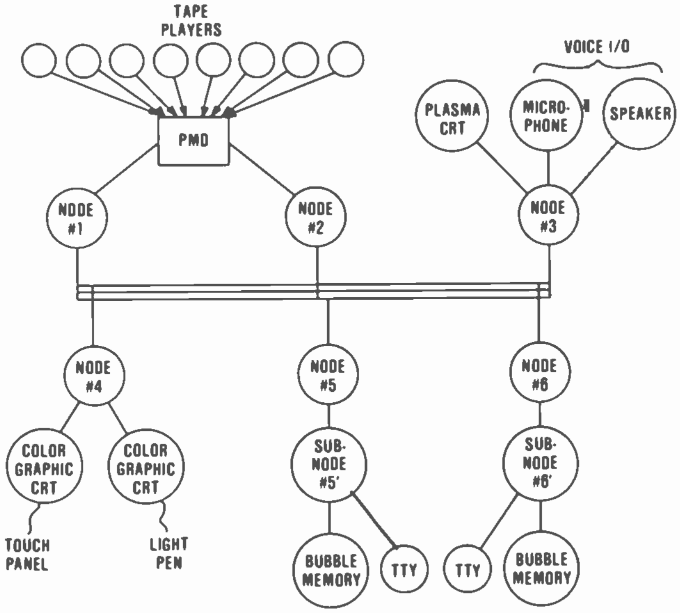


Fig. 1. Distributed functions.

System architecture

The fundamental engineering approach was to construct a relatively complex system out of smaller, less complex parts. The motivation behind this approach was to improve upon system maintainability, availability, and survivability.

In an attempt to identify a system architecture that would support these basic objectives, the distribution of application functions was first considered.

Distributed processing

One dimension within which architectures can be characterized is the degree of functional distribution. Figure 3 shows the spectrum of approaches, which includes a *fully centralized* architecture that assigns all functions to a high-powered central processor, a *federated* approach that consists of a smaller number of multi-functional systems, a *distributed* approach that assigns each major functional area to its own processor, and a *fully distributed* approach that assigns each subfunction to its own processor.

As can be expected, these approaches each have their disadvantages and advantages. The distributed approach was selected because it provides a flexible structure for introducing fault-tolerant design to support a high-

Reprint RE-26-8-15 Final manuscript received July 27, 1981.

Inventory of the network

Hardware

Processors

Nodes	6 INTEL 8080
Sub-nodes	2 INTEL 8080
Embedded	6 Various chips

Buses

RCA Contention (2)
MIL-STD-1553B
IEEE 488

I/O devices

Color Graphic CRT	2 Intecolor
Light Pen	2 Intecolor
Touch CRT Panel	1 Carrol Mfg.
Voice Input	1 Interstate Electronics
Voice Output	1 Interstate Electronics
Plasma CRT	1 Interstate Electronics
Teletype (Militarized)	1 UGC-74
RCA Programmed Demodulator	1 PMD
Bubble Memory	2 Intel 512 KB
Archival Tape Cartridge	1 USH-26
Input Tape Playbacks	8

Support equipment

Software Development	2 MDS-230 (INTEL) 1 LSI-11 (DEC) 2 Line Printers
Firmware Development	2 Techtronix 8002 1 PROM burner

Software

System software

	Lines of Code
Executive (per node)	400
Bus Management	
Contention/Node	500
1553/Node	750
1553 Controller	1000
IEEE 488 Controller	750
IEEE 483/Node	500
Performance Monitoring	2500
On-line Training	1500
Fault Tolerance	3000
Fault Isolation and Repair	2500
Start and Restart/Node	100
Man/machine interface	12000
Applications	15000

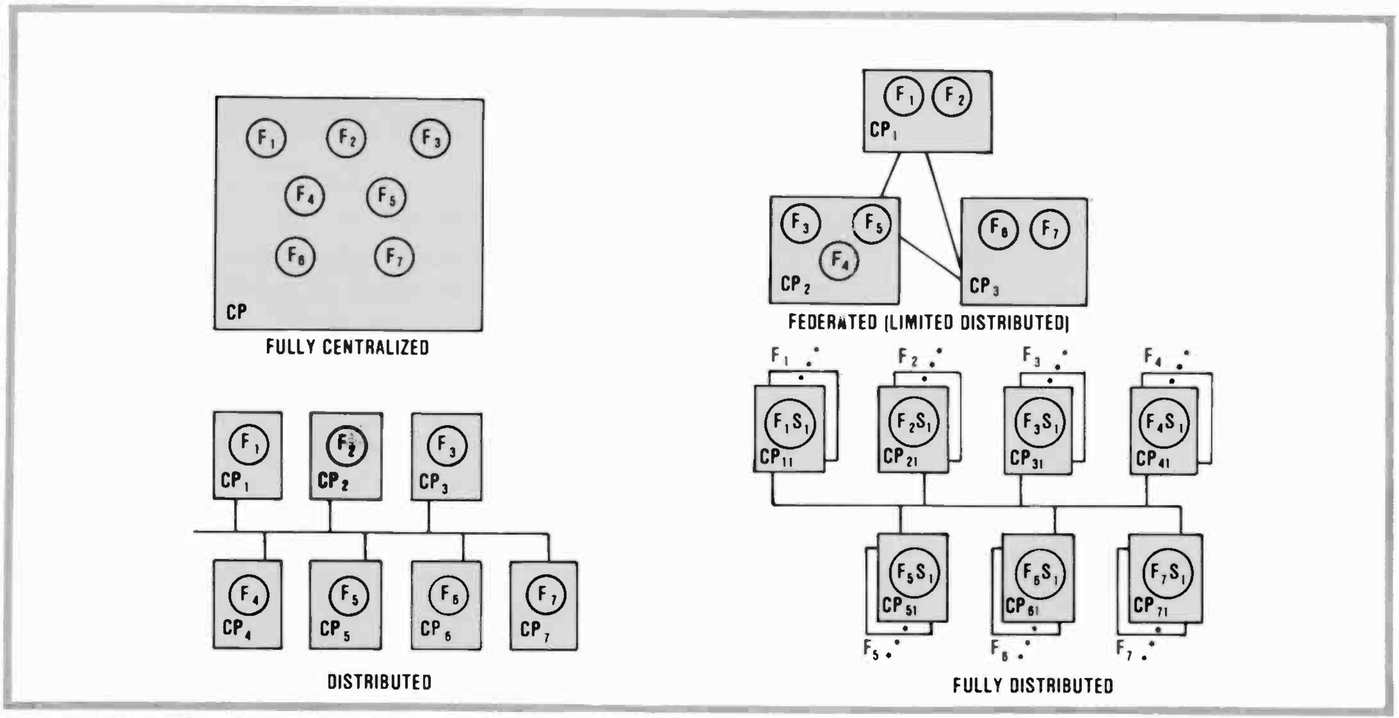

F_1 F_2 F_3 F_4 F_5 F_6 F_7
CP
FULLY CENTRALIZED
CP_1 CP_2 CP_3 CP_4 CP_5 CP_6 CP_7
DISTRIBUTED
FEDERATED (LIMITED DISTRIBUTED)
F_1S_1 F_2S_1 F_3S_1 F_4S_1 F_5S_1 F_6S_1 F_7S_1
CP_{11} CP_{21} CP_{31} CP_{41} CP_{51} CP_{61} CP_{71}
FULLY DISTRIBUTED

Fig. 3. Approaches to functional distribution.

availability grade of service and to decrease the complexity of software.

Single process per node

Once it was clear that a distributed architectural approach was to be used, the overhead associated with distributed systems was considered. The software to support these services — for example, process-to-process communications — can be quite complex and can deplete a large percentage of a project's resources. With this in mind, it was felt that it would be desirable to minimize these services without imposing upon the application.

A typical network service structure consists of a minimum of four levels of control: *physical*; *link*; *network*; and *process*. The services provided by each of these layers is shown in Table I.

A spectrum of approaches to distributing the application functions among processor units can be identified. The four approaches in Fig. 3 represent significant samples along that spectrum.

The physical, link, and process levels were viewed as essential control mechanisms. Also, the routing function within the network control level was also viewed as essential to support the functional reconfiguration. However, the most complex and costly service, the virtual channel mechanism which provides process-to-process communication, could be removed if there was a single process in each node. In such a situation, the link level would effectively provide the necessary process-to-process capability. So it was decided that all functions with more than one software process would be distributed to separate processors and that the virtual channel feature would not be included in the system.

Microprocessor-based nodes

The microprocessor was seen as the ideal host computer for the architecture — distributed single process nodes — that had been identified. Labor and funding resources helped determine that a total of six microprocessor nodes would be used in the net.

The basic nodal configuration consists of an INTEL 8080A-based CPU board and an RS 232C asynchronous serial data interface module, which can be populated with up to 6 ports, 8-kbyte or 16-kbyte PROM and RAM modules, and a module providing parallel interface to the contention data bus which connects all nodes. The architecture provides communication between all nodes. Each node is configured with serial I/O and memory boards to suit its particular application (Fig. 4).

Executive operating system

The executive operating system software developed for each node consists of the following elements:

Table I. Network Control Services.

Network Control Level		*Services*
Physical	Data Communications/ Transport Layers	Electrical signal interchange to establish/disconnect link and transfer data.
Link		Controls data transfer across physical link.
Network		Map logical destination to physical address; support virtual channels.
Process	Application	Interpret message contents and format messages.

□ PROM support program (boot and debug)

□ Single task executive

□ Contention bus driver

□ Peripheral device drivers

□ Real-time clock manager

□ Shared utility routines

An overview of the complete node software architecture is shown in Fig. 5.

Host software architecture

The executive program is continuously polling the prioritized queues for messages generated by device drivers, clock managers, or bus drivers. Depending upon routing information in the message, the appropriate subprocess is scheduled for processing. When the executive is idle, a performance-monitoring program is executing.

Application process allocation

Identified were: the functions which were to be supported; the architectural guidelines providing the framework within which more detailed system definition could be produced; and the physical constraints imposed by the availability and type of microprocessors and the selection of the microprocessor bus. An effort was made to distribute the application processes and associated peripheral equip-

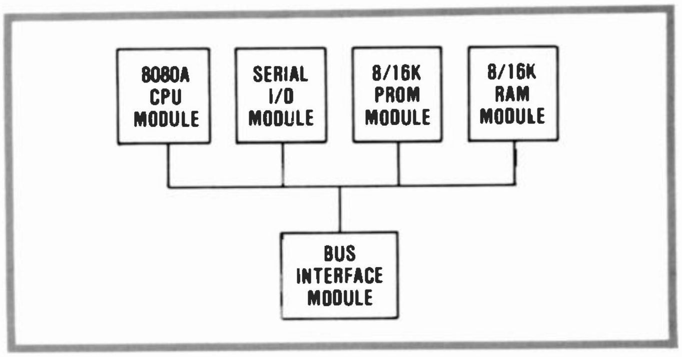


Fig. 4. Nodal microprocessor configuration.

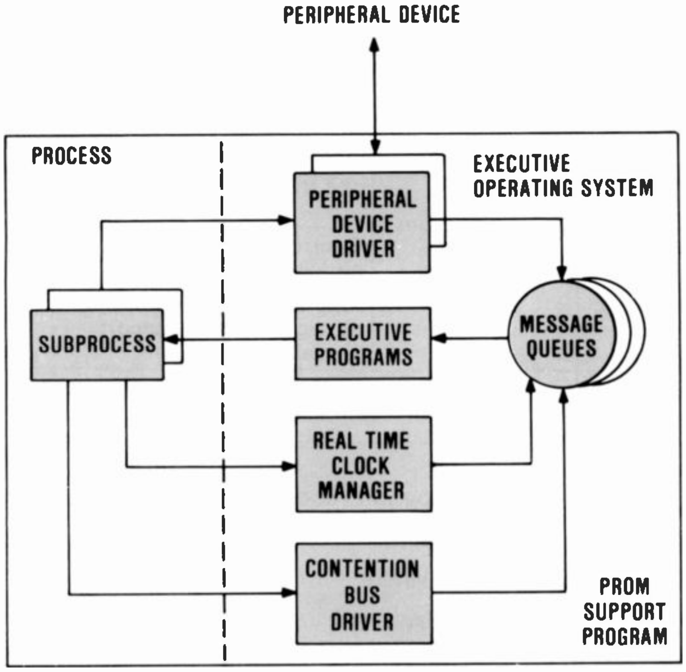


Fig. 5. Host software architecture.

ment among the nodal computers. The results of this effort are shown in Fig. 6.

System application

The system application basically consists of three sets of functional flows: application reception, operator control, and application transmission.

Operator control flows (Fig. 6)

1. The operator sends application control parameters.
2. The operator requests an index of some subset of stored received data.
3. The index is displayed.
4. The operator selects data from that index and requests retrieval from the data base.
5. The ·selected message is displayed.
6. The operator sends retrieved data for display.
7. The operator sends retrieved data to be printed.

Application transmission flows (Fig. 7)

1. The operator prepares data for output.
2. Forwarded to command center for authorization.
3. Following authorization, they are stored in the database for subsequent access by the operator.
4. The operator retrieves the desired data and adds it to a queue of data ready to be output.
5. The operator initiates output of the data.

Application reception flows (Fig. 8)

1. Data is input.
2. The data is forwarded to the MPP for printing.
3. This data is forwarded to the DBP for storage in the incoming data file.
4. High priority data is sent to the CEP.
5. A control message is sent to POP to update the incoming data.
6. The operator selects to review the next received data so a retrieve request is sent to the DBP.
7. The next data for review is displayed.
8. A title is appended and the data is returned to DBP for permanent storage.

Final system

Man/machine interface

Evaluation of the applicability of available man/machine interface technologies is being continued by incorporating touch panel and voice recognition/synthesis capabilities into the existing system.

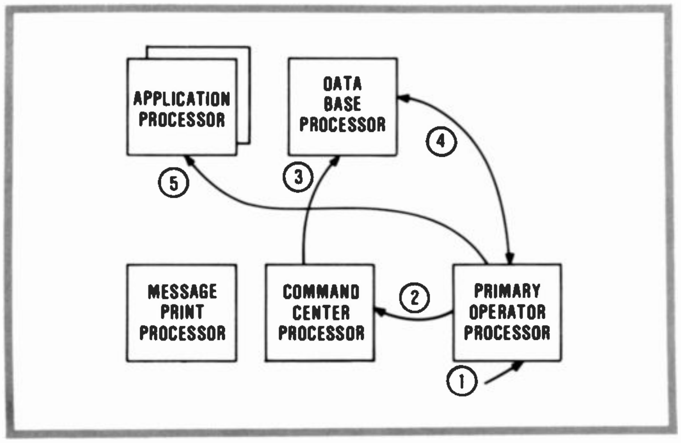


Fig. 6. Operator control flows.

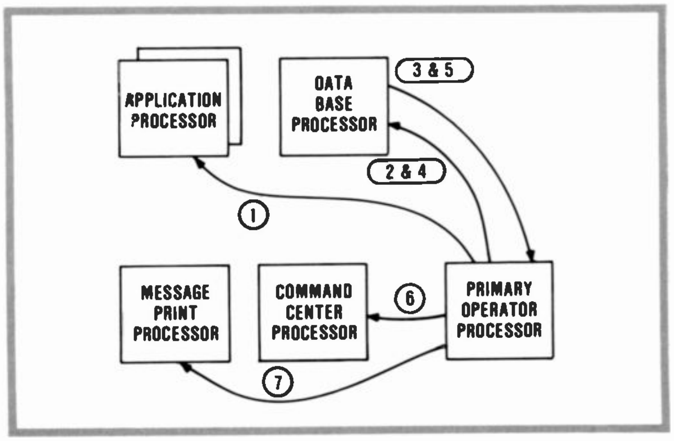


Fig. 7. Application transmission flows.

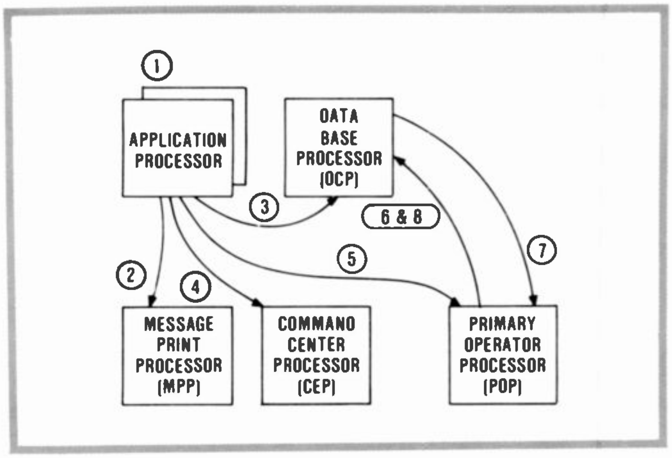

1
APPLICATION PROCESSOR
OATA BASE PROCESSOR (OCP)
3
6 & 8
5
7
2
4
MESSAGE PRINT PROCESSOR (MPP)
COMMANO CENTER PROCESSOR (CEP)
PRIMARY OPERATOR PROCESSOR (POP)

Fig. 8. Application reception flows.

Touch panel

To improve upon the poor reliability and difficulties of operation demonstrated by the light pen, a touch-panel made by Carroll Manufacturing, Champaign, Illinois is being installed on the ISC 8001B color graphics CRT used as the general operator display. Carroll's touch input system utilizes the scanning infrared technology, which is implemented by surrounding the display area with LED emitters and phototransistor detectors. The associated control logic converts interruptions into X,Y-coordinates and transmits them back to the host computer. Existing software handles these inputs identically to light pen input.

The system, which exhibits single character resolution, allows operators the capability of pointing at selection targets with their finger, thus eliminating the need to pick up and put down the light pen.

The availability of resources dictated the use of floppy disks for the storage medium for the existing system. Two enhancements are currently being made to the demonstration system. First, the real-time data base floppy disk system currently in use is being replaced by Intel bubble-memory devices. In addition, an AN/USH-26 cassette tape unit is being added to perform as a tape archival data base.

Other enhancements include the addition of a 1553 microprocessor bus, a quality-monitoring feature, and on-line training and diagnostic software.

Charles L. Ricker (left), Unit Manager in GCS Software Engineering, is shown discussing the impact of replacing keyboard control with voice input/output control. **Richard Moran,** also of GCS Software Engineering, was software architect for the demonstration system. He provided the design and managed the implementation of the software.

Charles Ricker is Unit Manager in Government Communications Systems Software Engineering at Camden, New Jersey. He joined RCA in 1977 and has 30 years of experience in software, with a specialty in the man/machine interface.

Contact him at:
Government Communications Systems
Camden, N.J.
TACNET: 222-4605

Richard Moran was employed in GCS Software Engineering. He has left the company.

Voice input/output

To improve ease of operation, two off-the-shelf circuit boards were added which permit speech exchange with the application system software.

Alerts and responses about input verbal commands will be spoken. Verbal inputs include application commands.

Patents

Advanced Technology Laboratories

Gilson, A.P.|Siryj, B.W.
Protective cartridge for optical discs—4273342

Siryj, B.W.|Gilson, A.P.
Optical disc player system—4271489

Siryj, B.W.|Moore, L.D.
Apparatus for converting rotary motion to linear motion—4274294

Astro-Electronics

Bilsky, H.W.|Callen, P.J.
Redundant battery protection system—4281278

Hubert, C.H.
Orientation of momentum stabilized vehicles—4275861

Phillips, K.J.
Nutation damping in a dual-spin spacecraft—4272045

Consumer Electronics

Fitzgerald, W.V., Jr.
Service switch apparatus—4272777

Fuhrer, J.S.
Defect compensation for color television—4272785

George, J.B.
Power supply arrangement for a tuning system—4281349

Parker, R.P.
Circuit for inhibiting radio frequency interference in a television receiver—4276566

Government Communications Systems

Nossen, E.J.
Range determining system—4278977

Packer, M.
Method for releasing printed wiring boards from printed wiring board racks—4279073

Laboratories

Abeles, B.
Precoated resistive lens structure for electron gun and method of fabrication—4281270

Abrahams, M.S.|Blanc, J.
Method of improving silicon crystal perfection in silicon-on-sapphire devices—4279688

Angle, R.L.
Method for making a closed gate MOS transistor with self-aligned contacts with dual passivation layer—4272881

Berkman, S.|Metzl, R.
Novak, R.E.|Patterson, D.L.
Heat radiation deflectors within an EFG crucible—4271129

Bube, K.R.
Glazing paste for bonding a metal layer to a ceramic substrate—4273822

Datta, P.
Video disc processing—4275100

Datta, P.|Friel, R.N.
Video discs and molding compositions therefor—4280941

Dholakia, A.R.
Selectively damped video disc stylus assembly—4280024

Dieterich, C.B.
PCM detector—4275416

Evans, R.M.
Method for tuning a filter circuit—4272743

Fisher, A.W.
Method of laying out an integrated circuit with specific alignment of the collector contact with the emitter region—4272882

Fukazawa, K.|Yamada, A.
Video disc locked groove clearance system—4278846

Gange, R.A.|Marlowe, F.J.
System for compensating for cathode variations in display devices utilizing line cathodes—4271377

Gibson, J.J.
Video disc playback apparatus with nonlinear aperture correction—4272786

Hanak, J.J.
Tandem junction amorphous silicon solar cells—4272641

Hinn, W.
Automatic kinescope biasing system with increased interference immunity—4277798

Hsu, S.T.
Method for forming buried contact complementary MOS devices—4276688

Hsu, S.T.
Method for forming an improved gate member utilizing special masking and oxidation to eliminate projecting points on silicon islands—4277884

Ipri, A.C.
CMOS SOS with narrow ring shaped *P* silicon gate common to both devices—4271422

Knop, K.
Apparatus and method for measuring the ratio of two signals—4272197

Miller, E.A.
Centering support for a rotatable wafer support susceptor—4275282

Rhodes, R.N.
Color filter having vertical color stripes with a nonintegral relationship to CCD photosensors—4277801

Rockett, L.R., Jr.
Quantizing circuits—4280191

Ross, M.D.
Slow down color processor for video disc mastering using a special mode VTR—4277796

Theriault, G.E.
Saw filter preamplifier—4271433

Tosima, S.
Surface acoustic wave pickup and recording device—4281407

Tracy, C.E.|Kern, W.
Bulk glass having improved properties—4273828

Wang, C.C.|Ekstrom, L.
Lausman, T.C.|Wielicki, H.
Video disc lubricants—4275101

White, L.K.|Comizzoli, R.B.|Schnable, G.L.
Method of detecting a cathodic corrosion site on a metallized substrate—4278508

Williams, B.F.
Method for forming an electrical contact to a solar cell—4278704

Wine, C.M.
Receiver with a channel swapping apparatus—4271532

Missile and Surface Radar

Schwarzmann, A.
Phase shifter—4275366

Picture Tube Division

Farmer, F.C., Jr.|Knight, D.P.
Precision cathode current regulator—4275347

Morrell, A.M.
Color picture tube with screen having light absorbing areas—4271247

Villanyi, S.T.
Cathode-ray tube having corrugated shadow mask with varying waveform—4280077

SelectaVision® VideoDisc Operations

Christopher, T.J.
PCM detector for video reproducer apparatus—4278992

Christopher, T.J.
Stylus position sensing apparatus for video disc player—4280023

Torrington, L.A.
Video disc player having record extracting mechanism—4272083

Wilber, J.A.|Yorkanis, B.J.
Amplifier having dead zone of controllable width and position—4277695

RCA Service Company

Crosby, E.L., Jr.
Balloon with deflation port—4280674

Solid State Division

Angle, R.L.
Method for making a closed gate MOS transistor with self-aligned contacts—4274193

Dawson, R.H.|Schnable, G.L.
Passivating composite for a semiconductor device comprising a silicon nitride (Si_3N_4) layer and phosphosilicate glass (PSG) layer—4273805

Goldman, M.B.|Morton, G.I.
A-C rectifier circuit for powering monolithic integrated circuits—4276592

Harford, J.R.
Gain controlled amplifier using a pin diode—4275362

Harwood, L.A.|Shanley, R.L., 2nd
Color-difference signal processing circuits—4272778

Kaplan, L.A.
Current scaling circuitry—4278946

Knapp, W.K.
Resettable bistable circuit—4275316

Leidich, A.J.
Amplifier circuit—4271394

Malchow, M.E.
Differential FM detector with series tuned filter—4272726

Rodgers, R.L., 3rd
Simplified vertical deflection circuit—4277729

Schade, O.H., Jr.
Switched current source for current limiting complementary symmetry inverter—4274014

Schanzer, H.I.|Stewart, R.G.
Circuit for reducing the loading effect of an insulated-gate field-effect transistor (IGFET) on a signal source—4281400

Webb, P.P.
Photodiode having enhanced long wavelength response—4277793

Wittlinger, H.A.
Differential-input amplifier circuit—4272728

Pen and Podium

Recent RCA technical papers and presentations

To obtain copies of papers, check your library or contact the author or his divisional Technical Publications Administrator (listed on back cover) for a reprint.

Advanced Technology Laboratories

F. Borgini|B. Suskind
The COS/SOS Automated Universal Array—The Custom Integrated Circuits Conference, Rochester, N.Y., *Proceedings* (5/12/81)

A. Feller
Automatic Layout and Checking Programs—The CAD Symposium, Ft. Monmouth, N.J. (4/30/81)

A. Feller
LSI and VLSI Automatic Layout Technique—The Industry Microelectronics Symposium at Mississippi State Univ. and published in the *Proceedings* (5/26/81)

K. Katsumata|S. Ozga
The Advantages of CMOS/SOS VLSI and the GPU Chip Set in Emulating Standard Military Computers—NAECON 81 Conference, Dayton, Ohio (5/19-21/81)

R.F. Kenville
Optical Disc Techniques—The IEEE Computer Elements Workshop, Vail, Colorado (6/23/81)

D. Smith
Automatic Hybrid Layout Program—The CAD Symposium, Ft. Monmouth, N.J. (4/30/81)

M. Stebnisky
Short Channel SOS Performance—The Industry Microelectronics Symposium at Mississippi State University published in the *Proceedings* (5/26/81)

Astro-Electronics

S.M. Fox
Initial Analysis of the Effects of Plume Impingement on the RCA SATCOM I Satellite—AIAA SAE ASME 17th Joint Propulsion Conference, Colorado Springs, Colo. (7/27/81)

C. Hubert
The Attitude Dynamics of Dynamics Explorer A—AAS/AIAA Astrodynamics Conference, Lake Tahoe, Nev. (8/3/81)

J. Swale|R. Josh
A Simple Attitude Data Filter for Three Axis

Attitude Initialization for Autonomous Ascent of Shuttle-Launched Spacecraft—AAS/AIAA Astrodynamics Conference, Lake Tahoe, Nev. (8/3/81)

Automated Systems

M.J. Cantella|H. Honickman|K.H. Kim
IR Focal Plane Array Noise Characterization and Measurements—IRIS Detector Specialty Group Meeting, Syracuse, N.Y. (6/81)

J.C. Phillips|D.R. Higgs (MSR)
Technical Reports and Proposals—Technical Writers Institute, Rensselaer Polytechnic Institute, Troy, N.Y. (6/81)

M.J. Kurina
Electronic Packaging for an Artillery Delivered Sensor—*Electronic Packaging and Production* (6/81)

E.H. Miller
Military Applications of Commercial Computer Technology—*Signal* magazine (7/81)

P. Berrett
A^3 for Project Seaguard—Microprocessor Applications Symposium, RCA, Princeton, N.J. (5/81)

H.L. Resnick
Practical Problems in Applying Microprocessors—Microprocessor Applications Symposium, RCA, Princeton, N.J. (5/81)

W.R. Wadden
Combat Analysis Center—Distributed Processing Symposium, RCA Moorestown, N.J. (6/81)

W.X. Johnson|W.A. Helbig (ATL)
B.R. Eldridge (Tetra Tech, Inc.)
Four Well Chosen Bits—Undersea Surveillance Symposium, Navy Post-Graduate School, Monterey, Calif. (7/81)

Broadcast Systems

D.S. Newborg
CP Antennas for TV Stations—*International Broadcast Engineer*, a UK publication (1/19/81)

Laboratories

A.E. Bell|F.W. Spong
Reversible Optical Recording in Trilayer Structures—*Appl. Phys. Lett.*, Vol. 38, No. 11 (6/1/81)

D. Botez
CW High-Power Single-Mode Operation of Constricted Double-Heterojunction AlGaAs Lasers with a Large Optical Cavity—*Appl. Phys. Lett.*, Vol. 36, No. 3 (2/80)

D. Botez|J.C. Connolly|D.B. Gilbert
High-Temperature CW and Pulsed Operation in Constricted Double-Heterojunction AlGaAs Diode Lasers—*Appl. Phys. Lett.*, Vol. 39, No. 1 (7/1/81)

K.K. N. Chang
An Analytic Solution of a Magnetic Electron-Beam Bender—*RCA Review*, Vol. 42 (3/81)

R.S. Crandall
Determination of the Drift Mobility in High-Conductivity Amorphous Silicon—*J Appl. Phys.*, Vol. 52, No. 3 (3/81)

P. Datta|G. Kaganowicz|A.W. Levine
Alignment and Wetting Properties of Nematic Liquid Crystals on Plasma-Polymerized Films—*Journal of Colloid and Interface Science*, Vol. 82, No. 1 (7/81)

D.P. Dorsey|R.S. Filson|W.H. Tsien
The Design, Construction, and Evaluation of a Porcelain-Steel-Substrate Hybrid-Circuit Module—*RCA Review*, Vol. 42, No. 2 (6/81)

J. Dresner|D.J. Szonstak|B. Goldstein
Diffusion Length of Holes in *a*-Si : H by the Surface Photovoltage Method—*Appl. Phys. Lett.*, Vol. 38, No. 12 (6/15/81)

K.W. Hang|J. Andrus|W.M. Anderson
High-Temperature Porcelain-Coated-Steel Electronic Substrates: Composition and Properties—*RCA Review*, Vol. 42, No. 2 (6/81)

E.O. Johnson|S. Tosima
Visual-Perception-Related Effects in Chinese-Japanese Written Characters—*RCA Review*, Vol. 42 (3/81)

H. Kressel
Lasers and LEDs for Optical Communications Using Fibers—*Radio Science*, Vol. 16, No. 4 (7/8/81)

I. Ladany|F.Z. Hawrylo
Comparison of Single and Two-Phase LPE Growth Methods for InGaAsP/InP Lasers and LEDs—*Journal of Crystal Growth*, Vol. 54, pp. 69-75, North-Holland Publishing Company (1981)

E.W. Maby|C.P. Wy
Carrier Mobility in Laser-Annealed Silicon-on-Sapphire Films—*RCA Review*, Vol. 42 (3/81)

C.W. Magee (Labs)|J.C. Bean|G. Foti
J.M. Poate (Bell Labs)
Observation of Gas Absorption in Evaporated Amorphous Silicon Films Using Secondary Ion Mass Spectrometry—*Thin Solid Films*, Vol. 81, pp. 1-6 (1981)

J.H. McCusker
Finite-Element Analysis of Stresses and Thermal Flow in Porcelain-Enamelled-Steel PC Boards—*RCA Review*, Vol. 42, No. 2, (6/81)

R.W. Nosker|L.A. DiMarco|R. Williams
Deliquescence as the Mechanism for Strong Dust Adhesion—*Appl. Phys. Lett.*, Vol. 38, No. 12 (6/15/81)

L.S. Onyshkevych
Introduction to Special Issue on Porcelain-Enamelled-Steel Boards for Electronic Applications—*RCA Review*, Vol. 42, No. 2 (6/81)

L.S. Onyshkevych|W.H. Tsien
T.T. Hitch|P.R. Smith
Manufacturing Steps in the Production of Porcelain-Enamel PC Boards—*RCA Review*, Vol. 42, No. 2 (6/81)

J.I. Pankove|J.T. McGinn|C.P. Wu
Bombardment-Induced Corrosion Resistance of Aluminum—*Appl. Phys. Lett.*, Vol. 39, No. 1 (7/81)

A.N. Prabhu|E.J. Conlon|A.Z. Miller
J.H. McCusker|T.T. Hitch
Fabrication of Large-Area Thick-Film Hybrid Circuits on RCA Porcelain-Coated-Steel Substrates—*RCA Review*, Vol. 42, No. 2 (6/81)

A.N. Prabhu|K.W. Hang|E.J. Conlon
T.T. Hitch|A. Kusenko
Characterization of Thick Film Compositions on RCA Porcelain-Coated-Steel Substrates—*RCA Review*, Vol. 42, No. 2 (6/81)

A.N. Prabhu|K.W. Hang
E.J. Conlon|S.M. Boardman
Optimization of RCA Porcelain for Compatibility with Thick Films—*RCA Review*, Vol. 42, No. 2 (6/81)

D. Raychaudhuri
Performance Analysis of Random Access Packet-Switched Code Division Multiple Access Systems—*IEEE Transactions on Communications*, Vol. COM-29, No. 6 (6/81)

W. Rehwald|A. Vonlanthen
Evidence for Solitonlike Discommensurations in K_2SeO_4 from Ultrasonic Measurements of the Shear Stiffness c_{55}—*Solid State Communications*, Vol. 38, pp. 209-212

G.A. Reitmeier
Spatial Compression and Expansion of Digital Television Images—*RCA Review*, Vol. 42 (3/81)

P.H. Robinson|R.V. D'Aiello
The Effect of Atomic Hydrogen Passivation on Polycrystalline Silicon Epitaxial Solar Cells—*Appl. Phys. Lett.*, Vol. 39, No. 1 (7/1/81)

A. Sussman|T. Ward
Electrophoretic Deposition of Coatings from Glass-Isopropanol Slurries—*RCA Review*, Vol. 42, No. 2 (6/81)

B.J. Thaler|J.H. McCusker|J.P. Honore, III
Electrical Properties of RCA Porcelain-Enamelled-Steel PC Boards—*RCA Review*, Vol. 42, No. 2 (6/81)

W.H. Tsien|J.H. McCusker|B.J. Thaler
Mechanical Properties of RCA Porcelain-Enamelled-Steel PC Boards—*RCA Review*, Vol. 42, No. 2 (6/81)

L.K. White
Etch Rates of SiO_2 Films in Deuterated Acidic Fluorides—*Thin Solid Films*, Vol. 79, L73-L76, (1981)

L.K. White|R.B. Comizzoli
C.A. Deckert|G.L. Schnoble
The Detection of Corrosion Phenomena with pH-Sensitive Fluorescent Dyes on Aluminum-and-Gold-Metallized IC Devices—*J. Electrochem. Soc.:* Electrochemical Science and Technology (5/81)

J.L. Vossen
Growth, Synergism, and Advocacy—The American Vacuum Society in 1980—*J. Vac. Sci. Technol.*, Vol. 18, No. 2 (3/81)

P.J. Zanzucchi|W.R. Frenchu
Multisampling of Microgram Quantities for Infrared Spectrometric Analysis—*Analytical Chemistry*, Vol. 53, No. 7 (6/81)

Missile and Surface Radar

J. Golub|B.B. Levy|D.M. Greeley
SIMATR, An Air Battle Simulation of the USAF Tactical Control System (TACS) with Advanced Tactical Radars—1981 Summer Computer Simulation Conference, Conference *Proceedings*, Washington, D.C. (7/81)

Solid State Division

C. Field,R. Jarl|C. Salerno
Apply Pulse-Width Modulators to Produce Variable dc Voltage—*EDN* magazine, Vol. 26, No. 16 (8/19/81)

Engineering News and Highlights

Stoeger is Division Vice-President, Engineering

Appointment of **Joseph E. Steoger** to the newly-created position of Division Vice-President, Engineering was announced today by **George D. Prestwich,** President, RCA Service Company.

Mr. Prestwich said that the new position was established to meet a growing need for greater engineering expertise among the company's increasingly complex electronic businesses.

Mr. Steoger will be responsible for integrating engineering knowledge and concepts across the company's varied businesses and for developing plans and programs to match company needs and capabilities to technological advances.

Prior to the appointment, Mr. Steoger was director of engineering support, Consumer and Commercial Services, since 1978. He joined RCA Service Company in 1942 as a field engineer and held a series of increasingly more responsible management positions before being named manager of engineering, Consumer Services, in 1971.

Yannotti is TPA at Astro

Frank Yannotti has been appointed Editorial Representative and Technical Publications Administrator at RCA Astro-Electronics, Princeton, N.J. Frank has recently rejoined AE after spending seven years with RCA Solid State Division, Somerville, N.J. In his present assignment he is Administrator, Engineering Operations, reporting to the Chief Engineer.

Frank joined RCA in 1952 as an electrical engineer in Harrison, N.J. In 1961 he moved to AE, where be built and managed the Environmental Test Center until 1974 when he transferred to SSD at Somerville. There, he served as Administrator, Power Operations, until May, 1981 when he returned to AE.

Contact him at:
RCA Astro-Electronics
Princeton, N.J.
TACNET: 229-3246

Klarmann is Astro's Ed Rep

Carol Klarmann is the new Editorial Representative for the *RCA Engineer* at RCA Astro-Electronics. She joined RCA Astro-Electronics in 1978 as an engineering writer responsible for IR&D documentation. She also prepares Patent and New Technology Reports for Air Force and NASA contracts. Before joining RCA, she worked at AT&T Long Lines editing documentation for tariff filings with the FCC. She holds a bachelor's degree in physics from Douglass College.

Contact her at:
RCA Astro-Electronics
Princeton, N.J.
TACNET: 229-2919

Staff Announcements

Thornton F. Bradshaw, Chairman of the Board and Chief Executive Officer, announces the following changes on his staff. **Kenneth W. Bilby** assumes responsibility for the Corporate Affairs activity. Mr. Bilby was elected as an Executive Vice-President at the August 5, 1981, meeting of the RCA Board of Directors. **William C. Hittinger,** as Executive Vice-President, in addition to his responsibilities for Research and Engineering, Licensing, and Patent Operations, assumes responsibility for RCA Communications, Inc., and for International. **Rocco M. Laginestra,** as Senior Vice-President, serves as special assistant to the Chairman and Chief Executive Officer and will continue his responsibilities for Planning, Marketing and Real Estate activities.

William C. Hittinger, Executive Vice-President, announces his organization as follows: **Stephen S. Barone,** Senior Vice-President, Licensing; **Eugene F. Murphy,** President and Chief Executive Officer, RCA Communications, Inc. (Mr. Murphy is also an RCA Group Vice-President); **John V. Regan,** Vice-President, Patent Operations; **Howard Rosenthal,** Staff Vice-President, Engineering; **Eugene A. Sekulow,** Vice-President, International; **William M. Webster,** Vice-President, RCA Laboratories.

Roy H. Pollack, Executive Vice-President, announces that the Board of Directors of RCA Corporation elected **James M. Alic,** Group Vice-President. Mr. Alic will continue his present responsibilities for: RCA Service Company, "SelectaVision" VideoDisc Operations, and VideoDisc Business and Operations Planning.

Consumer Electronics

J. Peter Bingham, Division Vice-President, Engineering, announces his organization as follows: **Larry A. Cochran,** Director, Signal Systems and Components; **Eugene Lemke,** Director, Advanced Products; **James A. McDonald,** Director, Display Systems Engineering; **Perry C. Olsen,** Director, Product Design Engineering; **Willard M. Workman,** Director, VideoDisc Player Engineering; and at the New Products Laboratory, **James E. Carnes,** Director, New Products Laboratory.

Eugene Lemke, Director, Advanced Products, announces his organization as follows: **Paul E. Crookshanks,** Manager, Project Engineering; **Harry W. Kidwell,** Administrator, Data Management; **James C. Marsh, Jr.,** Manager, Project Engineering; **Jereld R. Reeder,** Manager, Project Engineering.

James E. Carnes, Director, New Products Laboratory, announces his organization as follows: **Billy W. Beyers, Jr.,** Manager, Digital Products Development; **Scott A. Keneman,** Manager, Television Digital Systems; **James L. Newsome,** Manager, Technology Applications; **John C. Peer,** Manager, Television Systems Development; **Richard A. Sunshine,** Manager, Engineering Systems; and **Craig S. Young,** Manager, Advanced Mechanical Engineering.

Larry A. Cochran, Director, Signal Systems and Components, announces his organization as follows: **David J. Carlson,** Manager, RF/IF Systems Engineering; **Roger W. Fitch,** Manager, Components Engineering; **Jack S. Fuhrer,** Manager, Baseband Signal Processing; **Ronald R. Norley,** Manager, Taiwan Coordination and Competitive Analysis; and **Robert P. Parker,** Manager, Television Digital Applications.

Robert L. Pletcher, Manager, Consumer Acceptance and Reliability Testing, announces the appointment of Donald E. Peyton as Manager, Consumer Acceptance and Reliability Testing-VideoDisc Player.

Andrew G. Kolbeck, Manager, Material Engineering and Development, announces that **Robert R. Russo,** Manager, Process Development, will report to the Manager, Material Engineering and Development.

Laboratories

Bernard J. Lechner, Director, Video Systems Research Laboratory, announces his organization as follows: **Frank J. Marlow** continues as Head, Digital Video Research; **Charles B. Oakley** is appointed Head, Satellite Transmission Technology Research; **Leonard Schiff** continues as Head, Communication Analysis Research; **Paul Schnitzler** is appointed Head, Broadcast Systems Research; **Robert E. Flory** continues as Fellow, Technical Staff; **J. Guy Woodward** continues as Fellow, Technical Staff; and **Harold Staras** continues as Staff Scientist.

Dr. Jon K. Clemens, Director, VideoDisc Systems Research Laboratory, announces the appointment of **John G.N. Henderson** as Head, Signal Systems Research and **James J. Power** as Head, Player Control Research.

RCA International

William C. Hittinger, Executive Vice-President, announces the appointment of **Eugene A. Sekulow** as Vice-President, International.

Eugene A. Sekulow, Vice-President, International, announces his organization as follows: **G. Denton Clark,** Chairman of the Board and President, RCA Inc. (Canada); **Louis Couttolenc,** Vice-President, Latin America, RCA International, Ltd. (Bermuda); **Ming Hsu,** Staff Vice-President, International Trade Relations; and **John H. Rich,** Vice-President, Asia-Pacific, RCA International, Ltd. (Bermuda) (Tokyo Branch).

RCA Service Company

George D. Prestwich, President, RCA Service Company, announces his organization as follows: **Martin J. Barnabic,** Division Vice-President, Consumer and Industry Affairs; **George J. Brennan,** Division Vice-President, Management Services and Systems Planning; **Michael F. Camardo,** Division Vice-President, Finance; **Donald M. Cook,** Division Vice-President, Government Services; **Philip J. Martin,** Division Vice-President, Business Development and Strategic Planning; **George D. Prestwich,** Acting, United Kingdom Operations; **Melvin F. Riedberger,** Division Vice-President, Industrial Relations; **Raymond J. Sokolowski,** Division Vice-President, Consumer and Commercial Services; and **Joseph E. Steoger,** Division Vice-President, Engineering.

Solid State Division

Carl R. Turner, Division Vice-President, Product Assurance and Planning, announces his organization as follows: **Leonard J. Berton,** Manager, Systems Modernization; **Thomas L. Cambria,** Director, Management Information Systems; **Angelo D. Checki, Jr.,** Administrator, Strategic Planning; **Larry J. Gallace,** Director, Quality and Reliability Assurance; **Ralph S. Hartz,** Director, Latin American Operations; **Edwin W. Lyons,** Administrator, Systems and Services; **Leonard Mineur,** Director, Materials and Manufacturing Systems; and **Edward M. Troy,** Director, Operations Planning and Support.

Leonard J. Berton, Manager, Systems Modernization Project, announces his organization as follows: **Robert A. Friedman,** Manager, Applications Development; **Paul D. McNamara,** Manager, System Services; and **Lawrence E. Towner,** Manager, Data Base Administration.

Larry J. French, Division Vice-President, Solid State Technology Center, announces the appointment of **David S. Jacobson** as Director, Custom Large Scale Integration.

David S. Jacobson, Director, Custom Large Scale Integration, announces his organization as follows: **Richard H. Bergman,** Manager, Design Test and Applications; **Joseph J. Fabula,** Manager, Production Engineering; **David S. Jacobson,** Acting Manager, Quality and Reliability Assurance; **Burnett Sams,** Leader Technical Staff, CAM; **William C. Schneider,** Manager, Program Management and Business Planning; and **Evan P. Zlock,** Manager, Custom LSI Production and Product Control.

Editorial Representatives

Contact your Editorial Representative at the TACNET numbers listed here to schedule technical papers and announce your professional activities.

Commercial Communications Systems Division (CCSD)

Broadcast Systems

		TACNET
*Bill Sepich	Camden, New Jersey	222-2156
Krishna Praba	Gibbsboro, New Jersey	222-3605
Andrew Billie	Meadowlands, Pennsylvania	228-6231

Cablevision Systems

*John Ovnick	Van Nuys, California	534-3011

Consumer Electronics (CE)

*Clyde Hoyt	Indianapolis, Indiana	422-5208
Francis Holt	Indianapolis, Indiana	422-5217
Chuck Limberg	Indianapolis, Indiana	422-5117
Don Willis	Indianapolis, Indiana	422-5883

Government Systems Division (GSD)

Advanced Technology Laboratories

*Merle Pietz	Camden, New Jersey	222-2161

Astro-Electronics

*Frank Yannotti	Princeton, New Jersey	229-3246
Carol Klarmann	Princeton, New Jersey	229-2919

Automated Systems

*Ken Palm	Burlington, Massachusetts	326-3797
Dale Sherman	Burlington, Massachusetts	326-2985

Government Communications Systems

*Dan Tannenbaum	Camden, New Jersey	222-3081
Harry Ketcham	Camden, New Jersey	222-3913

GSD Staff

*Ed Moore	Cherry Hill, New Jersey	222-5833

Missile and Surface Radar

*Don Higgs	Moorestown, New Jersey	224-2836
Jack Friedman	Moorestown, New Jersey	224-2112

National Broadcasting Company (NBC)

*Bob Mausler	New York, New York	324-4385

Patent Operations

Joseph Tripoli	Princeton, New Jersey	226-2992

Picture Tube Division (PTD)

*Ed Madenford	Lancaster, Pennsylvania	227-3657
Nick Meena	Circleville, Ohio	432-1228
Jack Nubani	Scranton, Pennsylvania	329-1499
J.R. Reece	Marion, Indiana	427-5566

RCA Communications

American Communications

		TACNET
*Murray Rosenthal	Princeton, New Jersey	258-4192
Carolyn Powell	Princeton, New Jersey	258-4194

Global Communications

*William Hartweg	New York, New York	323-7300

RCA Limited (Canada)

Bob McIntyre	Ste Anne de Bellevue	514-457-9000

RCA Records

*Greg Bogantz	Indianapolis, Indiana	424-6141

RCA Service Company

*Joe Steoger	Cherry Hill, New Jersey	222-5547
Ray MacWilliams	Cherry Hill, New Jersey	222-5986
Dick Dombrosky	Cherry Hill, New Jersey	222-4414

Research and Engineering

Corporate Engineering

*Hans Jenny	Cherry Hill, New Jersey	222-4251

Laboratories

Eva Dukes	Princeton, New Jersey	226-2882

SelectaVision® VideoDisc Operations

*Nelson Crooks	Indianapolis, Indiana	426-3164

Solid State Division (SSD)

*John Schoen	Somerville, New Jersey	325-6467

Power Devices

Harold Ronan	Mountaintop, Pennsylvania	327-1633 or 327-1827
John Cadra	Somerville, New Jersey	325-6909

Integrated Circuits

Dick Morey	Palm Beach Gardens, Florida	722-1262
Sy Silverstein	Somerville, New Jersey	325-6168
John Young	Findlay, Ohio	425-1307

Electro-Optics and Devices

John Grosh	Lancaster, Pennsylvania	227-2077

Solid State Technology Center

Judy Yeast	Somerville, New Jersey	325-6248

***Technical Publications Administrators, responsible for review and approval of papers and presentations, are indicated here with asterisks before their names.**

RCA Engineer

A technical journal published by
Corporate Research and Engineering
"by and for the RCA engineer"

Bldg 204-2, Cherry Hill, NJ 08358
Forwarding and return postage guaranteed

Printed in the U.S.A.
Form No. RE-26-8