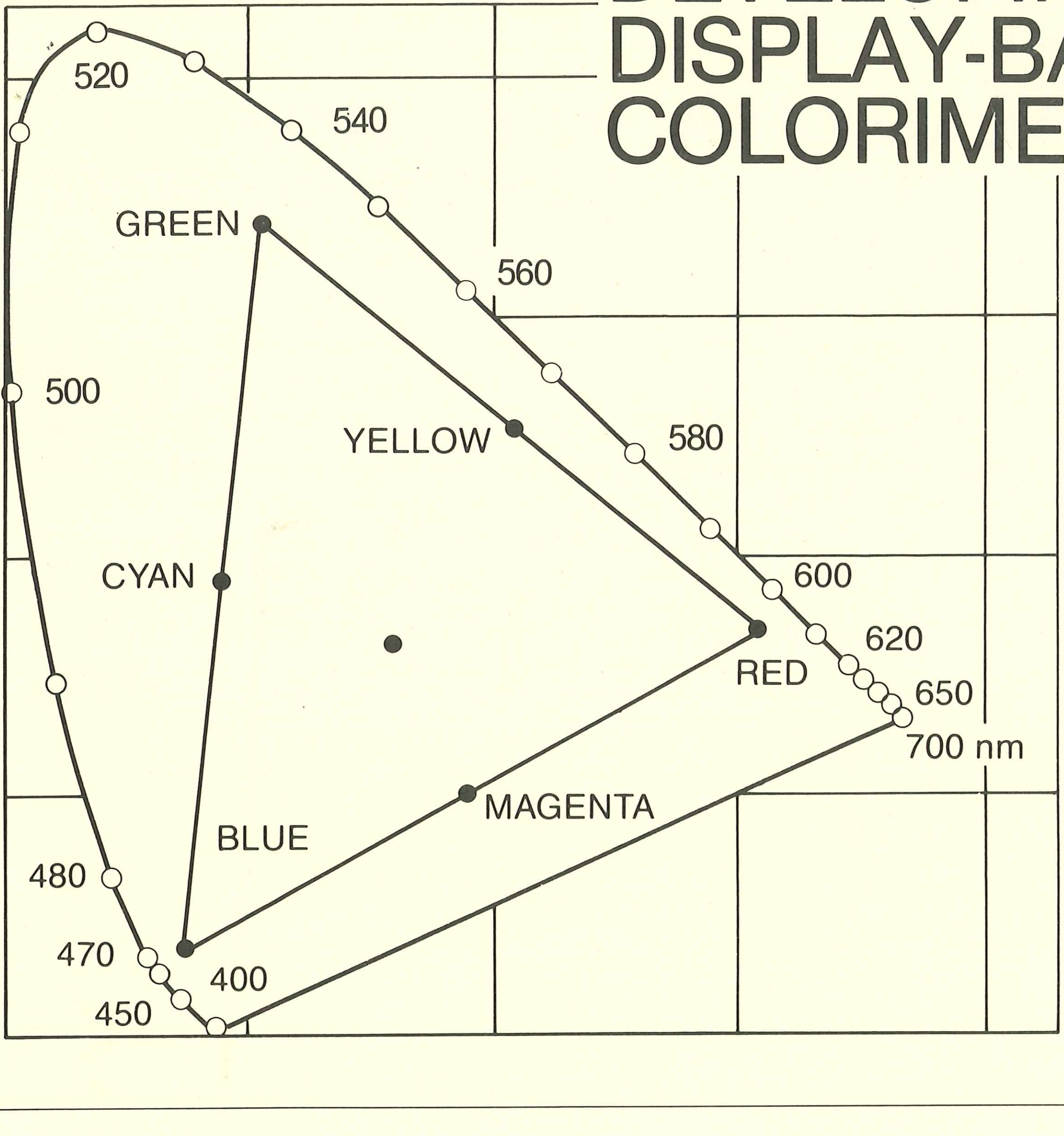
TEK INFORMATION FOR TECHNOLOGISTS FEBRUARY 1984

TECHNOLOGY report

COMPANY CONFIDENTIAL

DEVELOPING A DISPLAY-BASED COLORIMETRY



Tektronix®
COMMITTED TO EXCELLENCE

CONTENTS

Volume 6, No. 2, February 1984. Managing editor: Art Andersen, ext. MR-8934, d.s. 53-077. Cover: Jackie Miner; Graphic illustrator: Nancy Pearen. Composition editors: Jean Bunker and Sharlet Foster. Published for the benefit of the Tektronix engineering and scientific community.

Why TR?

Technology Report serves two purposes. Long-range, it promotes the flow of technical information among the diverse segments of the Tektronix engineering and scientific community. Short-range, it publicizes current events (new services available and notice of achievements by members of the technical community).

HELP AVAILABLE FOR PAPERS, ARTICLES, AND PRESENTATIONS

If you're preparing a paper for publication or presentation outside Tektronix, the Technology Communications Support (TCS) group of Corporate Marketing Communications can make your job easier. TCS can provide editorial help with outlines, abstracts, and manuscripts; prepare artwork for illustrations; and format material to journal or conference requirements. They can also help you "storyboard" your talk, and then produce professional, attractive slides to go with it. In addition, they interface with Patents and Trademarks to obtain confidentiality reviews and to assure all necessary patent and copyright protection.

For more information, or for whatever assistance you may need, contact Eleanor McElwee, ext. 642-8924. □

WRITING FOR *TECHNOLOGY REPORT*

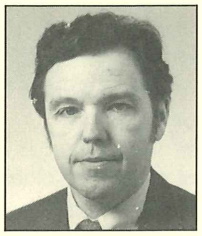
Technology Report can effectively convey ideas, innovations, services, and background information to the Tektronix technological community.

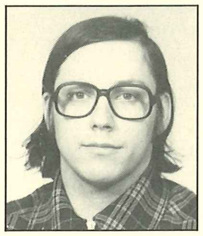
How long does it take to see an article appear in print? That is a function of many things (the completeness of the input, the review cycle, and the timeliness of the content). But the minimum is four weeks for simple announcements and six to 12 weeks for major technical articles.

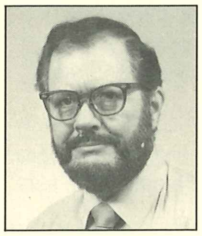
The most important step for the contributor is to put the message on paper so we will have something to work with. Don't worry about organization, spelling, and grammar. The editors will take care of that when we put the article into shape for you.

Do you have an article to contribute or an announcement to make? Contact the editor, Art Andersen, 642-8934 (Merlo Road) or write to d.s. 53-077. □

DEVELOPING A DISPLAY-BASED COLORIMETRY: PERCEIVED BRIGHTNESS AND COLOR CONTRAST OF COLOR DISPLAYS


Gerald Murch, Imaging Research Lab


Micheal Cranford, Imaging Research Lab


Paul McManus, Imagining Research Lab

Using color effectively in any display requires exact control of the visual parameters of color. But in some instances color science doesn't give us the tools to do this. Conventional color science doesn't do a good job of defining display parameters such as brightness and color contrast. Photometric measures of display luminance yield poor estimates of how people will perceive brightness contrast and no estimates at all of color contrast. (A paper on brightness and color contrast is being given at the NATO Conference on Electronic Displays, Farnsborough, England, February 28–March 1.)

Our research, is intended to help Tektronix designers and, ultimately, Tektronix customers use color effectively. Towards this end, we are developing an applied colorimetry of color displays. The work described here has produced a *table of equal color/ brightness ratios.* We recommend display system designers consider this table while structuring user controls and writing software to control colors and brightness.

In any display, color and brightness differences can be used to draw attention, de-emphasize, emphasize, and differentiate. To do this, the display user needs to control the parameters of color and brightness differences, just as the display designer needs to set these parameters to optimize a fixed-format display. Conversely, improper control or use of color and brightness may produce images in which subordinate elements stand out or key details are relegated to the background.

Control of brightness and color becomes imperative for qualitative applications in which uniform changes in color transmit information as in solids modeling and cartography. In fixed-format displays, such as used in avionics, color and brightness differences must be carefully set to be unambiguous over a wide range of ambient lighting conditions: In flying, color confusions could be fatal.

Human Perception

Studies indicate that people can't focus on edges created by color alone. Such contours appear fuzzy and indistinct.[1] The optimal edge should contain both color and brightness differences. Thus, as display system designers, we need a way to accurately assess *perceived brightness.*

Measuring the luminance (photometric measurement) of colors on a display provides only a rough estimate of how the brightness of the colors will be perceived. Luminance measurements are based on the CIE V_λ curve, which only approximates the eye's sensitivity to the visual spectrum. The V_λ curve was derived via *flicker photometry*, in which two small (2°), superimposed monochromatic lights are alternated rapidly to produce a flickering source. One light is fixed in intensity; the observer adjusts the intensity of the other light to obtain minimum flicker. The ratio at this point is a measure of equivalent brightness for the two sources.

Applying photometric measures to visual displays is questionable because:

1. The colors on visual displays are not monochromatic.
2. Colors are often used to fill panels exceeding 2° of visual angle.
3. Colors are viewed in high levels of ambient illumination.

Beyond these limits, recent studies indicate substantial deviations from the CIE V_λ curve when other psychophysical measures are used.[2,3] The most frequent measure is *heterochromatic matching,* in which two different color sources are adjusted to appear equally bright. Heterochromatic matches yield larger values of perceived brightness than flicker photometric matches. The observed differences are by no means trivial. The CIE (see reference 3) notes that the perceived brightness of a narrow-band of red light is underpredicted by a factor of 1.5 while a blue light is perceived as 2.75 times brighter than the photometric measure would predict.

The major source of this difference is that the color difference between the two colors (blue and red) contributes to the impression of brightness in heterochromatic matches, whereas in flicker photometric matches the color difference is eliminated.

The purpose of our research is to measure perceived brightness, perceived color contrast, and combined brightness and color contrast for the NTSC standard phosphor set (P-22) used in most Tektronix raster displays. While our results are valid for only this phosphor set, our procedure can be used to estimate the brightness and color contrast of any display.

Experiment I

In Experiment I, we tested brightness perception with heterochromatic brightness matching and flicker photometry. In the brightness matching, volunteers were asked to view a colored 5 x 5-cm square on a Tektronix 650HR monitor. (A high-resolution television picture monitor.) The luminance was set at 10 cd/m^2. Subjects were asked to vary the brightness of a neutral square contiguous with a colored one of the same size until they considered the two brightnesses to be matched. They were instructed to look for loss of boundary sharpness as an indication of match. This is known as the *minimally distinct border technique*.[4]

In the flicker photometry test, the same colors were presented at the same luminance levels as in the brightness matching test. But here there was only one square, of the same size as the individual squares of the first case. The square was presented as alternating raster lines of a color and of neutral white. The resolution of the display was sufficiently above the spatial resolution of the eye for the viewing distance, so that the square appeared to be of uniform but desaturated color. It also appeared to flicker. The subjects were instructed to adjust the brightness of the neutral component to obtain minimum flicker.

The tests were conducted in a semidarkened room with no light falling directly on the monitor. A black curtain was located behind the observer. At the viewing distance of about one meter, the squares subtended a 4.5° diagonal visual angle.

In Experiment I, a series of seven colors were presented in the sequence red, green, blue, cyan, magenta, yellow, and white. The colors were fully saturated in terms of the display phosphors. (Figure 1 indicates the positions of the phosphors in the 1936 CIE space.) The sequence was presented for heterochromatic brightness matching first, then immediately in the flicker mode, then back to brightness matching for the next sequence of colors and so on. The subject accordingly performed 14 different matches before the first repetition.

The testing was repeated five times, at one sitting, for each subject.

The color sequence was identical for each repetition, but the initial luminance of the neutral color was randomly selected by the program and could be above or below the level of the test square at any degree of mismatch available on the display. The subject controlled brightness through two keys. The keys could be tapped to step the brightness of the neutral square through each of 256 levels or held to give the appearance of continuous change.

For each of the 21 subjects, we calculated the mean and standard deviation of the five matches to each color under both test conditions.

Experiment II

Our second experiment tested the perception of the intensity of the color samples. We replicated Experiment I, except that the luminance of each sample was raised from 10 to 20 cd/m^2. Matches were obtained for only red, green, cyan, magenta, yellow, and white (the 650HR monitor would not produce a blue of 20 cd/m^2)

For each of seven subjects, we calculated the mean and standard deviation of the five matches to each color under both conditions.

Results and Discussion

The perceived brightness measures obtained with flicker photometry for Experiment I are shown in figure 2. The data are given as a percentage deviation of the perceived match from the actual luminance value of each color. While the matching luminances for yellow and cyan differ only slightly from the white, the matching luminances of red, magenta, and blue deviate markedly. Blue, for example, is perceived to be 29% brighter than an equal-luminance white.

The results of the heterochromatic matches for Experiment I are shown in figure 3. The perceived difference is dramatic. The sub-

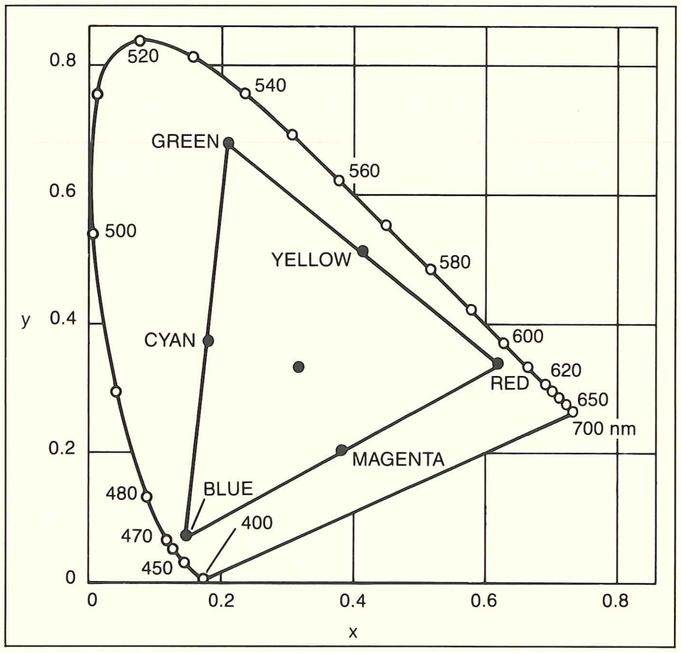


Fig. 1. The locations of test colors in CIE chromaticity space.

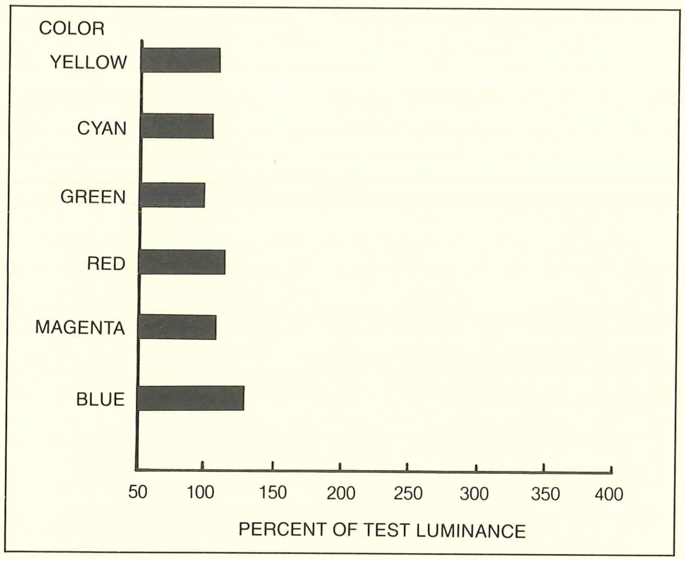


Fig. 2. Subjective brightness: low luminance flicker matches.

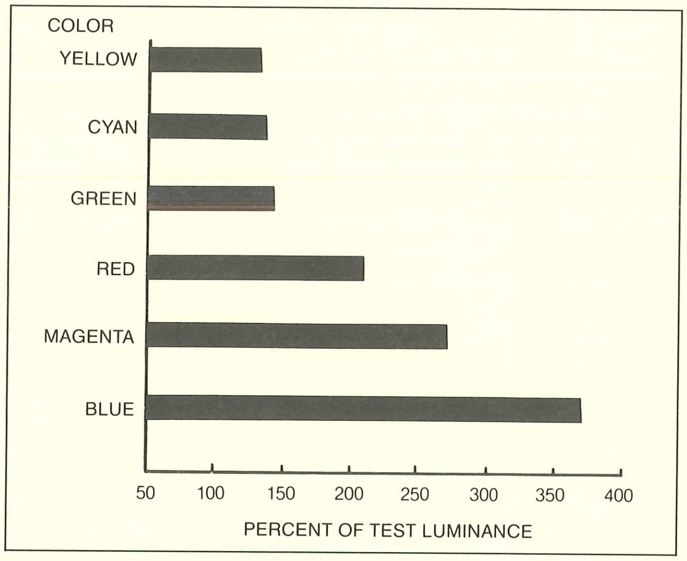


Fig. 3. Subjective brightness and color: low luminance heterochromatic matches.

jects perceive all six colors as brighter than the achromatic (neutral) white. To match the perceived brightness of the blue, for example, with the white required the blue luminance to be 3.7 times lower.

Figures 4 and 5 show the results of matches made at the higher luminance of Experiment II. Note that the flicker matches (figure 4) show roughly the same increase in perceived brightness over the photometrically measured luminance of each color. Statistically, the increase in perceived brightness found in Experiment I did not differ from that found in Experiment II (ANOVA, F=2.1, p<0.05). *Thus, we believe perceived brightness to be definable as a constant percentage of luminance.*

For heterochromatic matches, *perceived brightness cannot be defined as a constant percentage of luminance.* Heterochromatic matches show a significantly less photometric luminance difference from perceived brightness for the higher luminance condition (figure 5) compared to the percentage found for the low luminance condition (figure 3).

At the risk of generalizing beyond limited data, the percent variation from photometric luminance for the low and high luminance conditions can be related via a power function with a slope of 1.27. That is, the perceived variation from luminance for the high luminance test averaged 27% *less* than for the low luminance condition. One exception is the magenta match, which was 68% *less* for the high luminance condition.

Heterochromatic matching involves both brightness and color-contrast components that combine in some manner to yield an overall impression of contrast. A number of models attempt to assign a quantitative index to color/luminance differences.[5] One such metric is the CIE L*u*v* color difference formula (ΔE).

Figure 6 plots the heterochromatic-matching data as percentage increases in perceived brightness against the ΔE values for the six color samples relative to the matching white of equal luminance. The curve represents the best fitting regression line; the fit is very poor.

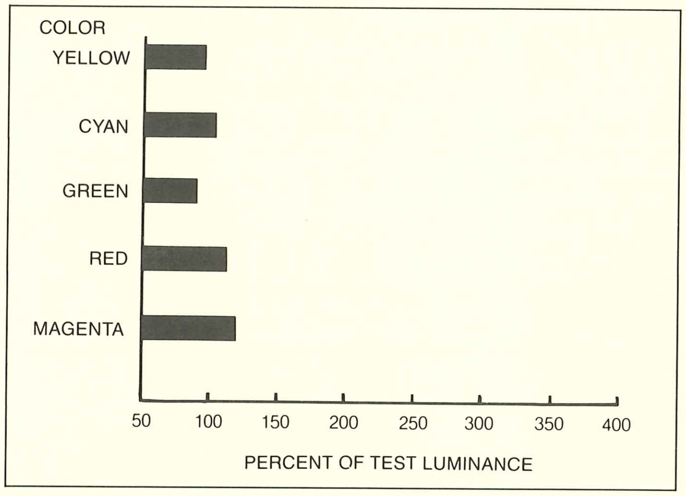


Fig. 4. Subjective brightness: high luminance flicker matches.

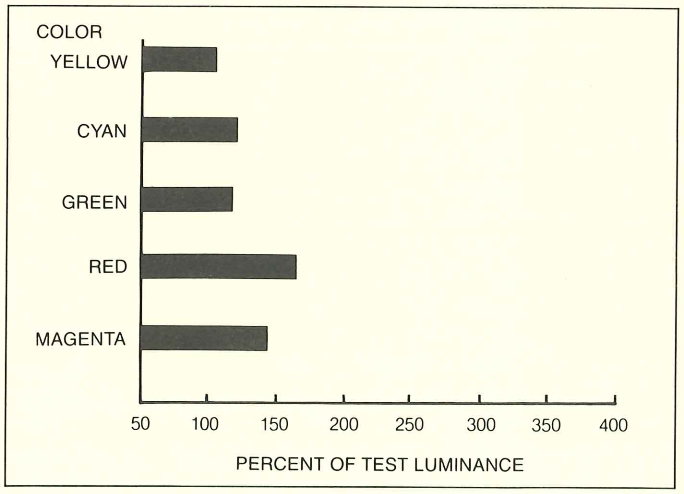


Fig. 5. Subjective brightness and color: high luminance heterochromatic matches.

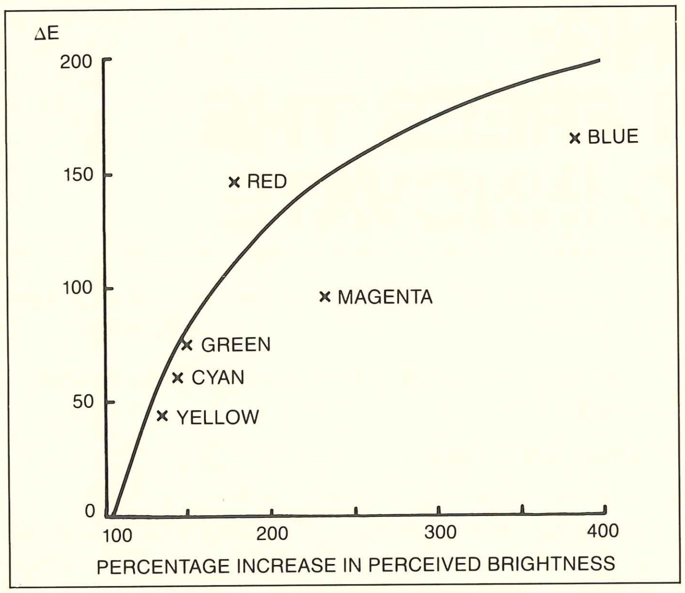


Fig. 6. Heterochromatic brightness matches and the CIE L*u*v* ΔE color differences.

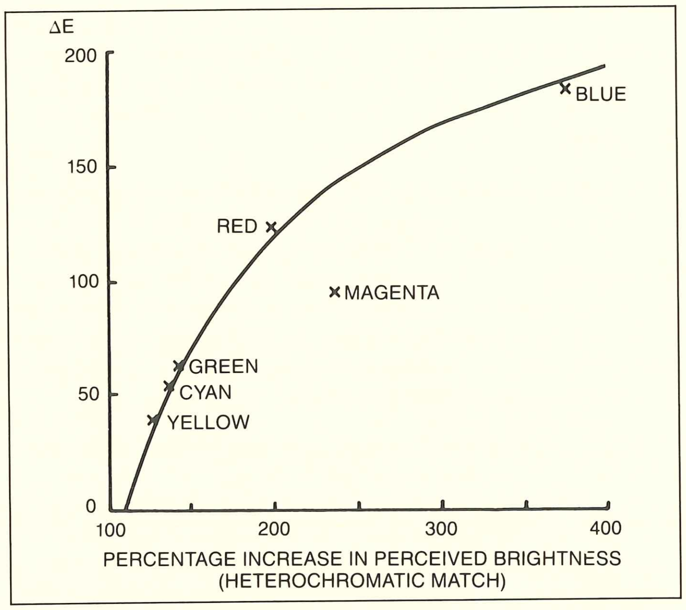


Fig. 7. Heterochromatic brightness matches and flicker photometric corrected CIE L*u*v* ΔE color differences.

In calculating the ΔE values, we defined luminance as the measured luminance of the display. The flicker-photometric matches indicated that *display luminance values do not correspond to the perceived brightness of the displayed colors and, hence, inadequately define color difference.*

Figure 7 contains a plot similar to figure 6 except the actual luminance has been corrected by multiplying by the flicker photometric-matched brightness. We calculated the regression line without the magenta match. With the obvious exception of magenta, this correction to the *CIE* V_λ luminance definition renders the CIE L*u*v* color difference formula a good predictor of color and brightness contrast. An equally impressive fit was found for the high luminance colors.

Because the color contrast predictions (figure 7) were so exact—linear regression R^2 values approached 98% when the magenta match was deleted – we repeated Experiment I with six subjects several months later. The entire measuring and display system was carefully repeated and calibrated. The results were identical to the earlier ones.

At this point in our work, we can't explain the "magenta problem." Others have had problems with uniform color scales in this part of the color space too.[6] The magenta problem can be viewed in two ways:

- Either the flicker-photometric match overestimates perceived brightness, or
- The CIE L*u*v* ΔE value underestimates the color difference of magenta relative to white.

Further research will have to uncover the exact nature of the magenta discrepancy.

In practical applications, the system designer needs to specify a palette of colors that are perceptually equal in values of both brightness and color contrast. Such values can be obtained by forming an adjustment ratio between the luminance of the heterochromatically matched white and the actual luminance of the colored stimulus. When white is set to a specific value, the appropriate luminance of the remaining colors can be obtained by dividing the white value by the adjustment ratio of the color to be matched. Using our test results, we find:

Table of Equal Color/Brightness Ratios

Color	Ratio	Example
White	--	10
Yellow	1.31	7.6
Cyan	1.35	7.4
Green	1.40	7.1
Red	2.06	4.9
Magenta	(2.68)	3.7
Blue	3.69	2.7

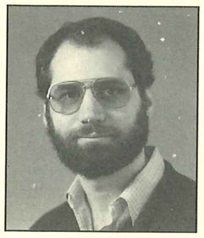
For More Information

For more information, call Jerry Murch 627-5273. ☐

References

1. Wolfe, J., "Hidden Visual Processes," *Scientific American,* 1983, 248, 94-103.
2. Booker, R.L., "Luminance-Brightness Comparisons of Separated Circular Stimuli," *Journal of the Optical Society,* 1981, 71, 139–144.
3. Kinny, J.S., "The Brightness of Colored Self Luminous Displays in the CIE," *Symposium on Self Luminous Displays,* 1982.
4. Boynton, R.M., "Implications of the Minimally Distinct Border," *Journal of the Optical Society of America,* 1973, 63, 1037–1043.
5. Post, D.L., E.B. Costanza, and T.M. Lippert, "Expressions of Color Contrast As Equivalent Achromatic Contrast," *Proceedings of the Human Factors Society,* 1982, 581–585.
6. Wyszecki, G. and W. Stiles, *Color Science,* 1982, Wiley and Sons: New York.

TEST PROGRAMMING: STANDARDIZATION FREES THE TEST ENGINEER TO INNOVATE



David D. Stubbs is a software engineer III in the Electronic Systems Laboratory. Dave joined Tektronix in 1976. He received a BS in history from Portland State University.

Twenty years ago, the control of automated test equipment systems posed unique problems for engineers and technicians developing production testing software. Over the years, most of these problems have been overcome by test-oriented operating systems, and by the evolution of BASIC, the most pervasive language/operating system used for automated testing. With these solutions in hand, attention is turning to more test-oriented languages. This article describes the state of test programming today and reviews the how such programming developed. The author presented a paper on this subject at MiniMicro 83 in November 1983.

Before most products leave the factory, they undergo some form of quality control. This process can vary from a glance for cosmetic acceptability to exhaustive testing throughout the product's full range of performance. Such testing has been costly, because only the more thorough forms of testing can verify the many claims made for complex electronic products. Such testing requires expensive equipment, skilled labor – and it takes time. Reducing these costs usually involved compromising thoroughness and accuracy.

In the 60s, the advent of affordable minicomputers and programmable instruments opened new avenues for streamlining the inspection process. The view then was that labor costs would be reduced by the robot-like activities of instruments and their controllers; these would require only a one-time programming effort by test engineers or technicians.

The machine would have a one-time, tax-deductible cost, and never ask for pay raises or go on strike. Automated test systems would be fast, never pausing for a moment's conversation or a coffee break. Human error would be eliminated in reading and remembering measurements, in calculations, and interpreting results. Thoroughness would be guaranteed because, under program control, every step of the process would be *programmed,* assuring consistent and precise completion of the process.

The task seemed straightforward and the rewards looked promising, and the programming language was usually BASIC.

The language is still BASIC. The rewards have, indeed, been worthwhile. But the task was anything but straightforward.

Historic Difficulties

From the very beginning, test engineers in this new environment faced a broad range of problems. Some problems, like computer programming, required entirely new skills. Others, such as production-data collection, required learning details of activities traditionally practiced elsewhere. There were also obstacles unique to their application. For example, *learning to compute.*

In 1960, programming was as foreign to electrical engineers as S-plane analysis is today to COBOL programmers. Engineers had to learn about commands and syntax, programming structures, files, and archiving. They even needed mundane skills like typing!

When the possibilities of automated systems were first explored twenty years ago, engineers faced still other problems. The early compiler languages were limited. Engineers using FORTRAN, for example, wrote their own routines for "exotic" number formats like octal or hexadecimal. They built their own facilities for manipulating strings of characters and extracting signal parameters from arrays of numbers. The languages also lacked any ability to help with handling interrupts.

The programming environments were fragmented. Engineers used operating-system command languages to invoke text editors. They used mnemonic editor languages to write in the programming languages. They used compiler, linker, and loader commands, and then debugged their programs in still another language. They resorted to assembly language, when they could, to implement communications with instruments through registers.

Writing Tests, Tests, and More Tests

With the introduction of automated test systems, engineers often had to go from "some involvement" in designing test procedures to specifying every minute step their test system was to take, and the number of steps can be enormous.

For simple products like power supplies, there may be as few as a dozen measurements and tolerance checks to verify a handful of performance claims. As more complex products are considered, however, the number of important characteristics increases rapidly.

The characteristics of a Tektronix 7D20 digitizer, for instance, are verified with no less than 70 different tests. If this number seems high, consider the many cases tried for each test. For precision products, characteristics must be verified for all operating modes and for a large sample of signals for each operating range. And, each case can require several changes of control settings and measurements. The control and measurement steps for a complete performance check can number a thousand or more.

Test Sequencing

Test engineers have always specified test sequences. But today, the task is no longer as simple as making some lists on paper. Engineers must build the mechanisms that control the sequence, and these mechanisms must be flexible to accommodate changing test strategies.

Many strategies can be applied to the sequence in which tests are performed. These strategies significantly affect the speed with which products can be examined.

A sequency order like "most-often-failed-first" can reduce the time required to detect a faulty product. If tests check several performance requirements simultaneously, a hierarchy can be constructed. First, high-level tests provide a broadly-defined confidence check. If these are failed, specific low-level tests can provide diagnostic information about individual characteristics. If the high-level tests are passed, the lower-level tests are not required.

Technical and organizational demands often complicate the issue. Tests can depend on other prerequisite tests. They might share measurement values or acceptance margins, or they may be ordered in some "proper" or "correct" sequence.

Controlling the Instruments, Directing the Operator

Test technicians on manual production lines traditionally operated their instruments. They turned the knobs and read the meters. *They made* the calculations and other determinations that led to the pass/fail decision.

In automated systems, the program performs these steps. The engineer selects the instruments, codes the messages that "make the knobs turn," synchronizes activities, asks for and decodes the results, and decides what it all means. These activities have to be orchestrated before the test is run, and perhaps even before the system is built.

Even today, most automated systems still rely on human operation. These human activities can be as simple as physically positioning the product and connecting cables. Or they may be as complex as test sequencing, pattern recognition, and the judging of a product's quality. In any case, test engineers need to describe most operator activities. This calls for precise communication, an unfamiliar task for some engineers, a task that is complicated for several reasons.

The entire operator-system dialogue must be anticipated, and the details are different for every test. The dialogue must be conducted with a host of human factors in mind. For example, if several choices are offered to an operator, the method of choosing should be consistent from test to test. Inconsistencies such as "Select the item," "Enter the corresponding number," and "Type the first letter" provide too much variety – and more difficult learning and human error. It would be better to always present either a numbered menu, a touch-screen menu, or line requesting a typed response.

The entire dialogue must also be encapsulated in the test program. This chore includes not only writing and encoding text, it includes implementing the details of screen formatting, receiving and checking answers, and handling invalid answers.

Presenting data graphically for the operator's consideration can be another challenge. Besides the details of the display, a test might require some response that corresponds to a point or interval on a graph.

And What Results Do We Need for Diagnostics, for Production Monitoring?

When products fail, test results are invaluable for zeroing in on malfunctioning circuits and components.

Test data about a batch or stream of products are also valuable to the manufacturing process itself. Such data can guide changes in sequencing. They can also help detect unwanted variations. The accuracy of a calibration instrument may, for example, be slowly drifting to an unacceptable value. Trends in measurement results can reveal the change before the problem is serious.

Although test engineers are not usually involved in the day-to-day collection and analysis of production data, they must provide for collection and analysis in test programs. A performance requirement, for example, may call for checking a characteristic at 100 different values. What pass/fail information is important? What measurements are relevant? How should the data be formatted and where should it be sent? For even a single test, these questions present a formidable coding exercise, and the complexity increases when the questions change. The production process may, for example, require information that changes daily.

Expecting the Unexpected

Asynchronous events demand a program's immediate attention. An operator's discovery of a misconnected cable, for example, might be cause for interrupting and restarting a test. While the actions to deal with these occurrences can be anticipated, the moment when they are reported is random – or asynchronous. Asynchronus events in test systems typically have four sources: the operator, the test instruments, the test program, and the product under test.

Test operators occasionally need to influence program execution: when some clarification is required, when an operation has gone awry, or for such unautomated activities as breaks, meals, or weekends. There usually needs to be a mechanism to suspend or terminate a test while the interruption is attended. This mechanism must look to the safety of the operator, the test system, and the product. It must maintain the integrity of measurements and test results already collected, and it must provide for restarting or resuming the test.

Instruments occasionally need to report status. Many instruments can report errors in the messages they receive. Some can report when they are overloaded, and some can signal their completion of a operation. Even though the mechanism used to service instrument interrupts is usually different than the one that supports the test operator, both mechanisms must often interact. For instance, an instrument could discover a potential danger for itself or for the operator. The program, when interrupted, might order all instruments to return to their safe settings, and then notify the operator.

And then there is the odd bug. Even in the most carefully written programs, a bug can bring a system crashing to a stop. Errors can be anticipated, and general mechanisms can be constructed that provide a fair chance at correction, recovery, and restart.

Learning to handle interrupts is one of the harder lessons in computer programming. Engineers building automated test systems have received no exemption from its study.

Test programming is responding to these issues: engineers are building test operating systems dealing with the size and complexities of the testing task. Tektronix and other instrument manufacturers are extending their versions of BASIC.

The Test Operating System

The number of tests, the number of cases, the degree of operator involvement, the variations of data collection, the interruptions possible, and a variety of lesser details, multiply a program's complexity and size. A program's documentation and maintenance costs multiply in the same way. Faced with issues like these, test engineers typically explore several avenues before settling on an approach. They might:

- Write large, monolithic programs to implement the entire test process.
- Take a modular approach that calls for writing programs for each test.
- Build an operating system that provides test-level resources – such as test sequencing, signal routing, and data logging – and write sets of compact tests.

In the right situation the monolithic approach may be entirely appropriate. Self-contained programs for incoming inspection of components, or fairly straightforward products like power supplies, can be compact and easy-to-write. But such programs implement only the simplest testing processes.

For more complex devices, even today's near-megabyte controller memories can be hopelessly inadequate unless some form of modularization can be practiced. Test engineers quickly discover the value of the "divide and conquer" strategy called modular programming.

In a common use of this modular strategy, an independent program is written to test each characteristic of the product. In this manner, tests are reduced in size. They are easier to understand, easier to document, and are more likely to work.

But each program still needs its own facilities for operator communications, data logging, and asynchronous events. Adding facilities, produces self-contained – but larger programs. Modular tests can even be divided among a group of engineers and the benefits of their parallel programming efforts realized. Of course, each program will then exhibit a measure of its author's creativity, prejudices, and individual programming style.

The wider problem of flexible operator communications and test sequencing, meanwhile, remains largely unsolved.

Instead of simply carving up the task along the lines of the unique problems – the tests – it is better to partition the overall testing project into common problems.

The partitioning idea allows building a testing language out of a language for general-purpose computing.

The Testing Lanquage

General languages provide resources for problems that must deal with numbers, characters, variables, and communications with particular I/O devices. For such problems, general languages are far easier to use than assembly language or machine codes. Those low-level languages only provide resources for using memory locations, registers, and primitive arithmetic.

As a testing language, pure BASIC is clumsy. Operators should be able to turn on their production systems and start testing. For instance, the appearance of a form for identifying the next device is more appropriate than a blank screen waiting for a BASIC command. A menu of tests is better than a tattered paper list of file names and the instruction:

Type: RUN <file> <Return>

Tests deal with instruments, settings, signals, measurements, tolerances, and pass/fail results. Production testing deals with product types and tracking, fixturing and cabling, and testing sequences. By designing a test-controlling framework and service routines to deal with these issues, engineers free themselves from produceing similar solutions again and again. The resources can be used in standard ways to produce a consistent and reliable process.

Writing modular tests makes far more sense with these issues resolved, too. With each test written to deal mainly with instrument control, data acquisition, signal processing, measurements, and tolerances. Modules are smaller, they are more tractable, and express the verification technique more clearly.

The Evolution of BASIC

BASIC emerged early as the natural choice for many specialized minicomputer applications. Its interactive nature makes BASIC easy to learn and easy to use. Its self-contained programming environment eliminates the need to learn about job control languages, editors, compilers, assemblers, linkers, loaders, and core dumps. The choice was clear for instrument manufacturers; their products could be interfaced with minicomputers and BASIC could be extended to control them.

BASIC has maintained its dominance for three reasons. First, a great many programs and systems have been written in BASIC; this software represents a considerable investment. Second, many engineers and technicians have been using BASIC for years. They know how it works and how to get their job done with it. They are good at it, too. Third, engineers just entering the industry have an investment too: They learned BASIC in school.

BASIC will continue to be the most popular programming language for some time. It is more widely taught than the "computer science" languages. It is taught for problem-solving, and now, it is widely available. In fact, the next generation of engineers is learning BASIC at home.

The test and measurement industry has also invested in BASIC, and this investment has changed it. Several extensions – some unique to our industry and others more widespread – have made BASIC a suitable language for system building:

Subroutines – Callable and functions; with arguments, and recursion (for interrupt processing)

Program overlaying and chaining – BASIC's memory management tool

Interrupt processing – for asynchronous events of all kinds

Device drivers – for communication with instruments

Long variable, label, and subprogram names – support for names up to 32 characters; this benefits program readability and maintenance

Character string manipulation – highly refined and essential for communications with instruments and operators alike

Graphics – for display of signals, trends, and diagrams

Signal processing – for domain transformation, statistics, and measurement extraction

Block programming structures – IF/THEN/ELSE, DO WHILE, CASE, and SUB-PROGRAM structures have kept BASIC in the programming mainstream

Labels – replacing line numbers as more readable branching destination

Development environments – highly flexible program editors and debugging capabilities, including break-pointing, single-stepping, and a variety of traces

Compilers – the latest and perhaps most significant change for BASIC. Compilers increase execution speeds by several times, and open possibilities for mixing, compiled and interpreted modules.

The four most important extensions to BASIC are worth a closer look:

1. *Subroutines* – The concept of reusing sets of instructions is fundamental to all programming. Reuse not only saves programming time and computer memory, it hides detail and makes the programming language more appropriate for the task. Early BASICs proved too restrictive for test-system building, providing only a GOSUB/RETURN construction for subroutines. And returning was a strictly ordered process.

Subroutines have been extended to handle interrupts. Some BASICs even allow higher-priority tasks to call routines already in use, this requires re-entrant subroutines. A variety of returns has emerged, corresponding to the variety of desirable ways to exit subroutines. It is appropriate, for instance, to return and resume an interrupted task after servicing some events. But other events require completely terminating the interrupted process and returning so the test operator can initiate another activity.

Some other helpful extensions of the subroutine concept have included:

- Named, callable subroutines
- Functions written in BASIC
- Argument passing by value and reference
- Local variables and error processing

2. *Program overlaying* – Program overlaying copies program segments from a storage device into high-speed memory under program control. Overlaying allows many segments to timeshare the same memory. It permits service routines and tests to be stored as independent modules, retrieveable. Overlaying and the delete command are the primary memory management tools for operating systems written in BASIC.

3. *Interrupt processing* – The ability to interrupt program execution to attend a priority task is essential for both general-purpose and test operating systems. Asynchronous events can arise from the operator, the instruments, or from program conditions. Several capabilities are required to implement interrupt processing.

There must be some way to associate – that is, tie – a potential interrupt with a body of program code that will service the condition. Associations can vary from general statements which detect a whole class of interrupts:

```
ONERROR GOTO 10000
```

to those which detect very specific conditions:

```
WHEN DMM HAS "Over-range"
CALL Over-Error
WITH PRIORITY 2 AS TASK 5
```

A notion of *interrupt priority* is also useful to test programmers. For instance, an "emergency stop" interrupt might take precedence over an instrument's "operation complete" message, while another product's "power-up" signal might be ignored altogether. Interrupt processing may require a number of ways to return from the handler. Some situations may be ignored altogether. Some might require retrying an operation. Still others could require terminating the current task and return to a known state for another try.

This last situation – returning to a known state – suggests the usefulness of *multitasking* – several separate but interdependent program segments operating with their own identities, priorities, and renewable life-times. Multitasking, for instance, makes it possible for a test to run as a separate task. The test can return when the task is done; or it can be terminated (for example, by errors or operator control) without knocking out the operating system and returning to BASIC's ubiquitous

```
*READY
```

4. *Built-in device drivers* – versions of BASIC provide commands for bit-manipulation, and other commands, like PEEK and POKE, for accessing memory locations. But there are far more satisfying ways to control instruments than through their interface registers. Instrument-control BASICs typically provide print and input access to external devices.

When instruments are used in a system, drivers prove extremely helpful. The drivers work out of sight, handling communications protocols and putting data into standard forms. They are accessed through IEEE-488-standard print and input commands or through other general extensions. Some manufacturers, however, tightly couple their instruments to their own BASIC by adding their own special-purpose commands. These differences reflect two diverging strategies of instrument system building:

Instrument-level system building is the more established strategy; the one followed by Tektronix. It provides general ways for connecting instruments and computers. The IEEE-488 protocol (General Purpose Interface Bus) is the standard for this kind of system. Almost all instrument-control BASICs and programmable instruments support its use. The RS-232 protocol is also available in many instruments, and standards for much faster channels – like IEEE-802 – are emerging. These standards all help system builders integrate programmable instruments more easily.

Board-level system building is based on the "sudden" availability and acceptance of personal computers and represents the trend to put instruments inside the computer. Because board-level systems have only recently gained momentum, standards for these computer-based instruments are not established. Board-instrument manufacturers either provide a language tailored for their instruments or leave it to the consumer to implement communications.

The Future of Test Programming

By building test operating systems, engineers have moved programming from the realm of for-loops and print statements into programming phrased in terms of the verification task. Four lessons are clear from this process.

User-extendible languages

Test system engineers have struggled and largely succeeded in building useful test-system control elements out of general-purpose languages. Their ability to develop understanding as they developed systems was one of the main reasons for their success. Test engineers have mastered the intricacies of controlling instruments with the simplest of commands and have packaged that mastery in the various module forms the programming language supports. These modules can be trusted because the engineers understood why and how they work.

These modules are easy to learn and change. They serve well as models for extensions because the extensions can be written in the same language.

Because such test system languages are user extendible, better mechanisms for building and using these modules are needed. Procedural routines should be added directly to the language as they are written. There is no advantage, for example, to having CALL before a routine name.

Though not yet widely used, data types have great potential in test programming. A waveform, for instance, might be built from a numeric array of signal data, strings for the horizontal and vertical units, and a scalar to hold the data-sampling interval. Operations on waveforms would keep these units in order. Languages like Modula-II and ADA provide powerful ways for such module extension.

System-building resources

System programming languages besides packaging both procedures and data together, must help activate and manage these packages. Languages must have highly refined capabilities for handling interrupts and errors. Serious test-system building requires facilities for managing multiple tasks: setting and changing task priorities, and using them in a re-entrant fashion. To coordinate systems activities, real-time (clock synchronized) scheduling is also useful (for example, for device synchronization, periodic data logging, and test system recalibration).

Unified programming environment

BASIC maintains its popularity because it offers a single, friendly programming environment. It is as English-like as any programming language, and it is interactive; a command can be entered and executed immediately and the results help correct the uncertainties right away.

BASIC also provides a complete programming environment: Algorithms are expressed in BASIC; they are edited in BASIC; they are debugged, saved, copied, and listed – all using the same command language.

Any system programming language offered to replace BASIC will have to have a similarly uniform and simple development facility.

Applications Utilities

It is fair to expect test-system vendors to provide utilities ranging from simple and isolated routines to complete computer-aided test operating systems – Tektronix does, some others don't.

These utilities, whether in manuals and application notes or available on electronic media ready for execution, "transfer" experience to system builders and save them valuable time.

For example, network-communication utilities can be helpful because test systems operate across different production facilities. Test results, instrument status, and inventory tracking information are generally required at several points in the manufacturing process. While a vendor-supplied set of communications routines might not fill the need exactly, they can be modified or serve to guide a development effort.

As a final example, a test-system vendor might supply code and documentation that generalizes three forms of high-level communications:

- Presenting instructions and waiting for continuation signals
- Presenting choices (options) and receiving the options selected
- Requesting and receiving data

Each of these operations might be implemented using a pair of routines at a level somewhat closer to the base language:

- Present a menu of option(s)
- Receive a selection

The menu routines would make screen formatting decisions and issue commands as screen coordinates and character strings to a generalized I/O device driver.

With the details of these straightforward examples in hand, test-system builders could save literally weeks of development, design, and implementation time even if the requirements or resources did not exactly match.

Conclusions

There is no need for test engineers to struggle with recreating the same system solutions that have been built so often. Test-system requirements are increasingly clear and many scattered innovations are waiting standardization. This standardization enables the test engineer to innovate without reinventing the basics. Such reinvention, beside being wasteful, complicates the support of systems and makes user extensions unwieldy.

To be successful, makers of instrument-system hardware and software must provide:

- User-extendible system programming languages
- Languages that provide system-building resources
- A unified test development environment
- Application utilities

For More Information

For more information call David Stubbs, 627-2627.

Acknowledgements

I would like to thank Bob Ramirez and Paula Ochs for their editorial help in preparing the MiniMicro 83 paper that was used to build this article. □

TEAM WILL SUPPORT VAX SYSTEM PROPOSALS

The DEC Systems Configuration Team has been established to support DEC VAX 11/750 and 11/780 computer systems in these areas: hardware maintenance, operating system and application software, data communications, networking, facilities management, computer operations, and procurement management. Before CCA approval, the Configuration Team will quickly, but thoroughly, examine each VAX proposal to ensure that these areas will be supported.

As Configuration Team chairman, Dawn Vance will coordinate user requests and questions through the appropriate support group. Dawn will also summarize supportability issues for each proposal; this will be done before CAD/CAM Committee review. Therefore, if you are going to acquire a VAX 11/750 or 11/780 you should contact Dawn when you initiate project planning; she will be involved throughout the CCA approval process.

Dawn can be contacted at 627-5068, 50-454. □

GOLD-PLATED SWITCHES GIVING YOU PROBLEMS?

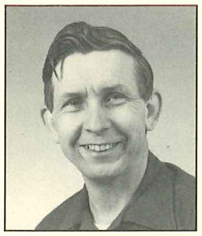
by Jerry Holly, Metals Research and Development

Having a problem with gold switch contacts?

Although I may not have a solution to your problem, I do have some knowledge and experience with gold-plated switch contacts. For example, I have found that the structure of the substrate to be plated is just as important as the properties of the gold itself. Also, some golds perform better than others, but these performance characteristics are not called out in Tektronix specs.

Interested? I would be willing to share this knowledge with you. Call me, Jerry Holly, 627-0303. □

DUALPORT MEMORY PACK FOR YOUR 4050 COMPUTER



Gary A. Spence is an engineering assistant with MOS Special Products, part of ICE. Gary joined Tektronix in 1972. He has helped develop CRTs for the 4503, 7912, and 7612 and has provided technology support for EBS and CCD. Earlier, he worked for Progress Electronics of Oregon on a NASA downrange missile tracking ship as an electronics specialist. Gary holds Electronics Technician, Precision Measurement, and ECM Specialist certificates from the U.S. Navy.

In these days of programmable tests and large volumes of test data, it is often necessary to save high-speed digital data in a buffer so that a lower-speed computer may interface with the buffer and manipulate the data. But most available buffers are cumbersome, too small, or too slow.

Because commercial buffers were not suitable, I designed and built the Dualport Memory Pack. The Dualport Memory Pack is part of a measurement system that includes a 4051 computer. The system produces a 1024 by 1275 memory map of the meshless scan expansion (MSE) lens (see figure 1). We use this map to evaluate lens cutting uniformity.

Another use for the Dualport is logging IEEE-488 (GPIB) communications; it can hold 2000 data handshakes and control line status indications with each. The Dualport has proven very useful in analyzing the communication formats of GPIB-compatible equipment (see figure 3).

The Dualport Memory is built as an intelligent dual-access buffer memory that can be controlled from a user interface or by a host 4051 computer (see figure 2).

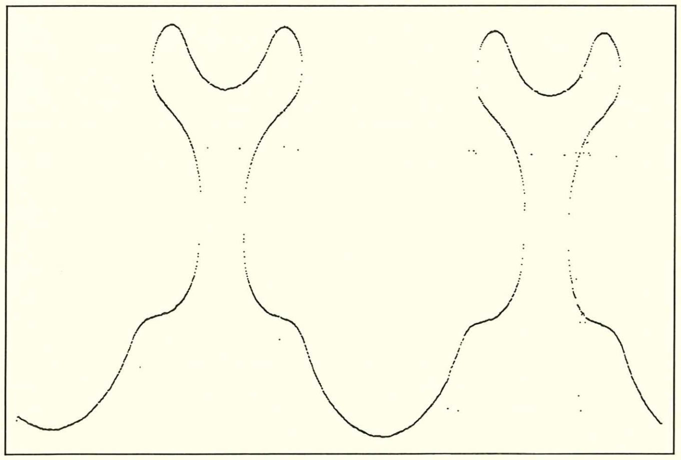


Figure 1. A MSE-lens data map used to evaluate lens components. This map is the "end product" of the 4051-based system employing the GPIB interface described in this article.

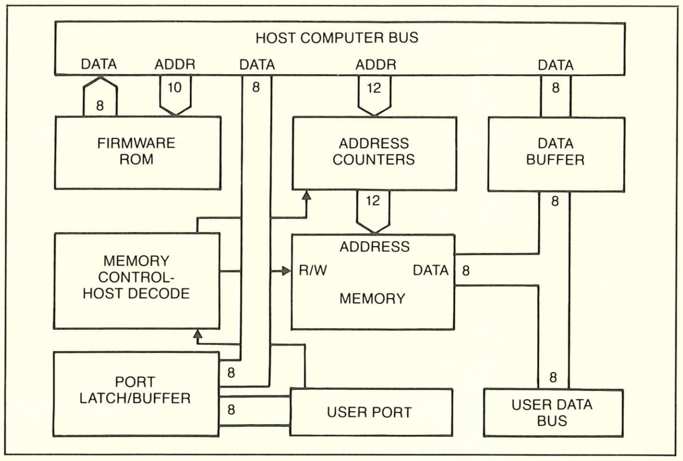
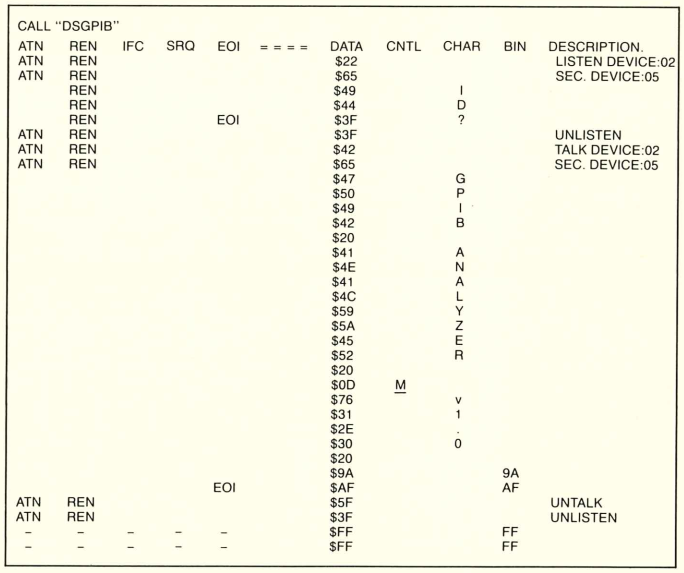


Figure 2. The Dualport Memory.



```
CALL “DSGPIB”
ATN     REN     IFC   SRQ   EOI    = = = =   DATA    CNTL   CHAR   BIN   DESCRIPTION.
ATN     REN                                  $22                         LISTEN DEVICE:02
ATN     REN                                  $65                         SEC. DEVICE:05
        REN                                  $49             I
        REN                                  $44             D
        REN                 EOI              $3F             ?
ATN     REN                                  $3F                         UNLISTEN
ATN     REN                                  $42                         TALK DEVICE:02
ATN     REN                                  $65                         SEC. DEVICE:05
                                             $47             G
                                             $50             P
                                             $49             I
                                             $42             B
                                             $20
                                             $41             A
                                             $4E             N
                                             $41             A
                                             $4C             L
                                             $59             Y
                                             $5A             Z
                                             $45             E
                                             $52             R
                                             $20
                                             $0D      M
                                             $76             v
                                             $31             1
                                             $2E             .
                                             $30             0
                                             $20
                                             $9A                   9A
                            EOI              $AF                   AF
ATN     REN                                  $5F                         UNTALK
ATN     REN                                  $3F                         UNLISTEN
 -       -       -     -     -               $FF                   FF
 -       -       -     -     -               $FF                   FF
```

Figure 3. GPIB monitor showing the English format of GPIB messages logged and displayed by the 4051.

The user data I/O bus can transfer data at 8 MHz in either direction between the user's circuit and the 4096 8-bit words of the Dualport Memory.

The user control lines allow the user interface to control the memory's address-counter, clock, clear, and read/write lines. Output lines in the user interface are a 500 KHz clock (4051-MCP1), +5 volts, ground, and counter overflow.

The user output port provides four latched TTL lines. With these lines, the computer interface can control the user's circuit. A BASIC command is used to write data to this port from a string variable.

The user input port provides four TTL sense lines so that the computer interface can read the user's circuit status. The user input port also has four TTL sense lines to read the latched output port. A BASIC command is used to read the 8-bit data into a string variable from the computer.

The computer interface of the Dualport Memory is a fully buffered and self-contained module – it takes no memory capacity away from the 4051. The Dualport plugs into one firmware expansion slot in a 4051 system. When the Dualport Memory Pack is installed, its firmware, memory area, and port all overlay in a 8K word-bank switched memory in the 4051. The BASIC commands that control access to the memory and port are in the convenient form of call-name statements. 4096 words can be transferred into the string in about three seconds.

Commands

The Dualport Memory responds to these commands in BASIC:

CLMEM	– set all dualport memory to logic zero
SETMEM	– set all dualport memory to logic one
DSPMEM	– display all dualport memory in a readable form with addresses (1120 bytes/page)
WTMEN,A$	– put hex data from string variable into dualport memory
RDMEN,A$	– put hex data from dualport memory into string variable
CLPORT	– set port to logic zero
RDPORT,A$	– put the 8-bit word from user port into string variable
WTPORT,A$	– latch hex data from A$ into user port
SETADD,A$	– set memory address counter equal to data in string variable
DELALL	– displays firmware version message

Application

The Dualport Memory is used as a high-speed digital-data acquisition unit for the 4051.

I have also built a 6144-word version of the Dualport on a separate board. By changing a few items in the firmware, the board can be used in the 4052, 4054, 4052A, and 4054A – after recompiling listings to provide new bank addresses. In the 4052 and 4054, the Dualport Memory could be as large as 15,368 words because the bank switch area has been expanded. To build a Dualport for other systems, a GPIB, RS-232, or other standard interface could replace the 4050 series computer.

For More Information

For more information call Gary Spence, 627-6892. □

SIMPLE GPIB INTERFACE FOR USER PROGRAMMABLE TEST CIRCUITS

Gary Spence, MOS Special Products, ICE

GPIB interfaces for test equipment are usually complex and almost always require a microprocessor. This article describes a small (2.6 × 4.4 inch) interface board that installs in a TM500 or TM 5000 plug-in. This simple interface couples directly to the GPIB, or to the GPIB through a 20-pin connector. Either method provides 8-bit data I/O and status to the user board in the plug-in.

We use this interface in three plug-ins that are part of a fully programmable charge-coupled device evaluation station (M800).

Applications

Many tests require some programmability for such tasks as changing a voltage, transferring data, switching a contact, or lighting a light. If a test can be completed by sending or receiving just a few bytes over a IEEE-488 bus (GPIB), this interface should work as a stand-alone unit. If large amounts of data are to be transfered, this interface with another four to six ICs on the user's board can transfer data at 2 MHz, providing EOI with the last byte.

Interface Capabilities

The interface (figure 1) is designed to the IEEE-488 standard. It supports subsets C0, E1, T0, TE0, T1, TE1, T5, TE5, L0, L3, SR1, PP1, DC1, DC2, RL1. These subsets may be selected by mode-select jumpers on the board. Two handshake-speed choices are provided in most modes, 2 MHz and 50 kHz.

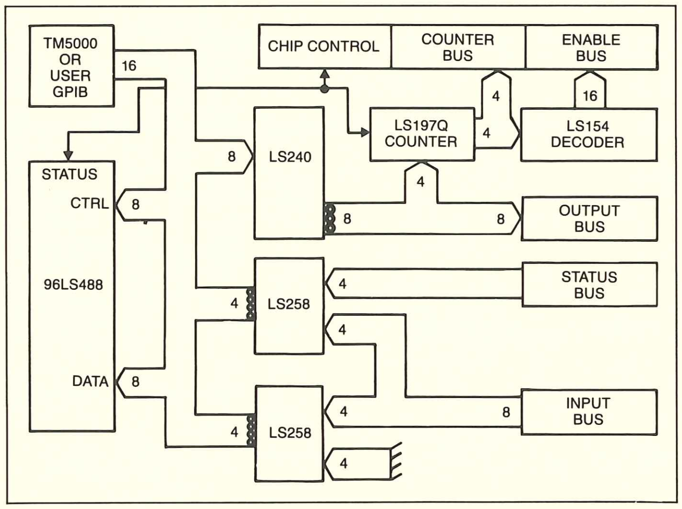


Figure 1. The simple GPIB interface for user-programmable test circuits. Several of these interfaces are used in a fully programmable evaluation station for charge-coupled devices.

The output bus provides the user circuit with tristateable, buffered, true-logic data. This data are sent from the controller containing synchronizing pulses. The user may control the buffer's output with either LADS, or with the user's circuit.

Data strobes and clock edges for externally latching data are provided to the user's circuit. The acceptor handshake may be delayed by the user's circuit.

The input bus allows the user's circuit to send 8-bit buffered true-logic data to the controller. The user's circuit may delay the source handshake to synchronize the controller (on the IEEE-488 bus).

A source acknowledge strobe is sent to the user's circuit when data is received by the controller.

The status bus enables the user's circuit to report its logical status. Four lines have pull-down resistors so unused lines can report status "zero" when serial polled on the IEEE-488 bus.

A status acknowledge strobe from controller enables more than one status byte to be sent. The status byte may be delayed by the user's circuit if desired.

The counter output bus is the 4-bit Q-latches of a 74LS197 counter. The Q-latches can be selected to store the lower 4-bits of the output bus. The latches are set to logic lows by either device clear, selected device clear, LADS going false – or by the user's circuit. The latches are incremented by the quantities of data bytes sent to the output bus or with each GET command on the IEEE-488 bus.

The enable-output bus provides the user's circuit a decoded 1 of 16 from the counter bus. This bus can be strobed during valid data, held enabled-always to select a relay, or enabled by the user's circuit.

The auxiliary control bus, itself, is an interface with the Fairchild 96LS488 integrated circuit. The auxilary control bus provides extended IEEE-488 bus state status and chip control to the user's circuit. To allow an intelligent instrument to communicate with the IEEE-488 bus with very little software overhead, a microprocessor may be used as an interface to this control bus.

Because the operating temperature range of the Fairchild 96LS488 chip is limited, the chip should be used only in a laboratory environment (20° to 50°C). Fairchild plans to increase the temperature range soon.

Interface Power

The only power needed is +5 VDC at 200 mA. This is usually available from either the user circuit board mounted in the TM500 or TM 5000 power connector, or a three-terminal regulator can be mounted in the plug-in that regulates the +8 VDC supply in the mainframe.

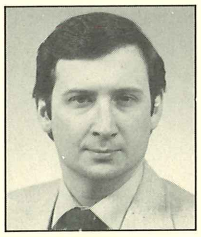
For More Information

For more information call Gary Spence, 627-6892. □

References:

Data sheet 156-1666-00
Fairchild Application Note 351
Fairchild Data Sheet 96LS488

SOLDERMASKS CAN IMPROVE CIRCUIT BOARD PERFORMANCE



Bob Forman is a chemical engineer in Manufacturing Engineering in Forest Grove. He joined Tek in July 1979. Previously he was employed at Omark Industries, Portland, and Bechtel Power Corp., San Francisco. He graduated from the University of Oregon 1975 with a BS in Chemistry.

The Forest Grove plant (F1) must do one thing very well – serve its customers. To do this two things are fundamental: that we know what our customers need and that we reduce cost and delivery times.

It is all too easy to think we know our customers' needs; after all, we've been building boards for years, in Beaverton and now in Forest Grove. "Knowing" our customers as we did, we could change processes to increase productivity and reduce costs. We did just that when we moved to Forest Grove, employing the improved tools and materials available in our state-of-the-art facility.

But after making these changes, we discovered that our customers, the business units, had mixed reactions to the results. Changing processes had changed important characteristics of their products. Discovering these unexpected problems lead us to dig into the underlying assumptions of soldermask performance. In discovering and solving these problems, we learned something every entrepreneur discovers: you can never relax, secure in your "knowledge" of customer needs.

More Than Just a Soldermask

Plastic polymeric coatings are placed on etched circuit boards for a variety of reasons. Originally, these coatings had only one purpose, to mask against solder. That's why they are called soldermasked coatings, or just "soldermasks." Although the original mask function related to only soldering operations, the mask can do much more – when properly formulated and when properly applied.

Because masks are applied over circuitry, they have side effects – some good, some bad – on the electrical and mechanical performance of circuitry. As product performance increased with time, soldermask side effects come more into play. Now the soldermask had to be considered not only as a solder barrier, but also as (1) an electrical component, (2) an environmental barrier, and (3) a mechanical encapsulant.

Because Tektronix soldermasks are now more than just solder barriers, Forest Grove needed to know the properties and electrical and mechanical performance of almost twenty different commercial coatings that can be used for masks, and – most important – know our customers' performance expectations and what we can do to satisfy them.

In selecting mask coatings, several performance criteria are crucial. These criteria have evolved as our customers' circuitry has become more sophisticated. We identified these needs by surveying the business units. These needs are described next.

Performance F1 Customers Want in a Soldermask

The mask must work as:

1. An effective solder barrier with these characteristics:
   - Good adhesion
   - Good appearance
   - Resistance to solvents and fluxes
   - Prevent reflow of plated tin under the mask
   - Prevent solder bridge between close conductors
2. The mask must have predictable electrical characteristics:
   - High leakage resistance in adverse environments
   - Low hook
   - Predictable capacitance
3. The mask must be a good encapsulant and provide the finished circuit board with:
   - A good humidity barrier
   - Immunity to surface contaminants after wave solder and wash
   - Good conductor protection from handling damage

What F1 Wants From a Soldermask

For manufacturing efficiency, a mask must have certain characteristics:

- Meet our customer's needs
- Screenable
- UV or quick curable
- Not noxious
- Reasonable process control
- Durable
- Strippable before cure
- U.L. approved for F1 processes
- Good thixotropic properties for screening
- Hot-air-level compatibility

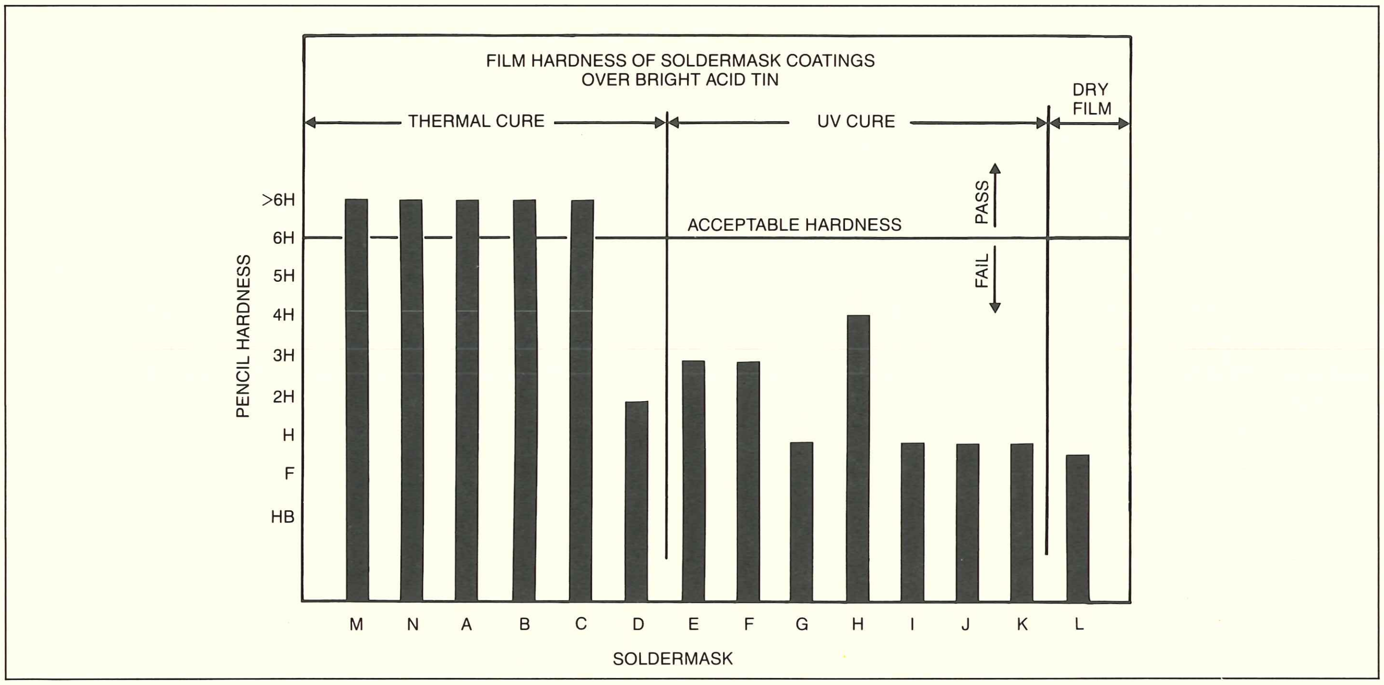


Figure 1.

Note: Rather than using brand names, we chose to identify soldermask coatings with letters in all figures and tables. Brand names are available on request.

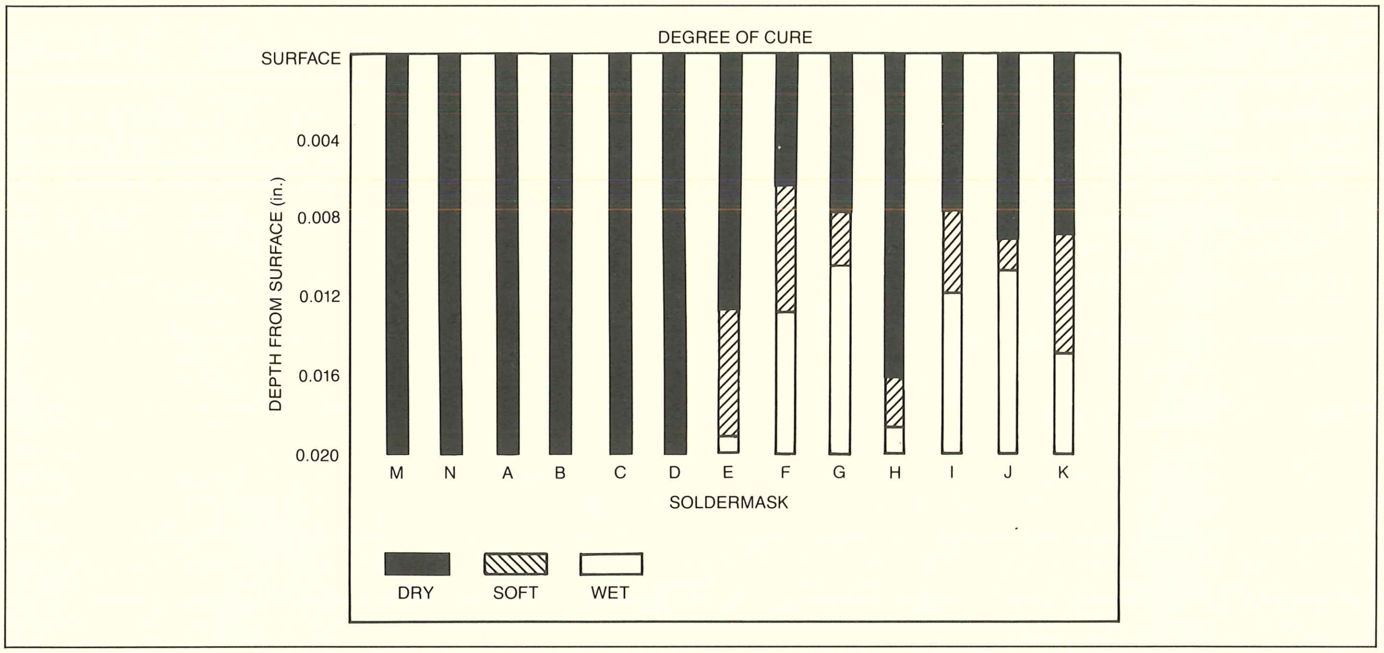


Figure 2.

Three Soldermask Choices

We offer three distinct types of soldermasks:

Thermal Cure – Masks applied as wet films by screening printing techniques to obtain the desired image. These masks are typically solvent and epoxy mixtures formulated to cure in batch ovens.

UV Cure – Masks applied as wet films by screening and printing techniques and cured by exposure to ultraviolet light. These masks are acrylic coatings with no solvent content.

Dry Film – Masks applied as a dry film using photo-imaging techniques. The films are cured by exposure to ultraviolet light followed by batch-oven baking.

Each soldermask has distinct and different manufacturing advantages. For example, *UV masks* allow greater manufacturing productivity because they cure easily and fast. *Dry-film* masks can cover finely detailed circuitry; this circuitry is difficult to cover using the screening methods in thermal-cure processes. The *thermal-cure* processes were the earliest used in circuit-board manufacturing. They have the widest use and acceptance in industry. They are the "conventional" soldermasks.

Each mask type is formulated differently, and thus can be expected to perform differently. You should know how they differ when you select a soldermask for your own circuit boards.

Testing

The performance of a soldermask – and its suitability for use on Tektronix boards – can be determined by subjecting the mask to tests that characterize it against the customer and manufacturing criteria listed earlier. We designed an experiment to do just that. The experiment tested six thermal-cure masks, seven UV-cure masks and one dry-film mask. We tested the masks of nine vendors. All of the masks fit the manufacturing processes at F1.

We characterized masks as solder barriers, measured electrical characteristics, and tested encapsulating properties using mechanical, chemical, environmental, and electrical processes. Several of the tests were designed to indicate which masks had characteristics that our customers wanted. The details of some of the tests are given next. Complete data is available on request.

Several of the tests performed quickly demonstrated the vast difference between the three different types of masks available. For instance, figures 1 and 2 illustrate the differences in film hardness and degree of cure among the various masks available at F1. A hard soldermask (⩾6H pencil) is important for two reasons: It protects the surface circuitry from damage and provides an impermeable humidity barrier that ensures proper electrical performance in adverse environments. The degree of soldermask cure indicates how thoroughly a film converts from a liquid to solid state after application to the board. For many UV masks, proper curing depends on deposit thickness; soldermasks that cannot dry thoroughly (more than .008 inches thick) won't survive adverse environments.

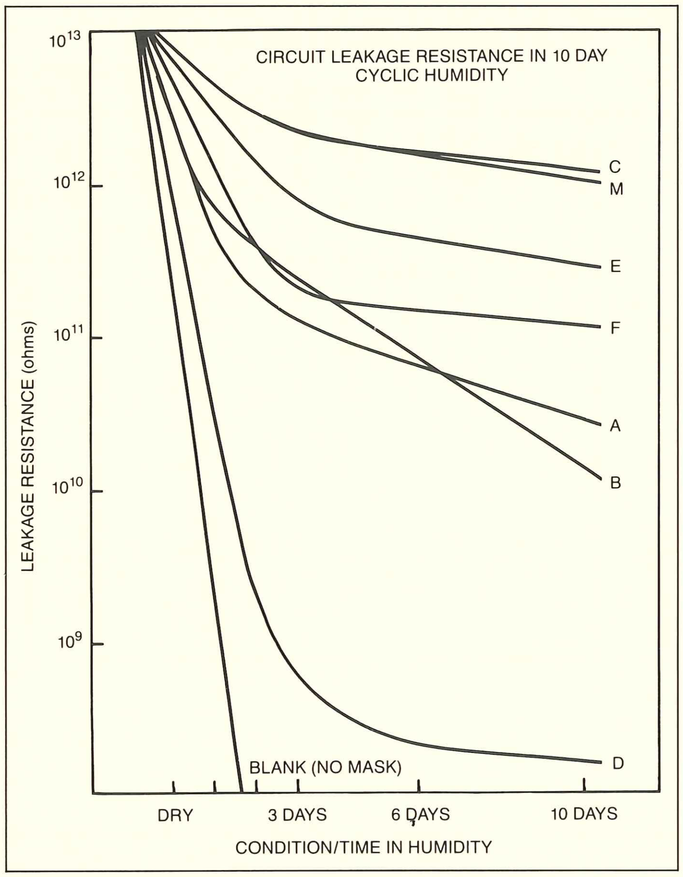


Figure 3.

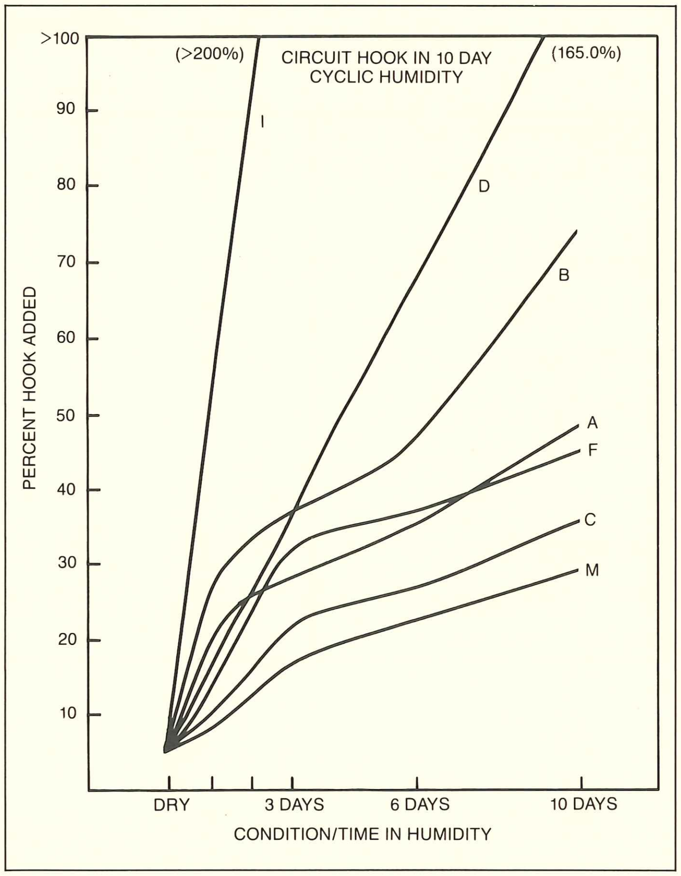


Figure 4.

After 10 days in humidity, we find a great range in the electrical performance of the masks. Figures 3 and 4 depict how the leakage resistance and hook of several masks changed during humidity testing.

The electrical test method is simple but time consuming; it employs Cal Diller's (Portables Engineering) dielectric analyzer and a controlled humidity chamber. (See "When the ECB Itself Is Part of the Circuit," *Technology Report,* Nov. 1983.)

In the circuit leakage test, the degradation of the resistance of a sample board is monitored while the sample is subjected to 10 days of changing temperature and humidity. The mask plays the role of protecting the circuitry by resisting extensive water absorption and contamination "creep" (conductive path growth due to surface and trapped contaminates).

Circuit hook is also impacted by what soldermask covers the board. We measured hook at the same time we measured leakage, using the Diller technique. Figure 4 illustrates the results. (Hook is discussed in the TR Article.).

In addition to the tests above, which evaluated masks for customer needs, we evaluated masks for suitability in our manufacturing process. In some cases, subjective analysis of the results was necessary because quantitative measurements were impossible.

For instance, screenability is a quality of soldermasks that is hard to rate but important in production. We had experienced production technicians rate screenability. Their ratings ranged from "good" to "terrible." These ratings were considered when compiling the Performance Table.

The Performance Table summarizes the qualitative results from our soldermask coating tests and evaluations. For non-electrical tests, numerical scores were assigned: pass = 1 point, fail = 0 points. This yielded subtotal A. Electrical tests were ranked from best to worst performance, and assigned numbers from 11 to 1 accordingly. This yielded subtotal B. When both subtotals are combined, a grand total is created which ranks the performances of each mask with the best having the highest score and the worst the lowest score.

(We chose to letter coatings rather than use actual brand names in the table. Brand names are available on request.)

Conclusions

Because of our tests and evaluations, we feel better qualified to understand circuit board performance from the customer's perspective. It is our intent to continue to add to this understanding – and to help you choose what's best for you.

For further information call Bob Forman 640-2288, ext. 4248 (F1-395). □

PERFORMANCE TABLE

	THERMAL						ULTRAVIOLET							DRY FILM
PARTIAL PERFORMANCE MATRIX	**M**	**N**	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**
Film Hardness	P	P	P	P	P	F	P	F	F	P	F	F	F	F
Depth of Cure	P	P	P	P	P	P	P	F	F	P	F	F	F	NA
Chemical Resistance	P	P	P	P	P	P	P	P	P	P	P	F	P	F
U.L. Flammability	P	P	P	P	P	P	P	P	P	P	P	P	P	F
Screenability	P	F	P	P	P	P	P	P	P	P	P	P	F	NA
Health Effects & Odor	P	P	P	P	P	F	P	P	P	P	P	P	P	P
Hot Air Level Compatibility	P	P	P	P	P	F	F	F	F	F	F	F	F	NA
Wicking, Blistering	P	P	P	P	F	P	F	F	F	F	F	F	F	NA
Bleedout, Smearing	P	F	F	F	P	P	P	P	P	P	P	P	P	P
Cleanliness (Omega Meter)	P	P	P	P	P	P	P	P	P	P	F	P	F	F
SUBTOTAL A	10	8	9	9	9	7	8	6	6	8	5	5	4	
Leakage Resistance	11	8	6	5	10	1	9	7	3	2	4	NA	NA	NA
Capacitance	11	6	8	9	7	5	4	2	3	10	1			
Hook	11	4	8	5	10	2	6	9	3	7	1			
SUBTOTAL B	33	18	22	19	27	8	19	18	9	19	8			
GRAND TOTALS	**43**	**26**	**31**	**28**	**36**	**15**	**27**	**24**	**15**	**27**	**11**			

Technology Report

MAILING LIST COUPON

☐ ADD

☐ REMOVE

Not available to field offices or outside the U.S.

MAIL COUPON TO 53-077

Name: ______________________ D.S.: ________

Payroll Code: ______________________

(Required for the mailing list)

For change of delivery station, use a directory change form.

YOU GAVE A PAPER, BUT . . .

Almost every working day, a Tektronix employee gives a paper or makes a presentation somewhere in the world. Almost every month, five to twenty magazines and proceedings publish pieces by Tek authors. Most – but not all – of these are reported in *Papers and Presentations:* We miss some each month; perhaps your effort was among the missing.

Why? We just did not know.

While the absence of this information doesn't burden us, our ignorance means you won't get the recognition you've earned, and the engineering community *inside* Tek may not gain from your experiences and knowledge.

One way to make sure we'll be informed is to work with TCS (Technology Communication Support, 642-8924, d.s. 53-077) in preparing your paper. Or, drop them a note listing:

- Title
- Your name, the names of co-authors, and your organization
- Publication or conference
- Date of presentation or publication

We will do the rest. □

COMPANY CONFIDENTIAL
NOT AVAILABLE TO FIELD OFFICES

TECHNOLOGY REPORT
RICHARD E CORNWELL
19-071

DO NOT FORWARD

Tektronix, Inc. is an equal opportunity employer.