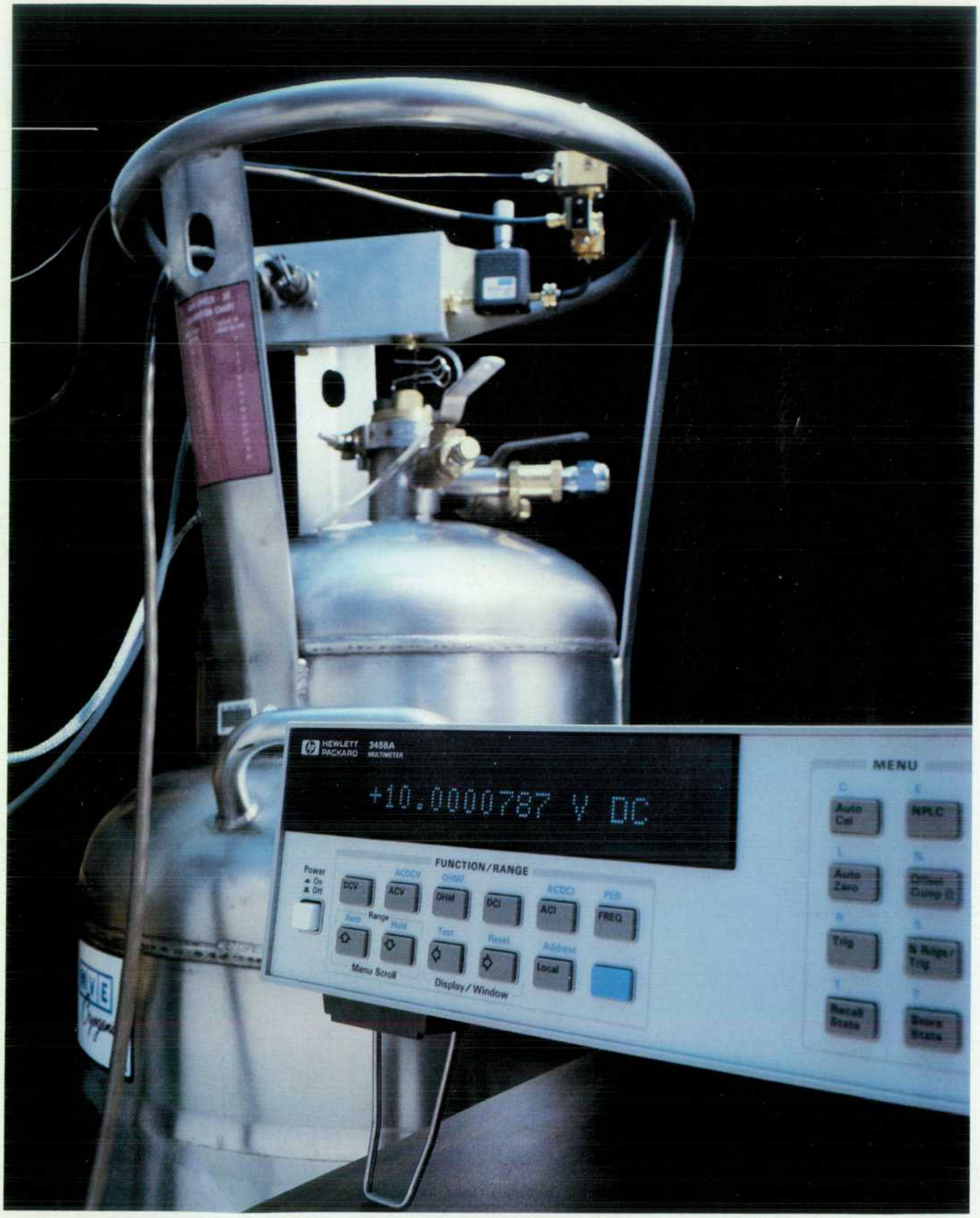
HEWLETT-PACKARD JOURNAL

APRIL 1989



HEWLETT PACKARD

HEWLETT-PACKARD JOURNAL

April 1989 Volume 40 • Number 2

Articles

6 **An 8½-Digit Digital Multimeter Capable of 100,000 Readings per Second and Two-Source Calibration,** *by Scott D. Stever*

8 **An 8½-Digit Integrating Analog-to-Digital Converter with 16-Bit, 100,000-Sample-per-Second Performance,** *by Wayne C. Goeke*

15 **Precision AC Voltage Measurements Using Digital Sampling Techniques,** *by Ronald L. Swerlein*

22 **Calibration of an 8½-Digit Multimeter from Only Two External Standards,** *by Wayne C. Goeke, Ronald L. Swerlein, Stephen B. Venzke, and Scott D. Stever*

24 **Josephson Junction Arrays**
28 **A High-Stability Voltage Reference**

31 **Design for High Throughput in a System Digital Multimeter,** *by Gary A. Ceely and David J. Rustici*

33 **Firmware Development System**
36 **Custom UART Design**

39 **High-Resolution Digitizing Techniques with an Integrating Digital Multimeter,** *by David A. Czenkusch*

42 **Time Interpolation**
46 **Measurement of Capacitor Dissipation Factor Using Digitizing**

50 **A Structural Approach to Software Defect Analysis,** *by Takeshi Nakajo, Katsuhiko Sasabuchi, and Tadashi Akiyama*

57 **Dissecting Software Failures,** *by Robert B. Grady*

62 **Defect Origins and Types**

Editor, Richard P. Dolan • Associate Editor, Charles L. Leath • Assistant Editor, Hans A. Toepfer • Art Director, Photographer, Arvid A. Danielson
Support Supervisor, Susan E. Wright • Administrative Services, Typography, Anne S. LoPresti • European Production Supervisor, Michael Zandwijken

64 **Software Defect Prevention Using McCabe's Complexity Metric,** *by William T. Ward*

66 **The Cyclomatic Complexity Metric**

69 **Object-Oriented Unit Testing,** *by Steven P. Fiedler*

75 **Validation and Further Application of Software Reliability Growth Models,** *by Gregory A. Kruger*

80 **Comparing Structured and Unstructured Methodologies in Firmware Development,** *by William A. Fischer, Jr. and James W. Jost*

86 **An Object-Oriented Methodology for Systems Analysis and Specification,** *by Barry D. Kurtz, Donna Ho, and Teresa A. Wall*

91 **VXIbus: A New Interconnection Standard for Modular Instruments,** *by Kenneth Jessen*

96 **VXIbus Product Development Tools,** *by Kenneth Jessen*

Departments

4 **In this Issue**
5 **Cover**
5 **What's Ahead**
56 **Corrections**
98 **Authors**

The **Hewlett-Packard Journal** is published bimonthly by the Hewlett-Packard Company to recognize technical contributions made by Hewlett-Packard (HP) personnel. While the information found in this publication is believed to be accurate, the Hewlett-Packard Company makes no warranties, express or implied, as to the accuracy or reliability of such information. The Hewlett-Packard Company disclaims all warranties of merchantability and fitness for a particular purpose and all obligations and liabilities for damages, including but not limited to indirect, special, or consequential damages, attorney's and expert's fees, and court costs, arising out of or in connection with this publication.

Subscriptions: The Hewlett-Packard Journal is distributed free of charge to HP research, design, and manufacturing engineering personnel, as well as to qualified non-HP individuals, libraries, and educational institutions. Please address subscription or change of address requests on printed letterhead (or include a business card) to the HP address on the back cover that is closest to you. When submitting a change of address, please include your zip or postal code and a copy of your old label.

Submissions: Although articles in the Hewlett-Packard Journal are primarily authored by HP employees, articles from non-HP authors dealing with HP-related research or solutions to technical problems made possible by using HP equipment are also considered for publication. Please contact the Editor before submitting such articles. Also, the Hewlett-Packard Journal encourages technical discussions of the topics presented in recent articles and may publish letters expected to be of interest to readers. Letters should be brief, and are subject to editing by HP.

Please address inquiries, submissions, and requests to: Editor, Hewlett-Packard Journal, 3200 Hillview Avenue, Palo Alto, CA 94304, U.S.A.

In this Issue

If you thought that voltmeters, those ancient and fundamental instruments, had evolved about as far as possible and that there couldn't be much more to say about them, you'd be wrong. Today, they're usually called digital multimeters or DMMs rather than voltmeters. You can find highly accurate ones in calibration laboratories, very fast ones in automated test systems, and a whole spectrum of performance levels and applications in between these extremes. Generally, more speed means less resolution—that is, fewer digits in the measurement result. Conversely, a higher-resolution measurement generally takes longer. Some DMMs are capable of a range of speeds and resolutions and allow the user to trade one for the other. The HP 3458A Digital Multimeter does this to an unprecedented degree. It can make 100,000 4½-digit measurements per second or six 8½-digit measurements per second, and allows the user an almost continuous selection of speed-versus-resolution trade-offs between these limits. You'll find an introduction to the HP 3458A on page 6. The basis of its performance is a state-of-the-art integrating analog-to-digital converter (ADC) that uses both multislope runup and multislope rundown along with a two-input structure to achieve both high speed and high precision (page 8). So precise is this ADC that it can function as a ratio transfer device for calibration purposes. With the ADC and a trio of built-in transfer standards, all of the functions and ranges of the HP 3458A can be calibrated using only two external traceable standards—10V and 10 kΩ. The article on page 22 explains how this is possible. At the high end of its speed range, the ADC allows the HP 3458A to function as a high-speed digitizer, an unusual role for a DMM (page 39). In fact, ac voltage measurements are made by digitizing the input signal and computing its rms value, eliminating the analog rms-to-dc converters of older designs (page 15). Finally, moving data fast enough to keep up with the ADC was a design challenge in itself. How it was met with a combination of high-speed circuits and efficient firmware is detailed in the article on page 31.

The seven papers on pages 50 to 90 are from the 1988 HP Software Engineering Productivity Conference and should be of interest to software engineers and users concerned with software defect prevention. Collectively, the papers spotlight areas where vigorous software engineering activity is occurring today, namely in structured and object-oriented analysis, design, and testing, and in the development of reliable metrics with which to measure software quality. ▶ In the paper on page 50, engineers from Yokogawa Hewlett-Packard and Tokyo University describe a joint effort to find the flaws in design procedures that increase the likelihood of human errors that result in program faults. Working backwards from faults to human errors to flawed procedures, they propose various structured analysis and design solutions to eliminate the flaws. ▶ The paper on page 57 urges expansion of the software defect data collection process so that project managers can not only determine how best to prevent future defects, but also build a case for making the necessary changes in procedures. The time required to collect and analyze the additional data is shown to be minimal. ▶ That complexity leads to defects is well-established, so monitoring the complexity of software modules during implementation should point out modules that will be defect prone. The paper on page 64 tells how HP's Waltham Division is taking this approach to improve the software development process, using McCabe's cyclomatic complexity metric to measure complexity. ▶ Object-oriented programming, originally conceived for artificial intelligence applications, is now finding wider acceptance. The paper on page 69 reports on problems and methods associated with testing software modules developed with an object-oriented language, C++, for a clinical information system. ▶ In the paper on page 75, Greg Kruger updates his June 1988 paper on the use of a software reliability growth model at HP's Lake Stevens Instrument Division.

The model has generally been successful, but there are pitfalls to be avoided in applying it. ▶ On a software project at HP's Logic Systems Division, some of the engineers used structured methods and some used unstructured methods. Was structured design better? According to the paper on page 80, the results were mixed, but the structured methods offered enough benefits to justify their continued use. ▶ In software development, system analysis precedes design. The paper on page 86 describes a new object-oriented approach suitable for analyzing today's increasingly larger and more complex systems. The authors believe that designs based on object-oriented specifications can be procedure-oriented or object-oriented with equal success.

Modular measurement systems consist of instruments on cards that plug into a card cage or mainframe. A user can tailor a system to an application by plugging the right modules into the mainframe. The VXIbus is a new industry standard for such systems. Modules and mainframes conforming to the VXIbus standard are compatible no matter what company manufactured them. The articles on pages 91 and 96 introduce the VXIbus and some new HP products that help manufacturers develop VXIbus modules more quickly. HP's own modular measurement system architecture conforms to the VXIbus standard where applicable. However, for high-performance RF and microwave instrumentation, HP has used a proprietary modular system interface bus (HP-MSIB). Patent rights to the HP-MSIB have now been assigned to the public so that this architecture can be used by everyone as the high-frequency counterpart of the VXIbus.

R.P. Dolan
Editor

Cover

So precise is the 3458A Digital Multimeter that verifying some aspects of its performance is beyond the limits of conventional standards. In the HP Loveland Instrument Division Standards Laboratory, the HP 3458A's linearity is measured using a 10-volt Josephson junction array developed by the U.S. National Institute of Standards and Technology. The array is in a specially magnetically shielded cryoprobe in the center of a liquid-helium-filled dewar (the tank with the protective "steering wheel.") On top of the dewar are a Gunn-diode signal source (72 GHz) and various microwave components. A waveguide and voltage and sense leads connect the array to the external components. For more details see page 24.

What's Ahead

Subjects to be covered in the June issue include:

- The HP 9000 Model 835 and HP 3000 Series 935 Midrange HP Precision Architecture Computers
- Programming with neurons
- A new 2D simulation model for electromigration in thin metal films
- Data compression and blocking in the HP 7980XC Tape Drive
- Design and applications of HP 8702A Lightwave Component Analyzer systems
- A data base for real-time applications and environments
- A hardware/software tool to automate testing of software for the HP Vectra Personal Computer.

An 8½-Digit Digital Multimeter Capable of 100,000 Readings per Second and Two-Source Calibration

A highly linear and extremely flexible analog-to-digital converter and a state-of-the-art design give this DMM new performance and measurement capabilities for automated test, calibration laboratory, or R&D applications.

by Scott D. Stever

THE DIGITAL MULTIMETER OR DMM is among the most common and most versatile instruments available for low-frequency and dc measurements in automated test, calibration laboratory, and bench R&D applications. The use of general-purpose instrumentation in automated measurement systems has steadily grown over the past decade. While early users of programmable instruments were elated to be able to automate costly, tedious, error-prone measurements or characterization processes, the sophistication and needs of today's users are orders of magnitude greater. The computing power of instrument controllers has increased manyfold since the mid-1970s and so have user expectations for the performance of measurement systems. Test efficiency in many applications is no longer limited by the device under test or the instrument controller's horsepower. Often either the configuration speed or the measurement speed of the test instrumentation has become the limiting factor for achieving greater test throughput. In many systems, the DMM is required to perform hundreds of measurements and be capable of multiple functions with various resolutions and accuracies.

In some applications, several DMMs may be required to characterize a single device. For example, measurements requiring high precision may need a slower DMM with calibration laboratory performance. Usually, the majority of measurements can be satisfied by the faster, moderate-resolution capabilities of a traditional system DMM. In extreme cases, where speed or sample timing are critical to the application, a lower-resolution high-speed DMM may be required. A single digital multimeter capable of fulfilling this broad range of measurement capabilities can reduce system complexity and development costs. If it also provides shorter reconfiguration time and increased measurement speed, test throughput can also be improved for automated test applications.

The HP 3458A Digital Multimeter (Fig. 1) was developed to address the increasing requirements for flexible, accurate, and cost-effective solutions in today's automated test applications. The product concept centers upon the synergistic application of state-of-the-art technologies to meet these needs. While it is tuned for high throughput in computer-aided testing, the HP 3458A also offers calibration laboratory accuracy in dc volts, ac volts, and resistance. Owners can trade speed for resolution, from 100,000 measurements per second with 4½-digit (16-bit) resolution to six measurements per second with 8½-digit resolution. At 5½-digit resolution, the DMM achieves 50,000 readings per second. To maximize the measurement speed for the resolution selected, the integration time is selectable from 500 nanoseconds to one second in 100-ns steps. The effect is an almost continuous range of speed-versus-resolution trade-offs.

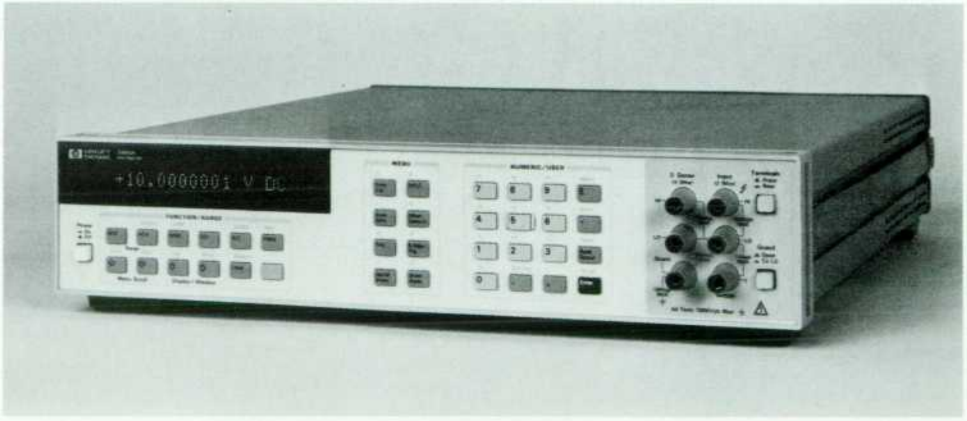


Fig. 1. *The HP 3458A Digital Multimeter can make 100,000 4½-digit readings per second for high-speed automated test applications. For calibration laboratory applications, it can make six 8½-digit readings per second. Fine control of the integration aperture allows a nearly continuous range of speed-versus-resoultion trade-offs.*

Measurement Capabilities

Measurement ranges for the HP 3458A's functions are:

- Voltage: 10 nV to 1000V dc
  <1 mV to 700V rms ac
- Current: 1 pA to 1A dc
  100 pA to 1A rms ac
- Resistance: 10 $\mu\Omega$ to 1 GΩ, 2-wire or 4-wire
- Frequency: 1 Hz to 10 MHz
- Period: 100 ns to 1 s
- 16-bit digitizing at effective sample rates to 100 megasamples/second.

The ac voltage bandwidth is 1 Hz to 10 MHz, either ac or dc coupled.

To increase uptime, the HP 3458A is capable of two-source electronic calibration and self-verifying autocalibration. Autocalibration enhances accuracy by eliminating drift errors with time and temperature. The dc voltage stability is specified at eight parts per million over one year, or 4 ppm with the high-stability option. Linearity is specified at 0.1 ppm, transfer accuracy at 0.1 ppm, and rms internal noise at 0.01 ppm. Maximum accuracies are 0.5 ppm for 24 hours in dc volts and 100 ppm in ac volts. Midrange resistance and direct current accuracies are 3 ppm and 10 ppm, respectively.

The HP 3458A can transfer 16-bit readings to an HP 9000 Series 200 or 300 Computer via the HP-IB (IEEE 488, IEC 625) at 100,000 readings per second. It can change functions or ranges and deliver a measurement 200 times per second (over 300/s from the internal program memory), about four times faster than any earlier HP multimeter.

The following five articles describe the technologies required to achieve this performance and the benefits that result. In the first paper, the development of a single analog-to-digital converter capable of both high resolution and high speed is discussed. The second paper describes the development of a technique for the precise measurement of rms ac voltages. The application of these technologies to provide improved measurement accuracy over extended operating conditions and to provide complete calibration of the DMM from only two external traceable sources (10V dc, 10 kΩ) is discussed in the third article. Hardware and firmware design to achieve increased measurement throughput is the topic of the fourth paper. The final paper discusses several applications for the HP 3458A's ability to perform high-resolution, high-speed digitizing.

An 8½-Digit Integrating Analog-to-Digital Converter with 16-Bit, 100,000-Sample-per-Second Performance

This integrating-type ADC uses multislope runup, multislope rundown, and a two-input structure to achieve the required speed, resolution, and linearity.

by Wayne C. Goeke

THE ANALOG-TO-DIGITAL CONVERTER (ADC) design for the HP 3458A Digital Multimeter was driven by the state-of-the-art requirements for the system design. For example, autocalibration required an ADC with 8½-digit (28-bit) resolution and 7½-digit (25-bit) integral linearity, and the digital ac technique (see article, page 22) required an ADC capable of making 50,000 readings per second with 18-bit resolution.

Integrating ADCs have always been known for their ability to make high-resolution measurements, but tend to be relatively slow. When the HP 3458A's design was started, the fastest integrating ADC known was in the HP 3456A DMM. This ADC uses a technique known as multislope and is capable of making 330 readings per second. The HP 3458A's ADC uses an enhanced implementation of the same multislope technique to achieve a range of speeds and resolutions never before achieved—from 16-bit resolution at 100,000 readings per second to 28-bit resolution at six readings per second. In addition to high resolution, the ADC has high integral linearity—deviations are less than 0.1 ppm (parts per million) of input.

Multislope is a versatile ADC technique, allowing speed to be traded off for resolution within a single circuit. It is easier to understand multislope by first understanding its predecessor, dual-slope.

Basic Dual-Slope Theory

Dual-slope is a simple integrating-type ADC algorithm. Fig. 1 shows a simple circuit for implementing a dual-slope ADC.

The algorithm starts with the integrator at zero volts. (This is achieved by shorting the integrator capacitor, C.) At time 0 the unknown input voltage V_{in} is applied to the resistor R by closing switch SW1 for a fixed length of time t_u. This portion of the algorithm, in which the unknown input is being integrated, is known as runup. At the end of runup (i.e., when SW1 is opened), the output of the integrator, V_o, can be shown to be

$$V_o\,(t_u) = -(1/RC)\int_0^{t_u} V_{in}(t)dt$$

or, when V_{in} is time invariant,

$$V_o(t_u) = -(1/RC)\,V_{in}t_u.$$

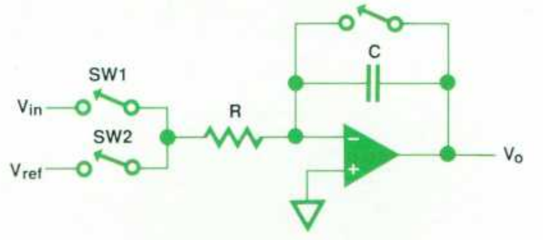
Next a known reference voltage V_{ref} with polarity opposite to that of V_{in} is connected to the same resistor R by closing SW2. A counter is started at this time and is stopped when the output of the integrator crosses through zero volts. This portion of the algorithm is known as rundown. The counter contents can be shown to be proportional to the unknown input.

$$V_o(t_2) = V_o(t_u) - (1/RC)V_{ref}t_d = 0,$$

where t_d is the time required to complete rundown (i.e., $t_d = t_2 - t_u$). Solving for V_{in},

$$V_{in} = -V_{ref}(t_d/t_u).$$

Letting N_u be the number of clock periods (T_{ck}) during



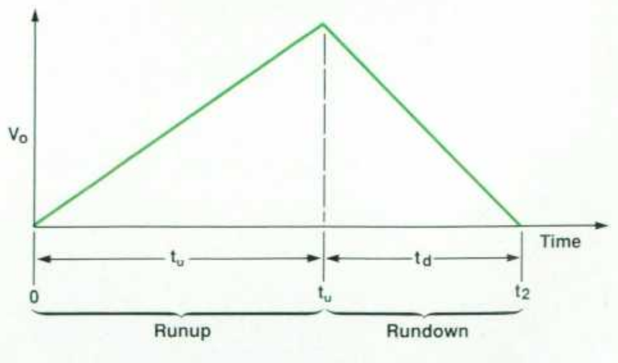


Fig. 1. *Dual-slope integrating ADC (analog-to-digital converter) circuit and a typical waveform.*

runup and N_d the number of clock periods during rundown, time cancels and

$$V_{in} = - V_{ref}(N_d/N_u).$$

The beauty of the dual-slope ADC technique is its insensitivity to the value of most of the circuit parameters. The values of R, C, and T_{ck} all cancel from the final equation. Another advantage of the dual-slope ADC is that a single circuit can be designed to trade speed for resolution. If the runup time is shortened, the resolution will be reduced, but so will the time required to make the measurement.

The problem with the dual-slope algorithm is that its resolution and speed are limited. The time T_m for a dual-slope ADC to make a measurement is determined by:

$$T_m = 2T_{ck}M,$$

where T_m is the minimum theoretical time to make a full-scale measurement, T_{ck} is the period of the ADC clock, and M is the number of counts of resolution in a full-scale measurement. Even with a clock frequency of 20 MHz, to measure a signal with a resolution of 10,000 counts requires at least 1 millisecond.

The resolution of the dual-slope ADC is limited by the wideband circuit noise and the maximum voltage swing of the integrator, about ±10 volts. The wideband circuit noise limits how precisely the zero crossing can be determined. Determining the zero crossing to much better than a millivolt becomes very difficult. Thus, dual-slope is limited in a practical sense to four or five digits of resolution (i.e., 10V/1 mV).

Enhanced Dual-Slope

The speed of the dual-slope ADC can be nearly doubled simply by using a pair of resistors, one for runup and the other for rundown, as shown in Fig. 2.

The unknown voltage, V_{in}, is connected to resistor R_u, which is much smaller than resistor R_d, which is used during rundown. This allows the runup time to be shortened by the ratio of the two resistors while maintaining the same resolution during rundown. The cost of the added speed is an additional resistor and a sensitivity to the ratio of the two resistors:

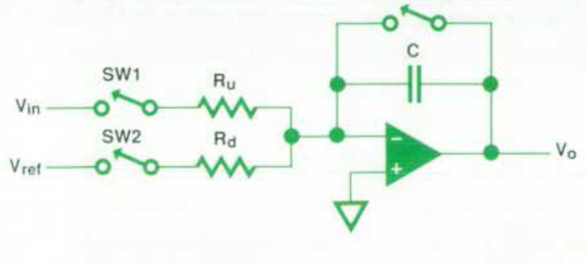
$$V_{in} = - V_{ref}(N_d/N_u)(R_u/R_d).$$

Because resistor networks can be produced with excellent ratio tracking characteristics, the enhancement is very feasible.

Multislope Rundown

Enhanced dual-slope reduces the time to perform runup. Multislope rundown reduces the time to perform rundown. Instead of using a single resistor (i.e., a single slope) to seek zero, multislope uses several resistors (i.e., multiple slopes) and seeks zero several times, each time more precisely. The ratio of one slope to another is a power of some number base, such as base 2 or base 10.

Fig. 3 shows a multislope circuit using base 10. Four slopes are used in this circuit, with weights of 1000, 100, 10, and 1. Each slope is given a name denoting its weight



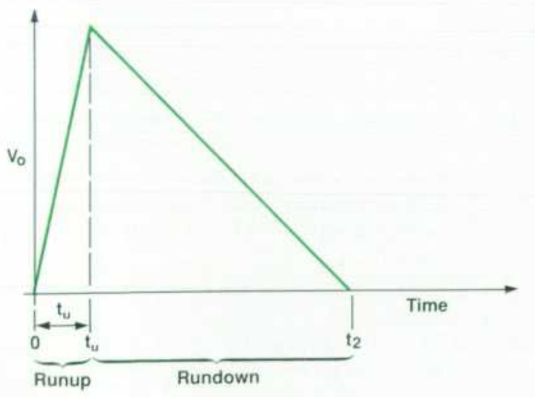
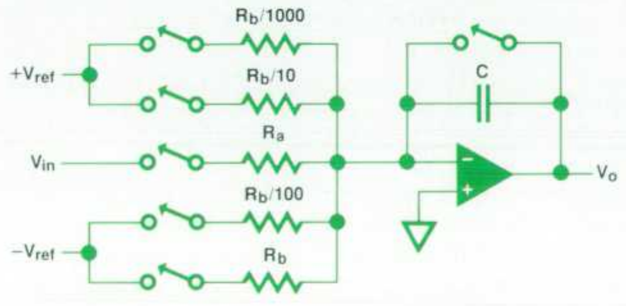


Fig. 2. *Enhanced dual-slope ADC circuit uses two resistors, one for runup and one for rundown.*



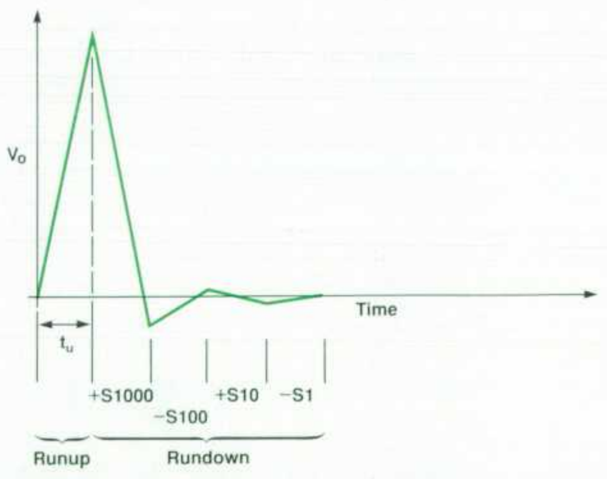


Fig. 3. *Base-10 multislope rundown circuit.*

and polarity. For example, S1000 is a positive slope worth 1000 counts per clock period and −S100 is a negative slope worth −100 counts per clock period. A slope is considered to be positive if it transfers charge into the integrator. This may be confusing because the integrator (an inverting circuit) actually moves in a negative direction during a positive slope and vice versa.

The multislope rundown begins by switching on the steepest slope, S1000. This slope remains on until the integrator output crosses zero, at which time it is turned off and the next slope, −S100, is turned on until the output crosses back through zero. The S10 slope follows next, and finally, the −S1 slope. Each slope determines the integrator's zero crossing ten times more precisely than the previous slope. This can be viewed as a process in which each slope adds another digit of resolution to the rundown.

If each slope is turned off within one clock period of crossing zero, then each subsequent slope should take ten or fewer clock periods to cross zero. Theoretically, then, the time t_d to complete a multislope rundown is:

$$t_d < NBT_{ck},$$

where N is the number of slopes and B is the number base of the slope ratios. In practice, the time to complete rundown is higher, because it isn't always possible to to turn off each slope within a clock period of its zero crossing. Delays in detecting the zero crossings and delays in responding by turning off the slopes cause the actual time to be:

$$t_d < kNBT_{ck},$$

where k is a factor greater than one. The delay in turning off a slope results in the integrator output's overshooting zero. For each clock period of overshoot, the following slope must take B clock periods to overcome the overshoot. Typical values of k range from two to four. The multislope rundown shown in Fig. 3 completes a measurement yielding 10,000 counts of resolution in 4.0 μs assuming a 20-MHz clock and k = 2. This is 125 times faster than the equivalent dual-slope rundown.

Multislope can be optimized for even faster measurements by choosing the optimum base. Noting that the number of slopes, N, can be written as $\log_B(M)$, where M is the number of counts of resolution required from rundown,

$$t_d < kB\log_B(M)T_{ck}.$$

This yields base e as the optimum base regardless of the required resolution. Using base e in the above example

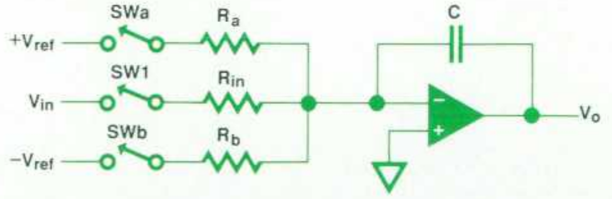


Fig. 4. *Multislope runup circuit.*

results in a rundown time of 2.5 μs. This is a 60% increase in multislope rundown speed as a result of using base e instead of base 10.

There is a cost associated with implementing multislope rundown. A resistor network must be produced with several resistors that have precise ratios. The tightest ratio tolerance is the reciprocal of the weight of the steepest slope and must be maintained to ensure linear ADC operation. If the ratio tolerances are no tighter than 0.05%, then this requirement is feasible. Multislope also requires a more complex circuit to control and accumulate the measurement, but with the reduced cost and increased density of digital circuits, this is also feasible.

Multislope Runup

Multislope runup is a modification of dual-slope runup with the purpose of increasing the resolution of the ADC. As mentioned earlier, the dual-slope technique's resolution is limited by the maximum voltage swing of the integrator and the wideband circuit noise. Multislope runup allows the ADC to have an effective voltage swing much larger than the physical limitations of the integrator circuit hardware.

The technique involves periodically adding and subtracting reference charge to or from the integrator during runup such that the charge from the unknown input plus the total reference charge is never large enough to saturate the integrator. By accounting for the total amount of reference charge transferred to the integrator during runup and adding this number to the result of rundown, a measurement can be made with much higher resolution. Fig. 4 shows a circuit for implementing multislope runup.

A precise amount of reference charge is generated by applying either a positive reference voltage to resistor R_a or a negative reference voltage to resistor R_b for a fixed amount of time. The following table shows the four possible runup reference currents using this circuit.

Slope Name	SWa	SWb	Integrator Direction	Current
S_+	$+V_{ref}$	0	↘	+I
S_{+0}	0	0	–	0
S_-	0	$-V_{ref}$	↗	−I
S_{-0}	$+V_{ref}$	$-V_{ref}$	–	0

Notice that, like multislope rundown, S_+ adds charge to the integrator and S_- subtracts charge from the integrator. If we design the S_+ and S_- currents to have equal magnitudes that are slightly greater than that of the current generated by a full-scale input signal, then the reference currents will always be able to remove the charge accumulating from the input signal. Therefore, the integrator can be kept from being saturated by periodically sensing the polarity of the integrator output and turning on either S_+ or S_- such that the integrator output is forced to move towards or across zero.

Fig. 5 shows a typical multislope runup waveform. The dashed line shows the effective voltage swing, that is, the voltage swing without reference charge being put into the integrator. The integrator output is staying within the limits of the circuit while the effective voltage swing ramps far beyond the limit. The HP 3458A has an effective voltage swing of ±120,000 volts when making 8½-digit readings, which means the rundown needs to resolve a millivolt to achieve an 8½-digit reading (i.e., 120,000V/0.001V = 120,000,000 counts).

The multislope runup algorithm has two advantages over dual-slope runup: (1) the runup can be continued for any length of time without saturating the integrator, and (2) resolution can be achieved during runup as well as during rundown. The HP 3458A resolves the first 4½ digits during runup and the final 4 digits during rundown to achieve an 8½-digit reading.

An important requirement for any ADC is that it be linear. With the algorithm described above, multislope runup would not be linear. This is because each switch transition transfers an unpredictable amount of charge into the integrator during the rise and fall times. Fig. 6 shows two waveforms that should result in the same amount of charge transferred to the integrator, but because of the different number of switch transitions, do not.

This problem can be overcome if each switch is operated a constant number of times for each reading, regardless of the input signal. If this is done, the charge transferred during the transitions will result in an offset in all readings. Offsets can be easily removed by applying a zero input periodically and subtracting the result from all subsequent readings. The zero measurement must be repeated periodically because the rise and fall times of the switches drift with temperature and thus the offset will drift.

Multislope runup algorithms can be implemented with constant numbers of switch transitions by alternately placing an S_{+0} and an S_{-0} between each runup slope. Fig. 7 shows the four possible slope patterns between any two S_{+0} slopes. Varying input voltages will cause the algorithm to change between these four patterns, but regardless of which pattern is chosen, each switch makes one and only one transition between the first S_{+0} slope and the S_{-0} slope, and the opposite transition between the S_{-0} slope and the second S_{+0} slope.

The cost of multislope runup is relatively small. The runup slopes can have the same weight as the first slope of multislope rundown. Therefore, only the opposite-polarity slope has to be added, along with the logic to implement the algorithm.

HP 3458A ADC Design

The design of the HP 3458A's ADC is based on these theories for a multislope ADC. Decisions had to be made on how to control the ADC, what number base to use, how fast the integrator can be slewed and remain linear, how much current to force into the integrator (i.e., the size of the resistors), and many other questions. The decisions were affected by both the high-speed goals and the high-resolution goals. For example, very steep slopes are needed to achieve high speed, but steep slopes cause integrator circuits to behave too nonlinearly for high-resolution measurement performance.

One of the easier decisions was to choose a number base for the ADC's multislope rundown. Base e is the optimum to achieve the highest speed, but the task of accumulating an answer is difficult, requiring a conversion to binary. Base 2 and base 4 are both well-suited for binary systems and are close to base e. Base 2 and base 4 are actually equally fast, about 6% slower than base e, but base 2 uses twice as many slopes to achieve the same resolution. Therefore, base 4 was chosen to achieve the required speed with minimum hardware cost.

Microprocessors have always been used to control multislope ADCs, but the speed goals for the HP 3458A quickly

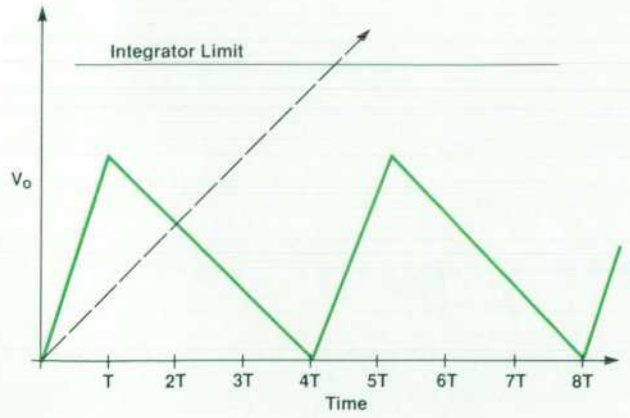


Fig. 5. *Integrator output waveform for multislope runup. The dashed line shows the effective integrator output voltage swing.*

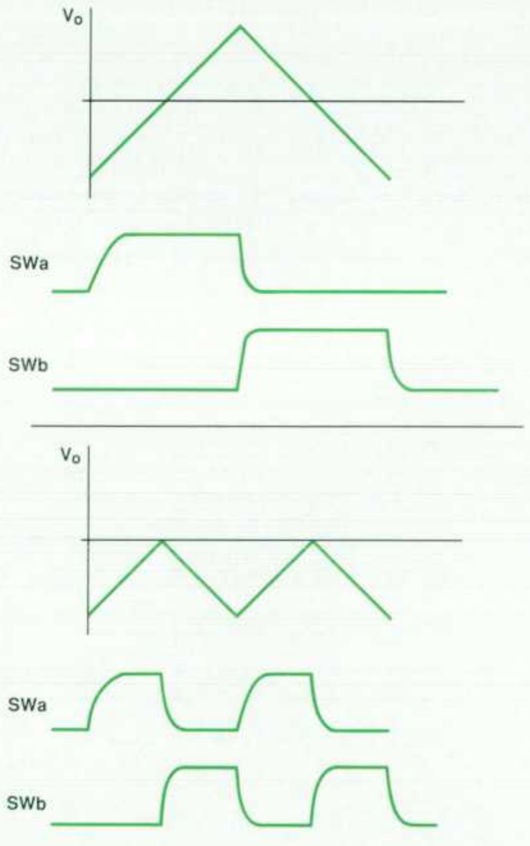


Fig. 6. *Ideally, these two waveforms would transfer equal charge into the integrator, but because of the different number of switch transitions, they do not.*

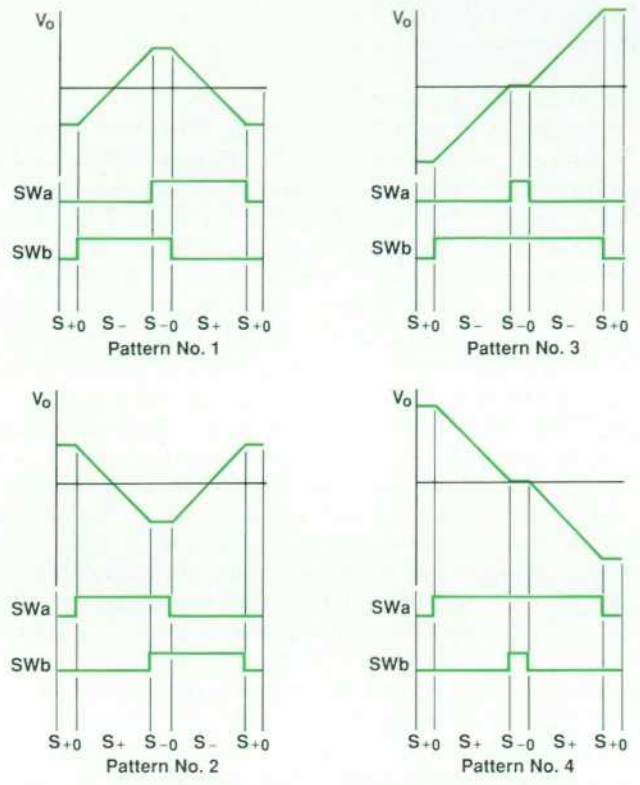


Fig. 7. *Multislope runup patterns for an algorithm that keeps the number of switch transitions constant.*

eliminated the possibility of using a microprocessor to control the ADC algorithm. It was anticipated that the ADC clock frequency would have to be between 10 and 20 MHz, and making decisions at these rates requires dedicated hardware. Therefore, a gate array was chosen to implement state machines running at 20 MHz for ADC control. The ADC control and accumulator functions consume approximately half of a 6000-gate CMOS gate array. The other half of the gate array is devoted to the timing and counting of triggers and a UART (universal asynchronous receiver/transmitter) to transfer data and commands through a 2-Mbit/s fiber optic link to and from the ground-referenced logic (see article, page 31).

The number of slopes and the magnitude of the currents for each slope are more subtle decisions. If the slope currents get too large, they stress the output stage of the integrator's operational amplifier, which can cause nonlinear behavior. If the currents are too small, switch and amplifier leakage currents can become larger than the smallest slope current, and the slope current would not be able to converge the integrator toward zero. A minimum of a microampere for the smallest slope was set to avoid leakage current problems. Also, it was believed that the integrator could handle several milliamperes of input current and remain linear over five or six digits, but that less than a milliampere of input current would be required to achieve linearity over seven or eight digits. On the other hand, greater than a milliampere of current was needed to achieve the high-speed reading rate goal. Therefore, a two-input ADC structure was chosen.

As shown in Fig. 8, when making high-speed measurements, the input voltage is applied through a 10-kΩ resistor, and the ADC's largest slopes, having currents greater than a milliampere, are used. When making high-resolution measurements, the input voltage is applied through a 50-kΩ resistor and the largest slopes used have less than a milliampere of current. The largest slope was chosen to be S1024, having 1.2 μA of current. This led to a total of six slopes (S1024, S256, S64, S16, S4, and S1) with S1 having about 1.2 μA of current. S1024 and S256 are both used during multislope runup; therefore, both polarities exist for both slopes. The $\pm$S256 slopes (0.3 mA) are used when the 50-kohm input is used and both the $\pm$S1024 and the $\pm$S256 slopes (1.5 mA total) are used in parallel when the 10-kΩ input is used. The S256 slope is 25% stronger than a full-scale input to the 50-kΩ resistor, which allows it to keep the integrator from saturating. The 10-kΩ input is five times

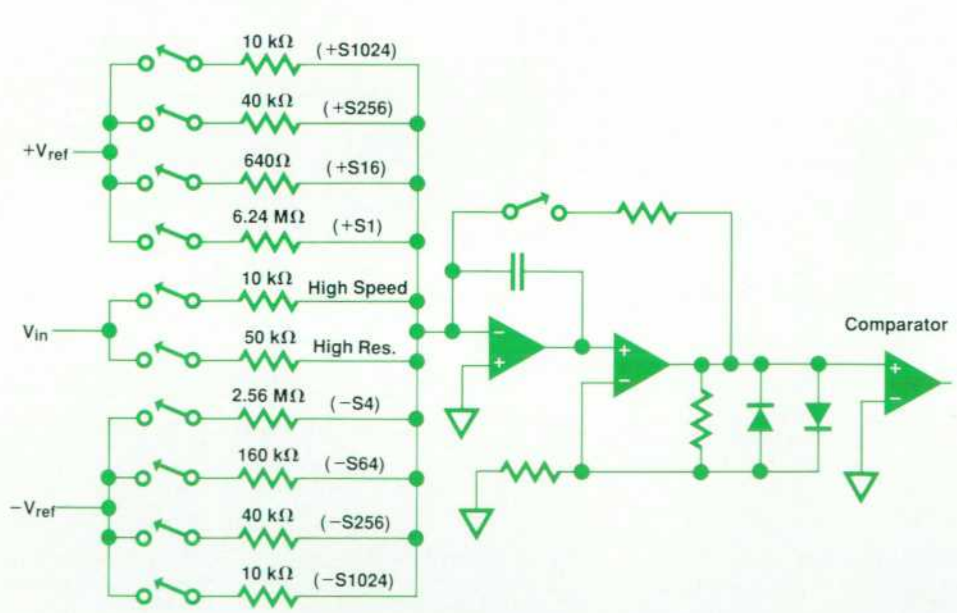


Fig. 8. *Simplified HP 3458A ADC circuit.*

stronger than the 50-kohm input; thus, by using both S1024 and S256, 25% stronger reference slopes can be maintained during high-speed measurements.

Integrator

The integrator's operational amplifier behaves more nonlinearly as the integrator slew rate (i.e., the steepest slope) approaches the slew rate of the amplifier. Two factors determine the integrator slew rate: the total current into the integrator and the size of the integrator capacitor. Wanting to keep the integrator slew rate less than 10V/μs led to an integrator capacitor of 330 pF. This capacitor must have a very small amount of dielectric absorption since 50 fC of charge is one count.

The integrator circuit has to respond to a change in reference current and settle to near 0.01% before the next possible switch transition (about 200 ns). It also has to have low voltage and current noise, about 100V/μs slew rate, a dc gain of at least 25,000, an offset voltage less than 5 mV, and a bias current of less than 10 nA. A custom amplifier design was necessary to achieve all the specifications.

Resistor Network

The resistor network has several requirements. The most stringent is to obtain the lowest ratio-tracking temperature coefficient possible. It is important to keep this coefficient low because the gain of the ADC is dependent on the ratio of the input resistor to the runup reference slope resistors. An overall temperature coefficient of 0.4 ppm/°C was achieved for the ADC. Even at this level, a temperature change of 0.1°C results in a five-count change in a full-scale 8½-digit measurement. (Autocalibration increases the gain stability to greater than 0.15 ppm/°C.)

Another requirement for the resistor network is to have a low enough absolute temperature coefficient that nonlinearities are not introduced by the self-heating of the resistors. For example, the 50-kΩ input resistor has a input voltage that ranges from +12V to −12V. There is a 2.88-milliwatt power difference between a 0V input and a 12V input. If this power difference causes the resistor to change its value, the result is a nonlinearity in the ADC. A 0.01°C temperature change in a resistor that has an absolute temperature coefficient of 1 ppm/°C causes a one-count error in an 8½-digit measurement. The network used in the HP 3458A's ADC shows no measurable self-heating nonlinearities.

The final requirement of the resistor network is that it maintain the ratios of the six slopes throughout the life of the HP3458A. The tightest ratio tolerance is approximately 0.1% and is required to maintain linearity of the high-speed measurements. This is a relatively easy requirement. To maintain the ADC's 8½-digit differential linearity at less than 0.02 ppm requires ratio tolerances of only 3%.

Switches

A last major concern for the ADC design was the switches required to control the inputs and the slopes. Because the switches are in series with the resistors, they can add to the temperature coefficient of the ADC. A custom chip design was chosen so that each switch could be scaled to the size of the resistor to which it is connected. This allows the ADC to be sensitive to the ratio-tracking temperature coefficient of the switches and not to the absolute temperature coefficient. Another advantage of the custom design is that it allows the control signals to be latched just before the drives to the switches. This resynchronizes the signal with the clock and reduces the timing jitter in the switch transitions. The result is a reduction in the noise of the ADC.

Performance

The performance of an ADC is limited by several nonideal behaviors. Often the stated resolution of an ADC is limited by differential linearity or noise even though the number of counts generated would indicate much finer resolution. For example, the HP 3458A's ADC generates more than 9½ digits of counts but is only rated at 8½ digits because the ninth digit is very noisy and the differential linearity is about one eight-digit count. Therefore, when stating an ADC's speed and resolution, it is important to specify under what conditions the parameters are valid. Fig. 9 shows the speed-versus-resolution relationship of the HP 3458A ADC assuming less than one count of rms noise.

Given a noise level, there is a theoretical limit to the resolution of an ADC for a given speed. It can be shown that the white noise bandwidth of a signal that is the output of an integration over time T is

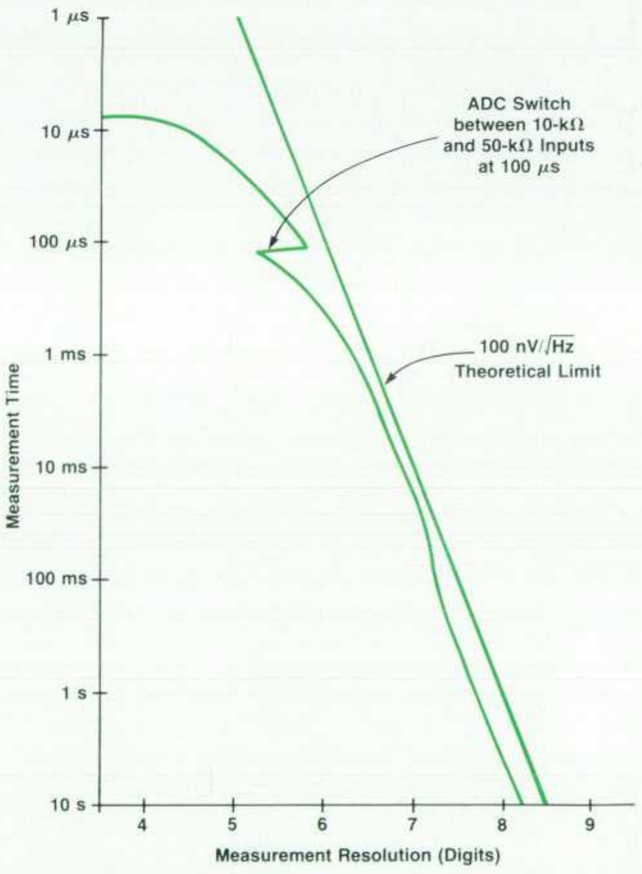


Fig. 9. *HP 3458A ADC speed versus resolution for one count of rms noise.*

$$BW = 1/2T.$$

If rundown took zero time then an integrating ADC could sample once every T seconds. At this rate. the counts of resolution, M, of an ADC are noise-limited to

$$M = (V_{fs}\sqrt{2T})/V_n,$$

where V_{fs} is the full-scale input voltage to the ADC and V_n is the white noise of the ADC in $V/\sqrt{Hz}$. Fig. 9 shows the best theoretical resolution for an ADC with rms noise of $100\ nV/\sqrt{Hz}$ and a full-scale input of 10 volts. The HP 3458A comes very close to the theoretical limit of an ADC with a white noise of $130\ nV/\sqrt{Hz}$ near the 7-digit resolution region. At lower resolutions the ADC's rundown time becomes a more significant portion of the overall measurement time and therefore pulls the ADC away from the theoretical limit. At higher resolutions the 1/f noise of the ADC forces a measurement of zero several times within the measurement cycle to reduce the noise to the 8½-digit level. This also reduces the measurement speed.

Another way of viewing the ADC's performance is to plot resolution versus aperture. The aperture is the integration time, that is, the length of runup. This is shown in Fig. 10 along with the $100\text{-}nV/\sqrt{Hz}$ noise limit and the ADC's resolution without regard to noise. At smaller apertures, the HP 3458A's resolution is less than the theoretical noise limit because it is limited by noise in detecting the final zero of rundown. That is, the algorithm does not have enough resolution to achieve the theoretical resolution.

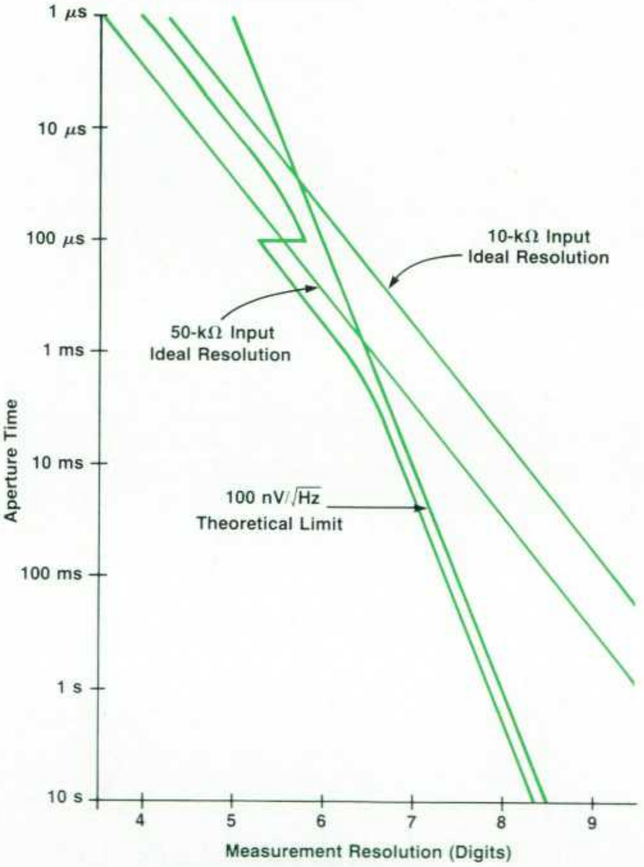


Fig. 10. *HP 3458A ADC aperture (runup time) versus resolution.*

Linearity

High-resolution linearity was one of the major challenges of the ADC design. The autocalibration technique requires an integral linearity of 0.1 ppm and an differential linearity of 0.02 ppm. One of the more significant problems was verifying the integral linearity. The most linear commercially available device we could find was a Kelvin-Varley divider, and its best specification was 0.1 ppm of input. Fig. 11 compares this with the ADC's requirements, showing that it is not adequate.

Using low-thermal-EMF switches, any even-ordered deviations from an ideal straight line can be detected by doing a turnover test. A turnover test consists of three steps: (1) measure and remove any offset, (2) measure a voltage, and (3) switch the polarity of the voltage (i.e., turn the voltage over) and remeasure it. Any even-order errors will produce a difference in the magnitude of the two nonzero voltages measured. Measurements of this type can be made to within 0.01 ppm of a 10V signal. This left us with only the odd-order errors to detect. Fortunately, the U.S. National Bureau of Standards had developed a Josephson junction array capable of generating voltages from −10V to +10V. Using a 10V array we were able to measure both even-order and odd-order errors with confidence to a few hundredths of a ppm. Fig. 4a on page 23 shows the integral linearity error of an HP 3458A measured using a Josephson junction array.

The differential linearity can be best seen by looking at a small interval about zero volts. Here a variable source need only be linear within 1 ppm on its 100-mV range to produce an output that is within 0.01 ppm of 10 volts. Fig. 4b on page 23 shows the differential linearity of an HP 3458A.

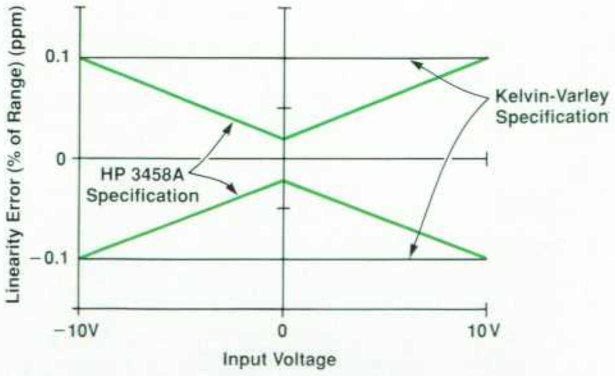


Fig. 11. *HP 3458A linearity specification compared with a Kelvin-Varley divider.*

Acknowledgments

The evolution of multislope ADC technology started with the development of the HP 3455A DMM over 15 years ago and has been refined by many designers working on several projects. Through the years Al Gookin, Joe Marriott, Larry Jones, James Ressmeyer, and Larry DesJardin have made significant contributions to multislope ADC concepts. I also want to thank John Roe of Hewlett-Packard's Colorado Integrated Circuits Division for his efforts in the design of the custom switch chip, David Czenkusch and David Rustici for the gate array development, and Steve Venzke for his valuable input to the integrator design. I would like to commend Clark Hamilton and Dick Harris of the U.S. National Bureau of Standards for developing the Josephson junction array voltage system that contributed to the refinement of the ADC technology developed for the HP 3458A.

Precision AC Voltage Measurements Using Digital Sampling Techniques

Instead of traditional DMM techniques such as thermal conversion or analog computation, the HP 3458A DMM measures rms ac voltages by sampling the input signal and computing the rms value digitally in real time. Track-and-hold circuit performance is critical to the accuracy of the method.

by Ronald L. Swerlein

THE HP 3458A DIGITAL MULTIMETER implements a digital method for the precise measurement of rms ac voltages. A technique similar to that of a modern digitizing oscilloscope is used to sample the input voltage waveform. The rms value of the data is computed in real time to produce the final measurement result. The HP 3458A objectives for high-precision digital ac measurements required the development of both new measurement algorithms and a track-and-hold circuit capable of fulfilling these needs.

Limitations of Conventional Techniques

All methods for making ac rms measurements tend to have various performance limitations. Depending on the needs of the measurement, these limitations take on different levels of importance.

Perhaps the most basic specification of performance is accuracy. For ac measurements, accuracy has to be specified over a frequency band. Usually, the best accuracy is for sine waves at midband frequencies (typically 1 kHz to 20 kHz). Low-frequency accuracy usually refers to frequencies below 200 Hz (some techniques can work down to 1 Hz). Bandwidth is a measure of the technique's performance at higher frequencies.

Linearity is another measure of accuracy. Linearity is a measure of how much the measurement accuracy changes when the applied signal voltage changes. In general, linearity is a function of frequency just as accuracy is, and can be included in the accuracy specifications. For instance, the accuracy at 1 kHz may be specified as 0.02% of reading + 0.01% of range while the accuracy at 100 kHz may be specified as 0.1% of reading + 0.1% of range. The percent-of-range part of the specification is where most of the linearity error is found.

If a nonsinusoid is being measured, most ac rms measurement techniques exhibit additional error. Crest-factor error is one way to characterize this performance. Crest factor is defined as the ratio of the peak value of a waveform to its rms value. For example, a sine wave has a crest factor of 1.4 and a pulse train with a duty cycle of 1/25 has a crest factor of 5. Even when crest factor error is specified, one should use caution when applying this additional error if it is not given as a function of frequency. A signal with a moderately high crest factor may have significant frequency components at 40,000 times the fundamental frequency. Thus crest factor error should be coupled with bandwidth information in estimating the accuracy of a measurement. In some ac voltmeters, crest factor specifications mean only that the voltmeter's internal amplifiers will remain unsaturated with this signal, and the accuracy for nonsinusoids may actually be unspecified.

Some of the secondary performance specifications for rms measurements are short-term reading stability, settling time, and reading rate. These parameters may have primary importance, however, depending on the need of the measurement. Short-term stability is self-explanatory, but the difference between settling time and reading rate is sometimes confusing. Settling time is usually specified as the time that one should wait after a full-scale signal amplitude change before accepting a reading as having full accuracy. Reading rate is the rate at which readings can be taken. Its possible for an ac voltmeter that has a one-second settling time to be able to take more than 300 readings per second. Of course, after a full-scale signal swing, the next 299 readings would have degraded accuracy. But if the input signal swing is smaller than full-scale, the settling time to specified accuracy is faster. Therefore, in some situations, the 300 readings/second capability is actually useful even though the settling time is one second.

The traditional methods for measuring ac rms voltage are thermal conversion and analog computation. The basis of thermal conversion is that the heat generated in a resistive element is proportional to the square of the rms voltage applied to the element. A thermocouple is used to measure this generated heat. Thermal conversion can be highly accurate with both sine waves and waveforms of higher crest factor. Indeed, this accuracy is part of the reason that the U.S. National Institute of Standards and Technology (formerly the National Bureau of Standards) uses this method to supply ac voltage traceability. It can also be used at very high frequencies (in the hundreds of MHz). But thermal conversion tends to be slow (near one minute per reading) and tends to exhibit degraded performance at low frequencies (below 20 Hz). The other major limitation of thermal conversion is dynamic range. Low output voltage, low thermal coupling to the ambient environment, and other factors limit this technique to a dynamic range of around 10 dB. This compares to the greater than 20 dB range typical of other techniques.

Analog computation is the other common technology used for rms measurements. Essentially, the analog converter uses logging and antilogging circuitry to implement an analog computer that calculates the squares and square roots involved in an rms measurement. Since the rms averaging is implemented using electronic filters (instead of the physical thermal mass of the thermal converter), analog computation is very flexible in terms of reading rate. This flexibility is the reason that this technique is offered in the HP 3458A Multimeter as an ATE-optimized ac measurement function (SETACV ANA). Switchable filters offer settling times as fast as 0.01 second for frequencies above 10 kHz. With such a filter, reading rates up to 1000 readings/second may be useful.

Analog computation does have some severe accuracy drawbacks, however. It can be very accurate in the midband audio range, but both its accuracy and its linearity tend to suffer severe degradations at higher frequencies. Also, the emitter resistances of the transistors commonly used to implement the logging and antilogging functions tend to cause errors that are crest-factor dependent.

Digital AC Technique

Digital ac is another way to measure the rms value of a signal. The signal is sampled by an analog-to-digital converter (ADC) at greater than the signal's Nyquist rate to avoid aliasing errors. A digital computer is then used to compute the rms value of the applied signal. Digital ac can exhibit excellent linearity that doesn't degrade at high frequencies as analog ac computation does. Accuracy with all waveforms is comparable to thermal techniques without their long settling times. It is possible to measure low frequencies faster and with better accuracy than other methods using digital ac measurements. Also, the technique lends itself to autocalibration of both gain and frequency response errors using only an external dc voltage standard (see article, page 22).

In its basic form, a digital rms voltmeter would sample the input waveform with an ADC at a fast enough rate to avoid aliasing errors. The sampled voltage points (in the form of digital data) would then be operated on by an rms estimation algorithm. One example is shown below:

Num	= number of digital samples
Sum	= sum of digital data
Sumsq	= sum of squares of digital data
ac rms	= SQR((Sumsq−Sum*Sum/Num)/Num)

Conceptually, digital rms estimation has many potential advantages that can be exploited in a digital multimeter (DMM). Accuracy, linearity over the measurement range, frequency response, short-term reading stability, and crest factor performance can all be excellent and limited only by the errors of the ADC. The performance limitations of digital ac are unknown at the present time since ADC technology is continually improving.

Reading rates can be as fast as theoretically possible because ideal averaging filters can be implemented in firmware. Low-frequency settling time can be improved by measuring the period of the input waveform and sampling only over integral numbers of periods. This would allow a 1-Hz waveform to be measured in only two seconds—one second to measure the period and one second to sample the waveform.

Synchronous Subsampling

A thermal converter can measure ac voltages in the frequency band of 20 Hz to 10 MHz with state-of-the-art accuracy. Sampling rates near 50 MHz are required to measure these same frequencies digitally, but present ADCs that can sample at this rate have far less linearity and stability than is required for state-of-the-art accuracy in the audio band. If the restriction is made that the signal being measured must be repetitive, however, a track-and-hold circuit can be used ahead of a slower ADC with higher stability to create an ADC that can effectively sample at a much higher rate. The terms "effective time sampling," "equivalent time

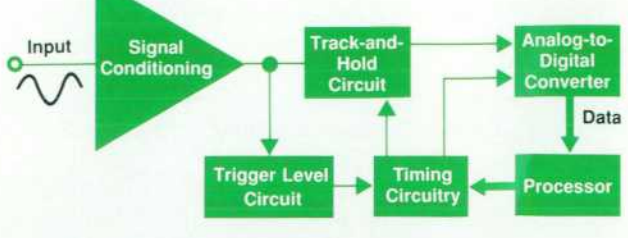


Fig. 1. *Simplified block diagram of a subsampling ac voltmeter.*

sampling," and "subsampling" are used interchangeably to describe this technique.

The concept of subsampling is used by digitizing oscilloscopes to extend their sample rate to well beyond the intrinsic speed of the ADC. The concept is to use a trigger level circuit to establish a time reference relative to a point on a repetitive input signal. A timing circuit, or time base, is used to select sample delays from this reference point in increments determined by the required effective sampling rate. For example, moving the sampling point in 10-ns increments corresponds to an effective sampling rate of 100 MHz. A block diagram of a subsampling ac voltmeter is shown in Fig. 1.

Fig. 2 is a simple graphic example of subsampling. Here we have an ADC that can sample at the rate of five samples for one period of the waveform being measured. We want to sample one period of this waveform at an effective rate of 20 samples per period. First, the timing circuit waits for a positive zero crossing and then takes a burst of five readings at its fastest sample rate. This is shown as "First Pass" in Fig. 2. On a subsequent positive slope, the time base delays an amount of time equal to one fourth of the ADC's minimum time between samples and again takes a burst of five readings. This is shown as "Second Pass." This continues through the fourth pass, at which time the applied repetitive waveform has been equivalent time sampled as if the ADC could acquire data at a rate four times faster than it actually can.

The digital ac measurement technique of the HP 3458A is optimized for precision calibration laboratory measurements. Short-term measurement stability better than 1 ppm has been demonstrated. Absolute accuracy better than 100 ppm has been shown. This accuracy is achieved by automatic internal adjustment relative to an external 10V dc standard. No ac source is required (see article, page 22). The internal adjustments have the added benefit of providing a quick, independent check of the voltage ratios and transfers that are typically performed in a standards laboratory every day. Fast, accurate 1-Hz measurements and superb performance with nonsinusoids allow calibration laboratories to make measurements easily that were previously very difficult.

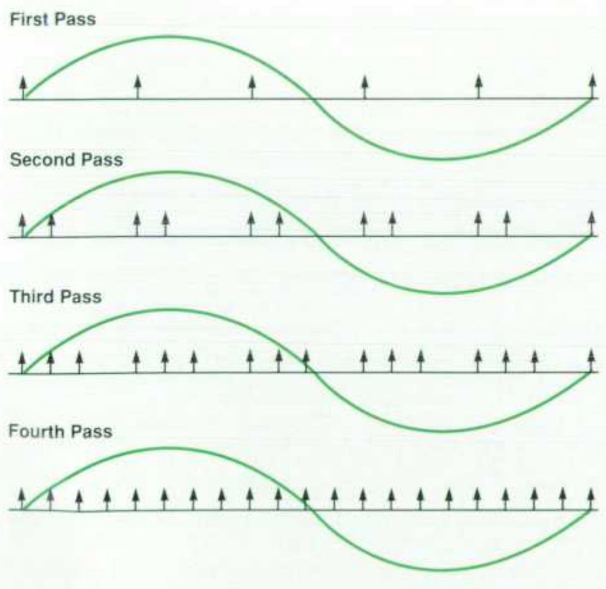


Fig. 2. *An example of subsampling.*

The HP 3458A enters into the synchronously subsampled ac mode through the command SETACV SYNC. For optimal sampling of the input signal, one must determine the period of the signal, the number of samples required, and the signal bandwidth. The measurement resolution desired and the potential bandwidth of the input waveform are described using the commands RES and ACBAND. The period of the input signal is measured by the instrument. The more the HP 3458A knows about the bandwidth of the input and the required measurement resolution, the better the job it can do of optimizing accuracy and reading rate. Default values are assumed if the user chooses not to enter more complete information. An ac measurement using the SYNC mode appears to function almost exactly the same to the user as one made using the more conventional analog mode.

Subsampled AC Algorithm

The algorithm applied internally by the HP 3458A during each subsampled ac measurement is totally invisible to the user. The first part of a subsampled ac measurement is autolevel. The input waveform is randomly sampled for a period of time long enough to get an idea of its minimum and maximum voltage points. This time is at least one cycle of the lowest expected frequency value (the low-frequency value of ACBAND). The trigger level is then set to a point midway between the minimum and maximum voltages, a good triggering point for most waveforms. In the unlikely event that this triggering point does not generate a reliable trigger, provision is made for the user to generate a trigger signal and apply it to an external trigger input. An example of such a waveform is a video signal. Even though video signals can be repetitive, they are difficult to trigger on correctly with just a standard trigger level.

With the trigger level determined, the period of the input waveform is measured. The measured period is used along with the global parameter RES to determine subsampling parameters. These parameters are used by the timing circuitry in the HP 3458A to select the effective sample rate, the number of samples, and the order in which these samples are to be taken. In general, the HP 3458A tries to sample at the highest effective sample rate consistent with meeting the twin constraints of subsampling over an integral number of input waveform periods and restricting the total number of samples to a minimum value large enough to meet the specified resolution. This pushes the frequency where aliasing may occur as high as possible and also performs the best rms measurement of arbitrary waveforms of high crest factor. The number of samples taken will lie somewhere between 4/RES and 8/RES depending on the measured period of the input waveform.

The final step is to acquire samples. As samples are taken, the data is processed in real time at a rate of up to 50,000 samples per second to compute a sum of the readings squared and a sum of the readings. After all the samples are taken, the two sum registers are used to determine standard deviation (ACV function), or rms value (ACDCV function). For example, suppose a 1-kHz waveform is being measured and the specified measurement resolution is 0.001%. When triggered, the HP 3458A will take 400,000 samples using an effective sample rate of 100 MHz. The timing circuit waits for a positive-slope trigger level. Then, after a small fixed delay, it takes a burst of 200 readings spaced 20 μs apart. It waits for another trigger, and when this occurs the timing circuit adds 10 ns to the previous delay before starting another burst of 200 readings. This is repeated 2,000 times, generating 400,000 samples. Effectively, four periods of the 1-kHz signal are sampled with samples placed every 10 ns.

Sources of Error

Internal time base jitter and trigger jitter during subsampling contribute measurement uncertainty to the rms measurement. The magnitude of this uncertainty depends on the magnitude of these timing errors. The internal time base jitter in the HP 3458A is less than 100 ps rms. Trigger jitter is dependent on the input signal's amplitude and frequency, since internal noise will create greater time uncertainties for slow-slew-rate signals than for faster ones. A readily achievable trigger jitter is 100 ps rms for a 1-MHz input. Fig. 3 is a plot generated by mathematical modeling of the performance of a 400,000-sample ac measurement using the HP 3458A's subsampling algorithm (RES = 0.001%) in the presence of 100-ps time base and trigger jitters. The modeled errors suggest the possibility of stable and accurate ac measurements with better than 6-digit accuracy.

Errors other than time jitter and trigger jitter limit the typical absolute accuracy of the HP 3458A to about 50 ppm, but there is reason to believe that short-term stability is better than 1 ppm. Many five-minute stability tests using a Datron 4200 AC Calibrator show reading-to-reading standard deviations between 0.7 ppm and 3 ppm. Other measurements independently show the Datron 4200 to have similar short-term stability. More recently, tests performed using a Fluke 5700 Calibrator, which uses a theoretically quieter leveling loop, show standard deviations under 0.6 ppm.

The above algorithm tries to sample the applied signal over an integral number of periods. To do this, the period of the signal must first be measured. Errors in measuring the period of the input waveform will cause the subsequent sampling to cover more or less than an integral number of periods. Thus, the accuracy of the subsampled ac rms measurement is directly related to how accurately the period of the input waveform is known relative to the internal sample time base clock.

Period measurements in the HP 3458A are performed using reciprocal frequency counting techniques. This method allows accuracy to be traded off for measurement speed by selecting different gate times. A shorter gate time contributes to a faster measurement, but the lower accuracy of the period determination contributes to a less accurate ac measurement. Fig. 4 is a graph of the error introduced into the rms measurement by various gate times. At high frequencies, this error is a constant dependent on the resolution of the frequency counter for a given gate time. At lower frequencies, trigger time jitter increases, causing increased error, because random noise has a larger effect on slower signals. At still lower frequencies, where the period being measured is longer than the selected gate time, this error becomes constant again. This is because the gate time is always at least one period in length, and as the frequency is lowered, the gate time increases just fast enough to cancel the effect of increasing trigger jitter.

Any violation of the restriction that the input waveform be repetitive will also lead to errors. A common condition is amplitude and frequency modulation of the input. If this modulation is of a fairly small magnitude and is fast compared to the total measurement time this violation of the repetitive requirement will probably be negligible. At most, reading-to-reading variation might increase slightly. If these modulations become large, however, subsampled ac accuracy can be seriously compromised. The signal sources typically present on a lab bench or in a calibration laboratory work quite well with the subsampling algorithm of the HP 3458A.

Random noise spikes superimposed on an input can make an otherwise repetitive input waveform appear nonrepetitive. Induced current caused by motors and electrical devices turning on and off is just one of many ways to generate such spikes. Large test systems tend to generate more of this than bench and calibration laboratory environments. Low-voltage input signals (below 100 mV) at low

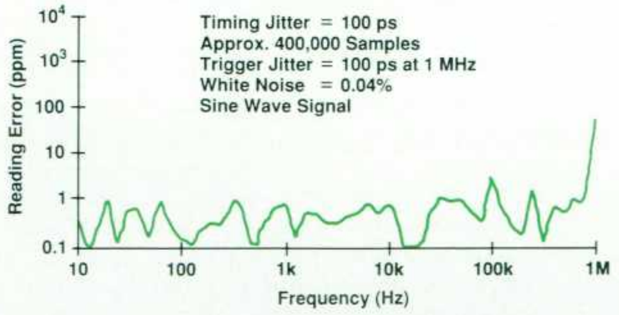


Fig. 3. *Subsampling errors resulting from timing uncertainties.*

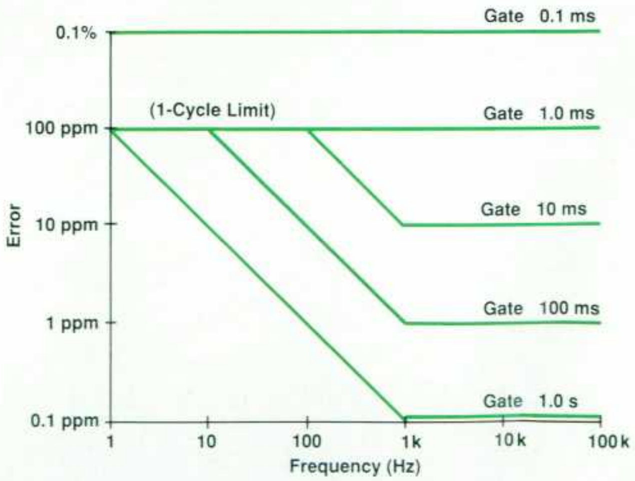


Fig. 4. *Subsampling error as a function of gate time.*

frequencies are the signals most susceptible to these errors.

Two ways are provided by the HP 3458A to deal with these potential errors. The first is to use the internal 80-kHz low-pass trigger filter to reduce high-frequency trigger noise (LFILTER ON). If this is not enough, provision is made for accepting external synchronization pulses. In principle, getting subsampled ac to work in a noisy environment is no more difficult than getting a frequency counter to work in the same environment.

If nonsinusoidal signals are being measured, the subsampling algorithm has some additional random errors that become greater for signals of large crest factor. All nonsinusoidal repetitive signals have some of their spectral energy at frequencies higher than their fundamental frequency. Signals of high crest factor generally have more of this high-frequency energy than those of lower crest factor. Random timing jitter, which tends to affect higher frequencies the most, will create measurement errors that are greater for large-crest-factor signals. These random errors can be reduced by specifying a higher-resolution measurement, which forces more samples per reading to be acquired. The additional measurement error induced by a signal crest factor of 5 can be as low as 50 ppm in the HP 3458A.

Track-and-Hold Circuit

Track-and-hold performance is critical to the accuracy of digital ac measurements. Track-and-hold linearity, bandwidth, frequency flatness, and aperture jitter all affect the error in a sampled measurement. To meet the HP 3458A performance objectives, track-and-hold frequency response flatness of ±0.0015% (15 ppm) was required from dc to 50 kHz, along with a 3-dB bandwidth of 15 MHz. In addition, 16-bit linearity below 50 kHz and low aperture jitter were needed. A custom track-and-hold amplifier was developed to meet these requirements.

The most basic implementation of a track-and-hold circuit—a switch and a capacitor—is shown in Fig. 5. If the assumption is made that the switch is perfect (when open it has infinite impedance and when closed it has zero impedance) and if it assumed that the capacitor is perfect (no dielectric absorption), then this is a perfect track-and-hold circuit. The voltage on the capacitor will track the input signal perfectly in track mode, and when the switch is opened, the capacitor will hold its value until the switch is closed. Also, as long as the buffer amplifier's input impedance is high and well-behaved, its bandwidth can be much lower than the bandwidth of the signal being sampled. When the switch is opened, the buffer amplifier's output might not have been keeping up with the input signal, but since the voltage at the input of the amplifier is now static, the buffer will eventually settle out to the hold capacitor's voltage.

The problem with building Fig. 5 is that it is impossible at the present time to build a perfect switch. When the switch is opened it is not truly turned off; it has some residual leakage capacitance and resistance. In hold mode, there is some residual coupling to the input signal because of this leakage capacitance. This error term is commonly called feedthrough. Another error term is pedestal voltage. The process of turning real-world switches off induces a charge transfer that causes the hold capacitor to experience a fixed voltage step (a pedestal) when entering hold mode.

Another problem with Fig. 5 is that it is impossible in the real world to build a perfect capacitor. Real-world capacitors have nonideal behaviors because of dielectric absorption and other factors. This dielectric absorption will manifest itself as a pedestal that is different for different input-voltage slew rates. Even if the capacitor is refined until it is "perfect enough," the switch and the buffer amplifier may contribute enough capacitance in parallel with C_{hold} that the resultant capacitance has dielectric absorption problems.

Fig. 6 is an implementation of Fig. 5 using real-world components. The switch is implemented with a p-channel MOS FET. When the drive voltage is −15V, the circuit is in track mode. If the FET has an on resistance of R, then the 3-dB bandwidth of the circuit is $1/(2\pi RC_{hold})$. C_{dg} (the drain-to-gate capacitance) is always in parallel with C_{hold}, so even if C_{hold} and the buffer amplifier have low dielectric absorption, the dielectric absorption associated with C_{dg} will cause this circuit to exhibit pedestal changes with different input signal slew rates.

When the drive voltage is changed to +15V, the FET turns off and puts the circuit into hold mode. The drain-to-source capacitance (C_{ds}) contributes feedthrough error equal to C_{ds}/C_{hold}. If the drive voltage changes infinitely fast, the pedestal error is $(30V)(C_{dg}/C_{hold})$. If the drive voltage changes at a slower rate, the pedestal error will be less, but a gain error term will now appear. Assume that the drive voltage changes slowly relative to the bandwidth of the track-and-hold circuit ($1/(2\pi RC_{hold})$). Assume also that the FET is on until the drive voltage is equal to V_{in} and that it is off when the drive voltage is greater than V_{in}. The process of going into hold mode begins with the drive voltage changing from −15V to +15V. As the voltage changes from −15V to V_{in}, C_{hold} experiences very little pedestal error since the current $C_{dg}(dv/dt)$ mostly flows into the FET, which is on. When the drive voltage reaches V_{in}, the FET turns off and all of the $C_{dg}(dv/dt)$ current flows into C_{hold}. The pedestal in this case is $(15V - V_{in})(C_{dg}/C_{hold})$. Notice that this is a smaller pedestal than in the previous case where the drive voltage changed infinitely fast. Also notice that there is a V_{in} term in the pedestal equation. This is a gain error.

Pedestal errors are easy to deal with in the real world. There are a number of easy ways to remove offset errors.

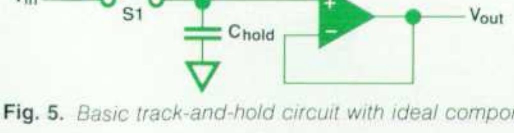


Fig. 5. *Basic track-and-hold circuit with ideal components.*

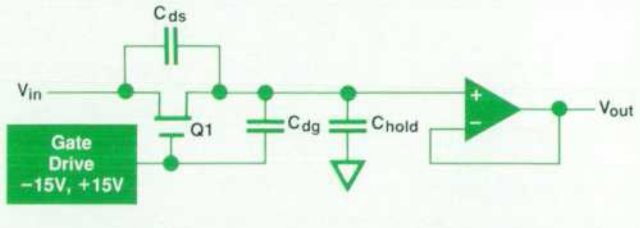


Fig. 6. *Basic track-and-hold circuit with real components.*

Gain errors are not necessarily bad either, since ideal gain errors can be corrected with a compensating gain stage. But because C_{dg} is a semiconductor capacitance, it tends to change value with the applied voltage. This leads to a form of error called nonlinearity. In general, gain errors that are caused by semiconductor capacitances (like that calculated in the above paragraph) lead to nonlinearity errors. A track-and-hold circuit application that is affected by nonlinearity errors is sampling a signal and calculating its Fourier transform. Feedthrough and dielectric absorption errors also are hard to deal with. Commonly, a different track-and-hold architecture is used to achieve better linearity, feedthrough, and dielectric absorption performance.

Fig. 7 is a diagram of the track-and-hold architecture used most often to achieve 16-bit or better resolution along with 2-MHz bandwidths. In track mode the drive voltage is −15V, turning Q1 on. The output voltage is the inverse of the input voltage. The inverse of the input voltage is impressed across C_{hold} during track mode. When the drive voltage is changed to +15V, Q1 turns off and V_{out} is held.

Fig. 7 has several advantages over Fig. 6. Since the switch (Q1) is at a virtual ground point, the pedestal voltage is constant with V_{in} and equal to $(30V)(C_{dg}/C_{hold})$. This is because the drain and source are always at zero so that when Q1 is turned off the same amount of charge is always transferred to C_{hold}. Also, since no point on Q1 moves with V_{in}, the FET does not contribute any dielectric absorption error terms.

Fig. 7 does have feedthrough error. It is equal to $\frac{1}{2}(C_{ds}/C_{hold})$. Theoretically this error could be substantially eliminated if a second switch could be turned on after entering hold mode to ground the junction of the two resistors. However, a real drawback of this circuit is that the op amp U1 has to have the same bandwidth and slew rate capabilities as the signal being sampled. In the descriptions of Figs. 5 and 6 it was mentioned that the buffer amplifier need not have the same bandwidth as the signal being sampled. So in summary, Fig. 7 eliminates some of the errors of the previous circuits but introduces at least one new limitation.

HP 3458A Track-and-Hold Architecture

Fig. 8 is a modification of Fig. 6 that has most of the advantages and very few of the disadvantages of the previous circuits. Here the switch is implemented with two n-channel JFETs and one p-channel MOS FET. In track mode the JFETs Q1 and Q2 are on and the MOS FET Q3 is off. Q1 and Q2 are on because their gate-to-source voltages are zero, since their gates track V_{in}. Their gates track V_{in} because in track mode point B is an open circuit and CR1 and CR2 act like resistances of about 1 kΩ. CR1 and CR2 are current regulator diodes, which are simply JFETs with their gates wired to their sources. In hold mode, Q1 and Q2 are off and Q3 is on. Q1 is now off because point B is now at −15V and thus the gate of Q1 is at −15V. CR2 now appears as a current source of high resistance and the gate of Q2 is clamped at about 7V below V_{out}, turning off Q2. Q3 is on because its gate (point A) is at −15V.

In hold mode, feedthrough error is very low, since the feedthrough caused by C_{ds1} is shunted into the ac ground created by Q3's being on. Also, the pedestal error caused by C_{dg2} is constant for all V_{in}, since the gate of Q2 is clamped at 7V below V_{out}. Since V_{out} is tracking V_{in} during track mode (or will settle out to V_{in} after hold mode is entered), the pedestal error caused by C_{dg2} is $(-7V)(C_{dg2}/C_{hold})$ and has no V_{in} dependent terms. Therefore it makes no difference to the linearity errors of the track-and-hold circuit whether C_{dg2} is nonlinear with bias voltage.

It is not so obvious that C_{ds2} contributes almost nothing to the pedestal errors and the nonlinearity errors of the circuit. In addition to being a T-switch that reduces feedthrough errors in hold mode, Q1, Q2, and Q3 when switched in the correct sequence act to remove almost all of the pedestal errors caused by C_{ds2}. This is very important, since C_{ds2} is nonlinear, and if its pedestal errors remained, the linearity of the circuit would be no better than that of Fig. 6. Q1 is selected such that its pinchoff voltage (V_{gsoff}) is greater than that of Q2. Thus, as point B is driven to −15V, Q2 turns off before Q1. Once Q2 is off, the only coupling path to C_{hold} is through the capacitance C_{ds2}.

Fig. 9 shows the various waveforms present in the circuit. When Q1 is finally turned off, the voltage on C1 has a pedestal error of $(V_{in} - 15V)(C_{dg1}/C_1)$. This pedestal couples into C_{hold} through C_{ds2}. The magnitude is $(V_{in} - 15V)(C_{dg1}/C_1)(C_{ds2}/C_{hold})$. Since C_{dg1} is nonlinear and the coupling has a V_{in} dependent term, the pedestal on C_{hold} now has a nonlinear component. But after Q1 and Q2 are off, point A is driven to −15V, turning Q3 on. C_1 is now connected to V_{out} through the on resistance of Q3 and approaches the voltage V_{out}. This voltage movement, which

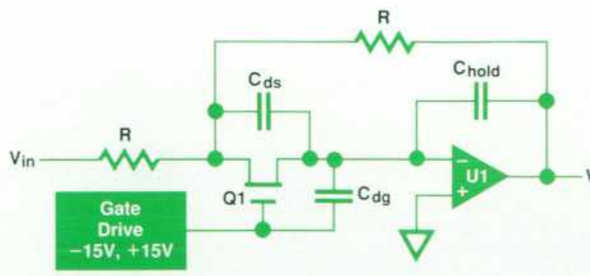


Fig. 7. *Conventional track-and-hold architecture.*

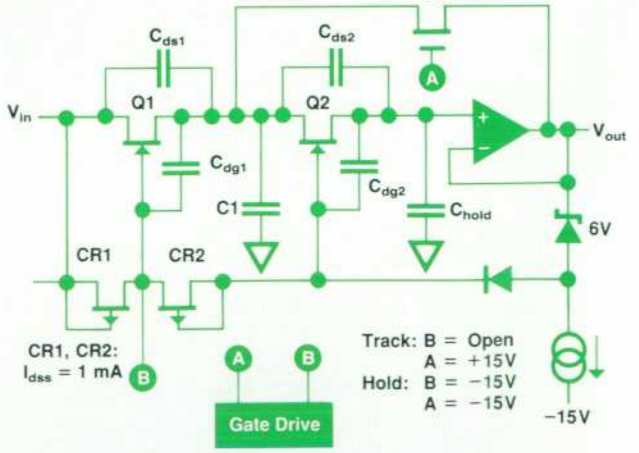


Fig. 8. *HP 3458A track-and-hold architecture.*

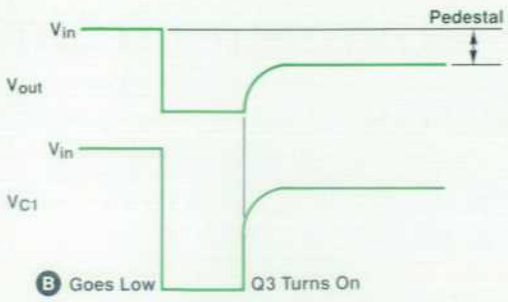


Fig. 9. *Waveforms in the circuit of Fig. 8.*

is of the same magnitude as C_1's previous change but in the opposite direction, couples into C_{hold} through C_{ds2} and totally removes the pedestal error previously coupled into C_{hold} through C_{ds2}.

Another point that also might not be so obvious is that Q1, Q2, and Q3 do not contribute any dielectric absorption errors to the track-and-hold circuit. Since in track mode the drains, sources, and gates of Q1 and Q2 are at the same potential (V_{in}), none of the FET capacitances has charge on it before it is put in hold mode. Therefore, the charge transferred to C_{hold} through the FET capacitances when hold mode is entered is the same for any value or slew rate of V_{in}, so it doesn't matter whether the FET capacitances have high dielectric absorption.

Summary

The performance of the HP 3458A with sinusoidal and nonsinusoidal inputs is known to be very good. The DMM was tested against a synthesized arbitrary waveform generator under development at the U.S. National Bureau of Standards which was capable of generating sine waves and ANSI-standard distorted sine waves with an absolute uncertainty of 10 ppm. The HP 3458A measured all of the various test waveforms with errors ranging from 5 ppm to 50 ppm for 7V rms inputs from 100 Hz to 10 kHz.

The digital ac measurement capability of the HP 3458A combines the best features of the traditional thermal and analog computational ac rms techniques in addition to adding several advantages of its own. Measurement accuracies for digital ac are comparable to thermal techniques for both sinusoidal (crest factor 1.4) and large-crest-factor nonsinusoidal waveforms. Like analog computation, digital ac reading rates are reasonably fast compared to thermal rms techniques. The major advantages of digital ac include linearity superior to traditional analog rms detection methods and significantly faster low-frequency rms ac measurements (less than six seconds for a 1-Hz input). Short-term reading stability is excellent, allowing previously difficult characterizations to be performed easily.

Acknowledgments

Credit should be given to Larry DesJardin for his direction and support for the development of the digital ac techniques implemented in the HP 3458A. I would also like to thank Barry Bell and Nile Oldham of the National Bureau of Standards for allowing access to their synthesized ac source to help validate the digital ac concept.

Calibration of an 8½-Digit Multimeter from Only Two External Standards

Internal transfer standards and autocalibration simplify external calibration and extend the period between external calibrations to two years.

by Wayne C. Goeke, Ronald L. Swerlein, Stephen B. Venzke, and Scott D. Stever

ONE OF THE EARLIEST PRODUCT CONCEPTS for the HP 3458A Digital Multimeter was to develop a means for calibrating its measurement accuracies from only two external reference standards. This is not possible with the traditional design for a DMM, which requires independent adjustment of the full-scale gain and zero offset for each measurement range and function.

Calibration is a process in which individual gain and offset values are adjusted, manually or electronically, to yield minimum error relative to an applied input, as shown in Fig. 1. Gain and offset calibration values are generally determined using precision ratio transfer measurements relative to a smaller set of working standards whose errors are directly traceable to national standards. In the United States, standards are kept by the National Institute of Standards and Technology (NIST), formerly the National Bureau of Standards (NBS). Dc voltages are often derived from a 1.018-volt saturated electrochemical cell known as a Weston standard cell. The output voltage is divided, or otherwise ratioed, to yield other traceable values. For example, the output would be divided by 10.18 to produce 0.1V. The ratio transfer process is, in general, different for each calibration value. It is prone to both random and systematic errors, which may propagate undetected into instrumentation through the calibration process. This calibration (or verification) uncertainty will produce a "floor" measurement error sometimes equal to or greater than the uncertainty of the instrument alone.

The objectives for two-source calibration are to reduce this floor uncertainty and to provide an independent method to increase confidence in the overall calibration process. The HP 3458A uses a highly linear analog-to-digital converter (ADC) to measure the ratio between a traceable reference and its divided output. The ADC performs the function of the precise ratio transfer device.

Sources of Error

The errors in any ratio measurement can be divided into two general types: differential errors (D) and integral errors (I). A differential error is a constant percent of full scale and is independent of the input. These errors are handled like dc offsets. An integral error is a function of the input, and the relationship is usually nonlinear. These errors are generally thought of as gain errors. The maximum total error can be expressed as:

$$E_1(x) = I(x/100\%) + D,$$

where x is the input to the ratio device and $E_1(x)$ is the error, both expressed as a percent of full scale. The general form of the error bound is shown in Fig. 2.

What is of concern is the error in the output or measured value expressed as a percent of that value. Expressed in this form, the maximum error is:

$$E_2(x) = I + D(100\%/x),$$

Where $E_2(x)$ is the total error in the output or measured value expressed as a percent of x. The general form of this error bound is shown in Fig. 3. For ratios less than one, the total error is dominated by the differential errors of the ratio transfer device. Since the differential error term is equal to the differential linearity error multiplied by one over the divider ratio, this error grows to infinity as the divider ratio grows smaller.

HP 3458A Uncertainty

The design goal for the HP 3458A DMM was for internal ratio transfer errors to be equal to or lower than those achievable with commercially available external ratio dividers. This set the total ratio measurement error (linearity) requirement for the ADC for a 10:1 transfer to approximately 0.5 ppm of output or 0.05 ppm of input.

Fig. 4 illustrates the integral and differential linearity achieved with the HP 3458A ADC design. The test data was generated using a Josephson junction array intrinsic voltage standard (see "Josephson Junction Arrays," page

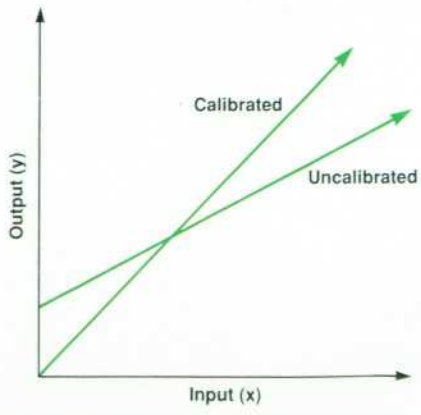


Fig. 1. *Calibrated and uncalibrated gain and offset in a measurement.*

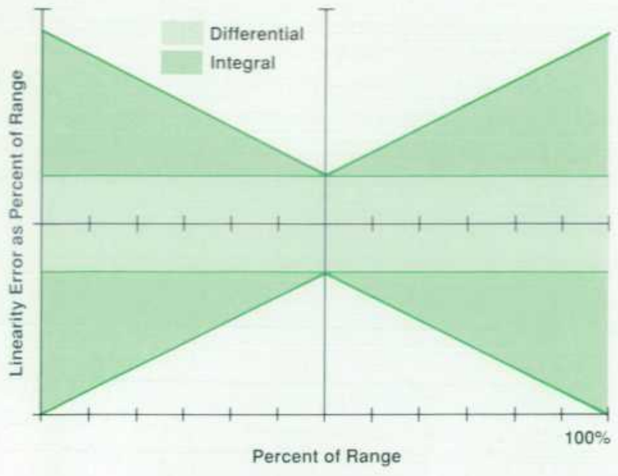


Fig. 2. *Linearity error as a percent of range.*

24). Fig. 4a shows typical deviation from a straight line for the input voltage range from minus full scale to plus full scale expressed in ppm of full scale. This expresses the test data in a form dominated by the integral linearity effects. Integral error less than 0.1 ppm of full scale was achieved. Fig. 4b shows typical test data expressed as ppm of reading (output). This data indicates differential linearity error less than 0.02 ppm of reading. For a 10:1 ratio transfer the predicted error would be approximately I + 10D or 0.3 ppm. Fig. 4c shows measured data, again using a Josephson junction array standard to characterize the error at 1/10 of full scale relative to a full-scale measured value. The data indicates a 10:1 ratio error of 0.01 ppm of the input or 0.1 ppm of the measured (output) value. This represents typical results; the specified 3σ ratio transfer error is greater than 0.3 ppm. Measurement noise contributes additional error, which can be combined in a root-sum-of-squares manner with the linearity errors.

Offset Errors

Linear measurement errors in a DMM are of two general types, offset errors and gain errors. Offset error sources include amplifier offset voltages, leakage current effects (IR), and thermocouple effects generated by dissimilar metals used in component construction or interconnection. Fig. 5 shows a simplified schematic of the dc measurement function. Switches S1 and S2 are used to provide a zero reference during each measurement cycle. Offset errors common to both measurement paths, for example the offset voltage introduced by amplifier A1, are sampled and subtracted during each measurement sequence. This is referred to as the autozero process.

Correction of the remaining offset error is achieved by

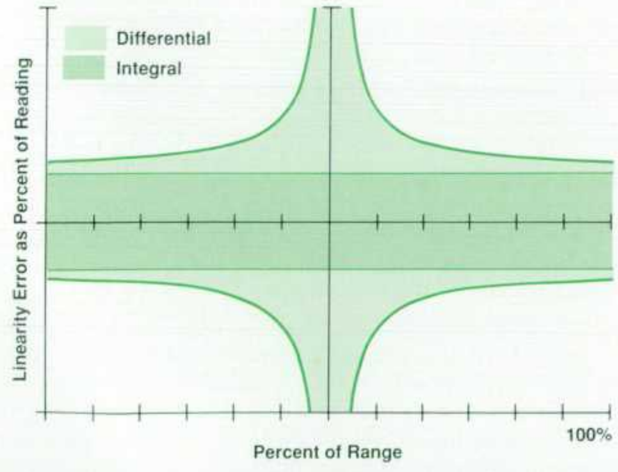


Fig. 3. *Linearity error as a percent of reading.*

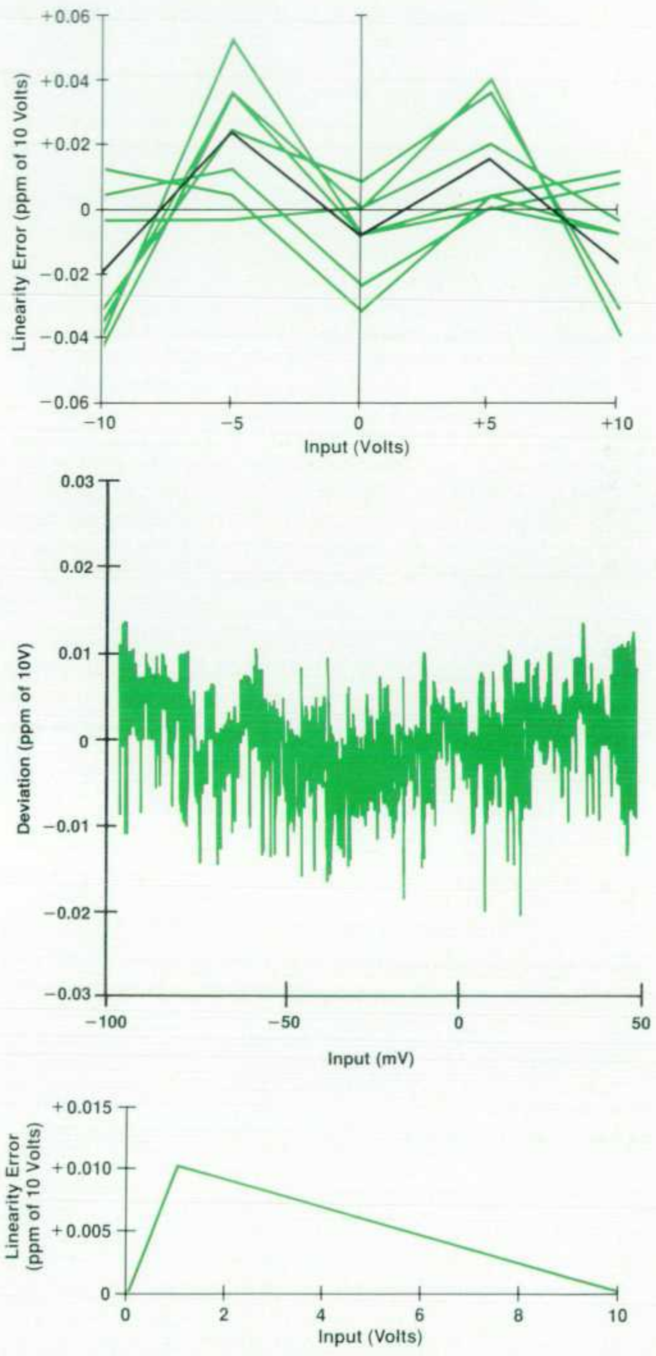


Fig. 4. *Results of HP 3458A linearity tests using a Josephson junction array. (a) Seven passes and the average result for linearity error characterization. (b) Differential linearity characteristic. (c) Linearity error for an internal 10:1 ratio transfer.*

Josephson Junction Arrays

A Josephson junction is formed by two superconductors separated by a thin insulating barrier. When cooled to liquid helium temperatures (4.2K), these devices exhibit very complex nonlinear behavior that has led to a wide range of applications in analog and digital electronics. A quantum mechanical analysis shows that these junctions generate an ac current whose frequency is related to the junction voltage by the relation $f = 2eV/h$ where e is the electron charge and h is Planck's constant. When the junction is driven by an ac current the effect operates in reverse. The junction oscillation phase locks to the applied ac current and the junction voltage locks to a value $V = hf/2e$. This phase locking can also occur between harmonics of the applied ac current and the Josephson oscillation. Thus, the junction I-V curve displays a set of constant-voltage steps (Fig. 1) at the voltages $V = nhf/2e$, where n is an integer. The Josephson junction thereby provides a means of translating the inherent accuracy of the frequency scale to voltage measurements.

In July of 1972 the Josephson effect was adopted as the definition of the U.S. legal volt. For the purpose of this definition the quantity 2e/h was assigned the value 483593.42 GHz/V. Since then, tests of the Josephson voltage-to-frequency relation have verified its precision and independence of experimental conditions to the level of a few parts in 10^{17}.[1]

The Josephson voltage standards of 1972 had only one or two junctions and could generate voltages only up to about 10 mV. This low voltage required the use of a complex voltage divider to calibrate the 1.018V standard cells used by most standards laboratories. To overcome the limitations of these low voltages,

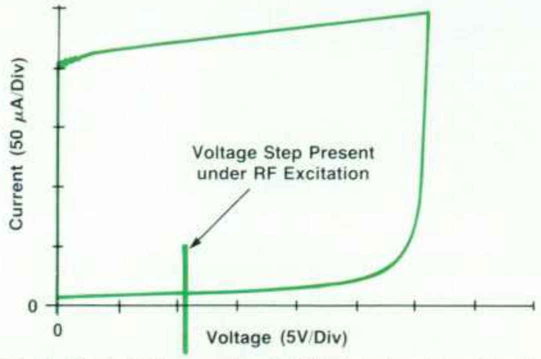


Fig. 1. *Partial I-V curve of an 18,992-junction Josephson junction array without RF excitation. Also shown is a typical I-V curve under 75-GHz excitation, which is a constant-voltage step at a voltage V = nhf/2e. The voltage V is between −12V and +12V, and is determined by controlling the bias current and source impedance to select the value of n.*

researchers at the U.S. National Institute of Standards and Technology (formerly the National Bureau of Standards), and PTB in West Germany have developed superconducting integrated circuits that combine the voltages of several thousand junctions. The most complex of these chips uses 18,992 junctions to generate 150,000 constant-voltage steps spanning the range from −12V to +12V (Fig. 2). The chip uses a finline to collect 75-GHz

providing a copper short across the input terminals. A reference measurement is taken and the measured offset is stored. Values are determined for each measurement function and range configuration. The offset is subtracted from all subsequent measurements. The HP 3458A performs all zero offset corrections by automatically sequencing through each of the required configurations and storing the appropriate offset correction during the external calibration process. These offsets are the b term in the linear equation $y = mx + b$, where y is the calibrated output result and x is the internal uncalibrated measurement. These calibrated offsets can be made small and stable through careful printed circuit layout and component selection.

Gain Errors

Gain errors in a DMM result from changes in amplifier gains, divider ratios, or internal reference voltages. Each gain term exhibits a temperature coefficient and some finite aging rate, and can potentially change value following exposure to high-humidity environments or severe shock or vibration. Periodically, known values close to the full scale of each measurement function and range are applied to the DMM to calibrate the gain ratio m such that $y = mx + b$ is precisely equal to the known input value, y. However, even after gain calibration, a DMM can easily be exposed to conditions that may introduce new errors. The HP 3458A DMM implements a special method for self-adjusting all instrument gain errors and many offset errors relative to its own internal references.

DC Calibration

Calibration of the dc function begins by establishing traceability of the internal voltage reference. The internal 7V Zener reference (see "A High-Stability Voltage Reference," page 28) is measured relative to an externally applied traceable standard. A traceable value for this internal reference is stored in secure calibration memory until the next external calibration is performed. Next, the gain of the 10V range is determined by measuring the internal 7V reference on this range. The gain value is stored in secure autocalibration memory. This gain value can be recomputed at any time by simply remeasuring the internal 7V reference. The stability, temperature coefficient, and time drift errors of the internal 7V reference are sufficiently small (and specified) compared with other gain errors that remeasurement or autocalibration of these gains will yield smaller measurement errors in all cases. Adjustment of the full-scale gain values of all other ranges relies on the precise ratio measurement capabilities of the HP 3458A ADC as demonstrated in Fig. 4c. For the 1V-range gain adjustment, the traceable internal 7V reference is divided to produce a nominal 1V output. The exact value of this nominal 1V is measured on the previously adjusted 10V measurement range at approximately 1/10 of full scale. The measured value, a ratio transfer from the internal 7V reference, is used to adjust the gain of the 1V range of the dc voltage function. This gain value is again stored in secure autocalibration memory. Neither the precise value nor the long-term stability of the nominal 1V internal source is important. The internal 1V

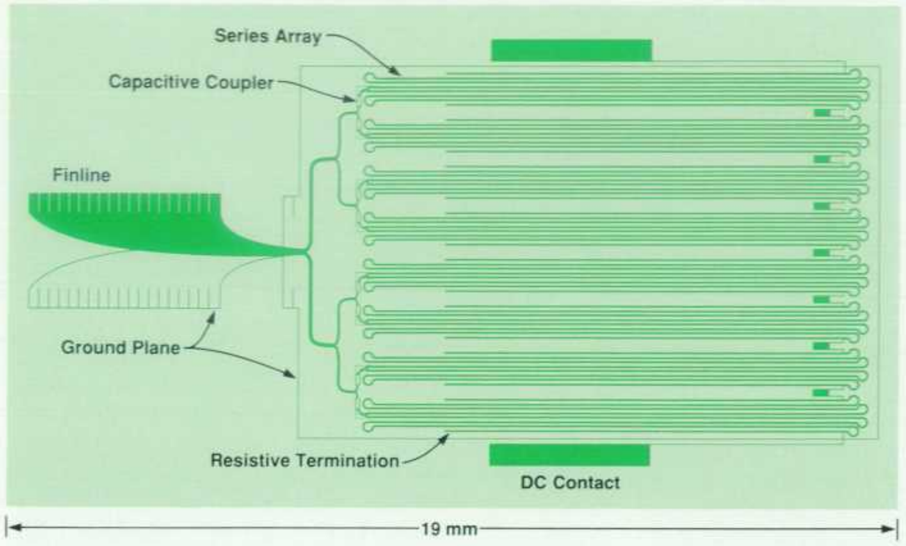


Fig. 2. *The layout for an 18,992-junction voltage standard array capable of generating voltage steps in the range of −12V to +12V. The horizontal lines represent 16 striplines, each of which passes through 1187 junctions. The junctions are too small to be distinguished on this drawing.*

power from a waveguide and direct it through a set of power splitters to 16 striplines, each of which passes through 1187 junctions. A network of high-pass and low-pass filters allows the microwave power to be applied in parallel while the dc voltages add in series.[2]

In operation, the array is cooled to 4.2K in a liquid-helium dewar. A Gunn-diode source at room temperature provides the required 40 mW of 75-GHz power. It is possible to select any one of the 150,000 constant-voltage steps by controlling the bias current level and source impedance. A continuous voltage scale can be obtained by fine-tuning the frequency. The accuracy of the voltage at the array terminals is equal to the accuracy of the time standard used to stabilize the Gunn-diode source. Actual calibrations, however, are limited by noise and thermal voltages to an accuracy of a few parts in 10^9.

The ability to generate exactly known voltages between −12V and +12V can eliminate the problems and uncertainties of potentiometry from many standards laboratory functions. For example, Josephson array standards make it possible to perform absolute calibration of voltmeters at levels between 0.1V and 10V without the uncertainty of a resistor ratio transfer from standard cells. Another application is the measurement of voltmeter linearity with an accuracy higher than ever before possible.

References

1. R.L. Kautz and F.L. Lloyd, "Precision of Series-Array Josephson Voltage Standards," *Applied Physics Letters*, Vol. 51, no. 24, December 1987, pp. 2043-2045.
2. F.L. Lloyd, C.A. Hamilton, K. Chieh, and W. Goeke, "A 10-V Josephson Voltage Standard," *1988 Conference on Precision Electromagnetic Measurements*, June 1988, Tokyo.

John Giem
Development Engineer
Calibration Laboratory
Loveland Instrument Division

source must only be stable for the short time required to perform the two measurements of the transfer.

Each of the remaining dc voltage ranges is automatically gain adjusted relative to the internal 7V reference through a similar sequence of full-scale-to-1/10-full-scale transfer measurements. All gain errors can then be readjusted relative to the internal reference to remove measurement errors at any later time. The only gain error that cannot be adjusted during autocalibration is the time and temperature drift of the internal 7V reference.

Ohms and DC Current Calibration

Calibration of the ohms functions is similar to that of the dc voltage function. Traceability for the internal 40-kΩ reference resistor is established first. The internal reference resistor is measured relative to an externally applied traceable 10-kΩ standard resistor. The traceable value for this internal reference is stored in secure calibration memory until the next external calibration is performed. Resistance measurements are made by driving a known current I through an unknown resistance R and measuring the resultant voltage V. The unknown resistance value R is computed from Ohm's law, R = V/I. Since the dc voltage measurement function has been previously traceably adjusted, only the values of the ohms current sources (I) need be determined to establish calibration.

Adjustment of the ohms current source values begins by applying the nominal 100-microampere current source (10-kΩ range) to the traceable 40-kΩ internal resistance standard. The value of the current source is computed from the traceable measurements and stored in secure autocalibration memory. The 100-μA current source can be remeasured (autocalibrated) at any time to correct for changes in its value. Residual errors in this autocalibrated measurement are reduced to those of the internal reference resistor and the autocalibrated error of the 10V dc voltage range—essentially the drift of the internal voltage reference. For resistance measurements, only drift in the internal resistance reference will affect measurement accuracies. The gains of the voltage measurements V and the current sources I, which are derived from the internal voltage reference, will also change as this reference drifts, but the computed value for R is not affected since the V/I ratio remains unchanged.

The known 100-μA current, its value determined in the previous step, is next applied to an internal 5.2-kΩ resistor (an internal 10-to-1 ratio transfer measurement). The value of this resistor is determined, again from Ohm's law. This new resistor R is computed (R = V/I) from the 100-μA current previously determined relative to known traceable standards and the previously calibrated dc voltage function. The value of this resistor is stored in autocalibration memory. This resistor is actually the 10-μA dc current function shunt resistor. With the shunt resistor R traceably determined, traceable dc current measurements can be computed from Ohm's law, I = V/R.

Now that the 5.2-kΩ internal shunt resistor is known, the 1-mA ohms current source (1-kΩ range) is applied and its value computed as a ratio relative to the 100-μA current source value. The 1-mA current source value is stored in autocalibration memory. This combined ohms current source and dc current shunt resistor ratio transfer process continues until all six currents and all eight shunt resistors are known relative to the two external standards.

As we set out to show, all gain errors for dc voltage, ohms, and dc current measurements have been traceably adjusted relative to only two external standard values: 10V dc and 10 kΩ. Table I summarizes the HP 3458A errors for the internal ratio transfer measurements described so far.

Table I
Internal Ratio Transfer Errors

External 10V dc	→ Internal 7V	0.03 ppm
Internal 7V	→ 10V dc	0.02 ppm
10V dc	→ 1V dc	0.33 ppm
External 10 kΩ	→ Internal 40 kΩ	0.30 ppm
Internal 40 kΩ	→ 100 μA	0.15 ppm
Internal 40 kΩ	→ Internal 5.2 kΩ	0.50 ppm
100 μA	→ 1 mA	0.50 ppm

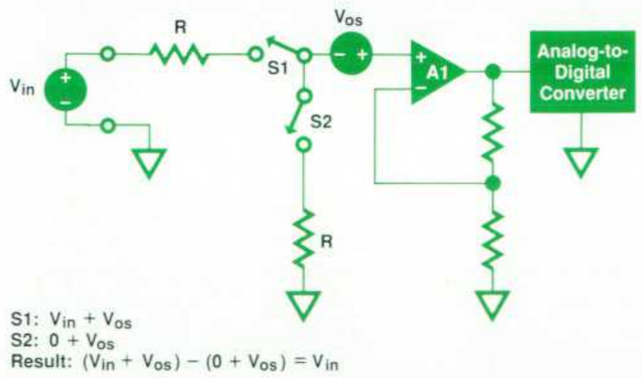
Additional Errors

Gain and offset variations are the dominant sources of measurement error in a DMM, but they are by no means the only sources of error. Measurement errors are also produced by changes in leakage currents in the input signal path. These may be dynamic or quasistatic leakages. A more complete schematic of the input circuit of the HP 3458A is shown in Fig. 6. Recall that switches S1 and S2 are used to null the dc offsets of amplifier A1 and its input bias current. However, the capacitance C1 causes an error current I_{err} to flow when S1 is turned on. This current, sourced by the input, generates an exponentially decaying error voltage $I_{err}(R + R_i)$. If R_i is large, as it is for ohms measurements, significant measurement errors can result.

These errors can be reduced by providing a substitute source (shown in the shaded section of Fig. 6) to provide the charging current for the parasitic capacitance C1. Amplifier A2 follows the input voltage so that when switch S3 is turned on between the S2 and S1 measurement periods, C1 will be precharged to the input voltage. Second-order dynamic currents flow because of the gate-to-drain and gate-to-source capacitances of the switches, which are FETs. The HP 3458A performs complementary switching to minimize these effects. During an autocalibration, the offset of buffer amplifier A2 is nulled and the gain of the complementary switching loop is adjusted to reduce errors further.

High ohms measurements are particularly sensitive to parasitic leakage currents. For example, 10 ppm of error in the measurement of a 10-MΩ resistor will result from a change of 5 pA in the 500-nA current source used for the measurement. Over the 0°C-to-55°C operating temperature range a 5-pA change can easily occur. During autocalibration, which can be performed at any operating temperature, several internal measurements are performed with various hardware configurations. The results are used to solve simultaneous equations for leakage current sources. Knowing these leakage currents allows precise calculation of the ohms current source value for enhanced measurement accuracy.

Many other errors are also recomputed during autocalibration. Autocalibration can be performed in its entirety or in pieces (dc, ohms, or ac) optimized for particular measurement functions. The dc voltage autocalibration, for example, executes in approximately two minutes. The autocalibration process for the ohms functions, which also calibrates the dc current function, takes about eight minutes to complete. If the user is only concerned with correcting errors for dc or ac measurements, the ohms autocalibration



R
V_{os}
V_{in}
S1
A1
S2
Analog-to-Digital Converter
R
S1: $V_{in} + V_{os}$
S2: $0 + V_{os}$
Result: $(V_{in} + V_{os}) - (0 + V_{os}) = V_{in}$

Fig. 5. *Simplified schematic of the dc voltage measurement function.*

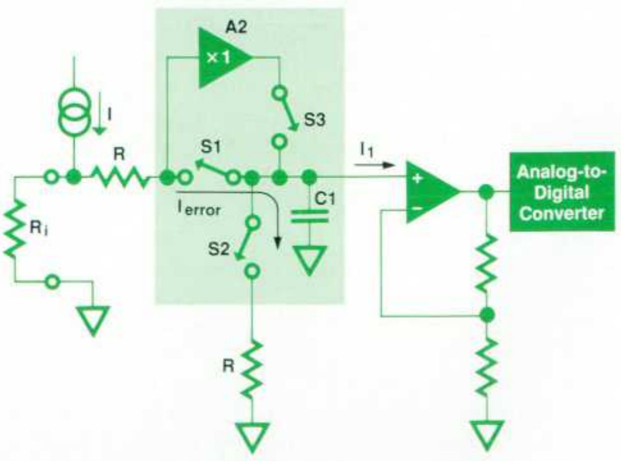
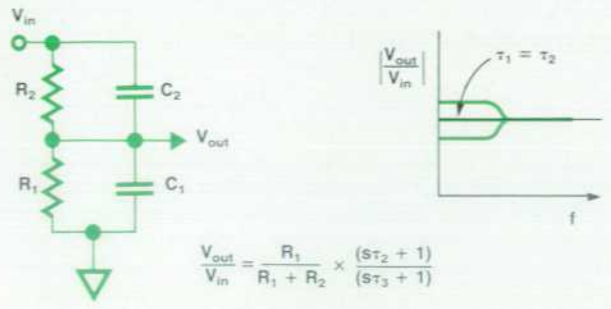


Fig. 6. *A more complete schematic of the HP 3458A input circuit.*



$$\frac{V_{out}}{V_{in}} = \frac{R_1}{R_1 + R_2} \times \frac{(s\tau_2 + 1)}{(s\tau_3 + 1)}$$

$$\tau_1 = R_1C_1 \qquad \tau_2 = R_2C_2$$

$$\tau_3 = \left(\frac{R_1R_2}{R_1 + R_2}\right)\left(C_1 + C_2\right)$$

Fig. 7. *RC attenuator gain-versus-frequency characteristic.*

sequence can be omitted to save time.

AC Frequency Response Calibration

The goals for self-calibration of the HP 3458A extended beyond the dc measurement functions. Just as the concept of sampling a signal and digitally computing its true-rms value goes against traditional DMM methods, so does the idea of adjusting the frequency response and gain of an ac voltmeter without applying external ac calibration sources. Normally, the first step in the calibration of an ac voltmeter would be to adjust the instrument for constant gain at all frequencies. This frequency flatness correction is generally performed by manually adjusting either resistive or capacitive circuit components. Resistive components usually determine gains at lower frequencies and capacitive components usually determine gains at higher frequencies. The frequency response characteristic of the HP 3458A ac measurement function is dominated by five compensated RC divider networks, which are used to condition the input signal for each measurement range. The gain-versus-frequency characteristic of an RC attenuator circuit is shown in Fig. 7. When the attenuator is properly compensated ($\tau_1 = \tau_2$), the resulting divide ratio is a frequency independent constant determined solely by the resistive elements.

It can be shown using Fourier transforms that if the input to a linear circuit is a perfect voltage step and the output of the same circuit is also a perfect voltage step, then the circuit transfer function is constant with frequency. The hardware used to implement the digital ac measurement technique of the HP 3458A is also used to sample a step output of the RC attenuator. The sampled data is used to compensate the internal RC divider networks for flat gain versus frequency without external inputs.

A simplified schematic for the 10V ac measurement range is shown in Fig. 8. The active compensation of the divider network is achieved by generating a "virtual trimmer" circuit element to allow the adjustment of the divider time constants. The trimmer is a programmable-gain bootstrap amplifier connected across resistor R1. The variable-gain amplifier allows control of the voltage across R1, effectively varying R1's value. The resistive divider ratio can be electronically servoed to match the fixed capacitive divider ratio given a measurable error function. The servo error signal is generated by applying an extremely square voltage step to the network. The step output is sampled at least twice. An amplitude difference between samples indicates the presence of an exponential component resulting from miscompensation of the attenuator. The digitally controlled loop servos the difference signal to adjust the virtual trimmer to achieve precise cancellation of frequency dependent errors. Sample times can be optimized for maximum sensitivity to the attenuator time constant RC, thus improving servo-loop rejection of second-order time constants resulting from capacitor dielectric absorption or other parasitic effects.

Sampling of the voltage step uses the same internal tools required to perform the digital ac measurement function. The flatness autocalibration voltage step is sampled with the integrating ADC configured for 18-bit measurement resolution at 50,000 conversions per second. An internal precision sampling time base is used to place samples with 100-ns resolution and less than 100-ps time jitter. Fig. 9 shows the range of attenuator output waveforms present during frequency flatness autocalibration. When the attenuator is compensated correctly, the output waveform will closely resemble an ideal voltage step as shown. Test data has shown that the automated compensation yields less than 50 ppm of frequency response error from dc to 30 kHz. Autocalibration of the frequency response will correct for component changes caused by temperature, humidity, aging, and other drift mechanisms. Correction

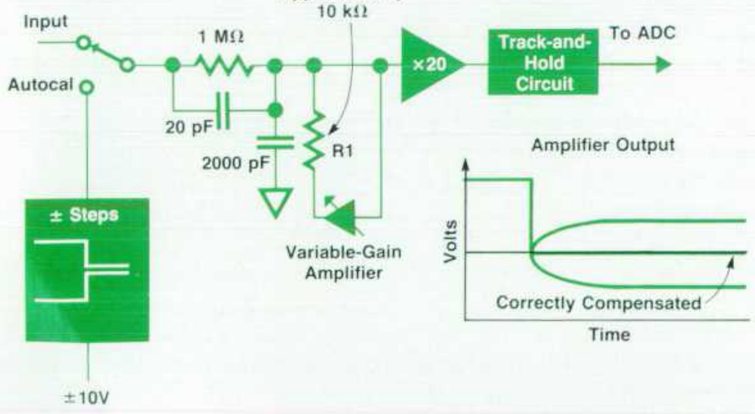


Fig. 8. *Simplified schematic of the 10V ac measurement range.*

A High-Stability Voltage Reference

Autocalibration in the HP 3458A Digital Multimeter is a process of transferring the gain accuracy of a single voltage reference to all measurement gains. The design goal for the internal reference of the HP 3458A was to provide long-term stability and temperature stability comparable to external standards that would normally be used to calibrate an 8½-digit multimeter. These goals were achieved by using a temperature-stabilized solid-state Zener reference. Without temperature stabilization, the Zener's voltage drift with temperature is approximately 50 ppm/°C. A proportional temperature control loop senses the chip temperature of the reference device and reduces this drift to less than 0.15 ppm/°C.

The long-term drift of each voltage reference assembly is measured by an automated drift monitoring and screening process. Reference assemblies, including the temperature controller, are monitored until the aging rate is shown to be less than the 8 ppm/yr stability requirement of the HP 3458A. Summarized test data for a number of 8 ppm/yr reference assemblies is shown in Fig. 1. Monitoring the references for additional time allows the selection of assemblies that exhibit aging rates less than 4 ppm/yr for the high-stability option.

David E. Smith
Development Engineer
Loveland Instrument Division

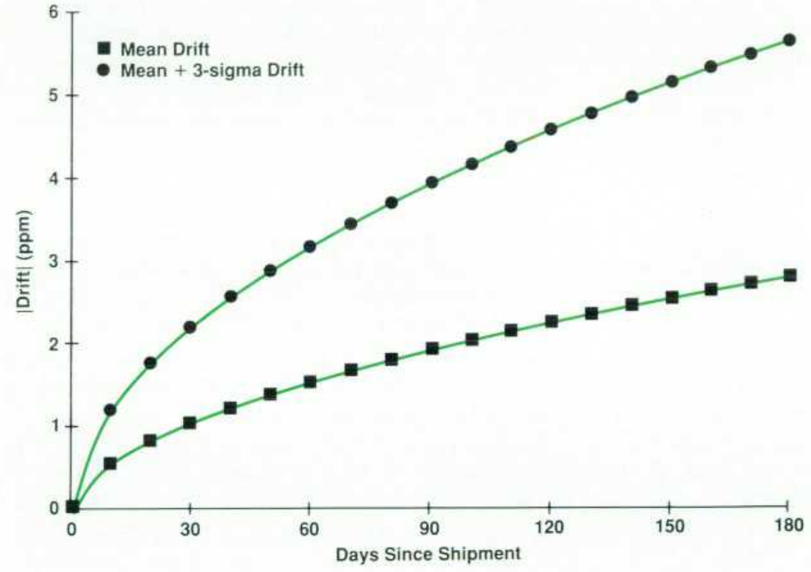


Fig. 1. *HP 3458A internal voltage reference drift distribution.*

of these errors allows a single specification to apply for extended operating conditions.

AC Gain Calibration

Once the frequency flatness characteristics are adjusted, the second step of calibration can be completed. Gain correction for the measurement must still be achieved. In Fig. 7 it can be seen that when frequency compensation is achieved, the attenuator gain can be established equally well at any frequency as long as the calibration signal amplitude is precisely known. Adjustment of the circuit gain using a dc signal is convenient since a traceably calibrated dc voltage reference and a dc voltage measurement function are available. Gain adjustment of the ac measurement function using known dc voltages allows complete autocalibration of ac measurement accuracy in much the same manner as the dc voltage measurement function.

Several mechanisms can limit the accuracy of a dc gain adjustment. Dc offsets or turnover errors can be minimized by performing gain adjustment calculations using known positive and negative voltages. Errors caused by white noise are reduced by averaging 40,000 samples for each voltage measurement made through the wide-bandwidth track-and-hold circuit. Low-frequency 1/f noise is minimized by chopping these 40,000 readings into groups of 1000, each group sampling alternating polarities of the known internal

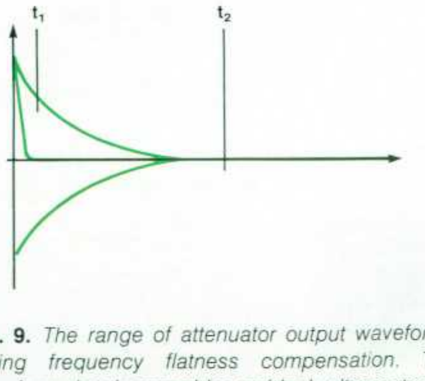


Fig. 9. *The range of attenuator output waveforms present during frequency flatness compensation. The output waveform closely resembles an ideal voltage step when compensation is correct.*

dc calibration voltages. This voltage chop is performed at a fast enough rate to achieve maximum cancellation of the 1/f noise voltage. A final error mechanism results from aliasing of internal spurious signals. The internal 10-MHz clock signal tends to be present in small amounts everywhere. The ac signal path and the track-and-hold circuit (2-ns sample aperture) each have sufficient bandwidth to couple the internal clock into measurements. If the sample spacing is a multiple of the 100-ns clock period, the internal spurious clock will be aliased or mixed down to contribute a dc offset in the measurement. A 100-μV-peak spurious clock signal can lead directly to a 100-μV error in measuring the internal dc calibration signal as shown in Fig. 10. The HP 3458A uses a random sampling time base mode during this calibration sequence. The time base generates randomly spaced sample intervals with a resolution of 10 ns. The chopped groups of random samples, 40,000 in all, are averaged together to obtain the net gain of the divider. Errors caused by dc offsets, white noise, 1/f noise, and clock aliasing are reduced using this internal calibration algorithm. Gain calibration of the ac measurement function relative to the internal dc reference is accomplished with less than 10 ppm error for intervals extending to two years. The residual dc gain calibration error will limit the absolute measurement accuracy for low-frequency inputs.

Additional Errors

Besides adjusting the frequency response and gain of each ac measurement range, other corrections are performed during autocalibration. Offset voltage corrections are determined for each ac amplifier configuration. The offset of the analog true-rms-to-dc converter is determined. The offset of the analog trigger level circuit is nulled. Internal gain adjustments for various measurement paths are performed. For example, the track-and-hold amplifier gain is precisely determined by applying a known dc voltage and measuring the output in track mode using 7½-digit internal dc measurements. A gain ratio is computed using this measurement and the hold mode gain is determined by averaging 40,000 samples using the 6-μs, 50,000-reading-per-second, 18-bit conversion mode of the integrating ADC. This gain is critical to the accuracy of the digitally computed rms ac voltage function and to the wideband sampling functions. Ac current measurements use the same shunt resistors as the dc currents. A differential amplifier is used to sample the voltage across the shunt resistors for ac current measurements, and the gain of this amplifier is computed during autocalibration.

As a result of autocalibration, the ac measurement accuracy of the HP 3458A is unchanged for temperatures from 0°C to 55°C, for humidity to 95% at 40°C, and for a period of two years following external calibration. Execution of only the ac portion of the autocalibration process is completed in approximately one minute.

Summary

Two-source calibration of a state-of-the-art digital multimeter provides several benefits:

- Increased process control within the standards labora-

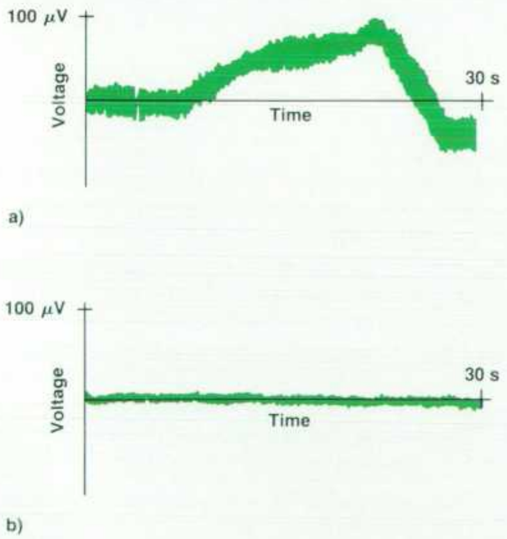


Fig. 10. *(a) Using the clock-derived time base, a 100-μV spurious clock signal can lead directly to a 100-μV error in measuring the internal dc calibration signal. (b) The HP 3458A uses a random sampling time base mode to eliminate this error source.*

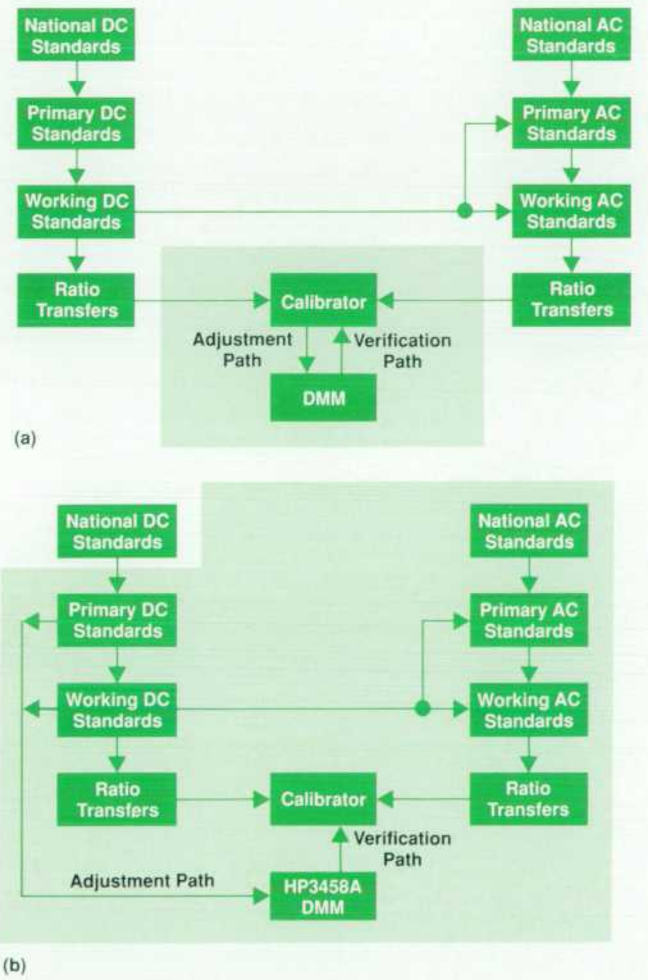


Fig. 11. *(a) Traditional calibration chain for dc and ac voltage. (b) HP 3458A calibration chain, showing the increased verification confidence that results from internal calibration.*

tory through the independent ratio transfers of the DMM
- Reduced calibration time
- Increased measurement accuracies in real environments
- Increased confidence through complete self-testing.

The greatest benefit of two-source calibration is seen not by the instrument end user but by the calibration facility supporting those instruments. Fig. 11 shows the normal instrument and two-source calibrated instrument traceability chain. When verifying the results of the two-source calibration process, the metrologist now has the independent checks of the HP 3458A to catch inadvertent human errors in the normal process. Technique, cabling, and other instruments used in the generation of calibration values are no longer open-loop errors that may propagate through a calibration laboratory. Two-source calibration can identify errors anywhere within the traceability chain, from primary standards to final values.

The HP 3458A autocalibration procedures are also performed during the instrument self-test, which takes about one minute. The only difference is reduced averaging of the internal results for faster execution. Also, the results are not retained in memory afterward. The self-test procedures perform highly accurate measurements on each range of each function, thereby providing a comprehensive analog and digital confidence test of the system.

Acknowledgments

Verification of the two-source calibration performance of the HP 3458A was not a particularly easy process. Identifying and quantifying errors, normally a fairly straightforward proposition, was made difficult by both the high accuracy and the independent nature of two-source calibration. Many people have contributed to the success of these efforts. Particular thanks are given to Bill Bruce, the Loveland Instrument Division standards lab manager, and his staff. Their assistance and guidance were of immeasurable help. They were disbelieving and skeptical initially, but in the end, their ardent support was greatly appreciated. Kudos go to our production engineering test development group of Bert Kolts and Randy Hanson for developing a fully automated system for performance verification testing of the HP 3458A. Last and most important, special thanks to the staff of the U.S. National Bureau of Standards: to Dick Harris and Clark Hamilton at NBS Boulder, Colorado for allowing us access to their Josephson junction array during our product development and for helping us set up our own Josephson array voltage standard system. Thanks also go to Neil Oldham of NBS Washington, D.C. for making available his transportable digital ac reference standard, also for performance verification testing.

Design for High Throughput in a System Digital Multimeter

High-speed custom gate arrays, microprocessors, and supporting hardware and a substantial investment in firmware design contributed to the design of the HP 3458A DMM as a system for moving data efficiently.

by Gary A. Ceely and David J. Rustici

MANUFACTURERS OF ELECTRONIC and other types of products have learned that high test system throughput is vital to maintaining production capacity. As a primary component of automated test and data acquisition systems, the system digital multimeter (DMM) has become a major factor in determining system throughput. A DMM must not only be able to take and transfer high-speed bursts of readings, but must also have the ability to reconfigure itself quickly when measuring several different parameters in rapid succession.

Historically, DMM performance has been hindered by a number of factors, such as relay switching times, ADC conversion delays, and the limited processing power of early-generation microprocessors. In addition to controlling the ADC hardware, taking and transferring readings, and parsing commands, the microprocessor has been saddled with scanning the front-panel keyboard, updating the display, and polling various peripheral ICs to monitor and update status information. Increasing demands on the capabilities of firmware written for these machines have only compounded the problem. Adoption of more English-like programming languages has added greatly to both bus overhead (because of the length of these commands) and parsing time, which formerly was a minor factor.

Another performance limitation in system DMMs has resulted from the need to make floating measurements, that is, measurements referenced to the **LO** terminal instead of earth ground. Since the **LO** terminal may be raised to a potential several hundred volts above ground, the ADC hardware must also float with this voltage. The problem here is that the HP-IB (IEEE 488, IEC 625), and therefore the hardware that interfaces to it, is earth-referenced, requiring that the ADC hardware be isolated from the controlling microprocessor. In many cases, the ADC hardware is designed around a second microprocessor which communicates with the main microprocessor via an isolated serial link, forming a bottleneck in high-speed ADC programming and data transfers.

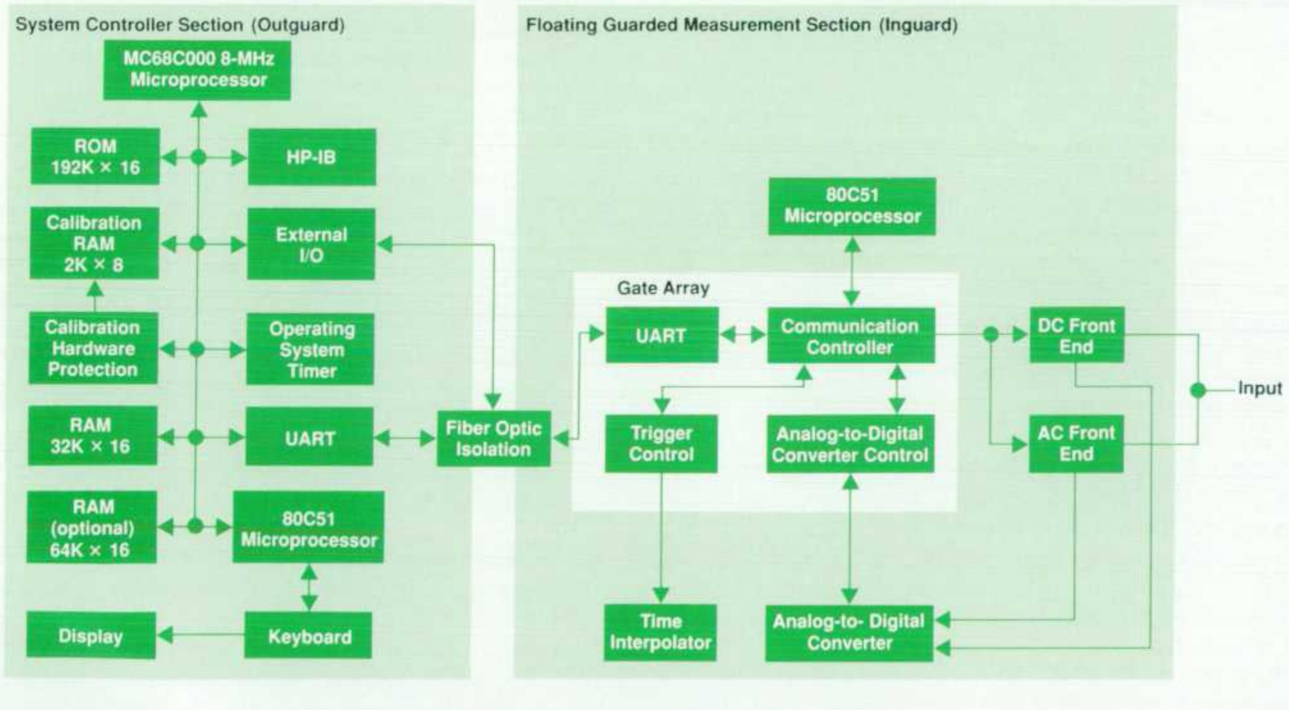


Fig. 1. *HP 3458A Digital Multimeter system block diagram.*

Considering the history of DMM performance, it becomes obvious that the design of the instrument as a system in itself is critical to the performance of the surrounding automatic test system as well. Two key design goals for the HP 3458A were that it be able to reconfigure itself and take a reading 200 times per second, and that it be able to take and transfer readings (or store them internally) at a burst rate of 100,000/s. To achieve these goals, system design for the HP 3458A focused on expediting the flow of data through the instrument, both in the hardware and in the firmware.

Design Overview

A simplified block diagram of the HP 3458A is shown in Fig. 1. Like previous designs, the DMM is divided into two sections, inguard and outguard, which correspond to the hardware inside and outside of the guarded (isolated) section of the DMM. In this design, however, the bottleneck of the serial interface between the two sections is overcome by the use of a high-speed (5 Mbits/s) fiber optic data link and custom gate arrays on each end to decode and buffer received data.

Performance features on the outguard side include an 8-MHz MC68C000 main processor, high-speed RAM and ROM (requiring no wait states from the processor), a separate 80C51 microprocessor to control the front-panel interface, and a programmable timer used as an operating system clock. This represents a significant upgrade in the outguard hardware over previous 6800-based designs, and not only yields faster execution of instructions, but also frees the main processor from polling peripherals, since all I/O and interprocessor communications are now interrupt-driven. Additional gains are realized through the use of a double-buffered HP-IB input scheme (the parser reads data from one buffer while an interrupt service routine fills the other) and a hardware HP-IB output buffer, which allows the main processor to write data to the HP-IB in words (16 bits) instead of bytes (8 bits).

Outguard RAM is divided into three sections: an EEPROM for storing calibration constants, standard RAM (nonvolatile), and optional RAM (volatile). Calibration RAM is distinct from the rest of RAM because it is protected from accidental overwrites by a hardware mechanism that also makes writing to it rather slow. Standard RAM is divided into program memory, reading memory (10K 16-bit readings), state storage, and system overhead (stacks, buffers, etc.). Nonvolatile RAM is used here to protect stored instrument states, subroutines, and user key definitions. Optional RAM is available only as additional reading storage (64K readings).

Inguard hardware is also under microprocessor control (an 80C51, in this case), but the heart of the inguard section is a 6000-gate, 20-MHz CMOS gate array. Functions performed by the gate array include communications with the outguard section through a custom UART, trigger logic control, analog-to-digital conversion, and communications between the UART and other parts of the inguard section. Shift registers are incorporated to minimize the number of interconnections between the gate array and other inguard circuits (the ADC, the ac and dc front ends, and the trigger control logic). Five shift registers containing 460 bits of information reduce the number of interface lines to just three per circuit. Communications are directed by the processor, which also interprets messages sent from the outguard section and generates response messages (see "Custom UART Design," page 36.

Firmware Structure

The division of tasks between the inguard and outguard processors is based on the need to minimize the flow of messages between them. Inguard firmware is responsible for controlling the ADC measurement sequence, controlling the trigger logic during measurements, and directing configuration data to the other inguard circuits. Outguard firmware responsibilities are as shown in Fig. 2. Primary functions, such as parsing, command execution, display updating, and keyboard input are performed by separate tasks under operating system control. Other functions, such as HP-IB I/O and interprocessor communications, are interrupt-driven, are coded in assembly language for maximum speed, and communicate with the primary tasks via signals and message exchanges. High firmware throughput is achieved by focusing on optimization of time-intensive tasks, such as data transfer and manipulation, parsing and execution of commands, task switching overhead, and the measurements themselves.

Fig. 3 shows the flow of data through the HP 3458A. Data flow is divided into two main paths: the input path for messages received from the controller, and the output path for measurements generated by the instrument. When a controller sends a command such as DCV 10, the data flow is from the controller to the HP 3458A through the HP-IB. The HP-IB handler accepts incoming data and passes it on to the outguard processor's parser, which interprets the command and then passes control to an execution routine. After determining the necessary actions, the execution routine sends state change data to RAM and inguard-bound messages to the UART. Messages sent to the inguard section are of two types: measurement messages, which control the type of measurement (e.g., dc voltage or ac voltage), and configuration messages, which define the state of the front ends and the ADC and timer control circuits. Data is received by the inguard UART and passed to the inguard processor, which parses the message and either acts upon it or directs it through the communication controller to one of the other inguard circuits. Once the configuration phase is complete, the ADC is ready to take a reading, and throughput becomes a matter of getting the reading out of the instrument quickly. Referring again to Fig. 3, the output data path is from the ADC to the inguard UART, through the fiber optic link, and on to the outguard processor. The processor performs any required math and formatting operations, and then directs the data either to reading storage or to the HP-IB.

Data Input, Configuration, and Measurements

Programming commands coming in over the HP-IB are received and buffered by an interrupt service routine, which in turn signals the HP-IB parser/execution task. The interrupt code is designed to continue reading characters from the HP-IB chip as long as they continue to come in at a rate of 100 μs/character or faster. In this manner, an

Firmware Development System

Firmware for the HP 3458A DMM was developed on four HP 9000 Computers (Models 320 and 350) under the HP 64000-UX microprocessor development environment. Each system was fully equipped to operate as an independent development station, and the systems were networked to facilitate transfer of code revisions (see Fig. 1). A fifth station was used for consolidating code modifications to be tested using a prototype HP 3458A and the HP 3458A production test system. After passing an extensive battery of tests, code was released in EPROM form for other prototype instruments.

Firmware tasks were divided along lines intended to minimize interdependence between the designers. The areas of responsibility were (1) measurements and calibration, (2) digitizing, (3) data processing, formatting, and storage, and (4) parsing, I/O, and operating system overhead. Fig. 2 shows a breakdown of the amount of object code generated by various modules. Al-

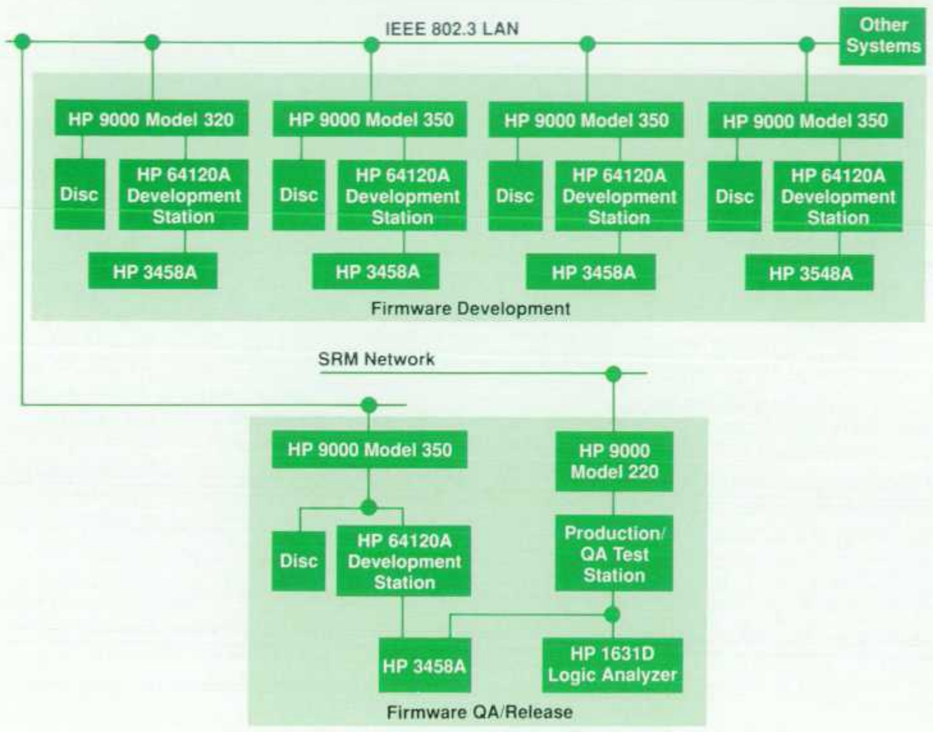


Fig. 1. *HP 3458A firmware development and QA regression test systems.*

entire command or string of commands can be read in during a single invocation of the interrupt routine, thereby generating only one signal to the parser task. In reality, two input buffers are used: one that is filled by the interrupt routine, and another that is read by the parser task. After the interrupt routine signals the parser that data is present in one buffer, that buffer belongs to the parser task, and the other buffer is used for the next command that comes in. When the parser empties a buffer, that buffer is freed for later use by the interrupt routine. Using two buffers simplifies pointer manipulation so that data can be read in and passed to the parser quickly.

To maximize the flow of data to the HP-IB parser/execution task, the instrument must first be programmed to an idle state (e.g., using TARM HOLD). This allows the operating system to keep the HP-IB parser task active so that no task switching is necessary when an HP-IB command is received. The parser is a table-driven SLR (simple left-right) design, with all critical components coded in assembly language. Simple commands can be parsed in as little as 1 ms; longer commands take as much as 3 ms. For a further increase in system throughput, command sequences can be stored as subprograms, in which case they are first compiled into assembly language by the parser/code generator. Executing command sequences in this fashion eliminates most of the overhead of bus I/O and parsing and allows the HP 3458A to perform reconfiguration and trigger operations almost twice as fast as the same sequence with individual commands (340/s instead of 180/s).

In many situations, the HP 3458A will be reconfigured for a different measurement setup with each test, which may include only one measurement. The setup changes in these cases may take more time than the measurement, so the configuration time must be minimized. To perform 180 reconfiguration and trigger operations per second, the instrument must be able to transfer, parse, and execute a command in slightly over 5 ms. Of this total, several hundred microseconds are spent in bus transfer and system overhead, and up to 3 ms may be spent parsing the command. Given that an additional several hundred microsec-

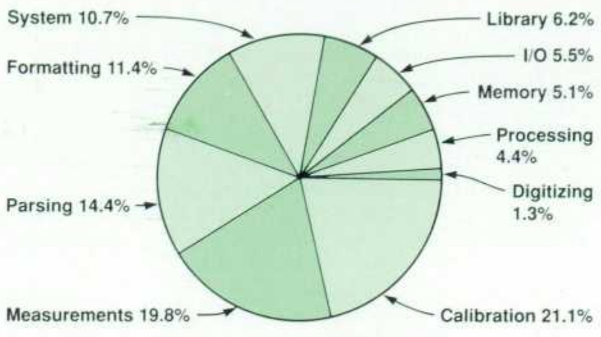


Fig. 2. *Outguard firmware modules.*

together, over 28,000 lines of C code were written, representing roughly 80% of the 356K bytes of object code generated. The remainder (12,000 lines) was written in 68000 assembly language.

During the most intense period of firmware development, code revisions were released on a weekly basis. To relieve the firmware team of the time-consuming task of generating and testing code revisions, a fifth team member was given this responsibility. Firmware designers uploaded source code weekly to the fifth system, where it was compiled, linked, and downloaded to an emulator. Having source code available on this system made it possible to trace and analyze defects using a dedicated QA system to reproduce them. The fifth development system was also used for archiving firmware revisions using RCS (UNIX revision control system). To reduce duplication of effort, the test system used for firmware development was a replica of the HP 3458A production test system, which had been developed earlier in the project cycle to be used in environmental testing and prototype characterization.

As the firmware construction phase neared completion, two engineers were added to the project so that test software could be developed in parallel with the firmware effort. To save test writers the trouble of learning the details of test system operation, drivers and utilities were written that allowed each new test to be written as an isolated subroutine. The test system executive simply loaded and ran each test as it was needed, thereby providing an efficient mechanism for adding new tests throughout the construction and test phases. Both hardware and firmware designers wrote tests for the test suite. Each was assigned a specific area of functionality to be tested, using both white-box and black-box approaches.

In addition to the tests written specifically to verify firmware operation, each revision of code was subjected to the production test software (which mainly tested the analog hardware for measurement accuracy). Additional test coverage included the entire HP 3458A user's manual, with emphasis on the command reference, example programs, and randomly generated combinations of valid and invalid syntax. As defects were found, they were fixed and the test code run again for verification. Following a successful run through the test suite, code was released and source code was saved using RCS. Saving old code revisions enabled the firmware team to recreate earlier code revisions to help track down defects that may not have been reproducible on a newer code revision. When a new defect was found, tests were written and added to the test suite to ensure that the defect would not recur. By the end of the project, the test suite had grown to where 12 hours were required to run all tests. To assess testing progress and effectiveness, defects were submitted to HP's DTS (defect tracking system). Metric reports were generated and analyzed on a weekly basis to help assess the firmware status.

Victoria K. Sweetser
Development Engineer
Loveland Instrument Division

onds will be spent taking and transferring the reading, only about 1 ms is left for the execution of the command. In this millisecond, the execution routine must range-check parameters, calculate the gain and offset values, and configure the trigger controller, the ADC, the front-end hardware, and the inguard processor. In the worst case, performing these operations takes considerably longer than a millisecond. A complete configuration of all the inguard sections takes 1.4 ms, and settling time for the front-end relays adds another 1.3 ms. In addition, a function command may require as many as six floating-point calculations, each taking 0.3 ms. This all adds up to well over 4 ms; therefore, a number of optimizations have been incorporated to reduce configuration time.

The first step is to avoid reconfiguring the instrument or a section of inguard if there is no change. For example, if the present function is ac volts and the new command is ACV, only the range is configured (if it changes), not the function. The ADC configuration is the same for dc volts, ohms, and dc current, so the ADC section is not reconfigured for changes between these functions. The trigger configuration changes only for digital ac voltage or frequency measurements, so a new configuration is sent only when entering or leaving these functions. In general, reconfiguration occurs only to the extent required by a given command.

Each combination of function and range uses different gain and offset values for the ADC readings. The gain and offset values are scaled by the ADC's aperture, so if the aperture increases by 2, the gain and offset are scaled by 2. An execution routine retrieves the gain and offset values from calibration memory and scales them by the aperture. Then the 120%- and 10%-of-full-scale points are calculated for overload detection and autoranging. The autoranging algorithm uses a different ADC aperture and has a separate set of 120% and 10% points. These two calculations were removed from the execution routine, and are done at calibration time since the autoranging algorithm always uses the same ADC aperture. To reduce the effect of the other four calculations, a data structure is used that saves the gain and offset for each function and range as it is needed. If the aperture of the ADC is changed, the data structure is cleared, and as function and ranges are revisited, the data structure is filled in. This eliminates recalculation of values that are constant for a given aperture.

An operation that is not always necessary but takes considerable time during a range or function change is a special sequence of relay closures in the front-end circuitry. This sequence protects the relays from damage when high voltage is on the input terminals during range changes, but is not needed when measuring low voltages or if high voltage is only present when the instrument is set to a high voltage range. Therefore, the HP 3458A provides an HP-IB programmable command to defeat the protection scheme, speeding up the relay sequence by a factor of five. If an overvoltage condition occurs while protection is inactive, an interrupt is generated and the relay sequence is reversed, thereby protecting the relays from damage. A delay of 0.4 second is then inserted to prevent a rapid recurrence of the overload condition, and the instrument reverts to normal (protective) relay sequencing thereafter.

Another technique used to reduce the configuration time is to defer computations until the last possible moment. The scale factor used in the format conversion of ADC readings from integer format to real or ASCII format is an example of this technique. Many commands cause the scale factor to change, so instead of each command computing the scale factor, a flag is set and the calculation is performed when the scale factor is first used. This eliminates wasted time from unnecessary calculations when many intermediate configuration changes are sent to the instrument, and reduces the time spent responding to even a single HP-IB command.

Data flow between the outguard and inguard sections has the potential to be a bottleneck, because the UART and the inguard processor can only accept configuration data at a rate of 20,000 words/s. Furthermore, commands to change relays can take a millisecond for the inguard processor to execute. To relieve the outguard processor of the need to wait on the inguard processor, a buffer was added to store messages bound for the UART. This buffer is deep enough to hold an entire configuration change—128 commands. This allows the outguard processor to overlap its activities with the inguard processor's. If the buffer is empty and the UART is not busy sending data, the 68000 will send a command directly to the UART, avoiding the overhead of the buffer. If the UART is busy, data is written to the buffer instead. In this case, the UART generates an interrupt when it is ready to accept the next word, which is then retrieved from the buffer and sent.

In addition to fast reconfiguration, system throughput depends on the time required to make a measurement. Fig. 4 shows the steps an ADC reading goes through before it is sent to the HP-IB. The first step is autoranging: if a reading is less than 10% of the range or greater than 120%, the instrument switches to the next range, changes the ADC's aperture for a fast measurement, and takes a reading. This procedure is repeated until the correct range is found, and then the final measurement is made with the ADC's original aperture. Although this algorithm is very fast (typically 8 milliseconds), it usually requires that the ADC take several

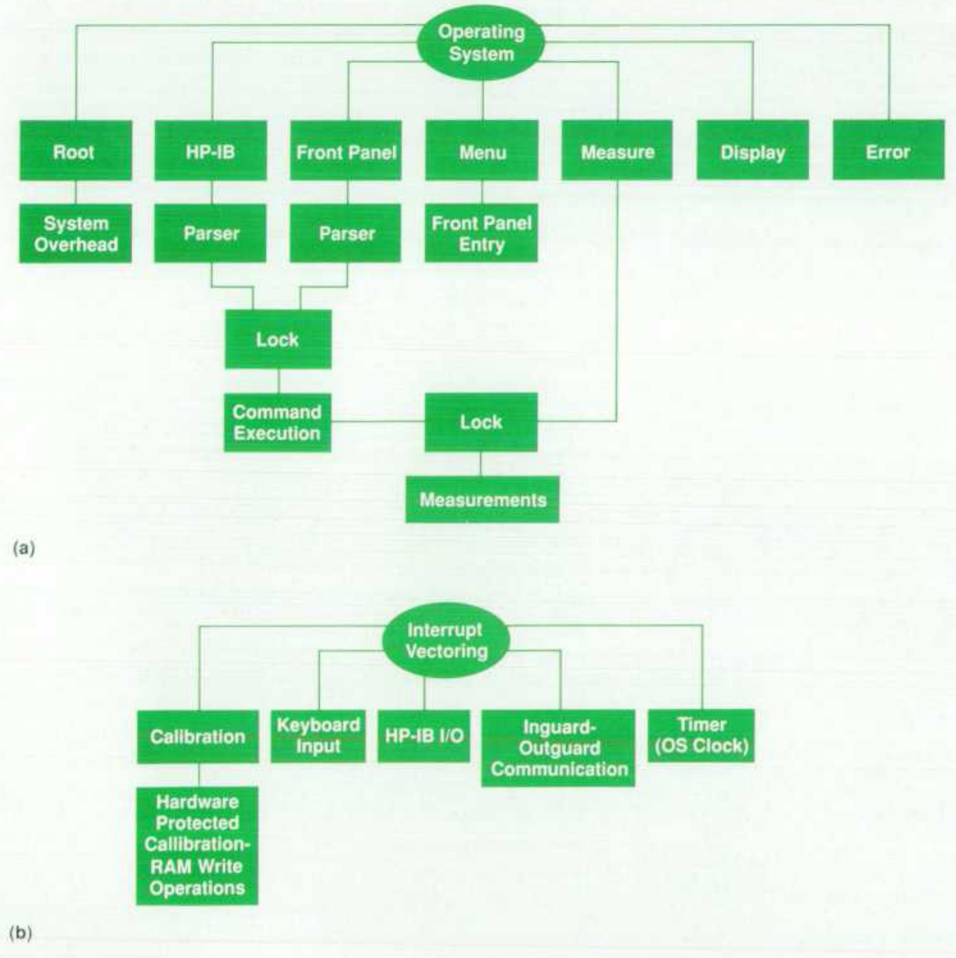


Fig. 2. *HP 3458A firmware structure. (a) Tasks under operating system control. (b) Interrupt service routines.*

Custom UART Design

At the center of the communications link between the inguard and outguard sections of the HP 3458A DMM is the custom UART (universal asynchronous receiver/transmitter). A serial interface was chosen because an isolated parallel interface would have been prohibitively expensive. Unfortunately, conventional UARTs are too slow to meet the HP 3458A's required data rate of 200 kbytes/s, which corresponds to a baud rate of 2 Mbits/s, counting start and stop bits. Therefore, fiber optic couplers were chosen, which also provide the benefit of infinite isolation resistance.

Conventional UARTs require a clock rate that is 16 times the baud rate; thus, to generate the 3458A's required baud rate, the clock rate would have to be 32 MHz. The 16× clock rate is needed to compensate for mismatched clock frequencies and waveform distortion. These two factors can be controlled within the HP 3458A, so a clock rate of three times the baud rate is used. The UART design is implemented as part of a CMOS gate array driven by a 10-MHz clock. This clock rate yields a baud rate of 3.3 Mbits/s, which meets the design goal with some margin.

The data format for the UART is shown in Fig. 1. The first bit is the start bit and indicates the beginning of a message. The next bit is the handshake bit. If this bit is high, a data/command message will follow immediately. If the bit is low, the message is a handshake and the next bit will be a stop bit. A handshake message is sent each time a data message is read by the processor, ensuring that a new message will not be sent until the previous message has been read. The next-to-last bit is the interrupt or command bit, used to indicate whether the preceding message was data or a command. A command message from the inguard section could be an ADC conversion failure, an end of sequence message, or a change in the front or rear terminals. Command messages generate interrupts, eliminating the need for software to check the data from the UART to determine the message type. The middle 16 bits of the message represent the data or command, and the last bit is the stop bit.

Fig. 2 shows a block diagram of the UART and the communication controller. When the decoding state machine detects that a start bit has been received, it waits three cycles to decide whether the message is a handshake. If so, the state machine returns to its initial state. If the message is data, the next 16 bits are clocked into the input shift register. The state machine then examines the next bit (the command/data bit). If the message is a command, an interrupt is generated.

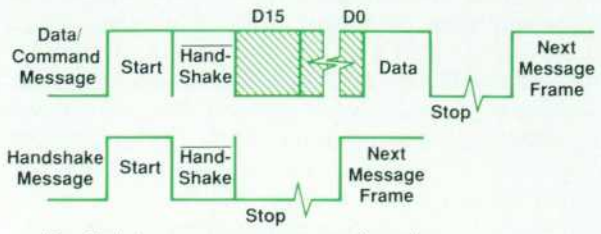


Fig. 1. *Interprocessor message formats.*

For transmitted messages, the encode machine first generates a start bit. If the message is a handshake, the next bit is set high; otherwise (if the message is data), the next bit is set low. The 16 bits of data are sent next (if required), and if the message is a command, the last bit is set high.

Buffers in the UART are used both for received data and data to be transmitted. This allows the ADC to leave data in the buffer while starting the next measurement, thus maximizing the overlap between outguard and inguard. Once the buffer has been emptied, the handshake message is sent and an interrupt can be generated. The interrupt can be used as a request for more data to be sent. The buffer queues requests from four sources: the ADC's error detection circuitry, the ADC's output register, the trigger controller messages, and the inguard processor.

The input buffer also has a direct output mode to the shift registers. When data is sent to the inguard section, the processor is interrupted, the data is parsed, and, if the message is a configuration message, the direct output mode is selected in the communication controller. This mode allows the next message to be sent to both the processor and the shift register, thereby sending the configuration data directly to the appropriate section. In this case, the processor receives the message but does not act upon it, thereby eliminating the overhead of processor intervention in the configuration process.

Although the use of microprocessors has enabled instruments to offer greatly enhanced measurement capability, a severe speed penalty may be incurred if firmware is burdened with tasks that are best left to hardware. The HP 3458A's use of a custom UART coupled directly to the measurement hardware optimizes performance by balancing the workload between hardware and firmware.

David J. Rustici
Development Engineer
Loveland Instrument Division

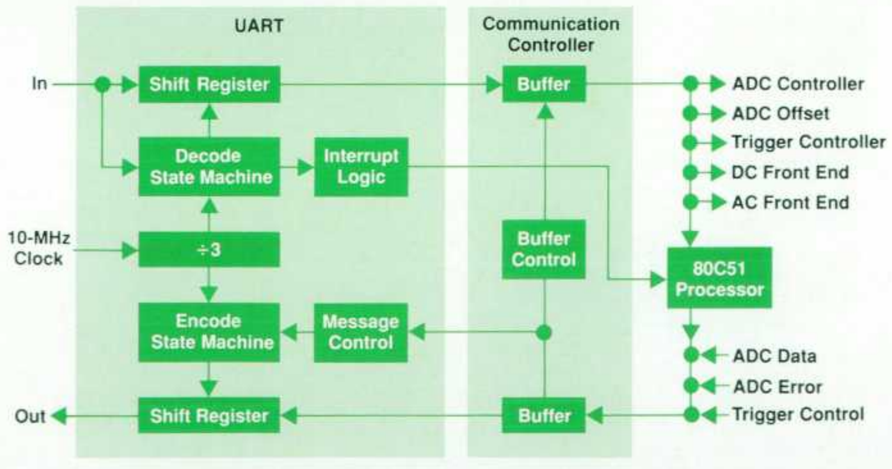


Fig. 2. *Block diagram of the UART and data communication portions of the inguard gate array.*

samples to generate one reading. Therefore, a faster measurement will be made if autoranging is turned off.

Throughput is also enhanced by minimizing operating system overhead. In cases where high throughput is not an issue (e.g., long integration times), measurements are handled by a background task, which runs whenever the instrument is not actively executing commands. This task simply monitors the trigger and trigger arm states to see if a measurement should be taken. When throughput is an issue, however, measurements are initiated directly by the HP-IB command parser/execution task. In this case, the overhead of task switching (approximately 250 μs) is eliminated, leaving only the overhead of communication between the interrupt service routine and the HP-IB task. Another speed enhancement is the use of preprogrammed states, which fall into two categories: predefined states (activated using the PRESET command), and user-defined states (stored using the SSTATE command and activated using the RSTATE command). Since these commands cause an extensive reconfiguration, their primary benefit is in putting the instrument in a known desired state. However, they can also save time when the alternative is to send long strings of commands to program the instrument to the same state.

Output Data Path

Once the instrument has been configured and triggered, a measurement is taken by the ADC and transmitted through the fiber optic link to the outguard processor. The format for this reading is either a 16-bit or a 32-bit two's complement result with the range offset subtracted. The next step is to convert the readings into volts, ohms, or amperes by multiplying by the gain of the range. If a math operation is active, it is initiated using a procedure variable that points to the math subroutine. At this point, the reading is in a 64-bit floating-point format, and a format conversion is required for an integer, ASCII, or short real format. The last step is to display the result and send it to memory or the HP-IB. Some steps can be eliminated using the appropriate HP-IB command; for example, the display operation is deleted using the DISP OFF command.

If autoranging, math, and the display are turned off and the output format matches the ADC's internal format, the measurement can be sent directly to the HP-IB or memory. Special assembly language routines were written to handle these high-speed modes. The time allowed to read the measurement and send it out is 10 μs (given a maximum reading rate of 100,000 per second). There are two data paths: one that sends readings to memory and one that sends them to the HP-IB.

Reading Storage. The memory structure dictated by HP's multimeter language is a general circular buffer in which readings may be added or removed at any time. This buffer can be used in either of two modes: FIFO (first in, first out) or LIFO (last in, first out), the main distinction being that the LIFO mode will overwrite the oldest readings when memory fills, whereas the FIFO mode will terminate when memory fills, thus preserving the oldest samples. A general program loop for receiving readings from the ADC and writing them into memory is as follows:

- Wait until the ADC has taken a reading.
- Write the reading into the current fill location and increment the fill pointer.
- Has the fill pointer reached the top of memory (buffer pointer wrap-around)?
- If memory is full and the memory mode is FIFO, stop.
- Terminate the loop when the end of sequence is sent.

Within 10 μs, the 68000 will allow only about three decisions to be made. Even using hand-optimized assembly

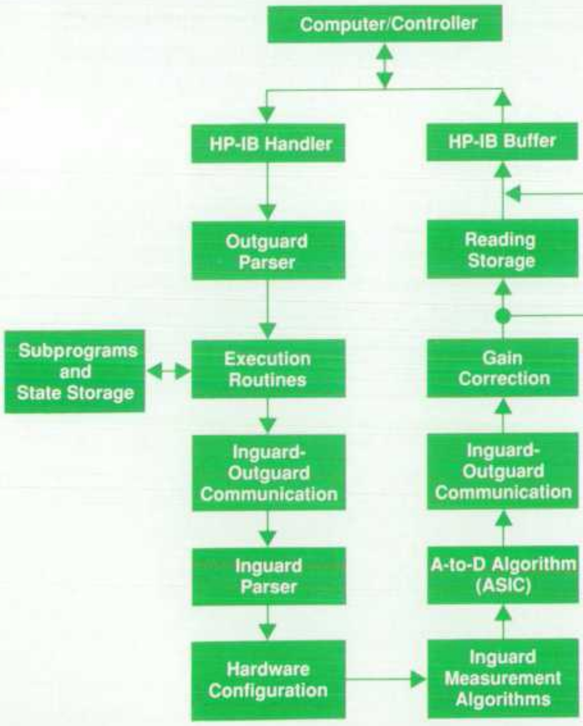


Fig. 3. *Input and output data flow paths.*

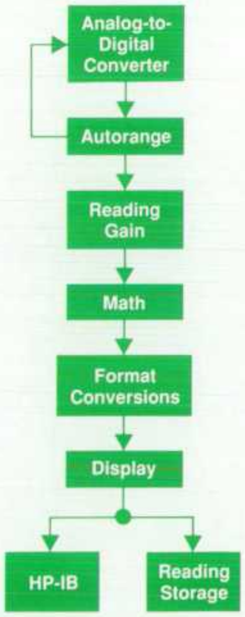


Fig. 4. *Processing of readings.*

language, a single program loop could not be written to implement the general memory model in the allotted time. The solution uses the fact that if enough decisions are made before the start of the burst, the number of on-the-fly decisions can be reduced. Before the start of a burst of samples, it is known how many readings can be added before the buffer pointers wrap around, and how much room is left before the circular buffer fills. The problem is divided into a set of special cases. For example, assume that 1000 readings are expected from the ADC. Memory fill and empty pointers indicate space for 2000 readings, but the fill pointer is only 100 samples from buffer wraparound. Under these conditions, the memory fill algorithm can be stated as follows:

- Fill memory with samples until the buffer fill pointer reaches the top of memory.
- Wrap around the fill pointer to the bottom of memory.
- Fill memory with samples until the sequence is complete.
- Exit the routine.

Any memory scenario can be expressed as a combination of the following special-case loops:

- Fill memory with samples until the fill pointer reaches the top of memory, then wrap around the fill pointer to the bottom of memory.
- Fill memory with samples until memory is full (fill pointer = empty pointer).
- Fill memory with samples until the sequence is complete.

Four factors influence the algorithm used: memory mode, number of readings expected, total available memory, and number of samples before wraparound. All possible combinations of these factors can be accommodated using only ten special-case combinations. Any particular special case can be built out of one to four of the routines listed above. Routines are linked together by pushing their addresses onto the stack in the reverse of the order in which they are to be executed (the address of the exit routine is pushed first), and the first routine is called. In the example above, the first routine is called to fill memory until it detects buffer wraparound. It then loads the fill pointer with the address of the bottom of memory and executes an RTS (return from subroutine) instruction, which pops the address of the next routine from the stack and jumps to it. The next routine continues filling memory until the burst is complete, then terminates in another RTS instruction, which pops the address of the exit routine. The exit routine performs some minor cleanup (restoring pointers, setting flags, etc.) and leaves.

HP-IB Output. The high-speed output routine for the HP-IB uses some of the same concepts as the memory routines. In this case, the algorithm is as follows:

- Initialize pointers.
- Wait until the ADC has taken a reading, then enter the readings.
- Wait until the HP-IB buffer is ready to accept more data.
- Transfer the reading to the HP-IB buffer.
- Terminate the loop when the end-of-sequence command is sent.

The HP-IB buffer accepts a 16-bit word from the processor and sends the lower eight bits to the HP-IB interface chip. Once this byte has been transmitted, the HP-IB chip signals the buffer, and the buffer then sends the upper eight bits without intervention from the processor. Use of a buffer relieves a congestion point in the output data flow that would occur if the processor wrote directly to the HP-IB chip, since the HP-IB is an eight-bit bus while all other internal data paths are 16 bits wide. Using this scheme, the HP 3458A is able to offer complete memory and HP-IB functionality at the full speed of 100,000 16-bit dc voltage readings per second.

Summary

Achieving high throughput in a system DMM is a matter of designing the instrument as a system for moving data efficiently. Hardware and firmware must be designed as integral elements of this system, not as isolated entities. In the design of the HP 3458A, experience with DMM performance limitations provided invaluable insight into key areas of concern. As a result, significant improvements in throughput were achieved through the development of high-speed custom gate arrays for ADC control and interprocessor communications. Use of high-performance microprocessors and supporting hardware also contributed greatly to meeting design goals, as did the substantial investment in firmware design and development that was necessary to translate increased hardware performance into increased system performance.

Acknowledgments

We would like to thank Greg Wale and Dave Czenkusch, who were responsible for a significant part of the firmware development, and Vicky Sweetser, who was responsible for coordinating, testing and releasing firmware revisions. We also thank Bill Lutton and Jerry Metz of the HP 3235A design team for contributing parts of their firmware to be reused in the HP 3458A.

High-Resolution Digitizing Techniques with an Integrating Digital Multimeter

Capabilities and limitations of the HP 3458A Digital Multimeter as a high-resolution digitizer are summarized. Performance data is presented for selected applications.

by David A. Czenkusch

WITH ITS INTEGRATING analog-to-digital converter (ADC) capable of making 100,000 conversions per second, the HP 3458A Digital Multimeter (DMM) raises the possibility that, for the first time, a voltmeter can satisfy many requirements for high-resolution digitizing.

What are the characteristics of a high-resolution digitizer? Digitizing requires a combination of fast, accurate sampling and precise timing. It also needs a flexible triggering capability. The HP 3458A allows sampling through two different signal paths, each optimized for particular applications.

Converting a signal using the dc volts function (which does not use a sample-and-hold circuit, but depends on the short integration time of the ADC) provides the highest resolution and noise rejection. The direct sampling and subsampling functions, which use a fast-sampling track-and-hold circuit, provide higher signal bandwidth and more precise timing.

High-Resolution Digitizer Requirements

As the block diagram in Fig. 1 illustrates, a digitizer consists of an analog input signal conditioner followed by a sampling circuit. A trigger circuit and time base generator controls the timing of samples. The output of the sampling circuit is converted to a number by an analog-to-digital converter (ADC). Once converted to a number, the sample data can be processed digitally and displayed to the user.

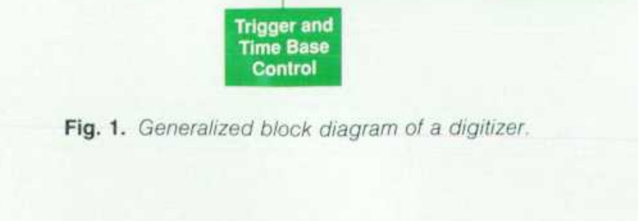
Many types of instruments fit this definition of a digitizer, including digital oscilloscopes, dynamic signal analyzers, and digital multimeters (DMMs). Digitizing products can be roughly differentiated by four characteristics: analog signal bandwidth, sample rate, signal-to-noise ratio (which can be expressed as effective bits of resolution), and type of data displayed (time, frequency, etc.). In general, digital oscilloscopes tend to have high bandwidth and sample rate and relatively low resolution, while DMMs and dynamic signal analyzers tend to have much higher resolution and correspondingly lower bandwidth and sample rate.

Digital oscilloscopes are known for their high bandwidth, typically 100 MHz or greater, and their digitizing rates of 50 megasamples to one gigasample per second, making them useful for capturing very fast, single-shot events. Their resolution of five to eight effective bits is well-suited for displaying waveforms on a CRT, since one part in 200 is perfectly adequate for the human eye.

Dynamic signal analyzers, on the other hand, are used in applications that call for higher resolution—typically 10 to 14 bits. Examples include dynamic digital-to-analog converter testing, telecommunications, SONAR, and seismic or mechanical measurements that require digital signal processing. These applications require higher resolution and typically involve frequency-domain analysis. Therefore, to judge the attributes of a high-resolution digitizer, we should also examine the characteristics of discrete Fourier transforms (DFTs) performed on the digitizer's output data.

Digitizer Spectral Attributes

"Effective bits" is a measure of the resolution of an ADC. Essentially, it is a measure of the signal-to-noise ratio in a



Input
Input Signal Conditioner
Sampling Circuit
Analog-to-Digital Converter
Data Processing and Presentation
Trigger and Time Base Control

Fig. 1. *Generalized block diagram of a digitizer.*

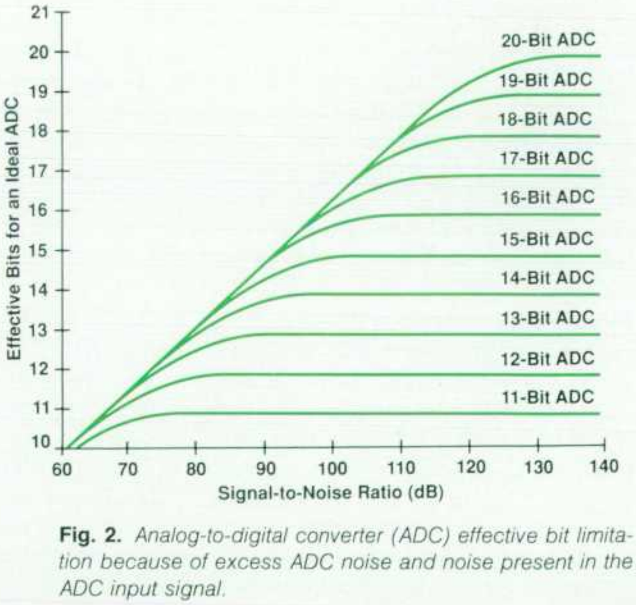


Fig. 2. *Analog-to-digital converter (ADC) effective bit limitation because of excess ADC noise and noise present in the ADC input signal.*

digitizing system expressed as a power of two. This can be expressed mathematically as:

$$N(\text{effective}) = (S/(N + D) - 1.8)/6.02$$

where S/(N + D) is the ratio of the signal power of a full-scale input to the total power of noise plus distortion, expressed in dB. Notice that the effective bits rating and the signal-to-noise ratio expressed in dB are both logarithmic scales related by the constant 6.02. This means that increasing the resolution of a measurement by one effective bit results in a 6-dB improvement in the signal-to-noise ratio. The system noise term, N + D, is the rms result of the power contributions of harmonic distortion and noise from various sources. For an otherwise noise-free, distortion free-system, there is minimum noise component because of the fundamental quantization error of the ADC. If this is the only source of error, the number of effective bits approaches the basic resolution of the ADC. Fig. 2 shows how the number of effective bits decreases as errors from other sources increase.

Other types of errors will appear as random noise. These include noise in the input signal, noise in the analog input circuits, random jitter in the timing of samples, and noise and differential nonlinearity in the ADC.

Linearity error is a measure of the deviation of the output of an ADC from the ideal straight-line relationship it should have with the input voltage. Fig. 3 shows a graph of the linearity error of a typical ADC as a function of input voltage. Integral linearity error is the large-scale bow in the total linearity error plot. This deviation from a straight line can often be described by a second-order or third-order function. Differential linearity error, on the other hand, has no large-scale structure, so it looks very much like noise.

If the noise in a digitizer is truly random, then a point-by-point average of many independent ensembles of waveform data taken with the same input signal will reduce this noise by the square root of the number of ensembles, provided the different ensembles of data have the same phase relationship to the input signal. Analog noise in the input amplifier and ADC and noise caused by random timing errors tend to be uncorrelated with the input signal, and so can be reduced by waveform averaging. On the other hand, differential linearity error in the ADC and systematic timing errors, while appearing like random noise in a single pass of data, are repeatable from pass to pass, and so are correlated with the input and cannot be reduced by averaging. This provides a way of determining if the signal-to-noise ratio of a given digitizing system is dominated by input noise or by differential linearity error.

Effective Bits from the DFT

One way to characterize the signal-to-noise ratio of a digitizer is to sample a quiet (low-noise) and spectrally pure full-scale sine wave and perform a discrete Fourier transform (DFT) on the resulting data. The dynamic range (in dB) from the peak of the fundamental to the noise floor of the DFT gives an idea of the low-level signals that can be resolved. The level of the noise floor depends on the number of frequency points (bins) in the DFT, and hence on the number of samples taken, since if the same noise power is spread over more frequency bins, there will be less noise power per bin.

The DFT spectrum can be used to produce an estimate of the signal-to-noise ratio of a digitizer by performing essentially the same measurement digitally that a distortion analyzer performs electronically. A distortion analyzer supplies a low-distortion sine wave as the input to a circuit under test. A notch filter is used to remove the fundamental frequency from the output signal. The power in the filtered signal is measured and a ratio is formed with the total output power of the circuit under test. A distortion analyzer measurement assumes that the power in the filtered output signal is dominated by harmonic terms generated by distortion in the circuit under test. In practice, however, the analyzer is unable to separate this power from the power contribution of wideband noise, and hence is actually measuring the signal-to-noise ratio of the output signal.

An analogous operation can be performed on the DFT spectrum of a digitized pure sine wave. A certain number of frequency bins on either side of the fundamental peak are removed from the DFT data. The data in each of the other frequency bins is squared (to yield a power term) and summed with similar results from the other frequency bins to calculate the total noise power. The data within the narrow band around the fundamental is squared and summed to give the total signal power. The ratio of these two terms,

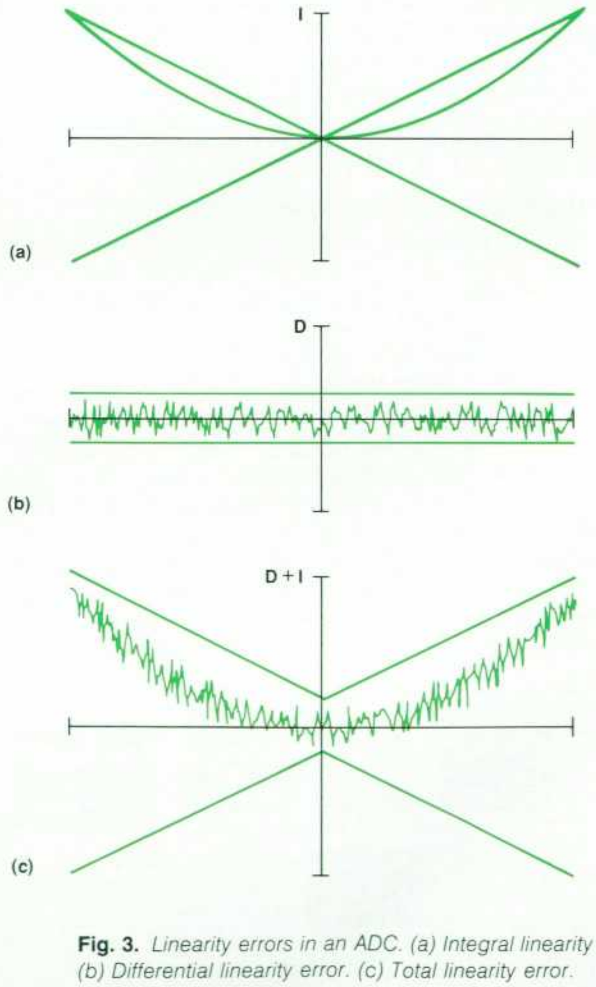


Fig. 3. *Linearity errors in an ADC. (a) Integral linearity error. (b) Differential linearity error. (c) Total linearity error.*

expressed in dB, can be used to compute the number of effective bits of resolution of the digitizer.

Calculations of effective bits from DFT spectra will show variations if the test is performed repeatedly. This variation can be reduced if the spectral values from many independent trials are averaged point by point (as opposed to averaging the time-domain data). Spectral averaging will not reduce the level of the noise floor in the DFT data, but only the amount it varies. Therefore, if enough ensembles of spectral data are averaged, the number of effective bits calculated will converge to a single number.

Fig. 4 shows the DFT for 4096 samples of a mathematically generated ideal sine wave quantized to 16 bits (±32,767 counts). From this, we see that a perfect 16-bit digitizer will show a noise floor of about −127 dB when quantization error is the only source of noise. If the signal-to-noise ratio is calculated using the method described above, the result is 97.0 dB, or 16.0 effective bits, which is what we would expect.

Other types of digitizer errors can show up on a DFT plot. Distortion reveals itself as harmonic components at multiples of the fundamental input frequency. This can be distortion in the input signal, harmonic distortion in the input amplifier, or integral nonlinearity in the ADC. As mentioned before, integral linearity error can be approximated by a second-order or third-order term in the transfer function of the ADC. These higher-order terms generate spurious harmonic components in the DFT spectrum.

Other spurious signals can show up in the DFT spectrum besides harmonic distortion. Internal clock signals can produce unwanted signal components (spurs) either by direct cross talk or through intermodulation with the input signal. These effects are commonly grouped together into a single specification of spurious DFT signals.

Effect of Sample Aperture

Another aspect of digitizers that should be considered is the effect of the finite acquisition time of the sampling circuit that provides the input to the ADC. This is typically some type of sample-and-hold or track-and-hold circuit. For maximal time certainty, an ideal track-and-hold circuit would acquire a voltage instantaneously when triggered to take a sample. In reality, of course, all sampling circuits require some finite time to acquire a sample. This sampling function can be approximated by a rectangular window of time T over which the input signal is sampled.

The Fourier transform of a square pulse defined over the interval $-T/2 \leq t \leq T/2$ in the time domain has the form $[\sin(\pi fT)]/\pi fT$, which is the familiar function sinc(fT). This means that sampling a signal for a time T is equivalent in the frequency domain to multiplying the input spectrum by the function sinc(fT). Fig. 5 shows that the spectral envelope of the sinc function approximates a single-pole low-pass filter with a 3-dB corner frequency of $f_c \approx 0.45/T$.

From this analysis we can conclude that making the sample time as short as possible produces the flattest possible response because it maximizes the aperture roll-off corner frequency. A less desirable trade-off, however, is that this also increases the equivalent white noise bandwidth of the sampler, thereby increasing its sensitivity to noise. Therefore, in applications where noise is a greater problem than frequency roll-off, it would be desirable to have a wider sample aperture to reduce the noise bandwidth.

The transform above was defined for a square pulse extending from $-T/2$ to $T/2$. Since a real sampler cannot anticipate its input, the sample must actually occur over the interval $0 \leq t \leq T$. This implies that any sampler that acquires a signal over a nonzero time interval T will introduce an apparent time delay equal to T/2 to the output. In most real applications, however, this distinction is not significant.

Another characteristic of the sinc function that can be useful is that its transfer function goes to zero at all frequencies that are multiples of 1/T. This means the sampler will reject all harmonics of a signal whose fundamental period is equal to the sample aperture. Therefore, a selectable aperture allows the rejection of specific interference frequencies that may be present in the measurement environment.

HP 3458A Digitizing Characteristics

Many of the same design characteristics required to make

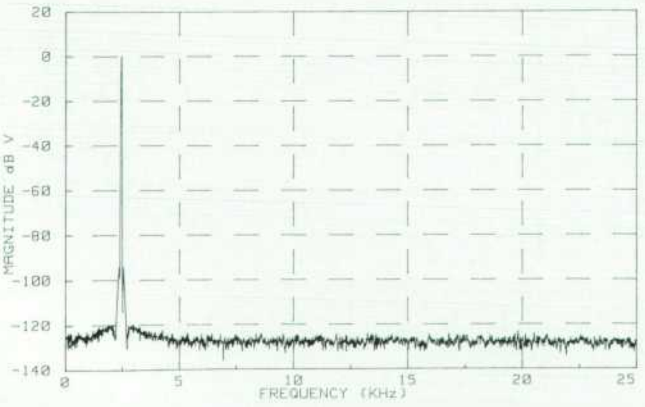


Fig. 4. *Discrete Fourier transform of an ideal sine wave sampled with an ideal 16-bit ADC.*

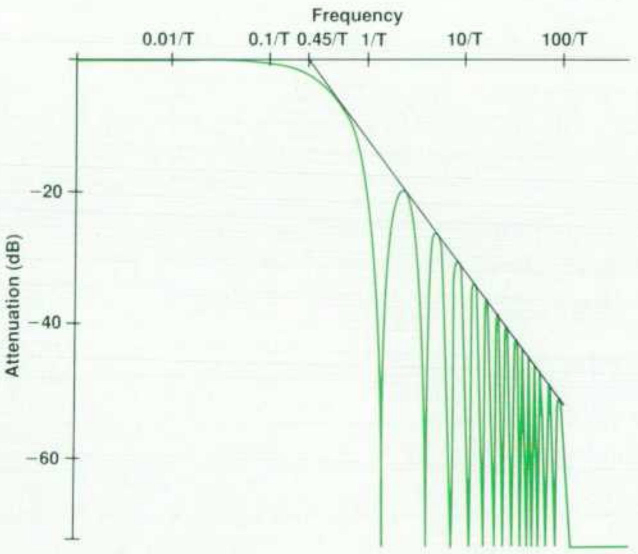


Fig. 5. *Attenuation of the input signal as a function of frequency resulting from sampling with an aperture of width T.*

Time Interpolation

To implement the subsampling (effective time sampling) required for the HP 3458A DMM's digital ac measurement technique, some means of synchronization with the input signal was necessary. To minimize errors caused by aliasing of the sampled data, a time base with 10-ns resolution was desired. However, the internal 10-MHz clock would only allow a sample resolution of 100 ns relative to a synchronizing trigger event. These design requirements dictated the development of the time interpolation circuit of the HP 3458A.

The instrument's 10-MHz clock is used to generate sample timing pulses of variable period in 100-ns (10-MHz) steps. The time interpolator extends the resolution of the time base from 100-ns steps to 10-ns steps for initial burst delays (the delay from a trigger event to the start of sampling). This enables the HP 3458A to digitize signals with spectral content up to 50 MHz without introducing aliasing errors.

The time interpolator, Fig. 1, uses analog techniques to convert time to stored charge on a capacitor. Before an input trigger, the interpolator is reset by shorting both capacitors (S1 and S2 closed) with the current source shorted to ground (S3 and S4 in position B). An asynchronous input trigger, generated either by the ac path's trigger level circuit or by an external trigger input, initiates charge accumulation on C1 by opening S1 and setting S3 and S4 to position A. This charge accumulation process continues until the next positive edge of the 10-MHz clock occurs.

On this edge S3 and S4 switch to position B, forcing the accumulated charge to be held on C1. This charge, Q_1, is directly proportional to the elapsed time (T_{var1}) between the input trigger and the next 10-MHz clock edge. Likewise, the voltage across C1 (V_{var1}) is also proportional to T_{var1}, which varies between 50 ns and 150 ns depending on the timing of the asynchronous input trigger relative to the internal 10-MHz clock.

The interpolator remains in this "hold" state for an integral number of clock cycles, T_{delay}. The next positive-going clock edge after T_{delay} initiates the second charge accumulation process. At this time, S2 opens and S3 and S4 are switched to position A. During this time, the same charge, Q_2, is accumulated on C1 and C2. This process continues until the voltage on C1, V_{ramp}, crosses the programmable comparator threshold V_t. This transition generates an output trigger that signals the track-and-hold circuit in the ac section to enter hold mode, thus acquiring a sample for subsequent ADC conversion. By programming V_t to various values, the system can alter this delay in increments of 10 ns, allowing precise timing of a burst of samples relative to an asynchronous starting event.

The output trigger also switches S3 and S4 to position B. This not only turns off the current source, but also creates a loop between C2, R1, and the buffer amplifier's input and output. Feedback forces a current through C2, removing its accumulated charge, Q_2. The resulting current flows through both C1 and C2, removing the charge Q_2 from capacitor C1. The process completes with C1 holding the original charge, Q_1, which is proportional to the delay between the first trigger and the rising edge of the internal 10-MHz clock. During the ADC conversion, (a

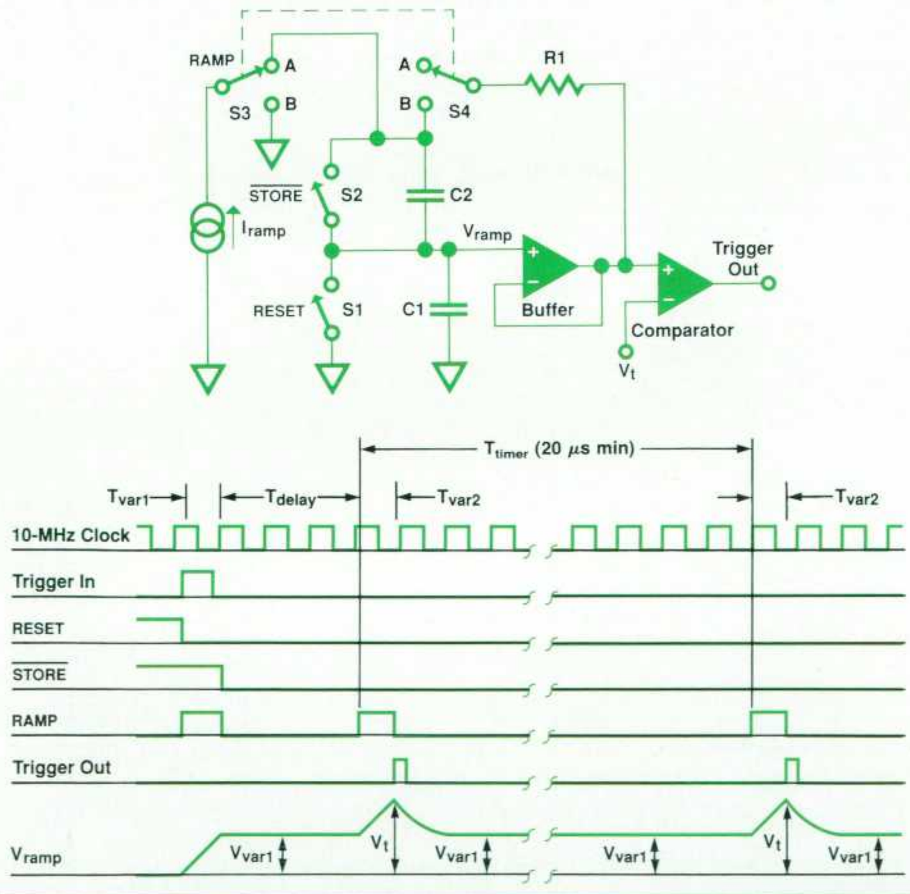


Fig. 1. *HP 3458A DMM time interpolator block and timing diagrams.*

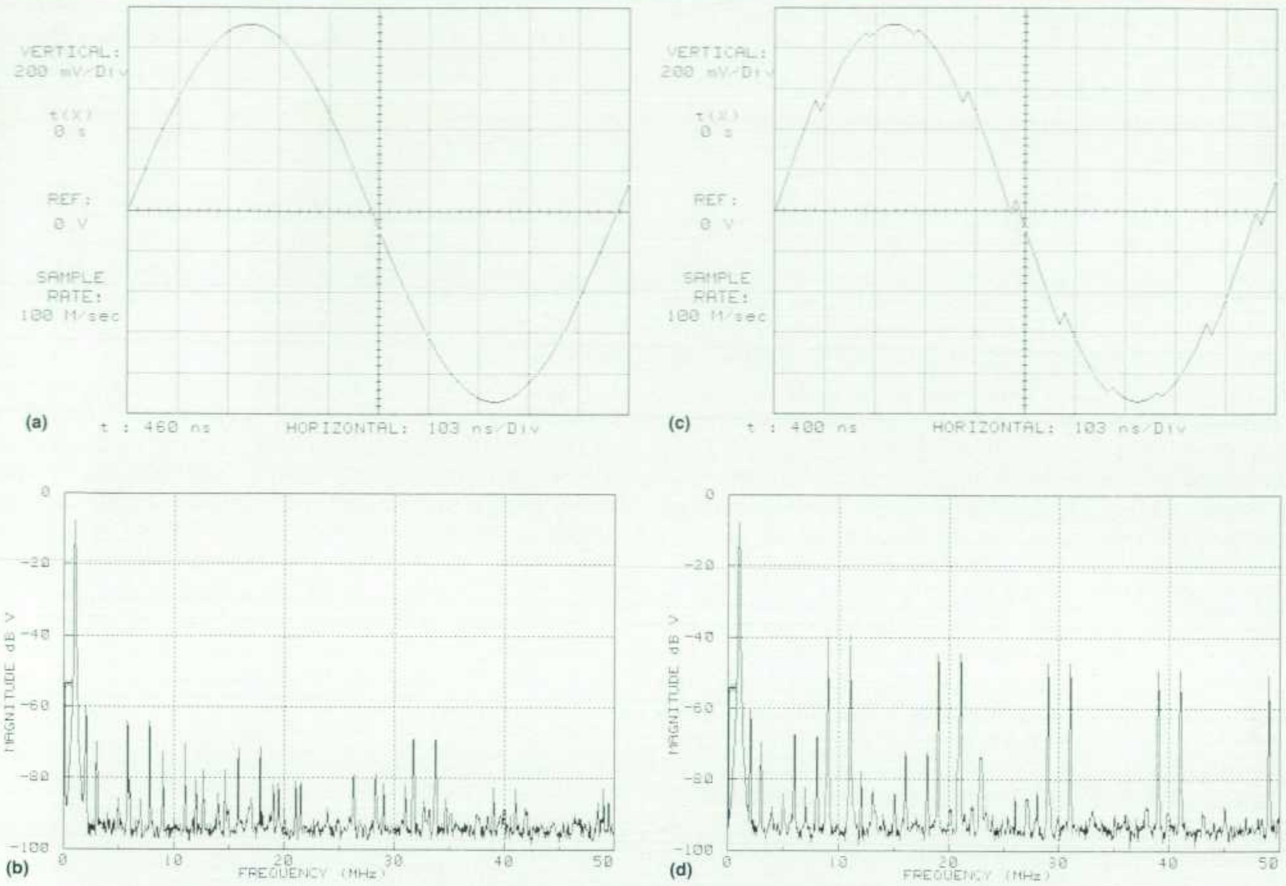


Fig. 2. *The time interpolator's accuracy is adjusted by calibrating I_{ramp}. (a) A digitized 1-MHz waveform after I_{ramp} calibration. (b) Fourier transform of (a), showing a noise floor 80 dB below the fundamental and spurious signals below −55 dB. (c) A digitized 1-MHz sine wave with I_{ramp} misadjusted. (d) Fourier transform of (c).*

minimum of 20 μs for subsampling) the time base circuit waits an interval T_{timer} before repeating the charge/discharge cycle.

The accuracy of the 10-ns increments is ensured by calibration of the circuit gain. Since the time interpolator's absolute delay is a function of I_{ramp}, C1, and V_t, many variables can prevent the 10-ns increments from being exactly one tenth of a 100-ns time base step. Interpolation for ten 10-ns intervals must precisely equal one 100-ns clock period (10 MHz) to minimize sampling errors. By adjusting I_{ramp} (Fig. 2), the slew rate and threshold errors are adjusted to yield 10-ns steps within ±50 ps. Time jitter is held to less than 100 ps rms. Low temperature coefficients for C1 and the DAC that generates V_t ensure interpolator accuracy over the operating temperature range. The time interpolator is adjusted by applying a 2-MHz sine wave to the input and executing a calibration routine which alternately programs 100-ns delays into either the time base or the time interpolator. By adjusting the DAC that controls I_{ramp}, the routine converges the two delays. This time base performance contributes to the a noise floor 80 dB below the fundamantal and spurious signals below −55 dB.

The design of the time interpolator circuit was refined using analog simulation methods on an HP 9000 Model 320 Computer. Computer-aided engineering provided timely feedback during development, allowing rapid evaluation of alternative circuit topologies. Critical design characterizations, difficult to achieve by traditional means, were performed accurately and simply using CAE simulations. The resulting circuit performance exceeded our original design goals.

David E. Smith
Development Engineer
Loveland Instrument Division

high-accuracy ac and dc measurements also allow the HP 3458A to perform well as a high-resolution digitizer. For instance, because it makes true-rms ac measurements using digital techniques, it has a scope-like trigger level circuit for waveform synchronization. A precise trigger timing circuit allows sample intervals to be specified to a resolution of 100 nanoseconds and initial delays from a trigger event to the first sample of a burst can be specified to a resolution of 10 nanoseconds using an analog time interpolator.

As the block diagram in Fig. 6 shows, the HP 3458A provides two distinct input paths for digitizing, corresponding to the two amplifiers used for the dc volts and ac volts functions. Each path has advantages and disadvantages. The dc input path should be used when maximum resolution and noise rejection are required and the bandwidth of the input signal is relatively low. Because it uses a track-and-hold circuit, the ac input path can be used on signals of higher bandwidth or when the signal must be sampled at a very precise point in time.

High-Resolution DC Input Path

The dc input path allows higher-resolution sampling as well as a higher single-shot measurement speed, providing 16-bit samples at up to 100,000 samples per second. The bandwidth of this amplifier varies from 50 kHz to 150 kHz, depending on the range selected. The widest bandwidth is available on the 10V range, when the amplifier is operating at unity gain. In this path, the sampling function is performed by the ADC itself with its selectable integration time (sample aperture). Historically, digital multimeters with integrating ADCs have allowed only a few discrete values for integration time. These values were chosen to be multiples of the power-line frequency—the most common signal to interfere with a high-resolution voltage measurement. In the HP 3458A, integration times can be specified from 500 ns to 1 s in increments of 100 ns. This allows the rejection of an interference signal of arbitrary frequency that may be present in the input, and provides attenuation of other frequencies above the sample rate by the approximate single-pole roll-off characteristic of the sample aperture's sinc function. The longer the integration aperture specified, the more resolution is provided by the ADC. Fig. 7 shows the resolution that can be obtained for a given aperture.

Because the dc input path is designed for extremely low noise, low offset, and part-per-million (ppm) accuracy, the DFT spectra produced in this mode are quite good. In fact, it is difficult to determine whether the harmonic distortion and noise floor measurements are dominated by the HP 3458A or by the input signal.

Fig. 8a shows the DFT calculated on 4096 samples of a 1-kHz waveform aquired at a rate of 50,000 samples/s with an integration time of 10 microseconds. The noise floor and spurious DFT signals are below −120 dB, and harmonic spurs are below −106 dB. If the signal-to-noise ratio is computed from the spectral data, the result is approximately 93.9 dB, yielding 15.3 effective bits.

The input signal for this test was provided by the oscillator output of an HP 339A Distortion Measurement Set, whose distortion at this frequency is specified to be less than −96 dB at the first harmonic. It is unclear whether the first-harmonic term at −107 dB is caused by distortion in the source signal or distortion in the HP 3458A at this sample rate. However, tests performed at slower sample rates (and greater resolution) also exhibit this second-harmonic term.

The averaging effect of the relatively wide sample aperture (10 μs vs. 2 ns) reduces random noise contributions to the DFT noise floor to a level comparable to those of systematic nonlinearities. Because of this, waveform averaging only provides an extra 4.4 dB improvement in the signal-to-noise ratio, yielding an extra 0.7 effective bit. Fig. 8b shows the DFT spectrum that results if 64 waveforms are averaged.

A striking example of the high-resolution digitizing capability of the dc volts sampling mode involves measuring an ultralow-distortion signal source used to characterize the performance of seismic measurement systems. The output of the source is a 0.03-Hz sine wave whose noise and harmonic distortion products are guaranteed by design to be at least 140 dB below the level of the fundamental. Superimposed on this is a 1-Hz sine wave whose amplitude is 120 dB below the level of the 0.03-Hz signal. The goal of the measurement system two-tone test is to be able to see the 1-Hz tone clearly in the presence of the full-scale

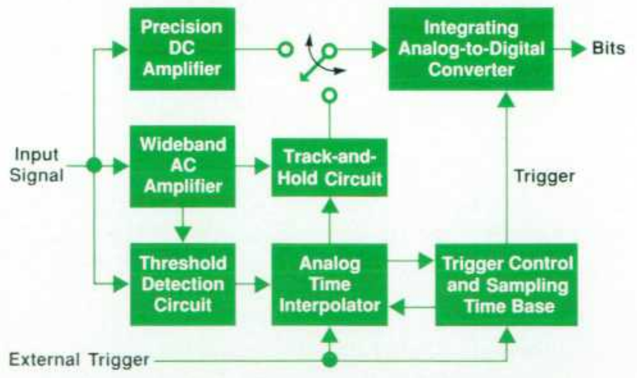


Fig. 6. *HP 3458A block diagram, showing the two measurement paths.*

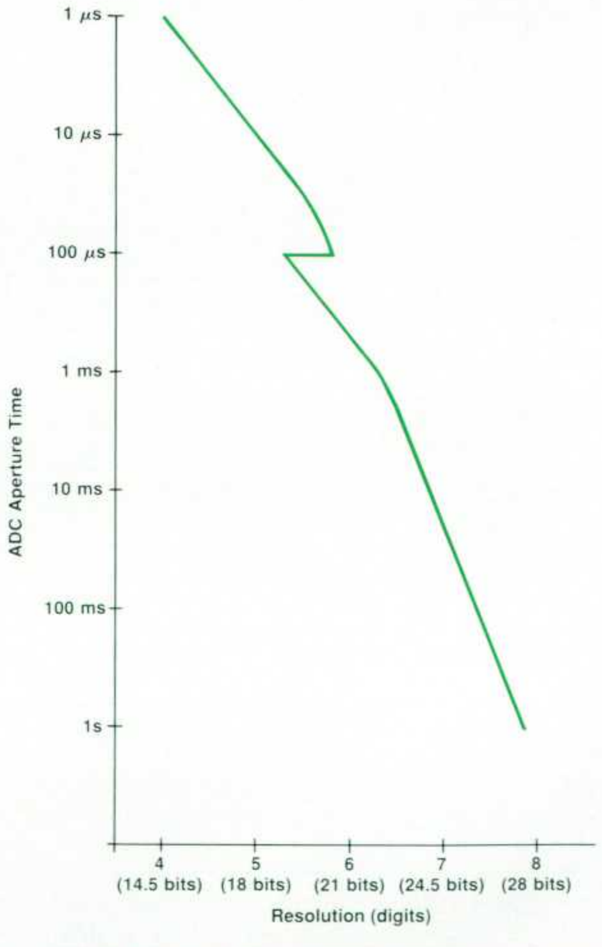


Fig. 7. *HP 3458A measurement resolution as a function of aperture time (speed).*

(10V peak) input without introducing extraneous distortion products. The HP 3458A was used to acquire 4096 samples with an ADC aperture of 20 milliseconds and a sample interval of 50 milliseconds, resulting in a resolution of 24 bits (7½ digits).

The DFT plot in Fig. 9 shows the result of this test. Only a portion of the full 10-Hz bandwidth is shown to make the component at 0.03 Hz more apparent. The 1-Hz spike at −120 dB is clearly visible above a noise floor of −150 dB. If the 1-Hz component is notched out along with the 0.03-Hz fundamental, and the remaining power is considered noise, a signal-to-noise calculation yields 19.6 effective bits. As before, it is not clear whether the DFT noise floor in this measurement is dominated by noise in the input signal or noise in the HP 3458A. If the rms noise of the HP 3458A is characterized with the same ADC aperture (20 ms) and a quiet dc source is substituted as input, measurements demonstrate a performance of 22 effective bits. The HP 3458A is clearly capable of verifying the performance of this source to the levels guaranteed by its designers. We are told that earlier measurements had never been able to achieve these low levels of noise and distortion.

In the dc volts mode, the input signal is sampled directly by the ADC. The sampling is synchronous with the instrument's internal 10-MHz clock. This leads to a 100-nanosecond peak uncertainty in the time latency of a sample or group of samples relative to an external or level trigger event. While a time uncertainty of 100 nanoseconds from an asynchronous trigger event is perfectly adequate for most applications, other applications require more precise sample timing.

Digital AC Input Path

The ac input path provides a wider analog bandwidth and more precise timing than the dc path. The bandwidth of the ac amplifier is 12 MHz on all ranges except the 10-mV and 1000V ranges, where the bandwidth is 2 MHz. Autocalibration guarantees a frequency response flatter than 0.01% (0.001 dB) throughout the frequency band from 200 Hz to 20 kHz, making this path ideal for characterizing frequency response in the audio band. While the maximum single-shot sample rate of 50,000 samples per second is somewhat lower than the dc input path because of the additional settling time required by the track-and-hold circuit, a precise timing circuit allows effective time sampling (subsampling) of repetitive input signals with effective sample intervals as short as 10 ns.

Achieving true-rms ac measurements with 100-ppm accuracy using digital techniques requires an extremely linear track-and-hold circuit. This same track-and-hold circuit provides 16-bit linearity in digitizing applications. A sample acquisition time of approximately 2 ns results in a 3-dB aperture roll-off frequency of at least 225 MHz. This means that amplitude errors caused by the sample aperture are insignificant through the entire measurement band. The timing of the track-and-hold circuit is controlled by an analog ramp interpolator circuit which operates asynchronously with the internal 10-MHz clock, giving a burst-to-burst timing repeatability error less than 100 picoseconds. The time interpolator allows programming of delays from an external or internal trigger with a resolution of 10 ns, allowing single samples to be timed very precisely.

While the greater equivalent noise bandwidth of the input amplifier and track-and-hold circuit results in fewer effective bits of resolution in a single-shot measurement than the dc input path, the DFT performance for this path is still quite good. Fig. 10a shows a typical 2048-point DFT plot for a 1-kHz sine wave sampled at the single-shot limit of 50,000 samples per second. A signal-to-noise ratio calculation on this data yields 10.4 effective bits. The ac input path has a greater equivalent noise bandwidth than the dc input path, so random noise dominates the signal-to-noise

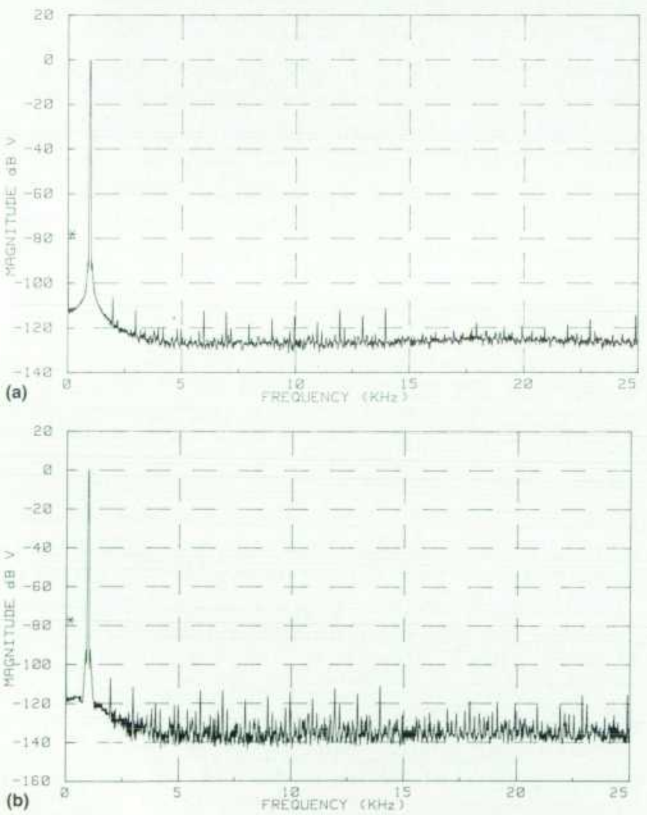


Fig. 8. *(a) Single-shot, 4096-time-sample discrete Fourier transform (DFT) of a 1-kHz input signal. (b) DFT for 64 averaged acquisitions of the 1-kHz input signal. Effective bits are 15.3 for (a) and 16.0 for (b).*

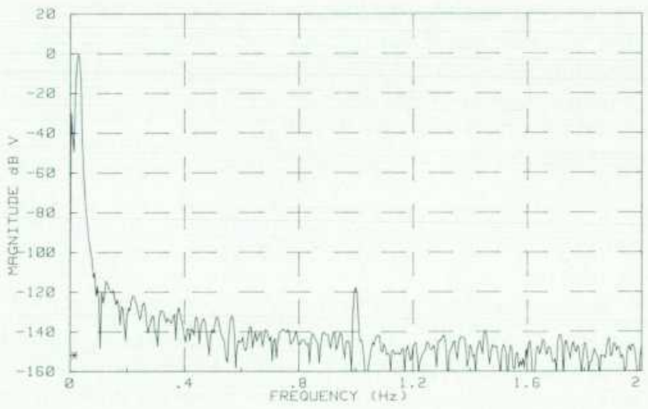


Fig. 9. *DFT for a 4096-sample HP 3458A acquisition of a 0.3-Hz sine wave with a 1-Hz, −120-dB sine wave superimposed. The effective bits rating is 19.6.*

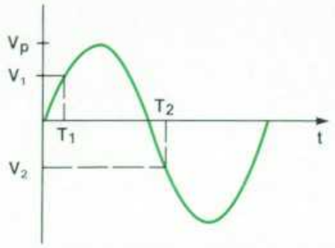
Measurement of Capacitor Dissipation Factor Using Digitizing

No capacitor outside of a textbook exhibits the theoretical current-to-voltage phase lag of 90 degrees. This is another way of saying that in the real world all capacitors are lossy to some extent. These losses are caused by a number of factors, such as lead resistance and dielectric hysteresis.

At a given frequency, the dissipation factor of a capacitor is defined to be the ratio of the equivalent series resistance (ESR) and the capacitive reactance. Dissipation factor is important for many applications. At high power levels, capacitors with poor dissipation factor can overheat. The precision of capacitively compensated attenuators can be compromised by dissipation factor. Also, the capabilities of track-and-hold circuits are degraded by the dissipation factors of their hold capacitors.

There are two common ways to measure dissipation factor. In the first method, the impedance of the capacitor under test (CUT) is measured at a given frequency and the deviation in phase angle from the ideal 90 degrees is used to calculate the dissipation factor. Bridges are another method used to measure dissipation factor. In essence, the CUT is in a bridge with three other capacitors, one of which is adjustable in both C and ESR. When the bridge is nulled, the values of the adjustable C and its ESR determine the dissipation factor of the CUT.

The ac attenuator in the HP 3458A uses a 20-pF capacitor that has a dissipation factor requirement of 0.0001 (0.01%) at 10 kHz. Commercially available automated equipment exhibits reading-to-reading noise of 0.01% and dissipation factor accuracies of 0.04%. This is inadequate to screen this capacitor reliably. High-quality manual bridges can do this job, but their operation is not well-suited to a production environment.



$V_1 = V_p \sin\phi$

$V_2 = -V_p \sin\phi$

$\phi = \sin^{-1} \frac{V_1 - V_2}{2 V_p}$

Fig. 1. *Measuring phase shift in a sine wave.*

By making use of the high-resolution digitizing and precision ac measurement capabilities of the HP 3458A, it is possible to construct an automated dissipation factor meter that is capable of making accurate and stable 0.001% dissipation factor measurements and capacitance measurements that are stable to 0.001 pF.

Circuit Description

In Fig. 1, a method of determining the phase shift of a sine wave relative to an external timing pulse occurring at the sine wave's zero crossing is shown. Theoretically, only V_1 is needed to determine this phase shift. The advantage of using a second sample (V_2) spaced one half cycle later in time is that ($V_1 - V_2$)

measurement to a much greater extent. Because of this, the noise floor can be lowered another 20.6 dB by waveform averaging, producing 13.8 effective bits as shown in Fig. 10b.

The ac input path supports two digitizing functions: direct sampling and subsampling, which is also referred to as effective time sampling. The article on page 15 describes the subsampling technique. Subsampling allows the sampling of repetitive waveforms with effective sample intervals as short as 10 ns, thus allowing the user to take full advantage of the 12-MHz analog input bandwidth. The subsampling parameters are somewhat complex to calculate for an arbitrary effective interval and number of samples, but the user need not understand the details of the algorithm. All that need be specified is the desired effective sample interval and number of samples, and the HP 3458A will compute the number of passes, the number of samples per pass, the delay increment per pass, and the ADC sample rate required to complete the task most efficiently. Furthermore, if the samples are directed to the instrument's internal memory, they will be sorted into the correct time order on the fly.

If the number of samples required for a subsampled measurement exceeds the size of the instrument's internal memory, the samples can be sent directly from the ADC to a computer via the HP-IB. Since the HP 3458A cannot sort the data in this mode, the samples recieved by the computer generally will not be in the correct time order. If this is the case, the waveform can be reconstructed in the computer's memory using an algorithm requiring three sorting parameters supplied by the HP 3458A.

Subsampling is essentially the same as direct sampling when the effective sample rate is less than or equal to 50,000 samples per second. Why, then, is direct sampling even offered? The answer is that the subsampling technique only allows sampling based on the internal time base, whereas the direct sampling function includes all the same trigger modes as the dc volts function. This means that the user can supply an external time base via the external trigger input to allow sampling at odd frequencies that cannot be realized with the 100-ns quantization of the internal time base. An example would be the 44.1-kHz sample rate required by many digital audio applications. Direct sampling is also useful for acquiring single samples with minimum time uncertainty. Samples can be precisely placed with 10-ns delay resolution relative to an external trigger event and with 2-ns rms time jitter. "Measurement of Capacitor Dissipation Factor Using Digitizing" on this page shows an example of these measurement capabilities of the HP 3458A.

HP 3458A Limitations

Since the HP 3458A must be a voltmeter first and a digitizer second, it is not surprising that it has some limitations as a digitizer. Perhaps the most significant is the lack of an anti-aliasing filter. Because no single filter could be included to cover all possible sample rates, and because it would degrade the analog performance, the design team

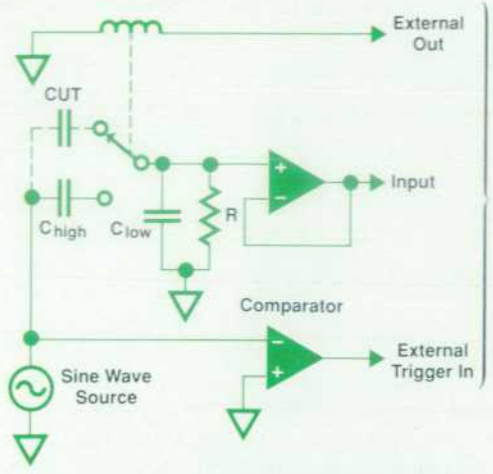


Fig. 2. *Circuit to measure dissipation factor.*

is insensitive to voltage offsets on the sine wave.

Fig. 2 shows a circuit using the technique of Fig. 1. A sine wave is applied to one of two capacitive dividers, one formed by the CUT and C_{low} and the other formed by C_{high} and C_{low} (R provides the dc bias for the buffer amplifier). This sine wave is also applied to a comparator that detects zero crossings. The output of the comparator is routed to the external trigger input of the HP 3458A and the output of the buffer amplifier is applied to the input of the HP 3458A. The HP 3458A can use its ac section to measure the rms value of this output waveform and thus V_p in Fig. 1 can be determined very precisely. The HP 3458A can also measure the period of the output waveform and set up sample timing parameters to sample the output sine wave relative to the external trigger signal as shown in Fig. 1. Thus all the information is present to determine the phase shift of the sine wave through the capacitor divider network.

The absolute phase shift of one side of the capacitor divider is not the information desired, however. What is desired is the phase shift caused by the dissipation factor of the CUT in the divider formed by the CUT and C_{low}. This will provide the information needed to determine the dissipation factor of the CUT.

Computing the difference between the absolute phase shift of the reference divider (C_{high} and C_{low}) and the input divider (CUT and C_{low}) is the first step towards determining the phase shift in the input divider resulting from the dissipation factor of the CUT. The HP 3458A's **EXT OUT** output is used to select either the reference divider or the input divider. Taking the phase difference between the reference and input measurements removes errors caused by the buffer amplifier and the comparator. If C_{high} had zero dissipation factor, CUT had the same capacitance value as C_{high}, and the switching relay was perfect, this phase difference would be entirely a result of the dissipation factor of the CUT. If this phase difference is ϕ, the dissipation factor of the CUT is:

$$DF = \tan(\phi)\ \frac{(CUT + C_{low})}{C_{low}}.$$

In general, the CUT will not be the same size as C_{high}, C_{high} will not have zero dissipation factor, and the switching relay will not be perfect. However, these conditions are easily controlled. The feedthrough capacitance of the relay in Fig. 2 can be reduced by implementing the relay as a T-switch. If the CUT is different in magnitude from C_{high}, a phase difference will be measured even if the CUT has zero dissipation factor. This is because the phase shift of the parallel combination of R and C_{high} and C_{low} is different from that of the combination of R and the CUT and C_{low}. This error can be removed by appropriate correction factors implemented in software. Also, in general, the dissipation factor of the CUT will not be zero. A zero calibration against a reference capacitor can remove this error.

Summary

The precision digitizing capabilities of the HP 3458A DMM have been applied to make a traditionally difficult measurement of capacitor dissipation factor. Test results show measurement accuracies approaching 0.001%. This corresponds to a phase error of 0.0005 degree or a time error of 150 ps at 10 kHz. Also, since the capacitance of the CUT is computed as part of the dissipation factor calculation, accurate capacitance measurements are also generated that are stable to 0.001 pF.

Ronald L. Swerlein
Development Engineer
Loveland Instrument Division

decided it would be impractical to include one.

Another limitation is the latency from an external or internal trigger to the start of sampling. The ramp time of the analog time interpolator produces a minimum delay of at least 400 ns. This means that if the input frequency is greater than about 500 kHz, the signal edge that is used to synchronize the waveform in a subsampled measurement will not even show up in the output data. Oscilloscopes typically include some form of analog delay to match the timing of the signal path to the trigger circuit, but again this was not compatible with the requirements of a high-precision DMM.

Another effect inherent in the design of the analog time interpolator is voltage droop. Essentially, the phase of the input signal relative to the internal 10-MHz clock is represented by a voltage stored on a hold capacitor, which is captured at the beginning of a measurement burst and held throughout the burst. Since there will always be some leakage in the circuits attached to this node, the voltage on this capacitor will slowly leak off, causing an apparent lengthening in the time between samples. This produces an apparent frequency modulation in the output data, which continues until the charge leaks off completely, at which time the sample interval will again be stable. This droop rate gets worse as leakage increases with higher temperature. Measurements on a typical unit at room temperature show a droop rate of about 500 ns/s, which persists for about 140 ms. In other words, during the first 140 ms of a reading burst, a sample interval of 20 μs will be lengthened by about 10 ps per sample.

Waveform Analysis Software

One factor limiting the effectiveness of the HP 3458A as a stand-alone digitizer is the lack of a built-in CRT for waveform display. This shortcoming has been addressed with a software library that turns an HP 3458A and a computer into a real-time single-channel digital oscilloscope and DFT analyzer.

The optional waveform analysis library allows a user with an HP 9000 Series 200 or 300 workstation or an IBM PC/AT-compatible computer with HP BASIC Language Processor to display waveforms in real time. In addition, routines are included to perform parametric analysis, waveform comparisons, and FFT spectral calculations and to store and recall waveforms from mass storage.

The real-time oscilloscope subprogram, Scope58, began as a means to demonstrate how quickly waveforms could be acquired by the HP 3458A and displayed. It soon became an indispensable tool in the development of the ADC and high-speed firmware. Since the program had proven so valuable during development, we decided it should be included in the waveform analysis library. A user interface was added to give the look and feel of a digital oscilloscope, including horizontal and vertical ranging, voltage and time markers, and an FFT display mode. The program can acquire and plot waveforms at a rate of approximately 10 updates per second—fast enough to provide a real-time feel.

The heart of the Scope58 subprogram is a set of specialized compiled subroutines for fast plotting, averaging, and interpolation of waveforms. Since speed was the overriding design consideration for these routines, most of these subroutines were written in MC68000 assembly language rather than a higher-level language like Pascal or BASIC. The fast plotting routine, in particular, required certain design compromises to achieve its high speed. It uses a simplified plotting algorithm which requires that there be one sample per horizontal display pixel, which means that the only way to change the horizontal scale is to change the sample rate unless the waveform data is interpolated to increase its time resolution before plotting. Also, the plotting routine bypasses the machine independent graphics routines and writes directly to the bit-mapped frame buffer of the graphics screen. This makes the routine fast, but it complicates the programming task, since a special version of the routine must be written for every supported display interface.

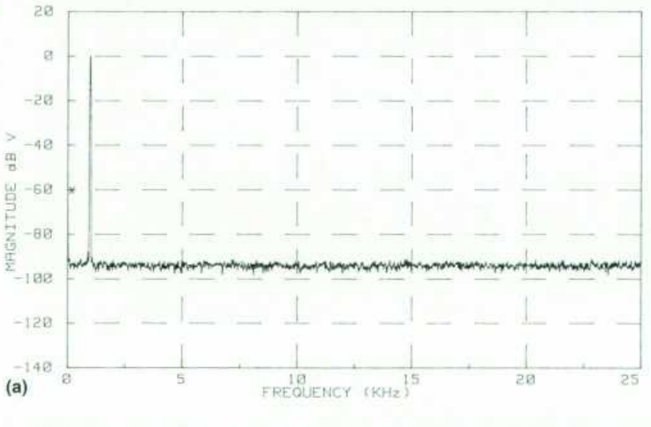
In addition to the Scope58 subprogram, the waveform analysis library includes routines that help with waveform acquisition, analysis, and storage. Since the HP 3458A is capable of synchronizing with external switching instruments like a normal DMM, it can be switched to acquire a waveform per channel in a multichannel data acquisition system. This feature, combined with the waveform analysis library, can be used to make many complex measurements in automated test applications.

The library's analysis capabilities include routines to extract parametric data such as rise time, pulse width, overshoot, and peak-to-peak voltage, and routines to compare waveforms against high and low limit arrays. There is also a compiled utility for calculating Fourier and inverse Fourier transforms. This routine can compute a 2048-time-point-to-1024-frequency-point transform in as little as 1.2 s if the computer's CPU includes a 68881 floating-point coprocessor. Finally, routines are provided for the interpolation of waveforms using the time convolution property of the sinc(x) function. This technique is common in digital oscilloscopes, and allows the accurate reconstruction of waveforms with frequency components approaching the Nyquist limit of half the sampling frequency.

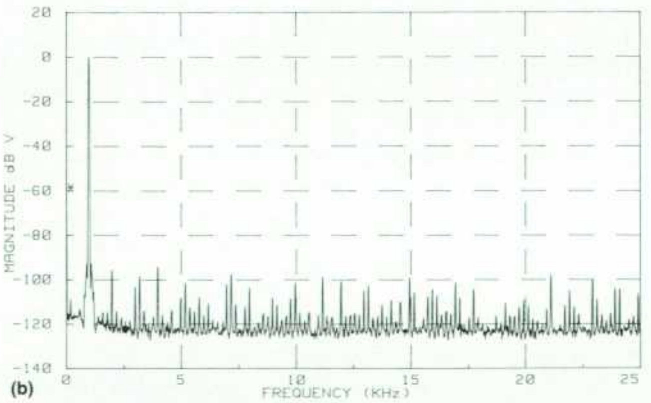
The precision digitizing characteristics of the HP 3458A and the display capabilities of the waveform analysis library combine to form a powerful waveform analysis tool in R&D or automated test applications. For instance, an HP 3458A together with a digital pattern generator can be used to test digital-to-analog converters (DACs). The waveform comparison capability of the waveform analysis library can be used to provide a pass/fail indication. Assuming a DAC settling time of 10 μs and an HP 3458A measurement time of 20 μs (only 10 μs of which is spent integrating the input signal), all codes of a 14-bit DAC (16,384 levels) can be acquired in approximately 328 ms.* The dynamic characteristics of the DAC can be tested using the FFT library routine. The DAC can be programmed to output a sine wave, which the HP 3458A can digitize. A DFT on the resulting data can be analyzed to characterize the DAC for noise floor and total harmonic distortion (THD).

Summary

The capabilities of a high-resolution digitizer can best be characterized by examining its performance in the frequency domain. To be able to resolve very low-level phenomena, it must have a wide dynamic range and very low levels of distortion and spurious signals. The excep-



(a)



(b)

Fig. 10. *(a) Typical DFT for 2048 samples of a 1-kHz sine wave sampled at the HP 3458A ac path's single-shot limit of 50,000 samples per second. Effective bits are 10.4. (b) Effective bits can be increased to 13.8 by averaging data for several acquisitions.*

*If you multiply 16,384 by 30 μs, the result is actually 492 ms. However, for at least 10 μs of each ADC conversion, the HP 3458A is not measuring the input, and provides a TTL signal indicating this fact. This time can be overlapped with the DAC's settling time, thereby reducing the total acquisition time.

tional DFT performance of the HP 3458A results from its combination of precise timing and the nearly ideal noise rejection capability of an integrating ADC. Also, its high-resolution track-and-hold circuit allows very fast sampling with maximal time certainty. These features, combined with the display capabilities of a host computer, are all that is needed to implement a high-resolution single-channel oscilloscope or DFT analyzer.

Acknowledgments

I would especially like to mention the contributions of Dave Rustici who implemented the original version of the real-time oscilloscope program, Evan Whitney of Santa Clara Division who wrote the sinc interpolation software, and Ron Swerlein who wrote the original compiled DFT routine. Also, I'd like to thank Brad Waite, who provided programming support on a prototype of the waveform analysis library, Greg Wale, who helped with software testing, and Brian Berry, whose inputs helped greatly to refine the definition of the waveform analysis library.

References

1. D.R. Flach, "Characterization of Waveform Recorders," *Digital Methods in Waveform Metrology*, NBS Special Publication 707, October 1985, pp 31-52.
2. B. Allen Montijo, "Digital Filtering in a High-Speed Digitizing Oscilloscope," *Hewlett-Packard Journal*, Vol. 39, no. 3, June 1988.

A Structured Approach to Software Defect Analysis

An effective software defect analysis requires that the relationships between program faults, human errors, and flaws in the design process be understood and characterized before corrective measures can be implemented.

by Takeshi Nakajo, Katsuhiko Sasabuchi, and Tadashi Akiyama

PROBLEMS THAT OCCUR IN SOFTWARE DEVELOPMENT because of human error negatively affect product quality and project productivity. To detect these problems as early as possible and prevent their recurrence, one approach is to identify flaws in present software development methodologies and procedures and recommend changes that will yield long-term defect prevention and process improvement. Typical approaches to software defect prevention have been to:

- Investigate only design methodologies and procedures and then recommend such things as different languages or more tools as defect prevention measures.
- Analyze the problems resulting from current design methodologies and procedures and develop solutions for each class of problem.

The first approach is the most widely used and has tended not to be data-driven, thus making the investigation tedious and the results ambiguous. In contrast, the analysis of problems tends to produce less ambiguous results and data collection is easier, but it has typically been used only to solve immediate problems and therefore has produced only short-term solutions.

To break out of the status quo, the instrument division of Yokogawa Hewlett-Packard (YHP) joined with Kume Laboratory of Tokyo University to analyze 523 software defects that occurred in three products developed by YHP. We tried to identify the flaws hiding in our current software design methodologies and procedures, and examine the impact of using the structured analysis and structured design (SA/SD) methods.[1,2] This paper discusses the results of this joint investigation.

Projects Investigated

The 523 software defects used for our investigation occurred during the development of three projects at YHP, which shall be called projects A, B, and C in this paper. Project A is a large all-software measurement system for analog-to-digital and digital-to-analog converters, and projects B and C are firmware for measurement instruments. 360 defects were studied from project A and 163 defects from projects B and C. These software systems have the following common characteristics:

- They are intended to control hardware, that is, initialization, setting registers, data retrieval, and so on. Therefore, they are greatly affected by the accuracy and clarity of hardware specifications.
- Their main parts are intrinsics, which are functions that can be used in measurement programs, or commands, which can be used sequentially to control devices.
- They are used to control hardware status, which means that they need many global variables to keep track of hardware states.

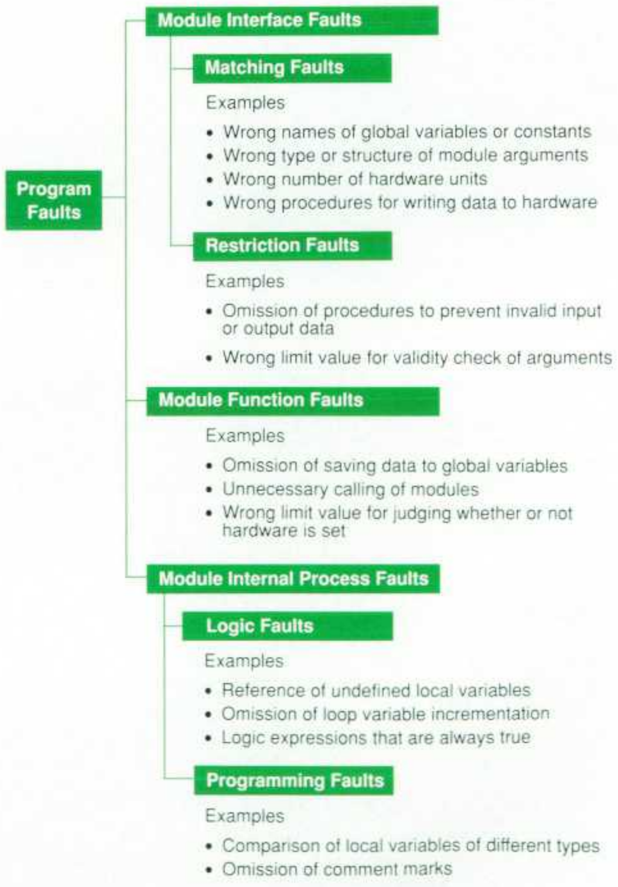

Program Faults
Module Interface Faults
Matching Faults
Examples
• Wrong names of global variables or constants
• Wrong type or structure of module arguments
• Wrong number of hardware units
• Wrong procedures for writing data to hardware
Restriction Faults
Examples
• Omission of procedures to prevent invalid input or output data
• Wrong limit value for validity check of arguments
Module Function Faults
Examples
• Omission of saving data to global variables
• Unnecessary calling of modules
• Wrong limit value for judging whether or not hardware is set
Module Internal Process Faults
Logic Faults
Examples
• Reference of undefined local variables
• Omission of loop variable incrementation
• Logic expressions that are always true
Programming Faults
Examples
• Comparison of local variables of different types
• Omission of comment marks

Fig. 1. *Types of program faults.*

Analyzing Software Defects

Three types of information useful for a defect analysis can be derived from a software defect: the human error (on the part of the developer), the program faults caused by the human error, and the flaws in the process causing the human error. Human error is an unintended departure from work standards or plans. Program faults are outright errors in the software which result in the system's crashing, producing wrong results, and in general not behaving as specified. Flaws are imperfections in the design methodologies or development procedures that affect the occurrence rate of human errors and the possibility of detecting human errors before they become program faults. Examples of flaws include no documentation, confusing specifications, nonstandard coding practices, bad methodology, no inspections, poor test planning, and so on.

To identify the flaws hiding in the design methodologies and procedures, we need to understand the mechanisms that cause human errors, and determine the relationship of these errors to program faults. This analysis is not easy because the human error process cannot be observed by objective methods, and usually, there isn't enough error data to analyze the relationship to program faults. However, the flaws must have some common factors, and they are reflected in the program faults caused by the human errors that occur during the design process. By design process we mean the portion of the software life cycle devoted to the definition and design of a product's features, software architecture, modules, and data structures.

Program Faults

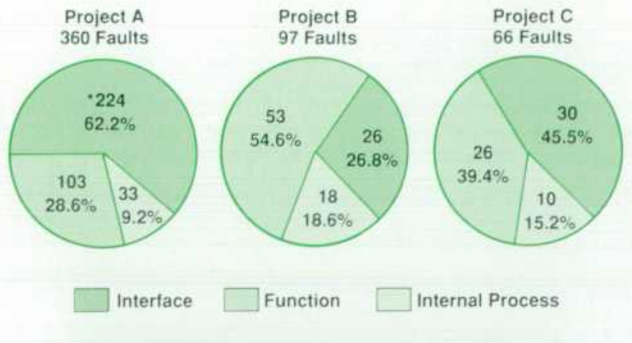
To identify the types of faults that occur in programs, it is necessary to study what caused the problem and what corrections were made to fix the problem. Classification of faults based only on their outward appearance does not work well. Categories of faults such as "wrong range of loop counters in DO statements" or "omission of conditions in IF statements" define the coding problem, but they do not provide a clear correspondence between the fault and the design process. We still need to know the role of each program segment in the system. For instance, in the DO loop range problem, was the range error related to the number of hardware units, or the length of the data file? Understanding program faults from the designer's point of view can help us link program faults to flaws in the design process. Fig. 1 shows our categorization of program faults along with examples of each category. Module interface faults relate to transferring data between modules, global variables, and hardware. Module function faults relate to a module's performing the wrong function. Module internal process faults correspond to logic errors, internal inconsistency, and programming rule violations.

Based upon the program fault classification given in Fig. 1, Fig. 2 shows the distribution of these faults among the three projects studied in this paper. The percentages of module interface faults and module function faults are similar for all three products (91%, 81%, and 85%). Since our design process was relatively the same for all three projects, we guessed that there must be some flaws in our design process associated with the way we do module interface definitions and module function definitions. Since module internal process faults had the lowest frequency of occurrence and because these faults are more directly related to the coding phase, they were not given further analysis.

Module Interface Faults. From Fig. 1, interface faults can be further classified into matching faults (mismatched data transfer between modules or hardware), and restriction faults (omission of checks on transferred data). The ratio of the number of matching faults to restriction faults turns out to be the same for all three projects and is about four to one. Consequently, we decided to focus our attention on matching faults for further study. Fig. 3 shows the five types of matching faults and their distribution for project A.* These five types of matching faults are defined as follows:

- Wrong correspondence between values of data and their meanings (e.g., storing the value r into a global variable that is supposed to contain the value r−1, or selecting the wrong destination hardware)
- Wrong data type, structure, or order (e.g., mismatch in the structure or order of arguments passed between programs, or mismatch in the order of arguments read from a data file or a hardware interface)
- Wrong correspondence between names and their mean-

*The same fault types and very similar distributions were discovered for projects B and C.



	Size (KNCSS)**	Language
Project A	318	Pascal, C
Project B	32	Assembly, C
Project C	110	Assembly, Pascal

*182 Matching Faults and 42 Restriction Faults
**KNCSS = thousands of noncomment source statements

Fig. 2. *Distribution of program faults for the three projects in this study.*

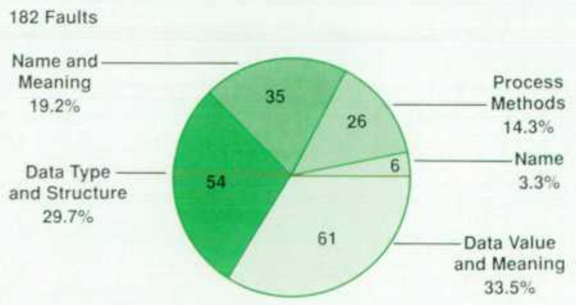


Fig. 3. *Distribution and types of module interface matching faults for project A.*

ings (e.g., using the wrong argument in a calling sequence, reading from the wrong global variable, or setting the wrong hardware registers)

- Wrong method of processing data (e.g., omission of certain steps when setting up hardware for some task such as a DMA transfer, or omission of initialization conditions or variables when calling other routines)
- Wrong name (e.g., using the wrong name to call a module or to access a global variable).

Module Function Faults. Function faults are program faults resulting from a module's performing the wrong internal operations. Fig. 4 shows the four types and the distribution of module function faults for project A.* These four types of function faults are defined as follows:

- Missing or unnecessary operations (e.g., failure to save calculated data to a global variable, or unnecessary calibration of hardware)
- Missing condition checks (e.g., saving data to a global variable before checking to see if it is permitted to save data to that particular variable)
- Wrong behavior of functions (e.g., making the wrong decision, or calculating the wrong value because the wrong coefficients are used in an equation)
- Wrong order of functions (e.g., checking whether a hardware unit exists after setting it).

The Design Process

Our design process for instrument control software consists of the following steps:

- Definition of unit functions and product features which are documented in the system external reference specifications (ERS)
- Definition of data structures and module interfaces which are documented in the internal reference specifications (IRS)
- Coding of each module
- Iteration through the previous steps as necessary.

*The same fault types and very similar distributions were discovered for projects B and C.

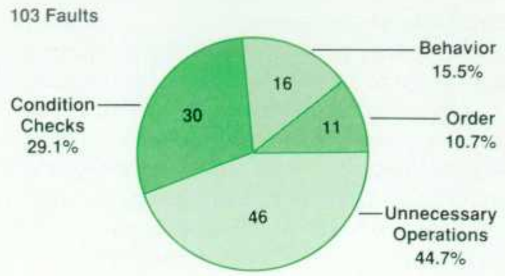
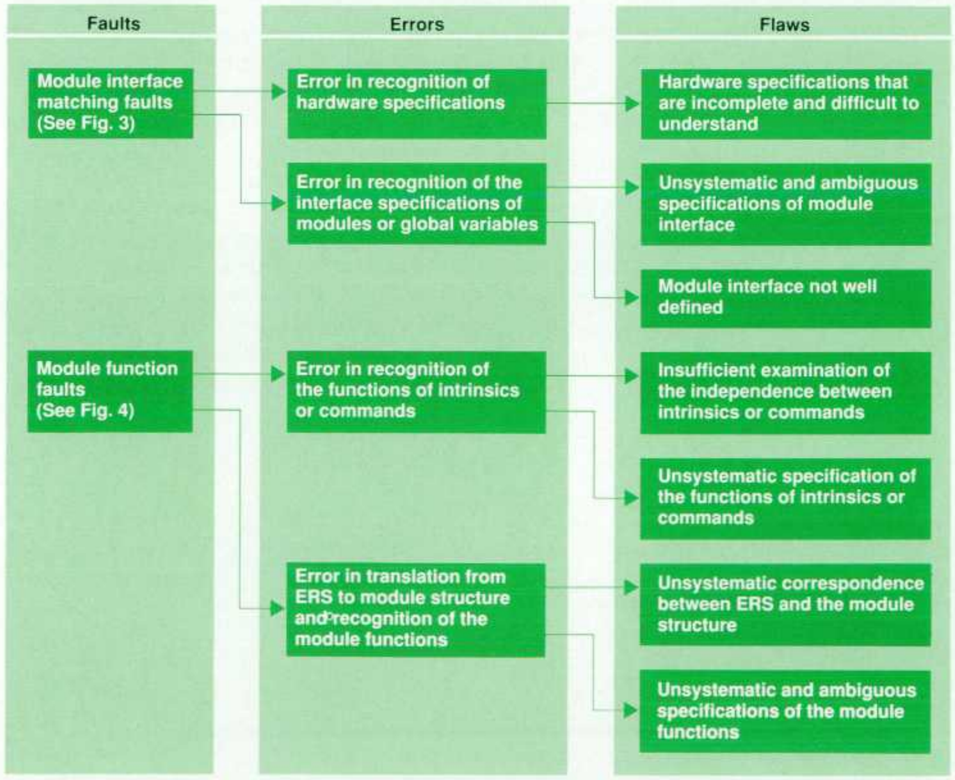


Fig. 4. *Distribution and types of module function faults for project A.*

Each of these steps includes the appropriate deliverables (specifications, test plans, etc.), and verification activities, such as design reviews and code inspections. Design reviews are done on the external and internal reference specifications, and code inspections are performed on selected modules.

These documents and procedures are intended to ensure that a defect-free product is eventually produced. However, this goal cannot be attained if we do not have a clear knowledge of the types of human errors that occur in the design process and of the features of the documents and procedures that affect the error occurrence rate and error detection. Consequently, we need to identify the types of human errors that cause program faults, the flaws in the present design documents and procedures, and the relationships



Faults
Errors
Flaws
Module interface matching faults (See Fig. 3)
Module function faults (See Fig. 4)
Error in recognition of hardware specifications
Error in recognition of the interface specifications of modules or global variables
Error in recognition of the functions of intrinsics or commands
Error in translation from ERS to module structure and recognition of the module functions
Hardware specifications that are incomplete and difficult to understand
Unsystematic and ambiguous specifications of module interface
Module interface not well defined
Insufficient examination of the independence between intrinsics or commands
Unsystematic specification of the functions of intrinsics or commands
Unsystematic correspondence between ERS and the module structure
Unsystematic and ambiguous specifications of the module functions

Fig. 5. *Relationship between program faults, human errors, and design process flaws.*

between them. From this perspective, we used the information gathered from investigating the two prevalent program fault types—module interface matching faults and module function faults—to derive the human errors associated with each fault type. These relationships were derived from interviews with the design engineers and our own analysis. Fig. 5 summarizes the relationships between the two main types of program faults, human errors, and flaws in the design process.

Human Errors and Process Flaws

Fig. 6 shows the distribution of the different types of human errors we discovered during our analysis. The human error that caused each software defect was not always clearly recorded. However, as we did for deriving the information in Fig. 5, we analyzed various documents and interviewed the designers and programmers who developed the system to come up with the numbers and percentages shown in Fig. 6.

Human Errors and Matching Faults. The human errors responsible for causing module interface matching faults are defined as follows:

- Module or Variable Specifications. Module interfaces and global variable definitions are missing or misunderstood.
- Hardware Specifications. Software developers overlook and misinterpret hardware specifications or other technical requirements of the hardware.
- Design Changes. Changes to the hardware interfaces, other related systems, or module interfaces are not communicated properly.
- Related System Requirements. Technical requirements are not communicated clearly between development groups and other related systems.

As shown in Fig. 6a, human errors associated with hardware interface specifications and module interfaces were the most frequent. Therefore, we examined the design process and found the following flaws associated with these error types.

- Hardware Specifications. Hardware specifications are difficult to read and understand for software engineers, and as a result, some important technical requirements about their interfaces were omitted. This flaw affected our external and internal design steps. We found that hardware interface information for writing software drivers was being derived from circuit diagrams, which were difficult for software developers to use without error.
- Module or Variable Specifications. The results of defining interfaces in the lower-level modules were not well-documented before defining internal algorithms and coding modules. Therefore, it was difficult to find module interface mismatching faults in design reviews. There was a lack of uniformity in the definition of certain features, and complicated interfaces between modules were not taken into consideration. These flaws also affect our internal design activities.

Human Errors and Function Faults. The human errors responsible for causing module function faults are defined as follows:

- Module Specifications. Errors in the translation from external specification to internal module specifications or misunderstanding of module specifications.
- Commands and Intrinsic Specifications. Misunderstanding the system external specification and providing the wrong or incomplete functionality for system features.
- Status Transition. Missing or misunderstanding the values of global variables that define the different state transitions of the system or hardware.
- Related System Requirements. Missing or misunderstanding the technical requirements of other related systems or hardware, resulting in such mishaps as the use of the wrong information from another subprogram to set hardware.

As shown in Fig. 6b, human errors associated with commands, instrinsics, and module functions were the most frequent. Therefore, we examined the design process and found the following flaws associated with these error types.

- Commands and Intrinsics. During the first part of our design process, when the external specification is defined, the independence between the functions of the commands and intrinsics was not sufficiently defined. For example, functions associated with the user interface were not partitioned properly, resulting in overlap in functionality, thereby causing program faults. Another problem was that the external specification documenting the commands, intrinsics, and other system requirements was not systematic. The specifications were mainly written in natural languages, which resulted in ambiguity regarding the uses and functions of commands and intrinsics.
- Module functions. During the internal design phase of our design process, when the modules and data structures are defined, developers designed the module structures based mainly on considerations about system per-

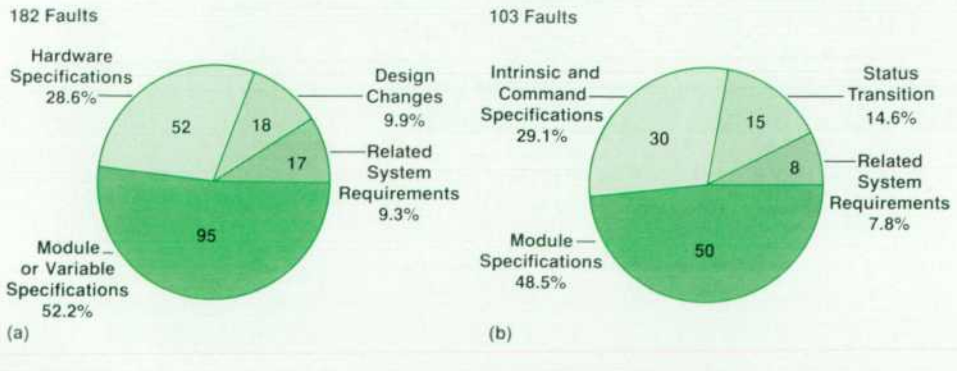


Fig. 6. *Distribution of human error types and frequency of occurrence for project A. a) Human errors related to module interface matching faults. b) Human errors related to module function faults.*

formance. This resulted in modules that had no clear correspondence with the system external specification. Another problem was that module functions were not completely specified before the internal algorithm and coding of each module were started. Internal design specifications also suffered from a lack of systematic documentation, resulting in ambiguous module functions.

Design Process Issues

In the previous section we determined the flaws in our design process that caused the human errors resulting in program faults in our products. Based upon what we learned about these flaws, three issues were derived from our analysis. Fig. 7 shows an example of the relationship between these issues and our design process.

Issue 1. A systematic method is needed to translate system features defined during product investigation into the details of a clear system external reference specification.

Issue 2. A systematic method is needed to translate external specifications into module structure and module functions.

Issue 3. A systematic method is needed to specify the technical requirements of hardware and to translate these requirements into software module interface specifications.

The above issues are vital to our design process. Since most of our products have similar characteristics, any solutions to these issues would pertain to all our software products. Issues 1 and 2 indicate that we need a method to translate from one level of abstraction to another, with each translation making it easier to perform a systematic engineering analysis of the system. With a good analysis methodology we can check the independence and sufficiency of functions and review their specifications to find unsuitable function definitions. Issue 3 requires that we have a methodology that enables hardware engineers to communicate hardware interfaces effectively to software engineers, and enables software engineers to communicate module interfaces and other system interfaces among themselves. With such a methodology, module structure and design can be effectively reviewed by the hardware and software engineers of the design team as well as those who must test and support the product.

SA/SD and Design Process Issues

Our investigation led us to believe that the structured analysis and structured design (SA/SD) methodology is the most suitable candidate for dealing with the three design process issues. We believe that SA/SD design methods can help prevent program faults by enabling us to detect and correct these problems before they become real defects.

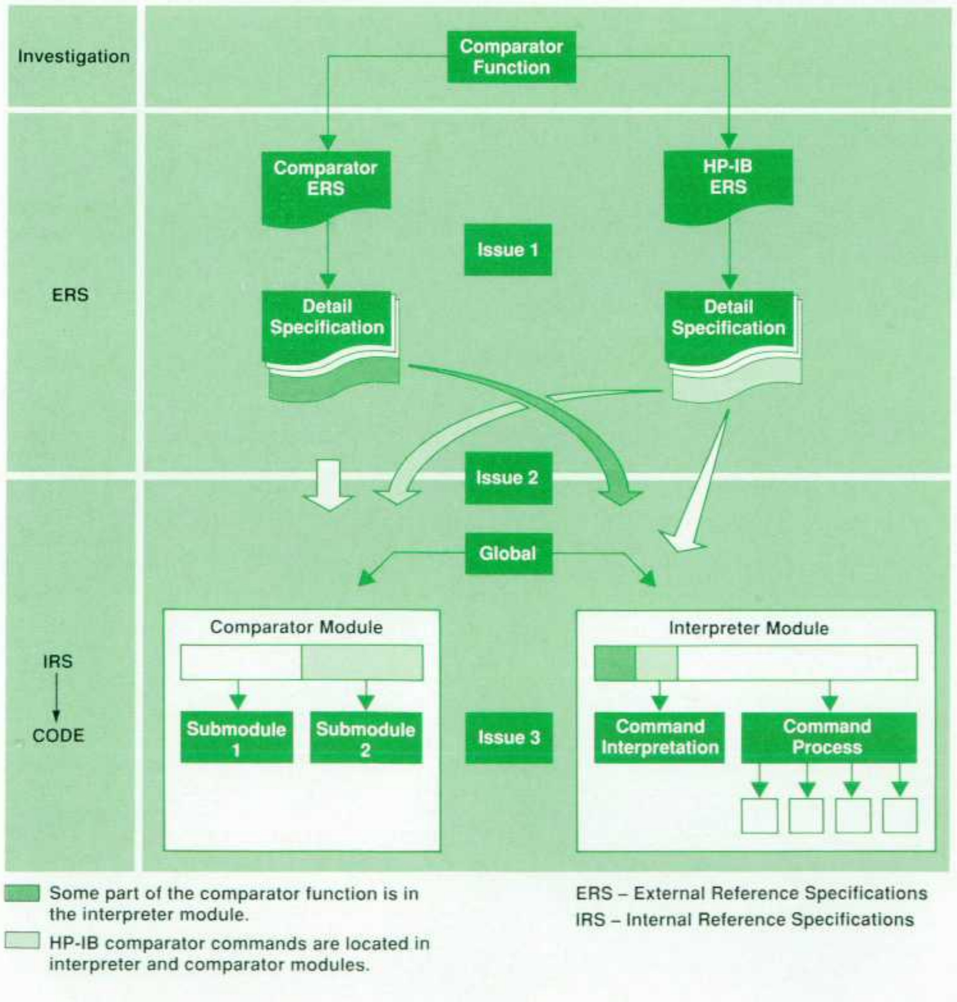


Fig. 7. *An example of where the design issues occur in the design process.*

Fig. 8 shows the correspondence between the three design issues and the solutions offered by SA/SD methods.

Proposed Solution for Issue 1. The key elements of structured analysis we found useful for dealing with issue 1 include:

- Context diagrams, which define the relationship between the software system and its environment (e.g., relationship between the hardware and the firmware elements in an instrument)
- Data flow diagrams, which define the actions of processes (modules or functions) in the system, and the data and control flows between these modules
- Process specifications, which define the functions and behavior of the processes in a precise structured language.

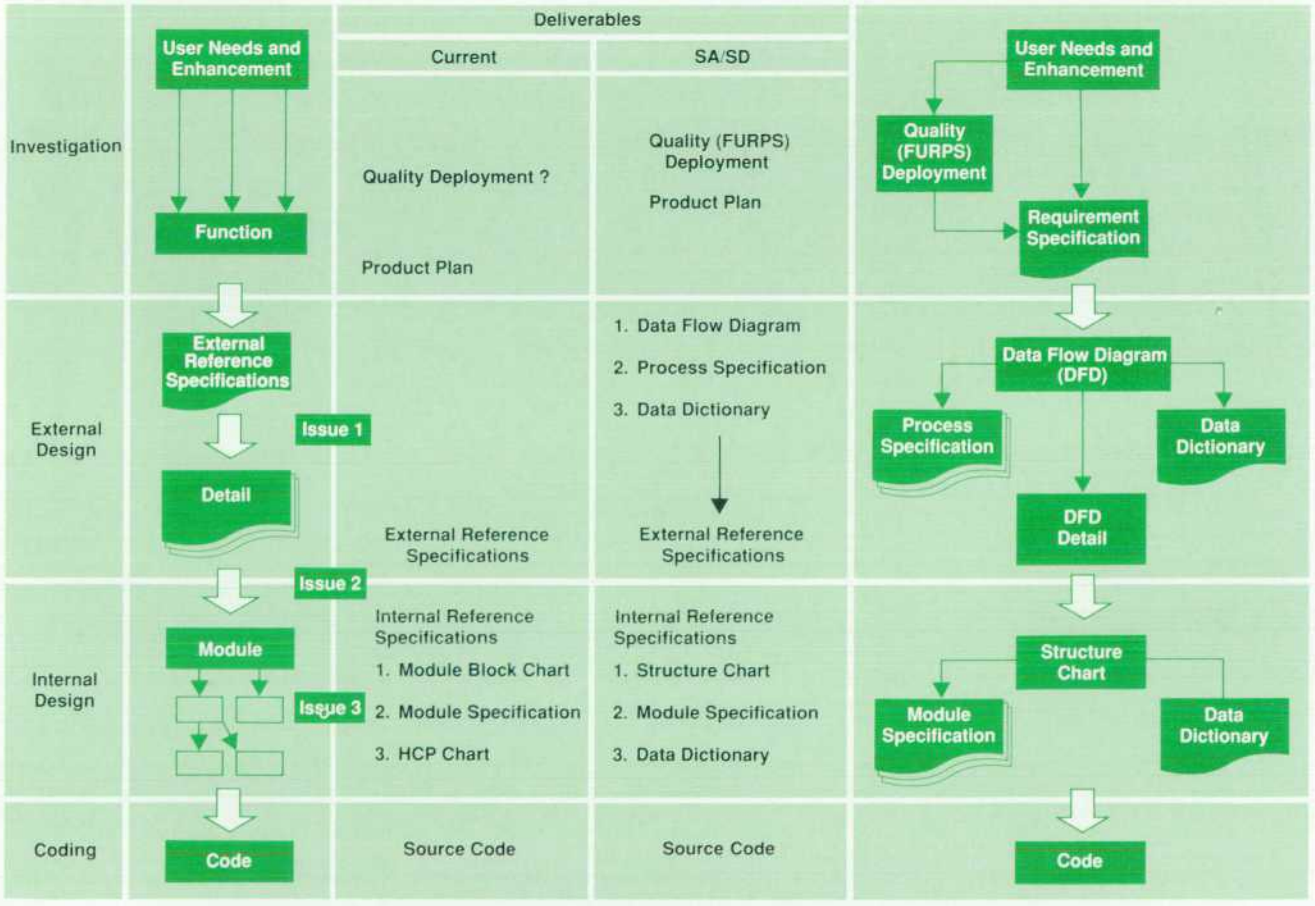
The functions we define for our systems are organized based on their relationship with data. The functions that depend on each other are difficult to classify into simple groups. In structured analysis, detailed functions of each intrinsic or command and their relationships can be represented by data flow diagrams. Also, the data relationships are clearly specified, and the operation of each function is defined in a structured language in the process specification.

Proposed Solution for Issue 2. The system external specification can be smoothly translated to the module structure by using the transformation analysis technique provided by SA/SD. Transformation analysis enables us to take each data flow diagram and transform it into more detailed data flow diagrams or module structure. By applying this method, we can make a module structure that has a clear correspondence to the system external specification.

Proposed Solution for Issue 3. The key elements of structured design we found useful for dealing with issue 3 include:

- Structure charts, which define the module hierarchy within a data flow diagram or within a module
- Module specifications, which define in a structured language the function of each module
- Data dictionaries, which define the data that flows between modules.

Among these elements, the data dictionary provides us with the greatest leverage to solve issue 3. With the data dictionary we can systematically specify the interfaces to the hardware and the interfaces between the software modules. With these interfaces consistently defined we can



FURPS – Acronym for Functionality, Usability, Reliability, Performance, and Supportability
HCP – Hierarchical and Compact Description Chart

Fig. 8. *Positioning of SA/SD methods in the software development process. Quality deployment is a tool that helps designers to focus on the features in the product that are important to meeting customer needs.*

easily detect mismatches between modules and hardware.

Conclusion

In this investigation, we tried to identify the flaws hiding in our current software design methodology and procedures and examine possible countermeasures against them. We analyzed about five hundred actual problems that occurred during software development for three instruments and used these defects as a basis for our investigation.

We believe that SA/SD methods can solve some of our design problems. However, there are still some challenges, which include:

- Elimination of the inconsistencies between the present specifications using natural languages and the new specifications using the SA/SD methods
- Installation of automated tools for using the SA/SD methods
- Establishment of an appropriate education and training system on the SA/SD methods for the software engineers
- Preparation of other groups in our division for dealing with documents written using SA/SD methods
- Establishment of design review methods based on the SA/SD methods
- Investigation and use of other tools and techniques provided by SA/SD, such as state transition diagrams and transactional analysis
- Investigation to find ways to model software behavior that cannot be analyzed with current SA/SD methods.

Acknowledgments

The authors wish to thank the following people: YHP instrument division manager Mitsutoshi Mori for his useful advice, Drs. H. Kume and Y. Iizuka of Tokyo University for their valuable comments, I. Azuma in the product assurance section for his help with the analysis of the software defects for one project, and the software engineers in the research and development sections for their help with the collection of the software defect information.

References

1. E. Yourdon and L.L. Constantine, *Structured Design*, Prentice-Hall,Inc., 1979.
2. T. DeMarco, *Structured Analysis and System Specification*, Prentice-Hall, Inc., 1979.

CORRECTIONS

Author David Chu has sent us two corrections for his article, "Phase Digitizing: A New Method for Capturing and Analyzing Spread-Spectrum Signals," in the February issue. Page 32, second column, last paragraph (just above Fig. 7) should read "Advanced phase means earlier in time, so $\delta_\phi(t_i)$ and $\delta_t(t_i)$ are always opposite in sign." In Fig. 2, the signal shown is BFSK, not BPSK.

Also in the February issue, in Fig. 2b on page 7, ΔE should be $2\pi\Delta E$ in three places, and in Fig. 4a on page 9, f_c should be $2\pi f_c$.

Dissecting Software Failures

Beyond collecting software defect data just to study defect frequency, this paper outlines a quality data collection process, an effective analysis process, and a method to justify changes in the software development process based on the defect analysis.

by Robert B. Grady

MOST PEOPLE DON'T LIKE TO BE TOLD that they've made a mistake. It's only human not to want to be wrong. On the other hand, software engineers don't intentionally make mistakes, so if we can understand why mistakes occur without accusing individuals, we might eliminate the causes of those mistakes. Unfortunately, discussions concerning software defects are confusing because different people describe them from different perspectives.

This paper discusses some of the terminology of these different views. It then examines some simple data collection and analysis techniques that help identify causes of defects and point to areas where improvements can be made. Finally, it presents some guidelines for justifying change based upon the results of analyses.

"A defect is any flaw in the specification, design, or implementation of a product."[1] Such flaws cause managers to lose control by reducing their ability to predict when development or maintenance will be completed. Thus, we encounter another human trait: people like to be in control of a situation. The opportunity, then, is for software developers and managers to record sufficient defect data while analyzing and resolving defects to understand and remove the causes of those defects.

Defect Perspectives

Fig. 1 illustrates three views of a defect. Each of these views is characterized by its own terminology and focus. When users of a product have a problem, they know that they can't get their job done because the software product isn't performing the way they expect it to. The level of their concern reflects how much their business is impacted, and terms like critical or serious mean that they stand to lose substantial time and/or money if something isn't done soon.

On the other hand, individuals responsible for communicating defect information and status to and from customers refer to which component is at fault, whether a patch exists, and when a permanent fix can be expected. They must extract enough detail from customers to discover workarounds and to provide maintainers enough information to seek a permanent fix.

The third perspective is that of the people responsible for maintaining and enhancing software. They speak in terms of what code was at fault, the priority associated with correcting the defect, how difficult it will be to fix, and when to expect the fix.

If we draw an analogy to medicine, the patient describes the problem in terms of what hurts and how much. The nurse setting up the appointment must ask enough questions to tell the doctor enough to form preliminary conclusions and to determine how urgent it is for the doctor to see the patient. The doctor must run tests to discover the real cause of the ailment and must prescribe the correct treatment to heal the patient.

Data Collection

The first step in treating software defects is data collection. Most organizations already gather much of the necessary data. What is proposed is a method to use the data for a long-term quality improvement effort, not just to solve the current problems.

For example, there is always justifiable pressure to fix urgent problems. The goal is to maximize customer satisfaction (with an emphasis on timeliness, in this case). In pursuit of that goal, data is collected to optimize the flow of information from the customer about a defect (see Fig. 2). Additional data is collected as engineers investigate the problem to provide information to the customer regarding status and possible fixes. Once customer satisfaction is

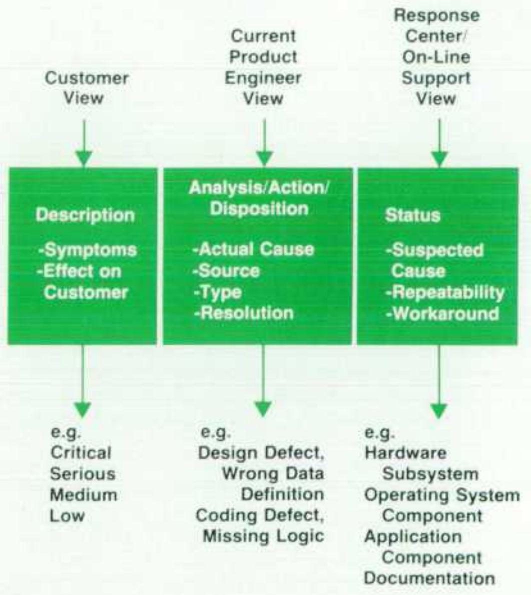

Customer View
Current Product Engineer View
Response Center/ On-Line Support View
Description
-Symptoms
-Effect on Customer
Analysis/Action/ Disposition
-Actual Cause
-Source
-Type
-Resolution
Status
-Suspected Cause
-Repeatability
-Workaround
e.g. Critical Serious Medium Low
e.g. Design Defect, Wrong Data Definition Coding Defect, Missing Logic
e.g. Hardware Subsystem Operating System Component Application Component Documentation

Fig. 1. *Different views of a defect based upon the responsibility for dealing with the defect.*

achieved and the customer has a workaround or permanent fix for the problem, data collection should not stop.

If we want to learn from past mistakes to improve development or support practices, then a small additional amount of time must be spent to collect additional data. What are some of the questions that this long-term goal prompts, and what data is needed to answer the questions? Some of the more obvious questions are:

1. What development or maintenance process failed?
2. How often do such failures occur?
3. How expensive is it to fix such failures?
4. Which components are most subject to failure?
5. What process change will detect or eliminate these failures?

Fig. 3 shows an example of the additional data needed for the defect described in Fig. 2. The numbers in Fig. 3 are related to the questions above. Question 2 could only be answered by analyzing the defect type information for a number of similar defect fix reports.

Resistance to data collection when defects are being fixed is natural, because there may be a backlog of defects and strong schedule pressures. A common request is for additional automation aids to capture the information suggested in Fig. 3, and until then, no data is collected. Such requests sometimes miss the point, however, and fall into the trap of what we'll call the "automation syndrome." In fact it is unlikely that entry of such data into an automated system would shorten the time involved on the part of the engineers reporting it. The problem with the automation syndrome is that it can prevent the collection of needed data for years if simple solutions and firm management don't prevail.

We need to ask what it costs (in time) to collect this additional data. Let's take a typical product of 100 KNCSS (thousands of noncomment source statements). We have seen average postrelease defect densities from less than 0.1 to as high as 1 or 2 defects per KNCSS in the first year after release of a product. For the sake of calculations, let us assume a very high value of 1 defect/KNCSS. For our product, then, we would expect 100 defects in the first year. If we look at the data requested in Fig. 3, it seems likely that it would take between five and ten minutes per defect to provide the fix information requested in Fig. 3. This means that the total incremental engineering hours for our defect-plagued 100-KNCSS product might vary from one to slightly over two days (see Fig. 4). Not a very large investment for such valuable data.

Suppose the product that you are going to perform post-release analysis on is ten times as large, or that you want to perform your analysis before product release (where we typically see about ten times the number of defects as in postrelease). Data collection time is always a sensitive issue, so you should consider how to capture the information that you need while minimizing the data collection time. This is done by collecting sufficient samples of defect data to yield an effective distribution of causes. The total will be adequate as long as the sample size is large enough (probably 100 to 150 samples) and sufficiently random. For example, you might restrict the capture of the additional data shown in Fig. 3 to just critical and serious defects, or turn selection into a game by rolling dice before starting work on each defect.

The goal of the data collection scheme is to optimize the amount and quality of information and to minimize the time (cost) spent by those supplying the information, whether the collection method is manual or automated.

Data Validation

When you initiate the collection of a new set of data, adjustments are needed in people's activities and existing processes. It is particularly important at the start to include procedures to ensure valid data.

A common cause of invalid data is different interpretations of definitions. These are somewhat alleviated by proper training before collection begins, but all too often we incorrectly assume that everyone will interpret instructions in the same way. For example, two different HP divisions reported that many defects labeled as coding defects were really caused by changed requirements or design. The incorrect labeling occurred because the defects were discovered or fixed during the coding phase.

It is desirable to reinforce initial collection of defect information with subsequent interviews. These should be initiated by the project manager or a representative to ensure that the data is accurate and to emphasize the importance of accuracy to the engineers reporting the data. These checks should examine a large cross section of the data in depth. Once this is accomplished, spot checks are probably all that are needed to maintain the flow of good data.

Data Analysis

In the previous sections, the focus was on collection of

Submitter: Bruce Davis	Date Submitted: 8/22/84
Company Name: Hewlett-Packard	Dept.: SEL
Support Engineer: John Michaels	Support Office: Factory
Computer/System Model: 3000/930	Identification No.: 0-13-821844-7
Defective Software: MPE-XL	Release Version: MIT X.B6.06
Severity (Critical, Serious, Medium, Low): Serious	
Workaround? (Y/N): Y	(easy/difficult)?: Difficult

Symptoms: System crashes with abort number of 1072. This has happened twice in the last week. At the time of the crash, there was one user on the system, mgr. official. The job running was newjobx1.

Fig. 2. *Simplified defect report.*

Fixed by: Lynn Smith
Date fixed: 9/19/84

③ Engineering Hours to Find and Fix: 66
① Defect Origin: Design
② Defect Type: Data Definition
Category of Defect
(Missing, Unclear, Wrong, Changed, Other): Wrong

④ Modules Changed: Disc_lo, Table 5
Other modules affected: Interx

⑤ How could defect have been found earlier:
Design walkthrough; more complete test coverage; more timely data dictionary updates.

Fig. 3. *Defect fix information. The numbers refer to the questions in the article.*

valid data. The next step of the process is the analysis of the data collected. Again, there is a danger that nothing will happen, because many managers have never taken the time to perform such an analysis. They believe that the time involved will be too great. Perhaps this belief is as unfounded as the one concerning the data collection time.

What are the steps involved in a typical analysis? The following estimates assume that the analysis is begun with 100 one-page completed defect reports and is done manually.

1. Sort the data collection forms by defect origin. Count the number in each group and total the number of engineering hours to fix the defects for each group. Arrange the totals in descending order of total engineering hours (30 min).
2. Calculate the average fix time for each of the totals from step 1 (5 min).
3. For the top two or three totals in step 1, count the defects sorted by defect type and multiply by the appropriate average fix times. Limit the number of types to the largest totals plus a single total for all others (15 min).
4. Add up the defects sorted by module changed. Limit the number of choices to the five most frequent plus a single total for all others (15 min).
5. Review the defect reports for the defects included in the largest totals from steps 3 and 4, and summarize the suggestions for how the defects could have been found earlier (1 hour).

Following the procedure above, project managers would know several valuable facts after only about two hours time. They would know what the most costly defects were, when they occurred, where they occurred, and the most likely steps to take to prevent their occurrence in the future.

But even two hours of a project manager's time is sometimes difficult to find. Other useful alternatives that have been successfully tried are to use engineers from a quality or metrics organization or to hire a student from a local university to perform the analysis.

$$\text{Product Size} \times \begin{array}{c}\text{High Average}\\ \text{Postrelease}\\ \text{Defect Density}\end{array} \times \begin{array}{c}\text{Time to Record}\\ \text{Data}\end{array} = \begin{array}{c}\text{Engineering}\\ \text{Cost}\end{array}$$

$$100\ \text{KNCSS} \times 1\ \text{Defect/KNCSS} \times \left\{ \begin{array}{l} 1/12\ \text{Hour/Defect} = 8\ 1/3\ \text{Hours} \\ 1/6\ \ \text{Hour/Defect} = 16\ 2/3\ \text{Hours} \end{array} \right.$$

Fig. 4. *Sample calculation of the cost of collecting defect cause data.*

A Model for Analyzing Defect Causes

Various reports have documented successful efforts to analyze defects, their causes, and proposed solutions.[2-11] But the terminology among them has differed considerably, and the definitions could possibly mean different things to different people. In the fall of 1986 the HP Software Metrics Council addressed the definition of standard categories of defect causes. Our goal was to provide standard terminology for defects that different HP projects and labs could use to report, analyze, and focus efforts to eliminate defects and their causes.

Fortunately, the IEEE had a subcommittee working on a standard for defect classification,[12] so it was possible to start from their working documents. The IEEE definitions covered all phases of defect tracking in an extensive general way. These will undoubtedly be of value to the people supporting defect tracking systems. Unfortunately, the IEEE document at that time was very long and too general to be applied specifically to any project. As a result, the metrics council extracted only the material related to defect

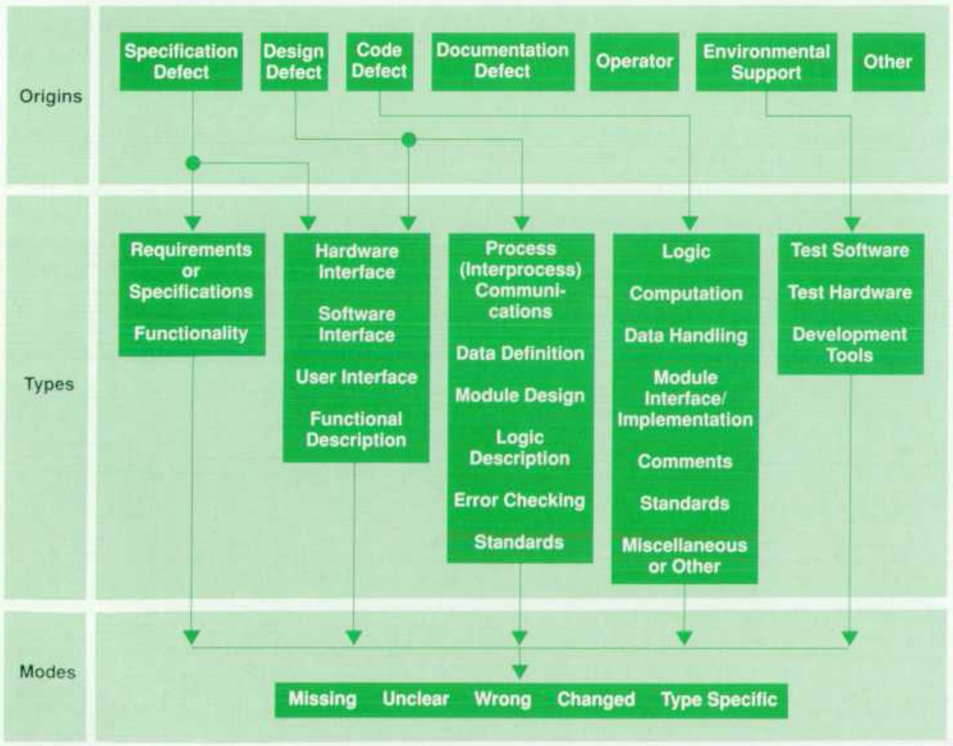


Fig. 5. *Categorization of sources of software defects.*

causes and produced a metrics guideline that is easier to use.[13] Fig. 5 illustrates a model of defect sources taken from the guideline, and the box on page 62 gives the definitions from the guideline.

The model is used by selecting one descriptor each from origins, types, and modes for each defect report as it is resolved. For example, a defect might be a design defect where part of the user interface described in the internal specification is missing. Another defect might be a coding defect where some logic is wrong.

An Example

Let us look at a specific example using the model presented in Fig. 5. The data for this example is taken from a detailed study of defect causes done at HP.[11] In the study, defect data was gathered after testing began. Fig. 6 shows the data sorted by the primary origins of defects.

It is desirable to focus attention on the causes of defects that cost the most to fix. The net cost of any given classification is represented by the total defects for the classification times the average cost to fix those defects. This study didn't accurately record the engineering times to fix the defects, so we will use average times summarized from several other studies to weight the defect origins.[14] In particular, the average engineering cost to fix coding defects that are not found until testing is about 2.5 times the cost of those found during coding. The factors for design and specification defects that are not found until testing are about 6.25 and 14.25, respectively. Fig. 7 shows the relative costs to fix the defect population from Fig. 6 when the weighting factors are applied. For the sake of this example, the other origins are assumed to have a multiplier of one, and we will normalize all fix times to assume that a coding defect fixed during coding takes one hour. The weighting factors then simply become engineering hours to fix the various defect categories.

These two figures illustrate step 1 of the five-step procedure described earlier and the weighting factors in Fig. 7 represent step 2. The study from which this data was taken only provided defect type data for coding and design defects. Therefore, we will perform step 3 of our procedure with a breakdown of data for only coding and design. This is shown in Fig. 8. It suggests that efforts should be focused on eliminating logic errors, computation errors, and process communication errors before final test.

These brief examples show how easy it is to apply the analysis procedure to discover where changes with the greatest impact can be made. They also show how incomplete data can force us to make assumptions that might impact the accuracy of our conclusions. In this example we didn't have complete data regarding specifications defects or data detailing engineering hours for all defects. These are probably not serious drawbacks in this case, but one must be certain to identify such uncertainties and their potential effects every time an analysis is performed.

Here is an interesting note to conclude our example. The use of weighting factors in the analysis above emphasized working on eliminating causes of problems that cost the most to resolve. The assumption was that we would institute process changes to eliminate the causes of those defects. An excellent source of these changes would be the suggestions collected when defects are resolved. If the emphasis is to reduce the defect backlog as quickly as possible, then the effort must be focused on those problems that are

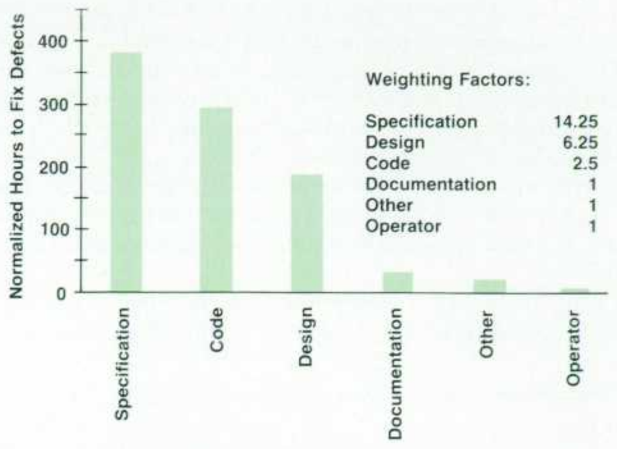


Fig. 7. *Weighted distribution of defect origins.*

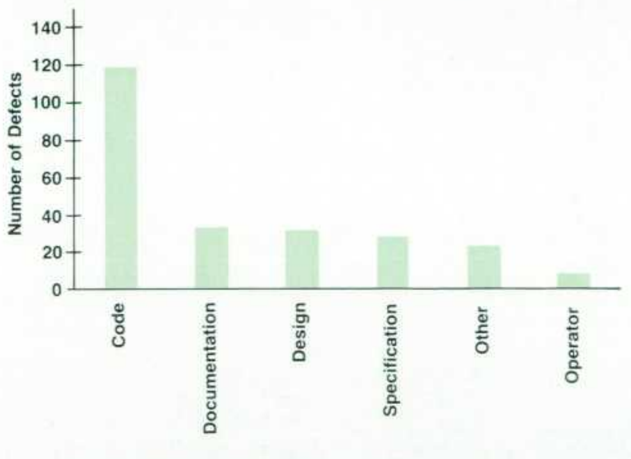


Fig. 6. *Distribution of defect origins.*

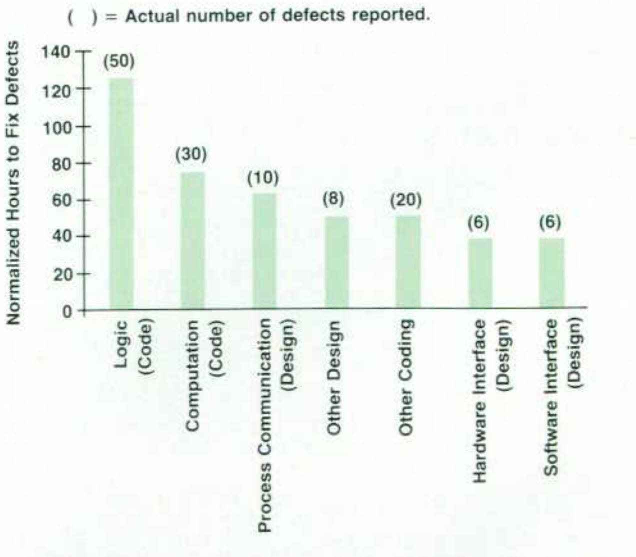


Fig. 8. *Weighted distribution of defect types (coding and design defects only).*

easiest to fix quickly. In that case, we would simply view the data differently to learn the answer. We would look for defect groups consisting of large numbers of defects that can be resolved quickly (e.g., documentation or some types of coding defects).

Justifying Change

Once you have collected the data necessary to understand which defects impact your operation and analyzed the data to determine what your tactics should be, you encounter the most difficult step. This step is to recommend and implement change based upon the discovered facts. It is the most difficult because of another human characteristic: resistance to change. There are many facets to such resistance that span the entire implementation process. These are discussed in detail elsewhere,[15] so this paper will focus on the first step in the change process only, that of initial justification of change. Recommendations for change take many forms, but most successful changes are based upon a cost/benefit analysis built from components such as those outlined in Fig. 9.

Most of the entries in the benefit column of an actual table would be represented in measurable units based on data already collected and analyzed using the techniques described earlier. The remaining items would be estimates. The entire table should be bound to a specific time period, such as one year. A summary table can be constructed from an appropriate subset of items, costs, and benefits given in Fig. 9 that should be convincing by itself. For extra emphasis, it can be supplemented by benefits beyond a year and by more-difficult-to-measure customer satisfaction benefits and increased sales.

An Example

Let's build a case for change based on data from two studies done at HP. The first study investigated the number of engineering hours used to find and fix different defect types primarily during implementation and test.[16] It found:

Average number of design defects = 18% of total defects.

The second study evaluated various factors related to design and code inspections.[17] It found:

- Optimum number of inspectors = 4 to 5
- Ratio of preparation to inspection time > 1.75
- Inspection rate = 300 to 400 lines of design text/hour
- Average number of defects found per inspection hour = 2.5.

Estimated number of design defects (average of 8 defects/KNCSS[18]) in 100 KNCSS of code:

$$100\text{ KNCSS} \times \frac{8\text{ defects}}{\text{KNCSS}} \times \frac{18\text{ design defects}}{100\text{ defects}} \doteq \text{144 design defects}$$

Time cost to find the design defects using inspections (assume 55% might be found[19]):

$$\frac{\text{144 design defects} \times 0.55}{\text{2.5 defects found/inspection hour}} \times \text{4.5 engineers} \times \frac{\text{(1.75 preparation + 1 inspection hour)}}{\text{1 inspection hour}} = \text{392 engineering hours}$$

Time to find the same 79 (144 × 0.55) design defects during test (same cost ratio used in Fig. 7[14]):

$$\text{392 design-find hours} \times \frac{\text{6.25 find/fix hours in test}}{\text{1 find/fix hour in design}} = \text{2450 engineering hours}$$

Net savings: 2450 engineering hours − 392 engineering hours = 2058 engineering hours to find defects

(a)

Items	Costs	Benefits
Training	48 engineering hours	
Start-up costs	48 engineering hours 0.5 month	
Reduced defect finding time		2058 engineering hours 2 to 4 months
Time to market		1-1/2 to 3-1/2 months

(b)

Fig. 10. *Results of using information from several studies to show a justification for design inspections. a) Analysis of times to find design defects. b) Sample cost/benefit analysis of design inspections.*

Fig. 10 shows the results of combining the information from these two studies into a justification of design inspections for a 100-KNCSS project.

Note that neither costs nor benefits were specified for the item management. For simplification we assume that roughly the same management time will be needed to introduce the new concepts to a project team as would have normally been needed to manage the additional engineering hours using the old techniques.

In summary, the introduction of design reviews seems

Items	Costs	Benefits
Training	Class Costs Reduced Time Devoted to Project	Future Expertise
Management	Increased Management Time	Reduced Management Time
Engineering	Start-Up Costs	Reduced Defect-Finding Times Reduced Defect-Fix Times Reduced Number of Defects Reduced Number of Factory Engineers Reduced Number of Field Engineers Shorter Development and/or Build Cycles
Capital Expenses	Purchased Hardware Purchased Software	
Time to Market		Product Available Sooner
Job Complexity		Less Experienced Engineers Required

Fig. 9. *Factors to consider when recommending change and measuring progress.*

Defect Origins and Types

Enhancement. A change that could not possibly have been detected, or, if detected, would not have been corrected.

An enhancement is not a defect. Restraint must be exercised when a software change is labeled as an enhancement. The use of the term enhancement should be restricted to those cases where the customer's needs and/or the product scope have truly changed since the release of the product, thereby creating new requirements that could not have been anticipated in the original development effort. For example, the performance of a software product was competitive upon release, but it needed to be improved two years later to remain competitive. Such a change is an enhancement. If the performance was not competitive at the original time of release, then any subsequent change to improve performance is considered a defect fix.

Specification Defect. A mistake in a specification that sets forth the requirements for a system or system component. Such mistakes can be in functional requirements, performance requirements, interface requirements, design requirements, development standards, etc.

- Requirements or Specifications. The specifications do not adequately describe the needs of target users. Also includes the effects of product strategy redirection and nonexistent product specifications.
- Functionality. Problems with the product feature set (e.g., incorrect or incompatible features). Includes cases where functionality is increased to add market value.
- Hardware, Software, and User Interface. Problems with incorrect specification of how the product will interact with its environment and/or users.
- Functional Description. Incorrect description of what the product does. Generally discovered during requirements or design inspection.

Design Defect. A mistake in the design of a system or system component. Such mistakes can be in system or component algorithms, control logic, data structures, data set use information, input/output formats, and interface descriptions.

- Hardware, Software, and User Interface. Problems with incorrect design of how the product will interact with its environment and/or users. For example, incorrect use of libraries, design does not implement requirements, device capabilities overlooked or unused, or design does not meet usability goals.
- Functional Description. Design does not effectively convey what the intended module or product should do. Generally a defect found during design inspection or during implementation (coding).
- Process or Interprocess Communications. Problems with the interfaces and communications between processes within the product.
- Data Definition. Incorrect design of the data structures to be used in the module/product.
- Module Design. Problems with the control (logic) flow and execution within processes.
- Logic Description. Design is incorrect in conveying the intended algorithm or logic flow. Generally a defect found during design inspection or implementation.
- Error Checking. Incorrect error condition checking.
- Standards. Design does not adhere to locally accepted design standards.

Code Defect. A mistake in entry of a computer program into the symbolic form that can be accepted by a processor.

- Logic. Forgotten cases or steps, duplicate logic, extreme conditions neglected, unnecessary function, or misinterpretation errors.
- Computation Problems. Equation insufficient or incorrect, precision loss, or sign convention fault.
- Data Handling Problems. Initialized data incorrectly, accessed or stored data incorrectly, scaling or units of data incorrect, dimensioned data incorrectly, or scope of data incorrect.
- Module interface/Implementation. Problems related to the calling of, parameter definition of, and termination of subprocesses. For instance, incorrect number of, or order of, subroutine arguments, ambiguous termination value for a function, or data types incorrect.
- Comments. Insufficient or incorrect commenting.
- Standards. Code does not adhere to locally accepted coding standard.
- Miscellaneous (other): This classification should be used sparingly, and when it is used, the defect should be very carefully and extensively described in associated documentation.

Documentation Defect. A mistake in any documentation related to the product software, except in requirements specification documents, design documents, or code listings. Mistakes in the latter three are assumed to be specifications defects, design defects, and coding defects, respectively.

Operator Defect: Any situation that involves the operator's misunderstanding of procedures, hitting the wrong button, entering the wrong input, etc. Does not necessarily imply that the product is in error.

Environmental Support Defect. Defects that arise as a result of the system development and/or testing environment.

- Test Software. Problems in software used to test the product software's capabilities. For example, another application program, the operating system, or simulation software.
- Test Hardware. Problems with the hardware used to run the test software, *not* the hardware on which the product software runs.
- Development Tools. Problems that are a result of development tools not behaving according to specification or in a predictable manner.

Other. This classification should be used sparingly, and when it is used, the defect should be very carefully and extensively described in associated documentation.

to be very desirable. The benefits in this example totally overwhelm the costs, so why aren't inspections more widely used today? It gets back to the issue of resistance to change. Remember that while this example is based on real data, it is suspect since the data was measured by someone else and is derived from several sources. When you justify change, you must organize your arguments as clearly and persuasively as possible, and you must be prepared to continue trying to persuade the people involved until the change has occurred.

The example was selected to illustrate the process of justifying change. The core of the justification was the data recorded in previous studies of defects and the times taken to resolve them. You can use such published data to help guide your decisions, but ultimately you must also collect enough data that is specific to your process or products to verify that the problems you pursue are the most important ones.

Conclusion

Managers of software development cannot afford to continue producing and supporting products with the same old techniques and processes. The field is changing rapidly, and improvements in both quality and productivity are necessary to remain competitive. The history of the application of software metrics includes the continuous application of basic scientific methods. We collect data and establish hypotheses for improvements. We take additional measurements to prove or disprove the hypotheses. And we revise our hypotheses accordingly and start the process again. The major problem of management without the use of data is that the hypotheses can never be really validated and institutionalized.

If we return to our medical analogy, it is like medical doctors having to practice medicine without understanding the human body through dissections and autopsies. For over a thousand years before the fifteenth century, medical doctors were prevented from dissecting human bodies because of fear and superstition. When the rules against dissections were eased, great progress occurred in a relatively short time. We must experience a similar renaissance period in software development. Perhaps it is time that our schools began to teach "software autopsies."

The techniques described here for collecting, analyzing, and presenting data are simple, yet effective means to improve software development. We saw that the collection of a small amount of additional data can yield a large payback in terms of useful information that fits into a standard framework for analysis. A five-step process for data analysis was given that organizes this information to point to areas and methods for improvement. And a framework for justifying change to both management and engineers suggests how changes are proposed initially and justified. What remains is for managers to use these techniques as quickly as possible to promote positive change.

Acknowledgments

The members of the HP Software Metrics Council who particularly contributed to the Defect Categorization Guidelines were Dave Classick, Craig Fuget, Mike Gourlay, Bob Horenstein, Chuck Leath, and Kathy Osborne. I would also like to thank Jan Grady, Debbie Caswell, Ken Hoyt, Barbara Noble, Danny Lau, and Sally Dudley for their helpful suggestions regarding this article's development.

References

1. R. Grady and D. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, Inc., 1987, p. 224.
2. M. L. Shooman, "Types, Distribution, and Test and Correction Times for Programming Errors," *IEEE Proceedings of the 1975 Conference on Reliable Software*, Los Angeles, Calif., April 1975, pp. 347-357.
3. A. Endres, "An Analysis of Errors and Their Causes in System Programs," *IEEE Transactions on Software Engineering*, Vol. SE-1, no. 2, June 1975, pp. 140-149.
4. R. J. Rubey, J. A. Dana, and P. W. Biche, "Quantitative Aspects of Software Validation," *IEEE Transactions on Software Engineering*, Vol. SE-1, no. 2, June 1975, pp. 150-155.
5. B. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. MacLeod, and M. J. Merritt, "Characteristics of Software Quality," *TRW Series of Software Technology*, Vol. 1. Amsterdam: TRW and North-Holland Publishing Company, 1978.
6. N.F. Schneidewind and H.M. Hoffman, "An Experiment in Software Error Data Collection and Analysis," *IEEE Transactions on Software Engineering*, Vol. SE-5, no. 3, May 1979, pp. 276-286.
7. C. Sieloff, "Software TQC: Improving the Software Development Process Through Statistical Quality Control," *HP Software Productivity Conference Proceedings*, April 1984, pp. 2-49 to 2-62.
8. G. Hamilton, "Improving Software Development Using Quality Control," *HP Software Productivity Conference Proceedings*, April 1985, pp. 1-96 to 1-102.
9. D. Kenyon, "Implementing a Software Metrics Program," *HP Software Productivity Conference Proceedings*, April 1985, pp. 1-103 to 1-117.
10. C. Fuget, "Using Quality Metrics to Improve Life Cycle Productivity," *HP Software Productivity Conference Proceedings*, April 1986, pp. 1-86 to 1-93.
11. C. Leath, "A Software Defect Analysis," *HP Software Productivity Conference Proceedings*, April 1987, pp. 4-147 to 4-161.
12. *A Standard for Software Errors, Faults, and Failures*, IEEE working group P1044, March 1987.
13. "Software Development Metrics Guideline: Defect Analysis," June 1987, (internal HP memorandum).
14. B. Boehm, *Software Engineering Economics*, Prentice-Hall, Inc., 1981, p. 40. Reprinted by permission of Prentice-Hall, Inc.
15. R. Grady and D. Caswell, *op. cit.*, pp. 82-95.
16. R. Grady, "Measuring and Managing Software Maintenance," *IEEE Software*, September 1987, pp. 35-45.
17. B. Scott and D. Decot, "Inspections at DSD—Automating Data Input and Data Analysis," *HP Software Productivity Conference Proceedings*, April 1985, pp. 1-79 to 1-89.
18. R. Grady and D. Caswell, *op. cit.*, p. 145.
19. C. Jones, *Programming Productivity*, McGraw-Hill Book Co., 1986, p. 179.

Software Defect Prevention Using McCabe's Complexity Metric

HP's Waltham Division has started to use this methodology and its associated tools to catch defect prone software modules early and to assist in the testing process.

by William T. Ward

IT IS POSSIBLE TO STUDY, MEASURE, AND QUANTIFY many aspects of the software development process, and if sufficient data about good practices used in recently released projects is available, real-time adjustments can be made to ongoing projects to minimize past mistakes and to leverage ideas from past successes.

HP's Waltham Division has maintained an extensive software quality metrics data base for products developed here over the past three years. We have been able to use this data base during project postmortem studies to provide insight into the strengths and weaknesses of Waltham's software development process.

Fig. 1 lists the basic software quality metrics for two major Waltham Division products that have been released within the past two years. Based on the extensive amount of code, both of these products can be classified as large-scale firmware projects. These projects had a short development time and a very low postrelease defect density. Since these products are considered technical successes, it was suggested that the software development data we had about them could be studied to improve our understanding of Waltham's software development process. This resulted in a formal effort to examine the project data in more detail.

A substantial amount of software process data was evaluated during the course of the study. This data represented each phase of the development process and addressed both quality and productivity issues (e.g., defect density and engineering hours). The results of the evaluation resulted in a set of recommendations that covered code inspections, development tools, testing, and people and process issues such as code reuse and code leveraging.

Since every issue could not be addressed at once, we decided to find one area in the development process that had the greatest need for improvement and would provide the greatest return on our process improvement funds. We wanted to select methodologies and tools that could be used to improve the weak process area and could be easily integrated into our development environment.

Process Improvement Area

Fig. 2 shows the relative percentage of prerelease software defects based on the development phase where the defect was inserted. The data shown here is from Project B, but similar results were found for Project A. Initially we were surprised by this data. It might be assumed, for instance, that defects are introduced into a product in equal amounts throughout each phase of development, or that the product design phase might be the most troublesome. However, the data presented here accurately represents the Waltham process, which of course may be different from other environments.

Since 60% of the total prerelease defects were introduced into the product during the implementation phase, it was obvious that any improvement in this phase would yield the greatest benefit. During the implementation phase the activities that occur include coding, inspections, debug-

	Project A	Project B
1. Code size (KNCSS*)	125	77
2. Number of prerelease defects requiring fix	269	133
3. Prerelease defect density (defects/KNCSS)	2.15	1.72
4. Calendar months for prerelease QA	9	6
5. Total prerelease QA test hours	3370	2055
6. Number of postrelease defects reported after one year	2	24
7. Postrelease defect density (defects/KNCSS)	0.02	0.31
8. Calendar months from investigation checkpoint to release	23	24

*KNCSS: Thousands of lines of noncomment source statements.

Fig. 1. *Basic software quality metrics for two HP Waltham Division products.*

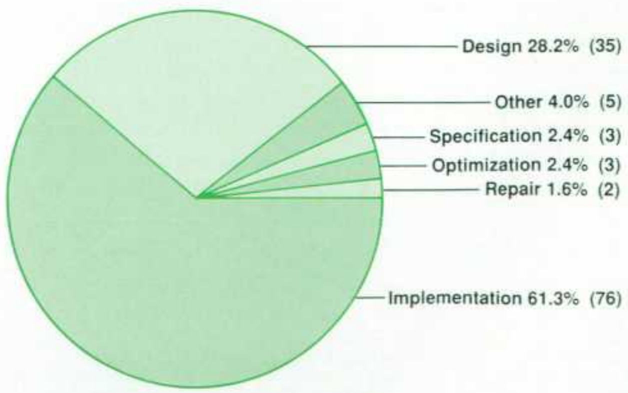


Fig. 2. *Summary of defects by phase for project B.*

ging, and all testing.

Finding a Methodology

Closer investigation of our metrics data base revealed that some modules were more defect-prone than others. These troublesome modules consumed a great deal of time and effort during the implementation phase. Therefore, we needed a method to identify these modules early so that we could take the appropriate corrective measures, such as more intensive code inspections.

After examining the current software engineering literature[1,2,3] and further scrutinizing of our project data, we found McCabe's cyclomatic complexity metric and its associated methodologies best suited our needs. This metric provided us with a measure for detecting error-prone modules and a methodology that fit right into our development process. The McCabe metric is a number that represents the complexity of a module. It is based on the number of decision statements in the module. It has been found that if the complexity measure of a module exceeds 10 the chance of that module being error-prone also increases. See the article on page 66 for more details.

Fig. 3 shows some of the project data we used to help us evaluate the utility of the McCabe metric. This graph shows a comparison between prerelease defect density and the complexity metric for programs belonging to project B (similar results were found for project A). Each program in Fig. 3 is a collection of many small modules and the complexity value shown is the sum of the complexity measures for all of the modules in a particular program. From this data we were able to compute a 0.8 (or 64%) statistical correlation between complexity and defect density.

Methodology and Tools

The McCabe metric has been around for a while and its correlation between the metric and defect-prone modules has been validated in the literature.[4,5,6] We found the following additional issues during our investigation of the McCabe metric and its associated methodology.

- The algorithm for calculating the McCabe metric for each module is very simple and the process for gathering the data to compute the metric can be automated.
- The McCabe metric is expressed as a unitless number. Industry experience suggests that a complexity measure in the range of 1 to 10 per code module is optimal for producing quality code. In fact, some organizations place a limit of 10 on all modules.
- The McCabe metric can play an important role in the module testing process. A methodology has been developed that allows determination of test paths and test cases using the complexity metric and the accompanying program flow graphs.
- The cyclomatic complexity of a code module can be presented graphically as well as numerically, and there are tools for plotting representations of modules as cyclomatic flow graphs.

Implementing Process Improvements

The primary goal of this effort was to find a methodology that would help reduce the number of defects introduced into a product during the implementation phase of development. Once a methodology was found, our next goal was to integrate it into the real, heavily loaded, often skeptical R&D environment. We have successfully incorporated the McCabe methodology into our software development process by using it in early recognition of code quality, testing, code inspections, and the software quality engineering process.

Recogition of Code Quality. As mentioned previously, the cyclomatic complexity of a module can be represented either numerically or graphically. As an example, consider Fig. 4. This diagram is the flow graph of a module written in C, which is part of a current development project at Waltham. This module has a cyclomatic complexity value of seven, which indicates a well-constructed module that may have a low defect density, or possibly no defects at all. The flow graph has been constructed using specific shapes to represent various programming structures. For instance, in this example IF statements are shown as branches and WHILE statements are shown as loops. The complete syntax of a language such as C can be illustrated in this manner. The numbers on the flow graph correspond

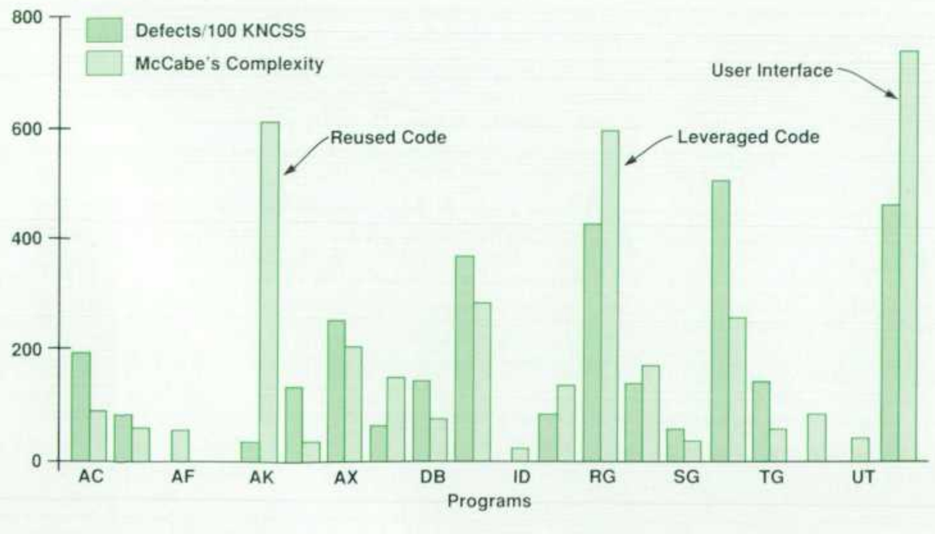


Fig. 3. *Comparison of defect densities and McCabe's complexity for programs in project B. The McCabe complexity value is the summation of all the complexities for the modules in a particular program. Reused code is code that is used with no changes, and leveraged code is modified reused code.*

The Cyclomatic Complexity Metric

The quantification of program complexity is related to the number of decisions (changes in control) in the program. This is opposed to the viewpoint that complexity can be quantified from program size or the number of independent program paths. Program size is misleading because a large program may have very few decision statements. For example, a 2-KNCSS program may have only one or two decisions implying one or two paths, whereas a 50-line program with 25 if-then statements in sequence could generate 33.5 million paths. Basing code complexity strictly on the number of paths is also misleading because the number of paths can be infinite for programs that have loops.

To provide a metric that indicates a meaningful set of program paths, the cyclomatic complexity metric quantifies a basic number of paths that have the following properties:

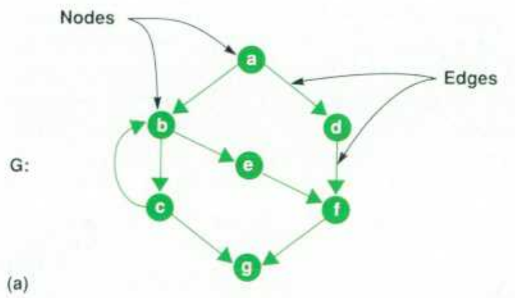
- They visit every node (program statement) in a graph of the program, and they visit every edge (change of control) in the graph.
- When taken together the basic paths can generate all possible paths in the program.

To develop these concepts, a definition and theorem are needed from graph theory.

Definition 1. The cyclomatic number v(G) of a graph G with n nodes, e edges, and 1 connected component is:

$$v(G) = e - n + 1$$

A connected component is a code module (function or procedure) from start to end.



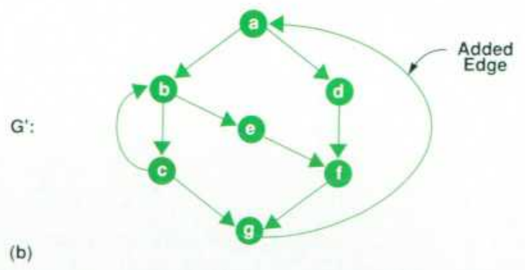


Fig. 1. *a) Program flow graph for a program with seven nodes (blocks of code) and ten edges (branches). b) Same control graph with added edge to satisfy the requirement that the graph must be strongly connected.*

Theorem 1. In a strongly connected graph G, the cyclomatic number is equal to the maximum number of linearly independent paths.

To apply this theory a program must be represented as a directed graph in which a node represents a sequential block of code, and an edge corresponds to a branch (transfer of control) between nodes (see Fig. 1). It is assumed that each node is entered at the beginning and exits only at the end.

The program flow graph in Fig. 1 has seven blocks (a through g), entry and exit nodes a and g, and ten edges. To apply the theorem the graph must be strongly connected, which means that, given the two nodes a and b, there exists a path from a to b and a path from b to a. To satisfy this, we associate an additional edge with the graph that branches from the exit node g to the entry node a as shown in Fig. 1b. Theorem 1 now applies, and it states that the maximal number of states in G' is $11 - 7 + 1 = 5$. The implication is that there is a basic set of five independent paths that when taken in combination will generate all paths. The five sets of paths for G' are:

b1 : abcg
b2 : a(bc)*2g { (bc)*2 means iterate loop bc twice }
b3 : abefg
b4 : adefg
b5 : adfg

If any arbitrary path is chosen, it should be equal to a linear combination of the basis paths b1 to b5. For example, the path abcbefg is equal to b2 + b3 − b1, and path a(bc)*3g equals 2*b2 − b1.

The general form of the complexity metric for a module is $v(G) = e - n + 2$. The association of an additional edge from exit to entry for each module is implicit. Therefore, we have:

$$v(G) = (e + 1) - n + 1 = e - n + 2$$

Applications of v(G)

The cyclomatic complexity metric has applications in the following areas:

- The cyclomatic number has often been used in limiting the complexity of modules in a software project. Experience and empirical data[1] have suggested that there is a step function in defect density above a complexity of 10. Therefore, it is good practice to limit the complexity of each software module to 10. The one exception to this rule is a case statement, which contains an arbitrary number of independent paths. This structure can have a high cyclomatic complexity and have a low defect density.
- The cyclomatic number can be used as a predictor of defects. Various projects have correlated cyclomatic complexity and defect density and have reported correlations of 0.8 as in the accompanying paper to 9.6.[2]
- The cyclomatic number and the accompanying program control flow graph can be used to identify test cases. The cyclomatic number corresponds to the number of test paths and the test paths correspond to the basic paths derived from the control flow graph. With this information, test conditions (test cases) can be generated for a program. This entire process can be automated with a language dependent preprocessor. The following example illustrates the derivation of test paths and test cases using the cyclomatic number and the control

graph.

The following procedure in C solves the well-known triangle graph problem, which takes three integers and determines what type of triangle they represent.

```
Node numbers
in Fig. 2

0           main () {
1                     int          a,
1                                  b,
1                                  c;
2                     scanf ("%d%d%d"  &a,&b,&c);
3                     if (a > = b + c)
4                       printf ("no triangle\n");
7                     else
7                       if (b > = a + c)
8                           printf ( "no triangle\n");
10                      else
10                          if (c > = a + b)
11                              printf ("no triangle\n");
13                          else
13                              if (a = b)
14                                  if (b = c)
15                                      printf("equilateral\n");
18                                  else
16,18                                   printf("isosceles\n");
19                              else
19                                  if (a = c)
20                                      printf("isosceles\n");
22                                  else
22                                      if(b =c)
23                                          printf("isosceles\n");
25                                      else
25,24,21,17,12,9,6,5    printf ("scalene\n");
5,6         }
```

This program has 32 edges and 26 nodes giving a complexity of eight (see Fig. 2). This means there are eight test paths for this program. The following list shows the first five test paths and the test conditions for each path.

Test Path	Node	Test Conditions
1. 0 1 2 3 4 5 6	3	(a >= b + c) → TRUE
2. 0 1 2 3 7 8 9 5 6	3	(a >= b + c) → FALSE
	7	(b >= a + c) → TRUE

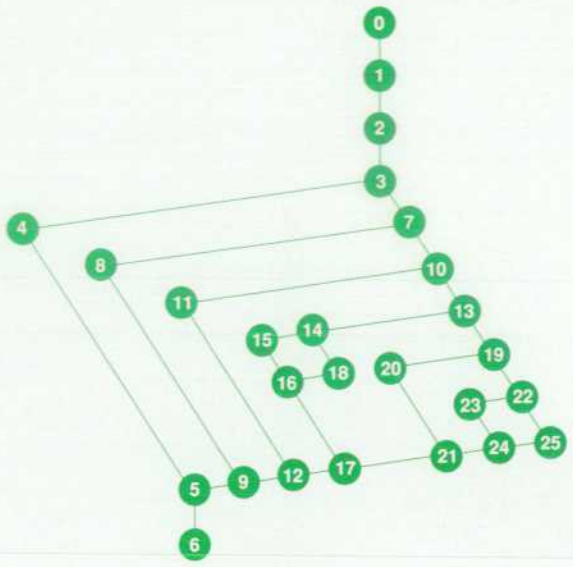


Fig. 2. *Program flow graph for the triangle graph problem.*

Test Path	Node	Test Conditions
3. 0 1 2 3 7 10 11 12 9 5 6	3	(a >= b + c) → FALSE
	7	(b >= a + c) → FALSE
	10	(c >= a + b) → TRUE
4. 0 1 2 3 7 10 13 19 20 21 17 12 9 5 6	3	(a >= b + c) → FALSE
	7	(b >= a + c) → FALSE
	10	(c >= a + b) → FALSE
	13	(a = b) → FALSE
	19	(a = c) → TRUE
5. 0 1 2 3 7 10 13 14 18 16 17 12 9 5 6	3	(a >= b + c) → FALSE
	7	(b >= a + c) → FALSE
	10	(c >= a + b) → FALSE
	13	(a = b) → TRUE
	19	(a = c) → FALSE

The symbol → indicates that the conditions in parenthesis must be set to TRUE or FALSE.

References

1. T. J. Walsh, "A Software Reliability Study Using a Complexity Measure," *Proceedings of the National Computer Conference,* AFIPS, 1979.
2. S. Henry, D. Kafura, and K. Harris, "On the Relationships Among Three Software Metrics," *1981 ACM Workshop/Symposium on Measurement and Evaluation of Software Quality,* March 1981.

Thomas J. McCabe
President
McCabe and Associates

to actual C source statements. This representation provides a useful reference between the flow graph and the code.

Fig. 5 is a similar diagram for another C code module from the same development project. Note here that the cyclomatic complexity is 36, and that the flow graph for the code is more complex. Since this module's complexity metric exceeds the optimal value of 10 it is likely that this module will be error-prone. In addition, Fig. 5 provides visual evidence that a complex module may be hard to understand, test, and maintain.

Our experience at Waltham indicates that the graphical representation of code complexity is a very effective vehicle for focusing lab-wide attention on code quality. The visual impact of an image of tangled code appears to attract more interest than mere correlation of numbers. Therefore, current projects are actively using cyclomatic flow graphs during the coding process to focus engineering and management attention on code quality.

Testing Process. The test case generation capability of the McCabe methodology has been very useful in establishing rigorous module testing procedures. The cyclomatic complexity values have been used as an indicator of which modules should be subjected to the most active scrutiny by the test group. Modules with abnormally high complexity values are selected as candidates for the most extensive test activities.

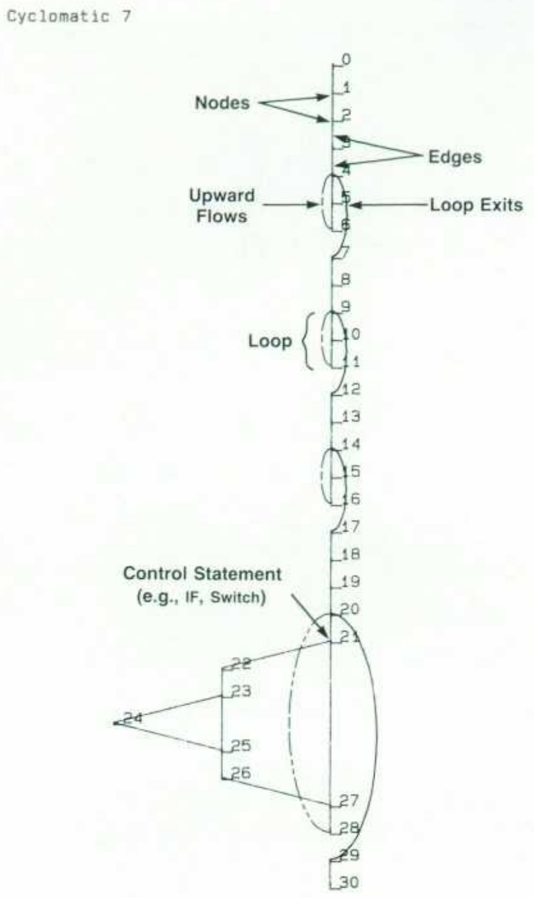


Fig. 4. *The program flow graph for a module with a cyclomatic complexity of 7.*

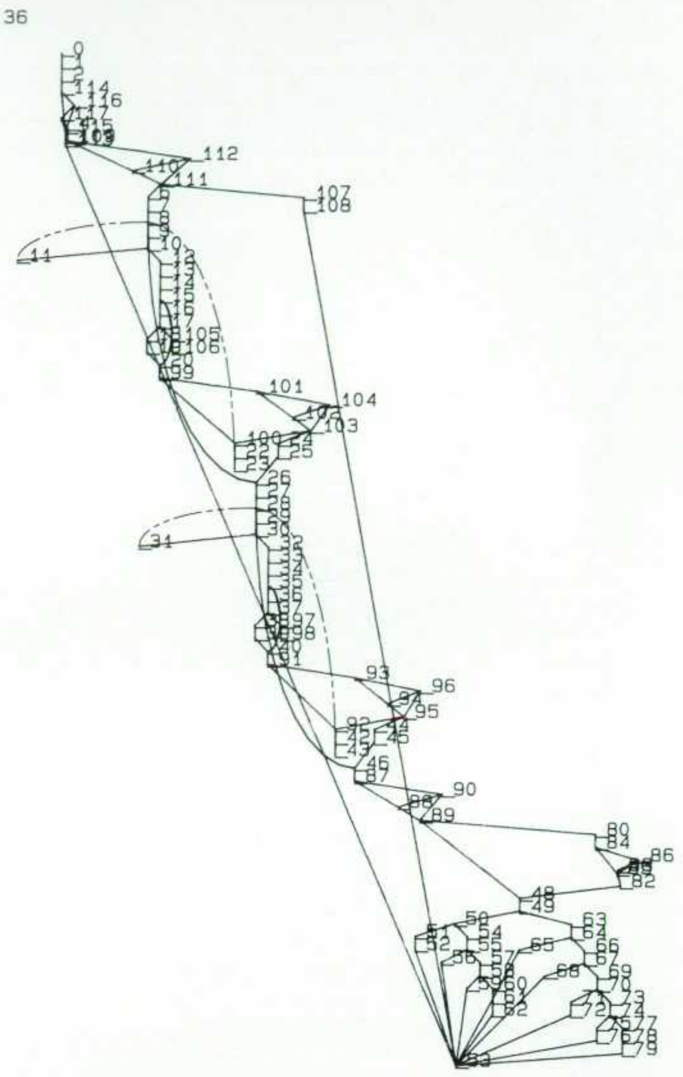


Fig. 5. *The program flow graph for a module with a cyclomatic complexity of 36. The high complexity value and the visual presentation indicates that this module is error-prone and very likely hard to maintain.*

Code Inspections. Recent studies have suggested that one of the most effective techniques for software defect prevention and detection is the use of formal inspections.[7,8] The complexity data and the flow graphs can be used to help evaluate various code paths during an inspection, and to help determine which modules should be given an inspection.

The Software Quality Engineering Process. The software quality engineering (SQE) group at Waltham has been actively promoting the use of McCabe's technology within the lab. Specifically, the SQE group is working with current projects so that all code is subjected to calculation of the cyclomatic complexity of each module. This process has been established as part of our software product release criteria. In addition, the SQE group has purchased and maintains a tool* that computes complexity and generates program flow graphs. As each project completes major blocks of code, the SQE group generates the flow graphs for that code and then provides feedback to project management and team members.

Conclusion

The McCabe methodology and toolset have been integrated into the Waltham software development process over the past year. This process has been accomplished with no disruption to current lab projects and has resulted in the following successes:

- Automatic identification of potentially faulty software before actual testing is started
- Automatic identification of code modules that could benefit from code inspections
- Automatic generation of test case data for all software modules
- Well-defined coding standards accepted throughout the lab
- Effective code defect prevention strategies based on restructuring of overly complex code.

Each of these successes has contributed to the overall success of the software defect prevention program presently underway at the Waltham lab. By identifying and correcting software code defects very early in the coding phase of product development, the McCabe methodology and toolset continue to have a major impact on our efforts to improve the productivity of the Waltham development process and the quality of the resultant products.

*ACT™ (Analysis of Complexity Tool), a product of McCabe and Associates.

References

1. S.D. Conte, H.E. Dunsmore, and V.Y. Shen, *Software Engineering Metrics and Models*, Benjamin Cummings Publishing Company, 1986.
2. R.S. Pressman, *Software Engineering: A Practioner's Approach*, McGraw-Hill, 1982.
3. C.G. Schulmeyer and J.I. McManus, *Handbook of Software Quality Assurance*, Van Nostrand Reinhold Company Inc., 1987.
4. T.J. McCabe, *Structured Testing: A Software Testing Methodology Using The Cyclomatic Complexity Metric*, National Bureau of Standards Special Publication 500-99.
5. T.J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, Vol. SE-2, no. 4, Dec. 1976, pp. 308-320.
6. T.J. McCabe, and Associates, Inc. *Structured Testing Workbook*, 14th Edition.
7. M.E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM System Journal*, no. 3, 1976, pp 182-211.
8. M.E. Fagan, "Advances in Software Inspections," *IEEE Transactions on Software Engineering*, Vol. SE-12, no. 1, July 1986, pp. 144-151.

Object-Oriented Unit Testing

HP's Waltham Division has taken a first step in applying new and traditional unit testing concepts to a software product implemented in an object-oriented language.

by Steven P. Fiedler

ALTHOUGH OBJECT-ORIENTED ENVIRONMENTS are being used more frequently in software development, little has been published that addresses object-oriented testing. This article describes the processes and experiences of doing unit testing on modules developed with an object-oriented language. The language is C++[1] and the modules are for a clinical information system. Because the system must acquire real-time data from other devices over a bedside local area network and the user requires instant information access, extensions were made to the language to include exception handling and process concurrency. We call this enhanced version Extended C++. Test routines were developed and executed in an environment similar to that used in development of the product. This consists of an HP 9000 Series 300 HP-UX 6.01 system.

Unit Testing

Unit testing is the first formal test activity performed in the software life cycle and it occurs during the implementation phase after each software unit is finished. A software unit can be one module, a group of modules, or a subsystem, and depending on the architecture of the system, it is generally part of a larger system. Unit tests are typically designed to test software units, and they form the foundation upon which the system tests are built. Since software units and unit tests are fundamental entities, unit testing is critical to ensuring the final quality of the completed system.

The unit testing process involves test design, construction, and execution. The test design activity results in a test plan. Because the primary intent of unit testing is to find discrepancies between unit specifications and the coded implementation,[2] the unit specification is the primary reference for the test plan. Test construction involves building the test cases based on the test plan, and test execution involves performing the tests and evaluating the results.

Both structural (white box) testing and functional (black box) testing techniques are used in unit testing. Since structural testing requires intimate knowledge of the design and construction of the software, unit testing requires intense developer involvement in the process.

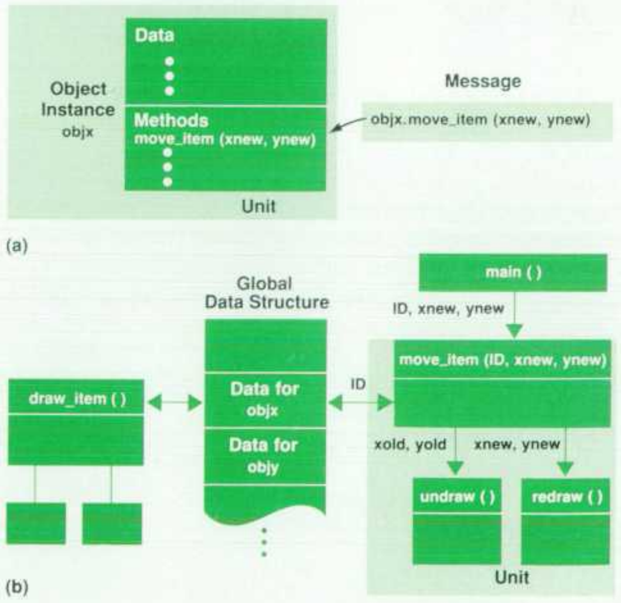


Fig. 1. *(a) Object instance* objx *being sent the message* move_item(xnew,ynew) *to invoke the method to move a graphical item from one location to another. (b) The same operation being processed in a procedural language environment.*

Objects

An object is the fundamental building block in an object-oriented environment and it is used to model some entity in an application. For example, in an office automation system, objects might include mail messages, documents, and spreadsheets. An object is composed of data and methods. The data constitutes the information in the object, and the methods, which are analogous to procedures and functions in non-object-oriented languages, manipulate the data. In most applications, there are many objects of the same kind or class (e.g., many mail messages, devices, etc.). C++ defines the data and methods for these similar objects in a data type called a *class*. Each object in an object-oriented language is an instance of a particular class. Also in C++, a data item is referred to as a *member* and the methods, *member functions*.

One of the main differences between object-oriented and procedural languages (non-object-oriented languages) is in the handling of data. In a procedural language environment such as Pascal, C, or Fortran, system design is based on the data structures in the system, and operations are performed on data passed to procedures and functions. The primary data items are typically global and accessible to all the modules in the system. In an object-oriented environment, the object's internal data structures and current values are accessible only to the methods within the object. The methods within an object are activated through messages passed from other objects. The messages indicate the method to be activated and any parameters required.

Fig. 1 illustrates these diferences in architecture between object-oriented systems and procedural-language-based systems. In Fig. 1a, to move a graphical item (objx) from one location to another, the message move_item(xnew,ynew) is sent to the object instance objx to perform the operation. The current location and geometric characteristics of the item are contained in the data structures of objx. The methods in objx will handle the transformation and translation of the item to a new location. Fig. 1b depicts the same operation in a procedural language environment. The graphical items's data structure and current values are kept in a global data structure which is accessible to all the modules in the system.

Objects and Unit Testing

The issues related to objects and unit testing include:

- When should testing begin? In a procedural language environment, a complete unit may not exist until several functions or procedures are implemented. In an object-oriented environment, once a class has been defined and coded, it can be considered a complete unit and ready for use by other modules in the system. This means that

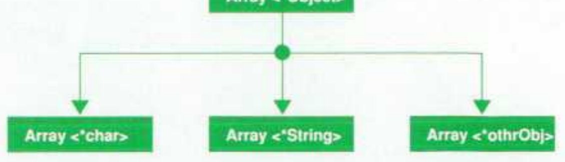


Fig. 2. *An example of parameterization in extended C++.*

```
Class: String                                              Rev: 1.9 87/08/12

Derived Class:  Sequence <char>

Include File :  #include "include/generic/String.h"

Role    : String may be used whenever a general-purpose sequence of characters is
          required. It can be used to store and manipulate character strings of any size.


Abstract Invariants: (statements about the properties of a class that help to explain its
behavior).
   1. The characters in String are terminated by a null character. This null is not accessible
      to the user.
   2. Length: (int, 0 <= Length) number of characters in String not including the null
      terminator.
   3. FirstIndex: (int) index of the first character in the character portion of String.
   4. LastIndex: (int) index of the last character in the character portion of String.

Public Functions : String  ~String  Append  DeleteString
                   Lowercase Print  []  +  =  ==  !=

Inherited Functions: AddFirst  Capacity  Empty  Flush  Size  Store

Public Function Specifications:

        String                  ()
              Returns           (String)
              Constructor (s = String())

              String is constructed with Length = 0 and character string
              portion set to null.

        ~String                 ()
              Returns           ()
              Destructor

              If the character portion of String is not null, DeleteString.is
              called to free the allocated heap area.

        Append                  (const String s1)
              Returns           void

              Appends s1 to characters in String this.Length = this.Length +
               s1.Length

        DeleteString            (const int StartIndex,
                                 const int Nchars)
              Returns           void
              Signals           (InvalidNumberOfChars,
                                 InvalidStartIndex)

              Remove Nchars characters from String starting at
              StartIndex. If Nchars > this.Length − StartIndex then all the characters from
              StartIndex to the end of String are deleted and this.Length = StartIndex,
              otherwise, this.Length = this.Length − Nchars.

              If this.FirstIndex > StartIndex or StartIndex > this.LastIndex, then
              InvalidStartIndex is raised. If Nchars < 0, then InvalidNumberOfChars is
              raised.

        Lowercase                ()
              Returns           void

              Converts all characters in String to lowercase.

        Print                    ()
              Returns           void

              For debugging purposes, prints the internal representation of String.


        Operator []             (const int index)
              Returns           char
              Signals           (InvalidIndex,
                                EmptyString)

              Returns the character at this.[index] of String. If index
              > this.LastIndex or index < this.FirstIndex, then InvalidIndex is
              raised. If String is empty, then EmptyString is raised.

        Operator +              (const String s1)
              Returns           String

              This function has the same behavior as String::Append

        Operator =              (const String s1)
              Returns           void

              Assigns the value of s1 to this String.

        Operator ==             (const String s1)
              Returns           Boolean

              Returns TRUE if this.Size = s1.Size and each character in String is matched
              by the corresponding character in s1, otherwise it returns FALSE.

        Operator !=             (const String s1)

              Returns           Boolean

              Returns TRUE if this.Size = s1.Size and at least one character in String is not
              matched by the corresponding character in s1, otherwise it returns FALSE.
```

Fig. 3. *Class specification for the class* String.

unit testing must be considered much earlier in an object-oriented environment.

- What testing techniques should be used? Since the paradigm of object-oriented programming emphasizes the external behavior of data abstractions rather than the internals, one would expect to employ only black box, functional testing techniques. However, a more robust testing structure employing complete path testing is actually needed.
- What should be tested? In an ideal situation, the answer to this question would be that all classes should be completely path tested, particularly for critical application systems. However, the resources required to meet this goal may be substantial and, in working towards it, trade-offs are likely to be made. Nonetheless, refinements can be added to the testing process that simplify labor intensive phases and improve chances that a minimal set of tests will be executed.
- Who should do unit testing? To answer this question, we need to consider what is being tested and the expertise required of the tester. Remember that units are typically modules that eventually become part of a larger system and only the developers know the detailed internals of the units they are responsible for building. As a result, an independent tester or a developer who is not involved in the design and generation of code for a specific class may find it difficult to perform adequate testing on that class. For example, a developer may design a data base class which is intended to make it easier for a user to perform transactions in a data base. The methods within the data base class are responsible for performing the data base interface tasks. An independent tester who is unfamiliar with the way in which these low-level functions work would certainly be ineffective in testing the internals of this class.

In the clinical information system, knowledge of Extended C++ was sufficient to become an effective tester for certain classes in the system. This was because of the formulation of generic classes. A generic class in the clinical information system is a class that provides general functionality. It can be considered an extension of the language's built-in data types that fills a utilitarian purpose for other components of the system. Strings and linked lists are examples of objects that provide such universal functionality.

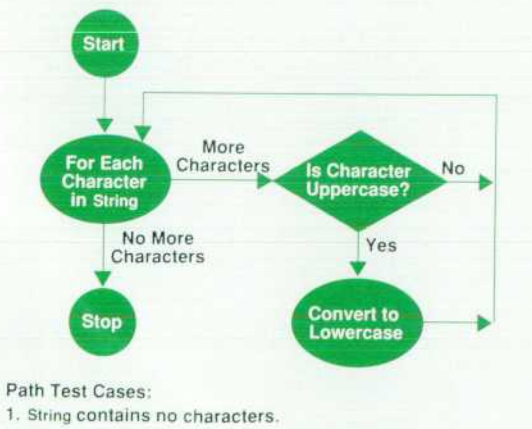
To build on this generic concept, parameterized type classes were introduced.[3] Parameterization permits a general definition of a class to be extended to create a family of type-safe classes, all with the same abstract behavior. For example, suppose we design a class called Array which contains pointers to some object. Through parameterization, we can extend this class definition to create arrays that point to characters, arrays that point to strings, or arrays that point to any other type of object (Fig. 2). The testing of a parameterized type class can provide a high level of reliability for a growing family of similar classes. From the experience gained in testing generic classes, we have developed an approach to the testing of other C++ classes.

Test Process

The tasks associated with the testing process for objects are the same as for regular unit testing: design, construction, and test execution.

Design. During the design phase, the tester determines the test approach, what needs and does not need to be tested, the test cases, and the required test resources. The inputs required to conduct the design phase for objects include:

- The header and source files of the target class (the class being tested), and a well-defined specification of the class.[4] An example of a class specification is shown in Fig. 3.
- An analysis of the effects of inheritance on the target class. When a class uses another class as a base to build additional functionality, it is said to be derived from that class and consequently inherits data and methods from the base (parent) class. If the target class is derived, we want to know if the base class has been thoroughly tested. Provided that the functionality of the base class has been proven, any member function of the target test class that leverages directly from a base class member function will require minimal testing. For example, the specification in Fig. 3 shows that String is derived from a parameterized class called Sequence. The functions that String inherits from Sequence (AddFirst, Capacity, etc.) require



Path Test Cases:
1. String contains no characters.
2. String contains only lowercase characters.
3. String contains both uppercase and lowercase characters.

Fig. 4. *Path test cases for the function* Lowercase.

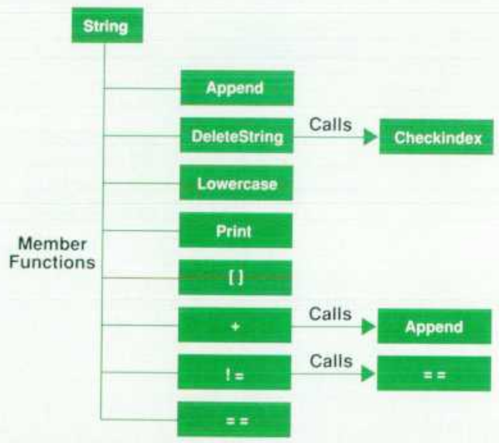


Fig. 5. *Dependencies for class* String.

only a basic functionality test.

- The cyclomatic complexity metric[5] of the individual member functions belonging to the target class. The complexity measure and its accompanying testing methodology play a key role in the implementation of this test strategy. Through their use, we can ensure that all the independent paths in the target member functions are tested. Fig. 4 shows an example of path test cases for the member function Lowercase. In our project, the predicate method[6] of calculating cyclomatic complexity has been built into the Extended C++ parser.
- A hierarchy or structure list which shows member function dependencies. In simple terms, what member functions call what other member functions of this class? Private member functions, which are not accessible to the end user directly, should also be included in this list. For example, in Fig. 5, the function operator + performs its task by invoking the Append function, which indicates that Append should be tested first.
- The signals or exceptions that are raised (not propagated) by each function. Extended C++ includes linguistic support of exception handling,[7] which permits a special kind of transfer of control for processing unusual but not necessarily erroneous conditions. These signals should not be confused with HP-UX operating system signals. Signals are defined for the various member functions in a class specification. For example, the specification for String indicates that the member function DeleteString raises a signal called InvalidStartIndex if the StartIndex parameter passed to the member function is not valid.

The last step of the design phase is to determine the approach to use to verify the test results. There are a number of options in this area. One approach is to print out the expected results for a test case and the actual results generated by the target class test to two different files. At the end of the test, the two files can be compared using the standard UNIX® tool diff (see Fig. 6). A second option for results verification uses similar ideas, but may require less actual programming time. An expected results file can be constructed by hand and the file can be used for comparison with actual target class output. If these two approaches prove impractical because of the behavior of the class being tested, a third alternative might be to include the expected observations in a test plan using the class specification as a basis for deriving these observations.

UNIX is a registered trademark of AT&T in the U.S.A. and other countries.

Fig. 7 shows an excerpt from the test plan for the class String. A test plan is the culmination of the test design process, and in addition to guiding test activities, it is an excellent respository of information regarding what was done to test an object.

Construction and Execution. The strategy for developing test cases for execution is to determine all the paths in a module that require test coverage, and then to create test cases based on the class specification (black box approach) and certain features in the code (white box approach). The white box strategy is based on the structured testing methodology resulting from McCabe's work (see article on page 64 for a discussion of the use of the McCabe complexity metric in our division). In this methodology, test cases are created to execute each decision path in the code. In the clinical information system, except for paths that contained code for exception handling, test cases were written to ensure complete path coverage of each member function. Exception handling situations were dealt with separately because they disrupt the normal control flow of a program. Based on the class specification and source code, test cases designed to ensure path coverage were derived using the other well-understood methodologies of equivalence partitioning and boundary-value analysis.[2]

In creating test cases for valid equivalence classes, realistic input values for the member functions were preferred over those that lacked relevance from an application standpoint. For example, if the primary use for our sample String class is to hold a single line of information on an electronic index card, we might expect it to hold, on average, 12 to 50 characters. Our test case would be to create

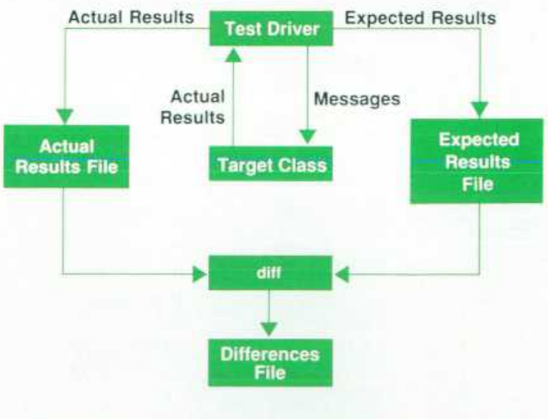


Fig. 6. *Data flow diagram for creating verification files and performing the result comparisons.*

Test Plan for Class: String

Date & Revision: 87/08/12 Rev. 1.9

Source: #include "include/generic/String.h"

Link/Load: EC $1.o -lGC -lorte -leorte -o $1

Test Items: String is the only item under test. String is derived from Sequence<char>.

Features to be Tested: All of the functions and operators of the class are tested.

Features not to be Tested: No functions of this class will go untested.

Approach to Testing: After one or more String member functions are called, the String::Print member function is used to verify the success of the functions. String::Print uses cout<<form. Another data structure that emulates the operation being performed on the String is also constructed. It, too, is output and the results of the String and emulator are compared. At times, String may also act as an emulator if the needed fuctions have been tested.

Pass/Fail Verification: The results of each String test are compared to the results from the emulator, using the HP-UX diff comand. No differences should be detected.

Test Environment: No workstation interaction is required for these tests. Tests run on HP-UX 6.01, HP 9000 Model 350.

Test Description: The following tests verify the operation of String.

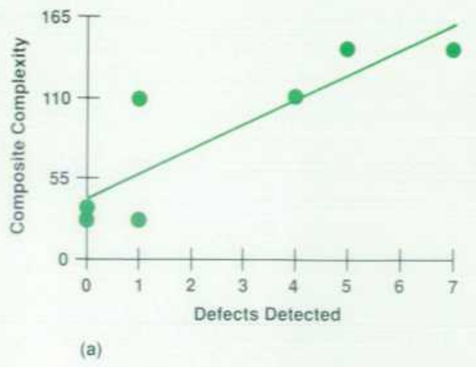
StrTest0.c. This module tests the following:

- Append with char* parameter
- Print with String empty and with String NOT empty

StrTest1.c. This module tests all of the String constructors using the heap and stack-based variables.

.
.
.
.
.
.

Fig. 7. *Portion of the test plan for the class* String.



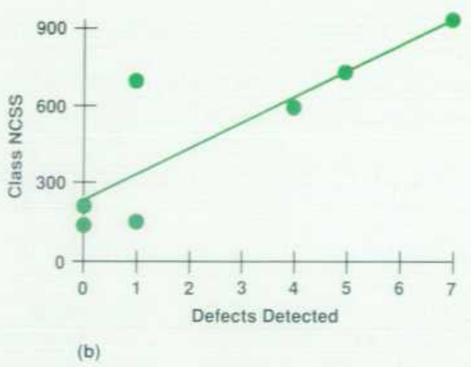


Fig. 8. *Defects detected in the classes tested after developers had completed black box testing. (a) Composite complexity versus defects for generic classes tested. (b) NCSS versus defects for the same classes.*

a string of 40 characters rather than 140.

Boundary-value analysis dictates that tests be built that create objects of extremes. Instances of null strings (size = 0) should respond as any non-null string would unless the specification states otherwise. Clearly, a null string appended with the value abc should yield the same result as the string abc appended with a null value. At the other extreme, tests should exist to stress objects (usually in size) beyond all expectations of normal use. For example, in HP-UX, main memory is managed in pages of 4096 bytes. Therefore, it should be valid to create a string that holds 4097 characters.

Tests to invoke exception handling capabilities were also included in class test suites. Boundary-value conditions were used to invoke these facilities. For example, if an exception is encountered when we index beyond the legal boundary of a string, the test case invokes the exception by trying to access the character just past the end of the string, not ninety-nine characters past it. Special care must be taken in coding exception test cases, because if a signal raised by a member function is not handled correctly, an aborted test program may result.

There are other areas for test cases that do not show up using the structured technique. For example, the effects of implicitly and explicitly invoking a class's constructor and destructor[†] functions should be examined for consistency. Initialization and casting operations should also be tested. In addition, defects have been discovered by applying associativity rules to member functions. That is, if string s1 is null, and string s2 is not null, s1 > s2 should yield the same results as s2 < s1. In addition, the use of the object itself as a member function input parameter proved valuable in uncovering subtle implementation errors. For instance, given s1 is a string, the test s1.Append(s1) becomes a legitimate and creative way of triggering certain test conditions. Much of this type of testing can be integrated into standard testing without creation of separate tests.

Results

The methodology presented here was applied to testing several generic classes after the development group had completed their testing using black box testing techniques. The results show the shortcomings of strict black box testing. Even though development group testing was extensive and appeared to be thorough, defects were still uncovered.

[†]In C++ a constructor and a destructor perform the initialization and termination for class objects, respectively.

Defects were found in each of the generic classes tested. The number of defects found seemed to be related to the composite (total) complexity of all of the class member functions and more directly to the number of noncomment source statements (NCSS) contained in the source and include files. The general relationship of complexity to defects is shown in Fig. 8a, and the correlation between defects and the NCSS of each class is shown in Fig. 8b. Each point represents a generic class. On average, a defect was uncovered for every 150 lines of code, and correspondingly, the mean defect density exceeded 5.1 per 1000 lines. Only the code contained in the source and include files for each class was counted for this metric. Code from inherited functions was not considered. These defect rates pertain to a small set of actual product code produced during the early stages of development. Another interesting relationship was observed when the NCSS values of source and test code were compared (see Fig. 9).

Conclusion

There is a cost associated with class testing. A significant investment of time is required to perform the testing proposed here. Assuming testers are already competent with the object-oriented environment, they must acquire familiarity with McCabe's complexity concepts as well as a basic understanding of the class being tested. Because testing so far has taken place concurrently with development, time estimates for the testing phase have been somewhat incon-

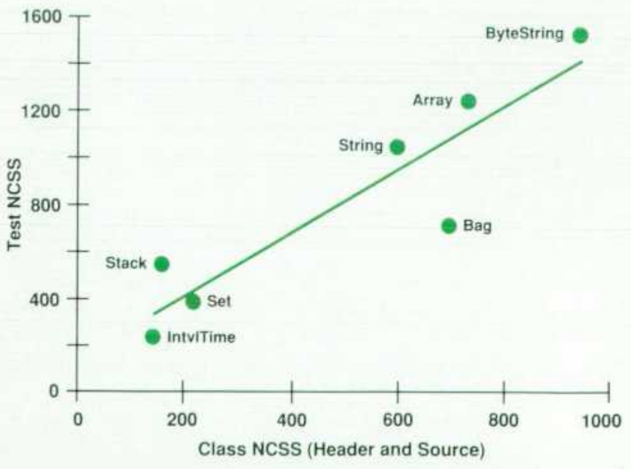


Fig. 9. *Test code NCSS versus class source code NCSS.*

Class Name	Composite Complexity	Test Development Hours	Number of Member Functions			Defects	NCSS	
			Public	Private	Inline		Source	Test
String	112	40	36	2	11	4	598	1047
ByteString	114	36.25	37	2	13	5	733	1241
Array<'T>	144	45	42	13	11	7	945	1526
Stack	29	9	23	0	17	1	158	555
Bag	110	29	28	14	13	1	696	712
Set	36	6	28	1	22	0	217	391
IntvlTime	27	5	17	0	9	0	142	247

Fig. 10. *Metrics for testing generic classes.*

sistent and do not yet suggest any clear conclusions. Fig. 10 summarizes the metrics we have collected thus far. (The classes are listed in the order they were tested).

In the object-oriented environment, objects and their definitions, rather than procedures, occupy the lowest level of program specification. Therefore, it is necessary to focus on them when implementing a thorough test methodology. Practices used in testing traditional procedural systems can be integrated in the approach to object-oriented testing. The main difference we have found so far is that each object must be treated as a unit, which means that unit testing in an object-oriented environment must begin earlier in the life cycle. Through continued collection of the class metrics and test results, we hope to gain more insight and continue to improve our object-oriented unit test efforts.

References

1. B. Stroustrup, *The C++ Programming Language*, Addison-Wesley, 1986.
2. G.J. Myers, *The Art of Software Testing*, John Wiley & Sons, 1979.
3. B. Stroustrup, "Parameterized Types for C++," *USENIX C++ Conference Proceedings*, 1988.
4. R. Seliger, E. Calm, and L. Smith, "Object-Oriented Design Methodology," *HP Software Engineering Productivity Conference*, 1987.
5. T.J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, Vol. SE-2, no. 4, 1976.
6. T.J. McCabe and Associates, Inc., *Structured Testing Workbook*, 14th Edition.
7. E. Horowitz, *Fundamentals of Programming Languages*, Second Edition, Computer Science Press, 1984.

Validation and Further Application of Software Reliability Growth Models

After two years of use, a software reliability growth model has been validated with empirical data, and now it is being expanded to estimate test duration before it begins.

by Gregory A. Kruger

AT HP'S LAKE STEVENS INSTRUMENT DIVISION, a software reliability growth model* has demonstrated its applicability to projects ranging in size from 6 KNCSS to 150 KNCSS (thousand lines of noncomment source statements), and in function from instrument firmware to application software. Reliability modeling curves have been used to estimate the duration of system integration testing, to contribute to the release-to-sales decision, and to estimate field reliability. Leveraging from the basic model, project managers are beginning to plan staffing adjustments as the QA effort** moves through the defect-fixing-limited phase and into the defect-finding-limited phase.

Basic Model

In the fall of 1986, a software reliability growth model's good fit to historical data on a previous firmware product led to the development of a set of release criteria, with defects per system test hour (QA hour) as the principal quality measure.[1] The model and release criteria were then applied in real time to a new application product. The modeling effort aided in predicting when the product was ready for release to customer shipments and provided estimates for the number of defects that might be found in the field.

*Software reliability growth modeling is based on the premise that as software is tested and defects removed, the reliability gets better (grows).

**QA effort or QA phase in this paper refers to the system integration test phase of the software life cycle.

The basic exponential model is based upon the theory that the software defect detection and removal effort will follow a nonhomogeneous Poisson process.[2] In this process the defect arrival rate is assumed to decrease with every hour of testing (or at least with every code correction). The model has two components.

The cumulative number of defects found by time t is given by

$$m(t) = a(1-e)^{-(k/a)t},$$

and the instantaneous new defect-finding rate at time t is given by

$$l(t) = ke^{-(k/a)t}.$$

Fitting the model requires the estimation of parameters k, the initial defect discovery rate, and a, the total number of defects. The data required is obtained by recording on a daily or weekly basis the time spent executing the software and the resulting number of defects discovered. The model parameters may be estimated by the least squares, nonlinear least squares, or maximum likelihood method. In most cases, the maximum likelihood method is preferred.

Considering typical software development and system testing practices, the assumptions necessary for the applicability of Poisson theory would seem to negate the use of the model. Key assumptions of the model and the

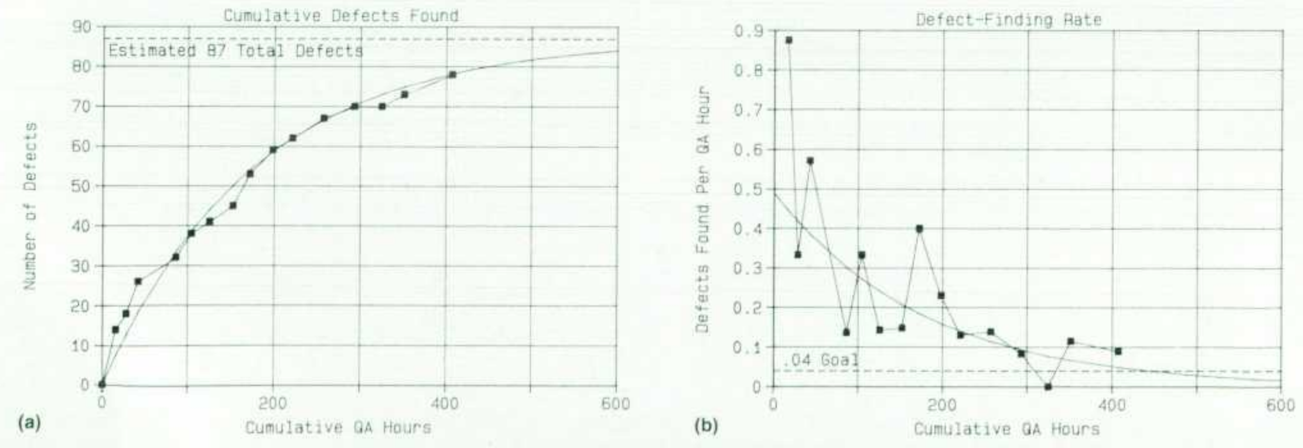


Fig. 1. *Project C results. (a) Cumulative defects found m(t). (b) Defect-finding rate l(t).*

correspondingly realities are:

- Assumption: All functionality is completed before the start of system testing.
  Reality: Many products enter system testing without all the features in place.
- Assumption: Testing can be considered to be repeated random samples from the entire input domain.
  Reality: There is some random testing, but typically testers are more structured and systematic in the selection of test cases.
- Assumption: Defects found are removed with certainty and no new defects are introduced (a perfect repair).
  Reality: A defect repair may introduce new defects.
- Assumption: The times between failures are independent.
  Reality: When a defect is found in a particular area of the software, because of the suspicion that there may be more defects in the same area, the area is probed for more defects. This process usually finds more defects, which is good, but makes the arrival rate of defects dependent on when the last one was found.

As has been said, with such a set of assumptions, it would seem unlikely that this model would fit real-world data. However, some aspects of the testing process at Lake Stevens approximate these conditions. First, our life cycle calls for all functionality to be completed by the time we start formal system integration testing. Typical projects have 95% or more of their functionality complete by this time. Second, the entire set of functionality is subdivided and assigned to different individuals of the testing team. Therefore, while the testing process cannot be considered to be repeated random samples from the input domain, it is at least sampling from the entire functionality set as time progresses. This is in contrast to a testing process wherein some subset of the functionality is vigorously tested to the exclusion of all others before moving on to another subset and so on. Regarding the third assumption, strict revision control procedures at least maintain some control over the rate of defect introduction. Finally, nothing about the Lake Stevens development process justifies the assumption that the times between failures are independent. After finding a serious defect in a portion of the product, testing effort often intensifies in that area, thus shortening the next time to failure.

The model's success in describing the projects at LSID demonstrates some degree of robustness to these assumptions. Our past and continued application of software reliability theory is not based on a fundamental belief in the validity of the assumptions, but in the empirical validation of the model. Therefore, we have continued to use software reliability growth models with the following objectives in mind:

- To standardize the application of the model to all software products produced at LSID
- To put in place a set of tools to capture and manage the data and obtain the best fit curves
- To use the defect-finding rate and the estimated defect density to define the release goal
- To predict the duration of the QA phase before its start
- To understand the relationship between model estimates and field results.

Standardized Application

To date, software reliability growth modeling has been conducted on eleven projects that have since been released for customer shipment. Two demonstrated excellent fit to the model, two very good fit, four showed a fair conformance to the model, and three showed a poor fit. Fig. 1 shows the curves for one of the projects on which the model gave an excellent fit. Contrast these results to the model's performance on the project shown in Fig. 2. Note that time in this case is measured in calendar days rather than test hours. Here the cumulative defects begin to taper off only to start up again. These results reflect inconsistent testing effort, which is not picked up by simply measuring calendar days of testing effort. The curves in Fig. 2 were obtained by independently fitting the basic model before and after the change in testing effort. These two best-fit models were then tied together to form the piecewise curves shown.

Tools

The defect tracking system (DTS),[3] an internal defect tracking tool, is used by all project teams to log defects found during system testing. In software reliability modeling it is important to record all time spent exercising the software under test regardless of whether a defect is discovered.

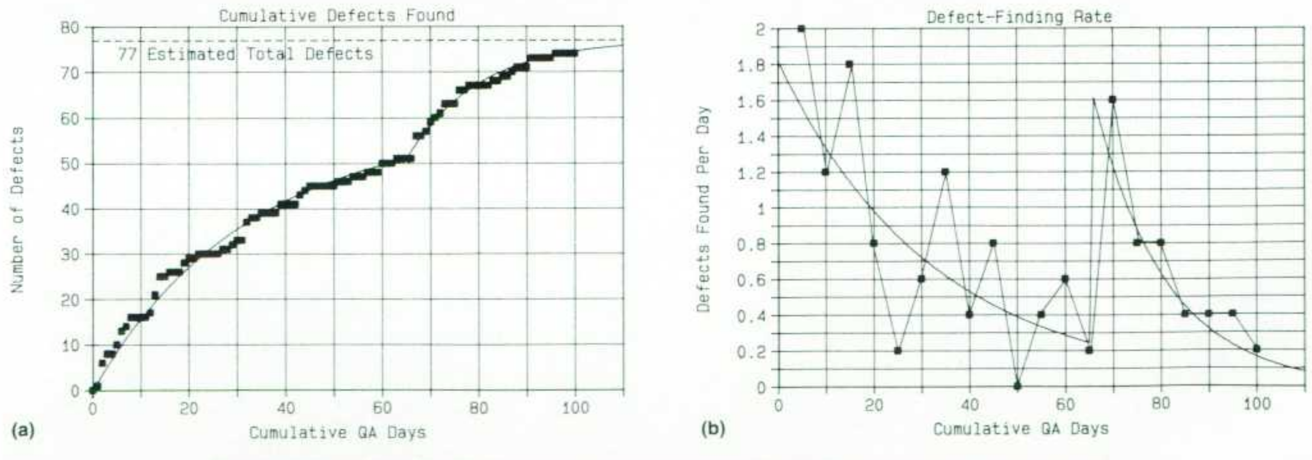


Fig. 2. *Project D results. (a) Piecewise curve fit for cumulative defects found m(t). (b) Piecewise curve fit for defect-finding rate l(t).*

DTS has proven to be unsatisfactory for capturing QA hours that do not produce a software defect. Therefore, project teams separately log test hours at the end of each day.

The DTS data is loaded into an Informix data base so that it can be sorted and retrieved as desired. On projects using DTS for tracking QA time as well as defect statistics, Informix reports generate files with weekly (or daily) QA hour and defect total data pairs. On projects tracking QA time separately, the weekly (or daily) defect totals are retrieved from the Informix data base and matched with the appropriate QA hours. In either case, the file of cumulative QA hours and cumulative defects found is submitted to a program that obtains the best-fit model parameters by the method of maximum likelihood. At the present time, plots for distribution are generated using Lotus®1-2-3®. Future plans call for using S, a statistical package that runs in the HP-UX environment, to generate the graphics, thereby conducting the data manipulation, analysis, and plotting all on one system.

Release Goal

The software modeling process provides two related metrics that help support a release-to-customer-shipments decision: the defect-finding rate and the estimated number of unfound defects. A specific goal for one of these two metrics must be established if the model is to be used for predicting the conclusion of system testing.

The defect-finding rate is a statistic you can touch and feel. It can be validated empirically—for example, 100 hours of test revealed four defects. On the other hand, one can never really measure the number of defects remaining. This metric can only be estimated. Although the two measures are related, it is not true that two projects releasing at the same defect-finding rate goal will have the same number of defects estimated to be remaining. Couple this fact with the recognition that the size of the product has no bearing on the model fit and the resulting estimated number of residual defects and it is clear that two projects releasing at the same find rate could have quite different estimated residual defect densities. Because of its observability, the defect-finding rate has been used as the principal release goal on all projects to date except one. However, both the failure rate and the estimated residual defect density are monitored and used in aiding the release decision.

Lotus and 1-2-3 are U.S. registered trademarks of Lotus Development Corporation.

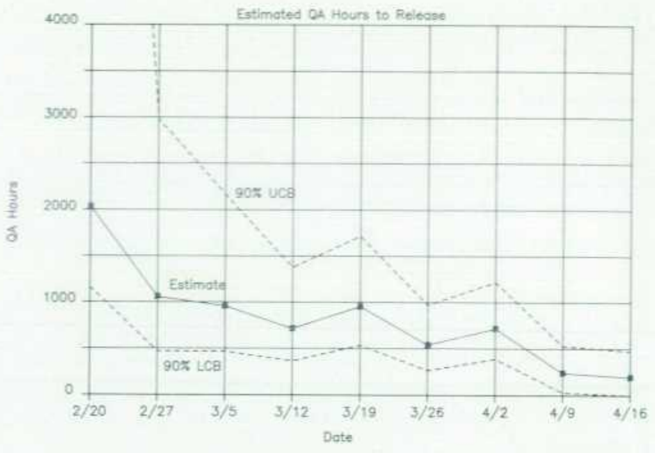


Fig. 3. *QA hour estimates on project E.*

The Project E Experience

The one project to date using a goal of ending system test with a certain residual defect density will serve as a good illustration of the contributions and limitations of software reliability growth models. Project E is an application software product of 156 KNCSS. This project represents a new release of a previously developed product and is roughly two-thirds reused or leveraged code. The stated goal at the start of system integration testing was to achieve an estimated residual defect density of 0.37 defects per KNCSS, a goal derived from the performance of the first release of this product. Such a goal means that the best-fit model should be estimating 58 residual defects.

A team of engineers was assembled to conduct testing while the project team fixed defects. The data was plotted at roughly 30-hour testing intervals and the model refit each week. The most recent curve was used to estimate the QA hours required to achieve the objective and these estimates were plotted weekly with statistical confidence limits as shown in Fig. 3. In mid-April, the decision was

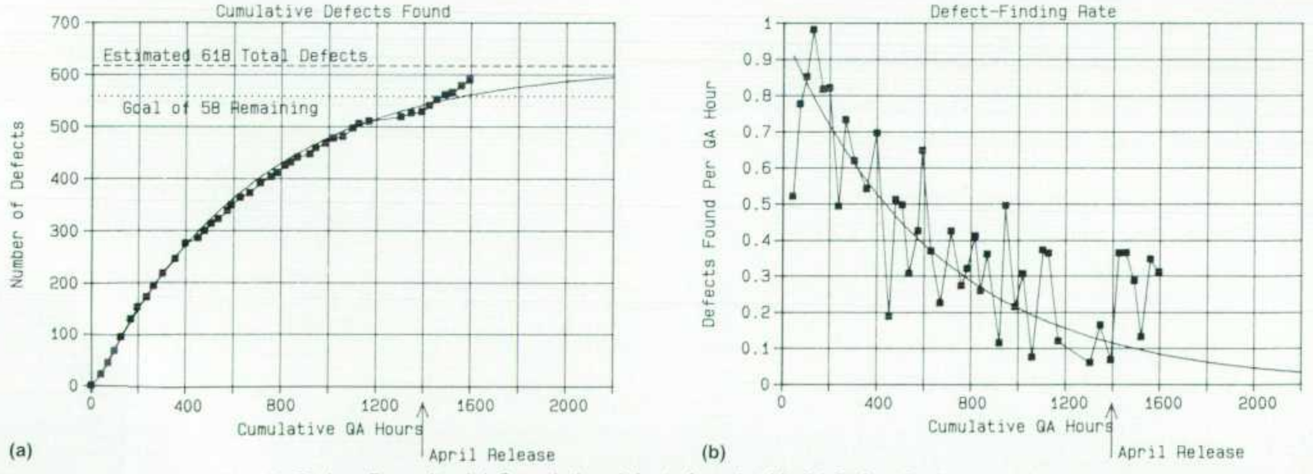


Fig. 4. *Project E results. (a) Cumulative defects found m(t). (b) Defect-finding rate l(t).*

made to release the project for customer shipments and to continue futher testing and refinements for a final release in June. The team had all but reached the goal and the data had tracked the model very well. At this point, the engineers on the testing team disbanded and returned to their original project assignments. The design team then took on the task of conducting both continued testing and defect resolution. With only the designers looking, the defect discovery rate jumped up rather than continuing to follow the curve as can be seen in Fig. 4. The designers were testing specfic areas of the code (directed testing), so an hour of testing now was not equivalent in intensity to an hour of testing with the previous test team. The testing process was not meeting the assumption that testing can be considered to be repeated random samples from the entire user input domain.

What is clear from this project is that the failure rate data and curves are modeling more than the software product alone. They are modeling the entire process of testing. The estimates of failure rates and residual defect densities are estimates only as good as the testing process itself. The degree to which these statistics match field results will depend upon the degree to which the testing matches the customer's use profile. The identification of the customer's use profile and the incorporation of that information into the testing strategy is a topic for further investigation.

Before QA Begins

Naturally we would like to estimate the duration of the QA phase before it begins. But fitting a model to do estimation must wait for testing to begin and for enough data to be collected before an effective statistical analysis can be conducted. However, it is possible to use results from past projects to estimate the two model parameters a and k.

In preparation for testing a recent software product, Project F, we reviewed the total number of defects discovered during system integration testing on past projects. Defect densities appeared to fall between 12 and 20 defects per KNCSS. Project F had 28.5 KNCSS, so the likely range for the first model parameter, a, was calculated to be 342 to 570 defects. Again looking at past projects, the initial defect discovery rate averaged around one defect per hour, so the other model parameter, k, could be set to one. Given a goal for the failure rate of 0.08 defects per hour, an expected range of 864 to 1440 QA hours was calculated.

Management ultimately needs an estimated date of completion so the expected QA hours required for system testing must be converted to calendar time. To accomplish this we again reviewed the data on past projects and discovered an amazing consistency of four QA hours per day per person doing full-time testing, and an average of 2.3 defects fixed per day per person doing full-time fixing. Given the number of team members capable of fixing, the number capable of finding and those qualified to do both, the required QA hours for testing could now be converted to calendar time. Fig. 5 shows the final QA projections for Project F and the staffing levels used to convert the QA hours into calendar time. Note that the staffing levels given correspond to the midrange assumption of 16 defects per KNCSS.

Recognize that as testing proceeds, testing and fixing resources will have to be shifted. Early in the process, the project is fixing-constrained because a few testers can find enough defects to keep all available fixers busy. Over time this changes, until late in testing, the project is finding-constrained since it takes many resources looking for defects to keep only a few fixers working. Also, the finders cannot be allowed to outstrip the fixers, creating a large backlog of unresolved defects. Such a situation only causes frustration for the finders because of testing roadblocks created by defects already found.

Our experience to date with using the model to estimate the duration of the QA phase before its start demonstrates the difficulty in estimating the two required model parameters without actual system test data. Project F concluded system integration testing with a total QA effort that was 225% of the original effort. Over twice the expected number of defects were found and resolved. Not all of this error in estimation can be blamed on the failure of the model. In hindsight, we completely disregarded the fact that this was not a stand-alone project. Many of the problems encountered were because Project F had to be integrated with another 130-KNCSS product.

These results indicate that adjustments are necessary in the way values for the model parameters are derived. For instance, currently the values for the parameters are aver-

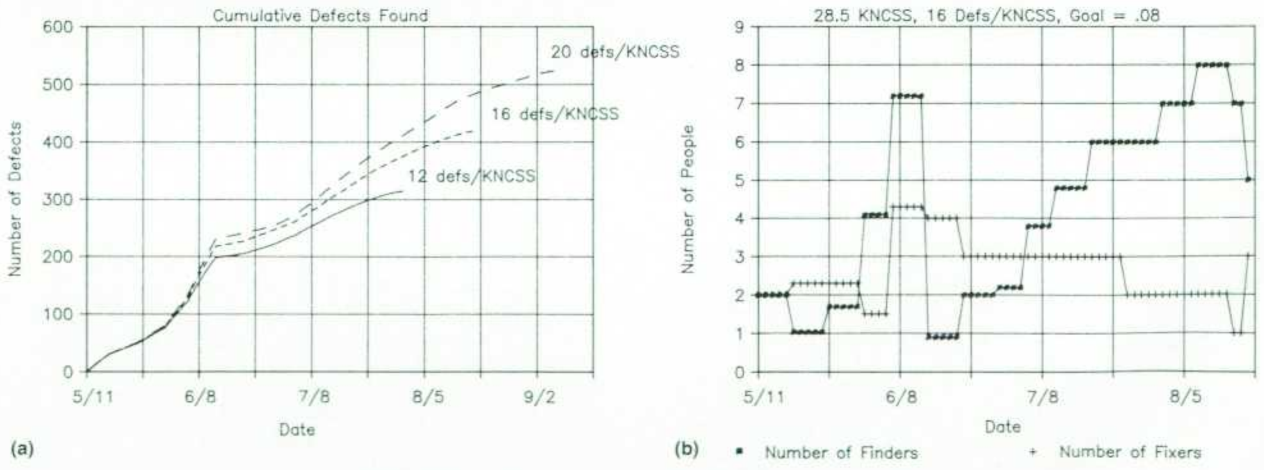


Fig. 5. *QA projections and staffing profile on project F.*

aged from a heterogeneous population of previous software projects. Fixing this problem means that if we want to estimate the QA duration for project X then we must base the model parameters on projects similar to project X. Project characteristics such as complexity, team experience, and the development language must be formally factored into any early estimates of QA duration.

Field Failure Results

Although the modeling process is helping to monitor progress through system testing and is aiding in the release decision, based upon limited defect data from our field defect tracking system it appears that the curves may be overestimating the number of defects customers will discover by as much as forty times.[1] However, it is likely that only a portion of the actual field failures find their way into the tracking system.

Our experience testing one firmware project that was an enhanced version of an old instrument puts an interesting perspective on estimated residual defect densities. This particular product had been shipped for ten years at an average volume of over 200 units per month. Since a market opportunity existed for an updated version of that product, both hardware and firmware enhancements were incorporated into a new version. During system test, an obscure defect in a math routine was discovered that had not only existed in the original product since introduction but in several other products shipped over the last ten years. To the best of our knowledge, no customer or HP personnel had previously found that failure. Its existence was further argument that the information coming back from the field is not giving a true perception of residual defect densities. Not only do customer observed failures go unreported, but it is highly likely that some failures will never be encountered during operation. It was reassuring to note that LSID's current testing process was uncovering defects so obscure as to be unobservable in ten years of field use.

Conclusion

With data collected during system integration testing, we have been able to use a software reliability model to estimate total testing effort and aid in assessing a project's readiness for release to customer shipments. Although the model appears to be somewhat robust to its underlying assumptions, future success will depend upon the integration of customer representative testing techniques into our existing testing process. In addition, there remains the challenge of using the model to estimate test duration before system integration begins. This will require a thorough analysis of data on past projects and key information on the current project to derive better early estimates of the model's parameters. Our ultimate objective remains to achieve validation of the modeling process through accurate field failure data. All of these areas will continue to be investigated because they are important in determining project schedules and estimating product quality.

References

1. G.A. Kruger, "Project Management Using Software Reliability Growth Models," *Hewlett-Packard Journal*, Vol. 39, no. 3, June 1988.
2. J.D. Musa, A Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, 1987.
2. S.R. Blair, "A Defect Tracking System for the UNIX Environment," *Hewlett-Packard Journal*, Vol. 37, no. 3, March 1986.

Comparing Structured and Unstructured Methodologies in Firmware Development

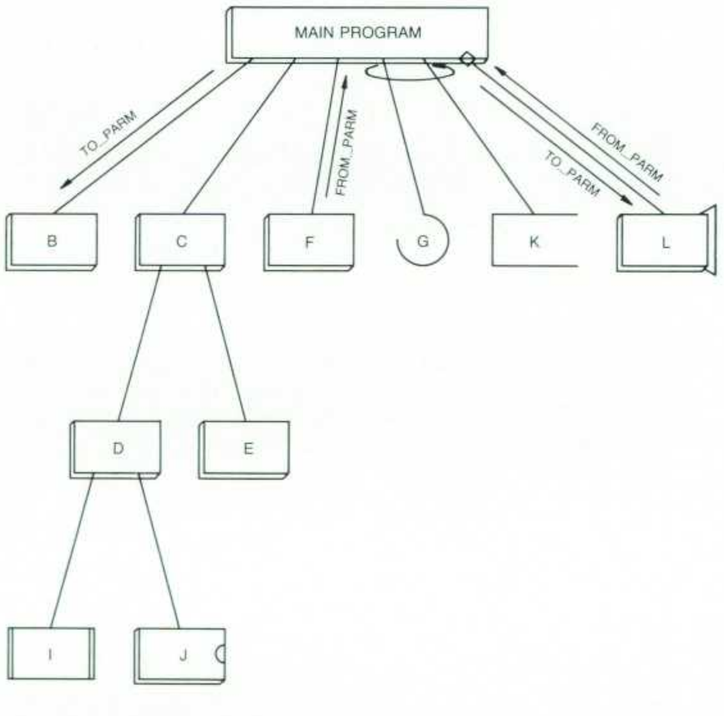
Structured methodologies have been promoted as a solution to software productivity and quality problems. At HP's Logic Systems Division one project used both structured and unstructured techniques, and collected metrics and documented oberservations for comparing the two methodologies.

by William A. Fischer Jr. and James W. Jost

STRUCTURED METHODOLOGY in software development tends to be very much a religious issue with many practitioners. They are either strongly for or against it, but they can point to very little data that supports their point of view. The purpose of this paper is to present some objective and subjective data on the relative merits of structured and unstructured (traditional) methodologies.

The data for this comparison came from the development of a medium-to-large-scale firmware project at HP's Logic Systems Division. It was the first project at this division to use structured methods. The project consisted of an embedded 68000 microprocessor design, coded in C and using a C compiler running on the HP-UX operating system. The firmware consisted of about 47 KNCSS (thousand lines of noncomment source statements) of new code and about 12 KNCSS of reused code.

At the start of the project, a goal was established to use



(a)

```
MODULE MAIN PROGRAM,B,C,D,E,F,H;
SYSTEM G;
EXTERNAL I;
HARDWARE J;
DATA K;
RECURSIVE L;
MAIN PROGRAM
  BEGIN
  B(TO_PARM);
  C
   BEGIN
   D
    BEGIN
    I;
    J;
    END;
   E;
   END;
  F(/FROM_PARM);
  *LOOP
   BEGIN
   G;
   K;
   END;
  *COND L(TO_PARM/FROM_PARM);
  END;
```

(b)

Fig. 1. *(a) An example of a simple hierarchy chart showing the different types of modules. MAIN PROGRAM, B,C,D,E,F, and H (not called) are modules. G is a system module. I is an external module. J is a hardware module. K is a data module. L is a recursive module. Module names can be up to 32 characters long. HCL draws each module name on up to three lines within a symbol. (b) The commands used to create the hierarchy chart.*

structured methodologies[1,2] to improve the software development process and increase product quality, but the decision of whether to use structured methods on each subproject was left up to the individual engineers. As a consequence, three of the subprojects were developed using structured techniques and the other six used traditional methods. Software designers using structured techniques did their analysis using data flow diagrams (DFDs) and structure charts for their design. They also did inspections on most of the analysis and design documents. HP Teamwork/SA, a graphical tool designed for structured analysis that also performs consistency checking, was used for structured analysis. HCL (Hierarchy Chart Language)[3] was used for structured design. HCL is an internal HP tool that plots a program structure from Pascal-like statements (see Fig. 1).

For engineers who used the traditional methods, the analysis phase typically consisted of creating a written specification. Informal design methods were used and coding was started earlier in the product development cycle. Inspections were generally not part of this process.

This was not a scientifically designed experiment to determine the better of the two methods. Rather, we simply collected data on the two groups of engineers as the project developed. As such, our data suffers from many of the common problems that beset unplanned experiments. However, since the data was collected from a single work group, many of the variables that are factors in most comparisons of project experiences have been eliminated. For example, lines of code and time expended were measured and reported in the same way. Intangibles, such as work environment, computer resources, complexity of task, and management attitudes are also identical.

Many experts say the most important variable influencing programmer quality and productivity is individual skill. The difference in the experience level between our two groups was not substantial. However, the unstructured group was more highly regarded by management than the structured group. It is possible that those in the unstructured group had already demonstrated winning techniques for which they had been rewarded, and so they were reluctant to try newer methods, while the structured group was more willing to try new methods to improve their overall skill level.

Data Collection

The data in this report was collected from the following sources:

- Engineering Time. The time spent by each engineer was reported to a central data base on a weekly basis. The time the engineer spent doing such things as system administration, meetings, and classes was not included in the reported time. Only time spent in analysis, design, test, or coding on the engineer's primary software project was included. Time data was reported by the individual engineers.
- Defect Data. The defect data was collected by DTS (defect tracking system),[4] an internal defect tracking tool. Defects were reported from the beginning of system integration testing. The defects discussed in this paper are only unique, reproducible defects. Duplicate, nonreproducible defects, operator errors, and enhancements were not included in the defect count. Defect data was reported by the individual development engineers and the formal and informal testers.
- KNCSS and Complexity. All the KNCSS counts and McCabe's Cyclomatic Complexity metrics were computed by an internal tool called Ccount.
- Design Weight. Design weight,[5] a measure of the effort of coding and testing, was calculated from the C code by an internal tool. This tool counts all the decisions and the unique data tokens that are passed into and out of a function. Any system calls (e.g., printf) are not included in the token count.

Comparison Results

To facilitate the comparisons, the project was broken down into subprojects that closely corresponded to the efforts of individual software designers. Each subproject was then categorized as being representative of either the structured or the traditional methodology. The results of the two methodologies were compared on the basis of productivity and quality. Development effort, manageability, communication, and reusability were the criteria used for productivity measurement. The FURPS (an acronym standing for functionality, usability, reliability, performance, and supportability) model was used as the basis for comparing quality.

We have used metrics to evaluate these factors wherever possible. Where metrics did not exist we have presented the subjective views of the authors who observed the project team throughout the development of the product.

All statistical tests referenced in this paper were made at the 95% confidence level. Since the sample size used for the comparisons between the structured and traditional methods is small, all conclusions are very tentative.

Productivity

Achieving productivity in software development requires using tools and techniques that yield optimal return on development money.

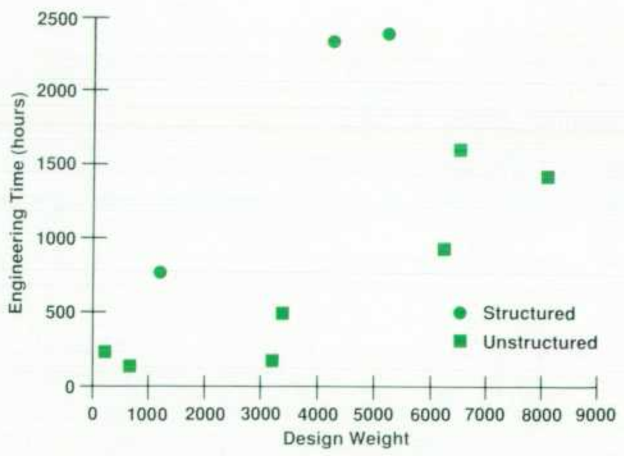


Fig. 2. *Engineering hours versus design weight. Design weight is a measure of the effort of coding and testing. It is calculated by an internal tool.*

Development Effort

Project managers tend to be most concerned with the completion of a project on schedule. Consequently, one of the first questions asked by managers considering the use of structured methods is whether it will shorten the development cycle.

Since a learning curve was involved with the structured methods, we expected that since they were being used for the first time, the structured team would be less productive than the unstructured team. To verify this assumption, two measures of productivity were used, design weight per engineering hour worked and NCSS per engineering hour. These graphs are shown for the various subprojects in Figs. 2 and 3. It can be seen that the three structured subprojects have lower productivity than the unstructured subprojects. A statistical test using general linear hypothesis techniques[6] showed that indeed a statistical difference did exist in the productivity rates between the two methodologies. Using the structured methods, the average productivity rate was 2.40 lines/engineering-hour, while using traditional methods resulted in an average productivity rate of 6.87 lines/engineering-hour.

A central problem with analysis performed with DFDs is that it is an iterative process. The DFDs were leveled until the lowest-level bubbles could be completely described in a minispecification of about one page as recommended by the methodology. This is a rather subjective requirement and we discovered that every designer using DFDs for the first time leveled them more deeply than required. A project review also confirmed that too many intermediate layers were created to keep the complexity per page of DFDs to a minimum. At the project postmortem, it was discussed that a major contributing factor to lower productivity with the structured methods was the lack of an on-site expert. Consultations were needed at various points in the analysis and design phases of the project to verify the proper application of the techniques.

Manageability

The structured work habits seemed to help the project manager and engineers understand the software development life cycle better. Designers had a better idea when to end one phase of the project and begin another, helping to make their own process better understood and easier to manage. Figs. 4 and 5 show the times spent in various project life cycle phases for structured and traditional methodologies, respectively. The structured methods graph shows cleaner, better-defined phase changes. These clear phase changes aid the management planning process by creating a better understanding of the state of a project. Plots showing the same data as Figs. 4 and 5 were done for each engineer, and these individual plots showed the same characteristics.

The regularity of the structured life cycle can be used to improve schedule estimates. Once the percentages of time spent in the phases are historically established, the time taken to reach the end of a phase can be used to project the finish date. For example, if it takes four months to complete the analysis phase and the historical figures indicate that 33 percent of the time is spent in analysis, then it could be estimated that the project would be completed in eight more months.

It is important to measure the progress of projects against the established schedule.[7] Keeping track of the actual completion time of each phase can provide an independent verification of established scheduling methods. If problems in meeting schedule commitments are uncovered, corrective action, such as adding additional resources, can be applied to the project.

Taking a project through the system test phase is unpredictable. The time it takes to complete the formal abuse testing is dependent on the quality built into the product. If the product is well-designed and coded, less time is spent repairing defects found during abuse testing and during the completion of formal regression tests. A reduced testing phase can shorten the overall project development time. More important, if fewer defects are embedded in the product, less time will be spent in the maintenance phase, which can consist of 50% of the project's overall cost. Fig. 6 shows a comparison of the times spent in the various development phases for the project. The graph indicates that a greater percentage of time was spent in the analysis and design phases with the structured methods. However, a very small percentage of time in comparison to the traditional methods was spent in testing.

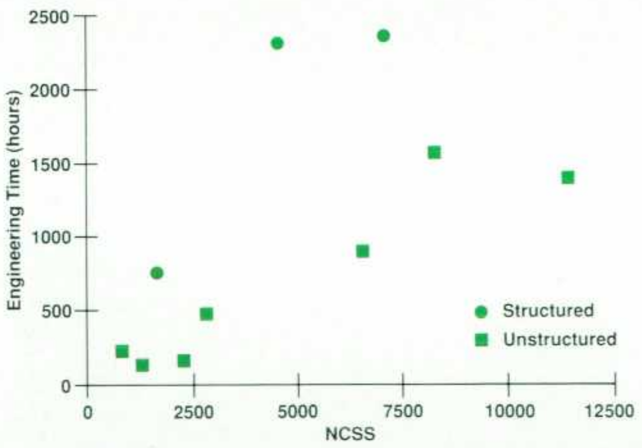


Fig. 3. *Engineering hours versus NCSS (noncomment source statements).*

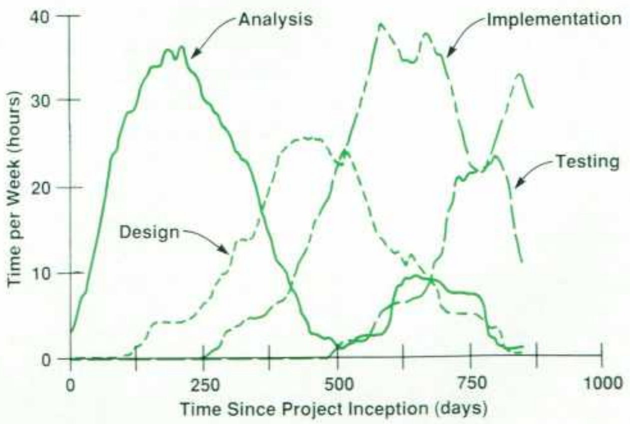


Fig. 4. *Engineering hours per week for the structured team. This is a 25-week moving average.*

Reusability

Reusability of code is a major productivity issue at HP in general. Several of the products at our division have already reached reuse figures of 50% and we are working to increase that figure in the future. Experts suggest that, on average, reusing a piece of software requires only 20% of the effort of writing it from scratch. Thus, any activity that leads to greater reuse of code will have major benefits for productivity. Code developed using structured techniques encourages more reuse in the future because it has better documentation on each function. In addition, the interface between that function and the outside world is more clearly stated by the structured design documentation.

A major concern is the maintenance of the software documentation. Documentation that is not kept up to date is of less value than no documentation, since it can be misleading. There is a very weak connection between structured analysis and structured design. This makes it difficult and often impractical to keep the structured analysis up to date because of changes in the design and coding. The structured team chose not to keep the structured analysis documentation up to date when design and coding changes were made later in the project. This takes away some of the documentation benefits provided by the structured methods. As back annotation methods are incorporated into the tools, this deficiency will be corrected.

Communication

One of the positive attributes cited in the literature about the structured methods is that they serve as a good communication tool between team members. The project saw some of this benefit, but not as much as was originally expected. Structured methods are not a team communications panacea.

Each software designer on this project was assigned a nearly autonomous subproject. Although there were interactions between the subprojects, the subprojects were defined to minimize these interactions. The structured analysis documentation for each subproject was large enough to cause difficulty in obtaining the number of design reviews that were necessary. For another team member to understand the analysis, much additional verbal explanation was required. However, the structured analysis was very useful to the developers of that analysis in fully understanding their own subprojects.

Fig. 7 shows the staffing profile of hardware and software engineers during product development. The staffing of software engineers lagged behind hardware engineers because of an initial underestimation of the software task and the shortage of software engineers. As a result, there were some problems discovered during the hardware/software integration that cost valuable schedule time. Since we were developing a system that consisted of both hardware and software, it would have been beneficial to have included the hardware interfaces into the structured analysis. This capability would have aided the hardware/software integration by providing additional communication links between the hardware and software groups.

There is probably a strong benefit in communication with structured analysis if the whole project team uses the methodology to develop the same system model. This enables each team member to have a better understanding of the product's requirements, and helps designers understand the task in the same way.

Quality

High product quality improves customer satisfaction, decreases maintenance costs, and improves the overall productivity of the development team. The FURPS model was used as the basis of comparison between the two methodologies.

Functionality

Functionality can best be measured by customer acceptance of the product. The product resulting from this project is still in the early stages of its product life and customer acceptance of its functionality is yet to be determined. It would also be difficult to separate the functionality of the code generated by the two methods.

However, we believe that the rigor required by the structured methods is an important contribution to the development life cycle. Structured analysis was a valuable exercise for the software designers in obtaining an understanding of the customer requirements. Although the structured

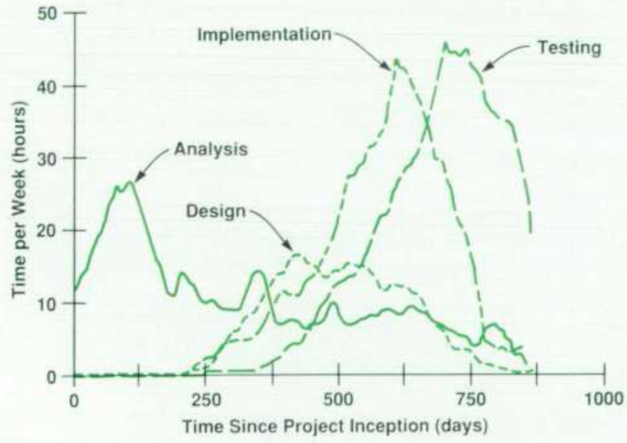


Fig. 5. *Engineering hours per week for the unstructured team. This is a 25-week moving average.*

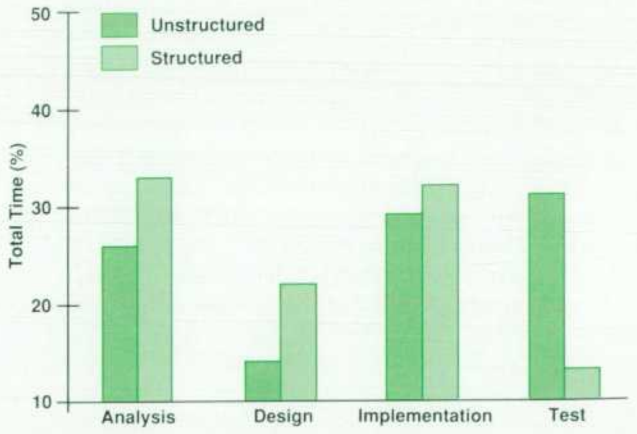


Fig. 6. *Percentage of time each method spent in the various phases of the software life cycle.*

analysis was not used directly as a communication tool, it helped the software designers identify the correct questions to ask when issues related to functionality were addressed. These benefits assist the product definition, and enhance the chances that the product will meet all user expectations.

We also believe that the entire team benefited from the efforts of the structured group. Since much of the initial product definition was performed using structured methods, software designers joining the project later benefited greatly from the initial structured work.

Usability

The look and feel of the user interface was of prime importance to the project. Structured analysis was useful in breaking up the interface functionality into its elemental commands. However, manual pages combined with verbal discussion proved to be more effective in defining the details of the interface.

Neither methodology appeared to enhance usability more than the other. However, the structured methods can help the software designer define user interface functionality. Another method that appears to be a better method for communicating and testing of user interface ideas is rapid prototyping. This method was not used on the user interface.

Reliability

Reliability is measured by the number, types, and frequency of defects found in the software. The defects found in code created using the structured methods were compared with those using the traditional methods. Table I outlines the defect rate normalized to defects per NCSS for both prerelease and postrelease defects. Prerelease defects were found during formal abuse testing, casual use by other individuals, and the code's designer. Postrelease defects include prerelease defects and the defects found in the first four months after product release. All the postrelease defects were found internally either by abuse testing or by casual use. No customer defects had been reported at the time of this study.

Table I
All Defects

	Prerelease Defects/NCSS	Postrelease Defects/NCSS
Unstructured	0.0041	0.0052
Structured	0.0036	0.0050

Although the structured code shows a slightly lower defect density than the unstructured code, the differences are not statistically significant (using a statistical test that compares two Poisson failure rates[6]).

Low-severity defects are considered to be caused by typical coding errors that occur at the same frequency, independent of the analysis and design methods used. Thus, using all the DTS defects is not truly indicative of the underlying defect density. Another way of characterizing the defect density is to look only at severe defects, those classified as serious or critical. Table II examines these defects.

Table II
Serious and Critical Defects

	Prerelease Defects/NCSS	Postrelease Defects/NCSS
Unstructured	0.0009	0.0010
Structured	0.0007	0.0008

Again, the structured code shows a slightly lower density but the difference is not significant. We knew that the code designers' rigor in logging defects that they found themselves varied a great deal. Since this might affect the quality results, we examined only the defects logged during formal abuse testing. It was again found that there was no statistical difference between the methodologies.

Our theory, developed from these results, is that the final reliability of a product is mainly a function of environmental factors, including management expectations and peer pressures. For example, reliability can either be designed in at the beginning using structured methodologies or tested in at the end of the project with thorough regression tests.

Performance

In the design of structured modules, one is encouraged to reduce complexity of the modules by breaking up functionality. This results in more function calls, and increases the processing time of critical functions.

The software for this project had critical performance requirements for communication rates. The processing of the bytes of data by the firmware was the critical path. The critical functions had to be recoded using in-line assembly code. Although structured methods were not used on this communication firmware, it is our opinion that the structured methods as used on this project would not have helped to identify performance-critical functions or to improve the performance of these functions.

A newer form of structured analysis[8] has been developed and is based on modeling of the problem data as the first step. The data is represented in an information model with the state control diagrams showing the data control. This may in fact prove to be a better model for real-time applications that have critical performance requirements.

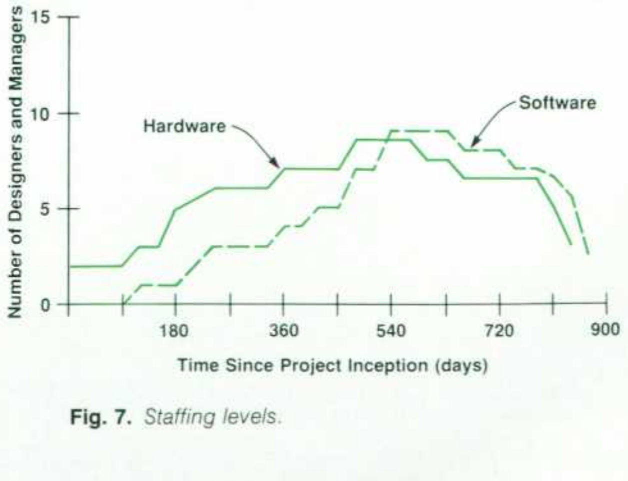


Fig. 7. *Staffing levels.*

Attribute	Structured	Unstructured
Productivity		
Development Time	longer	shorter
Manageability	better	worse
Reusability*	easier to reuse	harder to reuse
Communication*	slightly better	slightly worse
Quality		
Functionality*	better	worse
Usability*	no difference	no difference
Reliability	no difference	no difference
Performance*	worse	better
Supportability	better	worse

*These items had no objective metrics kept on them.

Fig. 8. *Summary of comparison between structured and unstructured methodologies.*

Supportability

A large part of the costs of software development and maintenance can be attributed to maintenance activities. Software that is supportable will have lower maintenance costs. Maintenance not only refers to the repair of defects, but also to software enhancement and modification.

One factor that relates strongly to software supportability is the complexity level of each function. Functions with lower complexity tend to make it easier to modify and repair defects. For this project, the average complexity of each module of the structured code was less than the average for the unstructured code. Table III shows a summary of the complexity statistics for the two code types.

Table III
McCabe's Cyclomatic Complexity

	Structured	Unstructured
Mean	5.7	7.2
Number of Functions	268	656

Modules with a cyclomatic complexity greater than 10 may be too complex and should be reviewed for possible restructuring.[5] Only 13% of the structured code had a complexity greater than ten, while the unstructured code had 20%. A statistical test (comparison of two binomial fractions[9]) showed this to be a significant difference.

The discipline of the structured methods helps the designer to write consistent documentation that can be used by those who must support the product.

Conclusions

Our analysis of structured and unstructured techniques produced mixed results. It was apparent that the structured methodologies on this project did not provide improvement in the project development time. In fact, a longer development time was measured. This was partially because of the time required for learning the structured methodologies. However, the manageability of designers using structured techniques is higher because of well-defined development cycles. Also, the structured code appears to be more reusable, so it should improve the productivity of future projects reusing this code.

In this project, structured methods didn't appear to improve the reliability of the final code significantly. It is our opinion that reliability is more a function of the team's environmental factors. The structured code appears to be more supportable since module complexity was lower. Structured methods do not appear to be a major benefit in developing code where performance factors are a main requirement. No significant benefit was seen for the areas of functionality and usability except in those projects where the techniques were used to enhance communication of the product design and specification.

Some aspects of the structured methodology were disappointing. However, what was most important for our development team was the discipline that the structured methods added to the early phases of the product definition. We feel the results of the structured methods are positive enough to continue using these methods on future projects. Fig. 8 summarizes the results of the comparison of the two methodologies.

References

1. T. DeMarco, *Structured Analysis and System Specification*, Prentice-Hall, Englewood Cliffs, 1979.
2. M. Page-Jones, *The Practical Guide to Structured Systems Design*, Prentice-Hall, Englewood Cliffs, 1980.
3. B.A. Thompson and D.J. Ellis, "Hierarchy Chart Language Aids Software Development," *Hewlett-Packard Journal*, Vol. 37, no. 3, March 1986.
4. S.R. Blair, "A Defect Tracking System for the UNIX Environment," *Hewlett-Packard Journal*, Vol. 37, no. 3, March 1986.
5. T. DeMarco, *Controlling Software Projects*, Yourdon Press, New York, 1982.
6. W. Nelson, "Confidence Intervals for the Ratio of Two Poisson Means and Poisson Predictor Intervals," *IEEE Transactions on Reliability*, Volume R-19, no. 2, May 1970.
7. W.A. Fischer, "Keeping Pace on Projects," *ComputerWorld*, July 25, 1988.
8. S. Shlaer and S.J. Mellor, *Object-Oriented Systems Analysis: Modeling the World in Data*, Prentice-Hall, Englewood Cliffs, New Jersey, 1987.
9. K. A. Brownlee, *Statistical Theory and Methodology*, John Wiley and Sons, New York, 1965.

An Object-Oriented Methodology for Systems Analysis and Specification

A methodology is proposed that enables analysts to model and specify a system's data, interactions, processing, and external behavior before design.

by Barry D. Kurtz, Donna Ho, and Teresa A. Wall

AS SOFTWARE SYSTEMS BECOME LARGER and more complex, the task of systems analysis continues to increase in effort and difficulty. Traditional methodologies for systems analysis sometimes fail to meet the analyst's expectations because of their limitations in properly capturing and organizing all of the information that must be considered. Object-oriented systems analysis (OSA) is an approach to systems analysis and specification that builds upon the strengths of existing methodologies and, at the same time, addresses their weaknesses. This paper describes the basic concepts of the OSA methodology.

Systems Analysis

A system is an organized collection of people, machines, procedures, documents, data, or any other entities interacting with each other to reach a predefined goal.[1] Analysis is the study of a problem, prior to taking some action.[2] Systems analysis is the study of a collection of related and interacting objects (existing or proposed) for the purpose of understanding and specifying its data, processing, and external behavior.

Object-oriented systems analysis (OSA) is a systematic approach to the study of a system problem. The foundation of OSA's conceptual model is the set of components or objects in the system. The study of these objects is organized and conducted in a manner that does not unnecessarily constrain implementation. In fact, we believe that designs based on OSA specifications can be procedure-oriented or object-oriented with equal success.

Objects in the OSA Methodology

An object is an abstraction of entities or concepts that exist in or are proposed for a system. For example, the rectangle in Fig. 1 depicts an OSA object called Jet Plane. This object represents a class of things where each member has the attributes and behavior of a jet plane. Two possible members of this class are a jet fighter and a jet passenger plane. In object-oriented terms, each member is called an instance of the object class. Fig. 2 further demonstrates the community of attributes and behavior required of each member of the object class. The circle on the top in the figure represents all of the attributes and behavior of a jet fighter. The circle on the bottom in the figure represents all of the attributes and behavior of a jet passenger plane. The shaded area represents the common attributes and behavior that allow each plane to be a member of the jet plane object class.

The key elements of a system are the objects that exist in the system, the data that must be processed, and the desired behavior of the system. Traditional methodologies offer two basic approaches to modularize these elements: a function-oriented approach which organizes the analysis

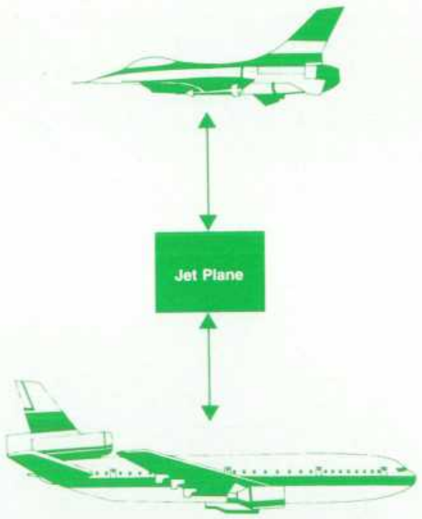


Fig. 1. *An OSA object. Each member of this object class has the behavior and attributes of a jet plane.*

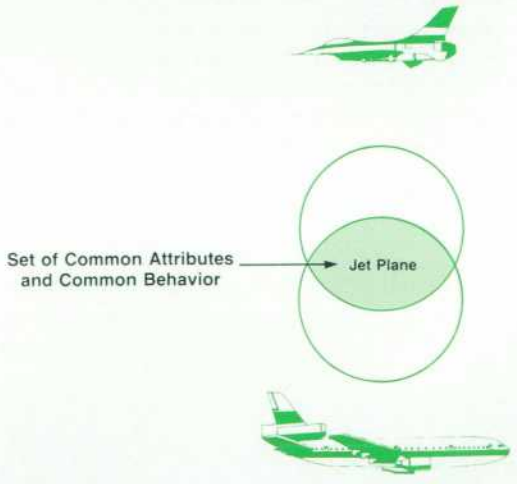


Fig. 2. *Commonality of attributes and behavior of objects of class* Jet Plane.

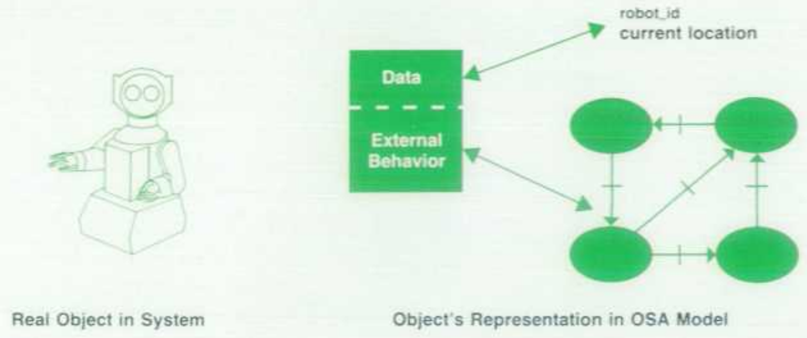


Fig. 3. *Representation of an object's data and behavior characteristics in the OSA system model.*

model through a hierarchy of functions, and a data-oriented approach which emphasizes a hierarchy of data structures. OSA imposes a natural modularization on the system model through the emphasis on objects (see Fig. 3). Each object has associated attributes (data) and behavior (legal states and functions). An object in the OSA system model has a one-to-one correspondence with an actual object in the system. This provides an easy mapping between the analysis model and the components of the system under study.

Collecting Preliminary Specifications

Before attempting to construct an OSA system model, it is important to collect or generate a preliminary list of specifications. These specifications are gathered from discussions and interviews with users and managers of the system under study. The preliminary specifications are rather informal and are usually documented with natural language text and hand-drawn graphics. These specifications should answer most of the following questions that may be posed by the analysis team:

- What is the subject of the analysis? Is it an existing system, a proposed system, or a combination of both?
- What are the specific problems that need to be solved and what are the objectives for the solution?
- What are the logical boundaries of the system study? Does the entire system need to be studied in detail or can a smaller subset be studied?
- What are the known constraints of the system? Are there special performance constraints, interface constraints, hardware constraints, and so on?
- What are the major components of the system? What needs to be done with each component? What should each component do in the system?

The specific format of the preliminary specifications may vary from project to project. The important factor here is that the preliminary specifications are collected before construction of the formal OSA specifications. The following is a small example of a preliminary specification that an analyst might use to gain a basic understanding of a system problem.

A vending machine must be produced for distributing products to customers. The customer begins the vending process by depositing a token in the machine's coin receptacle. The token must be checked for size, weight, thickness, and type of edge. Valid tokens (coins) are quarters, dimes, or nickels, and anything else is considered a slug. Slugs are rejected and sent to the coin return slot. When a valid coin is accepted by the machine and sent to the coin box, the amount of current customer payment is incremented by the value of the coin.

Each product dispenser contains zero or more products. All of the products in one dispenser have the same price. The customer's product selection is made by identifying the dispenser to be activated. If the dispenser is not empty and the current customer payment equals or exceeds the cost of the product, the product should be dispensed to the product delivery slot and any change due returned to the coin return slot. If the dispenser is empty, coins equaling the current customer payment should be dispensed to the customer in the coin return slot.

If the customer payment is less than the price of the products in the selected dispenser, the machine should wait for the customer to deposit the proper payment. If the customer decides not to make a selection the current amount deposited should be returned.

Building Object-Relationship Diagrams

After preliminary specifications are gathered for the major components of the system, the analyst can begin building object-relationship diagrams (ORDs). The ORD portion of the analysis provides a formal repository for

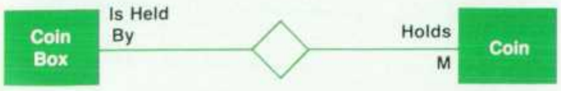
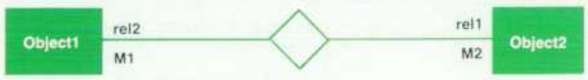


Fig. 4. *An example of an object-relationship diagram. The coin box holds many (M) coins.*



<Object1> is related by <rel1> to <Object2>.
<Object2> is related by <rel2> to <Object1>.
<M2> designates how many instances of <Object2> are associated with each instance of <Object1>.
<M1> designates how many instances of <Object1> are associated with each instance of <Object2>.

Fig. 5. *Formal definitions in object-relationship diagrams.*

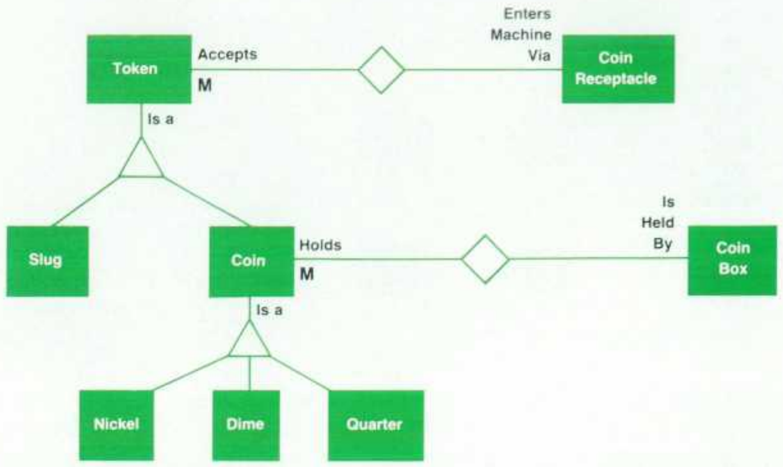


Fig. 6. *Partial object-relationship diagram for the vending machine example. It shows that a token enters the vending machine through the coin receptacle, and that a token may be a coin or a slug. The only objects considered to be a legitimate coin in the system are nickels, dimes, or quarters.*

answers to the following questions:

- What are the components of the system (both abstract and concrete)?
- What are the important formal relationships between the system components?

The first step for producing ORDs is to make a list of the nouns contained in the preliminary specification. These nouns are classified by the analyst as objects (system components), attributes of objects, or synonyms for other objects or attributes. Synonyms must be eliminated and attributes must be associated with existing objects. From our vending machine example, some of the nouns that represent objects are token, coin receptacle, slug, coin, coin box, nickel, dime, and quarter. Also from the same example, the nouns that would be considered attributes include weight, thickness, and price.

Objects are depicted in object-relationship diagrams by the object name enclosed in a rectangle. An ORD showing a relationship between coins and a coin box is shown in Fig. 4. The ORD in the figure states that a coin box holds one or more coins. The capital M in the figure is shorthand for one or more. A relationship line without an M signifies that the relationship is one-to-one. ORDs are similar in use and meaning to entity-relationship diagrams.[3] The formal meaning of ORD relationships is shown in Fig. 5, and Fig. 6 shows a partial ORD for the vending machine example.

Object-relationship diagrams have a subcomponent called *concept diagrams*. Concept diagrams provide a quick method for capturing the information relationships and interactions between objects. For example, Fig. 7 shows a concept diagram stating that a coin receptacle needs to know the thickness of a token. A dotted arrow depicts an informational relationship between objects. The information may be an attribute of an object or an event that another object needs to know about.

Fig. 8 shows a concept diagram that states that a customer deposits a token into the vending machine. A solid arrow is used to show an action that an object performs upon another object. Since informational relationships and object interactions are relatively easy to extract from the preliminary specifications, concept diagrams may be used frequently at the beginning of the analysis.

Natural Language Object Descriptions

The object-relationship diagrams identify classes of components in the system and formal relationships between the components. The next step is to provide more information about each object and its attributes. The object description is documented with natural language text and contains the following information:

- Name. The name of the class of system components represented by the object.
- Description. A brief description of the general characteristics for instances of the object.
- Assertions. Things that must always be true regarding attributes or behavior of instances of this class. For example, an object may need to be kept at a certain temperature to ensure the desired behavior.
- Attributes. Required attributes for each instance (such as identifiers, status variables, etc.). The domain or legal range of values for each attribute must be specified here.

The specific format of the object description may vary from project to project but it should include at least the above information. Fig. 9 shows a natural language description for the vending machine example.

Building Behavior Specifications

Using the foundation of object-relationship diagrams and individual object descriptions, the analyst builds a behavior specification for each object. Since internal behavior is usually a function of implementation, the analyst concentrates on the external behavior of each object. Exter-

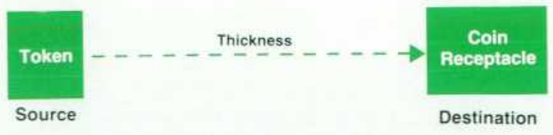


Fig. 7. *An example of a concept diagram. The coin receptacle needs to know thickness of the coin. The concept diagram shows the informational relationship and interactions between objects. The dashed line indicates information flow.*

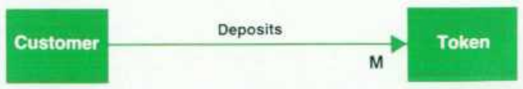


Fig. 8. *A concept diagram that shows an action of one object on another. The solid line indicates action.*

Object Name: Token

Description:

A token is something the customer deposits into a vending machine to purchase a product.

Assertions:

The diameter and thickness are small enough to allow it to fit in the coin slot.

Attributes:

Diameter Domain:	$0 <$ diameter $\leq$ height of coin slot
Thickness Domain:	$0 <$ thickness $\leq$ width of coin slot
Weight Domain:	not yet defined
Edge Type Domain:	(smooth, serrated)

Fig. 9. *A natural language description for a token object in the vending machine example.*

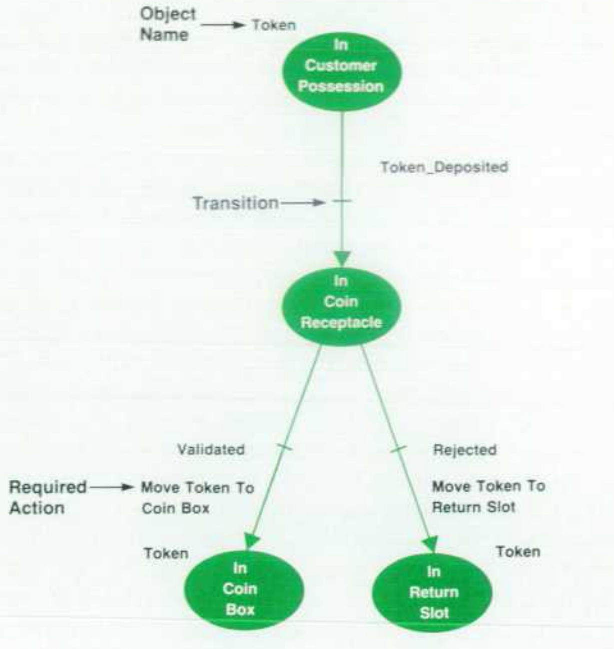


Fig. 10. *State net showing the behavior of a token object. When a token is deposited, it transfers from the customer's possession to the coin receptacle. If the token is a valid coin, it is transferred to the coin box as a coin. Otherwise, a rejected token is returned to the return slot as a slug.*

nal behavior has three components:

- The externally perceived states or conditions of each object
- The actions that must be performed on each object
- The actions that each object must perform.

Each of these components may be specified by natural language text. However, experience has shown that such an approach leads to ambiguity and confusion. To help ensure consistent interpretation of behavior specifications, a more formal method is required.

There are several existing formal methods for describing external system behavior. These methods include finite state machines, decision tables, Petri nets, R-nets, precondition/action/postcondition tuples and others.[4,5] The following criteria were used to choose a behavior specification method for OSA:

- The method should encourage the analyst to identify and classify system components before specifying behavior.
- The method should allow the analyst to specify an object that exhibits multiple action conditions (concurrent activities).
- The method should provide for high traceability between the actual system components, object-relationship diagrams, and behavior specifications.

The behavior specification technique developed for OSA is called *state nets.* State nets are based on a restricted form

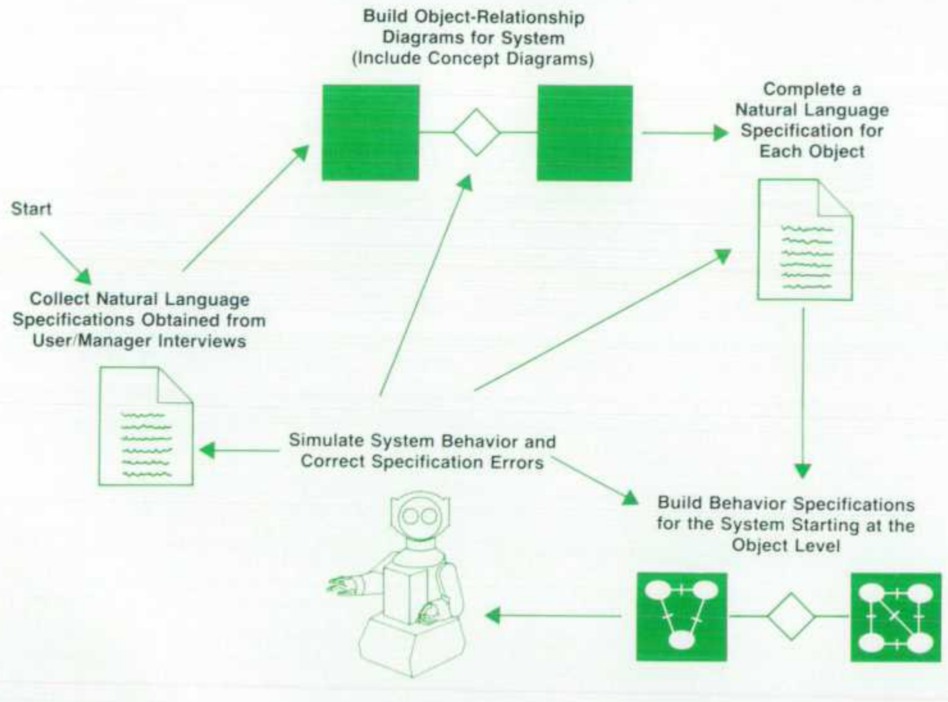

Build Object-Relationship Diagrams for System (Include Concept Diagrams)
Complete a Natural Language Specification for Each Object
Start
Collect Natural Language Specifications Obtained from User/Manager Interviews
Simulate System Behavior and Correct Specification Errors
Build Behavior Specifications for the System Starting at the Object Level

Fig. 11. *Process flow for the OSA methodology and the interactions between the object and methodology steps.*

of Petri nets. There are several possible configurations of state nets that use the full power of Petri nets for expressing concurrent activities. For this overview, however, we will limit the discussion to a model similar to a finite state machine.

A sample state net representing some of the states and actions required for a token is shown in Fig. 10. The states or conditions of the token object are represented by ovals in the figure. The states are typed with the name of the object being specified (token in this example). Labeling the state with the object name means that instances of that object may exhibit the specified state. The transitions between states are represented by short horizontal bars. The labels next to the transitions specify the events that cause instances of the object to transition from one state to another. In Fig. 10, the Token_Deposited command causes the machine to transition from the In Customer Possession state to the In Coin Receptacle state. The actions required to establish a new state are shown as labels on the input arrow connected to the new state. For example, the state net diagram states that the action Move Token To Return Slot must be performed before the machine is in the In Return Slot state.

System Behavior Simulation

Once the state nets for the system have been completed, the analysis team is prepared to simulate executions of the system. The simulation is performed using state nets to follow conditions and states exhibited by the system. The object-oriented nature of state nets aids in hiding complexity during simulations. This allows the analysis team to achieve good behavior coverage even when automated simulation tools are not available.

During system simulation, several classes of errors may occur that require modifications to the current OSA model or specifications. Simulation will aid the analyst in discovering the following problems:

- Missing or ambiguous information in the preliminary specification
- Missing or incorrect object-relationship diagrams
- Missing objects or incorrect object descriptions
- Missing or incorrect behavior specifications.

Any of these errors will require another pass through one of the analysis phases previously discussed. The object-oriented nature of the OSA model enhances traceability to the components of the system related to a particular error. Fig. 11 shows the process flow for the OSA methodology and shows possible interactions between the various methodology steps.

Conclusion

The contributions of the object-oriented analysis methodology include:

- It provides tools for capturing the key elements of a system—components, data, and behavior.
- It emphasizes analysis before design.
- It supports a natural modularization scheme by organizing the entire analysis around objects.
- Its system model provides high traceability between the model components and the actual components of the system under study.

The methodology is currently under research and verification. It has been applied to several problems involving hardware and software components with encouraging results. A prototype tool is being developed to aid analysts in processing textual specifications and capturing the meaning of systems problems using the OSA methodology. The goals now are to finish the tool, develop training materials, and continue to verify applicability of the methodology to real-world projects.

References

1. A.Z. Atkas, *Structured Analysis and Design of Information Systems*, Prentice-Hall, 1987.
2. T. DeMarco, *Structured Analysis and System Specification*, Yourdon, 1979.
3. P. Chen, *The Entity-Relationship Approach to Logical Data Base Design*, Q.E.D. Information Sciences, 1977.
4. A.M. Davis, "A Comparison of Techniques for the Specification of External System Behavior," *Communications of the ACM*, Vol. 31, no. 9, 1988, pp. 1098-1115.
5. B. Liskov and J. Guttag, *Abstraction and Specification in Program Development*, MIT Press, 1986.

VXIbus: A New Interconnection Standard for Modular Instruments

This standard will allow users to mix modules from different manufacturers in a system contained in a single mainframe.

by Kenneth Jessen

THE GOAL OF THE VXIBUS is to provide a technically sound standard for modular instruments that is based on the VMEbus (defined later) and is open to all manufacturers. It is a specification for interconnecting and operating various modules from a variety of manufacturers within a single mainframe to satisfy the need for high-performance, high-density instrumentation.

Users are able to select from four module sizes and are free to choose modules and a mainframe from different suppliers based on price and performance. The VXIbus standard ensures compatibility of all the elements within a VXIbus system. For example, a user may find that a particular digital multimeter module offers the best combination of price and measurement capability for a particular job, but that the best function generator for the application comes from another manufacturer. The user may then select a third manufacturer for the mainframe. This amounts to unprecedented flexibility in the creation of an instrumentation system.

VXIbus Evolution

Many years ago there were only bench instruments. They had front panels for control and displays for data output. As computers entered the picture, digital interfaces (such as binary-coded decimal) came into use. These were typically offered as options to the basic bench instrument and varied from instrument to instrument and from manufacturer to manufacturer. In the early 1970s, a better interface standard, the HP Interface Bus, or HP-IB, was developed by Hewlett-Packard. This became an industry standard (IEEE 488, IEC 625) and is currently used by a wide range of manufacturers.

Instruments continued to retain their front-panel controls, but as applications moved toward automatic test and measurement systems, the need for front panels diminished. Instruments began to appear on the market either without front panels or with detachable front panels used only during initial setup.

To reduce the size of a test system, modular instruments have grown in popularity, but until recently there were no industry standards for such systems, so there was no possibility of mixing modules from different manufacturers within a mainframe. Often modules from a manufacturer were not compatible with different mainframes from that same manufacturer.

In 1979, Motorola Semiconductor Products Corporation published a description of what it called the VERSAbus. It defined a computer-type backplane and cards. Eurocard board sizes were proposed and the VERSAbus-E version was renamed the VMEbus. In 1982 the IEC (International Electrotechnical Commission) proposed that the VMEbus be accepted as an international standard.

There was a great deal of pressure from both military and commercial users to have an open architecture for modular instruments. The U.S Air Force asked the MATE User's Group, formed in 1984, to develop recommendations for the standardization of instruments on cards and expressed a desire to use as much commercial equipment as possible. Out of this activity came the need for a standard for all instrument manufacturers.

Consortium Formed

In July 1987, a group of five instrument manufacturers committed to the development of a modular instrument standard based on the VMEbus standard. The original five are Colorado Data Systems Corporation, Hewlett-Packard Company, Racal Dana Instruments Corporation, Tektronix Corporation, and Wavetek Corporation. Other companies have since joined this consortium including Brüel & Kjær Corporation, National Instruments Corporation, Keithley Instruments Corporation, John Fluke Manufacturing Company, and GenRad Corporation.

The major limitations of the original VMEbus standard for instrument manufacturers were insufficient board space

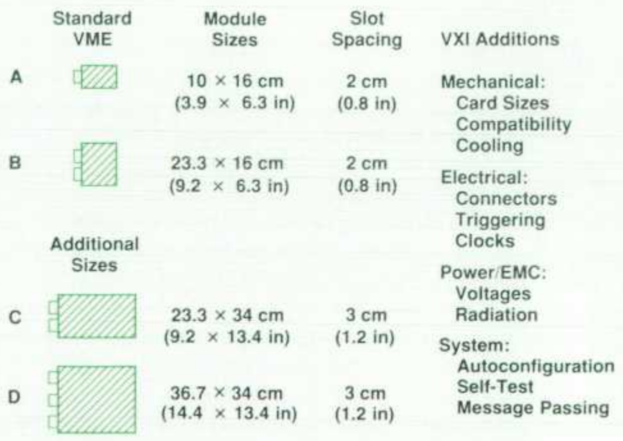

Standard VME
Module Sizes
Slot Spacing
VXI Additions
A
10 × 16 cm (3.9 × 6.3 in)
2 cm (0.8 in)
B
23.3 × 16 cm (9.2 × 6.3 in)
2 cm (0.8 in)
Additional Sizes
C
23.3 × 34 cm (9.2 × 13.4 in)
3 cm (1.2 in)
D
36.7 × 34 cm (14.4 × 13.4 in)
3 cm (1.2 in)
Mechanical: Card Sizes Compatibility Cooling
Electrical: Connectors Triggering Clocks
Power/EMC: Voltages Radiation
System: Autoconfiguration Self-Test Message Passing

Fig. 1. *The VXIbus standard (VMEbus Extensions for Instrumentation) includes the two original VMEbus module sizes, A and B, plus two new larger modules, C and D. Smaller modules can be inserted into mainframes designed for the larger sizes. The C size mainframe is expected to be the most popular for test systems.*

and the lack of an adequate connector definition for instrumentation. Space between modules was restricted to 0.8 inch. The VXIbus standard (VMEbus Extensions for Instrumentation) adds two new larger module sizes to the two module sizes set up under the VMEbus standard. With the new modules, the space between modules is increased to 1.2 inches to permit adequate shielding and cooling. The VMEbus was developed for computers and did not define a backplane specifically for instrumentation needs. The VMEbus standard also did not address the problems of electromagnetic interference (EMI), power dissipation, or chassis cooling, and left some connector pins undefined. The new VXIbus standard covers these items as well as other issues such as protocols between modules, configurations, memory allocations, and commands.

As mentioned before, the general concept of the VXIbus standard is an open architecture for modular electronic instrumentation that allows a variety of manufacturers to supply modules to operate together in the same mainframe chassis. The intention is for the standard to be open to anyone who wishes to use it whether they are among the original founders or not. In keeping with its objective, the VXIbus standard is in the public domain. There are no copyrights on the documentation, and there are no patents or licensing requirements. Manufacturer ID numbers are provided free from the VXIbus consortium, and over 70 manufacturers have requested and been granted numbers.

The idea of VXIbus instrumentation is not to replace traditional instruments but rather to offer distinct advantages for certain applications, including:

- High-speed communications between modules
- Multichannel data acquisition
- Precision timing between modules
- Smaller size for a typical instrument system
- Ease of integrating a test system.

The VXIbus Standard

There are four basic module sizes including the original two sizes that were part of the VMEbus standard. These original sizes are renamed A and B in the VXIbus standard. The larger two sizes, C (123 in^2) and D (193 in^2), can include additional connectors, as shown in Fig. 1. All the connectors are 96-pin DIN-type and are referred to as P1, P2, and P3. The A size board has the P1 connector. Sizes B and C may have the P2 connector as well as the required P1 connector. The largest module size, D, may have the P3 connector in addition to the P1 and P2 connectors. The idea is to provide improved capability as a function of increased size.

A VXIbus mainframe may accept up to a dozen C or D size modules in addition to a required Slot 0 module. For very complex products, a module may span more than one VXIbus slot.

To ensure compatibility, all pins on all three connectors are defined by the VXIbus standard. The P1 and P2 pin definitions are shown in Fig. 2. Their functional use and protocol are called out. Also defined are the interfacing of a module to the backplane and the electrical characteristics of the backplane. The capabilities of each connector can be viewed as a pyramid, with P1 covering the VMEbus specification. P2 adds capability required for instrumentation, such as more ground pins and more power supply pins. A 10-MHz clock and ECL and TTL trigger lines are

P1 Pin Number	Row a Signal Mnemonic	Row b Signal Mnemonic	Row c Signal Mnemonic	P1 Pin Number
1	D00	BBSY	D08	1
2	D01	BCLR	D09	2
3	D02	ACFAIL	D10	3
4	D03	BG0IN	D11	4
5	D04	BG0OUT	D12	5
6	D05	BG1IN	D13	6
7	D06	BG1OUT	D14	7
8	D07	BG2IN	D15	8
9	GND	BG2OUT	GND	9
10	SYSCLK	G3IN	SYSFAIL	10
11	GND	BG3OUT	BERR	11
12	DS1	BR0	SYSRESET	12
13	DS0	BR1	LWORD	13
14	WRITE	BR2	AM5	14
15	GND	BR3	A23	15
16	DTACK	AM0	A22	16
17	GND	AM1	A21	17
18	AS	AM2	A20	18
19	GND	AM3	A19	19
20	IACK	GND	A18	20
21	IACKIN	SERCLK(1)	A17	21
22	IACKOUT	SERDAT(1)	A16	22
23	AM4	GND	A15	23
24	A07	IRQ7	A14	24
25	A06	IRQ6	A13	25
26	A05	IRQ5	A12	26
27	A04	IRQ4	A11	27
28	A03	IRQ3	A10	28
29	A02	IRQ2	A09	29
30	A01	IRQ1	A08	30
31	−12V	+5VSTDBY	+12V	31
32	+5V	+5V	−5V	32

P2 Pin Number	Row a Signal Mnemonic	Row b Signal Mnemonic	Row c Signal Mnemonic	P2 Pin Number
1	ECLTRG0	+5V	CLK10+	1
2	−2V	GND	CLK10−	2
3	ECLTRG1	RSV1	GND	3
4	GND	A24	−5.2V	4
5	LBUSA00	A25	LBUSC00	5
6	LBUSA01	A26	LBUSC01	6
7	−5.2V	A27	GND	7
8	LBUSA02	A28	LBUSC02	8
9	LBUSA03	A29	LBUSC03	9
10	GND	A30	GND	10
11	LBUSA04	A31	LBUSC04	11
12	LBUSA05	GND	LBUSC05	12
13	−5.2V	+5V	−2V	13
14	LBUSA06	D16	LBUSC06	14
15	LBUSA07	D17	LBUSC07	15
16	GND	D18	GND	16
17	LBUSA08	D19	LBUSC08	17
18	LBUSA09	D20	LBUSC09	18
19	−5.2V	D21	−5.2V	19
20	LBUSA10	D22	LBUSC10	20
21	LBUSA11	D23	LBUSC11	21
22	GND	GND	GND	22
23	TTLTRG0	D24	TTLTRG1	23
24	TTLTRG2	D25	TTLTRG3	24
25	+5V	D26	GND	25
26	TTLTRG4	D27	TTLTRG5	26
27	TTLTRG6	D28	TTLTRG7	27
28	GND	D29	GND	28
29	RSV2	D30	RSV3	29
30	MODID	D31	GND	30
31	GND	GND	+24V	31
32	SUMBUS	+5V	−24V	32

Fig. 2. *Pin definitions for the P1 and P2 connectors. The P1 connector follows the VME bus assignments. The P2 assignments shown are for slots 1 through 12.*

also defined on P2. The P3 connector adds even more capability for specialized applications, including a 100-MHz clock similar to the 10-MHz clock on P2, a 100-MHz synchronizing signal, and ECL trigger lines in a star configuration for module-to-module triggering.

Slot 0 Module

To ensure that every mainframe can perform its minimum functions, a Slot 0 module is defined. For VXIbus systems with the P2 connector, the only Slot 0 module requirement is to provide VMEbus system controller functions, a 10-MHz, 100-ppm system clock, and module ID drivers. For systems with the P3 connector, the Slot 0 module must provide a 100-MHz clock. It is expected that manufacturers will add considerable more capability to the Slot 0 module than the standard requires.

Many Slot 0 modules will be message-based devices with commander capability in addition to the minimum functions. These will be able to identify the devices in the system, configure resources, manage self-test and diagnostics, configure address maps, configure the commander-servant hierarchies, and perform initial system operation. The capability of the Slot 0 module will usually be combined with the VXIbus to HP-IB (IEEE 488, IEC 625) interface function.

The 10-MHz clock originates in the Slot 0 module and is buffered on the backplane. It is distributed to each module as a single-source, differential ECL signal. The objective is to provide a high level of isolation and a common precision time base. Typical performance delivers less than 25 ps of jitter.

The ECL and TTL trigger lines defined on the P2 connector are bused to all modules including the Slot 0 module, as shown in Fig. 3. Any module may send or receive triggers over these lines, and the lines selected are user-programmable. Several protocols are used to ensure proper triggering between various manufacturers' modules. For example, a synchronous trigger protocol is defined to allow for triggering from one module to another. Other protocols include handshake responses from the receiving module. The ECL trigger lines can deliver trigger rates in excess of 50 MHz for high-performance applications exceeding the TTL trigger lines' 12-MHz capability.

System Requirements

It is a requirement that airflow and power supply capability be fully specified for the mainframe. A typical mainframe is shown in Fig. 4. Likewise, the necessary degree of cooling and power requirements must be defined for the modules. This allows the user to match the capabilities of the mainframe with the requirements of the modules. This type of information must be included in the product specifications, and the user will be able to determine in advance whether or not certain modules will work in a given mainframe.

The close proximity of modules within a mainframe makes EMI compatibility a necessary part of the specification. For this reason, tight electromagnetic compatibility and noise requirements are imposed on module manufacturers. In some cases, modules will have to be completely enclosed within a shield and grounded through the backplane. The requirements cover both near-field radiation

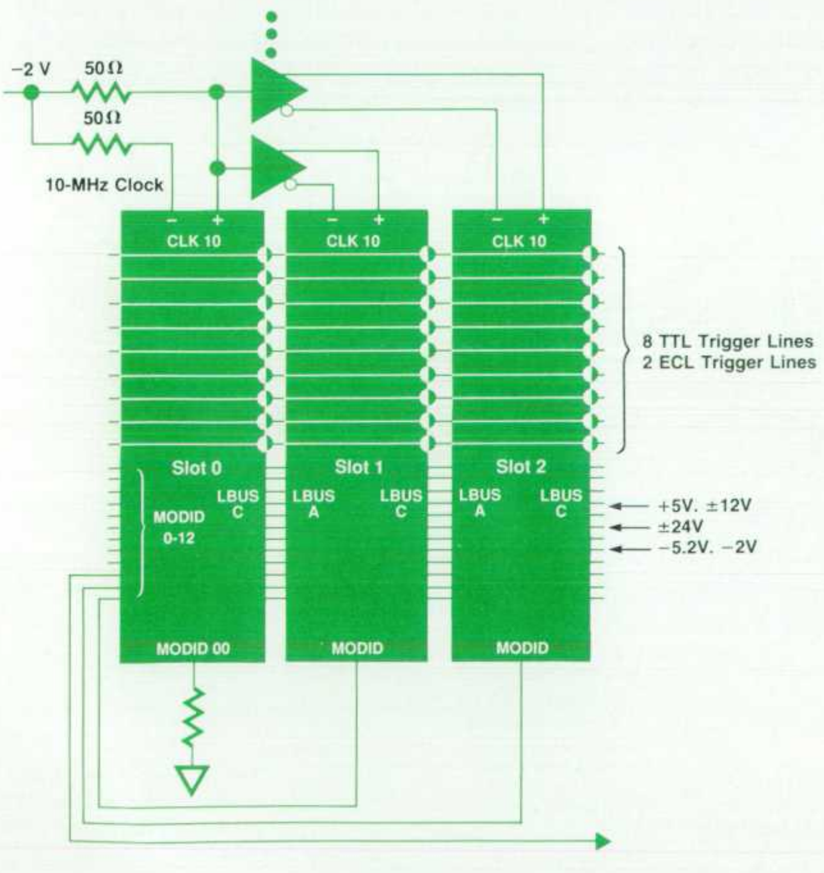


Fig. 3. *The 10-MHz clock originates in the Slot 0 module. ECL and TTL trigger lines are bused to all modules including the Slot 0 module.*

and susceptibility and conducted radiation and susceptibility of the power supply pins.

The VXIbus standard does allow for flexibility in the area of local bus communication. A local bus is one that runs from one module to its neighbor, but no farther. The local bus is intended for communication between modules either within a multiple-slot module or within a family of modules. The definition is left to the manufacturer. This flexibility creates a new problem, that of protection of electrically incompatible modules accidentally configured into adjacent slots by the user. A protection scheme using mechanical keying provides six logical classes of keys: TTL, ECL, analog low (−5.5V to +5.5V), analog medium (−16V to +16V), analog high (−42V to +42V), and a reserved class.

Along with hardware, the VXIbus standard also covers configuration and communication protocols. To avoid conflicts between modules, manufacturers must maintain common protocols. However, the VXIbus standard does not define things such as the operating system, system hierarchy, type of microprocessor, or type of network.

The VXIbus standard specifies device protocols so that nonconflicting portions of the VMEbus address space are used. A device will usually be a single module. However, several devices may exist on a single module and a single device may take up multiple slots. As many as 13 modules may exist in any one VXIbus subsystem, including the Slot 0 module. Up to 256 devices may exist in one VXIbus system, which may include several mainframes and several Slot 0 modules.

Types of Devices

The lowest level of capability is a set of configuration registers accessible on P1. These registers allow the system to identify the device type, model, manufacturer, address space, and memory requirements. Modules with this minimum capability are called register-based devices. All VXIbus devices must have these registers in the upper 16K of the 64K A16 address space. Each device is granted 64 bytes in this space, sufficient for many devices. There are a number of registers including ID, device type, status/control, and memory offset. The remaining register space is device dependent and may be used for communication purposes.

Memory requirements for devices needing additional address space must be readable in a defined register in the A16 address space. A resource manager reads this value shortly after power-on, then assigns the requested memory space by writing the module's new address into the device's offset register. This method positions a device's additional memory space in the A24 or A32 address space.

Instead of register-based, a VXIbus may be message-based. Message-based devices have a higher level of communication capability than register-based devices; they communicate using a word-serial protocol. Generally, message-based devices include a microprocessor and execute ASCII commands. They have communication registers accessible to other modules in the system. Other types of modules include memory devices and extended devices. A RAM or ROM card is an example of a memory device.

Control hierarchy is also defined in a VXIbus system. A commander is a module able to initiate communication with its servants based on the servants' capabilities. A servant may be either register-based or message-based. Commands to a message-based device can be sent using word-serial protocol. For register-based devices, communication is by register reads and writes and is device dependent.

The VXIbus concept of a command-servant relationship allows the creation of a virtual instrument whose collective capabilities yield a given measurement function. For example, a waveform generator might be teamed with a digitizer to form a network analyzer. A particular measurement

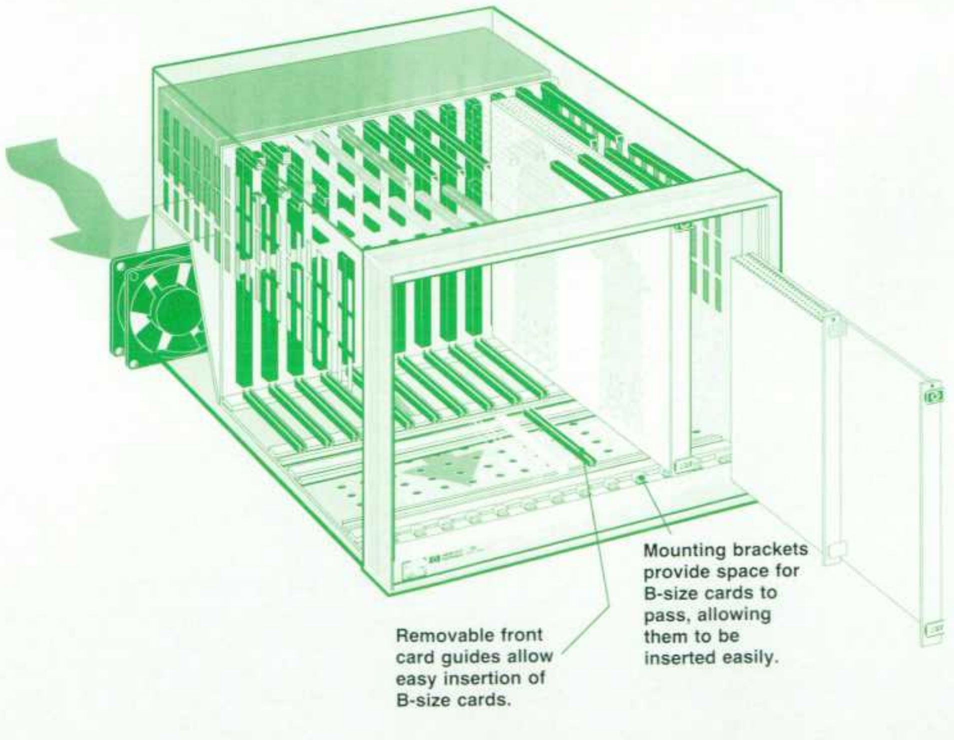

Mounting brackets provide space for B-size cards to pass, allowing them to be inserted easily.
Removable front card guides allow easy insertion of B-size cards.

Fig. 4. *A typical C size mainframe, showing C size cards, guides for B size cards, and air flow.*

capability is created through a combination of software and hardware. The elements to make up a particular functional capability are not necessarily in the same physical location within the system. A virtual instrument can be created within a mainframe filled with modules having a variety of basic measurement functions and rearranged to work together through software to create complex measurement functions.

The VXI bus standard offers a wide range of capabilities and communication protocols for manufacturers. However, these are invisible to the test system user. The user simply sets the logical address of each module and plugs it into the mainframe.

Summary

The VXIbus standard heralds a new type of instrumentation that gives users the flexibility to optimize a test and measurement system by selecting modules from a variety of manufacturers, knowing that they will all work together. Distinct advantages over traditional rack-and-stack instrumentation will also be offered, such as high-speed communication between modules, precise triggering, ease of system integration, and size reduction.

Bibliography

1. L. DesJardin, "Configuring a VXIbus System: What Users Need to Know," *ATE & Instrument Conference East 1988*, pp. 503-506.
2. C. Glasow, "A VXI Module Designer's Guide to Mechanical, Power, and EMC Considerations," *ATE & Instrument Conference East 1988*, pp. 491-493.
3. D.A. Haworth, "VXI Electrical Architecture," *ATE & Instrument Conference East 1988*, pp. 491-493.
4. A. Hollister, "A VXIbus Overview," *ATE & Instrument Conference East 1988*, pp. 469-480.
5. L.J. Klahn, Jr., "VXIbus Solidifies as Instrument Standard," *Electronic Engineering Times*, September 12, 1988, pp. T17-T19.
6. W.C. Nowlin, Jr., "VXIbus Device Communication," *ATE & Instrument Conference East 1988*, pp. 1-15 (handout).
7. C.H. Small, "Virtual Instruments," *EDN*, September 1, 1988, pp. 121-128.
8. C. Thomsen, "The Evolution of the VXIbus," *ATE & Instrument Conference East 1988*, pp. 465-468.
9. R. Wolfe, "VXIbus Becomes a Reality," *Evaluation Engineering*, July 1988, pp. 35-40.

VXIbus Product Development Tools

A VXIbus mainframe, a pair of modules, software, and accessories will help manufacturers develop VXIbus modules and systems more easily.

by Kenneth Jessen

TO PROVIDE MANUFACTURERS with tools to develop VXIbus products, Hewlett-Packard has developed a VXIbus C-size mainframe, a Slot 0 module, and VXIbus development software. Other accessories include a breadboard module and a chassis shield. These tools are designed to give the VXIbus user the ability to develop products faster and with reduced resources. The list of HP VXIbus development tools includes:

- C size mainframe
- C size Slot 0 and translator module
- C size register-based breadboard module
- C size carrier for adapting A size or B size modules to the C size mainframe
- C size chassis shield for EMI reduction between modules
- VXIbus development software for use with HP 9000 Series 200 and 300 controllers
- VMEbus interface for HP 9000 Series 300 controllers
- VMEbus preprocessor for HP 1650A and 16500A Logic Analyzers.

Selection of the C Size Module

HP and many other manufacturers have selected the C size module (9.187 by 13.386 inches with P1 and P2 connectors) as their primary module size for instrumentation. For HP, this choice was influenced by analysis of many existing HP modular systems, including the following types: data acquisition, electronic switching, logic analysis, waveform generation, microprocessor development, and spectrum analysis. All of these products were designed independently over a period of time, and their board sizes and power capacities were selected to optimize economics and performance. It was found that the module size chosen for all of these modular systems was equivalent to C size or smaller. Independent analysis confirmed that the vast majority of instrument functions could be provided within a B size or C size system, and a C size mainframe can support both sizes.

VXIbus Mainframe

The HP C size mainframe (Fig. 1) uses a carefully designed 12-layer printed circuit board for its backplane to provide the best possible noise immunity and signal integrity. This mainframe is designed to support the smaller VMEbus boards (A size and B size) using a hardware adapter kit. An optional chassis shield fits between the card slots to provide additional electromagnetic isolation between modules.

For cooling, two dc fans are used. Their speed is controlled by cooling needs based on a measurement of ambient intake air temperature. Air is delivered through a pressurized plenum to ensure even airflow through each module independent of the number or location of modules in the mainframe. In other words, unused slots need not be blocked off, so easy access is retained during module development. The variable-speed fans dramatically reduce acoustical noise in bench environments while delivering adequate cooling in warmer environments up to 55°C.

VXIbus Slot 0 Module

The HP Slot 0 module is aimed at operation directly from a VMEbus interface or VMEbus computer. This module includes word-serial communication registers, allowing communication between other VXIbus devices and computers such as HP 9000 Series 300 Computers with a VMEbus interface. Many other VMEbus computers will be able to communicate with VXIbus message-based devices using this module as a translator.

The translator consists of a VXIbus register-based device and a VXIbus message-based device. The register-based device is used to monitor and control the message-based device's word-serial communication register. It also provides the required Slot 0 backplane services. Each of these devices is a collection of registers in the VXIbus memory space that can communicate over the VXIbus. The benefit to the user of this structure is that it allows operation of the module as a message-based device from almost any VMEbus device.

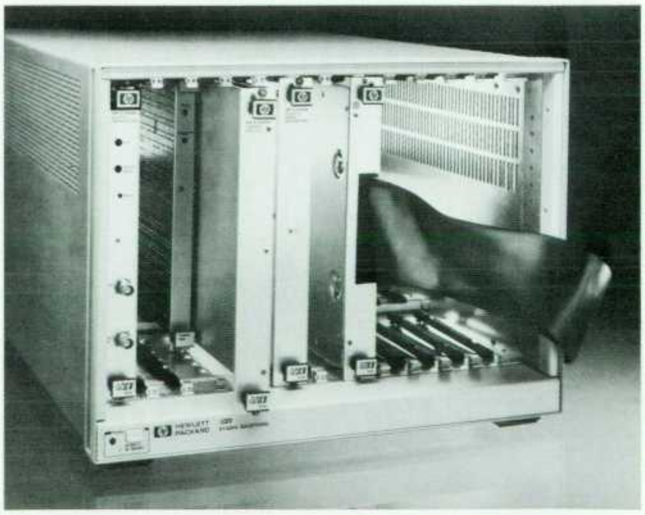


Fig. 1. *HP VXIbus development hardware includes a C size mainframe, a Slot 0 module, a register-based breadboard module, and a carrier module for smaller A size and B size cards.*

Fig. 2. *HP VXIbus development software runs on HP 9000 Series 200 and 300 Computers. It is designed to verify the operation of VXIbus modules and to serve as a learning tool for the VXIbus standard.*

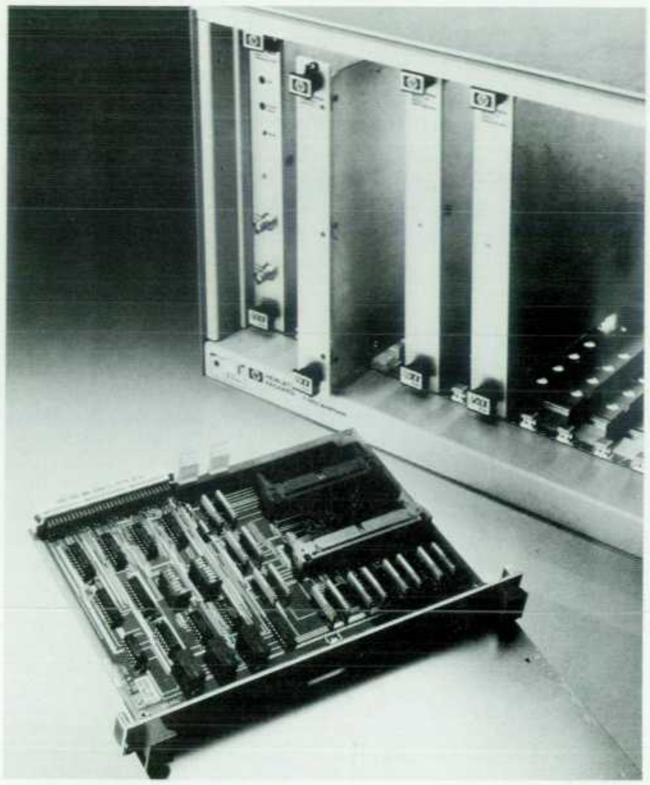


Fig. 3. *The HP 98646A VMEbus Interface for HP 9000 Series 200 and 300 Computers.*

The HP Slot 0 module also supports the synchronous and asynchronous (two-line handshake) trigger protocols. External BNC connectors allow the user to use external triggers from traditional HP-IB (IEEE 488/IEC 625) instrumentation with any of the VXIbus TTL trigger lines.

VXIbus Breadboard Module

HP also offers a C size register-based breadboard module which includes a 16-bit hardware interface to the VXIbus backplane. This module is designed to allow users to construct custom assemblies with minimum effort. This module includes backplane buffering to the VXIbus data lines and address lines. It also supports VXIbus autoconfiguration, bidirectional data transfers, interrupts, system fail, system status, and manufacturer's ID code. Module shielding and ejector hardware are also included.

Software

The HP VXIbus development software runs on HP 9000 Series 200 and 300 Computers (Fig. 2). It is used primarily to verify the operation of a VXIbus module. It is also a learning tool for the VXIbus standard. The program set is a combination of callable subroutines written in BASIC. The code is not protected so that users can modify lines to suit their specific needs.

The development software implements the VXIbus resource manager functions when used in conjunction with the Slot 0 module and the HP 98646A VMEbus Interface (Fig. 3). The resource manager configures the mainframe, assigns memory spaces to each module, reports what modules are in the system, and allows access to their VXIbus capabilities. Access routines allow the user to read and write to any register and implement the word-serial protocol for message-based devices.

The 22 routines in the VXIbus development software package include six for memory devices, six for message-based devices, and six for system configuration.

A VXIbus development system exerciser program allows interactive access to other programs included in this package. It prompts for parameter values and generates a call to a particular subprogram.

Authors

April 1989

6 — Digital Multimeter

Scott D. Stever
Author's biography appears elsewhere in this section.

8 — Integrating A-to-D Converter

Wayne C. Goeke
Author's biography appears elsewhere in this section.

15 — AC Voltage Measurements

Ronald L. Swerlein
Author's biography appears elsewhere in this section.

22 — Multimeter Calibration

Scott D. Stever

Scott Stever was project manager for the HP 3458A Multimeter. He has written for the HP Journal before; in the February 1985 issue, he reported on his work on the dc and ohms measurement sections for the HP 3457A Multimeter. He also is the coinventor of a patented automatic attenuator compensator for this instrument. He joined HP in 1979, after graduating from the Georgia Institute of Technology with a BSEE degree. Analog circuit design has become his main professional interest. Scott was born in Corpus Christi, Texas, is married, and has a six-year-old son. He lives in Loveland, Colorado. When he is not working on or flying his airplane, he likes to ski and play tennis.

Wayne C. Goeke

Design of the analog-to-digital converter for the HP 3458A Multimeter was Wayne Goeke's main responsibility. As a specialist in analog circuit design, he has contributed to the development of a number of instruments, including the HP 3497A Data Acquisition/Control Unit, the HP 3468A Digital Multimeter, and the HP 3478A Multimeter. Two pending patents on the HP 3458A are based on his ideas. Wayne studied at the University of Wisconsin at Madison, where he received both his BSEE (1977) and his MSEE (1979) degrees. He was born in Freeport, Illinois, is married, and lives in Fort Collins, Colorado. In his spare time, he is active in his church and enjoys volleyball, skiing, and backpacking.

Stephen B. Venzke

Since Steve Venzke joined HP in 1965, he has been associated with a wide range of products, including the HP 3455A Digital Voltmeter, the HP 3585A Spectrum Analyzer, and the HP 3577A Network Analyzer. He designed the input amplifier, ohms converter, current shunts, and autocalibration circuits for the HP 3458A Multimeter and has now become a production engineer for the product. He is the originator of two patents, one for a low-frequency current divider for function generators, the other for an analog-to-digital converter technique. Steve received his BSEE degree from the University of Minnesota (1965), and his MSEE degree from Colorado State University (1968). Born in Minneapolis, Minnesota, he is married and has two daughters. He lives in Loveland, Colorado, where he teaches eighth-grade Sunday school as an avocation. He is also interested in audio equipment and enjoys backpacking and fishing.

Ronald L. Swerlein

The design of the ac front end, the digital ac algorithms, the transformer, and the power supplies were Ron Swerlein's major contributions to the development of the HP 3458A Multimeter. His analog circuitry was also used in earlier instruments, like the HP 3468A Digital Multimeter and the HP 3478A and HP 3457A Multimeters. He is named inventor in a patent describing ac autocalibration circuits, and he has published two papers about digital ac measurement. His bachelor's degree in engineering physics (1978) and his MSEE degree (1979) are from Ohio State University. Ron was born in Toledo, Ohio, is married, and lives in Longmont, Colorado. He spends his leisure time reading science fiction.

31 — High-Throughput Design

Gary A. Ceely

Shortly after graduating from Virginia Polytechnic University with a BSEE degree in 1979, Gary Ceely joined HP's Loveland Instrument Division. His development projects have included firmware design for the HP 3456A Systems DVM, the HP 3488A Switch/Control Unit, the HP 3852A Acquisition/Control Unit, and the HP 3054A Data Acquisition System. During development of the HP 3458A Multimeter, Gary's responsibilities included the ERS, firmware architecture, operating system interface, and language parsing. Born in Alexandria, Virginia, he lives in Loveland, Colorado. His favorite leisure activities are skiing, waterskiing, kayaking, volleyball, tennis, and aerobics.

David J. Rustici

The measurement and calibration firmware of the HP 3458A Multimeter were David Rustici's focal responsibilities at HP's Loveland Instrument Division. He is now a firmware manager. Originally, he had joined the Civil Engineering Division in 1976, after graduating from the University of Wisconsin at Madison with a BSEE degree. His responsibilities there included the HP 3808A and HP 3850A Distance Meters. His work on the latter resulted in a patent, and another patent for the HP 3458 is pending. David is the coauthor of a paper describing firmware for an intelligent digital voltmeter. He was born in Racine, Wisconsin, is married, and has a son and a daughter. He lives in Loveland, Colorado. In his off-hours, he enjoys skiing, bicycling, golf, and travel.

39 — High-Resolution Digitizing

David A. Czenkusch

As R&D engineer in the voltmeter laboratory of the Loveland Instrument Division, Dave Czenkusch has been involved in a number of voltmeter and multimeter developments, including the HP 3458A, and a digital voltmeter for the HP 3235 Switch/Test Unit. One of his ideas for the HP 3458A resulted in a patent application. Dave's BSEE degree is from Purdue University (1983). He joined HP the year of his graduation and developed a professional interest in digital design and firmware. He was born in Speedway, Indiana, is single, and lives in Loveland, Colorado. He enjoys skiing in the winter and looking forward to the skiing season and bowling in the summer.

50 — Structured Defect Analysis

Takeshi Nakajo

Software defect analysis and the effects of human error on system quality are Takeshi Nakajo's main professional interests. He has written a number of papers on the subject of obviating program faults and flaws in development and manufacturing. Takeshi graduated from Tokyo University with a master's degree in engineering (1981) and received a PhD degree in 1986 from the same institution. He has been a research assistant at the University since 1987.

Katsuhiko Sasabuchi

As a quality manager, Katsuhiko Sasabuchi works in the product assurance department of Yokogawa Hewlett-Packard in Hachioji, near Tokyo. He is closely associated with a special program to improve software productivity and quality. In the study of software defect analysis described in this issue of the HP Journal, he functioned as one of the principal contacts with the Kume Laboratory of Tokyo University. Katsuhiko was born on Hokkaido, and his bachelor's degree in applied physics is from the University of Hokkaido. He joined YHP in 1973. He is married, has two sons, and lives in Hachioji. His favorite pastimes are playing baseball and reading historical novels.

Tadashi Akiyama

Tadashi Akiyama is a software engineer at Yokogawa Hewlett-Packard. He is responsible for quality assurance for new software projects, and his current interests focus on techniques to improve quality and productivity in software development. He attended the Science University of Tokyo, graduating with a degree in mathematics. Tadashi is a native of Yokohama and is married. His 2-year-old son and infant daughter monopolize most of his spare time.

57 — Dissecting Software Failures

Robert B. Grady

Software development and project management using software metrics have been the focal professional interests for much of Bob Grady's 19-year career at HP. He has been manager of a variety of major projects, including the HP Atlas Compilation System, the HP 2240A Measurement and Control Processor hardware, the HP 12050A Fiber Optic HP-IB Link, manufacturing and information systems, and HP's software engineering laboratory. Presently, he is section manager in the software methods laboratory of HP's Data and Languages Division. Bob is a member of the IEEE Computer Society, is coauthor of a book on software metrics, and has written and coauthored numerous papers and articles on software subjects, including several for the HP Journal. A native of Chicago, Illinois, he received his BSEE degree from the Massachusetts Institute of Technology (1965) and his MSEE degree from Stanford University (1969). He and his wife, who is a section manager at HP's Data Products Operation, have a daughter and a son and live in Los Altos, California. In addition to managing his son's basketball team, Bob plays basketball himself and enjoys hiking, camping, and skiing.

64 — Complexity Metric

William T. Ward

With software engineering methodologies and tools his main professional interest, Jack Ward has written a number of articles on the subject of software quality. He is software quality engineering manager at HP's Waltham Division (Massachusetts), and heads a group responsible for testing all software/firmware. As a software quality engineer in earlier assignments, he was involved in testing ECG arrhythmia monitoring systems. Before joining HP in 1982, he was a software support engineer for Data General Corporation. Jack's BS degree in linguistics is from the University of Illinois (1972), and his MS degree in computer science is from Boston University (1984). A native of Winona, Mississippi, he is married and has two children. He lives in Brookline, Massachusetts. In his spare time, Jack teaches computer science courses at Boston University. He also likes jogging.

69 — Object-Oriented Unit Testing

Steven P. Fiedler

The object-oriented unit testing discussed in Steve Fiedler's article forms part of his responsibilities as a software quality engineer. In this particular study, he implemented processes for assuring software quality in clinical information systems. Before transferring to the Waltham Division of HP, he was a systems support engineer for computer products and networks in the Valley Forge, Pennsylvania, sales office. He came to HP in a part-time position in 1979, then joined full-time in 1981, after receiving his BS degree in computer science from West Chester University. Steve is a member of the ACM. He was born in Milwaukee, Wisconsin, is married, and has two children. He resides in Leominster, Massachusetts. He pursues musical interests in his church and, with his wife, often sings at weddings and small gatherings. He also enjoys skiing, hiking, and travel.

75 — Reliability Growth Models

Gregory A. Kruger

In the productivity section of HP's Lake Stevens Instrument Division, statistician Greg Kruger's main responsibilities include R&D metrics, software reliability modeling, and process analysis. When he first joined HP at the Loveland Instrument Division almost eight years ago, Greg's assignment included implementing statistical quality control practices in manufacturing and training people to use and understand them. Later he was commissioned to spread Total Quality Control practices throughout the Lake Stevens Instrument Division. Greg was born in Waterloo, Iowa. His BS in mathematics/statistics (1979) and MS in statistics (1981) are both from Iowa State University. He is married, has two children, and offers much of his leisure time to his duties as deacon of his church. Greg is an avid archer, serves on the board of directors of the Washington State Bowhunters, and edits a newsletter on the subject. Vocal music is another of his interests.

80 Comparing Methodologies

William A. Fischer, Jr.

Bill Fischer is the coauthor of the comparative study of structured methodologies in this issue of the HP Journal. He is an R&D section manager in charge of Intel and Motorola emulation products. Previously, he has been an R&D project manager, a technical support engineer, and a product marketing engineer. In the ten years before he joined HP in 1984, Bill was an engineer at the Hamilton Standard Division of United Technologies, designing automated test equipment for the space shuttle. He holds a BSEE degree (1973), an MSEE degree (1978), and a master's degree in management (1980), all from Rensselaer Polytechnic Institute. He has authored and coauthored a number of papers and articles, mainly about software project management. Bill was born in Attleboro, Massachusetts, and lives with his wife and four children in Colorado Springs, Colorado. He enjoys running and playing basketball.

James W. Jost

As a statistician at HP's Logic Systems Division, Jim Jost is responsible for software metrics and process improvements. He collaborated on the comparative study of structured methodologies discussed in this issue of the HP Journal. Before he joined HP in 1984, he spent eight years doing breeder nuclear reactor research, particularly in the field of fuels. Jim holds a BA degree in chemistry from Tabor College (1970) and an MS degree in statistics from Oregon State University (1976). He was born in Hillsboro, Kansas, and lives in Colorado Springs, Colorado. He is married and has three children. His favorite off-hours activities are basketball, skiing, and reading biographies.

86 Object-Oriented Systems Analysis

Barry D. Kurtz

Barry Kurtz authored the OSA methodology and acted as a technical consultant. Among other software projects he has handled since he joined HP in 1976 were design of CAE tools for electrical schematic capture and printed circuit design and, as an R&D engineer and project manager, design of the Materials Management/3000 software for HP 3000 Business Computers. Barry's BS and MS degrees in computer science are from Brigham Young University (1987, 1988), and he is a member of both the ACM and the IEEE. He was born in Richmond, Indiana, and lives in Boise, Idaho. He's married and has a son. He is active in his church and enjoys family outings, fishing, and amateur radio.

Teresa A. Wall

Born in Norman, Oklahoma, Teresa Wall graduated with a BS degree in computer science from the University of Oklahoma in 1983. She joined the Fort Collins Systems Division of HP the same year. Among the projects she has been working on are command groups and HP-UX systems integration for the HP 9000 Series 300 and 500 Computers. Her contributions to the OSA development focused on the analysis methodology and toolset. Teresa's main professional interests are analysis and design methodologies and object-oriented languages. She lives in Santa Clara, California.

Donna Ho

Donna Ho is a software development engineer at HP's Software Engineering Systems Division. On the OSA project, she was responsible for the analysis methodology, tool prototype, and documentation. When she came to HP in 1985, she joined Corporate Engineering to work on software development. Previous professional experience includes work on DG-UX command groups at Data General. Donna attended Duke University, graduating in 1985 with a BS degree in computer science/psychology. She was born in Honolulu, Hawaii, and lives in Santa Clara, California.

91 VXIbus Interconnection Standard

Kenneth Jessen

Author's biography appears elsewhere in this section.

96 VXIbus Tools

Kenneth Jessen

Ken Jessen is a manufacturing development engineer associated with new products and the development of new processes for HP's Manufacturing Test Division. He joined HP in 1965 and, among various assignments, has held positions as service manager and distribution manager. During the past nineteen years, he has written many technical articles describing HP products for a variety of trade journals, some of them translated into foreign languages for publication abroad. He has published four books on Colorado history and contributed to two technical books. Ken's BSEE degree (1962) and MBA degree (1964) are both from the University of Utah. He was born in Evanston, Illinois, is married, and has three children. He lives in Loveland, Colorado. His hobbies are writing, hiking, skiing, and railroad history.

Hewlett-Packard Company, 3200 Hillview Avenue, Palo Alto, California 94304

ADDRESS CORRECTION REQUESTED

Bulk Rate
U.S. Postage
Paid
Hewlett-Packard
Company

HEWLETT-PACKARD JOURNAL

April 1989 Volume 40 • Number 2

Technical Information from the Laboratories of Hewlett-Packard Company

Hewlett-Packard Company, 3200 Hillview Avenue
Palo Alto, California 94304 U.S.A.
Hewlett-Packard Central Mailing Department
P.O. Box 529, Startbaan 16
1180 AM Amstelveen, The Netherlands
Yokogawa-Hewlett-Packard Ltd., Suginami-Ku Tokyo 168 Japan
Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive, Mississauga, Ontario L4V 1M8 Canada

```
00055601                  HPJ 12/88
C BLACKBURN
JOHNS HOPKINS UNIV
APPLIED PHYSICS LAB
JOHNS HOPKINS RD
LAUREL, MD                  20707
```

CHANGE OF ADDRESS: To subscribe, change your address, or delete your name from our mailing list, send your request to Hewlett-Packard Journal, 3200 Hillview Avenue, Palo Alto, CA 94304 U.S.A. Include your old address label, if any. Allow 60 days.

5953-8574