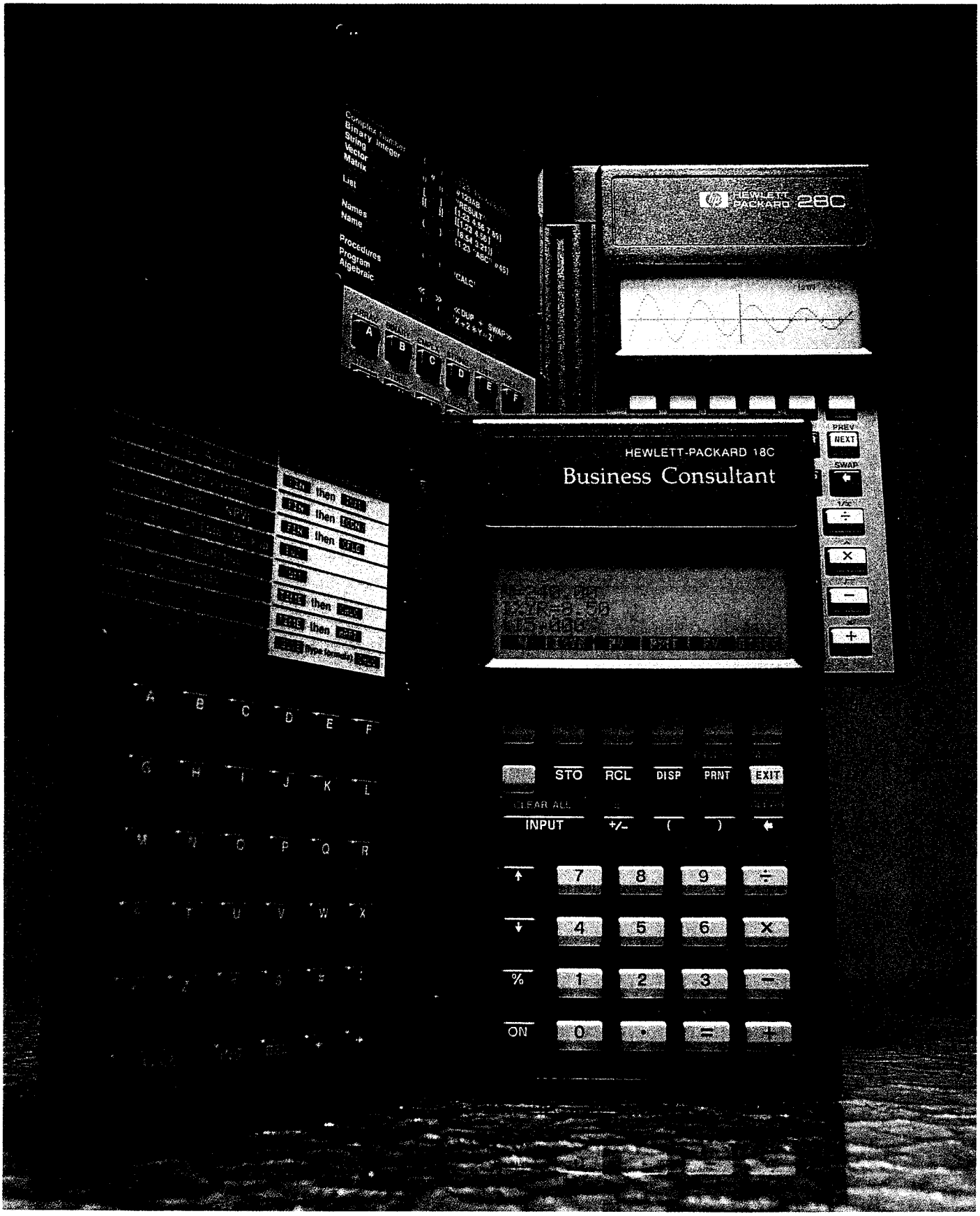
HEWLETT-PACKARD
JOURNAL

AUGUST 1987



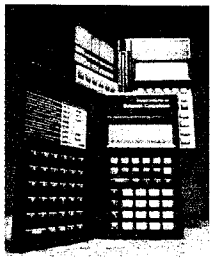
HEWLETT-PACKARD JOURNAL

August 1987 Volume 38 • Number 8

Articles

Departments

In this Issue



Handheld calculator designs always seem to require the expert application of a broader variety of disciplines than other product designs. Perhaps it's the challenge of building more and more capability into a severely limited volume. In this respect, this month's subjects, the HP-18C Business Consultant and the HP-28C Scientific Professional Calculator, are typical. Their design story has interesting aspects not only in circuit design and firmware development, but also in materials, packaging, operating system design, algorithms, user interface design, display technology, and ergonomics.

The first thing you notice is the package. Users didn't like keys that had three different labels, so the designers added more keys, and the package opens to reveal two keyboards side by side. How do you reliably connect the two halves of the electronics through a rotating hinge? See the mechanical design paper on page 17. Business users want to customize their calculators without programming, so the HP-18C has softkey menus and the Solve interface, which lets you type in an equation algebraically using any convenient variable names and then solve for an unknown variable by pressing the softkey labeled with its name (page 4). While the HP-18C is algebraic, the HP-28C still uses RPN, the programming language of earlier HP scientific calculators, but the language has been extended to include symbolic entry of variables and a variety of data types (page 11). Behind these advanced features of both calculators is a new operating system developed especially for handheld calculators. Called RPL, it has similarities to both Lisp and Forth (page 21). For solving equations, the iterative solver of earlier HP scientific calculators has been augmented with a direct solver and implemented in both the HP-18C and the HP-28C. The solver first tries to solve a user's equation by algebraic operations. If it can't, it uses trial-and-error methods (page 30).

You'll find the electronic design of these new calculators described in the papers on pages 25 and 34. In a future issue, we'll have papers on the calculators' accessory printer and its infrared interface, and on the manufacturing process.

-R. P. Dolan

What's Ahead

Next month's issue is another in our series devoted to the new HP Precision Architecture. Three papers will describe the design and development of the processor chip set for the first VLSI implementation of the architecture, and two papers will describe system processing units that use this chip set. One of the SPUs is for the HP 9000 Model 850 and HP 3000 Series 950 Computers, and the other is for the HP 9000 Model 825.

The HP Journal encourages technical discussion of the topics presented in recent articles and will publish letters expected to be of interest to our readers. Letters must be brief and are subject to editing. Letters should be addressed to: Editor, Hewlett-Packard Journal, 3200 Hillview Avenue, Palo Alto, CA 94304, U.S.A.

A Handheld Business Consultant

The latest model in HP's line of calculators designed for business and financial applications features a menu-driven user interface for selecting any of its many built-in functions or custom equations entered by the user.

by Susan L. Wechsler

HP'S BUSINESS CONSULTANT (Fig. 1) is an advanced handheld calculator that combines many of the most popular features of the earlier HP-12C with enhancements such as a menu-driven user interface, customization without programming, a four-line dot-matrix display, and an infrared transmitter for sending data to an optional cordless printer. Because the Business Consultant uses the same CPU as the HP-71B Handheld Computer,[1] its financial calculations run at least 15 times faster than those on the HP-12C.

The major applications (menus) contained in the HP-18C Business Consultant are (see Fig. 2):

- FIN—time value of money, cash flow analysis, interest conversions
- BUS—percent change, percent total, markup
- SUM—running total, one-variable statistics, forecasting with one of four models
- TIME—date arithmetic, running clock with the ability to set up to six alarms
- SOLVE—new way for users to solve their own special problems without programming.

What makes this product special is its ease of use. The popularity of the HP-12C told us that its feature set met customer needs, and yet we were confident that there were ways we could improve the usability of those features. To discover how, we contacted our customers through focus panels on both coasts of North America, and our contingent of sales representatives overseas. The response guided many aspects of the design of the business consultant.

Localization

From outside the United States came significant feedback regarding localization. Many people wanted a calculator that communicates in their primary (or perhaps only) language. To do this, every message and softkey label was put into a single table, thus eliminating the possibility of overlooking a message during the translation process. The idea of a single table, as opposed to strings scattered throughout ROM space, appealed to us for financial reasons also. To expedite release of the product, the HP-18C's operating system was initially stored in two 32K-byte ROM chips. It was highly desirable to be able to accomplish localization

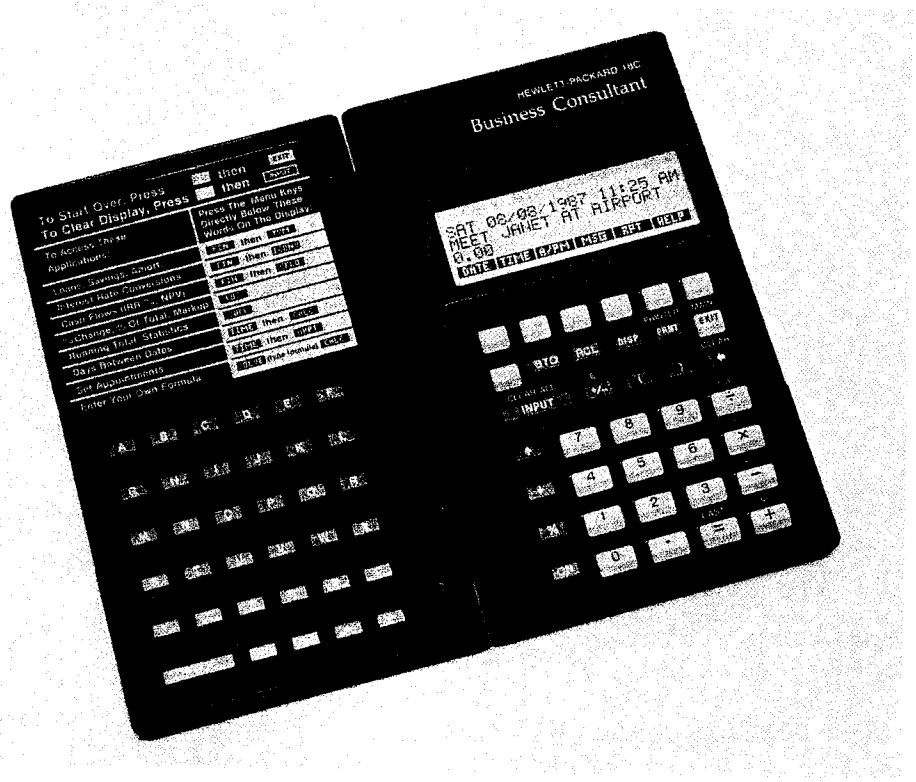


Fig. 1. *The HP-18C Business Consultant is HP's latest handheld calculator designed for business and financial applications. It features several built-in applications accessed by a menu-driven user interface, an equation solver, algebriac entry of formulas without the need for programming, and an infrared transmitter for sending data to an optional printer.*

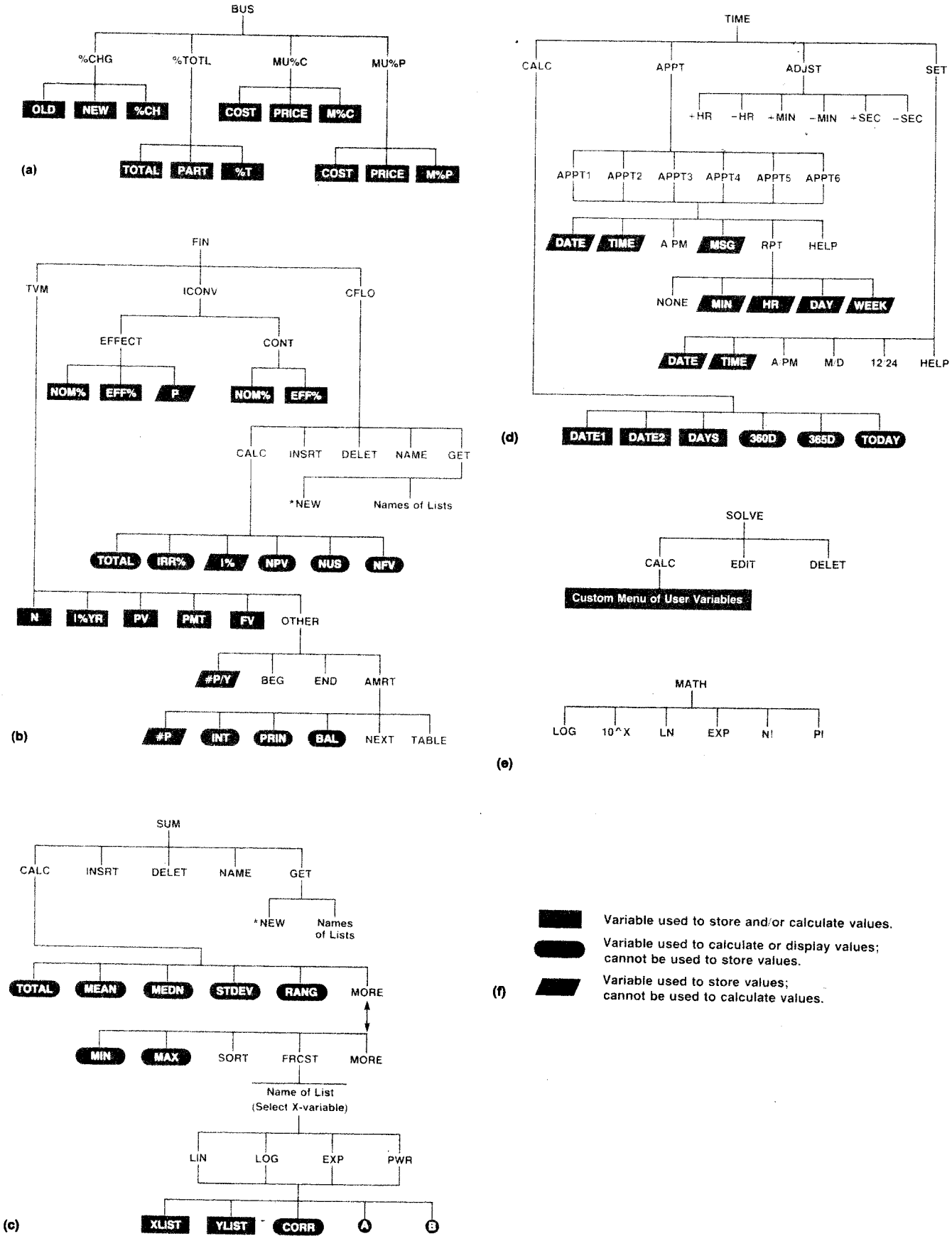


Fig. 2. *Built-in application menu structures for HP-18C.*

by replacing only one of these two chips. By ensuring that the message table did not cross a chip boundary, we were able to meet this goal. So far, the Business Consultant has been localized in German, French, Italian, and Spanish.

Softkey Menus

Many people in our target market disliked keyboards cluttered with labels. We did not want to sacrifice functionality to address this concern, so instead we added more keys, both physical and virtual. The physical keys were added by providing a second keyboard, using a clamshell package design. The virtual keys were added by incorporating a menu-driven interface, using six softkeys that are positioned directly beneath the display. There are no labels on these six keys. Instead, their functionality is indicated by the labels shown directly above them. For example, to get to the percent change application, the user presses the softkey labeled BUS (Fig. 2a), which brings up a choice of four menus: %CHG, %TOTL, MU%C, and MU%P. Pressing the softkey labeled %CHG puts the user into the percent change menu.

The Business Consultant is Hewlett-Packard's first menu-driven calculator. The menu scheme was not free—it came at a cost of approximately 5% of the ROM space. A menu table and corresponding menu handler had to be constructed to handle the changing execution address and ASCII string bound to each softkey, and to deal with the idiosyncrasies of each menu.

People are emphatic about wanting a calculator that they can learn to use by merely pressing the keys. They want an operating environment that is intuitive and consistent. So, the Business Consultant provides help messages to guide the user through the various applications, and within any application, the same interface produces answers quickly and simply. This generalized interface succeeds in providing one consistent method for solving problems throughout the machine. It is the same as the top-row-key interface (Fig. 3) used to solve time-value-of-money problems on the HP-38C and HP-12C. On these earlier HP financial calculators, the **n**, **i**, **PV**, **PMT**, and **FV** keys provided a great "what-if?" tool for time-value-of-money problems such as loans, savings, and leasing. To store a value into the number-of-periods register, the user keys in the desired value and presses the **n** key. After storing values into four of the five variables, the user simply presses the key corresponding to the unknown variable to solve for its value. In this fashion, any variable can be derived after values are assigned to the other four variables.

Through the use of the six softkeys positioned directly beneath the HP-18C's display, the Business Consultant user can bring up various built-in application menus that make use of the same top-row-key interface. When a given application is in effect, its variable names come up in the display directly above the associated softkeys. We call this generalized top-row-key interface the Solve interface. Built-in applications that use the Solve interface are listed in Table I, along with their associated softkey labels.

Table I

HP-18C Applications Using Solve Interface

Application	Softkey Labels				
Time value of money:	N	I%YR	PV	PMT	FV
Interest rate conversions:	NOM%	EFF%	P		
Percent change:	OLD	NEW	%CH		
Percent of total:	TOTAL	PART	%T		
Markup as percent of cost:	COST	PRICE	M%C		
Markup as percent of price:	COST	PRICE	M%P		
Days between two dates:	DATE1	DATE2	DAYS		

Using this standardized interface, functions that traditionally have been confusing to use on previous calculators become extremely intuitive. Two such functions are percent change and percent of total. To determine what percentage 17.5 is of 67, press BUS and then press %TOTL. The display shown in Fig. 4a appears. Then, pressing keys **6 7** TOTAL **17.5** PART %T results in the display shown in Fig. 4b.

As can be seen from Fig. 4b, the Solve interface has been further enhanced by adding the labeling of values. When a value is stored in a variable, a confirmation consisting

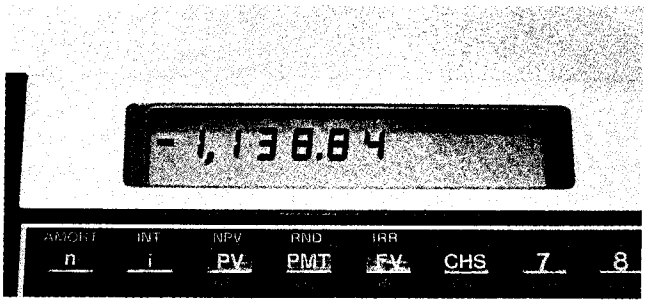


Fig. 3. *Top-row-key interface for solving time-value-of-money problems on HP-12C Calculator. On the HP-18C, the top-row-key interface has become a more generalized softkey structure where the key labels are displayed on the bottom of the display above the keys.*

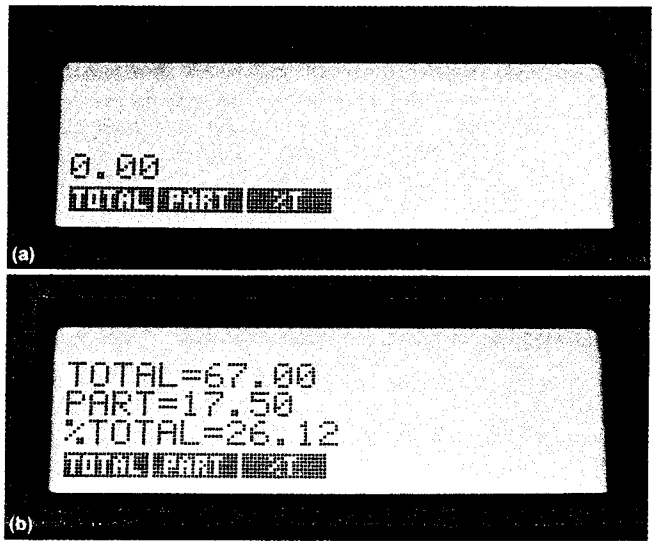


Fig. 4. *Use of the HP-18C's softkey user interface. (a) %TOTL menu of BUS application. (b) Answer to determining what percentage of 67 a value of 17.5 is.*

Cash Flow Analysis Using the HP-18C

An investor has an opportunity to purchase a piece of property for $100,000. Yearly cash flows are anticipated as indicated in Table I, and the investor expects to be able to sell the property for $120,000 in 10 years. The investor would like an 11.5% return.

Table I

Year	Cash Flow	Year	Cash Flow
1	$15,000	6	$10,000
2	12,000	7	9,500
3	12,000	8	9,500
4	12,000	9	9,500
5	10,000	10	120,000

Press the FIN softkey on the HP-18C to access its financial application menu and then press the CFLO softkey to select cash flow analysis. Then press keys **1 0 0 0 0 0 +/− INPUT**. The screen shown in Fig. 1a appears, prompting for the first flow in the series. A prompt is also given for the number of times a particular flow occurs, simplifying grouped flow entry.

Pressing keys **1 5 0 0 0 INPUT** gives the display shown in Fig. 1b. Note that the ▶ symbol has migrated down to the TIMES prompt. Note also the 1.00 in the calculator line. This value is put in the calculator line whenever a flow is entered, so that if a flow occurs once, only the **INPUT** key must be pressed to enter the number of occurrences. This feature was added so that the interface for simple cash flow problems wouldn't pay a penalty for the ease of use introduced for grouped cash flow problems.

Pressing **INPUT** causes the screen shown in Fig. 1c to appear. To finish entering the data, press:

1 2 0 0 0 INPUT 3 INPUT
1 0 0 0 0 INPUT 2 INPUT
9 5 0 0 INPUT 3 INPUT
1 2 0 0 0 0 INPUT INPUT

To calculate net present value and internal rate of return, press the softkey labeled CALC. The screen shown in Fig. 1d appears. To input the desired rate of return, press **1 1 . 5** I%. To get the net present value, press NPV and the HP-18C displays NPV = 2,914.83. To calculate the internal rate of return, press IRR%, obtaining a displayed result of IRR% = 12.01.

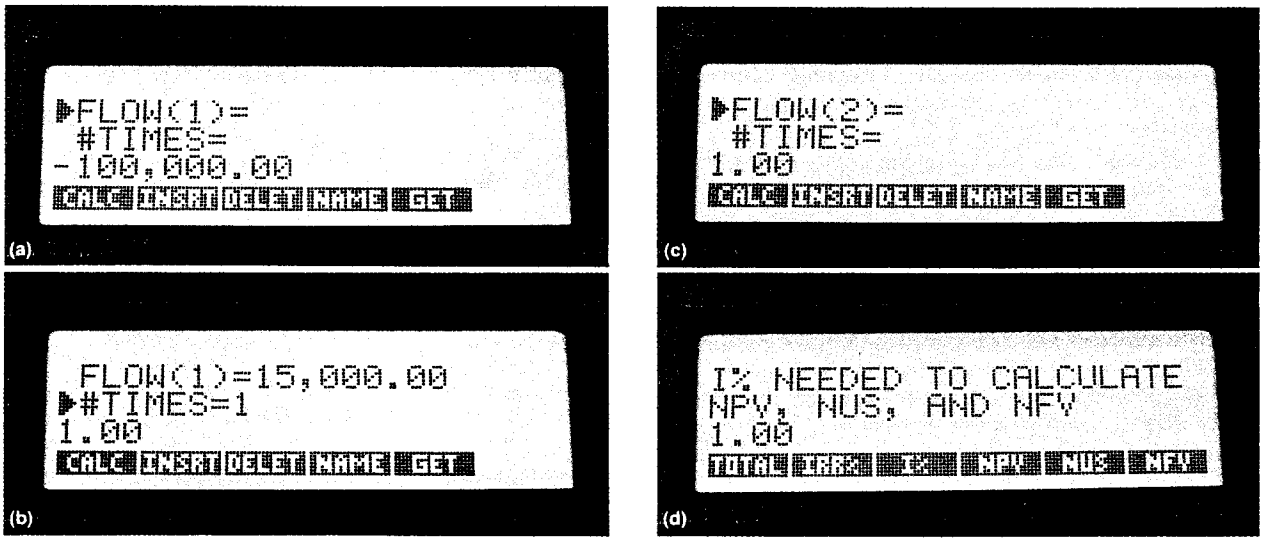


Fig. 1. *HP-18C displays for cash flow analysis example. (a) Prompt display for first cash flow entry. (b) After first entry of 15,000 is input, the display prompts for the number of times entry occurs. A value of 1.00 is displayed as a default value. (c) Pressing the* **INPUT** *key prompts for the next entry. (d) After completing the cash flow entries, pressing the* CALC *softkey gives this display. Entering the desired interest value and pressing the* I% *key then allows calculation of the net present value, net uniform series, or net future value.*

of the variable name and its value is shown in the display. In the percent change application, pressing **1 0** OLD **1 5** NEW %CH results in the display shown in Fig. 5. Whenever a variable is recalled, stored, or solved for, a confirmation is given. The Business Consultant maintains a history stack of the last four such confirmations given. Up to three can be viewed at a time; the fourth is easily accessed via the scrolling keys ↑ and ↓. The idea of labeling results is special to the Business Consultant.

Data entry for cash flow analysis, running total, and statistics is simplified by conceptualizing this data as number lists. The user is prompted for each item in the list. Using the scroll keys, the user can move up and down through the list for reviewing or editing, and with a single keystroke, items in the list can be inserted or deleted. Because the list can be named, several lists can exist in memory at a time (the exact number is limited only by available memory). The example in the box above illustrates the simplicity with which data can be entered for cash flow analysis.

The Equation Solver Menu in the HP-18C

The user programming language of the HP-18C Business Consultant is equations. The user types in an equation using variable names of the user's choice, traditional algebraic operators, and any of the HP-18C's built-in set of advanced mathematical functions and conditional expressions. Several equations can be entered in the HP-18C, the number limited only by available memory. A name can be typed at the beginning of each equation to identify it for future recall. An equation is selected from the stored list by moving a display pointer up and down the list of equations. When the pointer points to the desired equation, pressing the CALC menu key causes the HP-18C to interpret the equation and bring up the variable names as softkey labels at the bottom of the HP-18C's display. The associated softkeys, or menu keys, below the display are used to store and calculate solutions using the relationships in the equation. The user enters values for all but one of the variables and the HP-18C solves for the unknown variable.

The programming characteristics of the equation solver are enhanced by 26 advanced functions and conditional expressions that can be used in formulating an equation. While trigonometric functions are not provided, natural and base-10 logarithms, factorial, absolute value, minimum, maximum, pi, integer part, fractional part, rounding and truncation, modulo, sign, and square root are available. Another six functions specific to finance mathematics are available as are date and delta-days functions.

As a simple example, consider a formula that expresses the economics of performing a tune-up on an automobile engine:

COST × MPGBEFORE × MPGAFTER ÷ (MPGAFTER − MPGBEFORE) ÷ PRICEGAS = BEMILES

The equation includes five variables: cost of the tune-up, miles per gallon before and after the tune-up, price per gallon of gasoline, and break-even miles—the number of miles at which the cost of the tune-up is recovered by the benefit of the reduced gasoline consumption.

A user "programs" the HP-18C to solve the above equation by pressing the SOLVE menu key, typing in the equation (Fig., 1a), and then pressing the CALC menu key. When the CALC menu key is pressed, the keys are customized to the above equation. The variable-width character font for the softkey labels allows up to five characters of the variable name to show as a label. In this case, the labels are COST, MPGB, MPGA, PRICE, and BEMI (Fig. 1b). However, when a variable name appears as a result in the other lines of the display, the complete variable name is displayed.

Solving this problem parallels that for solving problems using the built-in functions of the machine. A solution can be calculated for each of the variables in the equation, given values for the other variables. A quick example using the tune-up formula is to key in 28 and press the softkey labeled MPGB, key in 33 and press softkey MPGA, key in 0.839 and press PRICE, and key in 15000 and press BEMI. Then press COST to solve the equation and see displayed COST=68.10, (Fig. 1c), the cost of a tune-up that would pay for itself by improved gasoline mileage for 15,000 miles. If the tune-up cost is $85, key it in, press COST, and then press BEMI to see displayed BEMILES=18,722.29, the number of miles that must be driven to break even on a tune-up costing $85 and improving mileage from 28 to 33 miles per gallon.

Other equations can be typed in just as easily. Each additional formula is added to the formula list in continuous memory. You can see and select each formula by using the scroll-up and scroll-down keys ↑ and ↓. The RAM in the HP-18C is sufficient to store about ten equations of the length and number of variables illustrated by the tune-up example above.

Direct Solution

The advantage of the HP-18C's equation solver as a programming language is evident—one equation with n variables does the work of n traditional programs written to solve for a single variable as a function of the n − 1 other variables. The equation solver solves directly for any variable that meets all of the following conditions:

- Appears only once in the equation
- Does not appear as an exponent
- Involves only the operators for addition, subtraction, multiplication, division, and exponentiation
- The only functions, if any, in which the variable appears are seven specifically identified functions such as logarithm, inverse logarithm, and square root.

These conditions are met for COST, PRICEGAS, and BEMILES in the example above.

Iterative Solution

The variables MPGBEFORE and MBGAFTER in the example do not meet the above conditions for direct solution. The solution

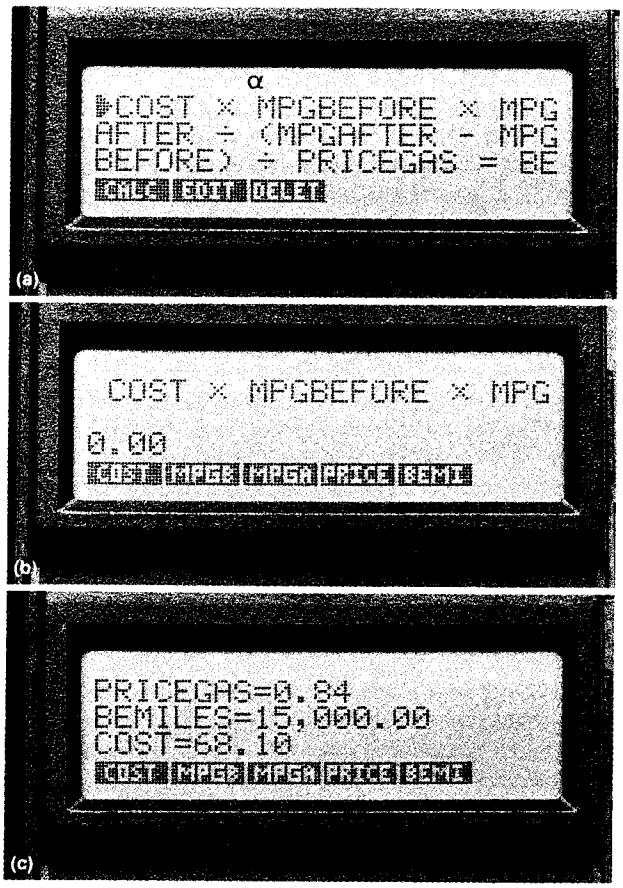


Fig. 1. *Displays during solution of tune-up cost study. (a) Entry of tune-up cost equation. (b) Pressing CALC key assigns labels to softkeys as shown. (c) Break-even cost for 15,000 miles with mileage improved from 28 to 33 mpg.*

for either of these variables in the equation solver uses an iterative search process. The process systematically varies the value of the subject variable until the value of the left side of the equation equals the value of the right side of the equation. While this search is taking place, the value used for each iteration is displayed to give the user a sense of the progress toward a solution. The user can start a search using one or two estimates of the solution; otherwise, default values are used.

Because the iterative solution is numerical rather than analytic, and because an arbitrary variable in an arbitrary equation may have one solution, more than one, or no solution, the *HP-18C Business Consultant Owner's Manual* describes some anomalies that the user might encounter, as well as procedures for learning values that might indicate minimum, maximum, or undefined points in the equation.

Conventions

There are a few conventions that the user must learn to type in general algebraic equations. For example, there are no implied operators (Z = 3Y must be typed Z = 3×Y), and there are no subscripts or superscripts (Y cubed must be typed Y^3).

Paul Swadener
Development Engineer
Handheld Computer and Calculator Operation

SOLVE Application

We were told that most users of financial and business calculators do not want to be bothered with programming in the traditional sense. But at the same time, there was no consensus by any focus panel as to desired functions. Clearly some form of customization was called for, but not in the guise of programming. The SOLVE application addresses this need. It incorporates the same unknown-variable solution concept that was generalized for the built-in applications, but extends it a step farther. The user types in an equation that describes a particular problem. Several variables can be used (the number is limited only by available memory), and each variable name can have up to 10 characters. At the press of a key, the equation is interpreted, and the variable names are extracted and used to label the softkeys. Here, as elsewhere in the machine, when variable names are assigned to the softkeys, the same Solve interface is in effect. The user keys in values for all but one of the variables, then presses the key corresponding to the unknown variable to solve for it. This ability to enter equations and then solve for different variables is known as the Equation Solver, a feature new to handheld calculators. (See box on page 8 for more details and an example of the use of the Equation Solver.)

Of all the decisions made by the design team regarding the user interface, the one that was by far the most difficult (as well as the most controversial) was the one that made the HP-18C operation algebraic, rather than RPN. But, a thorough survey told us that an algebraic HP financial calculator would appeal to new users, and algebraic notation is consistent with the Equation Solver interface.

Print Option

We repeatedly heard that a printer would be a welcome peripheral for our business/financial calculators. So, available as an option is the HP 82240A Infrared Printer that receives data from the Business Consultant via an infrared beam, thus eliminating the need for wires between calculator and printer. All variables and data associated with a particular application can be printed out, whether it be an amortization schedule or the variable values associated with a user-input equation. In TRACE mode, every keystroke is printed to provide a complete record of what the user has done.

RPL

The Business Consultant is one of the first HP calculators (the HP-28C being the other) that has a major portion of its operating system written in a high-level language—RPL. This assisted us in reaching our ease-of-use goal. It allowed us to prototype user interface changes quickly in response to focus panel feedback and to test the newly implemented modifications. Ironically, programming in a high-level language also provided us with one of our major implementation challenges—minimizing response time. In many cases, this meant careful review of RPL code to see where the code could be optimized. In critical areas it meant rewriting some sections in assembly code.

Throughout the design of the Business Consultant, we were confronted with the delicate balance between advanced functionality and ease of use. We devised ways to provide the functionality without sacrificing the product's short learning curve. Some of the techniques we use to accomplish this are displaying numerous help messages, asking for confirmation when attempts are made to clear significant amounts of data, using the top-row-key interface and history stack throughout the machine's many applications, and providing an easy mechanism for entering and modifying data.

Acknowledgments

I would like to acknowledge the rest of the software development team. Stan Blascow wrote all the math routines, with the exception of the solver, Pam Raby was responsible for the time application, Bruce Stephens developed the low-level code that interfaces directly to the hardware, such as the display driver, printer, and self-test software, Paul McClellan wrote the numeric solver, and Charlie Patton, Bob Miller, Gabe Eisenstein, and Laurence

Fig. 5. *Answer to percent change problem given in text.*

History and Inspiration of the Solve Interface

The equation solver concept and interface for the HP-18C and HP-28C Calculators were developed one evening after midnight in an effort to avoid the boredom of debugging the HP-71 Handheld Computer's Circuit Analysis Pac.

I was attempting to solve an equation by hand to determine whether the results of the Pac were correct. After three attempts and three different results with the same equation, I decided another approach was needed. Using the HP-71 and its Math Pac, I wrote a small function to represent the equation and solved for the equation's roots using the FNROOT function. At first, this seemed to be as difficult as using the pencil-and-paper approach. The variable in question needed to be isolated and pointed at, and the function needed to be in a specific form. Although I had used this feature before, it was necessary to read the manual again to remember how to do it. But once the function was completed, the roots were found quickly.

Once the mechanics were understood, it seemed simple to repeat the process with other equations. While experimenting with this structure, it became clear that the ability to select any variable easily would be very useful. Then the same equation could be solved readily for any of its variables.

Paul Swadener implemented a version that would accomplish this on several HP calculators. The variable was usually selected by specifying a number corresponding to the occurrence of the variable as it mapped to a register in an RPN program that represented the equation. He also accomplished this in the BASIC language by using subscripted arrays in place of the simple variables of the equations. The interface allowed a number to specify the required subscript.

What seemed to be missing at this point was an intuitive interface that could be used easily without burdening the user with the mechanics or strict requirements of the operating system. This interface should allow the user to enter any equation and solve for any unknown within that equation without requiring a manual each time the interface was used. It should support any additional operation that would contribute to using the results obtained from the equation. The interface should also reassure the user that the appropriate keys had been pressed and that the specified answer had been obtained. Finally, the requirements of a friendly equation solver interface became apparent:

- It should be possible to assign values easily and independently to all of the variables of an equation or formula.
- It must be possible to select any variable as the unknown to be determined.
- The keystrokes required to perform these operations should be minimal in number, and intuitive to the lay person.
- Output should be clearly labeled to confirm the solutions.
- The equation should not require any special processing by the user before it is typed in.

Pondering these factors, I saw the similarities between the requirements of general equation solving and the HP-12C's time-value-of-money keys. The HP-12C uses an efficient interface that supports the assignment and/or selection of any variable with the fewest keystrokes possible. The only problems were that it only worked for the variables that were printed on the keys and the results were not labeled. Could this interface be applied to any equation? The solution is to use a row of keys for the physical interface and let them be used for all equations. The display above them can be used to label the keys in a manner similar to the softkey approach used on many terminals and computers. This requires either a multiple-line display or a very long single-line display, since the display must also show the values as they are input or output.

The last missing piece was how to enter equations in the form required for the root-finding program. The easiest solution for the user would be simply to allow an equation to be entered in any form whether or not an equals sign is present.

When all of these ideas were combined and labels were added to the output, I was surprised at how easy the interface was to use. As different equations were tried, certain additional enhancements became desirable. Occasionally it was helpful to be able to use the value of a variable from one equation as a variable in another. To accomplish this, all variables are allowed to be global. Thus, a variable maintains its value from one equation to another unless it is recalculated or reassigned. At this point, the ability to scroll up and down a list of equations was added. This made it possible to solve for a variable, press one key, and be in a different equation with the value for that variable already assigned.

Once a working model of this interface was complete, simply showing its use to someone was enough to generate excitement and support, from both marketing and the lab. Here was an interface that could help write programs for us, and clear up some of the keyboard clutter that comes from many functions on a few keys. By using at least two display lines, we could make available many formulas or equation solutions without requiring more keys. The softkeys could also be used for more traditional menus, supporting the functions already found on our calculators and computers, reducing keyboard clutter even further, and improving some of our more traditional user interfaces.

Chris M. Bunsen
Development Engineer
Handheld Computer and Calculator Operation

Grodd developed the RPL kernel. Paul Swadener was our financial consultant, Chris Bunsen pioneered the SOLVE user interface with his early prototype (see box above), Anne Ellendman was our patient manual writer, and Sharon Bolden was our quality assurance person with a seemingly endless supply of energy.

Reference

1. S.L. Wechsler, "A New Handheld Computer for Technical Professionals," *Hewlett-Packard Journal*, Vol. 35, no. 7, July 1984.

An Evolutionary RPN Calculator for Technical Professionals

Symbolic algebraic entry, an indefinite operation stack size, and a variety of data types are some of the advancements in HP's latest scientific calculator.

by William C. Wickes

THE HP-28C (Fig. 1) provides the most extensive mathematical capabilities ever available in a handheld calculator. Its built-in feature set exceeds even the capabilities of the earlier HP-71B Handheld Computer[1] with its Math ROM.[2] Furthermore, the HP-28C introduces a new dimension in calculator math operations—symbolic algebra and calculus. A user can perform many real and complex number calculations with purely symbolic quantities, delaying numerical evaluation indefinitely. This allows a user to formulate a problem, work through to a solution, and study the mathematical properties of the solution entirely on the calculator.

The HP-28C has the following features:

- An RPN calculator interface allowing an indefinite number of stack levels and a variety of data types
- A softkey menu system for key-per-function execution of all built-in and user-defined procedures and data
- Extensive real and complex number functions
- Symbolic algebra and calculus
- An automated numerical root-finder (see article on page 30.
- Vector and matrix math operations
- Automatic plotting of functions and statistical data
- Unit conversions among arbitrary combinations of 120 built-in units and user-defined units
- Integer base arithmetic, bit manipulations, and logic operations in either binary, octal, decimal, or hexadecimal notation
- A keystroke-capture programming language enhanced by high-level program control structures
- An infrared printer interface for printing and graphics output on the optional HP 82240A Infrared Printer.

The HP-28C's physical package differs from that of the HP-18C Business Consultant (see page 4) in only two aspects. The HP-28C uses different key nomenclature optimized for its math operations, and it contains an addi-

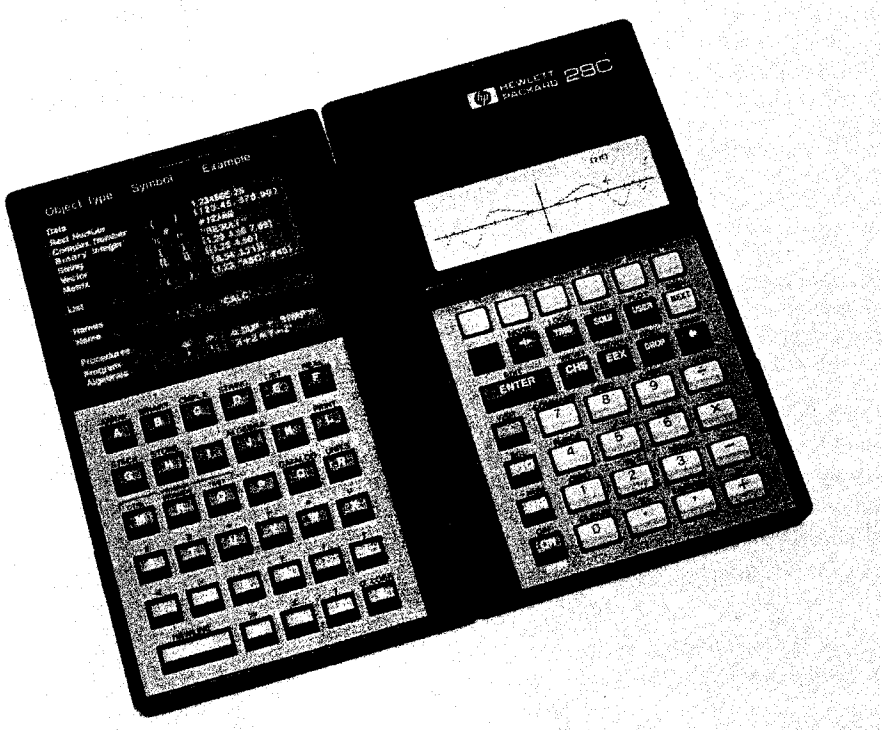


Fig. 1. *The HP 28C Scientific Professional Calculator features symbolic entry of algebraic expressions for an extensive range of functions capable of handling real and complex numbers, vectors and matrices, base 2, 8, 10, and 16 integers, lists, and built-in conversion factors. The display can display up to four lines of the indefinite-depth stack or be used to plot functions with a resolution of 32 × 137 pixels. An integral infrared transmitter allows output of data and graphs to an optional printer.*

tional 64K bytes of ROM, for a total of 128K bytes.

The design philosophy for the HP-28C was to generalize the ease of use, power, and flexibility of HP's RPN calculator interface to a wider class of data types and applications while also eliminating some of the shortcomings of that interface. In the remainder of this article, we describe some of the features of the HP-28C in the context of the evolution of the RPN interface.

Enhanced RPN

Reverse Polish notation (RPN), in which mathematical expressions are written with functions following their arguments, is embodied in computers and calculators by means of a last-in-first-out (LIFO) data stack. Mathematical and logical functions take their arguments (inputs) from the top of the stack, and return their results to the stack where they can be used as the arguments for subsequent operations. An RPN stack is the most efficient medium for chaining and nesting calculations, and provides the greatest keystroke efficiency in a calculator.

The original HP RPN calculator user interface was first used in the HP-35 Calculator[3] in 1972. In that and subsequent HP calculators, the stack consisted of four fixed-length registers, each of which could contain one floating-point number (the HP-41C Calculator[4] also permits alphanumeric data in the form of a character string constrained to fit in a fixed-length number register). This system was satisfactory for the numeric-only capability of the early calculators, but with the advent of programmability and algorithms for more complicated data types, the restrictions of the fixed stack became more and more of a design impediment. For example, in the HP-41C and HP-15C[5] Calculators, complex numbers are represented by two real floating-point numbers, one for the real part and one for the imaginary part. Two stack registers are needed for each complex number, which means that a four-register stack can hold only two complex numbers, severely restricting the types of complex-number math operations that can be performed on the stack. For example, the complex-number expression $(A+B)(C+D)$ cannot be evaluated without storing an intermediate result away from the stack.

The HP-28C is the first HP calculator to modify the traditional RPN interface. To begin with, the concept of a stack register is generalized to a stack level that can hold an object of indeterminate size. An object can be one of several types of data or procedures, each characterized by its internal structure and execution logic. Any object can be manipulated on the stack as a single unit. For example, a complex number is represented by an ordered pair of floating-point numbers that is entered and displayed in the form (number, number). Since a complex-number object now occupies a single stack level, it can be manipulated with the same keystrokes used for a real-number object. For example, complex numbers in the first two stack levels can be added by pressing the + key, multiplied by pressing the × key, etc.

Besides real and complex numbers, HP-28C data objects include real and complex-valued arrays (matrices and vectors), alphanumeric strings, binary integers, and lists. Binary integers are binary coded integers of 1-to-64-bit words which can be entered or displayed in binary, octal, decimal, or hexadecimal bases. Lists are ordered collections of other objects. Data objects are characterized by the simple property that the evaluation of the object just returns the same object.

The generalized stack concept permits the introduction of object classes that have no counterpart in previous RPN implementations. A name object, for example, is a character sequence that is used to identify other objects by name. In the HP-28C, the numbered storage registers on earlier calculators are replaced by variables. A variable is a combination of a name object and any other object stored together in a linked list independent of the stack. Name objects have the property that evaluation of the name returns the object stored with the named variable (and if the object is a program, executes the program). This means that a user variable behaves exactly like a built-in command. (In HP-28C terminology, a command is a built-in, programmable operation.)

A name for which no variable has yet been created fills the role of a formal variable in mathematics, upon which operations can be performed, even before evaluation. Such names just return themselves when evaluated. This property is central to the implementation of symbolic mathematics on the HP-28C.

Evaluation by name and the linked list of variables are modeled after a Forth dictionary. Built-in commands are compiled as their execution addresses, as in Forth. However, user-defined names are compiled unresolved. This permits compilation of undefined (formal) variables, and also allows selective purging of variables from memory, neither of which is possible in Forth. There is a degradation of performance compared to Forth because of the necessity for run-time resolution of user variables, but the overall throughput for user problem solving is usually better because of the ease of programming and flexibility of the HP-28C language.

The remaining new object class defined in the HP-28C is procedure objects. A procedure object contains an arbitrary number of other objects that are executed automatically and sequentially when the procedure object itself is evaluated. The procedure class includes programs, which are unrestricted sequences of data, commands, or variables, and algebraic objects, which represent mathematical expressions and equations and therefore must satisfy certain syntax rules. Both procedure object types can be manipulated on the stack, or named in a variable. In previous calculators, programs were created and edited only in a special program mode.

Variable Stack Depth

In another major break with earlier calculator architectures, the HP-28C stack grows dynamically as new objects are entered onto the stack and shrinks as they are removed. The number of objects on the stack is limited only by available memory. There are two major benefits of this approach. First, mathematical calculations of arbitrary complexity can be carried out entirely on the stack. Second, it facilitates structured programming—procedures can be defined externally in terms of the number and type of arguments they take from the stack and the number and type of result objects they return. Subroutines can be nested to an arbitrary depth without concern for stack overflow.

Example Problem

A farmer has 100 yards of fencing to enclose a rectangular field, which is bounded on one side by a river. What length L and width W of the field will enclose the maximum area?

Solution using HP-28C:

1. The length of the fence is 100, i.e., L+2W = 100. Enter equation using keystrokes: **' L + 2 × W = 1 0 0 ENTER**.
2. Solve for L, i.e., 100−2W, by pressing **' L SOLV ISOL**.
3. Assign this value to L by pressing **' L STO**.
4. The area of the field is LW, i.e., LW = AREA. Press **' L × W = A R E A ENTER**.
5. Substitute for L, i.e., (100−2W)W = AREA, by pressing **EVAL**.
6. To find the maximum area, differentiate by pressing **' W ENTER d/dx**, obtaining the expression −(2*W)+(100−2*W)=0.
7. Collect terms, i.e., 100−4*W=0, by pressing **ALGEBRA COLCT**.
8. Solve for W by pressing **' W SOLV ISOL**.
9. Assign this value, i.e., 25, to W and solve for L by pressing **' W STO L EVAL**. This gives a result of 50.

Answer: The width of the field should be 25 yards, and the length 50 yards. The entire problem can be formulated and solved in the HP-28C without recourse to pencil and paper.

The use of an indefinite stack size as the central user interface is again reminiscent of Forth. The names of various stack manipulation commands—DUP, SWAP, ROLL, PICK, etc.—were adapted from Forth. However, the HP-28C adds a dimension of user protection derived from its calculator heritage. It is not possible to cause memory loss by, for example, pushing too many objects onto the stack as most Forth programmers have experienced. The HP-28C has an elaborate low-memory handler that prevents such drastic results.

The memory stack is a stack of 5-nibble object pointers, not the objects themselves. The objects are stored either in a temporary object area or in user variable memory. Thus, when an object on the stack is duplicated, only the pointer is duplicated. But when the stack is decompiled, the objects are shown, not the pointers, so that the stack has the visible and logical behavior of a stack of the actual objects. The existence and management of the object pointers is entirely transparent to the user.

Command Line

In keeping with its theme of uniform treatment of all object types, the HP-28C provides a free-form command line in place of the multiple entry modes of its predecessors. For example, in the HP-41C the user enters floating-point numbers directly into the stack's X register, alpha data into an alpha register in alpha mode, and programs into program memory via program mode. In the HP-28C, all new objects are typed as character strings into the command line, which is created dynamically when a number or letter key is pressed. The contents of the command line can be edited with cursor, backspace, delete, and insert keys. The unrestricted size of the command line allows the entry of more than one object on one line, as well as calculator commands that are specified by name.

Different object types are identified within the command line by characteristic delimiter characters. For example, strings are enclosed in double quotes, variable names and algebraic expressions are surrounded by single quotes, and lists are enclosed in curly brackets. These delimiters are also used when objects are displayed on the stack.

The centerpiece of RPN keyboards has always been the **ENTER** key. On previous calculators the **ENTER** operation terminates digit entry, copies the contents of the X register into the Y register, and then disables stack lift. On the HP-28C, the concept of stack lift disable has been eliminated (with an indefinite-depth stack, it serves no purpose and would only add confusion), and the role of the **ENTER** key has been generalized to mean "parse and evaluate the command line."

Context-Sensitive Keys

The use of a command line entry method on a calculator that provides immediate key-per-function execution requires a dynamically configured keyboard that is sensitive to the current content of the command line. For example, the primary definition of the + key is to add the contents of stack levels one and two, and normally, the addition is performed immediately when the key is pressed. To preserve keystroke similarity with previous RPN calculators, + should act on the most recently entered arguments, whether or not they have been moved from the command line to the stack. On the other hand, if the user is entering an algebraic expression or a program, pressing the + key should just append the plus sign to the command line. There is a large group of such context-sensitive keys, including virtually all programmable command keys. The remaining keys either execute immediately, like **ENTER**, or add characters and numbers to the command line, like the letter and digit keys.

The action of context-sensitive keys is determined by three entry modes:

- In immediate mode, the default state, context-sensitive keys execute immediately. Where appropriate, most keys automatically perform ENTER before executing their own definitions. Thus the standard RPN sequence to add 3 and 6 of **3 ENTER 6 +** is preserved—pressing the + key enters 6 onto the stack and then executes the addition.
- In algebraic entry mode, keys corresponding to commands such as +, SIN, and LN that are legal in algebraic expressions append their function names to the command line. All other commands execute immediately. These include, for example, stack operations that are outside of the scope of ordinary algebraic expressions.
- In alpha entry mode, all context-sensitive keys append their labels to the command line.

The active entry mode is indicated by the shape of the command line cursor. An open rectangle indicates immediate mode, a rectangle with two horizontal lines inside shows algebraic mode, and a filled rectangle means alpha mode. Similar arrow shapes are used when command line entry is insertion mode rather than replacement mode.

The choice of entry mode depends most often on the type of object being entered into the command line. The HP-28C automatically changes entry mode when certain

delimiter keys are pressed. Pressing the ' key signifies the beginning of an algebraic object, which automatically changes the entry mode from immediate to algebraic entry. Similarly, pressing the " key (strings) or the ≪ key (programs) sets alpha entry mode. These automatic changes mean that most of the time, the user does not have to be concerned about the mode.

Visible Stack

The visible appearance of the stack is considerably different from previous calculators, beyond the simple consideration that the four-line display can show up to four stack levels simultaneously. A typical display might look like Fig. 2. Here we see that there is a 2×2 matrix in level 1, a complex number in level 2, and an algebraic expression in level 3. Note that each object occupies only one stack level, even though each is composed of several parts. The notation 1:, 2:, 3:, etc., was adopted to name the stack levels, because there was no logical extension of the traditional X, Y, Z, T sequence used for the earlier four-register stack to an indefinite number of stack levels.

A major design challenge was solving the problem of how to handle the partial decompilation of objects for viewing. In many cases, an object is too big to display on a single line, or even in the entire display. Therefore, it was necessary to devise a scheme to permit scrolling the display up or down through such an object. At the same time, the limited RAM of the HP-28C makes it preferable not to decompile an entire object into a character string form, since there might not be enough memory available to hold a long display string in addition to the object itself.

This problem is most severe in the case of algebraic expressions. The internal RPN order of the objects that define an expression is not the same as the order of the terms in the decompiled form. The first object in the written form of an algebraic expression may well be the last object in the RPN execution order of the expression. The solution is to generate a compact binary code to represent the display order of the objects in an algebraic expression, including the positions of parentheses and other special characters. This code is preserved as long as any portion of the algebraic expression is displayed. Pointers into the code indicate the currently displayed portion and which portions to display next if the user moves the display window.

Symbolic Mathematics

The first electronic calculators were characterized by their ability to apply a fixed set of operations to data supplied by the user in the form of real numbers. Programmable calculators provided a new generation of capability by allowing users to add their own custom operations to the built-in function set. The HP-28C represents a third generation of calculator design with its capability of applying logical and mathematical operations to programs.

A conventional program can be considered as a symbolic calculation. That is, the program is written in advance of the data to which it is to be applied, and refers to that data only by name or other form of abstraction. However, calculator languages share the common limitation that they cannot manipulate the programs themselves or their symbolic results in their unevaluated form. This includes non-keystroke languages like BASIC, which accept expressions in a pseudomathematical form.

The HP-28C provides two symbolic object types: name and algebraic. In this context, name objects can be considered as expressions consisting of only a single variable. Algebraic objects are just procedures that are entered and decompiled in expression form. They are identical internally to RPN procedures (called programs, for simplicity), except that they are marked as algebraic objects. They are restricted in their definition to satisfy so-called algebraic syntax—they must take no arguments from the stack, return exactly one result, and be completely divisible into a hierarchy of subexpressions, each of which also satisfies algebraic syntax.

The key to performing symbolic math operations on the HP-28C is the behavior of commands corresponding to mathematical functions, which accept symbolic arguments. Such a function examines its arguments, and if one or more is symbolic, returns a new symbolic object representing the function applied to the symbolic argument. For example, if 'A' and 'B' are on the stack, pressing the + key returns the result 'A+B'. Then pressing keys **2** and ∧ returns '(A+B) ∧2'.

When an algebraic object is evaluated by pressing the **EVAL** key, it behaves exactly like the equivalent program—each object in the algebraic object is evaluated in an RPN sequence. Consider the evaluation of the algebraic object '(A+B)*C', where A is has the value 2, B has the value X+Y, and C has no value. The expression is equivalent to the program key sequence **A B + C ×**. When the **EVAL** key is pressed:

1. A is evaluated, returning its value 2.
2. B is evaluated, pushing its value X+Y onto the stack, which now looks as shown in Fig. 3a.

(continued on page 16)

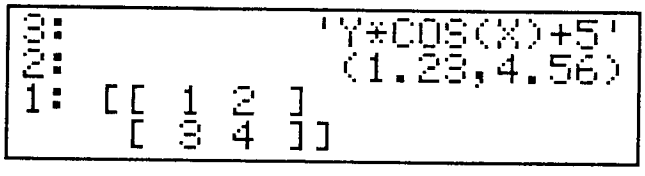


Fig. 2. *Typical HP-28C display with a 2×2 matrix in level 1, a complex number in level 2, and an algebraic expression in level 3.*

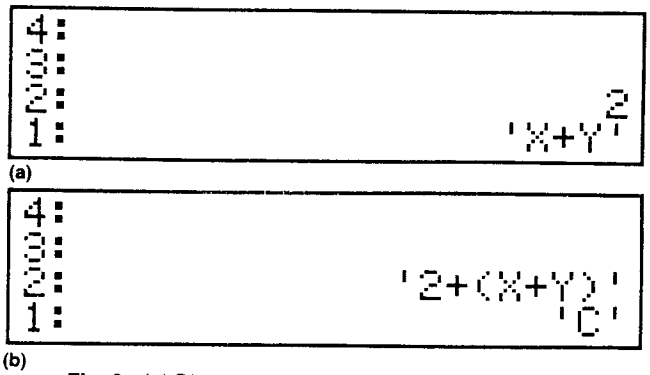


Fig. 3. *(a) Display after B is evaluated (see text). (b) Display after C is evaluated, returning just its name.*

HP-28C Plotting

The HP-28C includes a simple plotting capability for the generation of mathematical function plots and statistical data scatter plots. Although the size and resolution of the liquid-crystal display severely limit the detail and elegance of the graphics, such plots can be extremely useful in providing a global picture of the behavior of a function or a set of statistical data. In particular, a plot is almost indispensable for finding initial guesses for the HP-28C's equation solver, and for sorting out the ambiguities of multiple roots.

As an example, consider the equation:

$$x^3 - x^2 - 2x + .75 = 0$$

This equation has three roots, of which at least one is real. If we plot the expression on the left, using the default plot parameters, we obtain the display shown in Fig. 1a. From this picture, we can observe that there are three real roots, which correspond to the points where the plotted curve crosses the axis. To zoom in on the region containing the roots, we can execute .3 *W, which multiplies the horizontal range by 0.3, then plot again (Fig. 1b).

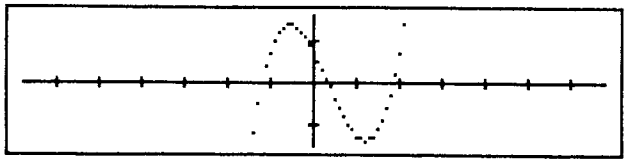
To determine a precise value for any of the roots, we:

1. Digitize two points from the plot at selected values of the horizontal coordinate on both sides of the root.
2. Exit the plot and combine the two digitized coordinates into a list.
3. Activate the equation solver and store the list into the variable X as a first guess for the root-finder.
4. Solve for X.

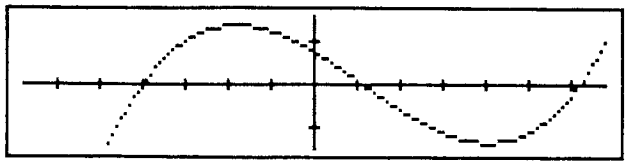
Digitizing is achieved by means of a cursor superimposed on the plot. The cursor can be moved with the cursor menu keys. In Fig. 1, the cursor is invisible because it coincides with the axes at the origin. We move the cursor up off the X-axis to make it more visible and over just to the left of the middle root by using the▲ and ▶ keys (see Fig. 2a). Pressing the **INS** key digitizes the cursor location by returning its coordinates to the stack as an ordered pair (x,y). Now we move the cursor to the right of the intersection as shown in Fig. 2b and digitize a second point. Pressing the **ON** key exits the plot so that the two point coordinates are shown on the stack (Fig. 3a).

Combine the two points into a list by executing →LIST. Then activate the equation solver by pressingt **SOLV** SOLVR. The resulting display is shown in Fig. 3b.

Finally, press the menu key X to store the list as a first guess, then **shifted** X to solve for the 12-digit root = .337301614083 (Fig. 3c). The message Zero indicates that the expression evaluates to machine zero at that point.

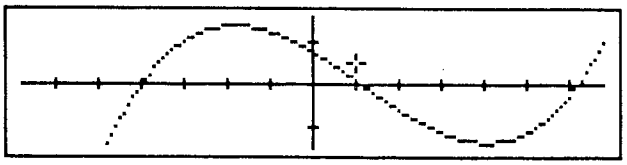


(a)

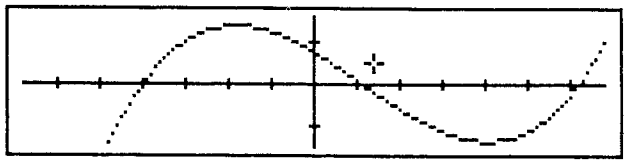


(b)

Fig. 1. *(a) Plot of $x^3 - x^2 - x + .75$. (b) Plot of (a) with horizontal range expanded.*



(a)

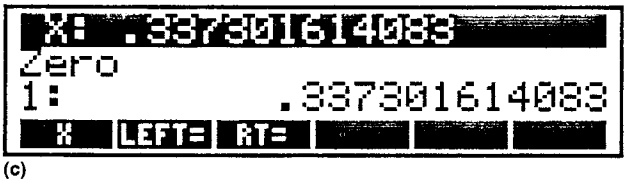


(b)

Fig. 2. *(a) Digitizing point just to left of middle root of plotted equation. (b) Digitizing point just to right of middle root.*

(a)

(b)



(c)

Fig. 3. *(a) Digitized points from Fig. 2. (b) Combining points in (a) into a list. (c) Solution for middle root of equation using list of points from (b).*

Other HP-28C plotting features include:

- Autoscaling for statistical data plots
- Turning on specified pixels
- Printing display images on the optional HP 82240A Infrared Printer.

Gabe L. Eisenstein
Development Engineer
Handheld Calculator and Computer Operation

(continued from page 14)

3. + examines its arguments, finds that one is symbolic, and so returns the symbolic sum 2+(X+Y).
4. C is evaluated. Since it is a formal variable, it just returns its name as shown in Fig. 3b.
5. Finally, * returns the symbolic result (2+(X+Y))*C.

In addition to mathematical functions that can be included within algebraic expressions, the HP-28C provides a host of operations that are not representable as functions, but can be applied to algebraic objects. These operations include expansion, collecting terms, subexpression substitution, symbolic variable isolation, and an expression editor that permits rearranging an expression according to standard rules of algebra. See the box on page 13 for an example of the formulation and solution of a problem involving algebra and calculus on the HP-28C.

This application of RPN principles to algebraic expressions reflects the conviction that algebra, perhaps even more than numerical calculation, is an interactive, postfix process where a user decides how to proceed with a calculation according to step-by-step, intermediate results as the calculation develops. An important aspect of the approach is that the HP-28C is the first calculator to offer a smooth integration of RPN and algebraic interfaces. A user can enter an entire calculation in expression form, as the user might using BASIC, or where appropriate, the calculation can be broken into subexpressions for partial evaluation, with the intermediate results conveniently held on the RPN stack.

Type Dispatching

The inspection of arguments described above for algebraic functions is an illustration of the more general type of checking and dispatching steps common to most HP-28C operations. Every HP-28C command has the following structure:

check_arguments, type_and_dispatch, dispatch_list

The check_arguments process determines if the appropriate number of arguments are present, and issues the "Too Few Arguments" error if not. Note that this error condition is not possible on previous RPN calculators, in which the four stack levels are never empty. The check_arguments process also saves copies of the arguments for possible retrieval by the LAST command.

The type_and_dispatch process returns a code representing the type and position of the arguments and then inspects the dispatch list until it finds a matching code. Adjacent to each argument code in the dispatch list is a pointer to the executable program code for the command corresponding to the argument combination. If a match is found, execution branches accordingly. If the dispatch list is exhausted without a match, the "Bad Argument Type" error is returned.

The type-and-dispatch command structure has some useful side benefits:

- The USE option, available when the CATALOG operation is active, inspects the dispatch list to create a stack-use table to guide a user in the correct use of a command. This provides an important help facility at very little cost in ROM use.
- The type_and_dispatch word is different for commands that are legal in algebraic objects and those that are not. In algebraic entry mode, it is only necessary to check this word to determine whether to execute a command or add its name to the command line. Similarly, the check_arguments word indicates whether to append an opening parenthesis to an algebraic command name as a typing aid.

In addition to the type and dispatch encoding, algebraic functions also include pointers to the code for their corresponding derivative and inverse functions.

Recovery Features

The LASTX feature on RPN calculators, which returns the contents of the X register before the most-recent X-register operation, serves a dual purpose. First, it provides a means of recovering from an incorrect operation—thus pressing **LASTX – LASTX** restores the stack to its state preceding an inadvertent press of the + key. Second, it permits repeated use of the same argument—pressing **SIN LASTX COS** + computes sin x + cos x. Both of these features are present in the HP-28C, but they have been separated and extended into more powerful operations.

The error recovery feature has evolved into the HP-28C's UNDO operation. When ENTER is executed by pressing the **ENTER** key or any other key that does an automatic ENTER, a copy of the current stack (object pointers) is saved in a temporary environment. After the command line is evaluated, the effects of the evaluation on the stack can be canceled by pressing the **UNDO** key, which replaces the new stack with the saved version.

All HP-28C commands that use stack arguments save copies of those arguments that can be retrieved by the LAST command. LAST pushes the recovered arguments onto the stack like its LASTX predecessor, but returns all (up to three —no command uses more) of the arguments, not just the one returned by LASTX.

These recovery features, together with the four-level command stack that saves the most recent command stack entries, can consume a significant amount of RAM in certain circumstances. Each of the three features can be disabled by the user when more RAM is required for an operation.

Development Methodology

The HP-28C firmware was developed in a year by a small team using the RPL operating system and language (see the article on page 21). The use of a highly structured language was necessary for the implementation of symbolic mathematics, but also yielded a significant increase in productivity compared with previous products, which were coded entirely in assembly language. A RAM-based prototype HP-28C was available only three months after beginning the project, which made possible significant design changes based upon customer testing of the prototype. Thus, the emphasis throughout the project was on rapid prototyping of features followed by design modifications based on actual keyboard use, rather than detailed advance specifications.

Acknowledgments

Because of the close interaction of the members of the HP-28C firmware team, it is difficult to separate the many

contributions of each person. Some major areas of responsibility were as follows. Gabe Eisenstein handled display and keyboard management, menus, printing, and plotting. Laurence Grodd took care of the operating system, real and complex scalar and array functions, numeric integration, and type and dispatch logic. Paul McClellan developed the solver, algebraic object parse and decompile, statistics, and unit conversions. Robert Miller worked on the programming commands, object decompile, the command catalog, low-memory handler, and binary integer operations. Charles Patton dealt with the operating system, hardware configuration, symbolic mathematics, command line parser, and program control macros. Max Jones, the author of the *Getting Started Manual* for the HP-28C, made numerous contributions to the user interface and the design of certain symbolic operations.

References

1. S.L. Wechsler, "A New Handheld Computer for Technical Professionals," *Hewlett-Packard Journal*, Vol. 35, no. 7, July 1984.
2. L.W. Grodd and C.M. Patton, "ROM Extends Numerical Function Set of Handheld Computer," *ibid.*
3. T.M. Whitney, F. Rodé, and C.C. Tung, "The 'Powerful Pocketful': an Electronic Calculator Challenges the Slide Rule," *Hewlett-Packard Journal*, Vol. 23, no. 10, June 1972.
4. B.E. Musch, J.J. Wong, and D.R. Conklin, "Powerful Personal Calculator System Sets New Standards," *Hewlett-Packard Journal*, Vol. 31, no. 3, March 1980.
5. E.A. Evett, P.J. McClellan, and J.P. Tanzini, "Scientific Pocket Calculator Extends Range of Built-In Functions," *Hewlett-Packard Journal*, Vol. 34, no. 5, May 1983.

Mechanical Design of the HP-18C and HP-28C Handheld Calculators

by Judith A. Layman and Mark A. Smith

THE HP-18C AND HP-28C represent a new mechanical design for HP handheld calculators. These products use a vertical clam-shell format with a simplified keyboard in a coat-pocket-size package. Using the productivity advantages provided by the use of CAD/CAM (computer-aided design and manufacturing) tools, the package was designed for manufacturability and then thoroughly tested for reliability to ensure quality performance for the customer.

Layout

The HP-18C/28C package was the first product at HP's Corvallis site to be designed principally on a CAD/CAM system. This system improved communication between design engineers and manufacturing engineering during the initial layout phase of the product. It also simplified checking tolerances and provided the expedient automatic transfer of information to the tooling shop for plastic part molds. CAD allowed easy analysis of the design such as package cross sections and the graphical simulation of case rotation (Fig. 1).

Case Design

The continual design challenge for handheld calculator products is providing more functionality in smaller packages. Many components in the HP-18C and HP-28C are integrated to provide more than one function (Fig. 2). This minimizes volume in the product and also decreases the part count for production assembly. For example, the bottom cases not only provide the cosmetic and protective shell, but also support the flexible keyboard assembly. In addition, the case half that houses the alphabetic keys is made to deflect slightly to create a latch which holds the product closed.

Heatstaking is a proven manufacturing process for providing uniform keyboard support. Using this in combination with the case assembly eliminates the need for screws. This process was easily automated because it is controllable, requires fewer parts that are easily presented to the tooling, and results in a sturdier product. The industrial design team chose to give the outside of the HP-18C and HP-28C a clean appearance by keeping the package simple and free of overlays. Because of this, reverse ejection is used to move the molding gate remnant from the cavity (outside) to the core (inside) side of the part. Contrary to convention, heatstaking is done from the top side of the keyboard. The existing keyboard overlay is used to cover the heatstake rivet heads in addition to providing the secondary function labels. The choice of polycarbonate as a case material helps ensure that the product will survive a one-meter drop on all six sides.

Dense Packaging

A hybrid printed circuit board (see article on page 25) is used because the high pin count of the two display driver ICs did not allow them to be packaged in the conventional manner for a surface-mounted device. Because of the high cost of the polyimide substrate material used for hybrid circuits, the board size was kept as small as possible. In all, five ICs, twelve discrete surface-mounted devices, and

four discrete leaded devices are contained on this board. The hybrid portion on one side of the board includes two display driver chips and the CPU chip with 263 wire bonds. The chips are surrounded by a molded plastic dike and encapsulated in epoxy. On the other side of the board, all the surface-mounted devices, including two 44-pin quad packs, are loaded by robots and then vapor phase soldered. The four discrete leaded devices that cannot be vapor phase soldered have their leads preformed in a fixture and then are loaded by hand. One of these leaded devices is an infrared light-emitting diode (LED) used to transmit data to a detached optional printer via an infrared link. In addition to the components, the hybrid circuit board has contact pads for 21 key lines, a beeper, battery springs, and 178 lines to the liquid-crystal display (LCD).

LCD Interconnection

The liquid-crystal display is a four-line, 23-character dot-matrix display with seven status annunciators. The 178 pads for connecting the circuit board to the LCD have a pitch of 0.032 inch and are laid out in two rows along the edges of the hybrid circuit board. The connection between the LCD and the hybrid board uses two elastomeric (zebra) connectors. To establish and maintain proper registration between the hybrid board and the LCD pads, the position of the LCD pads is determined optically. The LCD is then secured in a stainless-steel display clip using double-sided pressure-sensitive adhesive tape. The display clip is then positioned into the hybrid board using a hole that has been precisely punched with an accuracy of ±0.002 inch relative to the display pads. This assembly is then tested and crimped.

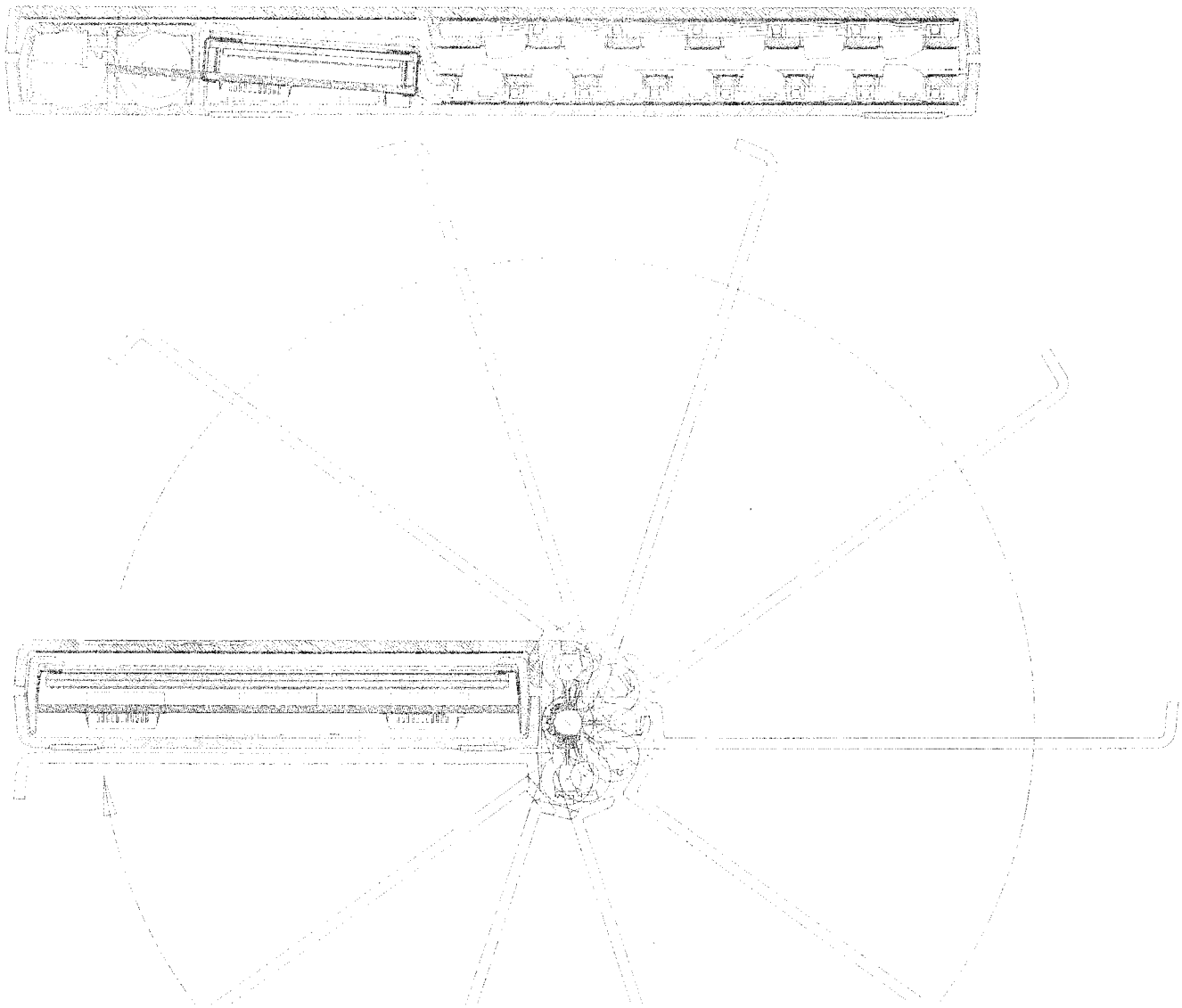


Fig. 1. *CAD drawing of a longitudinal cross section of the closed package (top) and a plot (bottom) of opened case half rotated in several positions.*

Although this basic display assembly concept has been used successfully in two earlier calculator product lines, a few improvements were made in the HP-18C Business Consultant. The first is the inclusion of a relief in the display clip along the edges of the LCD to eliminate stress concentration on the display glass. This allows the product to be dropped from a height of one meter onto all six faces with no functional damage. The second improvement is that the legs of the display clip are flared to allow a lead-in for easy assembly. The precisely punched hole not only establishes proper registration of the LCD to the hybrid board pads during assembly, but also ensures that the LCD will not shift after assembly.

Hinge Link

A compound hinge is used to connect the two halves of the HP-18C/28C case because it allows the product to be used in different positions throughout its 360 degrees of rotation (Fig. 1). By allowing full rotation, this also prevents a situation where the product might be highly stressed if dropped. Several methods of fastening the two hinge halves were investigated. These included gluing, ultrasonic welding, heatstaking, and fastening with screws. Even though it requires more complex plastic tooling, a snap-fit design is used because it offers the most repeatable, simplified process for assembly.

The hinge pins on which the link rotates perform several functions. They are conically tapered to provide axial self-centering of the hinge piece in each case half. The tip of the hinge pin is designed to preload against the inside of the hinge link. This creates frictional drag which provides a high-quality feel to the product as it is rotated. The fragile tip is supported by the main body of the hinge pin which carries any high-stress loads. The hinge pins are open on the top for inserting the interconnect portion of the keyboard into the case halves.

Keyboard and Flex Interconnect

The technology used for the integrated keyboard and flexible interconnect is conductive silver ink screened onto a polyester film substrate. This design allows a single substrate and screening to be used for both keyboards and the flexible interconnect, thus improving the reliability of the system. Twelve key lines run through the 0.140-inch inside-diameter hinge link between the two keyboard halves. Because of the trace width limitations of the screened silver ink process, a complex folded design was implemented to run four layers of the substrate through the hinge link with

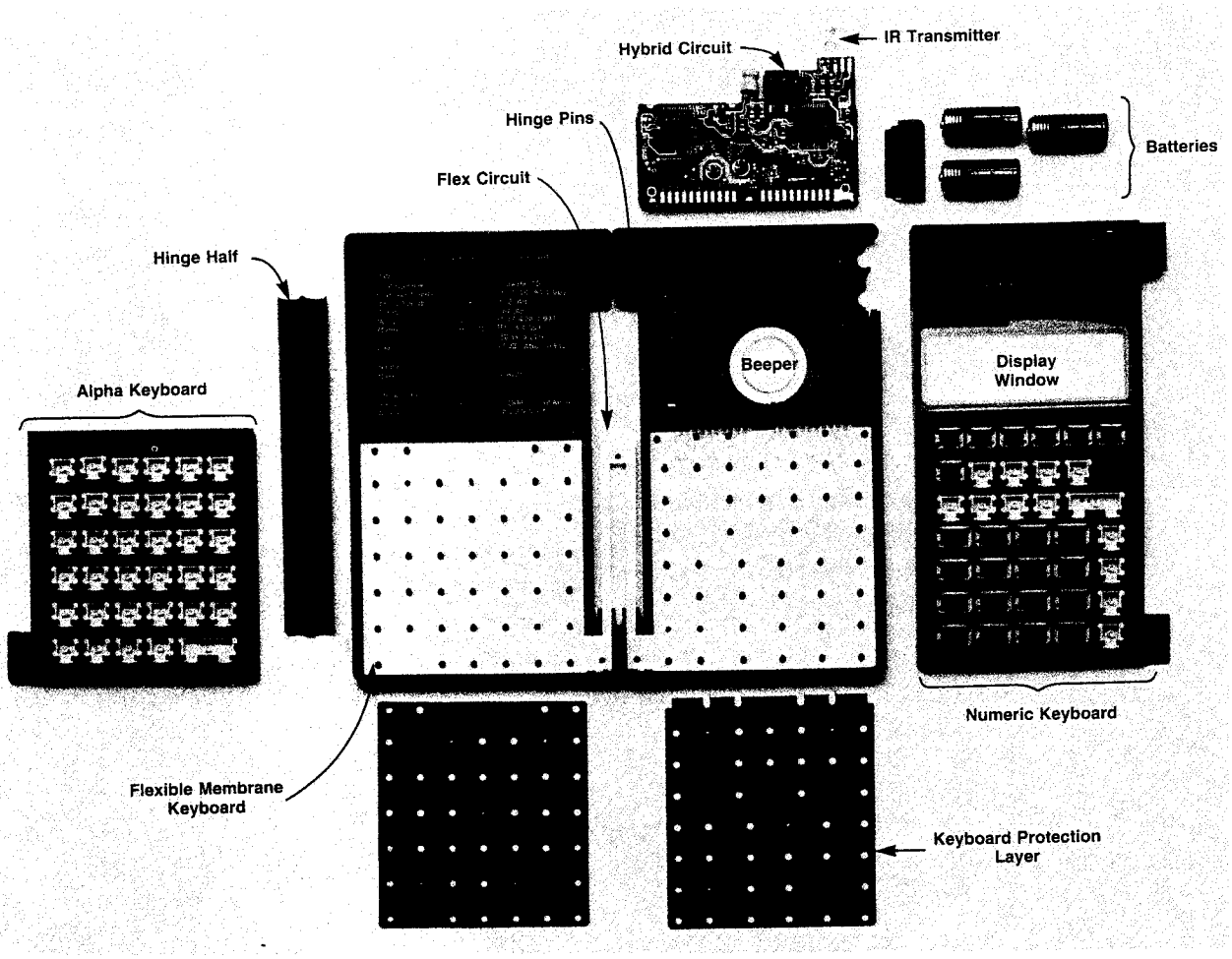


Fig. 2. *Exploded view of parts in HP-28C. The HP-18C is the same except for different key colors and labels and one less ROM on the hybrid circuit.*

each layer carrying three lines (see Fig. 3). Inside the link, the four layers run along one hinge axis, cross over to the other hinge axis, and return while being supported at both ends of each parallel segment.

The torsional stress is induced in a controlled fashion. Controlling the motion of the flexible circuit minimizes locations of stress concentration. As the product is rotated through its full range, the four flexible layers are twisted in torsion. Torsion was chosen over bending because it is less damaging to the conductive ink. The reliability of this design was verified by cycling each leg of the flex circuit through 180 degrees for two million cycles without a failure.

The two keyboards that are an integral part of this flex circuit use the same screened conductive ink for the keypads and circuit matrix. The keyboard technology is typical of that used for membrane keyboards. After the silver ink is cured, a second screening operation deposits a carbon/graphite layer over the silver ink traces. This protects the exposed key line connections to the hybrid circuit against silver migration. The carbon screening process also allowed the ready incorporation of 21 resistors for ESD (electrostatic discharge) protection. In one pass, a resistor is created in each key line by using the screened carbon to bridge a controlled gap in the silver traces. A pressure connection is made between the carbon on the key lines and the gold pads on the hybrid circuit board using two low-compression-set urethane foam pads.

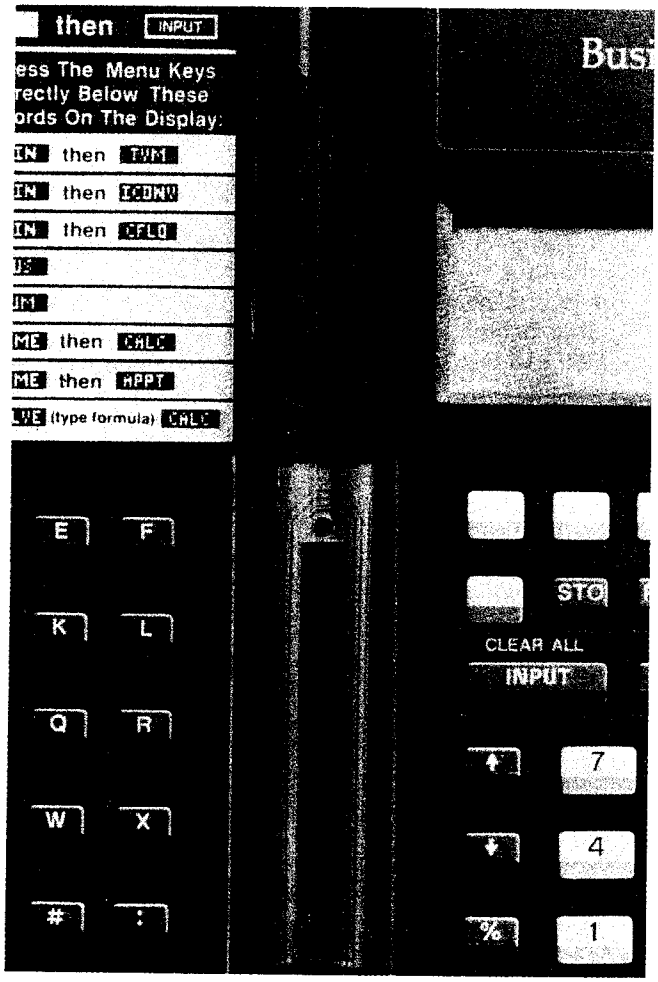


Fig. 3. *Double exposure photograph showing shape of flex circuit and its location within the hinge assembly.*

Tactile feel for the 72 keys is provided by two separate dome sheets of formed polyester. A spacer layer supports the domes while also providing a vent at pressure extremes and whenever a dome is actuated. These layers are all attached to create a single part for ease of product assembly. The keyboard assembly was tested to half a million key cycles with no electrical failures and minimal degradation in tactile feel. Life testing was done at both ambient temperature and under environmental conditions of high temperature and humidity. Several iterations of key design and testing were required to achieve the life and tactile feel desired.

ESD Protection

ESD testing has consistently been a challenge in trying to release products to production on schedule. The testing typically cannot be performed until late in the project because the completed product is required. Fixes that are a result of ESD testing, therefore, do not have time to be integrated into the product properly. With this in mind, special consideration was given to ESD protection early in the design of the HP-18C and HP-28C. A prototype model was built using a similar existing chip set on a prototype hybrid circuit. Results of this testing were incorporated in the final circuit design. Additional testing revealed a localized ESD susceptibility. As a result, an aluminum shield is incorporated in the back side of the keyboard assembly. This shield provides an alternate path for electrical discharge with lower impedance and higher capacitance to ground. Hence, the HP-18C and HP-28C can survive a 25-kV discharge with no permanent damage. This is a significant achievement for a handheld portable product with no external ground.

Conclusion

The attention given to manufacturability in the initial phases of development was worth the effort. The HP-18C Business Consultant was a fast-track project requiring 18 months to develop. Even so, it made a smooth transition from the lab to production. It was up to mature volumes and yields after only four months in production.

The mechanical design of the HP-28C leveraged the work done on the HP-18C. It required only the addition of one ROM to the hybrid circuit and different overlays and key nomenclature.

Acknowledgments

The mechanical development was truly a team effort from many departments throughout the division. The lab design engineers spent long hours learning a new CAD system while designing a fast-track product. Much effort was spent to maintain reliability and manufacturability goals while on a tight schedule. Manufacturing engineering did an excellent job in designing the plastic mold and assembly tooling as well as providing early input in product design. The production team that assembled the products through the prototype builds and into production had much to do with the success of these products.

Symbolic Computation for Handheld Calculators

by Charles M. Patton

A CALCULATOR OR COMPUTER operating system is primarily a set of conventions for memory organization, data structures, and resource allocation combined with a set of software tools to aid in performing operations in accordance with those conventions. In contrast, an application is software built using the resources and conventions of the operating system.

In software development cycles for previous HP calculators the overall scope of the project was small enough that it did not make sense to segregate code into operating system and applications, or even to formalize many of the conventions developed during the course of the project. However, we have passed the point at which it makes sense to create such disposable code for each new machine. Paradoxically, this has come about through our attempts to make calculators simpler, rather than more complicated. Simpler, of course, means simpler to humans, and what is simple and seemingly natural to humans is anything but simple and natural from the point of view of the machine (and vice versa).

In 1984 we began the design of an operating system to meet the needs of future calculators and handheld computers.

Design Goals

The design goals for the system were strongly influenced by the orientation of various research and development projects under way at the time. The goals included:

- Supporting symbolic mathematics operations in a handheld computing environment
- Allowing for maximal trade-off of ROM space for RAM space
- Providing a compact, extensible system able to support a variety of handheld computation systems
- Providing a rapid prototyping environment for calculator development
- Paving the way for future expert-system capabilities.

In the remainder of this article, I will try to explain what these goals entail, how the design team attempted to address them in the RPL operating system, and something of how the features are used in the HP-28C.

Design

The characteristic that most clearly differentiates a symbolic math system from numerical systems is the ability to use an expression (or more generally, a program) as both a procedure to run and as data to manipulate. An example that illustrates this requirement is the derivative operator. Suppose you were to implement the derivative operator on your programmable calculator so that you could take the derivative of a program. You would need to find some way to have the program passed to the derivative program unevaluated, since if it were evaluated all you would get as a result would be a number. Your derivative program would need to take the argument program apart, compute the derivative, and reassemble the result into a new program.

In the process of investigating the feasibility of implementing symbolic math operations on a calculator, the design team examined a variety of operating systems, including BASIC, Forth, and Lisp. While any of these systems can be made to support the capabilities necessary for symbolic math, it is Lisp that most fully integrates them into the structure of the system. On the other hand, the efficient memory management scheme of Forth, along with its RPN style consistent with previous HP calculators, made it a serious contender as well. The ultimate result is a combination of features from both Lisp and Forth that we call the ROM-based procedural language, or RPL.

RPL and Lisp. Features RPL has in common with Lisp include:

- The notions of atomic and composite objects and mechanisms to create and dissect composite objects
- Strict call-by-reference protocol
- The quote operation, whereby an unevaluated object can be passed as an argument
- Temporary (or lambda) variables useful in defining functions
- A temporary object area and a garbage collection scheme for reclaiming memory from this area.

RPL and Forth. Features RPL has in common with Forth include:

- Reverse Polish notation (RPN)
- Arguments passed to operators on an unlimited stack
- Full complement of stack-manipulation operations
- Threaded execution.

However, RPL differs from both Lisp and Forth in a number of significant respects. These differences are direct responses to the challenges posed by a handheld computing environment. While great strides have been made in increasing the amounts of random access memory available in handheld calculators at a reasonable price, RAM is still a relatively scarce commodity. Similarly, while the execution speed of central processing units at a given power consumption has increased dramatically, so have the overall power requirements for calculators. Consequently, calculator CPUs are often run at a leisurely pace compared to their rated speed. These two facts have had an especially significant impact on the design of RPL.

In RPL the fundamental data structure is an object. An RPL object is similar in design to a Forth word, and consists of the address of the executable code that determines the type of the object (the prolog), and the data that makes up the body of the object (Fig. 1).

Objects can be classified as either atomic or composite. The data part, or body, of a composite object consists of a sequence of objects and/or addresses of objects terminated with a special end marker (Fig. 2). Any other structure is classified as atomic.

This composite object structure is quite different from its Lisp and Forth analogs. In Lisp, a composite object is a binary tree of addresses corresponding to the address of the first object and the address of the rest. In RPL, the address of both the first object and the rest are computable from the address of the object, but they are not explicitly part of the object. This implicit addressing tends to decrease RAM use when objects don't stay in RAM very long, as is the case for a limited RAM system.

The Forth structure most analogous to the RPL composite is that of a secondary. The key difference is that in RPL a pointer to an object or a copy of the object itself can be included in a composite with operationally identical results. This embedding capability allows RPL to use address referencing when the addressed object is not likely to move (or be removed) and copy referencing otherwise. One consequence of this structure is that object addresses within composite objects can reference objects within other composite objects (Fig. 3). This capability also allows for more-sophisticated memory compaction schemes.

Object Types

Seventeen object types are currently defined for the RPL system although object types can be added and removed from the system in a relatively straightforward manner. We can break down the atomic objects further into classes depending on certain characteristics of their prologs. The classes are identifier class, data class, and procedure class.

Identifier Class Objects. There are three object types in the identifier class: ordinary identifier, temporary identifier, and ROM pointer. An ordinary identifier is a self-executing variable name. When an ordinary identifier is executed, it searches through the user's variable area for the value bound to the variable name, and executes the bound object.

A temporary identifier is similar to an ordinary identifier, except that when executed, it searches through a stack of temporary environments for its bindings and returns the bound object without evaluating it.

A ROM pointer is used in place of the address of an object when the referenced object is in a plug-in ROM which can move, or be removed from the system. When a ROM pointer is executed, it executes the object it references.

Data Class Objects. Data class objects have the property that when executed, they merely return themselves. These objects include:

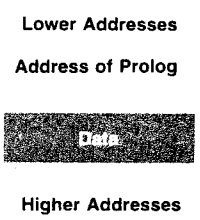


Fig. 1. *Structure of an RPL object.*

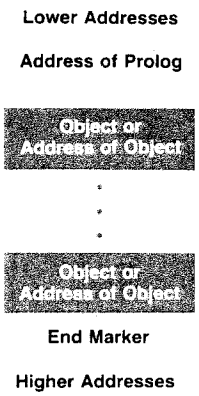


Fig. 2. *Structure of a composite object.*

- Standard and extended-precision floating-point real and complex numbers
- Sequences of characters and sequences of hexadecimal digits
- Unsigned short binary integers
- Arrays and linked arrays of objects of uniform type.

An unusual data class object supported by RPL is the RAM/ROM pair. A RAM/ROM pair is essentially a pair of name-object association lists, one of which resides in built-in or plug-in ROM, and the other of which resides in RAM. It embodies the idea of an extensible ROM-based vocabulary with subvocabularies and context switching.

Procedure Class Objects. Procedure class objects actually

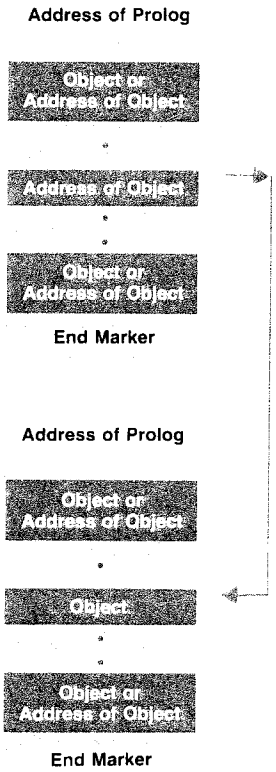


Fig. 3. *An address within a composite may reference an object within another composite.*

do something when they are executed. There is only one type of atomic procedure class object, and that is the machine-code object. The body of a machine-code object contains a sequence of instructions, interpretable by the native CPU of the system, which are executed when the object is executed.

Composite Objects. There are three composite object types: the list, which is data class, the program, which is procedure class, and the algebraic expression, which is also data class. The three types have a very similar internal structure, with a program body being a refinement of a list body, and an algebraic expression body being a refinement of a program body.

Supporting Symbolic Math

Systems that support symbolic mathematical capabilities are often classified by the amount of translation done in converting from a user's typed input to the internally defined data structures. A system is deemed to be radical if the translation is extensive and conservative if little translation is done. In these terms, the HP-28C is very conservative. The most significant translation done is converting the user's input from algebraic to reverse Polish notation. The decision to follow this approach was motivated by a variety of factors.

One motivation for conservative translation was to maximize the perceived responsiveness of the system. In a highly interactive system with the range of processing speed available in a handheld environment, data entry and translation can occupy a large fraction of the overall processing time. A simple translation minimizes this operating overhead.

Compact representation was a second motivation. Standard mathematical notation has evolved over the centuries toward a very compact encoding of the information relevant to the problem at hand. Our theory is that if a user types in an expression in a certain form, that form is likely to reflect important aspects of the problem the user has in mind. Those aspects are likely to become hidden in any radical translation of the form of the expression.

Another motivation was pedagogical. The HP-28C is designed to be a teaching tool as well as a problem solver. We wanted to provide operations that resemble pencil-and-paper operations so that the user can follow and/or choose each step of the operation. To allow this, the internal structure of an expression must correspond closely to the displayed form of an expression, and hence to the form that the user types in.

A final motivation was the uniformity of structure afforded by minimal translation. The internal structure of an expression is the same as that of a program. Thus, no special evaluation mechanism is needed to run an algebraic expression.

Pattern Matching

Pattern matching is another technique commonly used in symbolic math systems. A variety of pattern matching tools are used at a variety of levels in the HP-28C. The lowest-level pattern matching tool is type dispatching, wherein the data types of a set of objects on the stack are matched against a set of templates and the resulting match determines the operation to be applied in this case. This structure can be observed in the HP-28C's CATALOG operation. Each function includes a type-dispatching segment and the CATALOG operation examines the templates included in the function to generate the various possibilities shown by pressing the USE softkey in the CATALOG menu.

At the highest level, an expression-structure pattern matcher compares an expression with a set of templates. The resulting match determines the operation to be performed.

Between these two pattern levels are a number of more-specialized pattern matching utilities which are especially useful in the standard evaluation and simplification algorithms. Although these pattern matching utilities are each quite narrow in scope, the uniform RPN structure of expressions and programs, the ability to use programs and expressions either as executable procedures or data, and the ability to dissect and construct programs on the fly, enable quite general pattern matching operations to be constructed easily.

Symbolic operations are typically defined recursively, that is, the result of applying an operation to an argument is defined in terms of the result(s) of applying the operation to simpler argument(s). In this way, the operation need only be given for the simplest cases, together with a method for reducing a more-complicated case to simpler cases. This definition method is natural for symbolic operations and makes programming the operation simpler and less error-prone.

The RPL operating system is designed to support recursion in an efficient and flexible form. Efficiency is achieved through the uniform use of the stack for passing arguments to operations, the implementation of indirect execution instructions in the central processing unit, and other methods of minimizing the operating overhead inherent in function calls.

Flexibility is achieved by the automatic management of temporary variable environments, and a full complement of control structures that can help minimize the unnecessary buildup of operating overhead. In the HP-28C it is a fairly common occurrence, for example, for a program to create another program and then pass execution control to the newly created program (which itself may create a new program), all at the same execution depth. While this is not the usual case for recursion, it does illustrate the kind of flexibility available in the RPL system.

Trade-Off: ROM for RAM

In designing the RPL operating system, we decided to try to make use of the ROM available in a way that would allow us to get more use out of the limited RAM in the system. The idea is that if an answer exists in ROM, then it only needs to be referenced in RAM, and in effect it takes up very little room. While this seems like a straightforward technique, it was the determining factor in many of the design decisions encountered in implementing RPL. Some examples are the call-by-reference protocol, smart object creation, and embedded objects.

The standard RPL functions take their arguments from the data stack and return their results to the data stack. The data stack, however, is not a stack of objects, but rather

a stack of pointers to objects, that is, memory addresses of objects. Thus every function is passed the addresses of its arguments and it returns the address of its result. It is crucial to the operation of the system that the objects addressed on the stack be allowed to reside anywhere in the system—in built-in ROM, in a movable ROM, embedded as part of the value of some user's variable, or within the temporary object area. The arguments themselves are not altered by the operation of the function (indeed, they can't be if they reside in ROM), but this is not necessary since all that is required is that the function return the address of the result, which again can reside anywhere in the system. This protocol is put to good use in the HP-28C where a sizable number of frequently used objects, including one-letter variable names, are included in ROM. Furthermore, functions that return results equivalent to one of these objects do not create another copy in RAM but merely point to the existing copy.

The composite-object creation and dissection operations also play an important role in RAM-saving aspects of the RPL system. Since any object can occur within a composite object either as an embedded object (the whole object copied in) or as an object pointer (only the address of the object is copied in), with functionally equivalent results, the composite object creation operation can choose to handle objects residing in different areas of memory differently. If an object resides in ROM, only the address of the object is copied into the composite object. However, if the object resides in the temporary object area, the whole object is copied in, making it unnecessary to change the address within the composite object when the object is moved in memory. Other areas of memory with varying degrees of mobility are handled according to the needs of the system.

When a composite object is dissected, it is never necessary to copy any part of it. For example, if an object was embedded in some composite object in ROM, it is never copied to RAM, even if the original composite object is pulled apart.

To get the maximum use out of ROM, it is sometimes necessary to be able to copy objects from ROM to RAM and have these copies act in the same way. With one exception, the currently defined object types operate the same in RAM and ROM. The exception is the RAM/ROM pair, which by definition has a RAM component in which a user's variable values can be stored. Even so, a RAM/ROM pair can be converted to a ROM-like structure (a so-called ROMPART) which can itself then be referenced in a RAM/ROM pair.

Supporting a Variety of Calculators

RPL provides scaffolding for the construction of a system, as well as a basis for operation. The complete version has considerably more structure and functionality than was used in developing either the HP-18C or the HP-28C, although the subsets used in these two machines are rather different. There are explicit points at which the system can be either contracted or expanded and still maintain logical coherence and system integrity. This allows RPL to be used in a variety of situations.

Rapid Prototyping

Taken together, the stack method for passing parameters, the call-by-reference protocol, and the possibility of embedding arbitrary objects within procedures tends to result in very modular code. This modularity contributes to both the possibility of reusing code and the rapid generation of new code. Even when a programmer needs to perform nonstandard operations that require machine code, the simple interface with the stack, together with a complete set of memory management utilities, mean that a programmer can make full use of the central processing unit for the problem at hand. Since this eliminates resource allocation conflicts, the code is easier to write, test, and reuse.

A typical version of an RPL system is composed of a number of parts, each part relating to some facet of the structure. These include:

- Prologs: defining the execution behavior of each data type
- Memory management: resource allocation in the temporary object area, memory movement, address updating, and garbage collection
- RAM/ROM pair management: identifier resolution, variable store, recall, purge, ROMPART manipulation, and context manipulation
- Predicate, logic, and address arithmetic: equality, ordering, NOT, AND, and OR operations, addition, etc.
- Object creation and dissection: head, tail, nth-element, concatenation, composition, decomposition, length, etc.
- Data stack manipulation: stack depth, duplicate, swap, etc.
- Data type conversions: character to integer, integer to character, etc.
- Control structures: quote, evaluate, runstream manipulation, loops, and temporary variable binding
- Array manipulation: creating, redimensioning, accessing, and changing elements
- Configuration: chip-level configuration, ROMPART configuration, and polling
- Exceptions: error trapping, error generation, and error handling tools
- Interface management: key and menu map manipulation, and edit buffer manipulation
- Parser tools: token parsing and parser-generator tools.

While the operations provided by the bare RPL system are quite elementary, they are sufficiently generic and supported by enough structure to make it easy to get a prototype of a new system going in a short time. Once the design of a prototype is ironed out in a standard RPL implementation, it can be optimized further by implementing new data structures and/or translating critical RPL procedure objects to more-specialized machine-code equivalents.

Summary

RPL is an operating system designed to support a variety of applications in a handheld environment. It shares a number of features with both Forth and Lisp systems, but has a number of features that allow it to operate in systems with quite limited RAM. The key design aspects include:

- The universality of structured objects
- Implicit "tail" pointers in composite objects
- Functional equivalence of addressed or embedded ob-

jects within composite objects
- Strict call-by-reference protocol
- Uniform parameter passing on an unlimited data stack
- Automatic temporary variable management
- Quoting operation to allow procedures to be passed as data
- Full complement of RPN-style control structures.

While we do not expect the RPL operating system to be used outside of HP's Handheld Calculator and Computer Operation, we feel that it provides a firm foundation for advances in software technology for handheld computing environments.

Acknowledgments

In addition to all the people acknowledged in the article on page 11, I would like to acknowledge Susan Wechsler and Bruce Stephens for all their help in getting the RPL system up and running.

A Multichip Hybrid Printed Circuit Board for Advanced Handheld Calculators

by Bruce R. Hauge, Robert E. Dunlap, Cornelis D. Hoekstra, Chong Num Kwee, and Paul R. Van Loan

WHEN WE BEGAN the search for an IC packaging and interconnect system for HP's new series of calculators, the design challenges were formidable. Chief among them was achieving an effective compromise among increased circuit density, reduced package volume, greater reliability, and lower cost. Ultimately, our decision was to proceed with a hybrid printed circuit board. No other packaging technology could meet the combined requirements of high pin count, low package profile, environmental stability, and low cost.

The use of hybrid printed circuits is not new for HP. Beginning with development of the HP-41C Calculator nearly ten years ago, the technology has been designed into many of our handheld products. This evolution has led to the hybrid printed circuit board used in the HP-18C and HP-28C Calculators.

The advantages of hybrid printed circuit boards are:
- High density. The use of high I/O count chips (>100 pads) with less than 25% of the area required by comparable discrete packages, multichip and multicomponent applications, and linewidth and spacing geometries of 0.005 inch.
- Design flexibility. A double-sided board allows flexible adaptation to layout requirements. Artwork changes are inexpensive and rapid, selective gold plating for wirebond areas can be used, and finished via hole diameters can be as small as 0.0115 inch.
- Solderability. Components can be added using a wide variety of surface mount or lead insertion solder processes.
- High reliability. For example, customer line scrap on the latest HP-41C hybrid circuits is less than 700 ppm per IC, and the field failure rate is negligible.
- Rapid design turnaround. Artwork changes can be done in five weeks, and assembly prototyping can be done in one to two weeks.
- Low cost. The cost of a hybrid printed circuit compares favorably on a per-pin basis with all other medium-to-high pin-count package types.

Features

The two-sided hybrid printed circuit board used in the HP-18C and HP-28C Calculators measures approximately three inches by 1.5 inches (not including the tab for the infrared LED, see Fig. 1). The top side of the board (the side that mates with the liquid-crystal display of the calculators) bears three custom ICs, two display drivers, and a microprocessor, which are epoxy die attached to the board and connected by a total of 263 gold wire bonds to gold-plated pads. The three ICs are encapsulated by an epoxy layer retained by a dike structure. The reverse side of the board bears two custom ROM chips in plastic quadpacks and nine passive surface-mounted components. In addition, four through-hole components are attached, including an LED for wireless infrared transmission of data to an accessory printer. The finished hybrid circuit with 18 components constitutes virtually the entire electrical system of the calculator.

Technology

In its basic form, a hybrid printed circuit consists of one or more chips that are attached to a printed circuit board, wire-bonded, and then encapsulated to provide mechanical and environmental protection. The early plug-in video game cartridges, primarily designed to be inexpensive, contained a chip mounted on a single-sided board. These boards required no gold plating except for their edge connector, used aluminum wire bonding, and were encapsulated with a glob of epoxy over the IC. Because of their low cost and minimal environmental requirements, it was more cost-effective simply to replace defective cartridges, rather than develop a more reliable assembly. The hybrid process described here incorporates several improvements over

these "jelly bean" types of hybrid circuits.

Printed Circuit Board. A high-temperature laminate, either polyimide or a modified polyimide is used. This allows the use of high-speed gold thermosonic bonding and provides an additional margin for high-temperature applications.

Nickel/Gold Plating. The etched copper traces are conformally plated with a diffusion layer of nickel. Then high-purity, soft gold is conformally plated over the nickel. The conformal plating reduces the possibility of exposed copper, which could lead to dendritic growth between adjacent traces. The nickel also provides a hard underlying surface for wire bonding. The gold plating provides an oxide-free surface for gold wire bonding and protection in harsh and

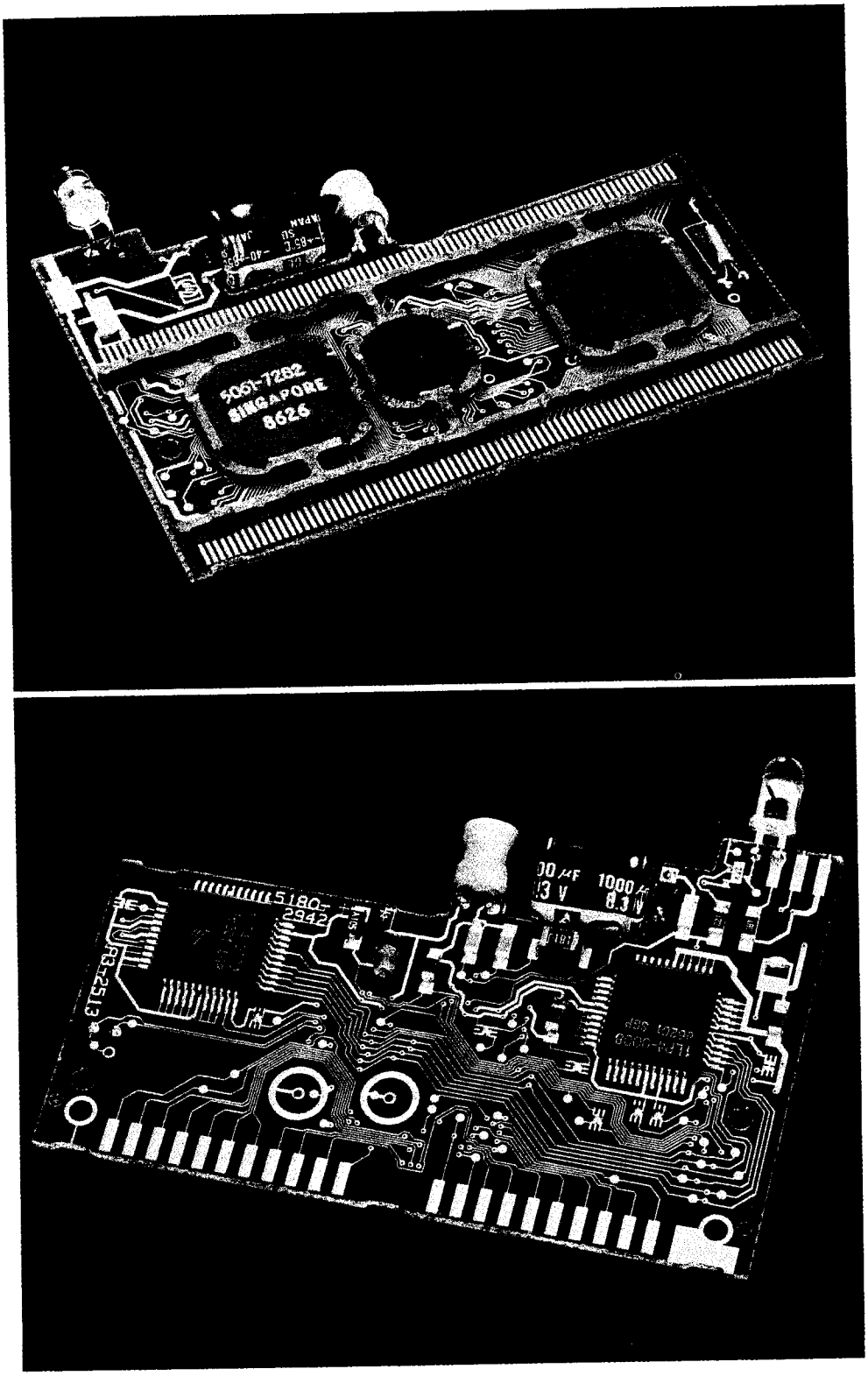


Fig. 1. *Front (top) and back (bottom) sides of hybrid printed circuit board. Precisely punched rectangular hole is located at upper left in top view and upper right in bottom view.*

moist environments.

Die Attach. A high-purity, silver-filled epoxy is used to attach the ICs to the board. This provides good thermal and electrical conductivity to the die-attach pad.

Wire Bonding. Thermosonic (a combination of temperature and ultrasonic energy) wire bonding is used to connect the IC pads to the board traces. The use of gold wire with a diameter of 0.00125 inch and an average pull strength of 15 grams provides extra margin at temperature extremes.

Encapsulation. A low-ionic-content epoxy is used to encapsulate the ICs and their bond wires. Entrapped air is minimized by precuring the boards in a vacuum oven. The thermal coefficient of expansion and the cure cycle are important aspects of encapsulation that must be carefully controlled to minimize thermomechanical stresses on the ICs and bond wires.

Development Program

Because of the high number of wire bonds (263), it is imperative to obtain the highest yield possible at this operation. Our defect rate goal of less than 100 ppm translated to a part yield at the wire bond operation of 97.4%. To achieve this, we needed the expertise of the people at HP's manufacturing facility in Singapore. Therefore, we added a Singapore engineer to our design team in Corvallis for six months to work on the tooling and optimization of the wire bonder.

Tooling modifications were necessary to allow for bonding boards processed in a panel configuration, rather than a single board at a time. This required changes to the X and Y travels of the bond head, a much larger heater block, and a different clamping arrangement.

For the optimization, a partial factorial experiment was conducted to determine the primary parameters that affect bond quality. After this experiment was completed, an operating window study determined the limits of the key parameters. Table I outlines the results.

Table I

Key Bonding Parameters

Parameter	Specification Limits
Temperature	160 to 170°C
Power	40 to 45 pulses/s (die side)
	70 to 75 pulses/s (lead side)
Sink depth	30 to 90 μm (die side)
	150 to 250 μm (lead side)
Force	40 to 50 grams (die side)
	90 to 100 grams (lead side)
Time	15 to 25 ms (die side)
	35 to 45 ms (lead side)

The choice of laminate for the hybrid substrate was narrowed to materials that could withstand the wire-bonding temperatures and times without deterioration. FR-4 boards are normally not usable for thermosonic bonding since a high glass-transition temperature (T_g) is required. A modified-polyimide laminate was chosen because of its relatively high T_g (180°C), and its lower cost and ease of machinability compared with polyimide laminates.

The mechanical design of the hybrid was determined to a large extent by the needs of the calculator design group and by the surface mount process in printed circuit assembly. The board thickness was increased from the normal value of 0.031 inch to 0.047 inch to provide a more rigid substrate. The height of the encapsulating epoxy must be kept to less than 0.080 inch because of tight spacing between the board and the LCD assembly. A molded plastic dike is used to contain the epoxy and maintain a uniform thickness.

To exploit automated board assembly fully and to maximize material use, the boards are delivered from the vendor in a four-board subpanel, using an interdigitated layout. The parts are kept in subpanel form throughout the hybrid and surface mount processes. Only at the hand-soldering operation are the boards separated from the subpanel.

One of the special requirements is a precisely punched rectangular hole (see Fig. 1). This hole must be punched and referenced to the board artwork to an accuracy of ±0.002 inch. The purpose is to allow the use of prealigned display assemblies (LCD plus crimped metal can). This differs from previous HP calculator assembly techniques that require the LCDs to be adjusted manually for each calculator. A vendor was located that had developed equipment to achieve this. Once we proved the accuracy of the machine, we arranged for the precision-punching to be done by the board vendor.

Some of the more severe tests for hybrid circuits are moisture resistance, thermal shock, and multicycle vapor phase soldering. We ran engineering tests to determine whether we had sufficient margin to pass our qualification tests. We saw no problems in the thermal shock and vapor phase soldering tests. However, during the moisture resistance test, we discovered some procedural and humidity chamber design problems. By discovering and correcting these problems before the final qualification run, we averted any program schedule delays.

Because of differences in the coefficient of thermal expansion between the printed circuit board, silicon chips, and encapsulating epoxy, mechanical stresses can develop during heating and cooling. After cooling the subpanel down to room temperature from an epoxy curing temperature of 150°C, a noticeable warpage of the subpanel developed, often greatly exceeding the allowable maximum of 0.075 inch.

We focused on revising the cure cycle as the method to minimize the panel warpage. We needed to cure the epoxy as completely as possible, but not lock in a high state of stress. Hence, a two-stage cure cycle was implemented. The parts are cured for three hours at 125°C, then ramped down to 58°C over two hours. This results in parts that consistently pass the maximum warpage criteria.

In a normal printed circuit board manufacturing process, the areas to be plated are defined by a negative resist on the copper-clad laminate. The exposed copper areas are then plated with additional copper, nickel, and gold. After plating, the resist is stripped off and the exposed copper between the plated areas is etched away. However, this results in traces with exposed copper on their sides. The

exposed copper can react with moisture and an applied bias to form copper dendrites. Hence, several plating enhancements were implemented to improve the reliability and reduce the costs.

The first of these is to process the board in a conformal-plating configuration. This means that the copper etching is done before any plating occurs. Since the sides of the traces are now exposed during plating, the copper is sealed in by the nickel and gold plating steps, thereby reducing the likelihood of dendritic growth.

For conformal plating, all of the features are electrically bused together. This is normally achieved by running small traces off the board for connection to the plating bus. However, since this hybrid board will have a metal can crimped around it to hold the LCD, we risked shorting to these plating traces. Therefore, an alternative method was developed. A plating ring is set up around each of the three die-attach pads. Small traces from each bond finger are connected to these rings, and a single trace is then run off the edge of the board in a safe area. After plating, a fine-diameter router is used to cut away each of the plating rings and open up the shorted plating traces. A solder mask layer on top of the rings and traces helps anchor them to the board to minimize any smearing of the copper during routing.

To reduce the amount of high-purity gold plated on each board, a selective plating resist is screened on after the boards receive a flash gold plating 5 to 20 microinches thick. This resist exposes only the bond fingers, which are subsequently plated with 40 microinches of high-purity, soft gold for wire bonding. The resist is then stripped off.

Test Program

The test software and hardware embodies many features absent in the evaluation of previous hybrid circuits. The prominent features are a modular test program, the use of solid-state analog multiplexers to leverage a few available tester channels for testing continuity on many pins, and a large free-standing test fixture for testing multihybrid panels. The test system tests a hybrid circuit with 250 test points in 25 seconds using a tester with 60 active test pins and 24 additional pads accessible via multiplexing.

The hybrid test program was developed in two separate parts with the objective of achieving complete modularity for the two parts. One engineer wrote the hybrid program, which contains a shell for insertion of the display driver portion written by another engineer, a subset of the microprocessor IC test program (old), a system test (new), and initialization and exit routines for hardware checkout and operator interface (new). This engineer was also responsible for interfacing with the fixture designer and doing prototype hardware debug. A second engineer was responsible for the display driver wafer test program (new), the display driver continuity test (new), the final integration of the test, documentation, and release to production. Programming proceeded in parallel, with each test of the hybrid circuit able to be debugged independently. As needed, common hooks between the separate portions of the final hybrid test program were agreed upon to activate debug features and maintain summary data.

As development proceeded, the display driver portion of the test program was periodically updated with more complete code by the transfer of a single block of code from the middle of the display driver wafer program. This worked well, and ensured that after release to production updates to the display driver wafer program could be easily transferred to the hybrid circuit test program.

Even at the wafer level, the pin count (112) of the display driver chip was too high to access all pins of the part, even with multiplexing of the 60 tester channels to the 24 extra pads. In light of this, the display driver was designed to allow virtually complete testing of all 92 display pins via just four specially designed display pins. Using a combination of internal connectivity switching and scan path methodology, the display pins are tested for functionality, pin leakage, and pin shorts through these four pins.

At the hybrid level, however, we were confronted with the need to confirm the presence of wire bonds to the display pins of the two display driver chips. This continuity check requires a physical connection to each display pin and is done by forcing a current into each pin and detecting the presence of a diode voltage drop across a pad protection diode. Originally it was anticipated we would need to do this test on a separate dedicated commercial continuity tester. However, we devised a solution that allows the continuity test to be done as an integral part of the total hybrid circuit test. This provides the considerable advantage of eliminating the need for a separate commercial continuity tester and being able to do a complete test in a single pass.

The solution consists of a box of solid-state analog multiplexers which use a total of seven tester channels to test 184 pins for continuity in less than 2 seconds. The box contains two identical circuit boards, each with six 16-channel analog multiplexers, one decoder, and one counter. The boards plug into edge connectors connected to the display pins and the tester control channels. To access any pin, the tester increments the counter, which together with the decoder selects an analog channel connected to a particular pin. Since all display pins are in just two contiguous groups, the only short circuits that are physically likely are adjacent pin shorts. Thus to identify shorted pins as well as open circuits we simply wired the box so that the multiplexer channels of the two boards are interleaved, that is, every other display pin is connected to successive channels on the same board. The tester opens one channel on each board simultaneously, forces a current into one channel and forces a zero level on the other. A short to an adjacent pin then results in a current path from the programmable measurement unit through one multiplexer board, through the short, and back out through the other multiplexer board to the tester channel forcing the zero level. Since the part is designed to allow detection of shorts via functional testing, no bad parts would be shipped if this shorts test were not done. However, the ability to identify short circuits directly rather than by interpretation of functional test data has proven to be indispensable in failure analysis and hybrid process monitoring.

Test Fixture

The design of this hybrid circuit implied some new challenges for the capability of the test fixture. Contact had to

be made to 250 points distributed on both sides of a printed circuit board about 3.5 inches long by 1.5 inches wide. The minimum spacing between LCD pads is 0.016 inch, with a pad size of 0.016 inch. We strongly desired a single fixture design that would test the hybrid circuits in both the four-board panel form during manufacture and the single-board form during final assembly. A fixture meeting these requirements was designed and implemented by HP's Handheld Calculator and Computer Operation. This fixture (Fig. 2) has also served well for line scrap analysis.

The fixture weighs about 50 pounds and is manually operated and pneumatically actuated. The four-board panel (or single board) slides into the fixture on a movable X-Y stage. The panel or board under test is positioned by mechanical stops to align closely with the upper and lower spring-loaded test pin blocks. Activated by a manual switch, the upper and lower pin blocks then move to the center and sandwich the panel or board between them. Precision alignment is achieved by the mating of a fixture guide pin to the precisely punched hole in the hybrid circuit.

Originally the test pins chosen for the fixture were solid cylinders with a conical cavity at the contact end, yielding a circular knife edge for contact. This configuration was chosen to satisfy the need for both a sharp edge to penetrate oxides and a large potential contact area to make up for registration errors. These pins performed fine when new, but soon tended to plug up with particulate contamination. Several months into prototype production, a switch was made to more conventional pencil-point, spring-loaded test pins, 0.027 inch in diameter, with favorable results.

Initially, the connection to the fixture consisted of a three-foot-long bundle of coaxial cables terminated by connectors at both ends, mating at the tester end to connectors wired to a DUT board mounted on a performance board. This arrangement was quickly discarded as noise levels were intolerable, and was replaced by a set of shorter cables terminated by connectors at only one end, and directly wired to the performance board at the other end (see Fig. 2). The coaxial cable shields are all soldered to a brass grounding ring offset from the board. The center wires of the coaxial cables are soldered directly to the performance board pads with strain-relief loops. Wire lengths are kept to a maximum of 18 inches. Inside the fixture, lines believed to be critical are also wired in coaxial cable to the spring-loaded test pins, while the remaining lines are twisted pair. As might be expected, even with this arrangement noise is still a problem. This is compensated for by setting input levels to the rails and output levels to 0.33 and 0.67 V_{DD}. This is acceptable because all parts are tested to full level specifications at the wafer stage.

Qualification Results

The qualification plan for the hybrid circuit included:

- 1000 hours of dynamic burn-in at 100°C
- 168 hours moisture-resistance testing at 65°C and 90% relative humidity
- 200 thermal shock test cycles
- 5 vapor-phase solder cycles.

The 1000-hour dynamic burn-in is normally done at 150°C. We lowered the temperature to 100°C because of thermal limitations imposed by some of the soldered components. Earlier moisture-resistance testing of hybrid circuits at 85°C and 85% relative humidity had shown poor results; 65°C and 90% relative humidity was felt to be an adequate condition. Table II summarizes the results. To date, the hybrid circuits in the HP-18C and HP-28C have

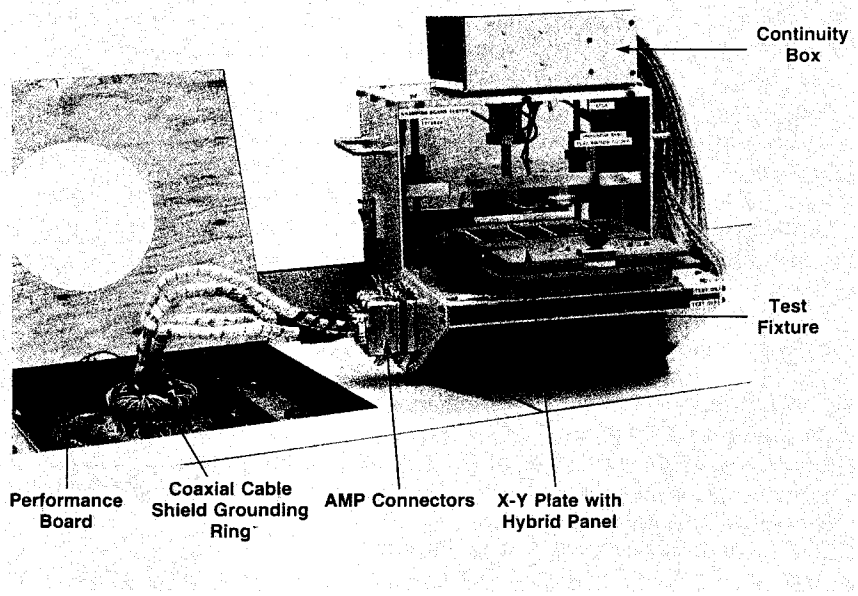


Fig. 2. *Test fixture for hybrid printed circuit board.*

performed very well in the field, with no known reliability problems. This would appear to confirm the validity of our qualification matrix.

Table II

Hybrid Circuit Qualification Results
(Failures per number tested)

Test Procedure	Quality Criteria	Results
Dynamic burn-in: 100°C, 1000 hours	1/129	1/129
Moisture resistance: 65°C, 90% R.H. 500 hours	2/105	0/105
Thermal shock: 200 cycles	5/116	5/116
Vapor phase solder: 5 cycles	0/22	0/22

The six failures from dynamic burn-in and thermal shock were analyzed. The one failure during dynamic burn-in failed the self-test on the crimper tester. This part was subsequently retested on the HP 3065 Circuit Board Test System and it passed. It was then retested on the crimper tester and it passed. No further failure analysis was performed.

The remaining five units showed LCD pad leakage and functional failures during the first thermal shock tests. Examination after decapping showed fractures along the outside edge of the ICs. These problems were shown to be stress related, associated with the large size of the ICs and incomplete die-attach epoxy coverage under the corners. Additional units were built in Singapore, with particular care to obtain complete epoxy coverage under the ICs. The thermal shock test was repeated with no failures and no evidence of fracturing after decap. Singapore has since incorporated a screening method for the die-attach epoxy to ensure process integrity and reliability.

Acknowledgments

The authors wish to thank the many contributors to the hybrid printed circuit board project. Ron Keil developed the modified encapsulation curing profile, Bill Hanna and Jim Traut performed failures analysis on the parts, John Shea managed the test project, Khoa Tran worked on test and fixture development, Marty Marino designed and assembled the test fixtures, Lucy Cornelius worked on the performance board and continuity board wiring, and Preston Brown designed the special test fixtures and provided the test vectors. Manufacturing implementation in Singapore was spearheaded by Soo Kok Leng and Tay Ewee Liang.

An Equation Solver for a Handheld Calculator

by Paul J. McClellan

THE IDEAL EQUATION SOLVER reliably finds all solutions for an arbitrary variable in any equation defined by the user. Since this is provably impossible in general,[1] more realistic expectations are to solve for an arbitrary variable in a wide range of equations, to provide understandable and reliable diagnostic information should the solver fail to find a solution, and to provide the means for using the solver to obtain multiple solutions of an equation if more than one solution exists. These were the design objectives for the equation solver in the HP-18C Business Consultant.

A Combination of Direct and Iterative Solvers

The HP-18C employs a combination of a direct solver to solve simple equations reliably and quickly and an iterative solver to search for solutions of more-difficult equations. The direct solver attempts to solve an equation by applying rules of algebra to isolate the unknown on one side of an equation. If it succeeds, the value of the other side of the equation is the solution to the equation. The iterative solver applies a trial-and-error search procedure to obtain a solution to the equation.

The need for a combination of direct and iterative solvers became clear early in the development of the HP-18C. Although iterative solvers can be applied to a wide variety of equations, they can, depending upon the starting point, take an unacceptable amount of time to find a solution or even fail for trivial equations. For example, consider attempting to solve the equation $1/x = -0.1$ for x by applying the secant method to the difference between the left and right sides of the equation. Fig. 1 illustrates the shape of the function $1/x + 0.1$ near $x = 0$. With initial guesses -1 and 1 the iterates converge to the pole at $x = 0$. With initial guesses 1 and 2 the iterates diverge toward ∞. But with initial guesses -1 and -2, the iterates converge to the solution at $x = -10$. Although a direct solver would handle

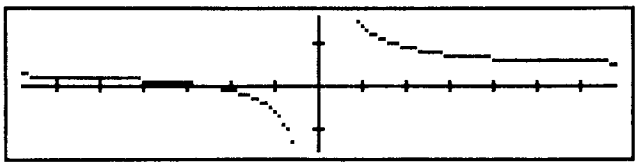


Fig. 1. *HP-28C plot of f(x) = 1/x + 0.1.*

this situation easily, direct solutions to other equations may not exist or may require an excessively complex direct solver. Furthermore, simple direct solvers will return at most one solution to an equation with multiple solutions, which forces the user to rewrite the equation to obtain alternate solutions. Hence, an iterative solver that can accept user-supplied initial guesses can be useful in tackling harder equations or obtaining multiple solutions without rewriting the equation.

To solve an equation, the HP-18C first applies its direct solver. If the direct solver succeeds, the HP-18C displays that solution. If an arithmetic error occurs within the direct solver, then the HP-18C displays the message SOLUTION NOT FOUND. This screens some equations that obviously have no solution. If the unknown appears more than once or if it appears as the argument of a function that the direct solver cannot invert, then the direct solver fails and the iterative solver is invoked.

Direct Solver

As described above, the direct solver solves an equation by applying rules of algebra to isolate the unknown on one side of an equation. If the direct solver succeeds, the value of the other side of the equation is the solution. Direct solvers can fail, either because no closed-form solution exists or because the solution method is too difficult. The first case is illustrated by attempting to solve the equation $x^x = 2$ for x.

The second case is illustrated for the HP-18C by attempting to solve $x+x = 1$ for x. The solution of this equation is difficult for the HP-18C because, considering the product's applications and resources, we decided that the HP-18C's direct solver would perform no algebraic simplification of the equation and thus would require the unknown to appear only once in the equation.

The HP-18C parses an equation into an RPN internal representation of its left and right sides. It parses an expression as though it were an equation with the expression as the equation's left side and a zero on the equation's right side.

The direct solver begins by scanning each side of the equation and finding the side containing the unknown. If the unknown appears in both sides, then the direct solver fails. Otherwise, it initializes the solution accumulator to the value of the other side and discards that side. The direct solver then repeatedly applies the following procedure to the solution accumulator and the remaining subexpression containing the unknown. If the subexpression consists of only the unknown, the direct solver has succeeded and it returns the value of the solution accumulator. Otherwise, the subexpression is an RPN expression ending in a function (or operator). If the direct solver does not know how to invert that function, it fails. Otherwise, it scans the function's arguments to find the occurrence(s) of the unknown. If the unknown appears in more than one argument, or in an argument position for which the direct solver does not know how to invert the function, the direct solver fails. Otherwise, it performs the inversion using the accumulated solution and the current values of any other function argument. If an arithmetic error occurs during this inversion, or if the result violates a rule of algebra, the direct solver terminates and displays the message SOLUTION NOT FOUND. Otherwise, the direct solver discards all but the argument expression containing the unknown and continues this process.

Two situations for which the direct solver aborts and reports SOLUTION NOT FOUND can be illustrated by solving the equations $1/x = 0$ and $0/x = 1$ for x. When the direct solver attempts to invert the first equation, it triggers a divide-by-zero error. When it inverts the second equation, it obtains the result $x = 0$, which indicates a divide-by-zero error in the original equation.

For the most part, the HP-18C's direct solver will only invert functions that have unique inverses for the unknown's argument position. However, we decided to also invert an expression containing an unknown raised to a power. When the power is even, the inverse can be either positive or negative. The HP-18C selects the positive inverse. Sometimes the choice the direct solver makes causes an arithmetic error later in the inversion process and the HP-18C reports SOLUTION NOT FOUND in spite of the fact that the equation has a solution that would have been found had the direct solver chosen a negative inverse.

Even if the direct solver succeeds with its choice, other equation solutions may exist. The user can force the direct solver to choose the other inverse by rewriting the equation. In effect, the direct solver will select the negative inverse if the user negates the subexpression that is raised to the even power. This feature can be illustrated by the following two examples:

- Solve the equation $[1-(1/x)]^2 = 1$ for x. Because the direct solver takes the positive inverse of an expression raised to a power, later in the inversion process it encounters the simplified equation $1/x = 0$ and reports SOLUTION NOT FOUND. If the original equation is rewritten as $[(1/x)-1]^2 = 1$ the direct solver returns the solution $x = 0.5$.
- Solve the equation $1/(1-x)^2 = 0.25$ for x. The direct solver returns the solution $x = -1$. If the equation is rewritten as $1/(x-1)^2 = 0.25$ the solver returns the other solution, $x = 3$.

We decided not to invert other multivalued inverse functions, such as integer part, because such functions have an infinite number of mathematical inverses (and a large number of machine-representable ones) and it would be more difficult for the user to specify any but the default inverse that the direct solver would supply. The iterative solver with its feature of accepting initial guesses from the user seemed better suited to solve such equations.

Iterative Solver

The iterative solvers in the HP-18C and the HP-28C Cal-

culators are very similar. The HP-18C's iterative solver is described first and the HP-28C version's differences are described later.

When a parsed equation is evaluated internally, the current values of the equation's left and right sides are returned. The iterative solver searches for a zero difference between the left and right sides by repeatedly varying the value of the unknown and computing the difference between the sides for that value.

Suppose the goal is to solve the equation $A(x) = B(x)$ for x. We represent the difference between the equation's left and right sides by $f(x) = A(x) - B(x)$. Then the goal is to find a value of x such that $f(x) = 0$. If the iterative solver succeeds, it has found a numerical solution to the user's equation or a zero of the user's expression and the solver terminates immediately and reports that solution. Because of the finite-precision floating-point arithmetic used by the HP-18C, a solution may satisfy the equation numerically but not mathematically.

The set of values available to the iterative solver as candidates for the value of x is the set of machine-representable numbers available to the user. During the search process the iterative solver displays selected iterates to show the region being searched and the corresponding sign of f(x) to provide hints of the shape of the curve and the method in progress. The user can interrupt the search process by pressing any key. If an arithmetic error occurs during the evaluation of f(x) for some x, then f(x) is not defined for that value of x and we say that x lies outside the domain of definition of f(x). The displayed sign at that point will be a question mark.

The iterative solver begins by claiming adequate scratch storage, setting initial search bounds $b_1 = -\infty$ and $b_2 = \infty$, and obtaining and ordering two distinct starting values, say x_1 and x_2, for the unknown x. It obtains x_1 and x_2 by using the last two values stored into x. The default values are zero. If these values are identical, one is perturbed by the solver. At this point we have $b_1 < x_1 < x_2 < b_2$. The iterative solver evaluates $f(x_1)$ and $f(x_2)$. If neither x_1 nor x_2 is within the domain of f(x), that is, f(x) is not defined for x_1 and x_2, then the solver terminates with the message BAD GUESSES. If only one value, say x_1, is within the domain of f(x), the solver sets $b_2 = x_2$ and attempts to find another value within the domain by first using a modified bisection search of the interval from x_1 to x_2. The search bound b_2 is reset to any sample value found out of the domain of f(x) during this search. If the bisection search exhausts all machine-representable values in the interval from x_1 to x_2 without finding one in the domain of f(x), the solver samples the next machine-representable number just before x_1 in the direction of b_1. If this value is also not in the domain of f(x), the iterative solver terminates with the message BAD GUESSES. Otherwise the iterative solver has the ordered pairs (x_1,x_2) and (f_1,f_2) where $b_1 < x_1 < x_2 < b_2$, $f_1 = f(x_1) \neq 0$, and $f_2 = f(x_2) \neq 0$.

If $f_1 = f_2$, the solver searches for a slope by alternately extending the interval bounds x_1 and x_2 until it either finds x_1 and x_2 such that $f_1 \neq f_2$ or it exhausts the search interval. If during this slope-hunting process a sample value is found outside the domain of f(x), the search bound in that direction is set to that value and a modified bisection search is employed to find a sample value in the domain of f(x) in that direction. If the values sampled on one side, between b_1 and x_1 or between x_2 and b_2, are exhausted, then subsequent sample values will lie in the other side. If the solver fails to find a slope, it terminates with the message SOLUTION NOT FOUND.

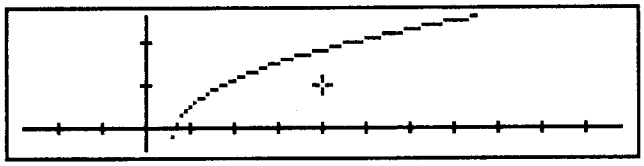


Fig. 2. *HP-28C plot of* $f(x) = \sqrt{x + \ln(x)} - 0.5$.

Otherwise, f_1 and f_2 have different values. If they have the same sign, the iterative solver resets the search bound closest to the value generating the larger f(x) magnitude to that value, sets a counter to seven, and extrapolates in the direction of decreasing f(x) magnitude using a modified secant method.[1] It continues searching in that direction until the value of f(x) changes sign, its magnitude increases, or the search interval is exhausted.

In the last case, the solver terminates with the message SOLUTION NOT FOUND. If during this extrapolation a sample value is found that lies outside the domain of f(x), the search bound in that direction is set to that value and a modified bisection search is employed to find a sample value in the domain of f(x) in that direction. This can be illustrated by solving the equation $\sqrt{(x + \ln x)} = 0.5$ for x. The left side of the equation is not defined for $x < -\ln x$ (see Fig. 2). With initial guesses of 1 and 2, the solver repeatedly samples within this undefined region, eventually succeeds, and reports $x = 0.662195081464$ as the approximate solution.

If the value of f(x) does not change sign, but increases in magnitude during secant extrapolation, the search bound in the direction of search is reset to the sample value for x where the magnitude of f(x) increases. The solver then employs quadratic interpolation and selects the value where the fitted quadratic expression has minimum magnitude as the next sample value. Depending upon the position of this fitted point, the solver resumes modified secant extrapolation in the same or opposite direction. Each time quadratic interpolation is employed, a counter is decremented and tested. Each time secant extrapolation finds a value for f(x) with decreasing magnitude, that counter is reset to seven. When the decremented counter value is zero, the solver returns the last sample value as an approximate solution and displays the values of the left and right sides of the equation for that solution.

If the user immediately asks the calculator to solve the equation again for the same variable, the iterative solver uses initial guesses in the region of the last sample value. Hence approximations to local f(x) magnitude minima can be found by repeatedly solving for the same variable. However, the search procedure is designed to find zeros—not local magnitude minima. This case can be illustrated by solving the equation $x^2 + x = -1$ for x with initial guesses 0 and 1 (see Fig. 3). The solver reports the approximate solution, $x = -4.99999994899E-1$, with the values of the

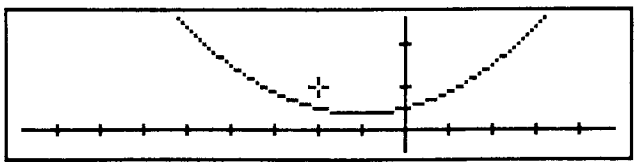


Fig. 3. *HP-28C plot of $f(x) = x^2 + x + 1$.*

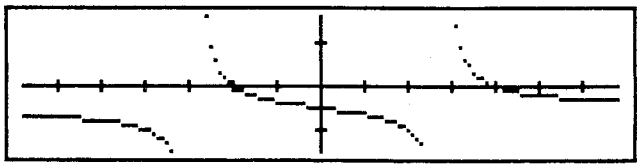


Fig. 4. *HP-28C plot of $f(x) = x/(x^2 - 2) - 1$.*

equation's left and right sides for that solution.

If f_1 and f_2 have opposite signs, or if during extrapolation their values change sign, the solver sets the search bounds to those sample values resulting in the values of f_1 and f_2 having opposite signs and begins attempting to narrow the interval bracketing the change of sign. This process may employ an adaptive combination of modified bisection and secant, cubic, and hyperbolic interpolation to obtain a sequence of sample values. For each iteration where the sample value is within the domain of f(x), one of the search bounds is reset to that value and the interpolation process continues. The process continues until it finds one of the following cases:

- A solution
- Neighboring values x_1 and x_2 bracketing a sign change in f(x)
- A value out of the domain of f(x).

The first case can be illustrated by solving the equation $x^2+x = 6$ for x with initial guesses 0 and 1. The iterative solver reports the positive solution $x = 2$. If the equation is immediately solved again for x, the solver again reports the solution $x = 2$.

The second case occurs when the equation has a solution that is not representable in the HP-18C's 12-digit floating-point format. (The set of 12-digit numbers includes 0, $-1.00000000000 \times 10^{-499}$ to $-9.99999999999 \times 10^{499}$, and $1.00000000000 \times 10^{-499}$ to $9.99999999999 \times 10^{499}$.) It can also occur if the function f(x) is discontinuous between two adjacent machine-representable values. In any event, the solver returns the value of x_1 or x_2 that gives a minimum f(x) magnitude as the solution. It stores the other value in a dedicated location such that if the user immediately solves again for the same variable, x_1 and x_2 are used as initial guesses. The solver also displays the values of the left and right sides of the equation for that solution if either x_1 or x_2 is the only value sampled in the interpolation process, or if the process strongly suggests that the result represents a pole.

For example, with initial guesses 0 and 1, the solver returns the approximate solution $x = 1.30277563773$ to the equation $x^2+x = 3$. If the solver is immediately reinvoked it displays the values of the equation's left and right sides for the same approximate solution. For the first try, the solver is able to make some progress from the initial guesses and the data does not strongly suggest a pole. For the second attempt, the solver is unable to progress beyond its initial guesses so it returns the values of the left and right sides as a warning that the equation is not exactly satisfied.

If the solver is applied to the equation $x/(x^2-2) = 1$ with initial guesses 1 and 1.5, it returns the approximate pole $x = 1.41421356238$ and displays the values of the left and right sides of the equation for that solution. The solver in this case was able to make some progress from its initial guesses, but the process strongly suggested that the result was near a pole (see Fig. 4).

In the third case the iterative solver splits the current search region, which brackets a change of sign in the value of f(x), at the out-of-domain value. We then have two intervals, x_1 to g_1 and g_2 to x_2, where initially $x_1<g_1 = g_2<x_2$. The points g_1 and g_2 will later be adjusted such that the interval between g_1 and g_2 defines a gap within which the function f(x) is presumed to be undefined. The solver alternately samples values in the left and right subintervals using a modified bisection search. Each time, if the sampled value is out of the domain of f(x), the appropriate g bound is reset to that value and the iteration continues with a wider gap between g_1 and g_2. If the value of f(x) at that sample value has the same sign as the value of f(x) at the corresponding x bound, that bound is reset to that sample value and the iterations continue with a narrower outer interval $[x_1, x_2]$. The process continues until it either finds a solution, it finds a value for x where the sign of f(x) is the opposite of the sign at the corresponding x bound, or it exhausts both subintervals $[x_1,g_1)$ and $(g_2,x_2]$. If the solver finds a value for x where the sign of f(x) is the opposite of the sign at the corresponding x bound, the solver discards the other interval and resumes narrowing the region around the change of sign in f(x) as above.

This case can be illustrated by solving the equation $\sqrt{x/(x+0.3)} = 0.5$ for x with initial guesses -1 and 2 (see Fig. 5). The left side of this equation is not defined for x in the interval from -0.3 to 0. With these initial guesses the solver first samples on either side of this interval and then in this interval, triggering the gap-narrowing process just described. Eventually the solver exits that process and finds the solution $x = 0.1$.

If the solver exhausts both subintervals it returns the value of x_1 or x_2 giving minimum f(x) magnitude as an approximate solution. This case can be illustrated by attempting to solve the equation $(x/(3x-1))^3 = 1$ for x with initial guesses 0.3 and 0.4 (see Fig. 6). The solver stores the other value in a dedicated location such that if the user immediately solves again for the same variable, x_1 and x_2 are used as initial guesses. The solver also displays the values of the left and right sides of the equation for that

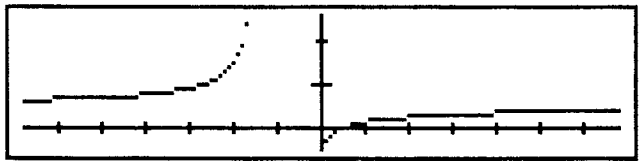


Fig. 5. *HP-28C plot of $f(x) = \sqrt{x/(x + 0.3)} - 0.5$.*

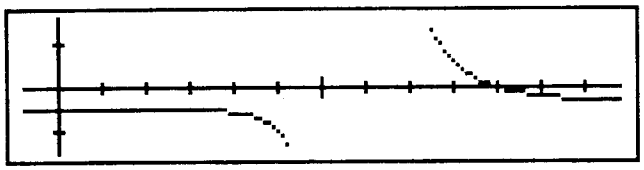


Fig. 6. *HP-28C plot of* $f(x) = (x/(3x-1))^3 - 1$.

solution as a warning that the solution is not exact.

HP-28C Iterative Solver

The HP-28C's iterative solver assumes a higher level of sophistication on the part of the user. It also searches for a real solution to an equation or a real zero of an expression. It differs from the HP-18C version only in the manner that the user specifies initial guesses, how the solver displays current iterates, and the solver's termination display.

The HP-28C uses the initial contents of the unknown to obtain up to three initial guesses, with zero as a default. The user specifies one initial guess by storing a real or complex number in the unknown. The HP-28C takes the real part of a complex number as an initial guess. The user can specify one, two, or three distinct initial guesses by including those guesses in a list and storing that list in the unknown. The HP-28C uses up to the first three distinct real numbers or real parts of complex numbers in the list. The reason for handling complex numbers in this way is to facilitate the user's specifying initial guesses obtained by digitizing points from plotted equations.

The iterative solver is faster if it does not need to display iterates, so by default the HP-28C solver does not do so. However, the user can trigger the display of current iterates by pressing any key other than ATTN. Additional pressing of such keys has no effect and the solver purges the key buffer when it terminates. Pressing ATTN always aborts the iterative solver, which then returns a list of the three current iterates on the display stack and stores the list in the unknown.

If the HP-28C cannot obtain at least two values in the domain of f(x) using the initial guess(es) of the unknown, then it leaves the unknown unchanged and displays Bad Guess(es). If the HP-28C cannot obtain a slope, then it leaves the unknown unchanged and displays Constant. Otherwise, the HP-28C overwrites the initial contents of the unknown during the search process. When the search is complete, the solver returns a message and an exact or approximate solution on the display stack and stores the solution in the unknown. The HP-28C displays the message Extremum if it exhausts the search interval without finding a change of sign. If it finds a change of sign but not an exact numerical solution, it displays Sign Reversal. If it finds an exact numerical solution, it displays Zero.

The solver application menu has labeled softkeys that can be pressed to evaluate the left and right sides of the current equation for the current values of the equation's variables. The user can use these keys to inspect the quality of a solution in more detail.

Acknowledgments

Charles Patton prototyped the HP-18C direct solver. Prof. W. M. Kahan of the University of California at Berkeley authored the original iterative solver algorithm common to both the HP-18C and the HP-28C.

References

1. W.M. Kahan, "Personal Calculator Has Key to Solve Any Equation f(x) = 0," *Hewlett-Packard Journal*, Vol. 30, no. 12, December 1979.
2. *HP-18C Business Consultant Owner's Manual*, Hewlett-Packard Company, Publication 00018-90057, October 1986.
3. *HP-28C Reference Manual*, Hewlett-Packard Company, Publication 00028-90051, January 1987.

Electronic Design of An Advanced Technical Handheld Calculator

by Preston D. Brown, Gregory J. May, and Megha Shyam

THE DESIGN of an advanced handheld calculator such as the HP-28C requires solutions of some special problems: how to package the system in a limited space, how to provide power from three small batteries for six months, how to keep the cost down, and how to release the new design in less than 18 months. These challenges were met by designing three custom CMOS ICs, packaging the electronics using chip-on-board and surface-mount technologies, and using powerful design aids. The HP-28C includes a four-line liquid-crystal display (LCD), 128K bytes of ROM, 2K bytes of RAM, a clock, and an infrared transmitter for sending data to an optional detached printer. The HP-18C Business Consultant contains the same electronics, but only one ROM.

The electronic design (Fig. 1) of the HP-28C provides a 20× improvement in computational speed over its predecessor, the HP-15C. Custom ICs and custom packaging were required to achieve this functionality on a small circuit board measuring 3 by 1.5 inches.

A hybrid board design (see article on page 25) is used

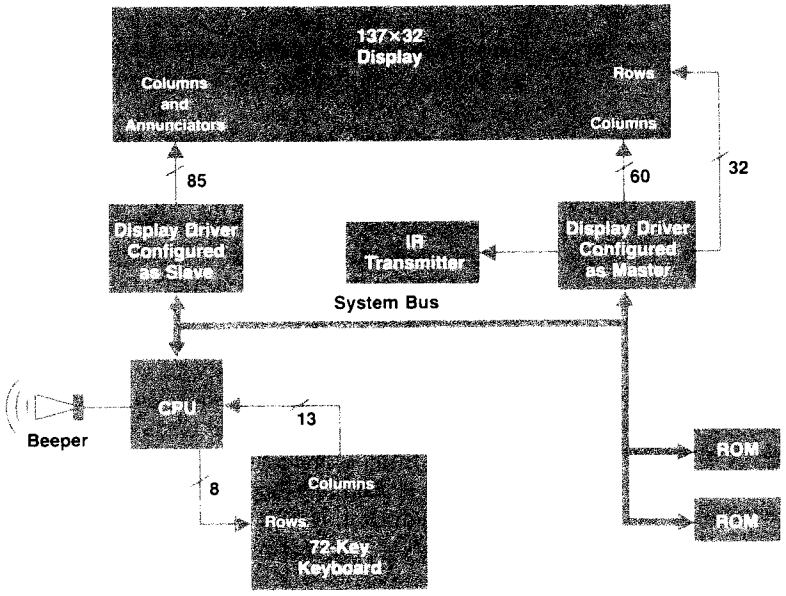


Fig. 1. *System block diagram.*

for the entire system. Two display drivers and the CPU are bonded directly to the front of the printed circuit board using 263 bonds. Two ROMs in flatpacks and the rest of the discrete components are placed on the back of the board. Pressure contacts are made from the board to the LCD on the front, and from the board to the keyboard on the back. The use of chip-on-board technology has proven to be reliable and cost effective.

Custom Microprocessor

Commercially available microprocessors have a number of limitations that make them unsuitable for use in a calculator. They require too much power, many support chips, regulated supplies, or a wide system bus which takes up too much room on a printed circuit board. Hence, a custom microprocessor was developed for the HP-28C to avoid these problems.

The processor used in the earlier HP-71B Handheld Computer[1] was an excellent starting point for the design; this processor already met the low-power and interconnect requirements, but it would not run at 3V (the minimum battery voltage). By porting the design into the newer, smaller CMOSG process, the part price and the power supply requirements were reduced and the speed was increased. At the same time, new instructions were added to improve data manipulation and the interrupt structure was enhanced.

The instruction set of the processor is highly optimized for binary-coded decimal operations on both integer and real numbers. The main working registers in the processor are 64 bits long and are broken into three fields: the exponent, the mantissa, and the sign fields. Individual nibbles or bytes of the registers can be handled independently.

The processor has 16 input pins and 12 general-purpose output pins, some of which are used to scan the keyboard. Most of the work of scanning the keyboard is the responsibility of the firmware including the scan sequence, key debouncing, and type-ahead buffer. Hardware is kept simple.

Bus Definition

To reduce printed circuit board area, the bus width must be limited. A four-bit multiplexed command and data bus may seem to be an extreme solution, but is necessary to save space. The challenge is to maintain reasonable performance with a four-bit bus. Each IC in the system maintains its own copy of the 20-bit program counter (PC) and a data pointer (DP) which are only broadcast on the bus when necessary. After a read operation to an address pointed to by the PC, each IC automatically increments its copy of the PC. Therefore, the PC need only be updated if a branch is taken. In this case, the PC must be reloaded. The command LOADPC is placed on the bus followed by the five nibbles of the new address. The other fifteen bus commands include starting reads and writes to the address pointed to

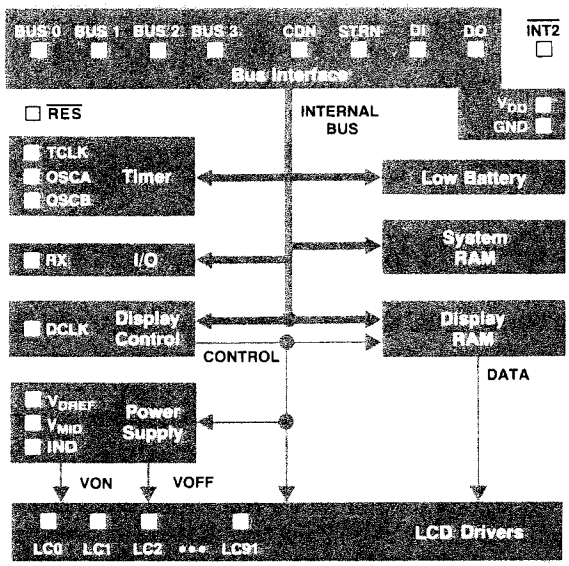


Fig. 2. *Layout and architecture of display driver chip.*

by the PC or DP, loading the DP, and resetting and configuring the system.

Another feature of this bus definition is soft configuration, which allows the memory space to be allocated as desired. A daisy-chain signal is routed from one IC to the next. If an IC has not yet been configured, it drives its daisy-out (DO) line low. When its daisy-in (DI) line is high, the IC responds to identification and configuration commands which place it in the address space. Once configured, the IC's DO line goes high so that the next chip in the chain can be configured.

The bus consists of 10 pins: data (pins 0 to 3), CDN (signals if the transfer is a command or data), STRN (system strobe), DI, DO, V_{DD}, and GND. The bus supports data transfer at up to one megabyte/second. However, in the HP-28C the transfer rate is limited to 325 kilobytes/second because of other limitations.

Display Drivers

The liquid-crystal display requires 184 drivers. Since there are too many pins to be driven by a single IC, two identical display driver ICs (Fig. 2) are used, each driving 92 lines. Each driver IC also requires 20 additional pins for a total of 112 pins per IC. Other system needs are also integrated onto the display drivers; the CPU and ROM are the only features that would not fit because of area limitations.

The 32-way multiplexed (see waveforms in Fig. 3) liquid-crystal display requires up to nine volts peak-to-peak to operate. This presented some difficulty since the CMOSG process allows only seven volts maximum because of two problems. First, the process' polysilicon field threshold runs around 12V and there would be significant subthreshold conduction at nine volts. Second, although a p-channel transistor can handle the high electric fields produced, an n-channel FET would only last a short time before it was damaged. To live within these constraints, restricted layout rules and circuit designs were developed to allow nine-volt operation. The layout rule changes included increasing the minimum gate length, increasing the polysilicon-to-diffusion spacing, eliminating polysilicon p-well crossings, and not allowing two transistors to share the same gate polysilicon. By making use of a supply level already needed for the display, V_{MID} (1.8V), two n-channel

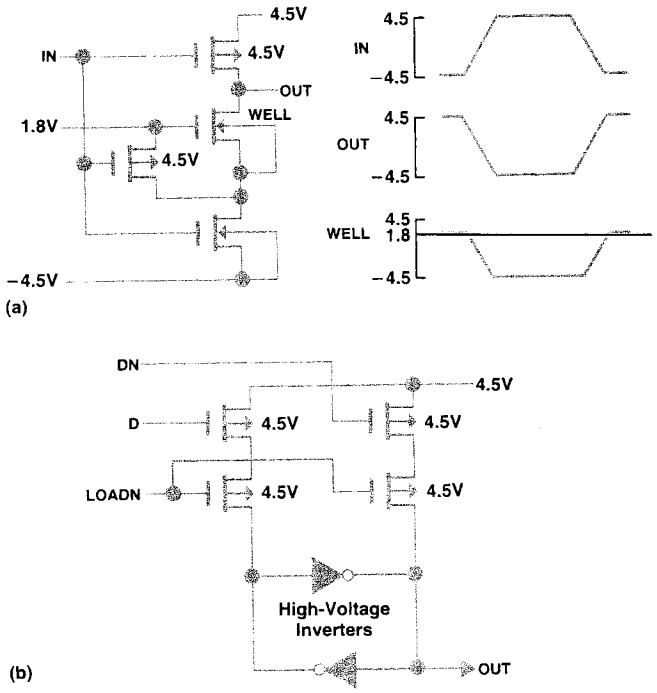


Fig. 4. *(a) High-voltage inverter. (b) Nine-volt interface circuit for supplying display drivers.*

devices can be placed in series and biased to maintain a voltage drop of less than seven volts across each of them (see the high-voltage inverter in Fig. 4a).

While the majority of the system is powered by three N-cell batteries (4.5V), the display drivers need nine volts. Therefore, an interface circuit was necessary to provide ±4.5V. The high-voltage inverter could allow a 0-to-4.5V logic input to produce a ±4.5V output, but current drains would be high since both the pull-up and pull-down transistors are on when the input is at ground. However, by incorporating two high-voltage inverters into a latch (Fig. 4b), the full voltage swing is placed across the inputs of both inverters, and no dc current flows.

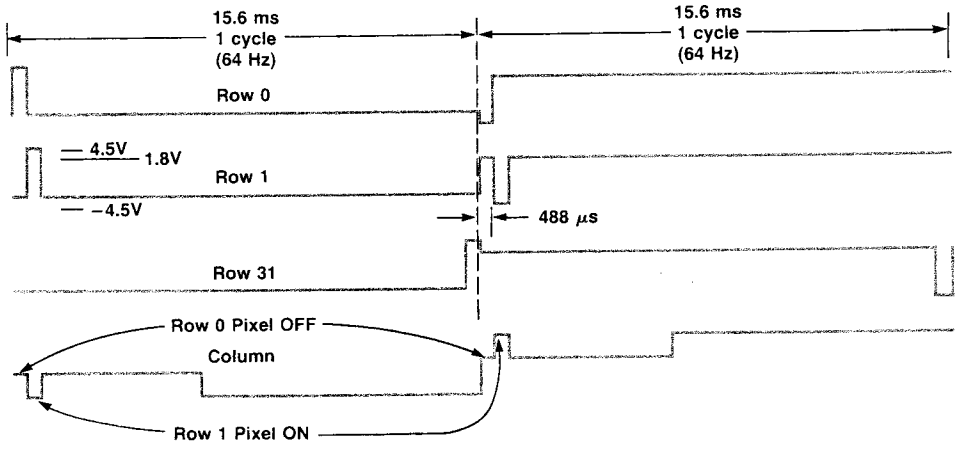


Fig. 3. *Multiplexed waveforms for driving liquid-crystal display.*

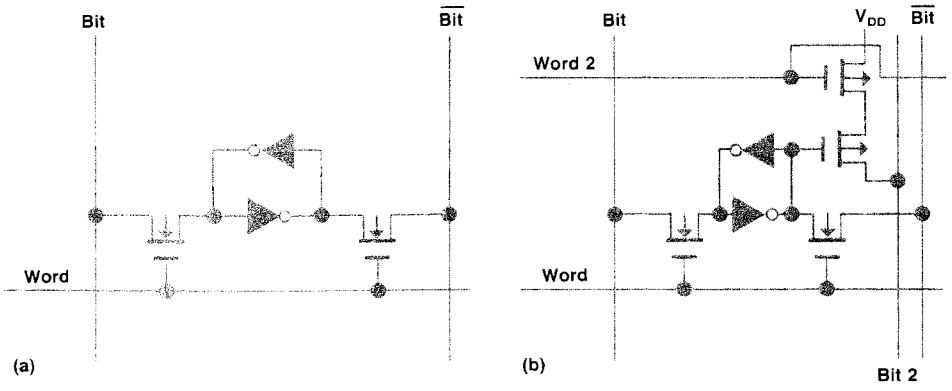


Fig. 5. *Static RAM cells. (a) Traditional static RAM. (b) HP-18C and HP-28C display RAM cell.*

Programmable Switching Supply

The power supply uses only three discrete components and a 50-to-150-kHz clock signal to generate a negative display supply (V_{DREF}, −4.5V) from a 2.7-to-6V input. The other display voltages are generated by buffering voltages from a resistor divider strung between V_{DD} (4.5V) and V_{DREF}. The negative supply is adjusted versus ambient temperature by comparing the voltage from a string of three parasitic npn transistors to the voltage from a switched capacitor divider driven by V_{DREF}. A 5-bit register which directly varies the ratio of this divider gives the user control of the display contrast by altering the negative supply voltage with respect to V_{DD}.

System Functions

Other features necessary to complete the system are integrated onto the display driver (see Fig. 2). A two-port RAM consisting of ninety-two 32-bit words is used for a display bit map. The read-only second port is formed by the addition of a second word line, a second bit line, and two transistors to the basic static RAM cell (see Fig. 5). Each display driver also provides 1K bytes of system RAM.

In the low-battery detection circuit, the supply voltage V_{DD} is divided down and compared to a bandgap reference. The reference produces 1.3V±15 mV over process variations and the operating temperature range of −30 to +75°C.

Display control logic handles the display refresh and synchronization of multiple chips.

A 32-bit crystal-controlled timer provides a real-time clock and other timing functions.

A flexible I/O pin allows TTL-level serial communications and several other I/O possibilities. In the HP-18C and HP-28C Calculators, this pin drives an infrared LED transmitter for sending data to an optional printer with an infrared receiver. The timer and I/O sections provide minimal hardware support for these features; as much of the complexity as possible is handled by the firmware.

512K-Bit ROM

The third custom IC used in the HP-28C is a 512K-bit ROM. One or two ROMs in flatpack packages are soldered to the back of the hybrid circuit board. The ROMs are not bonded directly to the board for three reasons. First, by placing the ROMs in separate packages the HP-28C and all language versions of the HP-18C can be produced by simply loading the boards with a different ROM. Second, that much ROM would consume a large amount of tester memory and is best tested separately. Third, the CPU and display drivers require most of the room on the front of the board, and directly bonding chips to both sides of the board is not practical.

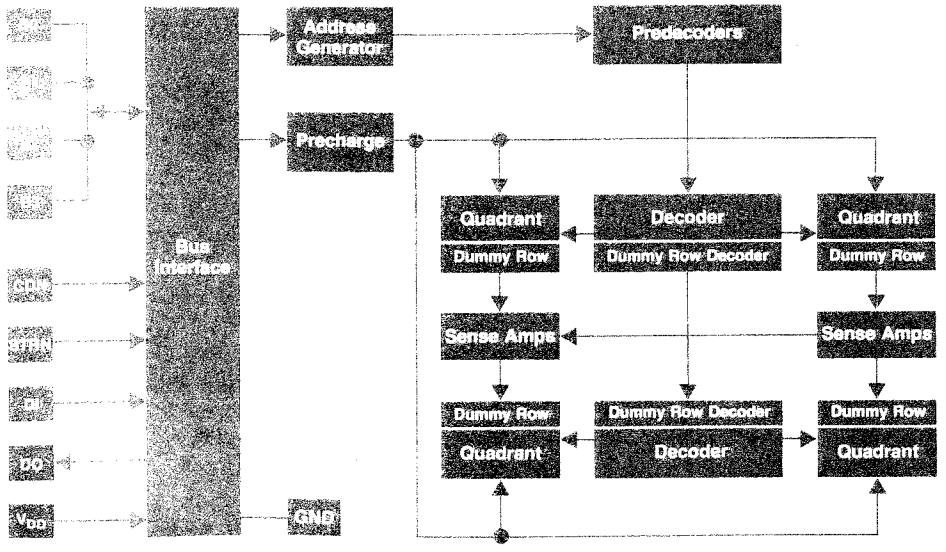


Fig. 6. *Architecture of 512K-bit ROM chip.*

Early in the development cycle it became apparent that a high-density CMOS process was essential to help keep the cost of the product at realistic levels. Hence, this ROM chip was designed using a third-generation CMOS process developed at HP's Northwest Integrated Circuits Division.

The chip architecture (Fig. 6) consists of four 128K-bit quadrants, each organized in 512 rows and 256 columns. The data from each quadrant is read four nibbles at a time. Considerable design effort went into minimizing power supply drain by the ROM in both the operating and standby modes. Our design approach incorporates decoded virtual ground drivers so that only part of each quadrant is active at any time. The use of virtual ground drivers minimizes precharge current contribution to the operating current.

Our choice for the ROM core cell is the so-called X core, where the polysilicon word line snakes around the island line at 45° angles. The traditional diffusion resistance to ground is not present in this design, which eliminates periodic ground bus lines. Data from the ROM core is sensed by special differential sense amplifiers that detect the difference between the selected cell and a dummy cell. The ROM is island programmable, which implies that a one or zero is detected by the presence of an island completing the transistor. The design calls for the ROM to operate from 3.0 to 5.5 volts with 200-ns access time at 65°C.

The ROM interfaces to the CPU using the 4-bit data bus and two control lines. For proper operation, the chip needs only 11 pads. The chip was designed to be configurable in the address space of the CPU either by hard configuration (i.e., the address is predefined and set) or soft configuration (i.e, the address can be mapped anywhere in the CPU address space). The interface of the ROM core with the CPU consists of a command decoder and two 20-bit program counter and data pointer registers.

The CMOS process development played a key role in the availability of the ROMs. Some of the principal characteristics of this new CMOS process are:

- It is an n-well process, as contrasted to previous p-well processes.
- It uses p-type epitaxial silicon on a p+ substrate instead of a monocrystalline silicon structure.
- 5× optical steppers are used for all critical lithography levels.
- The metal interconnection layers (first and second) have a linewidth-spacing pitch of 4.0 μm.
- The polysilicon lines are drawn 2.8 μm wide and are placed at least 1.2 μm apart.
- The islands that define p-channel or n-channel transistor widths are 2.8 μm wide.
- The n-channel and p-channel threshold voltages are 0.75V and symmetrical.
- The effective size of a minimum-geometry device is 1.8 μm wide and 1.3 μm long.
- The maximum operating voltage is 5.5V.

ESD and EMI Design Considerations

From the very beginning of the project, the design goal for electrostatic discharge (ESD) protection was to eliminate any breakdowns through the case with the calculator placed on a reference ground plane for discharges up to 25 kV. Hence, the emphasis on sealing the product with an RTV compound was partly because of ESD requirements. However, the problem with that design philosophy is that any one entry point can eliminate all chances of success. In this case, the weak point was the battery door. Interestingly enough, the observed arc path was from the battery door to the battery case and over through the electronics to the CPU key lines. The reason for this was simple—the keyboard provides a larger capacitance to reference ground than anything else in the product (i.e., the highest charge path is through the electronics). The solution is to isolate the keyboard using a series resistance and to provide an alternate path for this charge with an appropriately placed ground plane. The keyboard-to-ground capacitance is reduced by inserting a grounded metal shield between the keyboard and the reference ground. This design at the same time provides an alternate, more desirable charge path to this internal ground plane because of its predominant capacitance to the reference ground. This technique has proven to be quite successful in previous projects.[2]

Electromagnetic interference (EMI) generation was not a problem, mainly because of an early emphasis on proper printed circuit board layout. Possible RF sources are eliminated by minimizing the physical loop areas created by the signal and ground return paths, and by laying out the power and ground lines first on the hybrid. This is an extremely quiet product, considering its speed capabilities.

Tools

An aggressive schedule was met with this project. Since the CPU was a redesign, the display driver and ROM were the most critical IC designs. Three months were required for design, schematic entry, and simulation. Our first prototypes, built six months later, were fully functional.

The Hierarchial Custom Design System, developed at HP for in-house use, is a highly integrated set of tools running on HP 9000 Computers. The schematic capture system produces a net list which is fed to the circuit (HP Spice) or logic simulator. The logic simulator handles CMOS designs including bidirectional transmission gates and circuit fights created, for example, when overdriving cross-coupled inverters to load a latch. A high-level, Pascal-like language is used to develop all the test patterns. This language can then be compiled for the logic simulator or for the IC test system. Often the output of the simulator is used to create the production test patterns, but recompiling the high-level language has three benefits. First, changes in the test patterns can be implemented quickly, without having to resimulate the entire IC. Second, the test patterns are well documented. Finally, the features of the language reduce the time required to develop test patterns for both simulation and production testing.

Standard cells were used whenever possible, but some layout was done manually while the RAM was drawn by a module generator. All design rule and electrical errors in the layout were caught with a hierarchical design rule check (DRC) and schematic compare program. One error was caught in the module generator's work. With thousands of dollars of mask charges and months of debugging time at stake, correct by construction is a nice goal, but it cannot beat correct by double checking. The DRC and compare

program ran quickly and produced concise listings of any errors.

References

1. J.P. Dickie, "Custom CMOS Architecture for a Handheld Computer," *Hewlett-Packard Journal*, Vol. 35, no. 7, July 1984.
2. G.J. May, "Electrostatic Discharge Protection for the HP-75," *Hewlett-Packard Journal*, Vol. 34, no. 6, June 1983.

Authors

August 1987

4 Business Consultant

Susan L. Wechsler

Susan Wechsler has been with HP since 1980 and worked on the operating system for the HP-71B Handheld Computer before contributing to the design and implementation of the operating system and user interface for the HP-18C Calculator. She's the author of two technical papers related to product quality and to the HP-71B, and is named coinventor on a patent application for calculator software. Born in Burbank, California, Susan attended California State University at Long Beach, completing her BA in mathematics in 1979. She and her husband and baby live in Corvallis, Oregon. When she's not working on home remodeling projects, she enjoys sewing and gardening.

11 Technical HP-IL Calculator

William C. Wickes

With HP since 1981, Bill Wickes was R&D project manager for the HP-28C Calculator and contributed to the operating system design for the HP-18C. He has developed ROMs for several other products, including the HP-41C Calculator and the HP-75C and HP-71B Computers. He's named inventor on two patent applications related to the operating system for the HP-28C. Born in Lynwood, California, he studied physics at the University of California at Los Angeles (BS 1967) and at Princeton University (PhD 1972). Before coming to HP he was an assistant physics professor at both Princeton and the University of Maryland. A member of the American Astronomical Society, he has published six papers on observational cosmology. He has also written and published a book on synthetic programming on the HP-41C. Bill lives in Corvallis, Oregon, is married, and has two children. He's active in the Boy Scouts and likes sailing and volleyball.

17 Mechanical Design

Judith A. Layman

Judi Layman studied mechanical engineering at Montana State University and received her BSME degree in 1981. After joining HP the same year, she contributed to the design of an expansion pod for the HP-75C Portable Computer and continued with it into production. More recently she worked on the keyboard and flexible interconnect design for the HP-18C and HP-28C Calculators. She's named coinventor on a patent application related to the articulating hinge for the HP-18C and HP-28C. A resident of Corvallis, Oregon, Judi enjoys running, bicycling, volleyball, backpacking, textiles, and travel.

Mark A. Smith

Mark Smith joined HP in 1980, the same year he graduated with a BSME degree from California Polytechnic State University at San Luis Obispo. He has worked on the design and production of a number of portable computer and calculator products, including the HP-10, HP-15C, HP-16C, HP-18C, and HP-28C Calculators, and the HP-75D Portable Computer. He's also the author of an article on the flex circuit that is used in the HP-18C and HP-28C. Born in Livermore, California, Mark now lives in Corvallis, Oregon. He's married and has two sons. He's finishing work on the home he built and enjoys soccer and skiing.

21 Symbolic Computation

Charles M. Patton

Charlie Patton earned a PhD degree in mathematics from the State University of New York at Stony Brook in 1977 and was a mathematics professor at the University of Utah before coming to HP in 1982. He's a specialist in mathematical software and algorithm development and has worked on a number of ROMs for the HP-75C and HP-71B Computers. He also contributed to the development of the HP-18C and HP-28C Calculators. He's author or coauthor of five papers in mathematics and mathematical physics and is a member of the American Mathematical Society and the American Association for the Advancement of Science.

25 Hybrid Printed Circuit

Chong Num Kwee

Chong Kwee was born in Kulai, Malaysia and attended the University of Dundee. His BSc degree in mechanical engineering was granted in 1981. He joined HP upon graduation and is now an R&D engineer in the Singapore IC Operation. His contributions to the HP-18C and HP-28C were the wirebonding process and transfer of the assembly process to Singapore. Chong is married and is expecting his first child this year.

Cornelis D. Hoekstra

With HP since 1977, Casey Hoekstra has held a number of process and test engineer positions. He has worked on thermal printheads, plasma deposition, photolithography, and CMOS IC yield and reliability. Born in Schyndel, The Netherlands, he served as a medic in the U.S. Army and then studied physics at the University of Oregon. His BS degree was granted in 1976 and his MA in 1977. Casey and his wife and two daughters live in Corvallis, Oregon. Mountain climbing, hiking, and gardening head his list of leisure activities.

Robert E. Dunlap

Bob Dunlap started at HP in 1976 as a wafer fabrication engineer and is now a project leader for R&D packaging. He has been responsible for developing hybrid PC boards for the HP-41C, HP-85A, HP-86A, HP-75C, and HP-71B Computers and for the HP-18C

and HP-28C Calculators. Born in La Jolla, California, he completed work for his BS degree in chemistry from San Jose State University in 1966. He was a wafer fabrication engineer before coming to HP. Bob is married, has four children, and lives in Corvallis, Oregon. Outside of work he enjoys skiing, boating, and backpacking.

Bruce R. Hauge

A graduate of Oregon State University, Bruce Hauge completed work for his BS in chemical engineering in 1979. He worked as an R&D engineer for Cordis-Dow Company before coming to HP in 1981. An IC packaging engineer, he has worked on quad packs for HP-10 Series Calculators, printed circuit board hybrids for several handheld computers and calculators, and pin-grid array packages. An Oregon native, Bruce was born in Salem and now lives in Corvallis. He's married and has two daughters. His outside interests include church activites, personal computers, playing guitar, and recording music.

Paul R. Van Loan

With HP since 1977, Paul Van Loan specializes in IC package development and is a project manager for the company's Northwest IC Division. His first project at HP involved the development and production of LCDs. Later, he contributed to work on the HP-10C, HP-15C, HP-16C, and HP-41C Calculators. His work on ceramic and refractory materials and on thick-film resistor formulations has resulted in three patents. He's also the author of over 25 papers in the fields of mineralogy, materials science, hybrid microelectronics, and display technology. Born in Toronto, Canada, Paul earned a BS in earth sciences in 1957 and an MS in mineralogy in 1958 from the University of Toronto. He continued his studies at McGill University, from which he received his PhD degree in crystallography in 1968. His professional experience before coming to HP included the development of LCDs and flat panel displays and thick-film and thin-film hybrids. He and his wife and two children are residents of Corvallis, Oregon. His hobbies include gardening, music, running, hiking, and reading to his children.

30 Equation Solver

Paul J. McClellan

An Oregon native, Paul McClellan received his BS in physics and mathematics in 1974 from the University of Oregon and his PhD in statistics in 1984 from Oregon State University. Before coming to HP in 1979 he developed software for digital measurement system applications and for robot calibration. At HP, he has contributed to the development of a series of calculators and handheld computers, including the HP-15C Calculator, HP-71B Computer, HP-18C, and HP-28C. He's coauthor of a May 1983 HP Journal article on the HP-15C. Paul was born in Salem and lives in Corvallis. He's a member of Corvallis Mountain Rescue and enjoys Nordic and alpine skiing, rock climbing, mountain climbing, and reading.

34 Handheld Calculator Electronics

Megha Shyam

A native of India, Megha Shyam received his BSEE degree from the Indian Institute of Science in 1961 and his PhD in electrical engineering from Stanford University in 1967. He worked for several companies, including Fairchild Semiconductor, Bell & Howell, and Data General, before joining HP in 1977. He has contributed to the design of integrated circuits for the HP-85B and HP-75C Computers, was project leader for the card reader for the HP-71B Computer, and designed portions of the HP-18C and HP-28C. He's the author of 14 papers related to diodes, GaAs characterization, and solid-state devices, and his work in these fields has led to four patents. He's also an adjunct professor at Oregon State University and teaches courses on VLSI design. A resident of Lewisburg, Oregon, Megha is married and has three children. He's active in the Baha'i faith and enjoys meeting people, reading, and listening to music.

Preston D. Brown

Preston Brown joined HP in 1981 and has contributed to the development of the HP-IL module for the HP-71B Handheld Computer, to the ThinkJet Printer, and to the IC design and system design for the HP-41C Handheld Computer. He was responsible for the display driver IC and was a production engineer for the HP-18C Calculator. He has written two papers for internal HP conferences on IC artwork verification and on a display driver. Born in Pensacola, Florida, Preston studied electrical engineering at the University of Florida (BSEE 1981). He's now a resident of Philomath, Oregon and likes skiing and hiking.

Gregory J. May

Greg May was born in Dayton, Ohio and earned a BSEE degree in 1980 from the University of Tennessee at Knoxville. After coming to HP the same year, he worked on the power supply and memory board for the HP-75C Portable Computer and later was a production engineer for the product. He has also worked on the HP-94 Handheld Computer and on bar code wands. His contribution for the HP-18C and HP-28C Calculators was the analog design for the display driver IC and system ESD and EMI. Greg and his wife and daughter are residents of Corvallis, Oregon. He's an amateur radio operator (KB40X) and woodworker and recently restored a 1972 Fiat 124 Spyder. He's also working toward an MBA degree from Oregon State University.

Hewlett-Packard Company, 3200 Hillview Avenue, Palo Alto, California 94304

HEWLETT-PACKARD JOURNAL

August 1987 Volume 38 • Number 8

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

Bulk Rate
U.S. Postage
Paid
Hewlett-Packard
Company

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.

5953-8563