

HEWLETT-PACKARD

Journal

NOVEMBER 1998

Technical Information from the Hewlett-Packard Company

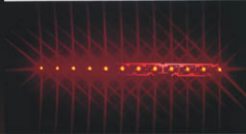
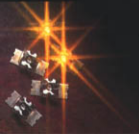
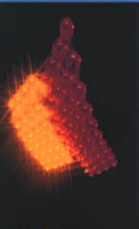




table of contents

November 1998,
Volume 50, Issue 1

Articles

1

HP SnapLED: LED Assemblies for Automotive Signal Lighting

by James W. Stewart

2

OTDR APIs Enable Customers to Build Their Own Systems

by Torsten Born and Peter Thoma

3

Updating a UNIX Application Suite for the Windows NT World

by Thomas W. Hutchinson and Ronald R. Derynck

4

Integrating Real-Time Systems with Corporate Information Systems

by Ronald R. Derynck and Thomas W. Hutchinson

5

New Approaches to Creating and Testing Internationalized Software

by Harry J. Robinson and Sankar L. Chakrabarti

6

Comparison of Finite-Difference and SPICE Tools for Thermal Modeling of the Effects of Nonuniform Power Generation in High-Power CPUs

by Jeffrey L. Deeney and C. Michael Ramsey

7

A Low-Complexity, Fixed-Rate Compression Scheme for Color Images and Documents

by Nader Moayeri

HP SnapLED: LED Assemblies for Automotive Signal Lighting

James W. Stewart

Decreased packaging cost and improved performance have helped LEDs gain acceptance as light sources in automotive applications such as signal lighting. An assembly technique is described that allows the creation of thin taillamps that can be customized to conform to the shape of a particular vehicle.

Because of developments in high-brightness LED materials and high-power LED packaging, LEDs are more frequently being used in automotive signal lamps. A new product, HP SnapLED, combines the latest LED technology with a three-dimensional assembly technique to create a thin taillamp that conforms to the shape of the vehicle (see **Figure 1**).

Background

The first LED signal lamp appeared on a passenger vehicle in 1985. The 1986 Nissan 300ZX center high mounted stop lamp (CHMSL) used 72 absorbing substrate AlGaAs, 5-mm LEDs assembled on a printed circuit board. From this start, LEDs have gained acceptance in the automotive industry because of a decrease in the cost of LED light sources. These cost reductions have come from an increase in the performance of the LEDs, which results in a reduction in the number of LEDs required for a function. As the number of LEDs required goes down, LED packaging cost and assembly cost are proportionally reduced. **Figure 2** shows the number of LEDs required to meet the CHMSL specification from 1985 to 1997.

The main factors that have contributed to the increased performance of LEDs are more efficient LED materials (see **Figure 3**) and improved LED packaging.

Since their introduction into automotive applications, LED efficiency has increased over 500%. In 1994, HP introduced a new LED package designed specifically for automotive lighting. This new package, called *SuperFlux*, was



James W. Stewart

An automotive product development manager at the HP Optoelectronics

Division, James Stewart is responsible for electronic packaging and automotive lighting. He joined HP in 1990 after graduating from the University of California at Santa Barbara with a BS degree in mechanical engineering. James was born in Santa Maria, California.

Figure 1

Automobile taillamps. (a) Contains LED technology. (b) Contains incandescent lamps.

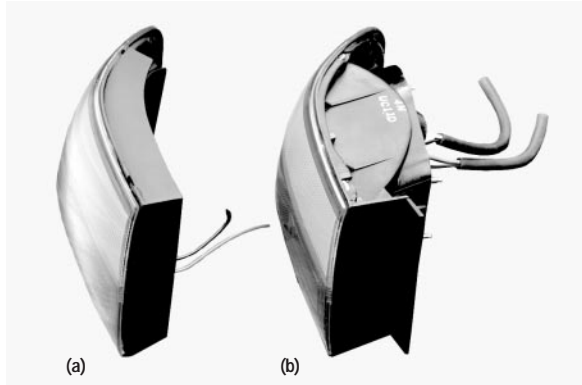


Figure 2

Number of LEDs required for center high mounted stop lamp (CHMSL) applications from 1985 to 1997.

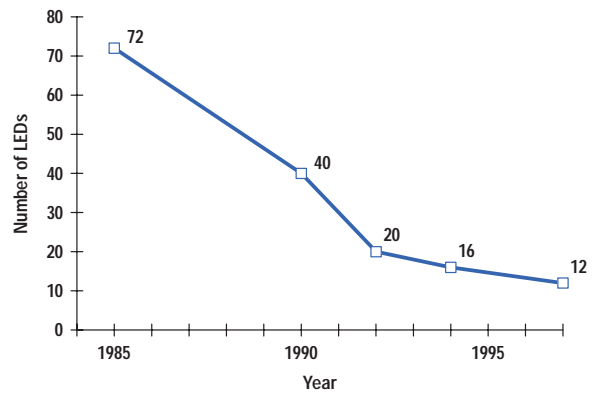


Figure 3

Time evolution of red and amber LED technology.

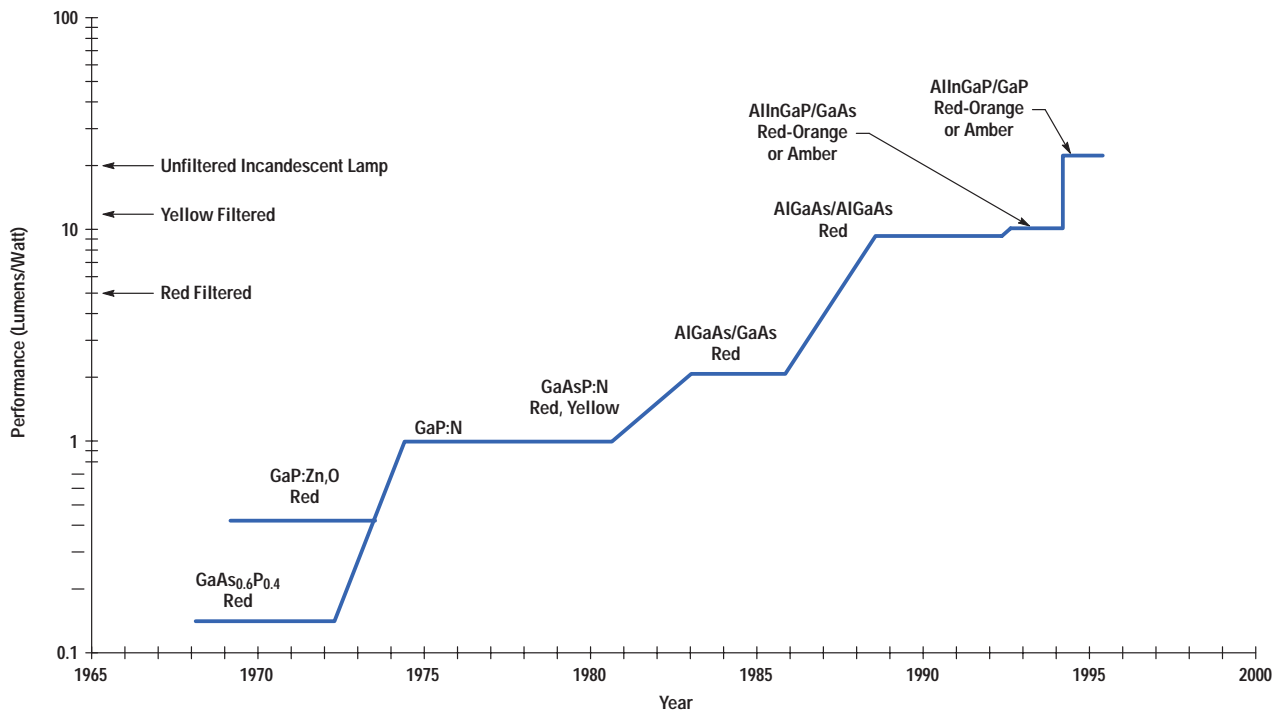
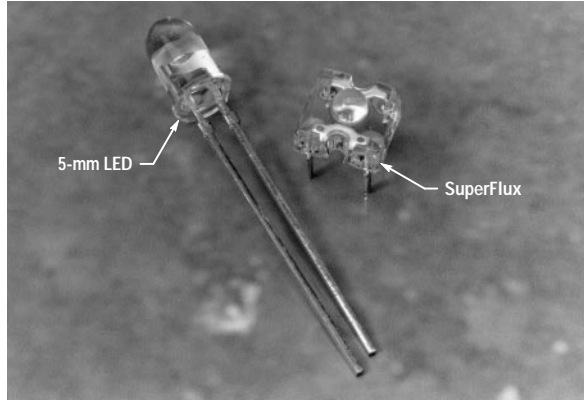


Figure 4

Superflux and 5-mm LED packages.

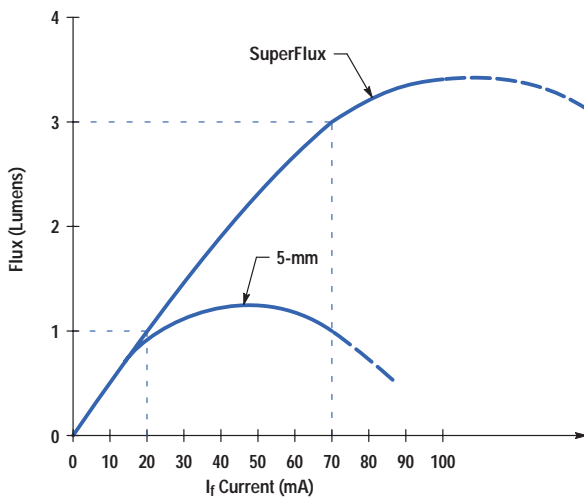


more optically efficient and could be operated at twice the drive current of conventional LED packages. The SuperFlux package and a conventional 5-mm LED are shown in **Figure 4**.

Figure 5 shows the light-output performance of a SuperFlux package compared to the 5-mm package in an automotive application. Each package contains the same type of LED chip (TSAlInGaP/GaP).

Figure 5

Flux versus forward current for a SuperFlux LED package compared to a 5-mm LED package. Both contain AlInGaP LEDs.



The AlInGaP LED light output shown in **Figure 5** does not increase linearly with forward current (I_f) in typical use conditions. **Figure 6** shows that AlInGaP demonstrates an exponential drop-off in light output as a function of increasing junction temperature (T_j). At higher drive currents, more heat is created in the device and T_j rises. This rise in T_j , and the corresponding decrease in device efficiency, eventually offsets the increase in light output because of increased I_f .

The combination of AlInGaP materials and SuperFlux LED packaging are the basis for HP's automotive lighting products and the foundation for SnapLED. Today HP's LED technology can be found on approximately 15% of the cars in production and many heavy duty tractors and trailers.

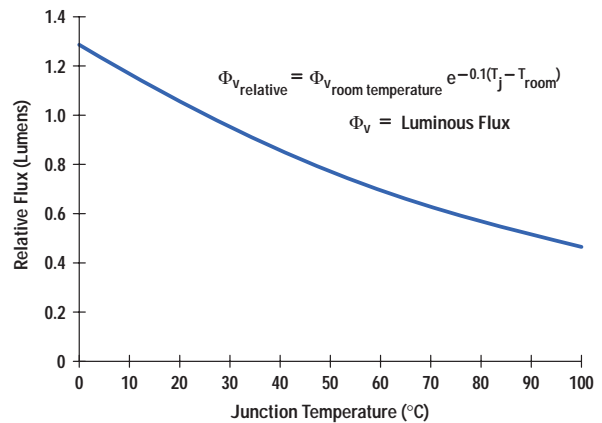
LED Rear Combination Lamps

The signal lamps at the rear corners of the vehicle, which combine tail, turn, stop, and often side marker and reverse functions in one package, are commonly referred to as rear combination lamps (RCLs). RCLs used in modern vehicles have complex 3D outer lens surfaces that follow the contour of the car body. An LED light source for this application must have the following attributes.

Styling Flexibility. RCLs are used to differentiate the look of a vehicle. An LED light source needs to accommodate the range of designs of conventional incandescent RCLs and at the same time offer additional styling flexibility.

Figure 6

Relative flux versus junction temperature for AlInGaP LEDs.



Low System Cost. System cost refers to the cost for the automobile manufacturer to purchase, install, and maintain a signal lamp on a vehicle. Some elements of system cost include warranty expenses, electrical power consumption, adding features to body panels to mount the lamp, wires, and other electrical connections. LED light sources are more expensive than their incandescent counterparts. However, the system cost of an LED signal lamp can be comparable to an incandescent signal lamp.

Power Consumption. Power use is of particular interest in modern vehicles because ever increasing electrical demands are resulting in excessive alternator loads. Many vehicle manufactures now assign a cost savings per ampere in the range of U.S. \$3 to \$5. A conventional incandescent RCL will consume approximately 2.5A to 5A in the stop mode. An LED RCL will consume 0.5A to 1.5A in the stop mode, saving approximately 2A to 3A, which correlates to a system cost savings of U.S. \$6 to \$15 per side and U.S. \$12 to \$30 per vehicle.

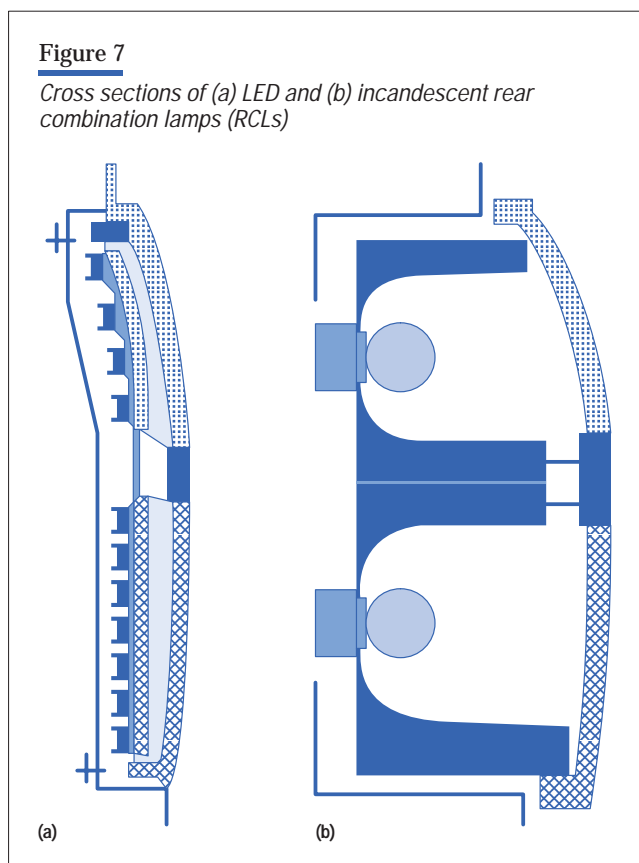
Space Savings. Saving space is another area of interest in modern vehicles. By creating a three-dimensional distributed light source, the thickness of the RCL can be reduced as shown in **Figure 7**.

Incandescent RCLs can be as deep as six inches compared to LED RCLs which can be as thin as one inch. Incandescent RCLs consume valuable trunk space and require the addition of deep-drawn pieces to the rear quarter panel of the vehicle to accommodate their depth.

SnapLED Design

The SnapLED assembly consists of a modified SuperFlux LED emitter, which is clinched to a metal frame called a clinch frame. The clinch joint mechanically and electrically attaches the emitter to the clinch frame. The clinch frame performs the task of a printed circuit board, providing the mechanical structure for the assembly and forming the desired electrical circuit. **Figure 8** shows a SnapLED emitter, a clinch frame, an assembled SnapLED array, and a formed array.

SnapLED Emitter Design. Both the SuperFlux and SnapLED emitters are manufactured on the same assembly line, and except for the lead frame, share the same materials and piece parts. The optical design and body outlines are identical. By using a proven emitter design (SuperFlux), the risk, investment, and time to market were minimized.



In addition, the combined volumes of SuperFlux and SnapLED emitters allow for reduced manufacturing costs.

As can be seen in **Figure 8**, large pads extend from the anode and cathode leads of the device and serve as the areas for the clinch attachment. (A better view of these pads is shown in **Figure 14**.) The exposed portions of the leadframe, including the attachment pads, are nickel plated to prevent tarnish and corrosion.

SnapLED Clinch Frame Design. The clinch frame must provide adequate mechanical support, form the electrical circuit, and dissipate heat as efficiently as possible. The base material used for the clinch frames is the same copper alloy used for the emitter leadframe. The plating used on the clinch frame is a tin alloy, which resists tarnish and is well suited for slide-on electrical connectors. Special features are added at all bend locations to ensure well controlled, properly formed bends.

Drive Circuitry Design. Standard electronic components cannot be clinched to a SnapLED array. For this reason,

Figure 8

The components of a SnapLED array.

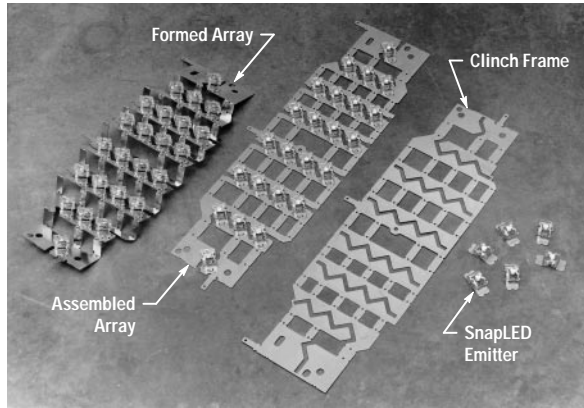
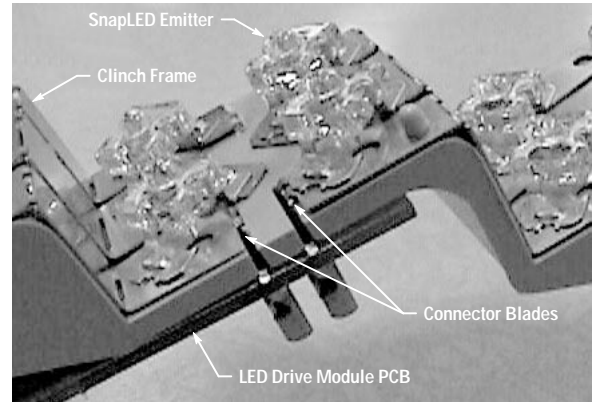


Figure 9

SnapLED assembly attached to the LED drive module PCB.



all drive circuitry is either mounted in the wire harness or on a remote printed circuit board. For CHMSL (center high mount stop lamp) applications, in which drive circuitry typically consists of a current-limiting resistor and a reverse-voltage-blocking diode, the drive circuit can be mounted in the wire harness. For RCL applications, current control drive circuits are recommended. These circuits are so complex that mounting the circuitry in the wire harness is impractical. For RCLs, a remote printed circuit board contains the drive circuitry. This LED drive

module is connected directly to two connector blades on the SnapLED assembly (see **Figure 9**).

The remote location of the LED drive module isolates the heat generated by the drive circuit from the LED array. This is advantageous because light output in AlInGaP LEDs degrades with elevated temperatures. It is also advisable to use low-dropout, low-power, current control circuitry to minimize the heat generated. Switching current regulators are ideal because of their high efficiency.

Figure 10

Block diagram of a switching power supply for LED rear combination lights (RCLs).

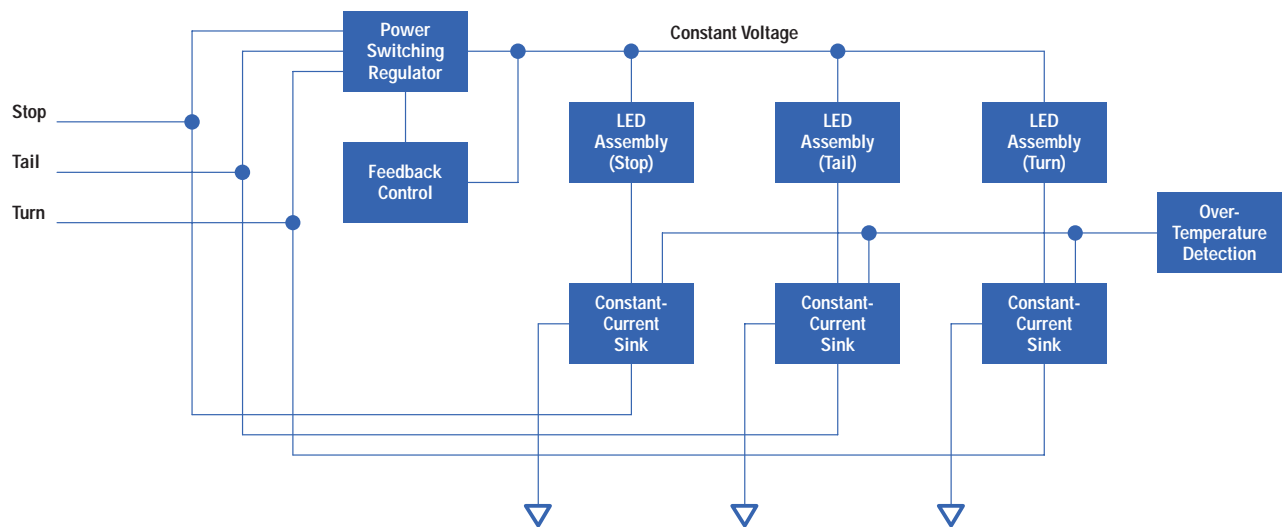
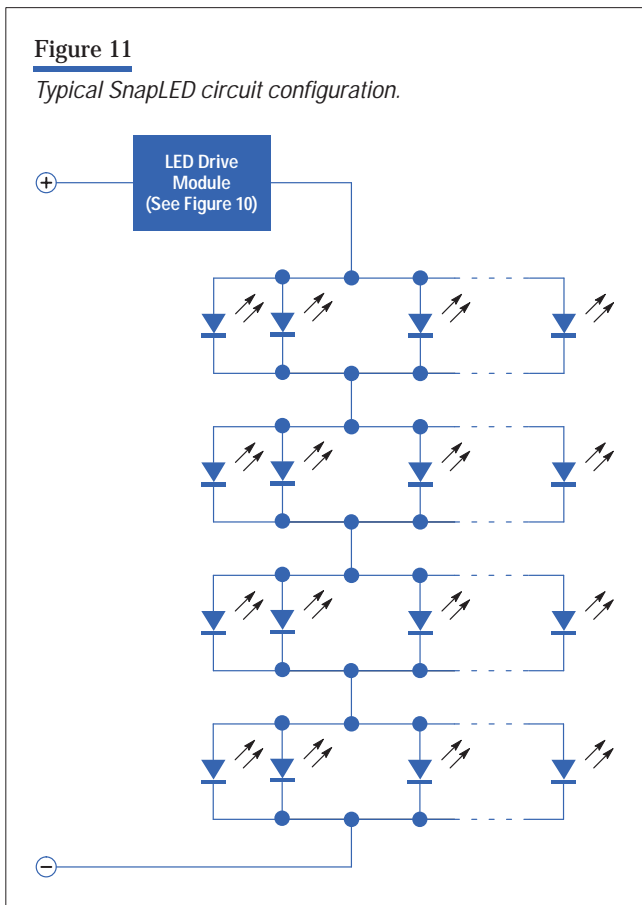


Figure 11

Typical SnapLED circuit configuration.



However EMI and added cost and complexity have limited their use. A block diagram of a switching power supply for LED RCLs is shown in **Figure 10**.

Clinch frames cannot accommodate the complexity of printed circuit board circuit designs. In addition, all drive circuitry must be located remotely. For these reasons, SnapLED circuits are generally parallel-series configurations as shown in **Figure 11**.

With conventional drive circuitry, no more than three to four LEDs are run in series so the LED array will operate under low-voltage conditions (9V). This is not true in the case of switching drive circuits in which the input voltage can be converted to a higher or lower level. To operate LEDs in parallel, the LED emitters within a parallel string must be voltage matched. HP sorts and categorizes its automotive LEDs into 120 mV bins for this purpose.

SnapLED Clinch Process

Clinching is a method for mechanically joining nonferrous metals. In a typical application, several clinch joints are used to attach two metal sheets together. Because of space constraints, there is no redundancy in clinch joints on SnapLED, and each joint performs a critical mechanical and electrical function. This new clinching application required extensive process development.

Among several clinching methods available, a pierce and form clinch joint was chosen for SnapLED because of its compact size and the simplicity of tooling. **Figure 12** shows a perspective view of the clinch tool used for SnapLED joint formation.

Figure 13 shows the clinch process. **Figure 13a** shows the materials to be joined positioned under the clinch tooling. **Figure 13b** shows the punch and stripper position after piercing the clinch frame and LED lead. **Figure 13c** shows the compression and expansion of the displaced

Figure 12

SnapLED clinch tool.

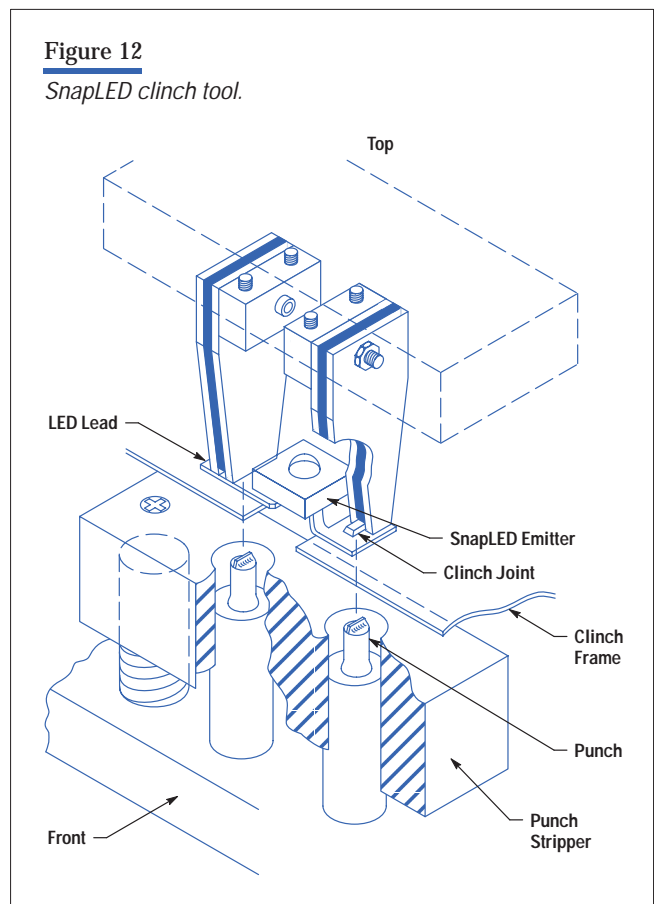
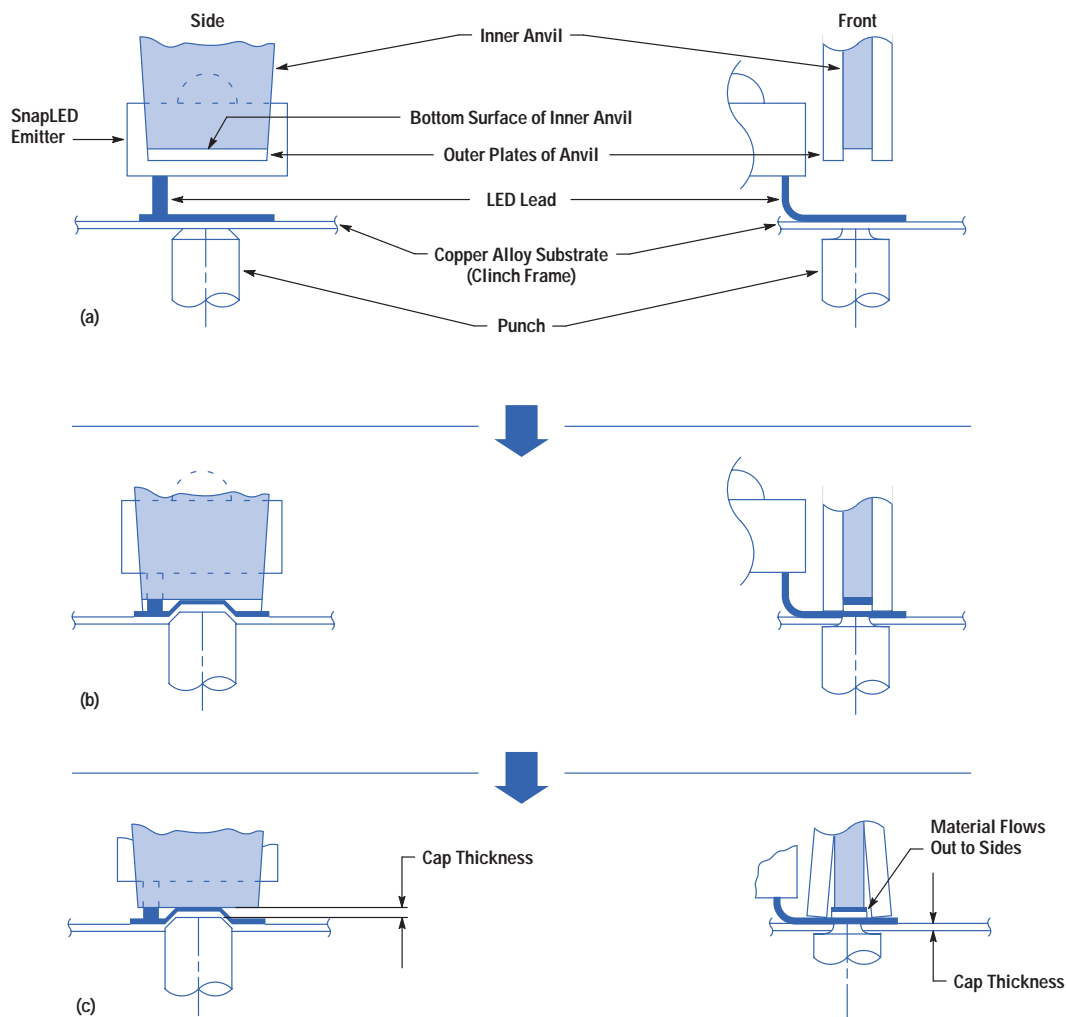


Figure 13

The clinching process. (a) Placement of components. (b) Punch pierces substrate and LED lead. (c) LED lead is clinch-locked to substrate.



material as the punch continues its travel. A photograph of an individual clinch joint is shown in **Figure 14**.

Many parameters must be controlled to ensure a proper clinch joint. The following is a list of critical parameters.

Cap Thickness. The thickness of the upset material, or cap, must be controlled to within 0.10 mm. If the cap is too thick, this indicates that it has not been compressed enough to ensure proper material expansion and interlocking. If the cap is too thin, this indicates over compression which results in weakening of the base material. Over compression also results in premature wear of the

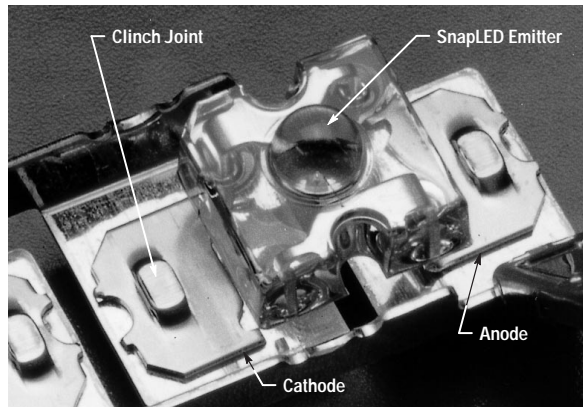
tooling. Inductive position sensors are used to monitor cap thickness on production assembly equipment.

Tooling Alignment. If the top and bottom halves of the SnapLED tooling are not properly aligned, the clinch joint will not be properly formed. The joint becomes D-shaped because shearing only takes place on one side of the joint. Sample inspection is currently used to detect this defect. An investigation is under way to use dynamic force data to monitor tooling alignment.

Material Hardness. The hardness of the materials to be joined must be matched and controlled. If the materials

Figure 14

A close-up view of a clinch joint.



are too soft, the joint will be weak. If the materials are too hard and brittle, they may fracture during clinching or may not flow together during compression. If the hardness of the materials is not matched, the two materials will not flow evenly, again resulting in a weak joint. Hardness of base materials is monitored on a sample basis. In addition, piezoelectric force sensors monitor the force during the clinching process. If the force required to form the joint is too high or too low, this indicates a base material that is too hard or too soft.

SnapLED Manufacturing

SnapLED required the development of special electronic assembly equipment and processes. A flow chart of the SnapLED manufacturing process is shown in **Figure 15**.

Clinching. Two types of clinching automation have been developed. First, there is clinching equipment that is dedicated to a single product, is fully automated, and incorporates clinch frame feeding, emitter placement and clinching, and shearing. The second type of automation involves flexible clinching equipment that automates only emitter placement and clinching but can accommodate a wide variety of array designs. Flexible equipment is preferred for automotive products because taillight designs change every three to five years as new vehicles are designed. An example of a flexible clinch machine is shown in **Figure 16**.

In this design, one or more clinch frames are loaded into a clinch fixture. Several clinch fixtures can be loaded and staged before clinching. SnapLED emitters are positioned under the clinch tooling by the pickup turret as the clinch frames are positioned by the x-y positioning stages.

Figure 17 shows the emitter placement mechanism.

SnapLED emitters are packaged in tubes and fed into a track. Parts slide down the track to a rotary stage where the part is tested and oriented to the proper polarity. Parts are then staged in a buffer zone at the end of the track.

Figure 15

SnapLED manufacturing process.

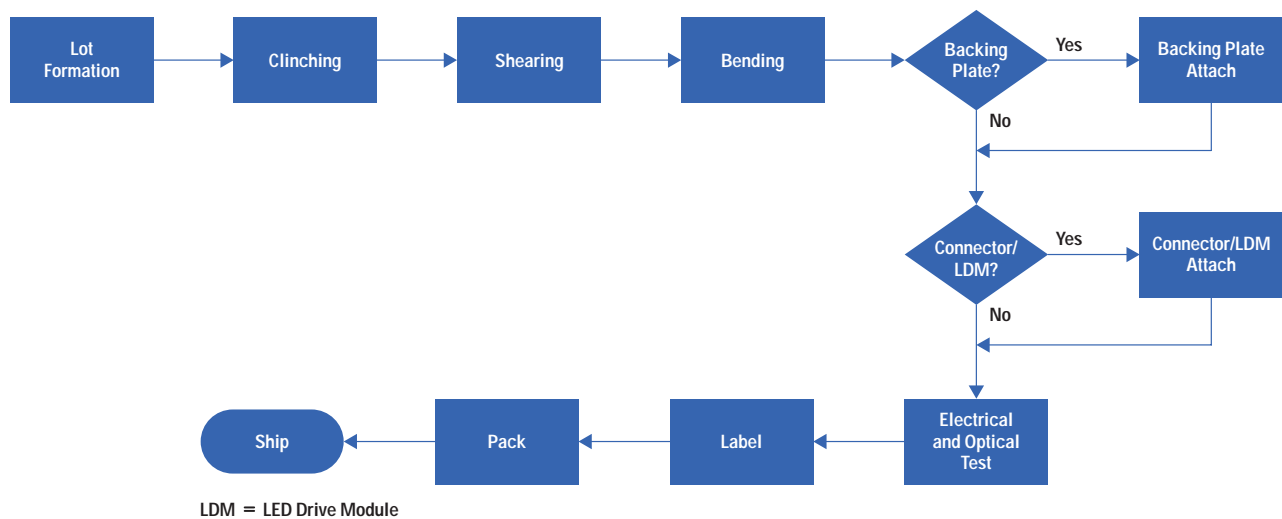
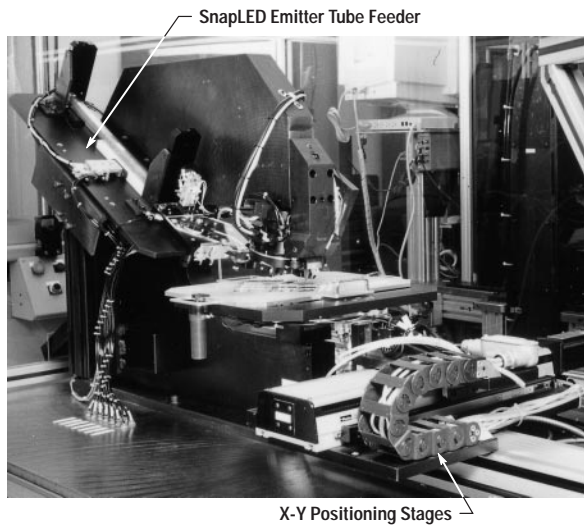


Figure 16

A flexible clinch machine.



The vacuum pickup turret picks the part from the end of the buffer and rotates it 45 degrees. As the turret rotates, the emitter at the other side of the turret is positioned under the clinch tooling. In parallel, the clinch frames are positioned by the linear stages. Once the emitter and clinch frame are in the proper orientation, the press is actuated and the punches rise to form the clinch joint shown in **Figure 14**. When all the emitters have been clinched into position, the stages return to a home position so another clinch fixture can be loaded.

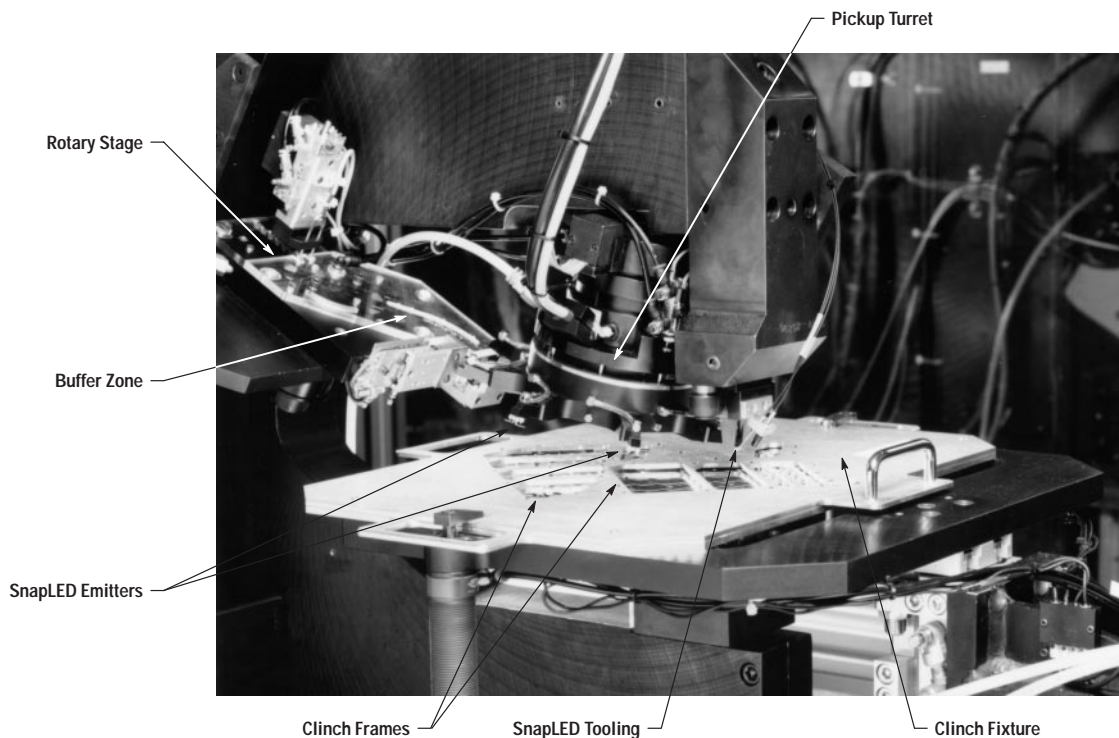
Testing. SnapLED emitters are tested and separated into flux, color, and forward voltage (V_f) categories. Typically, a single category of emitter is used to create an array to ensure proper performance and uniformity. However, for some applications, different color and V_f bins may be used.

After assembly, the array must pass a final electrical and optical test. Here, the array is checked for current compliance at the designed voltage input, light output uniformity* between the individual LED emitters, and

* Ratio of dimmest to brightest LEDs anywhere on the array.

Figure 17

Parts of the SnapLED assembly machine.



overall light output. Light output uniformity is controlled within a ratio of 1:1.9 (dimpest : brightest) for red devices and approximately 1:1.7 for amber devices. Overall light output is determined by averaging the light output of the individual emitters.

The testers are modular to maintain a high degree of flexibility. The modules consist of a PC, power supply, optical detector array, and test fixture (see **Figure 18**).

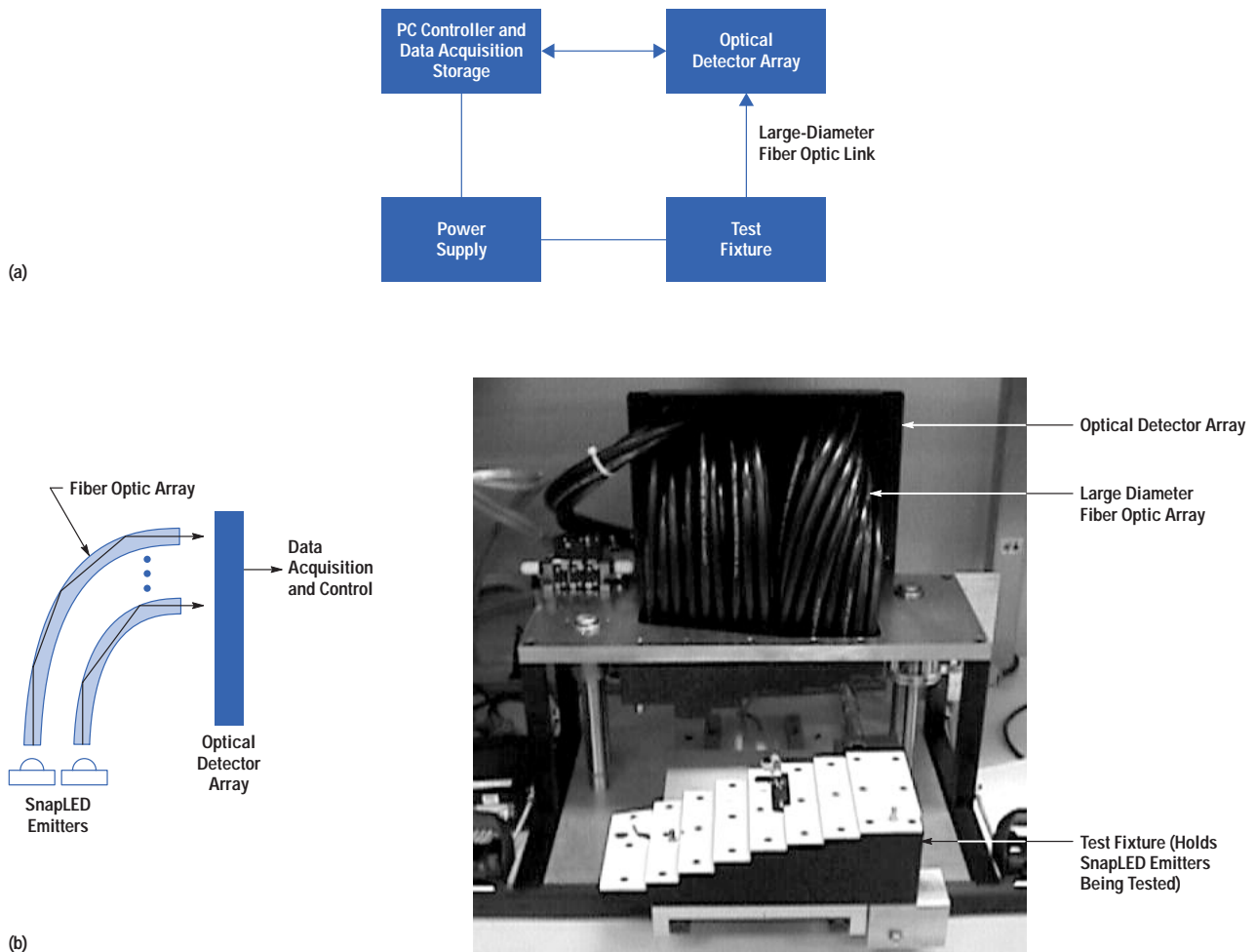
The PC is used to provide control, data acquisition, data storage, and a user interface for the system. The test fixture properly positions the array under a large-diameter, fiber-optic array. The fiber-optic array channels the light from the emitters to the optical detector array. The

optical detector array consists of a 10 by 12 grid of surface mount optical detectors attached to a printed circuit board contained in a housing. The top plate of the housing accurately positions the fibers over the detectors and allows the individual fibers to be installed and removed as needed.

Because the clinch frames must conform to different tail-light designs, each SnapLED array must have a dedicated test fixture, but all other test hardware can be shared. In addition, minor modifications to the tester's software are needed for each array to account for different test limits, number of LEDs, and type of LEDs used.

Figure 18

(a) Block diagram for a SnapLED tester. (b) SnapLED test station.



SnapLED Array Design

The SnapLED array design process begins with design reviews with the automobile manufacturer and lighting supplier to make sure the lamp design is optimized in terms of style, performance, manufacturability, and economics. Once the RCL design is complete, the SnapLED array can be designed.

Solid models of the RCL housing, provided by the customer, are imported into the PE/Solid Designer CAD system. The LED array is then created with the LEDs in their proper locations. Attachment and alignment features, electrical connectors, bend lines, and electrical traces are then added. After completing the three-dimensional model, the clinch frame is “unbent” into its two-dimensional form using Sheet Advisor (a module in PE/Solid Designer). The two-dimensional clinch frame is then modified to include supporting structures needed to fix and stabilize the part during array assembly.

Early prototypes are created using chemically etched clinch frames. The prototype parts are bent, sheared, and tested using flexible tooling and equipment. Prototype arrays can then be checked for fit and ease of assembly using plastic housing prototypes fabricated by rapid prototyping techniques.

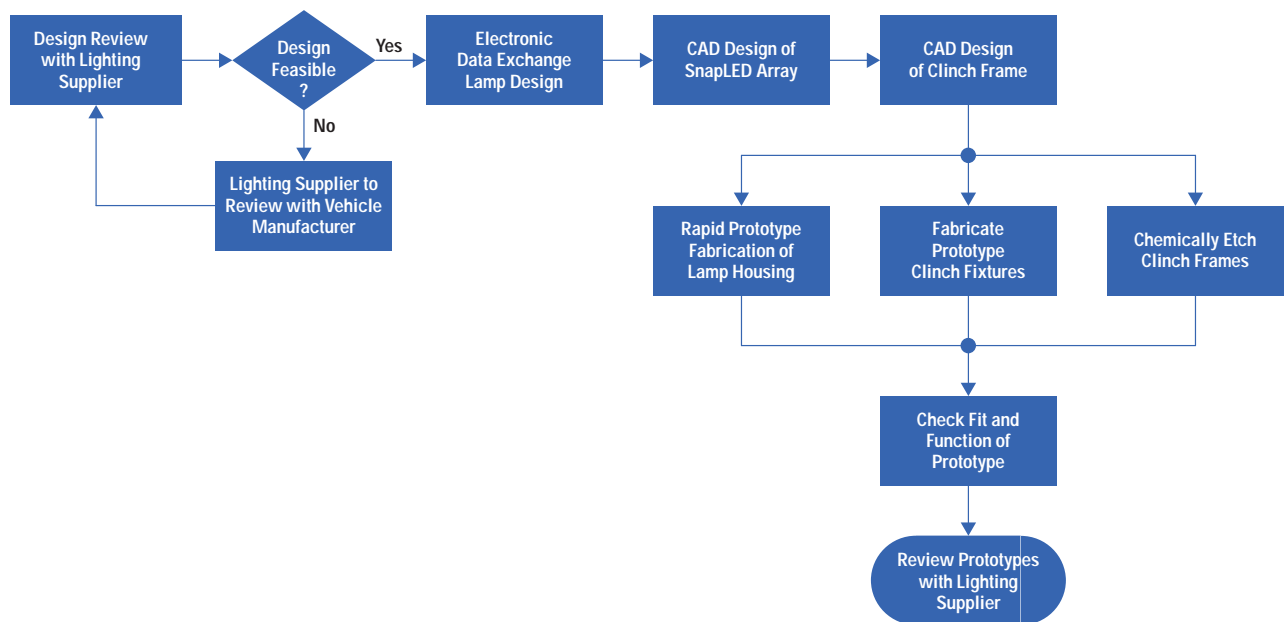
A flow chart of the SnapLED design cycle is shown in **Figure 19**.

SnapLED on the Road

The first SnapLED array in production appeared on the center high mounted stop lamp of the 1998 Ford Explorer. SnapLED has been designed into other vehicles slated for late 1998 production. They will be used in applications ranging from the world’s first LED turn signals to rear combination lamps.

Figure 19

SnapLED design cycle.



Acknowledgments

The original concept and much of the pioneering work in clinched LED arrays was conducted by Dick Klinke and Gary Sasser. Wayne Snyder and Jim Leising provided unwavering management support throughout the project, and for this, the author is grateful. Much of the credit for the development and commercialization of SnapLED belongs to Dave Brandner, Todd Swanson, Douglas Woolverton, Chris Togami, and Leong Ak Wing. Many others, too numerous to acknowledge individually, have contributed to the development of SnapLED and SnapLED arrays for specific vehicles.

Bibliography

1. M.G. Craford and G.B. Stringfellow, *High Brightness Light Emitting Diodes*, Academic Press, 1997.
2. *Multiple Light Emitting Diode Module*, U.S. Patent #5,404,282.
3. *Moldable Nesting Frame for Light Emitting Diode Array*, U.S. Patent #5,519,596.

OTDR APIs Enable Customers to Build Their Own Systems

Torsten Born

Peter Thoma

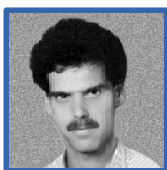
In the past few years, OTDRs have evolved from being used only as standalone measurement instruments with limited functionality to become key instruments for servicing and characterizing global fiber-optic communication links. This trend has spurred the creation of standard file formats for OTDR data and standard software interfaces to control OTDRs remotely.



Torsten Born

A software engineer at the HP Optical Communication Measurements Operation,

Torsten Born is presently the technical supervisor for the OTDR toolkit and other projects and is the HP contact for the Bellcore SOR (standard OTDR record) file format. He joined HP after graduating from the University of Paderborn in 1994 with a Diplom Ingenieur in electrical engineering (focus on computer science). He was born in Hamburg, Germany, and his recreational interests include photography, music, traveling, volleyball, and inline skating.



Peter Thoma

Peter Thoma is an R&D project manager for the mini-OTDR HP E600A. He

is currently investigating future optical network testing. He came to the HP Böblingen Instrument division in 1993. He received a Diplom Ingenieur in technical optics and control systems from the University of Stuttgart in 1984. Peter was born in Tuebingen, Germany. He is married and has three children.

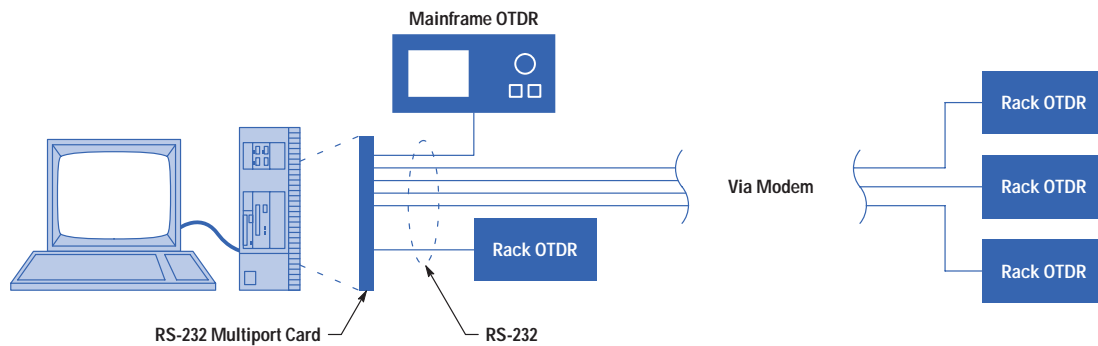
Optical Time Domain Reflectometers (OTDRs) have become key instruments for characterizing fiber-optic communication links. They are used to test the transmission performance of optical fibers by an optical pulse-response measurement method. During installation and maintenance of networks, OTDRs are primarily used to verify fiber and component parameters like optical loss and reflectance. The basic result from an OTDR measurement is the characteristic signature of the link called a *trace*, which is the amount of reflected light recorded versus time and distance. For an OTDR measurement, parameters such as pulse width, wavelength, range, and refractive index have to be defined. The trace can be postprocessed by a scantrace algorithm, which is used to detect faults, bad connections, and so on.

Today, optical network operators are driven by the need for high-quality service and high flexibility for quickly evolving the network according to market needs. The ability to access information about the fiber plant is essential for configuration, fiber network upgrade, and cost-effective maintenance.

Thus, many network operators archive installation data to be used as a reference for later comparisons during the lifetime of the network. OTDR measurement files form the core of this data collection. The total amount of data increases exponentially each year because of the worldwide deployment of optical fibers.

Figure 1

Remote OTDR control.



In some networks, OTDRs are not only being used as operator-based instruments but are also being integrated as monitoring probes at the terminal and are controlled remotely (see **Figure 1**). With this approach the link quality of fibers and cables can be checked online. Restoration and repair can be optimized in case of a failure. Degradation can even be detected before the transmission quality suffers.

These trends call for a broader use of standards in OTDRs, such as standard file formats, standard remote interfaces, and application support with these standards.

Hewlett Packard provides a range of products (for example, the HP E6090A OTDR toolkit and the HP 81700 Series remote fiber test system) that offer complete solutions for monitoring OTDR measurement data during the complete network life cycle. These solutions are based on the TMN (Telecommunications Management Network) approach and use standard databases and interfaces.¹

Market Needs

The telecommunications environment has a number of requirements for customers and solution providers like Hewlett-Packard. For system software that works with OTDR measurement data, these requirements include:

- **Data portability.** Over the lifetime of an optical fiber (greater than 20 years), all acquired measurement data must be accessible. Thus, old data formats must be supported. This also implies compliance with industry standards such as the Bellcore OTDR file format.²

- **Multivendor capability.** Measurement data is typically generated by instruments from various vendors.
- **Quality.** The software volume covered by tests grows with every new product generation.
- **Cost-effective development.** Since the general principles of the software remain the same when developing a new product, it is very important to offer a large reuse potential to save development resources and allow cost-effective development.
- **Knowledge concentration.** Most customers do not have expertise in OTDR measurement data analysis.
- **Integration into customer-specific systems.** Besides integrating OTDR data into existing customer asset management systems, the software should also be able to:
 - Establish communication with an instrument
 - Provide instrument control
 - Decode and encode OTDR measurement data
 - Perform offline analysis of measurement data.
- **Long-term consistency.** To follow the technological evolution of operating systems or measurement capabilities, a solid platform (operating system) is required for future migrations.

To address these market needs, HP offers two libraries: OTDR-API and OTDR-CAPI, which are application programming interfaces that are based on HP's OTDR design and

application knowledge. These libraries allow customers to postprocess measurement results and control the OTDRs from their own software environment. These libraries are compiled as 16- or 32-bit Windows® DLLs (dynamic link libraries) and offer interfaces to standard C and Visual Basic. Both libraries are designed to work together and offer customers a transparent programming environment that can be integrated into any proprietary system software.

The rest of this article describes the features and implementation details of the OTDRAPI and OTDRCAPI libraries.

Solutions

Features offered in the OTDRAPI library include loading and storing measurement data in standard Bellcore format, performing analysis operations (scantrace, loss calculations), and creating configuration* files and templates. The OTDRCAPI provides complete encapsulation of the serial interface and the SCPI (Standard Commands for Programmable Instruments) command language. Both libraries provide access to all measurement data and parameters on the remote OTDR.

The hardware component is a faceless OTDR for system integration, the HP E605x rack OTDR, which offers maximum measurement accuracy at a minimum price. The OTDR APIs help customers to write their own system software around this measurement hardware without the need to be OTDR specialists or have intricate serial interface knowledge. By encapsulating the communications functions, it is possible to offer optimal connectivity and data transmission performance as well as transparent error handling.

Standards

The OTDR libraries support two industry standards: the Bellcore file format for optical data and the SCPI command language for communicating with the instrument.

Bellcore File Format. In the past, every OTDR manufacturer had at least one file format of its own for storing OTDR measurement results. This has been a problem for the customers who have to cooperate with companies

that use different measurement equipment and data formats. Since several companies are involved in manufacturing, installing, monitoring, and servicing optical fibers, a common data format and documentation standard has become necessary. The Bellcore SOR (standard OTDR record) format is the first attempt to create a common, portable OTDR data format and overcome these problems.

Although the SOR format solves the data exchangeability problems, it generates some new tasks for those customers who need to decode the data because SOR:

- Is in binary format
- Is a “living” standard in that it will experience regular updates
- Represents a superset of OTDR data from different manufacturers, so that its structure is complicated, and it contains data that may not be relevant to all customers
- Stores data in uncommon formats for portability.

Why would a customer want to decode the SOR? The main reason for decoding a binary data file is to create documentation. Typically, customers have their own documentation rules (for example, how to organize the different OTDR data on a printed report). There is no standard rule about which data to use and how to interpret it.

SCPI Remote Language. SCPI is a standard that is being driven by Hewlett-Packard. Many HP instruments support SCPI for remote control via HP-IB (IEEE 488.1, IEC 625), RS-232, or LAN.

SCPI uses ASCII commands to offer a functional interface to the instrument. Therefore, long-term and multivendor compatibility are taken care of by design. The complete programming language is organized in a tree structure in which each root represents a different view of the instrument's measurement or configuration data. Branches allow for accessing the relevant data of any particular view. For example, changing the instrument's serial baud rate is achieved by using a command from the SYSTEM tree, which allows for accessing configuration data. For example:

```
SYSTEM:COMMUNICATE:SERIAL:BAUDRATE 57600
```

where COMMUNICATE indicates that the data is related to communication, SERIAL accesses data for the serial interface, and BAUDRATE indicates the actual value.

* The configuration file contains a subset of all instrument settings, including measurement parameters that are needed to configure a measurement. This allows all necessary parameters to be sent in just two commands (file transfer and load setting) in a very compact format to the OTDR.

Since commands can become quite large using such syntax, SCPI defines a short form that only uses the first three or four characters of each branch:

```
SYST:COMM:SER:BAUD 57600
```

If the instrument supports more than one serial interface, the command can be adjusted to address the correct port. For example,

```
SYST:COMM:SER2:BAUD 57600
```

addresses serial port 2.

Querying the instrument's configuration is achieved by the same command with a "?" instead of a value appended to the command string.

```
SYST:COMM:SER2:BAUD?
```

For more detailed information about SCPI see reference 3.

SCPI also defines the instrument's behavior for such things as syntax errors and parameter overflows and underflows. Higher-level interfaces have been defined on top of the SCPI language (for example, the VISA plug-and-play drivers that offer an interface to applications such as HP VEE or LabView® from National Instruments).

Although remote control of an OTDR instrument mainly serves the needs of a monitoring application, it also helps to reduce the need to maintain an engineering staff with a lot of knowledge about fiber-optic measurements. For example, one engineer can set up and control several measurements from the central office, while field technicians can focus on setting up the equipment at various measurement sites. This allows instrument control to be done by modem.

The OTDRCAPI library allows controlling an OTDR without even knowing the SCPI language because the library offers a high-level functional interface for the most common applications. For special applications, the OTDRCAPI library offers pass-through functions, which allow sending commands directly to the instrument without checking or processing. This provides complete transparency.

Software Architecture Considerations

When designing system software, several aspects have to be considered. First the operating system and hardware platform play an important role. Since the WINTEL combination (Microsoft Windows/95/NT® + Intel processor) is becoming more and more standard, we have focused

on the Windows operating systems for our APIs. However, since most of the API libraries' implementation is done in standard C and C++, a port to a real-time or HP-UX operating system would be quite easy.

The interfaces to both API libraries are standard C, C++, and Visual Basic. Visual Basic enables customers to rapidly prototype applications, while standard C allows the creation of a very elaborate software system with good performance.

Implementation Details

Both libraries are subject to regular updates because the Bellcore and SCPI standards are living standards. Therefore, a major focus has been put into a generic definition for the API function calls because these interfaces will probably stay constant while some dynamic link libraries may change.

OTDRAPI. The core of the OTDRAPI library is a C++ library called OTDRLIB. This library is also an important element of all OTDR products such as the HP E4310A mainframe OTDR, the HP E6000 mini OTDR, the HP E6090 OTDR toolkit, the HP E605x rack OTDR, and the HP 81700 Series 200 remote fiber test system. This library provides all the mathematical and file access functions, which guarantees that all changes to the instruments' firmware can easily be updated in the OTDRAPI library.

Figure 2 shows HP OTDR products and their associated operating systems that are already covered by the OTDRLIB. The data storage shown in **Figure 2** contains the following OTDR measurement data:

- Trace data points
- Measurement parameters
- Hardware-specific data
- Event and landmark tables
- Trace information such as cable and fiber identifiers.

All OTDRLIB internal definitions (such as structures and constants) are encapsulated by the OTDRAPI library, so that even major changes to internal data handling will not affect the library's external interfaces such as function calls and data types.

We decided not to offer an object-oriented C++ interface because standard C is already a subset. We chose to give

Figure 2

HP OTDR products and their associated operating systems covered by OTDRLIB.

HP E4310A OTDR	HP E4320A PC Software	HP E6000A OTDR HP E605xA OTDR	HP E5510A RFTS	HP E6090A OTDR Toolkit	HP OTDRAPI C and Visual Basic Library	} Operating Systems
Windows 95	Windows 95 Windows NT 4.0	pSOS	HP-UX	Windows 3.x Windows 95 Windows NT	Windows 3.x Windows 95 Windows NT	
OTDRLIB: <ul style="list-style-type: none"> • Data Storage • Scantrace Algorithm • Loss Calculation Routines • Bellcore File Format Reader and Writer • HP 8146 Reader 						

customers the flexibility of defining their own objects depending on the available software structure.

Figure 3 shows the internal architecture for the OTDRAPI library. The OTDRAPI library allows read access to all data in the internal Bellcore data storage. However, it is not possible to modify core measurement data such as trace data, measurement parameters, and hardware-specific data. This guarantees measurement data integrity under all circumstances. It is possible to modify all documentation data such as the event table or the trace comments.

The library can also be used to define a set of measurement parameters for a new measurement that can then be stored in a configuration file.

Because of the large amount of data handled internally, MS-DOS® is not supported. The size of one Bellcore trace normally exceeds 32K bytes, and we prefer to use flat memory internally. For 16-bit Windows, we were able to minimize the restrictions by using a large memory model, which is unfortunately not possible with standard MS-DOS.

OTDRCAPI. One major problem that we had to overcome with the OTDRCAPI library is the cumbersome implementation of serial communications under different operating systems. Even the different Windows operating systems do not have a consistent interface. To have a stable level, we decided to use CommLib from GreenLeaf as a serial interface abstraction layer. The CommLib offers a consistent function layer for initializing, opening, closing, and communicating over a serial interface. However, the customer will not see any of the GreenLeaf functions because the OTDRCAPI library encapsulates the instrument's SCPI remote language (see **Figure 4**).

For example, opening a communication with an OTDR instrument using the OTDRCAPI library involves:

- Configuring the baud rate, handshake, and parity
- Opening the serial interface
- Reading the instrument's identification
- Configuring some of the OTDRCAPI functions regarding the detected instrument type

Figure 3

The internal architecture for the OTDRAPI library.

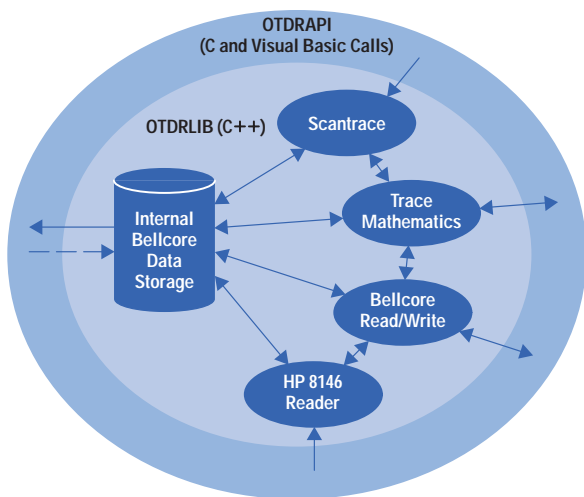
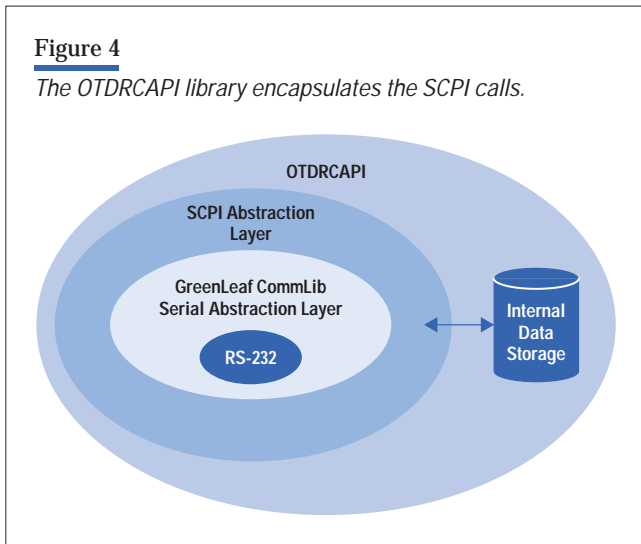


Figure 4

The OTDRCAPI library encapsulates the SCPI calls.

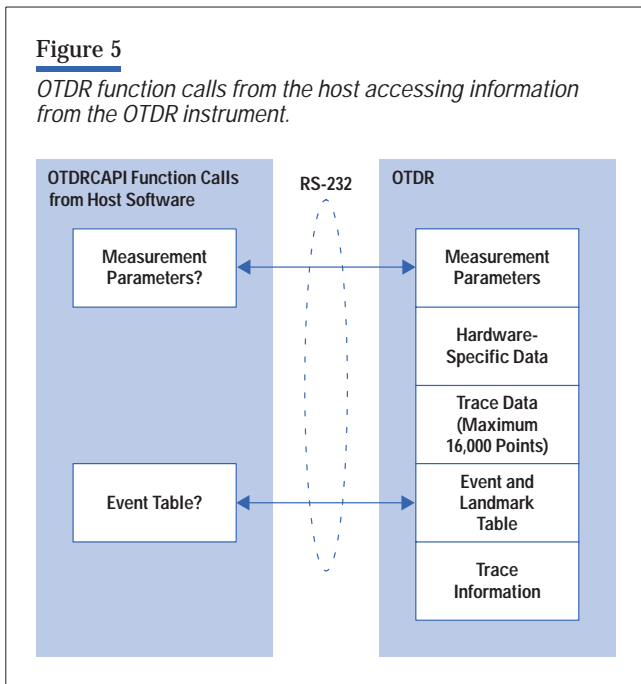


- Reading the instrument's status (for example, measurement running)
- Switching to a higher transfer baud rate if necessary
- Returning an error when one of the above operations fails.

The same error handling applies to all other functions such as accessing trace data, measurement parameters, and event tables (see **Figure 5**). Using a higher transfer

Figure 5

OTDR function calls from the host accessing information from the OTDR instrument.



baud rate is useful if the instrument always boots at a default baud rate (19.2 Kbits/s) because transferring large amounts of measurement data requires a higher transfer rate (115.2 Kbits/s). The OTDRCAPI library takes care of all synchronization, necessary delays, error handling, and returning to the default baud rate when the port is closed.

Unlike the internal data storage for the OTDRAPI library, the OTDRCAPI library holds only a part of the Bellcore file. The OTDRCAPI library only provides a window to the Bellcore data stored on the instrument. Therefore, because all data structures are automatically handled between the OTDRCAPI and OTDRAPI libraries, the programmer receives the same data structures when reading from a remote instrument as when reading data from a Bellcore file. Communication between the two libraries can be handled by either using the same data structures for items such as parameters and trace data or passing a storage pointer from the OTDRAPI library to the OTDRCAPI library.

By combining both APIs, it is possible to read a complete Bellcore trace from a remote instrument into the OTDRAPI's internal Bellcore storage and then perform all data and mathematical operations on this data locally using the functions in the OTDRAPI library. This capability allows distribution of the steps of an OTDR measurement between the OTDR and the controlling host software. The following steps illustrate this concept:

- Perform the measurement on the OTDR without scantrace
- Upload the data to the host
- Perform the scantrace on the host (this will help to improve performance)
- Postprocess the event table on the host
- Edit trace comments, event comments, and landmarks on the host
- Download the data back to the OTDR
- Store the measurement on the host and OTDR.

Figure 6 shows a programming example that demonstrates how simple the implementation of the steps listed above can be using the two APIs. All calls with the prefix OTDRCOM are from the OTDRCAPI library, and all calls with the prefix OTDR are from the OTDRAPI library.

Figure 6

Example of code using OTDRAPI and OTDRCAPI library functions.

```
// declarations ...
COMDEV pComdev; // the communications
                // device pointer
OTDRP pOTDRapi; // the OTDRAPI pointer
short status;   // holds the status of
                // function calls

// initialize OTDRCAPI ...
if(OTDRCOMInitLib( NULL ) )
{
    printf("Unable to initialize
    OTDRCAPI!\n");
    exit(0);
}

// initialize port ...
pComdev = OTDRCOMInitPort(&status, "COM1");
if( status )
{
    printf("Error: %s\n",
    OTDRCOMStatusMessage(pComdev));
    exit(0);
}

// ... code for setting up the port goes
// here ...

// open the port ...
if( OTDRCOMOpenPort( pComdev ) )
{
    printf("Error opening the serial
    port!\n");
    exit(0);
}

// initialize the OTDRAPI; further error
// checking not listed!
pOTDRapi = OTDRInitLib ( &status );

// start a remote measurement and wait
// for 30 sec...
OTDRCOMStartMeasurement(pComdev);
Sleep(30)

// stop the measurement ...
OTDRCOMStopMeasurement(pComdev);

// check if the measurement is stopped ...
BOOL bRunning=TRUE;
OTDRCOMIsMeasurementInProgress(pComdev,
&bRunning);
while (bRunning)
{
    Sleep(500);
    OTDRCOMIsMeasurementInProgress(pComdev,
&bRunning);
}

// load the measurement data into the
// OTDRAPI ...
OTDRCOMGetFileToAPI(pComdev, pOTDRapi,
NULL);

// perform a scantrace locally; maybe set
// parameters before ...
OTDRScanTrace(pOTDRapi, NULL);

// send the data back to the OTDR ...
OTDRCOMSendFileFromAPI(pComdev, pOTDRapi,
NULL);

// store locally ...
OTDRSaveTrace(pOTDRapi, "LOCAL.SOR");

// store remotely ...
OTDRCOMSaveCurrentTrace(pComdev,
"REMOTE.SOR");

// clean up all pointers ...
OTDRCOMClosePort(pComdev);
OTDRCOMFreePort(pComdev);
OTDRFreeLib(pOTDRapi);
```

Outlook

Both libraries will be subject to regular updates, as the Bellcore and the SCPI standards develop further. Bellcore plans to release a new revision of the SOR format before middle of 1998, which must be updated in the OTDRAPI library. Also the remote function set of the HP E60xx and HP E4310 OTDRs is growing.

Another area of development is the number of supported platforms. Currently, all Windows platforms are supported (16-bit only with a large memory model). There might emerge a need for UNIX[®] versions or even real-time operating systems like pSOS + .

The OTDRAPI library uses no Windows-specific functions, so a port to other 32-bit platforms will be quite easy. For the OTDRCAPI library, the RS-232 abstraction layer will need to be replaced when porting to another operating system. For example, the GreenLeaf CommLib approach might not work on a UNIX operating system.

Other elements such as the scantrace algorithm or the loss calculation routines will experience regular improvements that will also be updated in the OTDRAPI library.

Conclusion

The OTDRAPI and OTDRCAPI libraries offer a cost-effective, high-quality solution for fiber-optic system integrators. This was achieved by reuse of the software modules developed for our OTDR products.

This approach provides a fast development time for customers and guarantees that they will benefit from future enhancements of the remote control facilities and the data handling (Bellcore file format and scantrace algorithm) of Hewlett-Packard's OTDRs.

By offering these APIs, we enable the customers to maximize the benefit obtained from their OTDRs.

Acknowledgments

The authors would like to thank the OTDR software team, especially Jürgen Sang, Alf Clement, Joachim Winkler, Jonathan McEwan, and Oliver Berger for their contributions. We also want to thank John Peters at Bellcore.

References

1. J. Nemeth-Johannes, "A Standardized Instrument Programming Language Based on IEEE Standard 488.2," *Autotestcon '89*.
URL: hpswtsvr.lvlld.hp.com/goodstuff/standards/scpi/index.html
2. *Generic Requirements for Optical Time Domain Reflectometers*, GR-196, no. 1, Bellcore, 1995 (Rev. 1, December 1997).
URL: www.bellcore.com
3. P. Ghadayammuri, "A Platform for Building Integrated Telecommunications Network Management Applications," *Hewlett-Packard Journal*, Vol. 47, no. 5, October 1996.

Bibliography

1. T. Born, "OTDR Measurements Harmonize With Bellcore Standard", *Test & Measurement World*, August, 1997, pp. 37-38, *Test & Measurement Europe*, November, 1997, pp. 21-24.

LabView a registered trademark of National Instruments.

HP-UX 10.20 and later and HP-UX 11.00 and later (in both 32- and 64-bit configurations) on all HP 9000 computers are Open Group UNIX 95 branded products.

UNIX is a registered trademark of The Open Group.

Windows is a U.S. registered trademark of Microsoft Corporation.

Microsoft and MS-DOS are U.S. registered trademarks of Microsoft Corporation.

Updating a UNIX Application Suite for the Windows NT World

Thomas W. Hutchinson

Ronald R. Derynck

A project team learned some useful lessons in porting a real-time software platform for industrial applications to an environment that typically runs desktop applications such as word processors, database programs, and spreadsheet applications.



Thomas W. Hutchinson

An R&D project manager at the HP Calgary Product Development Center,

Thomas Hutchinson is responsible for software development of HP RTAP products. He has been with HP since 1987. He received a BSc degree in computer science in 1979 from the University of Calgary. Tom was born in Taber, Alberta, Canada. He is married and has five children. In his leisure time he enjoys hiking, camping, cross-country skiing, and researching family history.



Ronald R. Derynck

Ronald Derynck is an R&D section manager at the HP Calgary Product Development Center.

He was the R&D manager for the HP Enterprise Link and HP RTAP development programs and is now the business team leader for both of these products. He holds BSc (1970) and MSc (1972) degrees in electrical engineering from the University of Calgary. He came to HP in 1986. Ron was born in Calgary, Alberta, Canada, and his interests outside of work include biking and cross-country skiing in the Canadian Rockies.

One of the remarkable trends in software in the 1990s has been the influence of Microsoft® technologies on many industries. The area of industrial automation and supervisory control is no exception. HP has updated a successful UNIX®-based product to respond to this trend while maintaining many of the product's strengths. The product, HP RTAP (real-time application platform), is an example of a real-world industrial application of component software technology.

Background

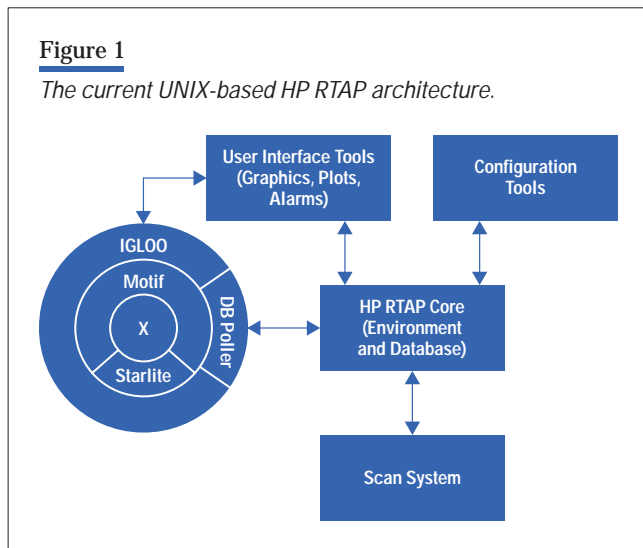
HP RTAP is a complete software framework for building industrial automation systems. It is used in a wide variety of industries and is sold mainly through channel partners, such as system integrators, who focus on a particular industry or geographical area. HP RTAP has been a successful product and has been available on the UNIX operating system since 1990.

Figure 1 shows the architecture of the UNIX-based version of HP RTAP. The core of HP RTAP is a high-performance real-time process database based on object-oriented concepts. It contains a spreadsheet-like calculation engine. It also has a master timekeeper, a scan system for remote devices, and facilities for managing events, detecting alarms, recording historical data, and determining the health of the system.

HP RTAP also contains a complete graphical user interface (GUI) including alarm and trend displays, reports, schematics, control panels, and a suite of

Figure 1

The current UNIX-based HP RTAP architecture.



configuration tools for core and graphical components. The GUI is built on an HP graphics library that is provided for customers to build their own graphical applications. The GUI library, in turn, is built on X-Windows and Motif. Early versions of HP RTAP were on the X10 windows system. Later, ports were done to X11 and Motif. Customers were protected from changes because they were hidden inside the HP graphics library.

The HP RTAP product was first introduced on the HP-UX operating system and soon became known for its open systems focus. It is now supported on several of the most popular UNIX platforms.

This article describes some of the modifications we made to the HP RTAP product to port it to the Windows® NT environment. The goals we had for this project were to:

- Offer the Windows NT market a supervisory control and data acquisition system with the power and flexibility of HP RTAP
- Enhance the usability of the UNIX product by allowing full integration with the Microsoft desktop on the client side
- Support HP corporate goals for growth and integration of both UNIX and Windows systems, taking advantage of the strengths of each.

Moving to Windows NT

Microsoft Windows NT has recently been gaining acceptance as a platform for industrial automation systems. However, as is often the case, there is a difference of

opinion among users. Some users with mission-critical applications are not ready to move to Windows NT. The UNIX system, on the other hand, has been on the market for over 10 years and is a mature, stable, and robust operating system. Other users are ready to embrace the Microsoft world fully because this environment better satisfies their need to incorporate real-time information into the decision making process. Still others want to keep the UNIX operating system for their servers and provide the benefits of the Microsoft world on their desktops.

We considered all this information in our plans for a product on Windows NT. Our first major decision was to split the product structure along client/server lines. There was general agreement that since Microsoft dominates the desktop world, HP RTAP needed a Windows interface. On the other hand, opinion was still divided on the server side, so we chose to support both UNIX and Windows NT operating systems for servers and allow our Windows clients to connect to both.

HP RTAP is well known and respected for its core data server, so it was ported to Windows NT with existing functionality but with a different graphical user interface. Although third-party products are available to allow a port with the Motif look and feel, we decided to rebuild

Glossary

ActiveX. A COM-based framework for software component integration. An ActiveX Control is a prebuilt, reusable software component that performs a particular function (often visual and interactive) within the context of container applications such as VisualBasic or a Web browser.

CORBA (Common Object Request Broker Architecture). An object request broker that provides the services which enable objects to make and receive requests and responses in an object-oriented distributed environment.

COM (Component Object Model). A binary programming interface standard that allows components written separately (even in different languages) to communicate correctly.

DCOM (Distributed Component Object Model). The COM distributed wire protocol.

OLE (Object Linking and Embedding). Windows' compound document protocol that allows one document to be embedded within or linked to another document.¹

parts of the HP RTAP user interface and purchase other parts based on the new Microsoft model for component software.

With this philosophy, we created a graphical user interface based on Microsoft COM (component object model). It provides powerful drawing and animation capabilities, Visual Basic for applications, and a set of industrial ActiveX controls that users can supplement by adding their own components. For example, a user can select an ActiveX control that looks and acts like a gauge to keep track of filling a vessel. We also added a new configuration tool that leverages from available Microsoft usability features and emphasizes the class-based nature of the HP RTAP database. This allows complete integration with other tools on the Microsoft desktop. All client-side tools use a new ActiveX component to communicate with the HP RTAP server on either Windows NT or UNIX operating systems.

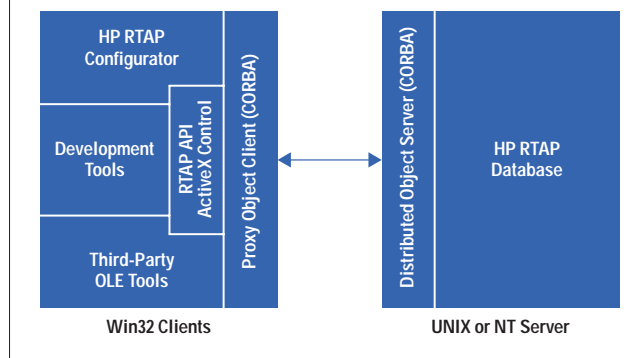
In addition to ActiveX, another important Microsoft technology is the ODBC (Open Database Connectivity) standard for relational databases. Users want to create reports using their regular spreadsheet, database, or word processor tools. They do not want to learn a special report generator. The new HP RTAP ODBC interface takes advantage of the class-based nature of the HP RTAP database, allowing useful queries of similar points as if they were relational tables. For example, HP RTAP will now accept an SQL SELECT statement for a query like "show me the site locations for all gas wells having a flowing temperature greater than 20 degrees Celsius."

We emphasize that the PC client-side environment is truly a Microsoft world so that all the RTAP PC clients integrate perfectly with Visual Basic 5, Visual C++ 5.0, Delphi, Access, Excel, and all the other tools and applications users expect. Access to the HP RTAP database and the rest of the server environment is through ActiveX components and ODBC. Users get all the benefits of integration on the desktop and access to the servers on either NT or UNIX systems. This allows existing customers with UNIX-based HP RTAP systems to add the PC clients whenever they are ready.

An important building block that enables this client/server split is the HP RTAP API ActiveX control (see **Figure 2**). This ActiveX control is the mechanism by which all PC clients connect and communicate with the server. All HP RTAP database components are represented as objects

Figure 2

New HP RTAP clients based on Microsoft OLE.



with methods and properties like any other ActiveX object. This enables the full power of the HP RTAP server-side API, but with a much simpler object-oriented interface. For example, **Figure 3** shows a simple Visual Basic code fragment for reading a database value. The API supports more than simple reads and writes. It allows for more complex operations such as caching and large grouped data transfers. We chose to create a full object model, allowing advanced functions such as configuration of the database from a PC client through the ActiveX interface.

The interface seen by the user of the HP RTAP ActiveX control is the familiar OLE-style interface seen in Visual Basic and other desktop tools. Thus, a user, in say Visual Basic, sees an OLE style API. On the other side of the ActiveX control (hidden from the user) is the infrastructure for communicating with servers. We used CORBA as the underlying platform because of its portability and maturity.

Climbing the Learning Curve

Connecting PC clients was the largest technical challenge that we faced in this migration project. Our development team had years of UNIX experience, but ActiveX and CORBA were new to us. We hired some local contractors to alleviate this problem, but our collective learning curve was still quite steep. The job turned out to be much bigger and more difficult than we or the contractors expected.

One major challenge was to create an object model that would represent all the aspects of HP RTAP and be exposed to users (for example, database points and attributes). The interface had to be consistent and simple to understand but powerful enough to support configuration

Figure 3

Simple Visual Basic code fragment for reading a database value.

```
'General declarations for the RTAP environment & db objects
Dim g_env as HPRTAPLib.Environment
Dim g_db As HPRTAPLib.Database
Dim g_pt As HPRTAPLib.Point
Dim g_attr As HPRTAPLib.Attribute

'Open a connection to the environment and the database when the
'form is loaded. The environment name and host computer are
'specified in the properties of the RTAP Custom Control.

Private Sub Form_Load()

    'Connect to the RTAP environment
    Set g_env = HpRtap1.Environment
    g_env.Connect

    'Connect to the database and read the current value
    If g_env.Connected Then
        Set g_db = g_env.Database
        Set g_pt = g_db.PointByAlias("PT-101")
        Set g_attr = g_pt.Attribute("process value")
        Label1.Caption = g_attr.Value
    End If
End Sub
```

and fast access to values. The resulting object model has about 40 classes and over 300 methods and properties.

After the object model was defined, we had to implement it in the ActiveX control and CORBA layers. Aside from learning how ActiveX and CORBA work, developing this code was not particularly difficult. We were familiar with the HP RTAP API, and programs that receive CORBA requests on the server side were fairly straightforward to write and implement.

Another challenging area was the memory allocation model. Both COM and CORBA base their memory schemes on the concept of reference counts. In principle, it is simple: for every created object, a counter is maintained on how many times the object is used, and when that count drops to zero, it is safe to delete the object. However, in practice it is not so simple, and an error can have serious consequences. Freeing an object too soon can cause invalid memory references later, leading to a disastrous failure of the program. Failing to free the object can create a memory leak, which leads to a slower death but is still fatal in a program that must run for weeks or

months. In particular, CORBA implementations do not automatically forward or synchronize reference counts between the client and the server, which became quite difficult for us to manage.

Our first implementation uses CORBA because it is a mature technology and is available on Windows NT and all the UNIX platforms that we support.

Scanning the Results

Porting the HP RTAP server to Windows NT maintains all its UNIX functionality and existing API. It should be noted that this server, even on a PC, still retains some of its UNIX behavior. We believe this is a reasonable trade-off because it provides the following advantages:

- It maintains compatibility with UNIX servers, allowing the same clients to work with both.
- It gives customers a migration path for their current and future server-side applications.
- It provides scalability that PC-only software cannot match.

These modifications to HP RTAP met our goals for the project.

Conclusion

This migration from UNIX to Windows NT has taught us a lot about new technologies that will be useful in future development efforts. In particular, we learned two surprising lessons about this type of project.

First, it is not any easier to develop complex software on Windows NT than it is on a UNIX operating system. There is a tremendous framework available for making applications easier for users to use, and the infrastructure is definitely improving for developers as well—at least for certain kinds of applications. For the kinds of industrial automation software that we write, our development schedules cannot be made significantly shorter just because we are on Windows NT.

Secondly, we have learned useful lessons about using third-party software to develop applications. We have gone from an environment in which we are in control of everything to one in which we rely on many pieces of

software from at least six vendors (not including Microsoft). This has many implications for release scheduling, documentation, and reliability of the end product. It has caused problems and will undoubtedly continue to do so. However, in the final analysis, we are convinced that there are many gains to be made from Microsoft's component software model and from generally being able to use software created by others.

Reference

1. A. Freedman, *The Computer Glossary*, AMACOM, 1995, p. 276.

UNIX is a registered trademark of The Open Group.

Microsoft is a U.S. registered trademark of Microsoft Corporation.

Windows is a U.S. registered trademark of Microsoft Corporation.

Integrating Real-Time Systems with Corporate Information Systems

Ronald R. Derynck

Thomas W. Hutchinson

Integrating distributed systems involves more than just connecting different communications technologies. It also involves connecting different information environments.

Traditionally, supervisory control and data acquisition systems have been designed to meet operational needs in the field and on the factory floor. Little attention has been given to how well these systems will integrate with the corporate information infrastructure. Today, every industry has felt the impact of several major trends: globalization, consolidation, deregulation, rapid cycle times, compressed time-to-market, and new products and services. The net effect is that “islands of automation” in the field or factory no longer meet business needs. Since 1990, HP has been exploring new ways to build distributed systems that integrate measurement and control systems with the business enterprise.

The Information Gap

Linking real-time systems to an enterprise system is a difficult undertaking, usually requiring custom code at each end. However, such links will have to become pervasive to give employees access to the real-time information they need to make timely business decisions with confidence.

It is common to have two central information systems: a traditional management information system (MIS) for business functions and a technical information system (TIS) for mission critical real-time measurement and control. These systems are typically managed by separate groups.

Establishing connections between the two systems is relatively simple. The real challenge is achieving true information integration. This is because of the four types of gaps between the MIS and TIS worlds (see **Table I**).

Table I

Information Gaps between Management Information Systems and Technical Information Systems

Type	Management Information System	Technical Information System
Time	Months, weeks, days	Seconds or less
Information	Transactions	Real-time events
Execution	Oriented to planning and scheduling	Oriented to measurement, control, and engineering
Culture	Driven by business needs	Driven by process needs

The width of these gaps is further illustrated by a recent study in which 43 percent of the companies surveyed reported only minor or occasional dialog between their management information systems and technical information systems groups.¹

Bridging the Gap

Numerous deployed and emerging technologies promise to provide information integration (see **Figure 1**).

Most of the familiar technologies are in the lower left corner, offering point-to-point connections as opposed to real integration. These include the original DCE products and the more recent CORBA and DCOM technologies, which enable objects to exchange messages over a network. However, interoperability and standard objects are still a long way off in these technologies.

From an end-user's perspective, better integration is achieved by the Microsoft® ActiveX component model. This provides a substantial level of desktop integration, allowing sophisticated applications to be built from modular components. However, ActiveX still does not serve the total integration needs of an enterprise.

The World Wide Web provides another promising form of integration, with documents interlinked across a network of systems. The huge popularity of the Web is encouraging rapid extensions to the technology, including dynamic content of web pages and push rather than pull distribution. However, building large distributed systems without managing huge numbers of point-to-point connections requires a true peer-to-peer communications environment.

New Connections

The publish/subscribe messaging model implements a scalable peer-to-peer communication architecture. It sets up an information pool in which publishers broadcast information identified by topic and applications subscribe to receive messages of interest. This is built on top of the Internet Protocol, allowing access to the large installed base of TCP/IP networks.

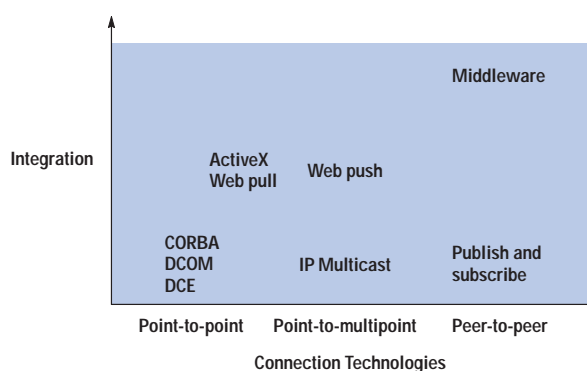
Information Integration

All these technologies achieve varying degrees of connection but fall short of true information integration. A new class of software, often called *middleware*, is emerging to connect MIS applications, such as SAP's R/3, to real-time systems. These applications are built on top of the connection technologies and provide an additional degree of integration.

When only two applications need to be integrated, it is tempting to build a simple custom interface. However, as the number of applications increases, this approach results

Figure 1

Integration technologies.



CORBA = Common Object Request Broker Architecture
DCE = Distributed Computing Environment
DCOM = Distributed Component Object Model

in several different interfaces, making development and maintenance difficult. Instead, properly designed middle-ware implements a software bus so that each application shares a single interface to the bus. Good middleware also provides a rich set of object-oriented features that are reused by each application. Such features include easy graphical configuration, object model reconciliation, data transformations, flexible triggers, one user interface for integrating multiple products, code reuse to simplify adaptation to new applications, spooling and caching of data, diagnostics and message tracing, and failure detection and recovery.

HP Enterprise Link² currently integrates real-time systems with the SAP PP-PI and PM modules and with the Oracle™ database.

Putting It All Together

With all these technologies and products to choose from, the challenge is to decide which one to use for a particular distributed system. As for most classes of problems, there is no single “one-size-fits-all” solution. Rather, each of these technologies has a role to play in linking front-line measurements to backroom systems, allowing users to have access to the real-time information they need. Examples of these products include:

- The HP RTAP (real-time application platform) supervisory control and data acquisition system is a complete software framework for building industrial automation systems. It also serves corporate needs via the HP Enterprise Link product.
- A desktop ActiveX interface that uses CORBA for communications with RTAP servers running on a variety of UNIX® platforms and on Microsoft Windows® NT.

Conclusion

There are many existing and emerging connection technologies that help to build distributed systems. However, integration means much more than just connection. On top of any connection technology, considerable work is needed to achieve real information integration. Middle-ware products help to fill this gap.

One important factor will characterize the successful companies of the future: the ability to provide real-time information to decision makers at all levels, anywhere, any time. The integration of mission-critical, real-time applications with corporate information systems will be essential.

References

1. C. Moore, “IT in the Electric Power Industry,” *Information Technologies for Utilities*, September 1996, pp. 48-52.
2. K. Jennyc, “Linking Enterprise Business Systems to the Factory Floor,” *Hewlett-Packard Journal*, May 1998, Vol. 49, no. 2, pp. 62-73.

UNIX is a registered trademark of The Open Group.

Microsoft is a U.S. registered trademark of Microsoft Corporation.

Windows is a U.S. registered trademark of Microsoft Corporation.

Oracle is a U.S. trademark of Oracle Corporation, Redwood City, California..

Ronald R. Derynck

Author's biography appears on page 21.

Thomas W. Hutchinson

Author's biography appears on page 21.

New Approaches to Creating and Testing Internationalized Software

Harry J. Robinson

Sankar L. Chakrabarti

Creating high-quality software that runs in any language is a big challenge. By changing our development process to stress early defect detection and by using the World Wide Web as a collaboration tool, we have dramatically improved the quality of our internationalized software.

Most software companies want to sell their software in every country in the world. Since users prefer to use software in their native language, it makes good marketing sense to develop software that can run in those languages. After the initial investment has been made to write and test software in English, a software company can make significant profits by reselling the same software to other countries if the cost of conversion into other languages can be kept low.

Traditionally, software testing occurs at the end of the development cycle. This kind of testing works against creating high-quality software. When bugs are found late in the cycle, there is little time to fix them. This is especially true in internationalized software development where the developers, testers, and translators are spread all over the world. Our new approach allows a team



Harry J. Robinson

A software design engineer at the HP Corvallis Imaging Operation, Harry

Robinson was a test engineer for the HP Common Desktop Environment internationalization project. He recently left HP to become a lead test engineer at Microsoft Corporation. He has a BA degree in religion (1980) from Dartmouth College and BSEE (1985) and MSEE (1988) degrees from Cooper Union. He is interested in all aspects of software testing. Born in Staten Island, New York, he is married and has three children.



Sankar L. Chakrabarti

Sankar Chakrabarti is a member of the technical staff at the HP InkJet

Business Unit. Currently, he is responsible for developing software for a print quality testing tool. Sankar received a doctorate in chemistry in 1974 from the Tata Institute of Fundamental Research in Bombay, India. He joined HP in 1981 after receiving an MS degree in computer science from Oregon State University. Born in Azimganj, West Bengal, India, Sankar is married, has two children, and enjoys traveling and hiking.

to test the software during the entire process and to release foreign language versions simultaneously with the English version.

Developing Internationalized Software

The I18N Approach

One common method of creating internationalized software at a reasonable cost is called I18N.* The essence of the I18N approach is to separate the executable code from any character strings that the user will see. User messages are placed into files called message catalogs. Two numbers, the set number and the message number, index each string in the message catalog. The executable code uses these numbers to retrieve strings.

For example, every C language programmer knows the classic hello, world program:

```
hello.c:
main()
{
    printf("hello, world\n");
}
```

In the I18N methodology,¹ this program would be written as follows:

```
hello.c:
main()
{
    my_cat=catopen("hello.cat", NL_LOCALE);
    printf(catgets(my_cat, 1, 5,
        "hello, world\n" ));
    catclose(my_cat);
}
```

The program accesses the string "hello, world\n" by retrieving set 1, message 5 of the "hello.cat" message catalog file.

```
hello.cat:
    $set 1
    5 hello, world\n
```

The string "hello, world\n" that appears in the printf statement is a default string that is used if no message catalog file can be found.

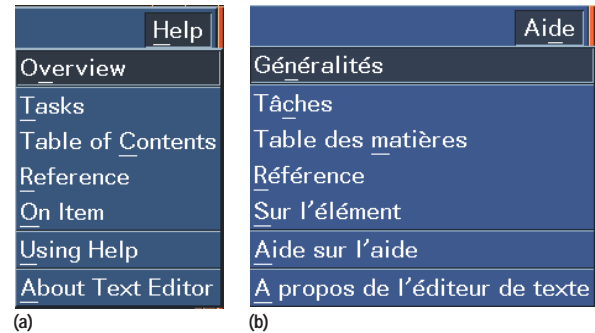
Separating executable code from user-visible strings is very useful when working with translations. If we want to run our hello, world program in French, a translator merely changes the string in the message catalog to:

```
$set 1
5 bonjour, le monde\n
```

* I18N = I[nternationalizatio]N. 18 is the number of letters between the I and N.

Figure 1

A typical application Help menu in (a) English and (b) French.



Users running the hello.c program with the translated message catalog will see bonjour, le monde as their output. No changes are made to the executable code to support the French version. Only the user interface strings need to be translated. The executable code remains unchanged.

Figure 1 shows an example of what a typical application Help menu looks like in both English and French. The same executable code was used to generate each menu, and only the message catalog was changed.

Process Flow

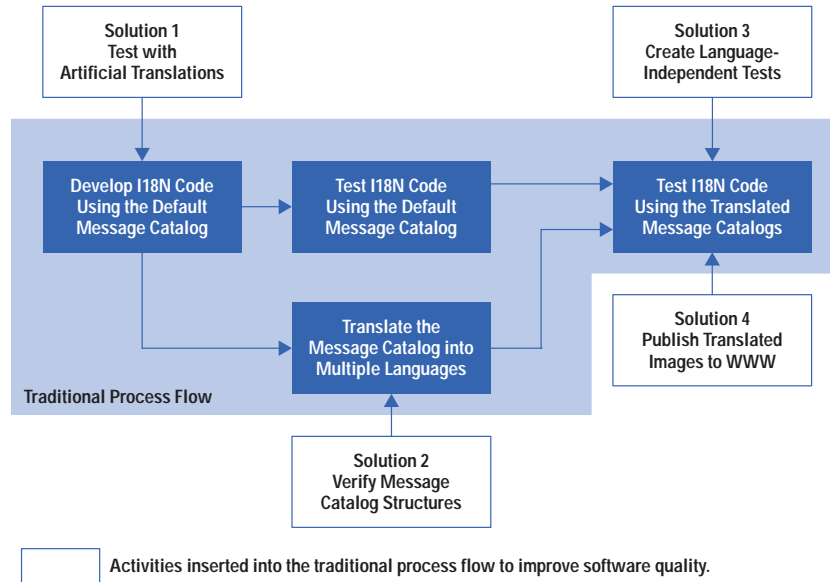
The idea behind creating internationalized software is conceptually simple, especially when the number of languages is small and the application is as simple as hello.c. On the other hand, producing real-world applications in a dozen languages can pose several challenges to a development team.

The shaded area in the diagram in **Figure 2** shows the traditional process flow for developing an internationalized application.

1. The programmers write an application with the appropriate I18N calls for fetching strings from the message catalog. They also produce the original message catalog in English.
2. The message catalog is sent to translators (called *localizers*) who translate each string into a target language, such as French.
3. The application (with the original message catalog) is delivered to the test team, who verify that everything works correctly.

Figure 2

Process flow for developing an internationalized application.



4. The localizers provide the translated message catalogs to the test team. The testers must now verify that the application works in the intended languages.

The Team

Our development team designed and implemented the graphical user interface for Hewlett-Packard's UNIX[®] workstations. This interface is made up of several applications and runs in a dozen different languages: English, French, Spanish, Italian, German, Swedish, Korean, two forms of Japanese, and three forms of Chinese. The sheer scale of our work causes problems in creating internationalized software because of the wide range of skills and resources needed and the distances between involved parties.

The Programmers. Our entire programming team is located in Corvallis, Oregon. They are software specialists and are not expected to know multiple languages. After they have written their internationalized code, they must wait for the message catalogs to be translated before they can verify that their I18N features are correctly implemented.

The Localizers. Our localizers live in widely separated areas of the world. Most of them are contractors who have never met the rest of our development team. The localizers are not expected to have programming or testing expertise. In fact, they may not even have UNIX workstations on which to run the applications. Often, because the applications they are translating are still in development, they are unlikely to be familiar with the application when they are creating their translations.

The Testers. The test team is located in Corvallis. They are test specialists and are not expected to speak multiple languages. Although I18N methodology is a boon for programmers, it can be a nightmare for the testers. In regular software testing, there is rarely enough time to test an application thoroughly. In the I18N arena, the test team must provide assurance that the software operates correctly in the dozen languages in which it will run. Furthermore, it is very difficult to verify correct operation in a language that one does not speak because mistakes that would be obvious to a native speaker can easily pass unnoticed by the test team.

Challenges and Solutions

We have developed several strategies to deal with many of the challenges we have encountered while creating I18N-enabled code.

Challenge 1: Developers cannot test the I18N-enabled code easily. There are several common mistakes people make when writing I18N-enabled code. One mistake is to neglect to leave enough room in message buffers for the translated message string. Some languages, such as German, require more space than English for the same message. The length of the translated message string cannot be known until run time, though a good rule of thumb is to allow for 60% text growth during translation. If a message string still exceeds the buffer length provided, the program should usually truncate the string.

A second common mistake is when developers neglect to accommodate languages, such as Japanese, that require two bytes to store a single character. Most Western languages require only one byte of storage per character, but several Far Eastern languages have large character sets and need more than one byte per character. If the code does not handle double-byte characters, the results could range from corrupted characters to a crash of the application.

It would be very handy to provide a way to test I18N-enabled code early in the development process, perhaps even as soon as the code is written. The chief difficulty in early testing of I18N-enabled code is that an actual translation may not be available. Message catalog translation is time-consuming. The development team may have to wait several weeks before getting a translated message catalog back from a localizer, losing precious testing and debugging time.

Solution 1: Test with artificial translations.² Our solution is to construct artificial message strings that mimic the kinds of problems we see in real translated message strings. This instantaneous creation of a translated message string against which to test our software provides us with quick feedback about how our application will perform with translated components.

For instance, to simulate languages with long text strings, we created a message catalog in a language we call the *Swedish chef*. Using a freeware Internet utility called the Encheferizer,³ we appended long nonsense strings onto each English string. The results can be seen in **Figure 3**.

Figure 3

Nonsense strings appended to English words to create the Swedish chef language.



A screenshot of a menu bar with a dark blue background and white text. The menu items are: Help [Help Bork! Bork! Bork!], Overview [Oferfeeoo Bork! Bork! Bork!], Tasks [Tesks Bork! Bork! Bork!], Table of Contents [Tebler—a ooff Coontents Bork! Bork! Bork!], Reference [Refference—a Bork! Bork! Bork!], On Item [On Item Bork! Bork! Bork!], Using Help [Useeng Help Bork! Bork! Bork!], and About Text Editor [Ebooooot Text Ideetoor Bork! Bork! Bork!].

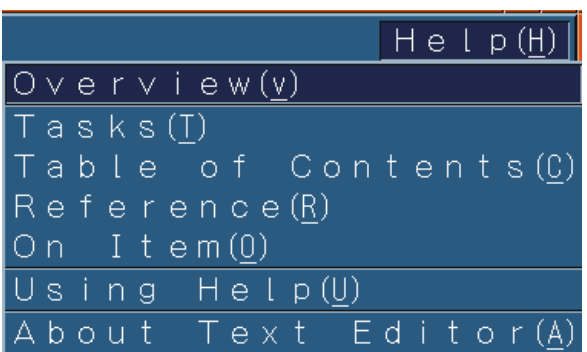
Each Encheferized string is at least double the size of the corresponding English string.

Likewise, to simulate double-byte characters such as those used in Japanese, we used a small program that maps ASCII characters into a double-byte format as shown in **Figure 4**. This translation was easy to use and allowed us to detect whether the code handled double-byte strings properly.

Despite these tools, many developers still felt reluctant to test in a language that they did not know. To overcome this reluctance, we demonstrated that it is possible to test even in the worst case imaginable: we created a Klingon target language.⁴ Words, chosen at random from a Klingon version of Shakespeare's Hamlet, were inserted into an application's message catalog as shown in **Figure 5**. Even though this version of the application might not make

Figure 4

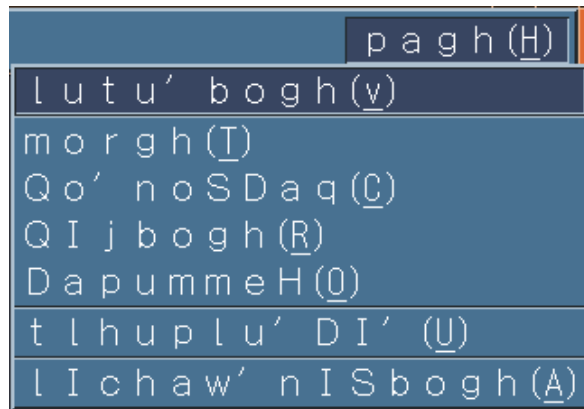
Text strings mapped into a double-byte format to simulate double-byte characters.



A screenshot of a menu bar with a dark blue background and white text. The menu items are: Help(H), Overview(v), Tasks(T), Table of Contents(C), Reference(R), On Item(O), Using Help(U), and About Text Editor(A). The letters in parentheses are underlined.

Figure 5

An excerpt from a Klingon version of Hamlet.



sense to anyone, it can still be tested. This test case provided one more level of confidence to our testing.

In all three cases, we used small scripts to create message catalogs from the default English catalog so that the “pseudo-translated” messages were available as soon as the code was ready to be tested (see **Figure 2**). Because the translations we chose were simple to use, somewhat whimsical, and not at all intimidating, programmers found it easy to perform I18N testing.

Challenge 2: Translators can introduce structural defects into the message catalogs. Each message catalog has a structure. Sometimes the structure is as simple as the set and message numbers, and at other times it can be complex. If the structure of the message catalog is changed during translation, the software will not behave correctly. For instance, suppose the original English message catalog contains:

```
$set 1
5 hello, world\n
```

and the French translation contains:

```
$set 1
55 bonjour, le monde\n
```

The message number has been inadvertently changed from 5 to 55. The application will not work correctly because it cannot find the translated string. These types of mistakes are very hard to catch because of the complexity of some message catalog structures.

Solution 2: Verify message catalog structures. We created small utility programs to verify catalog structures automatically.⁵ In our organization, we call these utilities *poka-yokes* after the Japanese quality assurance method that inspired their use.⁶ These programs often take less than an hour to write. One typical poka-yoke program might check that each message in the translated catalog corresponds to a message in the original English catalog. Such a check will detect the error in the above example in which the message number is changed from 5 to 55.

These utilities are surprisingly effective. When we ran the utilities on eight applications that had been translated into twelve languages, we found 833 defects in the message catalogs. An even bigger benefit was that the utilities did not need an application. The message catalogs could be checked as soon as they were received from the localizers, and any defects could be fixed immediately.

Poka-yoke scripts are also very useful for finding mistakes overlooked by visual verification. For example, in the French menu in **Figure 1b** there is an error in the “shortcut keys” or mnemonics designated for the last two items in the menu. Shortcut keys must be unique within a menu. The letter A is designated as the shortcut key for both of the last two items, violating the uniqueness rule. People often miss these type of errors, but a poka-yoke script can catch them automatically.

Challenge 3: Testers cannot manually test each application in a dozen languages. After the localizers return the translated message catalogs, it is time to system test the applications in each language. Testing software applications thoroughly is always a challenge. Trying to test the same application in twelve languages can be a catastrophe.

For instance, if we wanted to test the output of the hello.c program listed above, we would typically create a test case that looked something like this:

- Step 1: Set the target language to English
- Step 2: Run the hello.c program
- Step 3: Verify that the output is hello, world

If we wanted to test the French version, we would need to change the test case:

- Step 1: Set the target language to French
- Step 2: Run the hello.c program
- Step 3: Verify that the output is bonjour, le monde

What do we do when we need to test the same program in the other ten languages? Will we need to write a dozen versions of each test? Also, how will we run the tests? Hiring additional people to test the foreign language versions is costly, and traditional record-and-playback test methods do not port well across languages. What is needed is an automated test method in which the cost of testing does not become prohibitive as the number of supported languages grows.

Solution 3: Create language-independent tests.⁷ We determined that I18N-enabled tests were needed to test I18N-enabled applications. So, instead of creating a multitude of static tests that would each check for a different string such as hello, world or bonjour, le monde, we created tests that could use the application's own message catalogs to verify output. When an internationalized test needs to verify a program's output, it retrieves the expected message from the message catalog just as the application does. The new I18N form of the automated test for the hello world program would look as follows:

- Step 1: Set the target to the desired language
- Step 2: Run the hello.c program
- Step 3: Retrieve the string stored in set 1, message of hello.cat
- Step 4: Verify that the program's output matches the retrieved string
- Step 5: Choose the next language to test
- Step 6: Repeat until all languages have been tested

This approach can be used to verify that the program is working correctly by iterating through all available languages. Internationalized tests are particularly useful in automatically verifying basic functionality.

Challenge 4: Testers cannot always detect errors in unfamiliar languages. One significant problem in testing internationalized software is that testers unfamiliar with a language usually cannot see errors that might be obvious to

a native speaker of the language. For example, **Figure 6** shows part of an application window in Japanese. The string in front of the (V) is transliterated Hyoji and means View. **Figure 7** shows that same window except that the string in front of the (V) has been corrupted by a typo in the script that was used to compile and build the application. Thus, the string in **Figure 7** is meaningless.

This corruption error is immediately apparent to someone fluent in Japanese. Yet, how many non-Japanese speakers would recognize the error, especially since the corrupted form of the string is still in Japanese?

As mentioned above, we don't require that our software testers know multiple languages, and it is certainly unlikely that we could find a test team fluent in the dozen languages our software supports. On the other hand, the localizers who do speak those languages are unlikely to see the error because they do not have the current revisions of the code or the UNIX workstations on which to run the software. Even if they did have the code and a workstation, it would be difficult to know whether they had adequately exercised the application to display all the relevant screens. Previously, one common solution was to fly the translators to our Corvallis site and have them spend several days watching testers execute tests for their particular language. This method was inconvenient and costly.

How could we provide a medium of collaboration between the testers who run the software and the language experts who could judge whether the output was correct?

Solution 4: Publish translated images to the World Wide Web.⁸ The World Wide Web can be used to distribute test outputs to those who can help judge if the output looks correct. We call this approach a "traveler tour" because it reminds us of a traveler who visits other countries and then returns home and displays the pictures taken on the trip.

Figure 6

Part of an application window in Japanese containing the correct string preceding the (V).



Figure 7

The same character string shown in Figure 6 but with an error in the string preceding the (V).



Figure 8

HTML version of an English help page used for comparing screen images during automated testing.

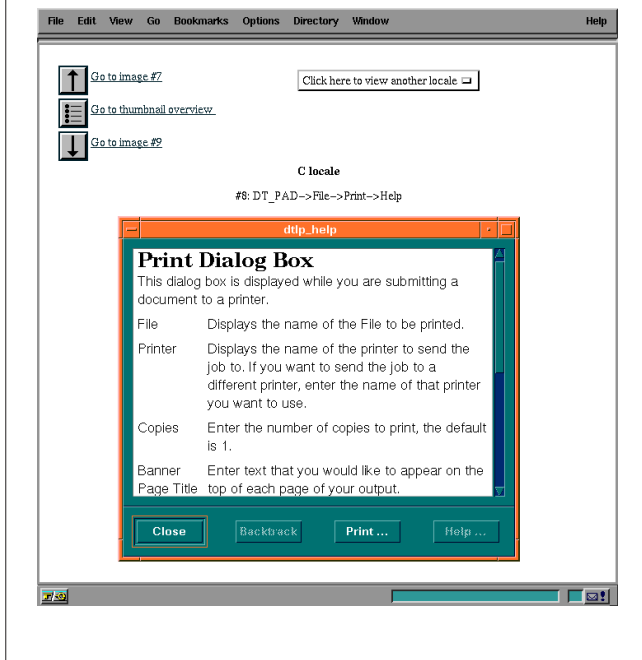
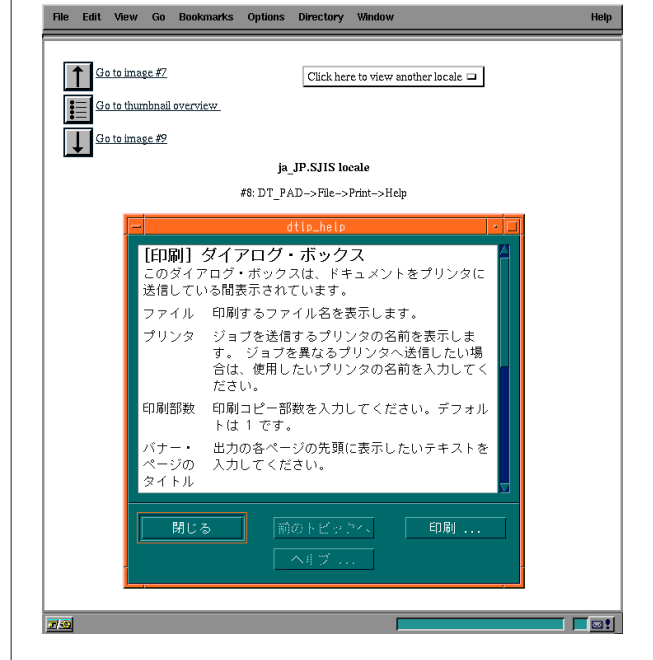


Figure 9

HTML version of a Japanese help page used for comparing screen images during automated testing.



Using the I18N test techniques mentioned in solution 3, we set the target language and drive the application to display various user screens. As each screen is displayed, we capture the image to a file. After all desired images have been captured in all languages, we run a script that creates HTML pages automatically from the images.

Figures 8 and 9 show the HTML version of English and Japanese help pages. The format allows Web page visitors to move easily between the different language versions of an image, as well as move through the sequence of images in a single language.

After the application's images have been captured and moved to the World Wide Web, we invite interested parties, such as the translators and our partner labs, to access the Web pages and give us feedback about whether the translated applications are correct. This arrangement is very useful to all the teams. The testers can ensure that all the relevant screens are displayed, and the language experts can view the output without the expense and administrative overhead of maintaining workstation test systems at their sites.

The traveler tour approach also helps localizers translate the original default English messages into the desired foreign languages. Previously, the localizers were only provided with the message catalogs, which were difficult to translate because the user messages lacked the context of the application. Now, we send the localizers a traveler tour of the application in English along with the message catalogs.

Results

By changing our development and testing processes and creating tools to support early detection of defects, we have revolutionized the way internationalized software is developed at Hewlett-Packard. With these changes:

- Developers can test the I18N features of their code immediately.
- Poka-yoke utilities can catch message catalog defects at the translation stage.
- Language-independent test suites permit automatic testing in all languages.

- Traveler tours allow testers and localizers to work together to detect subtle translation bugs.

This new approach to developing I18N software has reduced the resources needed for testing by a factor of five. It has eliminated the travel costs previously incurred in bringing translators to the development site to assist in testing. The approach has dramatically increased the quality of our internationalized software while at the same time decreasing the time devoted to development, translation, and testing.

Finally, our translators and our partner labs are so pleased with the outcome of this work that they have asked that these changes be incorporated into our regular delivery mechanisms.

Conclusion

Internationalized software has great advantages for the marketplace and is a worthwhile and growing trend, but high quality levels can only be achieved if internationalization is integrated with the rest of the software development process. Current development models do not encourage easy integration of coding, localizing, and testing. We have designed tools to promote early detection of defects and collaboration among the different groups involved in software creation.

Acknowledgments

The authors would like to thank Ken Bronstein, Arne Thormodsen, Dan Williams, and Barbara Wingert-Burbach for their help in developing and promoting the techniques used in this testing program.

References

1. T. McFarland, *X Windows on the World*, Prentice-Hall, 1996.
2. H. Robinson and A. Thormodsen, "Parlez-Vous Klingon? Testing Internationalized Software with Artificial Locales," *Proceedings of the 1997 Pacific Northwest Software Quality Conference*, pp. 185-195.
3. J. Hagerman, *Ze Svedish Chef*, <http://www.almac.co.uk/chef/chef.html>
4. M. Okrand, *The Klingon Dictionary: English/Klingon Klingon/English*, Pocket Books, 1992.
5. H. Robinson, "Using Poka-Yoke Techniques for Early Defect Detection," *Proceeding of the Sixth Annual Conference on Software Testing, Analysis and Review*, 1997, pp. 119-142.
6. S. Shingo, *Zero Quality Control: Source Inspection and the Poka-yoke System*, Productivity Press, 1986.
7. S. Chakrabarti and H. Robinson, "Testing CDE in Sixty Languages: One Test Is All It Takes," *Proceedings of the Fourteenth International Conference on Testing Computer Software*, 1997, pp. 419-428.
8. S. Chakrabarti and H. Robinson, "Catching Bugs in the Web: Using the World Wide Web to Detect Software Localization Defects," *Proceedings of the Tenth International Software Quality Week*, 1997, p. 7A2.

UNIX is a registered trademark of The Open Group.

Comparison of Finite-Difference and SPICE Tools for Thermal Modeling of the Effects of Nonuniform Power Generation in High-Power CPUs

Jeffrey L. Deeney

C. Michael Ramsey

This paper describes a thermal study of junction temperature variation across the surface of a large CPU resulting from nonuniform power generation. Results from Flotherm finite-difference thermal analysis software were compared to results from a SPICE simulation. Both simulations provided results close to measured values. Each tool offered strengths and benefits in different areas.

Because of unevenly distributed functionality in large integrated circuits, power dissipation can vary significantly across the surface of a device. In very high-power devices, these power variations can lead to significant variations in junction temperature across the die surface. If these temperature gradients are not fully understood or anticipated, product reliability may be degraded.

The HP PA 8000 microprocessor is a high-performance implementation of HP's PA-RISC computer architecture.¹ It is the central processor of the top-performing models of the HP 9000 C-class and J-class workstations and K-class servers. With a die size of 17.7 mm by 19.7 mm, it is the largest integrated circuit designed by HP to date. At the announced operating



Jeffrey L. Deeney

Jeff Deeney is a lead thermal design engineer for packaging development at HP's High-Performance Systems Division. He joined HP in 1981 after receiving his BSME degree from Washington State University. He is married, has three children, enjoys backpacking and climbing, and is active in public land use issues.

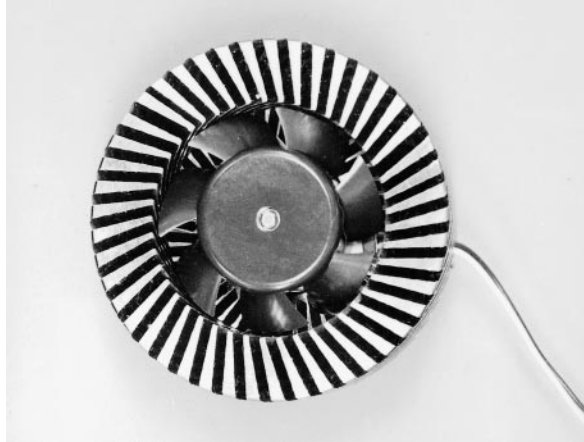


C. Michael Ramsey

Mike Ramsey is a member of the technical staff involved with presilicon verification of advanced processors at HP's VLSI Technology Center. With HP since 1980, he received his BSEE and MSEE degrees from Texas A&M University in 1979 and 1981. He is married, has a son, and is a student of Tae Kwon Do.

Figure 1

Fan/heat-sink forced air cooling solution.



frequencies of 160 MHz, 180 MHz, and 240 MHz, its power dissipation is also quite high. Removing this power presented challenges at both the chip and system levels and was anticipated early in the design process. However, results were still not without surprises. Specifically, the large size of the PA 8000 die and the nonuniform distribution of its heat-generating circuits led to unanticipated temperature differences across the die. The study described in this paper investigated how the thermal characteristics could be better modeled.

The PA 8000 is packaged in a 1089-pin ceramic land grid array (LGA) using solder bump bonding with epoxy underfill onto an alumina substrate. The back of the die is attached to a sintered copper-tungsten lid by means of silver-filled epoxy for enhanced thermal performance. The entire package is socketed onto its host printed circuit module. Different heat sink solutions are used in the various systems. In the C-class workstation, the PA 8000 is cooled by a fan/heat-sink forced air cooling solution (see **Figure 1**). In the J-class workstation and K-class server, heat is conducted through a three-pipe heat pipe assembly cooled by the cabinet's forced air flow (**Figure 2**).

Conventional IC design methodology partitions the functions of the PA 8000 into localized units, as shown in **Figure 3**. The floating-point processing unit is one such unit. The floating-point unit is a specialized coprocessor that delivers improved performance to applications using floating-point arithmetic operations. It is the largest source

Figure 2

Heat pipe mounted to printed circuit board.

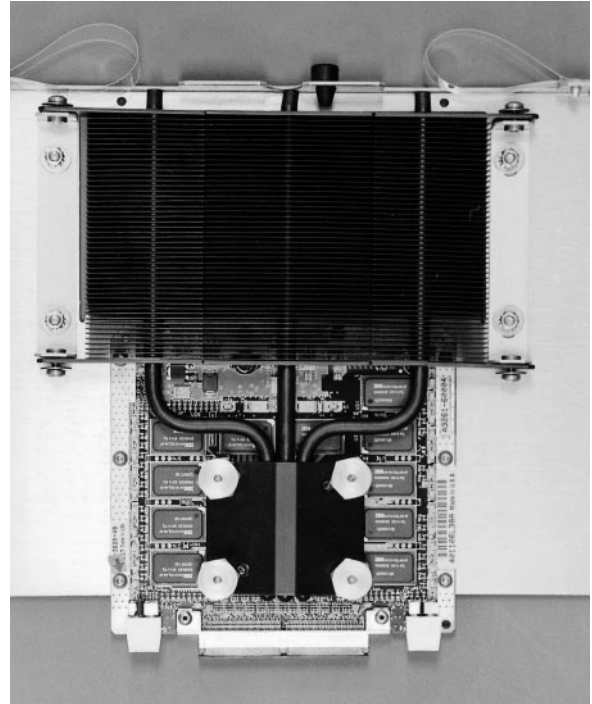
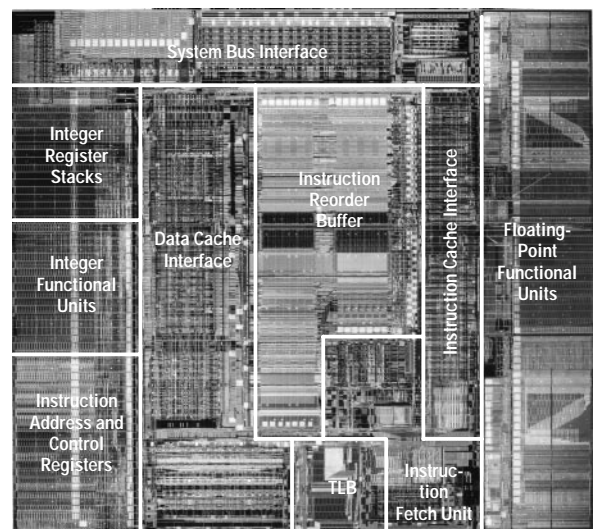


Figure 3

HP PA 8000 processor.



of heat in the PA 8000. The floating-point unit circuits normally dissipate power only when floating-point operations are being executed. Since this results in large power supply current transients, the floating-point unit can be configured to dissipate power even when inactive.

The PA 8000 includes two circuit cells whose purpose is to measure die temperature. Each of these cells contains a serpentine metal trace connected between two dedicated package pins. The resistance of these cells is temperature-dependent. These temperature monitor (TMON) cells must be calibrated for each die and package. The TMON cells were placed on the die in locations dictated by available pins. In retrospect, the placements were less than ideal.

During electrical characterization of the PA 8000, the effect of floating-point unit power-on die temperature was investigated. The TMON cells were instrumented and calibrated. Additional temperature measurements were made at the package/heat-sink interface and of the ambient air.

Electrical power to the die was supplied by a characterization fixture. This fixture is a system of power supplies, clock generators, a thermal chamber, and other instruments controlled by a computer workstation. It allows automated experiments and measurements to be performed on a circuit.²

Package-to-ambient-air thermal conductivity was consistent with previous experimental results as well as design objectives. However, TMON temperatures during peak floating-point unit power dissipation did not correlate with the known die input power and die-to-package thermal conductivity. The temperatures measured at the TMONs were lower than expected.

SPICE Simulation Using Electrical Analogs

It was suspected that the TMON temperatures were not representative of temperatures elsewhere on the die. The TMONs are both located 4 mm from one edge of the PA 8000 die. The floating-point unit, whose variable power was the object of the investigation, occupies a 4-mm-wide strip along the opposite side of the die. More than 10 mm of silicon lie between the floating-point unit and the TMONs, the only available means of measuring the floating-point unit's temperature. Thus, direct measurement of floating-point unit temperature was not possible. Another method of determining its temperature was needed.

Simulation of electrical circuits is used extensively in IC development. Drawing an analogy between electrical circuit theory and heat transfer made it possible to apply familiar IC design tools to model heat flow in the PA 8000. Heat flow can be modeled as electrical current flow, thermal resistivity as electrical resistance, thermal mass or capacity as electrical capacitance, and temperature as voltage. Using these analogies, a circuit model was simulated using HP Spice, a version of the public-domain SPICE program. The model was constructed using Piglet, a proprietary schematic capture and artwork design tool with a long history in IC design at HP.

The circuit model divides the PA 8000 die into 360 rectangular elements of 1 mm² each (**Figure 4**). Each element is connected to its neighbors through the thermal resistivity of the intervening volume of silicon. Heat flow out of the die is modeled by a resistive path from each element to a common node representing the heat sink.

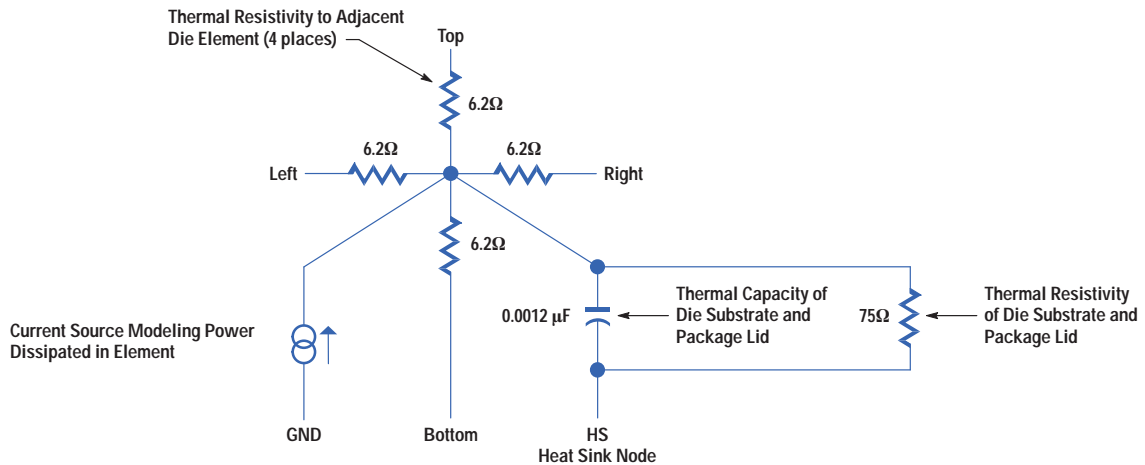
The thermal resistivity and thermal mass of the heat sink are modeled by a parallel resistor and capacitor to a node representing the ambient. A constant-voltage source sets the ambient temperature. A current source into each element of the die model models the power input into the circuits of that element. Power input was based on early simulations using Powermill software from Synopsys, Inc. This simulation summarized the power dissipated in each functional block at the top level of the design hierarchy. Power was assumed to be evenly distributed across the area of the block and a proportional current was sourced into the model elements corresponding to the physical location of the block.

Values for the components in the circuit model were derived by one of two methods. Components located between the package and ambient could be measured experimentally. Because the temperature measurements made using the TMON cells were suspect, components inside the package had to be calculated from the physical properties of silicon. The values used in the model are listed in **Table I**.

Steady-State Analysis

The first SPICE simulation confirmed that temperatures across the die varied greatly. As expected, the highest temperature was found in the floating-point unit. The corners on the side opposite the floating-point unit exhibited

Figure 4
SPICE cell.



the lowest temperatures. There was a surprising temperature difference of 21°C across the die.

Transient Analysis

A strength of SPICE is the analysis of transient phenomena. This is a very common application in IC design. A transient analysis of the PA 8000 thermal environment yielded insight into how the large temperature gradient across the chip was possible.

SPICE was modified to calculate an initial condition with the floating-point unit power turned completely off (see

Figure 5). A short time into the simulation, the floating-point unit power was turned on to its maximum value.

Over the first 0.5 second after the floating-point unit power is applied, the floating-point unit temperature rises until the shape of the temperature profile across the die is the same as was found in the steady-state analysis. Subsequently, the differences in temperature between any two points remain constant. The absolute temperatures, however, are not at their final values. With a time constant of 70 seconds, the temperature of the entire die continues to rise towards the final temperature. This slow rise corresponds to the heating of the package lid and heat sink.

Table I
Electrical Analogs of Thermal Elements

Component	Method	Value	Electrical Analog
Thermal Resistance of Silicon	Calculated	12.35 °C/W	12.35Ω modeled as one 6.18Ω resistor on each side of the element
Thermal Mass of Silicon	Calculated	0.0012 W/s·°C	0.0012 F
Thermal Resistance of the Die and Package per mm ²	Measured	64 °C/W	64Ω
Thermal Resistance of the Heat Sink	Measured	0.5 °C/W	0.5Ω
Thermal Mass of the Heat Sink	Measured	140 W/s·°C	140 F

Figure 5

Temperature gradient with floating-point unit power off (from SPICE simulation).

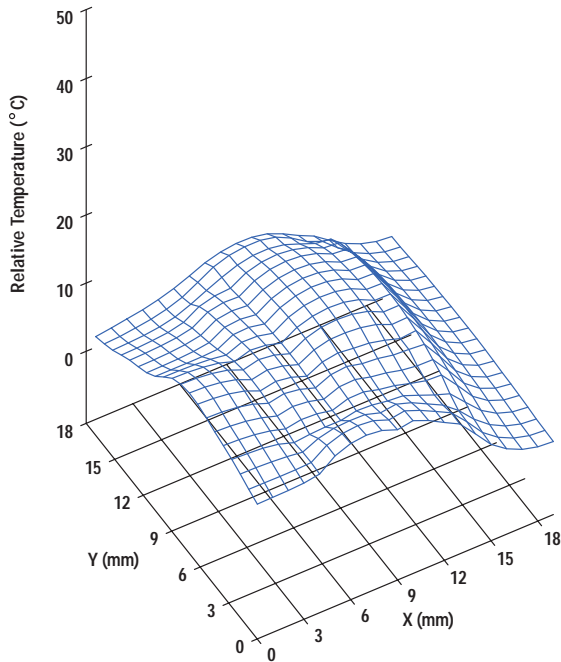
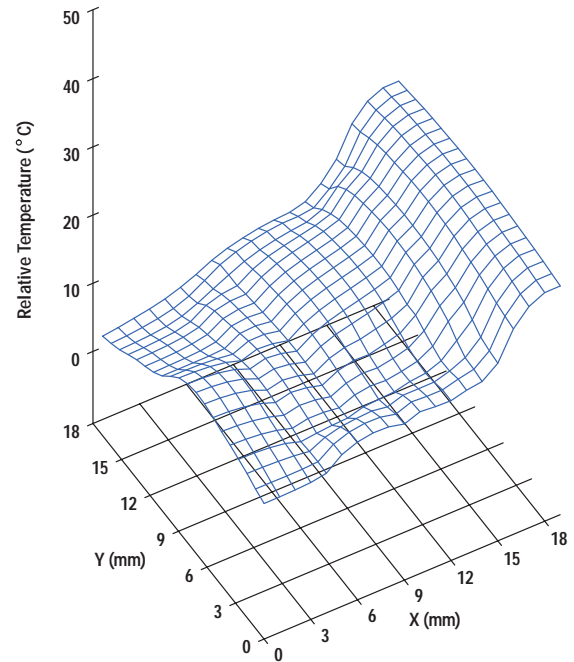


Figure 6

Temperature gradient profile established after floating-point unit power-on (from SPICE simulation).



The shape of the temperature profile as heat spreads out from the floating-point unit explains the magnitude of the gradient. **Figure 5** illustrates the initial state with the floating-point unit power off. Upon application of floating-point unit power, the heat from the floating-point unit can be seen to spread across the die. The advancing wave of heat is attenuated until it stops approximately halfway across the chip. **Figure 6** depicts this condition. The temperature profile of the remainder of the die is not affected directly by the floating-point unit. Instead, it rises only when the added power of the floating-point unit causes the temperature of the package and heat sink to rise. The final temperature gradient can be seen in **Figure 7**. This leads to the conclusion that the thermal resistance across the PA 8000 die is large enough relative to the thermal resistance to the package that locations more than 5 mm apart are effectively insulated from heat transfer.

Parameter Sweep

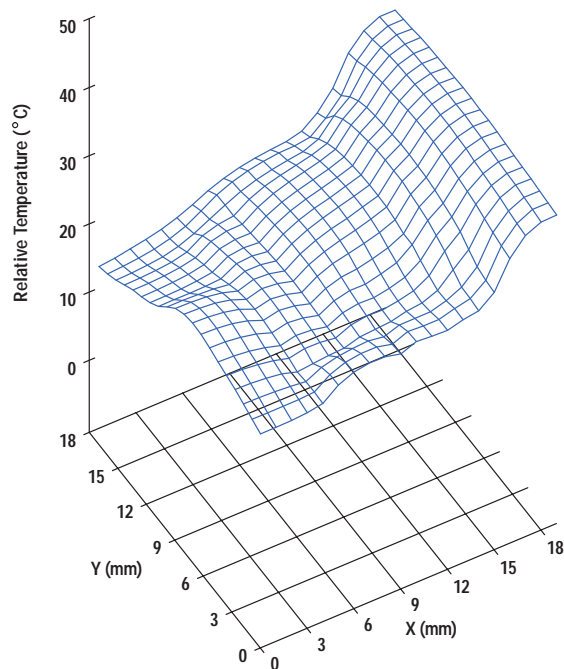
It seemed intuitive that the high temperature gradient across the die was a result of the low thermal resistance to the package. While the low resistance was important in

maintaining a low die temperature, it might also be causing a less-than-ideal temperature gradient by reducing heat flow across the die. How is the temperature gradient across the die affected by the package thermal resistance?

A typical application of SPICE is to execute multiple simulations while varying some characteristic of the circuit. This method was applied to the PA 8000 thermal model to determine the effect of changing the thermal resistance of the package. The resistors in the model that represented the package thermal resistance were swept across a range from one tenth of the typical value to 100 times the typical value. The gradient across the die increased with higher package resistance to a peak value. Beyond this peak, heat flow across the die became significant relative to the flow into the package and the gradient decreased with higher package resistance. The value at which the peak temperature difference occurred was 30 to 70 times the typical value, depending on which two points were compared. The die temperature reached unacceptable values long before this. Thus, for large values of package thermal resistance, the temperature gradient decreases with higher

Figure 7

Temperature gradient final state after heat sink heating (from SPICE simulation).



resistance, but at typical values, lower package thermal resistance is consistent with both lower die temperatures and a lower temperature gradient.

Finite-Difference Modeling

Finite-difference techniques involve solving a set of coupled, nonlinear, second-order partial differential equations. For solving thermal problems, the solution space is divided into a series of discrete cells and the differential equations for the conservation of momentum, energy, and mass are solved for each cell. For the purpose of this study, the Flotherm computational fluid dynamics software from Flomerics was used. Although the most popular application of Flotherm software is in the analysis of system-level air flow and convection cooling, it also contains features that allow accurate modeling of package-level details and conduction mechanisms. For the purpose of this study, a finite-element package such as Mechanica by Ransa, Inc. or ANSYS by ANSYS, Inc. would also have worked well.

Model Construction

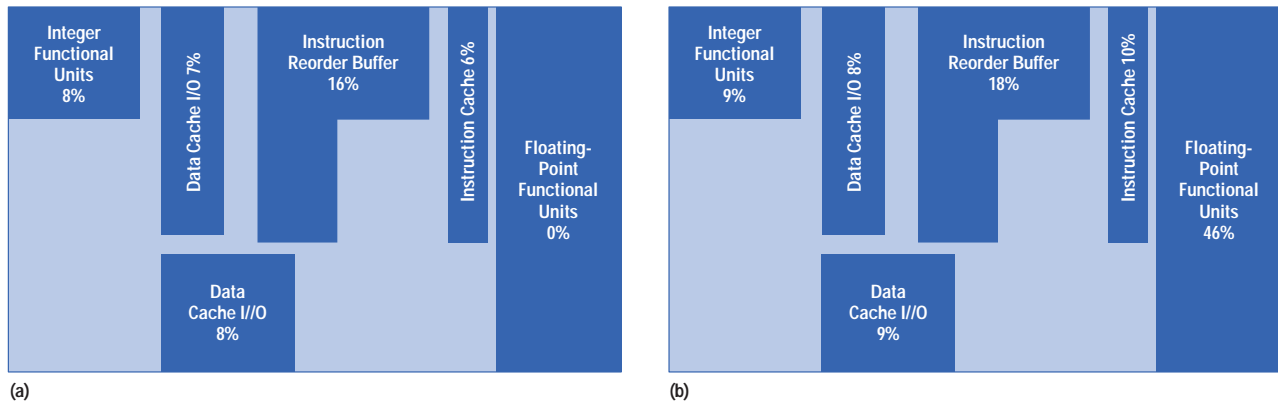
Power generation on the PA 8000 device is roughly symmetrical about a centerline. This allowed a reduction in complexity by creating a model representing only half of the device geometries. This is accomplished by placing the centerline of the half device against a surface that is considered to be a perfect insulator. The original PA 8000 thermal model used a single planar power source evenly distributed over the surface of the die. For our study, this power source was replaced with several distributed power sources. The top six highest-power-generating segments of the PA 8000 chip were isolated and mapped onto the half surface. The remaining elements contributed less than 20% of the power and were fairly evenly distributed. It was felt that their omission would not significantly alter the temperature gradient across the die. Simplified power distribution across the half surface of the chip in the high-power and low-power states can be seen in **Figure 8**. For the purpose of this study, the floating-point unit power in the low-power state was set to the minimum of zero. In the actual application the floating-point unit power is maintained in a 75% standby state to minimize sharp transient loads on the power supply. Maintaining a 75% power level also reduces microcycling stresses on the level 1 and level 2 interconnects.

All elements in Flotherm software must be created from cuboid blocks and plates. The major elements in the model consisted of the silicon die, ceramic substrate, printed circuit board, copper-tungsten package lid, and heat sink. All interface materials, solder bump junctions, and solder columns were modeled using internal plates. The reason for this change was twofold. First, Flotherm interprets internal plates as two-dimensional elements of fixed resistance. This simplifies the solution grid and eliminates many of the high-aspect-ratio cells. Secondly, internal plates allow quick changes to the model for sensitivity analysis. For example, rather than completely regridding the solution for a thicker epoxy, a simple resistivity change can be input and the new solution time shortened by using the previous results. While Flotherm internal plates only allow heat flow in a direction perpendicular to the plate, this was adequate for the elements that were chosen.

To decouple the effects of packaging and the final thermal path (i.e., heat sink or heat pipe), the heat sink was modeled by a large block of aluminum attached to the top of

Figure 8

Power distribution in the (a) low-power and (b) high-power states.



the package. The top surface of this block was defined as an isothermal (constant-temperature) surface. This is not unlike a heat pipe, in which the boiling point of the fluid creates an isothermal region. For the purposes of this study, the isothermal surface was set to 60°C. If a different ambient temperature is expected, the results from this study can be scaled up or down by an amount equal to the difference.

The ceramic package in this model is attached to a printed circuit board. For simplicity, the printed circuit board is modeled with a single lumped power plane to enhance heat spreading. A low air flow rate of 0.29 m/s is present, primarily to minimize the natural convection plumes. With the solution as modeled, more than 90% of the power flows to the isothermal surface. Convective heat transfer from the package and printed circuit board are almost negligible. Details of the model construction can be seen in **Figure 9**.

The Flotherm model was first run using the highest expected power densities, as outlined above. Modeling results indicate a 20°C temperature difference between the hottest and coolest portion of the chip. Two-dimensional thermal contour lines on the surface of the half die can be seen in **Figure 10**.

The Flotherm model was next run using the lowest expected power densities, as outlined above. Modeling results indicate a 10°C temperature difference between the hottest and coolest portions of the chip (see **Figure 10**).

Compared to the full-power situation, the highest temperature on the chip decreased by 16°C, while the lowest temperature on the die decreased by only 5°C.

Sensitivity Analysis

Since the original physical thermal tests had been performed with uniform power distribution, one of the first questions to be asked was how much the maximum junction temperature in the nonuniform CPU differed from that measured in the uniform test die. To quantify the disparity between the test die and the PA 8000, a PA 8000 model with a uniform power source was created. The total power was equal to the power of the gradient model in the full-on condition. Under these conditions, the maximum junction temperature is 86°C (compared to a range

Figure 9

Model construction for finite-difference simulation.

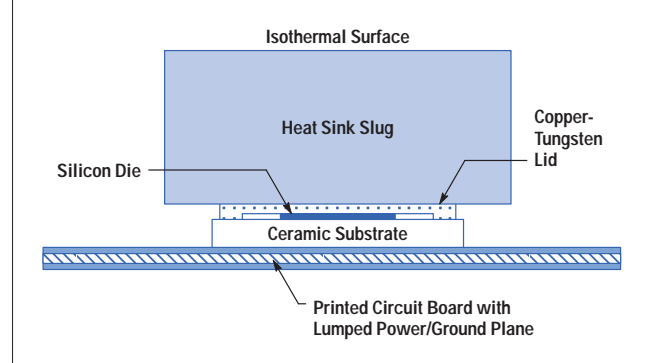
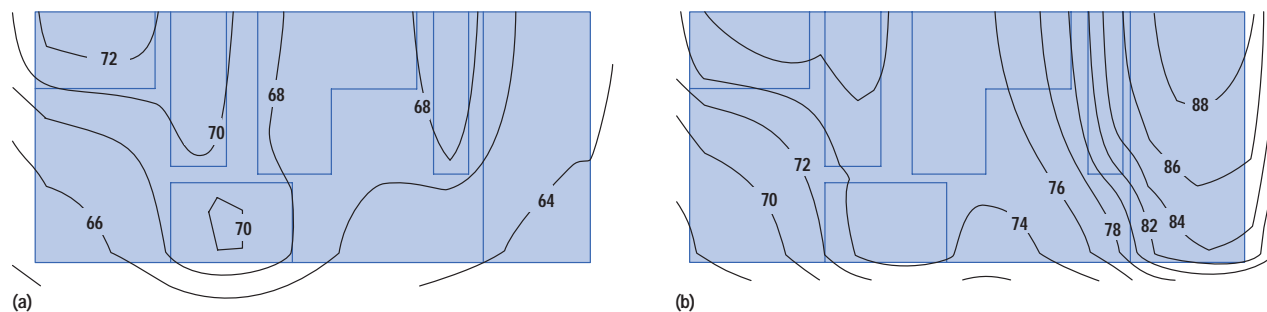


Figure 10

Temperature ($^{\circ}\text{C}$) gradients during (a) floating-point unit low-power mode and (b) full-power operation (from finite-difference simulation).



of 90 to 68 $^{\circ}\text{C}$ for the full-power condition with nonuniform distribution). This simulation showed that thermal testing with a uniform power source may have underpredicted the maximum junction temperature by 4 $^{\circ}\text{C}$. Depending on the amount of margin in the total thermal solution, this may or may not be an issue.

Once it was ascertained that a significant temperature gradient existed across the die surface, various sensitivity studies were undertaken to determine how the package construction might be used to minimize the temperature gradient. It was originally thought that the ceramic substrate might contribute enough heat spreading to equalize the temperatures across the die surface. A study was performed to determine the effect of either increasing or decreasing the thermal conductivity of the ceramic substrate. The nominal conductivity of alumina is 18 W/m $\cdot^{\circ}\text{C}$. The model was run with thermal conductivity an order of magnitude lower (1.8) and an order of magnitude higher (180). The lower thermal conductivity number is representative of organic laminate substrates. The upper thermal conductivity number is representative of an aluminum nitride substrate.

With reduced substrate thermal conductivity, the temperature gradient was increased by two degrees. With increased thermal conductivity, the thermal gradient was reduced by six degrees. The conclusion is that varying substrate thermal conductivity has a small to moderate effect on the temperature gradient. This is partially because the primary heat path for the PA 8000 package is through the back of the die.

As mentioned in the SPICE analysis discussion above, the excellent thermal path out of the package is a major contributing factor to the high temperature gradients. A thicker copper-tungsten lid on the package would encourage better heat spreading, but would reduce junction-to-ambient thermal performance. Similarly, increasing the thermal resistance of the interface to the heat sink would encourage more uniform temperatures, but would increase the overall junction temperature.

Conclusion

The accuracy of both models was verified by performing various measurements of the package case temperature and using the temperature monitor features built into the PA 8000 chip. Measurements agreed well with the temperature predictions.

From the results of both the SPICE and the Flotherm simulations, it is clear that there is a significant temperature variation across the surface of the PA 8000 die. This variation is caused by the disproportionately high power density of the floating-point unit. This temperature variation may result in underestimation of the actual junction temperature.

Table II compares the SPICE and finite-difference predictions. Both models were surprisingly similar in predicting the temperature gradient across the surface of the die. The predicted areas of maximum and minimum die temperatures were approximately the same with both techniques. Because the SPICE model used higher ambient

Table II

Comparison of SPICE and Finite-Difference Predictions

Simulation Tool	Gradient across Die with Floating-Point Unit On	Gradient across Die with Floating-Point Unit Off	Difference in Maximum Chip Temperature between On and Off States
Finite-Difference	20°C	10°C	16°C
SPICE	21°C	9°C	18°C

temperatures and assumed a higher case-to-ambient resistance, the predicted temperatures were higher overall. Adjustment of the model would bring the nominal temperatures closer to the finite-difference predictions.

Both models predicted similar transient responses as the die was switched to the high-power state. The initial rise time in the SPICE model was somewhat shorter because the model did not include capacitive elements to represent the thermal mass of the ceramic substrate and printed circuit board.

Each model was found to have application advantages. Because of the two-dimensional nature of the SPICE model, the solution time was on the order of seconds, rather than hours, as required for the finite-difference model. At the same time, the three-dimensional nature of the finite-difference model offered a better visual exploration into the effects of various elements in the package construction. For an experienced user, the amount of time to construct each model was about the same. The SPICE model allowed the electrical designer to work with a familiar set of tools to perform thermal analysis, but required a solid understanding of the electrical analogs of thermal elements. Finite-difference tools required a good understanding of how to apply the elements available in the software package to the physical problem. The tools used to preprocess and postprocess the SPICE data allowed automated input of data directly from the chip power models. The Flotherm user interface required manual input of all geometry and power data.

The results of these simulations were discussed with the PA 8000 packaging vendor. They identified localized mechanical strain on the solder bump joints as a result of thermal cycling as a potential reliability issue. Because of concerns over step loads on the system power supplies, the minimum floating-point unit power in existing products is limited to 75% of the full power. Using updated nonuniform power distribution data, the packaging vendor performed finite-element stress simulations and determined that the mechanical integrity of the columns was not compromised at the current power cycling levels.

The knowledge gained from these simulations is also being applied to the development of future Hewlett-Packard microprocessors.

Acknowledgments

Many thanks to Rich Blanco for sharing his PA 8000 Flotherm model, which was modified for this study. Thanks also to the folks at Flomerics for their excellent customer support.

References

1. A.P. Scott, et al, "Four-Way Superscalar PA-RISC Processors," *Hewlett-Packard Journal*, Vol. 48, no. 4, August 1997, pp. 8-15.
2. J.W. Bockhaus, et al, "Electrical Verification of the HP PA 8000 Processor," *ibid*, pp. 32-39.

A Low-Complexity, Fixed-Rate Compression Scheme for Color Images and Documents

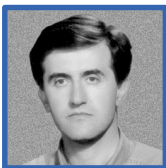
Nader Moayeri

Based on one-dimensional differential pulse code modulation, the coder is multiplication-free, codes each pixel as soon as it is available, and outputs a fixed number of bits for each pixel. Hence, there is no need for any buffering of the input image or coder output bitstream. The compression scheme is visually lossless and yields a modest compression ratio of 3 to 4. Because of its simplicity, it is useful when hardware is limited and coding delays cannot be tolerated.

There is often a need for some level of data compression in many imaging devices and products, such as scanners and facsimile machines. The reason can be the limited bandwidth of the data bus connecting the scanner to the host computer or the size of the memory available in the product. In addition, strict hardware limitations can preclude the use of compression schemes that are computationally intensive.

A desirable feature of a compression scheme is that it be a fixed-rate coder. Specifically, it is desirable for the compression scheme to produce the same number of bits in compressing an 8.5-by-11-inch sheet of paper *regardless of its content*. This guarantees that the memory of the device will never overflow and that the data is always transmitted to the host in a fixed time. Many popular compression schemes, such as the lossy JPEG and its newly adopted lossless JPEG-LS algorithms, are variable-rate coders. They have the advantage that they yield higher compression ratios than fixed-rate coders of comparable complexity. However, their variable-rate nature implies that in compressing a sheet of paper, there can be a large variation in the number of bits they generate, depending on the content of the sheet.

In this paper we present a very simple compression scheme that adequately addresses all the issues raised above. It compresses a given sheet of paper



Nader Moayeri

From 1994 to 1997, Nader Moayeri was a member of the technical staff of the Imaging Technology Department of HP Laboratories. Before joining HP, he was an assistant professor of electrical and computer engineering at Rutgers University, and he is currently manager of the Wireless Communications Technologies Group of the U.S. National Institute of Standards and Technology. A senior member of the IEEE, he received his PhD degree in electrical engineering from the University of Michigan at Ann Arbor in 1986. He is married, has a son, and enjoys volleyball, soccer, and opera.

(which may contain images, text, graphical art, etc.) pixel by pixel without any need for buffering the image data as it is being scanned. It produces a fixed number of bits for every pair of pixels, which can be stored or sent to the host over the data bus. An equally simple algorithm decompresses the received bits to produce a visually lossless rendition of the image data. The coder is lossy, but the loss level is so small that it is perceptually negligible. The coder is based on one-dimensional differential pulse code modulation (1D DPCM) using fully integer operation predictors and quantizers.

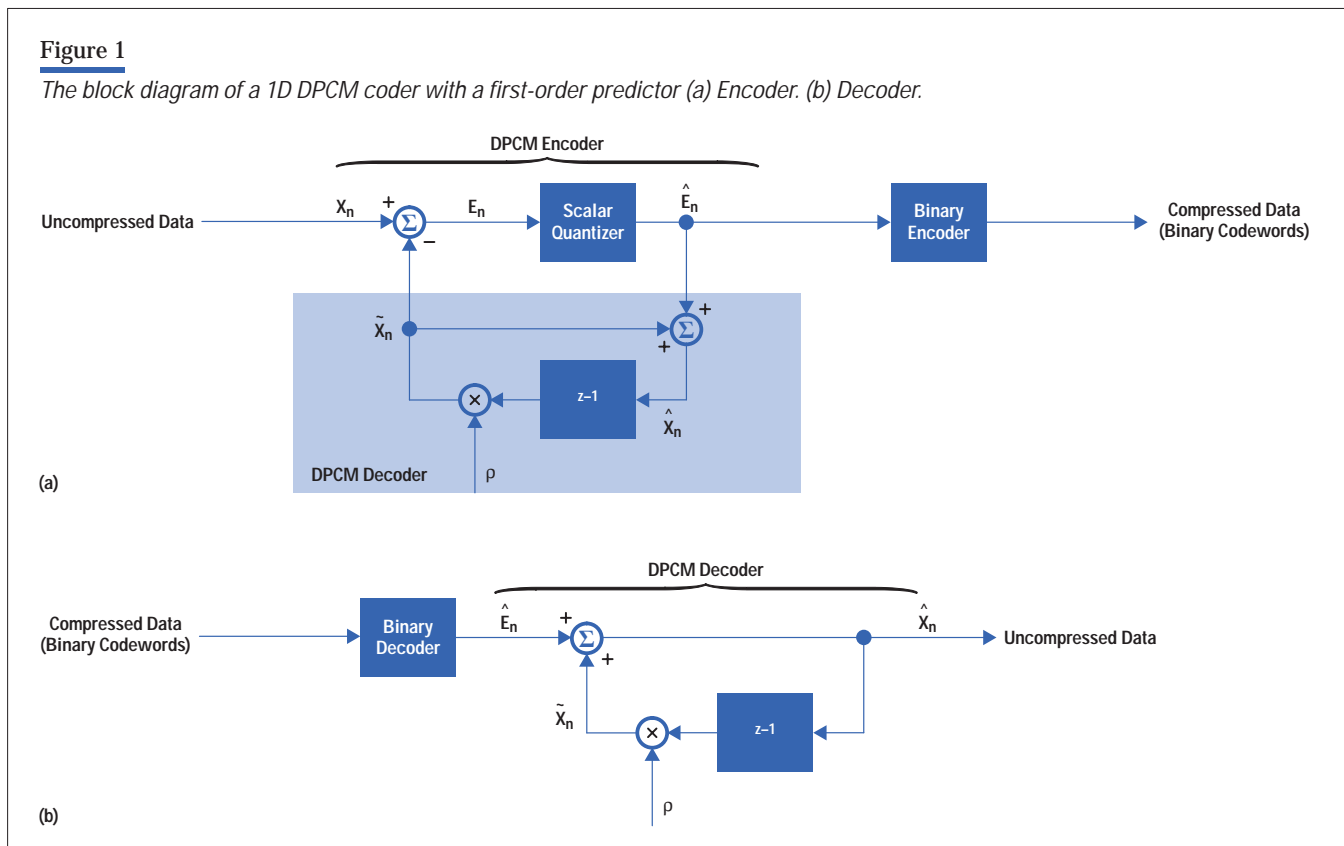
Proposed Compression Scheme

In this section we describe the various components and aspects of the compression scheme.

Differential pulse code modulation (DPCM) is a classical lossy data compression technique originally developed at Bell Laboratories for compressing the television signal.¹ It is the next level of sophistication in the compression technique hierarchy after pulse code modulation (PCM), which is the same as nonuniform scalar quantization.

The block diagram for a DPCM coder with a first-order predictor is shown in **Figure 1**. The task of the binary encoder is to map the output \hat{E}_n of the scalar quantizer to a binary codeword in a one-to-one fashion. For example, if the quantizer has 16 output levels, then the binary encoder will assign a four-bit distinct codeword to each possible quantizer output level. The binary decoder is simply the inverse of the binary encoder and maps a given binary codeword back to its respective quantizer output level. In other words, the input to the DPCM decoder is simply \hat{E}_n . Also note that there is a copy of the DPCM decoder at the DPCM encoder as shown by the shaded box in **Figure 1**.

The main idea of DPCM is to decorrelate the source data before coding it. Correlations exist in many data sources, such as images. DPCM removes these correlations by making a prediction \tilde{X}_n of the next source sample X_n on the basis of the past reconstructions and then coding the prediction error E_n instead of X_n itself. (A PCM system codes X_n directly.) In the system of **Figure 1**, the predictor uses the reconstruction for the most recent source



sample to predict the next one. That is, $\tilde{X}_n = \rho \hat{X}_{n-1}$, where ρ is a constant coefficient. A key property of a DPCM coder, even those employing higher-order predictors, is that:

$$X_n - \hat{X}_n = E_n - \hat{E}_n. \quad (1)$$

Hence, the mean squared error (MSE) distortion of the system is given by:

$$D = E[(X_n - \hat{X}_n)^2] = E[(E_n - \hat{E}_n)^2]. \quad (2)$$

If the predictor does a good job of predicting the source, then E_n will have a small variance (or effective dynamic range) compared to X_n . Consequently, an N-level optimal scalar quantizer for E_n will yield a smaller average distortion than one for X_n .

The predictor coefficient ρ depends on the samplewise correlation in the source; a larger value of ρ is used when the source is highly correlated. From a theoretical point of view, ρ should always be smaller than one so that the decoder filter remains stable. From a practical point of view, however, it is advantageous to use $\rho = 1$. We have opted for this choice in this work in the interest of making the coder as simple as possible. We have not experienced any instability problems in our experiments with the coder presented here.

We also note that two-dimensional DPCM, in which each pixel is predicted based on reconstructions for the previous pixel on the same image row and a couple of pixels on the previous row just above the present pixel, is a better choice for image compression. The predictor in 2D DPCM performs better than the one in 1D DPCM, making it possible to achieve better coded image quality at the same compression ratio. Using a 2D predictor requires buffering the reconstruction for the previous row of the image and the part of the present row that has already been encoded. In the interest of keeping the memory requirements of our coder at a minimum, we use the 1D predictor.

Quantization

In this work we assume that each output level of the scalar quantizer is assigned a binary codeword of length $\log_2 N$ by the binary encoder. (It is also possible to entropy-code the quantizer output. Since that would lead to a variable-rate coder, we are not interested in entropy-coded DPCM

in this work.) If the probability distribution of the prediction error E_n is known, then an optimal N-level non-uniform scalar quantizer for E_n can be designed. Such quantizers are called Lloyd-Max quantizers and there are algorithms for their design based on a training sequence of samples from the source.^{2,3} We used the Lloyd algorithm to design the quantizers needed in this work.

The problem of quantizer design in a DPCM system is complicated because the probability distribution of E_n depends on the quantizer and vice versa. This problem has been addressed by Arnstein,⁴ who describes an iterative design algorithm for the case of a first-order Gauss-Markov source. Arnstein's work, however, cannot be readily extended to the case of a DPCM system for images. Therefore, we use an *open-loop* approach to quantizer design. Since we are interested in a visually lossless compression scheme, it is reasonable to assume that $\hat{X}_n \approx X_n$. This approximation can be used to generate a training sequence of samples of the prediction error. Using a set of training images, we form the training sequence for E_n by simply subtracting each image pixel from the next one. Note that:

$$\begin{aligned} E_n &= X_n - \tilde{X}_n = X_n - \rho \hat{X}_{n-1} = X_n - \hat{X}_{n-1} \\ &\approx X_n - X_{n-1}. \end{aligned} \quad (3)$$

Finally, to make the coder as simple as possible, we force the quantizer output levels to be integers by properly modifying the centroid condition in the Lloyd algorithm. Specifically, we replace each centroid by the integer closest to it. It is easy to show that this is indeed the optimal way of designing integer valued scalar quantizers. The problem that has to be solved is the following:

Problem: Given a random variable X , find an integer C that minimizes $E[(X - C)^2]$.

Solution: For any integer C we can write:

$$\begin{aligned} E[(X - C)^2] &= E[(\{X - E[X]\} + \{E[X] - C\})^2] \\ &= E[(X - E[X])^2] + (E[X] - C)^2 \\ &\quad + 2(E[X] - C)E[X - E[X]] \\ &= \sigma_X^2 + (E[X] - C)^2, \end{aligned} \quad (4)$$

where σ_X^2 is the variance of X . The righthand side is minimized by choosing C to be the integer closest to $E[X]$. In addition, since the input to each quantizer is integer valued, we implement each quantizer with a small lookup table.

DPCM for Image Compression

It is straightforward to design a compression system for monochrome images based on the simple ideas presented above. Usually the system specifications (bus bandwidth and memory size) dictate a maximum rate r in bits per pixel (bpp) for the images that are to be compressed by this system. We assume that the rate is integer valued. (Otherwise, several quantizer outputs should be grouped together before being assigned a binary codeword. This would imply a need for some buffering.) The number of quantizer output levels is then given by $N = 2^r$, and the quantizer is designed according to the procedure outlined earlier.

In the case of color images the situation is more complicated. Of course, it is always possible to compress the image data in the RGB color space. However, it is advantageous to first transform the data into some other color space and then code it. Such transformations result in energy compaction in the data, which makes it possible to achieve better compression performance. For our work we picked the YUV color space. (The image data in some HP scanner pipelines is already in the YUV format.) It is well known that most of the image information is in the Y signal. Hence, in many color image compression algorithms, such as JPEG, the U and V signals are subsampled by a factor of 2 in both the horizontal and vertical directions before being coded. This subsampling compresses the U and V data by a factor of 4 even before any coding is done. In this work we have investigated both alternatives—subsampling the U and V signals and not doing any subsampling. In the latter case we have to use quantizers of lower rate for the U and V signals so that the overall coding rate (or compression ratio) is the same as in the subsampling case.

Yet another crucial factor in color image compression is the allocation of the available bit rate to the three color planes. There are optimal bit allocation algorithms that maximize the peak signal-to-noise ratio (SNR) and are commonly used in monochrome image coding.^{5,6} In the case of color images there are three color planes to worry about, and the peak SNR is even less relevant than in the

monochrome case. In this work we have taken an experimental approach for finding the best rate allocation. Our criterion is to get the best possible subjective image quality for a given overall rate. We are interested in rate values that yield visually lossless coded image quality. Since we have restricted quantizer rates to be integers, there are only a small number of bit allocations possible for any given overall rate. We found the best bit allocation experimentally and by trial and error.

Experimental Results

We designed a compression system for color images based on the ideas presented in the previous section and tested it on a test image we obtained from the HP Greeley Hard-copy Division. This is a compound image having photographic content, text, line art, and a rainbow ramp. In coding the U and V signals we investigated three alternatives:

- (i) Coding these signals at full resolution
- (ii) Subsampling them in the horizontal direction by simply discarding every other pixel and using pixel replication after the coding process
- (iii) Replacing each pair of successive samples by their average and then using a 3-tap area-based interpolation filter⁷ after coding.

Since the choice of the subsampling strategy or lack thereof affects the samplewise correlation of the U and V signals, hence influencing the probability distribution of the prediction error, we had to design two quantizers for each of the above alternatives. Since the Y signal is not subsampled, one quantizer is sufficient for that signal. Therefore, we designed a total of seven sets of quantizers, each set containing quantizers with 2, 4, 8, 16, 32, and 64 output levels. We used a training sequence consisting of the above test image and four other large images to design the needed quantizers. We transformed these images into the YUV color space, formed three *mother* training sequences for the Y , U , and V signals by taking differences of successive samples as described under “Quantization” above, and formed two more sequences from each of the mother sequences for U and V signals by properly subsampling them according to the alternatives (ii) and (iii) above. Using these seven training sequences we designed the needed seven sets of quantizers. Then we experimented with different systems, with or without U and V subsampling and with various choices of quantizers, to

code the test image. **Table I** shows the compression performance obtained with various choices for compressing the 24-bit test image at 12, 11, 10, 9, and 8 bits per pixel (bpp). The first two lines in the table correspond to transmitting certain numbers with infinite precision to the receiving end. This is not practical, of course, but it sets an upper bound on the best performance that can be expected from strategies (ii) and (iii).

For all the coders in the table designated with an asterisk (*), the quality of the coded image is visually lossless.

(Of course, the image quality is somewhat better at 12 bpp than at 8 bpp when the image is scaled.)

In reading the bit rates in **Table I**, note that with strategies (ii) and (iii) there are half as many U and V samples to be coded as there are Y samples. Also note that the test image is a particularly difficult image to work with. With other photographic images, such as the other four images in the training set, we were easily able to get a compression ratio of 4 to 6 and yet maintain visually lossless image quality.

Table I

Bit Rate and Peak SNR (Signal-to-Noise Ratio) of the Proposed Coder with Various Test Image Parameters

Overall Coder Rate (bpp)	Subsampling Strategy for U and V	Subcoder Rate (bpp)			Peak SNR (dB) YUV Color Space				Peak SNR (dB) RGB Color Space				
		Y	U	V	Y	U	V	Avg.	R	G	B	Avg.	
∞	(ii)	∞	∞	∞	∞	33.73	33.77	∞	31.21	35.28	29.22	31.90	
∞	(iii)	∞	∞	∞	∞	37.91	37.72	∞	35.07	38.75	33.39	35.73	
12	(i)	6	3	3	45.26	39.54	40.06	41.62	36.58	40.56	34.53	37.22	*
12	(i)	4	4	4	38.27	44.84	45.14	42.75	36.81	37.85	36.08	36.91	
12	(ii)	6	6	6	45.26	33.60	33.65	37.51	30.95	34.76	29.01	31.57	
12	(iii)	6	6	6	45.26	37.42	37.39	40.02	34.41	37.61	32.65	34.89	
11	(i)	5	3	3	42.48	39.54	40.06	40.69	36.07	39.46	34.21	36.58	*
11	(ii)	6	5	5	45.26	33.50	33.43	37.40	30.71	34.61	28.91	31.41	
11	(ii)	5	6	6	42.48	33.60	33.65	36.58	30.82	34.42	28.92	31.39	
11	(iii)	6	5	5	45.26	36.94	37.07	39.76	34.08	37.36	32.20	34.54	
11	(iii)	5	6	6	42.48	37.42	37.39	39.10	34.10	37.03	32.43	34.52	
10	(i)	4	3	3	38.27	39.54	40.06	39.29	34.72	36.97	33.24	34.98	
10	(ii)	6	4	4	45.26	33.24	33.41	37.30	30.73	34.57	28.67	31.32	
10	(ii)	5	5	5	42.48	33.50	33.43	36.47	30.58	34.29	28.82	31.23	
10	(ii)	4	6	6	38.27	33.60	33.65	35.18	30.37	33.44	28.63	30.82	
10	(iii)	6	4	4	45.26	37.02	36.84	39.71	33.93	37.29	32.33	34.52	*
10	(iii)	5	5	5	42.48	36.94	37.07	38.83	33.79	36.82	31.99	34.20	
10	(iii)	4	6	6	38.27	37.42	37.39	37.30	33.21	35.41	31.79	33.47	
9	(ii)	6	3	3	45.26	32.37	32.35	36.66	29.80	33.88	27.88	30.52	
9	(ii)	5	4	4	42.48	33.24	33.41	36.37	30.61	34.25	28.59	31.15	
9	(ii)	4	5	5	38.27	33.50	33.43	35.07	30.15	33.35	28.54	30.68	
9	(iii)	6	3	3	45.26	35.16	34.92	38.45	32.11	36.07	30.64	32.94	
9	(iii)	5	4	4	42.48	37.02	36.84	38.78	33.65	36.74	32.14	34.18	*
9	(iii)	4	5	5	38.27	36.94	37.07	37.43	32.93	35.29	31.40	33.21	
8	(ii)	5	3	3	42.48	32.37	32.35	35.73	29.70	33.61	27.81	30.37	
8	(ii)	4	4	4	38.27	33.24	33.41	34.97	30.19	33.31	28.31	30.60	
8	(iii)	5	3	3	42.48	35.16	34.92	37.52	31.91	35.66	30.50	32.69	
8	(iii)	4	4	4	38.27	37.02	36.84	37.38	32.83	35.21	31.56	33.20	*

The asterisks in the rightmost column designate the coder that gave the best image quality for each rate.

Table II

Number of Arithmetic Operations per Image Pixel Needed by the Encoder and Decoder With Each of the Three U and V Subsampling Strategies

Strategy	Encoder			Decoder		
	Adds, Subtracts	Table Lookups	Shifts	Adds, Subtracts	Table Lookups	Shifts
(i)	6	3	0	3	3	0
(ii)	4	2	0	2	2	0
(iii)	5	2	1	5	2	3

It can be seen from the table that the naive subsampling strategy (ii) of the U and V signals is always inferior to the more careful strategy (iii). This is also true in terms of coded image quality. However, implementing strategy (iii) adds a little bit to the computational requirements of the coder. We also note that strategy (iii) is in some cases inferior to strategy (i), namely not subsampling at all. In fact, we made the observation that at high bit rates, even if no quantization at all is done after subsampling with either strategy (ii) or (iii), the quality of the reconstructed image can be inferior to an image obtained with coding and strategy (i). This can be seen by comparing the first two lines of **Table I** with the third line, for example. The reason is the inherent loss of the subsampling and reconstruction operations. Finally, strategy (iii) seems to be better than strategy (i) at bit rates 10, 9, and 8 bpp. This might be because in our experiments we restricted our attention to cases where the U and V signals get equal shares of the available bit rate and the quantizer rates are all integers.

Finally, we like to stress that all three coding strategies proposed in this paper are very simple. **Table II** shows the exact number of encoding and decoding operations required by each strategy. All three methods have negligible computational complexity and hence would have a very simple implementation and would run very fast. The amount of memory needed at the encoder is simply that for the three lookup tables implementing the three quantizers. Each method would need three tables of 511 bytes each with a conservative allocation of one byte per table entry. Each decoding method would also need three lookup tables to map the quantizer indices to the quantizer output levels. These tables would be on the host and not on the scanner. They are even smaller than those needed

at the encoder, because each would have as many entries as the number of quantizer output levels.

Conclusion

In this paper we have presented a compression scheme for color images based on one-dimensional differential pulse code modulation (DPCM). The image data is assumed to be in YUV or YCrCb color space, and the chrominance information may or may not have been subsampled. There is one DPCM coder for each of the three color components, and an optimal nonuniform scalar quantizer (a Lloyd-Max quantizer) for each coder. The three quantizers have been designed in such a way that all their output levels are integer valued. Therefore, the coder uses integer operations exclusively. The quantizers are implemented by lookup tables and the predictors are simple. Consequently, the coder is multiplication-free. The coder operates in real time in the sense that each pixel is coded as soon as it is available (this would be useful in a scanner). In addition, the coder outputs a fixed number of bits for each pixel. Hence, there is no need for any buffering of the coder output bitstream before transmission. All these features make the coder very simple and easily implementable with minimal hardware. Even the memory requirement of the coder is negligible because the lookup tables are small and there is no need to buffer the image pixels.

This compression scheme has been tested on several images of different types and contents. The coded image quality is very good and visually lossless in all cases. The coder yields a modest compression ratio of 3 to 4. The coder's compression performance is not as high as the industry-standard JPEG algorithm, but then JPEG is a variable-rate coder that is orders of magnitude more complex. Because of its extreme simplicity, the coder is

suitable for applications in which there is limited hardware capability and coding delays cannot be tolerated. It may be the simplest coder that achieves modest compression without compromising the image quality.

References

1. C.C. Cutler, *Differential quantization for television signals*, U.S. Patent 2605361, July 1952.
2. S.P. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, Vol. IT-28, Part I, March 1982, pp. 129-137.
3. J. Max, "Quantizing for minimum distortion," *IEEE Transactions on Information Theory*, Vol. IT-6, March 1960, pp. 7-12.
4. D.S. Arnstein, "Quantization error in predictive coders," *IEEE Transactions on Communications*, Vol. COM-23, April 1975, pp. 423-429.
5. Y. Shoham and A. Gersho, "Efficient bit allocation for an arbitrary set of quantizers," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-36, September 1988, pp. 1445-1453.
6. E.A. Riskin, "Optimal bit allocation via the generalized BFOS algorithm," *IEEE Transactions on Information Theory*, Vol. IT-37, March 1991, pp. 400-402.
7. P.W. Wong and C. Herley, "Area-Based Interpolation for Scaling of Images from a CCD," *Proceedings of the IEEE International Conference on Image Processing*, Santa Barbara, California, November 1997, pp. 1905-908.