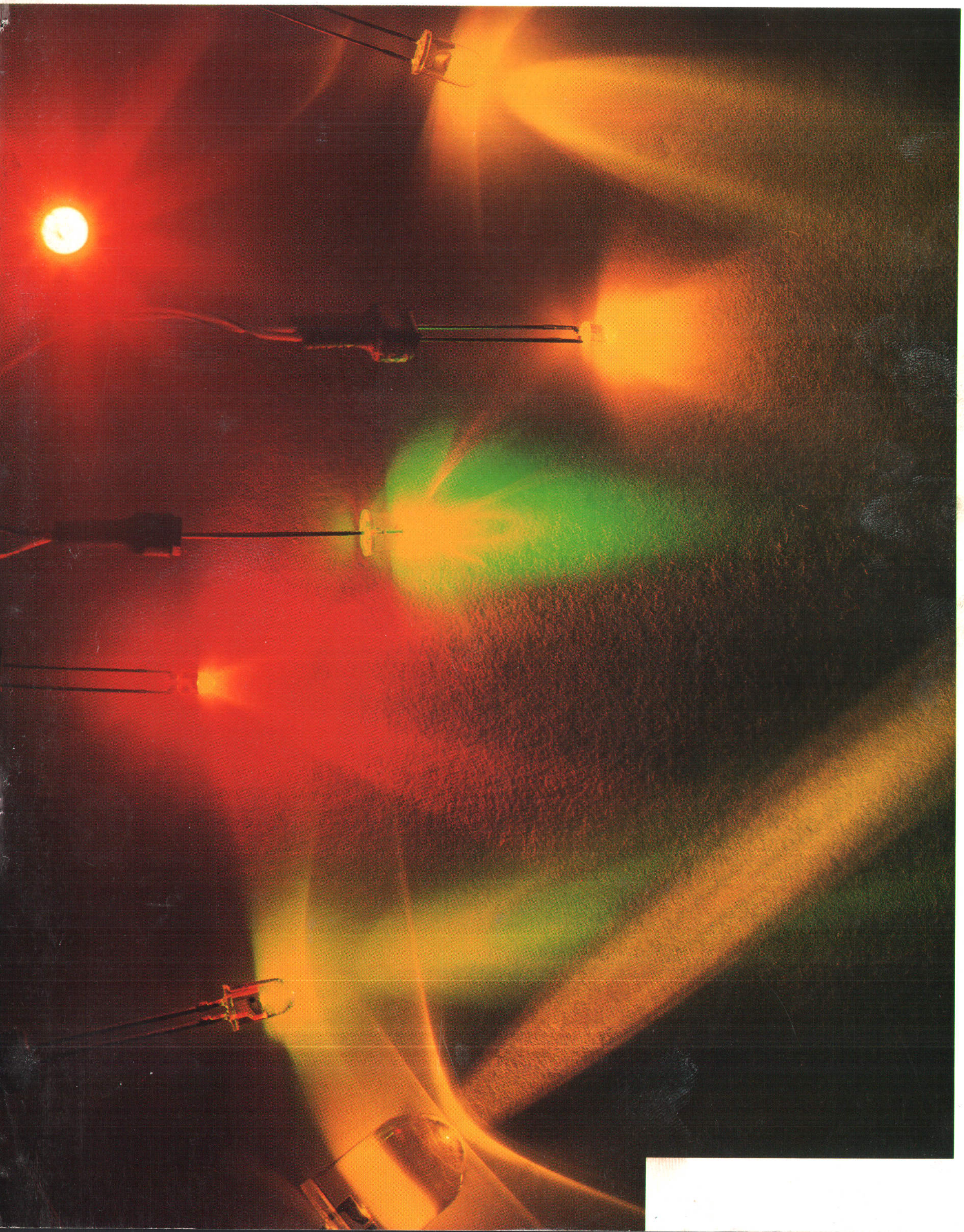
HEWLETT-PACKARD
JOURNAL

August 1993



HEWLETT
PACKARD

HEWLETT-PACKARD
JOURNAL

August 1993 Volume 44 • Number 4

Articles

Editor, Richard P. Dolan • **Associate Editor**, Charles L. Leath • **Publication Production Manager**, Susan E. Wright • **Illustration**, Renée D. Pighini
Typography/Layout, Cindy Rubin • **Test and Measurement Organization Liaison,** Sydney C. Avey

The Hewlett-Packard Journal is printed on recycled paper.

Research Report

Departments

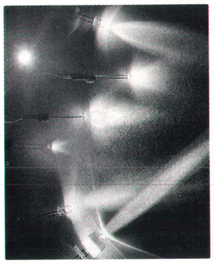
The Hewlett-Packard Journal is published bimonthly by the Hewlett-Packard Company to recognize technical contributions made by Hewlett-Packard (HP) personnel. While the information found in this publication is believed to be accurate, the Hewlett-Packard Company disclaims all warranties of merchantability and fitness for a particular purpose and all obligations and liabilities for damages, including but not limited to indirect, special, or consequential damages, attorney's and expert's fees, and court costs, arising out of or in connection with this publication.

Subscriptions: The Hewlett-Packard Journal is distributed free of charge to HP research, design and manufacturing engineering personnel, as well as to qualified non-HP individuals, libraries, and educational institutions. Please address subscription or change of address requests on printed letterhead (or include a business card) to the HP headquarters office in your country or to the HP address on the back cover. When submitting a change of address, please include your zip or postal code and a copy of your old label. Free subscriptions may not be available in all countries.

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

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

In this Issue



Light-emitting diodes bright enough for outdoor applications in bright sunlight—automobile tail lights, for example—have been a long-sought goal of LED research. HP's latest LEDs, described in the article on page 6, should meet the needs of many outdoor applications. Made from aluminum indium gallium phosphide (AlInGaP), they surpass the brightness of any previously available visible LEDs and come in a range of colors from red-orange to green. Technically, they are double-heterostructure LEDs on an absorbing substrate and are grown by means of a technique called organometallic vapor phase epitaxy, which has been used for producing semiconductor laser diodes but not for the mass production of LEDs. In addition to the technical details of the new LEDs, the article provides a history of LED material and structure development.

Let's say you have a computing network in which users need to share resources. A user needs to move a compute job to a remote machine to free local compute cycles or access remote applications. You would like your computers to be equally loaded, and you would like to make remote access as automated as possible. Also, you want disabled machines to be automatically avoided. HP Task Broker (see page 15) is a software tool that distributes applications among servers efficiently and transparently. When a user requests an application or service, HP Task Broker sends a message to all servers, requesting bids for providing the service requested. Each server returns its "affinity value," or bid, for the service, and the server with the highest value is selected. Tasks are distributed at the application level rather than the procedure level, so no modifications are required to any application. Besides load balancing and increased availability, the benefits of HP Task Broker include multiple-vendor interoperability, easier network upgradability, and reduced costs.

Real-time systems, unlike timesharing and batch systems, must respond rapidly to real-world events and therefore require special algorithms to manage system resources. The HP-RT operating system is the result of porting an existing operating system to the HP 9000 Model 742rt board-level real-time computer. The HP-RT kernel implementation, including the concepts of threads, counting semaphores, and priority-inheritance semaphores, is described in the article on page 23. The article on page 31 discusses the handling of interrupts in HP-RT and tells how the HP PA-RISC architecture of the Model 742rt affected the operating system design.

The HP Tsutsuji logic synthesis system (page 38) takes logic designs expressed as block diagrams and transforms them into netlist files that gate-array manufacturers can use to produce application-specific integrated circuits (ASICs). In many applications, the system reduces the time required to design an ASIC by a factor of ten or more. Tsutsuji was developed jointly by HP Laboratories and the Yokogawa-Hewlett-Packard Design Systems Laboratory in Kurume, Japan. Because the World Azalea Congress was being held in Kurume when the project began, Tsutsuji—the Japanese word for azalea—was chosen as the name of the system. Currently, Tsutsuji is only being marketed in Japan. The article covers its architecture, its operation, and several applications.

A desktop scanner digitizes photographs, documents, drawings, and three-dimensional objects and sends the information to a computer, usually for electronic publishing applications. The HP ScanJet IIc scanner (page 52) is a 400-dot-per-inch flatbed scanner that has black and white, color, and optical character recognition capabilities. Using an HP-developed color separator design, it provides fast, single-scan, 24-bit color image scanning. The article describes the color separator design and discusses the challenge of trying to duplicate human vision so that colors look the same in all media.

Issues in the design of a workstation computer for industrial automation applications include serviceability, input/output capabilities, support, reliability, graphics, front-to-back reversibility, mounting options, form factor, airflow management, acoustics, and modularity. How these issues are addressed by the mechanical design of the HP 9000 Models 745i and 747i entry-level industrial workstations is the subject of the article on page 62.

Franco Canestri is an application and technical support specialist for HP cardiology products in Europe. He also continues the medical laser research he began as an assistant fellow at the National Cancer Institute of Milan, focusing on orthopedic surgery applications. In the paper on page 68, he describes recent work on an algorithm for real-time surgical laser beam control using HP 9000 computers.

The final three papers in this issue are from the 1992 HP Software Engineering Productivity Conference. ▸ On page 73 is a description of a defect management system created for software and firmware development at two HP divisions. The system uses a commercial relational database management system. ▸ The C++ language and object-oriented programming offer potential productivity gains, including code reuse, but there can be pitfalls. The article on page 85 discusses these as well as some new features of the language. ▸ In developing real-time software, it may be difficult to go from a structured analysis model to a structured design. To help make this transition for HP medical ultrasound software, one HP division used a high-level design methodology called ADARTS. It's discussed on page 90.

R.P. Dolan
Editor

Cover

This photograph illustrates many of the features of the new HP AlInGaP light-emitting diodes, including their range of colors, their package types, their narrow-beam light output, and their brightness when viewed head-on. Although we took the picture in the dark, the main applications are daylight-viewable displays and automotive lighting.

What's Ahead

Featured in the October issue will be the design of the HP 54720 sampling digitizing oscilloscope family, which offers sample rates up to 8 gigasamples per second and bandwidths from 500 megahertz to 2 gigahertz, the HP E1430A 10-megahertz analog-to-digital converter module, which has 110-dB linearity and built-in memory and filter systems, and the HP 4396A 1.8-gigahertz vector network and spectrum analyzer, a combination analyzer with laboratory-quality performance in all functions.

High-Efficiency Aluminum Indium Gallium Phosphide Light-Emitting Diodes

These devices span the color range from red-orange to green and have the highest luminous performance of any visible LED to date. They are produced by organometallic vapor phase epitaxy.

by Robert M. Fletcher, Chihping Kuo, Timothy D. Osentowski, Jiann Gwo Yu, and Virginia M. Robbins

Since light-emitting diodes (LEDs) were first introduced commercially in the late 1960s, they have become a common component in virtually every type of consumer and industrial electronic product. LEDs are used in digital and alphanumeric displays, bar-graph displays, and simple on/off status indicators. Because of their limited brightness, LEDs have tended to "wash out" under sunlight conditions and have not generally been used for outdoor applications. (Recall the quick demise of digital watches with LED displays in the early 1970s.) However, the introduction of bright red-light-emitting AlGaAs LEDs in the mid and late 1980s partially eliminated this drawback. Now, another family of LEDs, made from AlInGaP, has been introduced. These LEDs surpass the brightness of any previous visible LEDs and span the color range from red-orange to green. With this breakthrough in brightness in a broad range of colors, we should see a wide variety of new applications for LEDs within the next decade.

History

Although the various LED display and lamp packages are familiar to many (for example, the usual LED single-lamp package with its hemispherical plastic dome, or the seven-segment digital display package), the diversity of materials used in the chips that go into these packages is not as familiar. Fig. 1 summarizes the various semiconductor materials used in LEDs and charts the evolution of the technology over the past 25 years. In the figure, luminous performance, measured in lumens* of visible light output per watt of electrical power input, is plotted over time starting from 1968 and projected into the mid-1990s.

The first commercial LEDs produced in the late 1960s were simple p-n homojunction devices made by diffusing Zn into GaAsP epitaxial material grown by vapor phase epitaxy on a GaAs substrate.[1] GaAsP is a direct-bandgap semiconductor for compositions where the phosphorus-to-arsenic ratio in the crystal lattice is 0.0 to 0.4. Above 0.4, the bandgap becomes indirect.** The composition of 60% As and 40% P produces red near-bandgap light at about 650 nm. Quantum efficiency in a simple homojunction device such as this is low, but these so-called "standard red" LEDs were and still are inexpensive and relatively easy to produce. The red numeric displays in the first pocket calculators were made of standard red LEDs.

At around the same time, GaP epitaxial layers doped with zinc and oxygen and grown on GaP substrates by liquid phase epitaxy were introduced. The GaP substrate, unlike GaAs, is transparent to the emitted light, allowing these devices to be more efficient than the GaAsP standard red diodes. However, the emission wavelength at 700 nm is near the edge of the visible spectrum, which limits their usefulness.

A major breakthrough in LED performance came in the early 1970s with the addition of nitrogen to GaAsP and GaP epitaxial materials.[2,3,4] Nitrogen in these semiconductors is not a charge dopant; rather it forms an isoelectronic impurity level in the bandgap which behaves as an efficient radiative recombination center for electrons and holes. In this way, even indirect-bandgap GaP and indirect compositions of

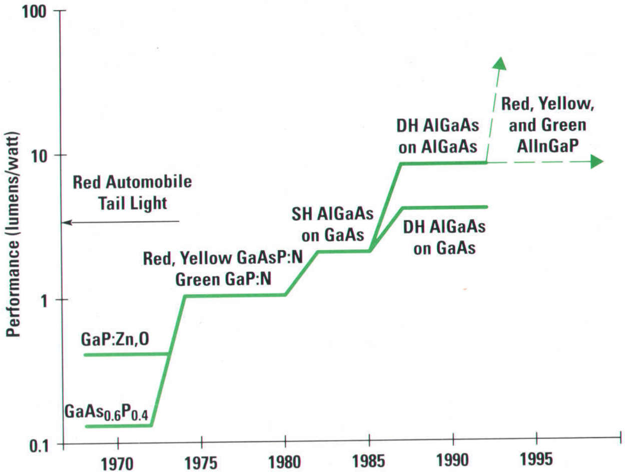


Fig. 1. Time evolution of light-emitting diode technology.

* A lumen is a measure of visible light flux that takes into account the wavelength sensitivity of the human eye. An LED's output in lumens is obtained by multiplying the radiant flux output of the LED in watts by the eye's sensitivity as defined by the Commission Internationale de l'Eclairage (CIE).

** In a direct-bandgap semiconductor, the recombination of electrons and holes has a high probability of occurring through a band-to-band radiative process in which a photon is emitted. In an indirect-bandgap semiconductor, radiative band-to-band recombination requires the interaction of a lattice vibration (a phonon) with the electron and hole. For this interaction the probability is low, and consequently nonradiative recombination processes dominate.

GaAsP can be made to emit sub-bandgap light efficiently. By the mid-1970s, orange and yellow LEDs made from various alloys of GaAsP and green LEDs made from GaP appeared on the market.

The next breakthrough occurred almost a decade later with the introduction of AlGaAs red-light-emitting LEDs, grown by liquid phase epitaxy. These provided two to ten times the light output performance of red GaAsP.[5,6] The reason for the range of performance of AlGaAs is that it can be produced in various structural forms: a single heterostructure on an absorbing substrate (SH AS AlGaAs), a double heterostructure on an absorbing substrate (DH AS AlGaAs), and a double heterostructure on a transparent substrate (DH TS AlGaAs). (See page 8 for an explanation of heterostructures.) This was an important milestone in LED technology because for the first time LEDs could begin to compete with incandescent lamps in outdoor applications such as automobile tail lights, moving message panels, and other applications requiring high flux output. Included in Fig. 1 is the flux required for a red automobile tail light, which is well within the performance range of AlGaAs LEDs. Unfortunately, AlGaAs LEDs can efficiently emit only red (or infrared) light, which makes them unsuitable for many applications.

The latest technology advance, and the subject of this paper, is the development of AlInGaP double-heterostructure LEDs. These devices span the color range from red-orange to green at light output performance levels comparable to or exceeding those of AS and TS AlGaAs.[7,8] The AlInGaP materials are grown by a technique called organometallic vapor phase epitaxy. This growth technology has been used for the production of optoelectronic semiconductors, especially laser diodes, for a number of years, but it has not been previously used for the mass production of LEDs.

Hewlett-Packard's AlInGaP devices currently being introduced to the market have the highest luminous performance of any visible LED to date. As the technology matures through the 1990s, performance levels are expected to increase further and reach into the tens-of-lumens-per-watt range.

Properties of AlInGaP

The bandgap properties of several compound semiconductors used in LED technology are shown in Fig. 2. Illustrated is the bandgap energy as a function of crystal lattice constant. In a diagram such as this, binary compound semiconductors, such as GaP and InP, are plotted as single points, each with a unique bandgap and lattice constant. Ternary compounds, such as AlGaAs, are represented by a line drawn between the two constituent binary compounds, in this case AlAs and GaAs. Finally, quaternary compounds, such as AlInGaP, are represented by an enclosed region with the constituent binary compounds at the vertices. The complex nature of the crystal band structure and the transition from a direct-bandgap semiconductor to an indirect-bandgap semiconductor are what give the enclosed region its characteristic shape. Properties such as this are usually obtained from both experiment and theory.

This type of diagram is useful for designing LED materials for at least two reasons. First, it shows what compositions of AlInGaP are direct-bandgap and therefore readily useful for making efficient LEDs. Second, for high-quality epitaxial growth it is necessary for the epitaxial layers to have the same lattice constant as the substrate on which they are grown. This diagram shows what compositions of AlInGaP will provide this lattice matching condition for a given substrate. For visible LEDs, the two common substrates used are GaAs and GaP. Clearly GaP is not immediately useful here because it is at the indirect-bandgap end of the AlInGaP composition region. This leaves GaAs as the only suitable substrate. A vertical line drawn from the x axis through the GaAs point intersects the AlInGaP region and indicates the compositions that lattice match to a GaAs substrate. The composition that gives this lattice match condition is written as:

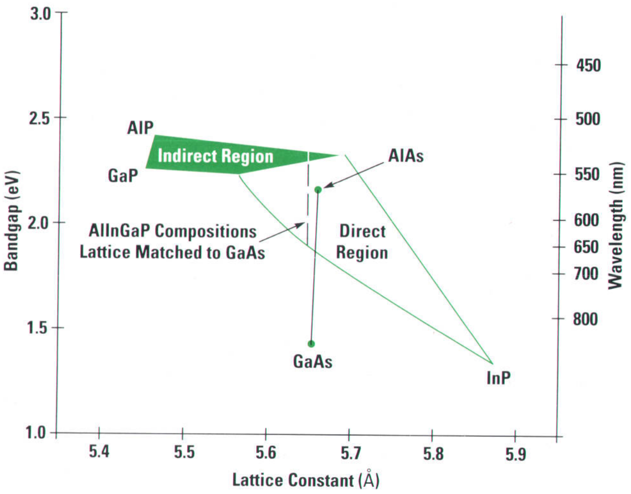


Fig. 2. AlInGaP alloy system.

$$(Al_xGa_{1-x})_{0.5}In_{0.5}P.$$

This notation, which is typical for describing compound semiconductors, indicates the proportions of the constituent atoms within the crystal lattice. In this case, half the group III atoms are indium and the other half are some mixture of aluminum and gallium. By coincidence, aluminum and gallium have approximately the same atomic size within the lattice. As long as the amount of indium remains fixed at 0.5, the aluminum-gallium mix can vary continuously from all aluminum to all gallium, and the lattice constant will not change appreciably. What will change is the bandgap of the material. If the aluminum is kept below $x \approx 0.7$, the bandgap is direct; above values of $x \approx 0.7$, the bandgap becomes indirect. This case is illustrated in Fig. 2 where the line of lattice match crosses from the direct region into the indirect region.

The bandgap diagram indicates the potential of a material for making LEDs, that is, whether a material has a direct bandgap and whether the bandgap energy is within the proper range for producing visible photons. The actual performance of a device depends on a number of additional factors. First, the growth of high-quality epitaxial material must be possible. Ideally, the growth should take place on a commonly available, inexpensive substrate and should be lattice matched to that substrate. Second, it must be possible to form a p-n junction in the material. Third, to obtain the highest quantum efficiency, it should be possible to grow a double heterostructure. In the case of AlInGaP, all three of these conditions are satisfied.

The Structure of LEDs: Homojunctions and Heterojunctions

Light-emitting diodes come in a variety of types, differing in materials and in epitaxial structure. GaAsP and GaP are used for the majority of red, orange, yellow, and green LEDs currently in use. All these LEDs are homojunction p-n diodes with either diffused junctions or junctions grown-in during the epitaxial process. Fig. 1 shows a cross section of a typical GaAsP homojunction chip. In other material systems, such as AlGaAs and AlInGaP, it is possible to grow layers of different compositions (heterostructures) and therefore different bandgaps while keeping the lattice constant the same in all the layers. This capability means that more complex and efficient LED structures can be grown with these materials.

Fig. 2 illustrates an AlGaAs single-heterostructure (SH) chip. The epitaxial part of the device consists of an n-type active layer where the light is generated, and a single p-type window layer on top. The composition of the window layer is chosen to have a significantly larger bandgap than the active layer, and as such it is transparent to the light generated in the active layer (hence the name window layer). The single heterojunction (excluding the one with the substrate), which in this case is also the p-n junction, is what defines this as a single-heterostructure device. The efficiency increase is a result of the transparency of the window layer and increased injection efficiency at the p-n heterojunction.

A modification of the single heterostructure is the double heterostructure (DH) shown in Fig. 3, again using AlGaAs as an example. In this case an additional layer is grown between the active layer and the substrate. In a double heterostructure, the two high-bandgap layers surrounding the active layer are referred to as confining layers. Together they act to confine electrons and holes within the active layer where they recombine radiatively. The lower confining layer efficiently injects electrons into the active layer and helps channel some of the light out of the chip, while the upper confining layer acts as a window for the generated light.

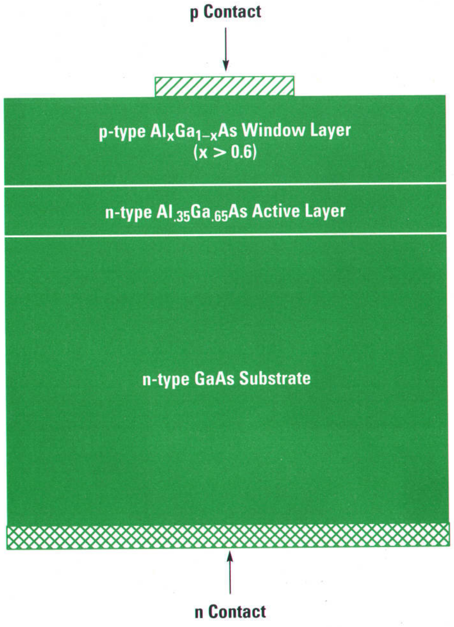

p Contact
p-type Al$_x$Ga$_{1-x}$As Window Layer (x > 0.6)
n-type Al$_{.35}$Ga$_{.65}$As Active Layer
n-type GaAs Substrate
n Contact

Fig. 2. AlGaAs single-heterostructure LED on an absorbing GaAs substrate.

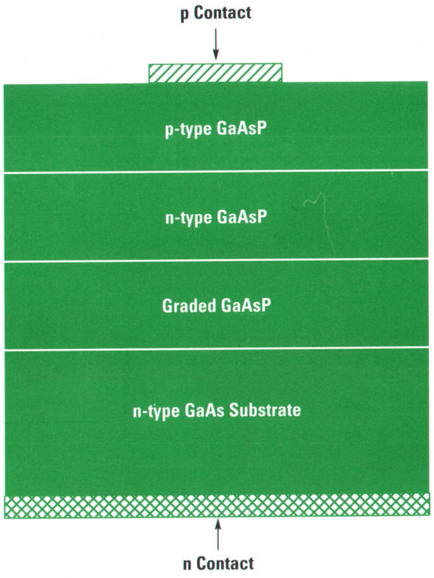


Fig. 1. GaAsP standard red homojunction LED.

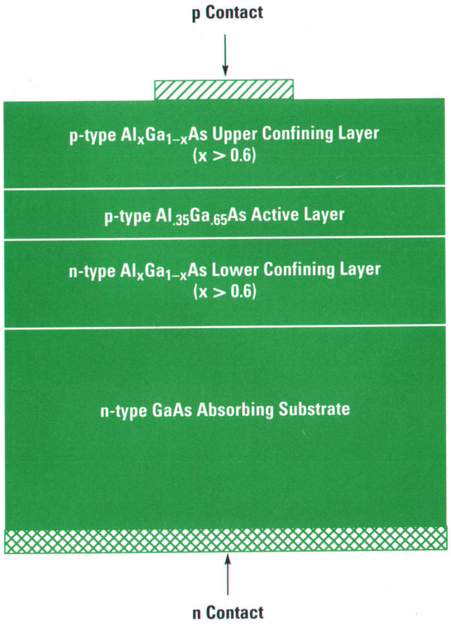

p Contact
p-type Al$_x$Ga$_{1-x}$As Upper Confining Layer (x > 0.6)
p-type Al$_{.35}$Ga$_{.65}$As Active Layer
n-type Al$_x$Ga$_{1-x}$As Lower Confining Layer (x > 0.6)
n-type GaAs Absorbing Substrate
n Contact

Fig. 3. AlGaAs double-heterostructure LED on an absorbing substrate.

OMVPE Growth of AlInGaP

AlInGaP and its related compounds GaInP and AlInP have been the subject of study since the 1960s. Only within the last eight years, however, have researchers been able to grow AlInGaP controllably and with high quality. Double-heterostructure AlInGaP semiconductor lasers that have a GaInP active layer have been commercially available for at least five years. The development of techniques for producing AlInGaP LEDs has been slower because of the greater epitaxial layer thicknesses required and because of the larger quantities needed to supply market demand. Also, high-performance LEDs require higher-quality epitaxial growth than semiconductor lasers. This is because LEDs generally operate at much lower current densities than semiconductor lasers (tens of amperes per square centimeter versus hundreds or thousands of amperes per square centimeter), and nonradiative defects can dominate the recombination process.

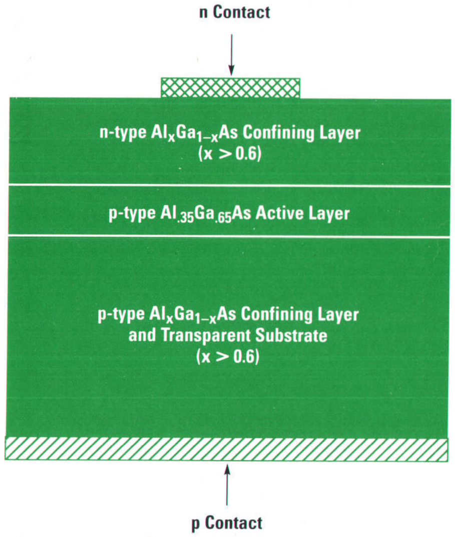

n Contact
n-type $Al_xGa_{1-x}As$ Confining Layer (x > 0.6)
p-type $Al_{.35}Ga_{.65}As$ Active Layer
p-type $Al_xGa_{1-x}As$ Confining Layer and Transparent Substrate (x > 0.6)
p Contact

Fig. 4. AlGaAs double-heterostructure LED on a transparent substrate.

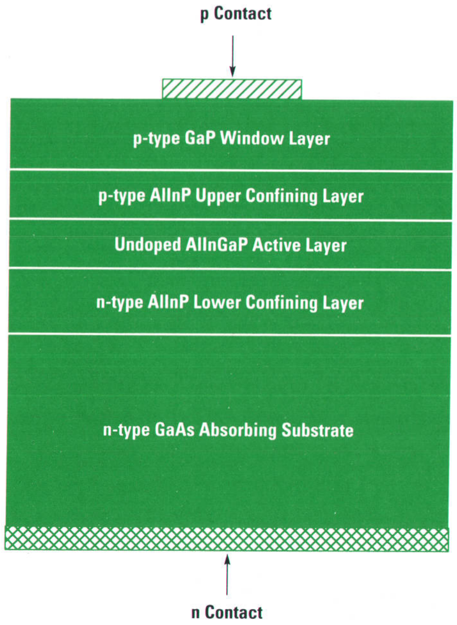

p Contact
p-type GaP Window Layer
p-type AlInP Upper Confining Layer
Undoped AlInGaP Active Layer
n-type AlInP Lower Confining Layer
n-type GaAs Absorbing Substrate
n Contact

Fig. 5. AlInGaP double-heterostructure LED on an absorbing substrate.

If the upper confining layer is grown especially thick, it can act as a mechanical "substrate," and the original absorbing GaAs substrate can be removed by chemical etching. This is a transparent-substrate double-heterostructure (TS DH) device and is shown in Fig. 4. In Fig. 4 the chip is turned upside down so that the thick AlGaAs confining layer is on the bottom. This is the most efficient type of LED chip, with external efficiencies approaching 15% for red AlGaAs lamps.

Finally, there is the AlInGaP LED structure. This device is shown in Fig. 5. It resembles the AlGaAs double heterostructure except for the presence of the GaP window layer. In the case of AlGaAs, the upper confining layer can be grown many micrometers thick, enough to couple light out of the chip efficiently. With AlInP, however, for epitaxial growth reasons it is not possible to produce a thick enough layer of high-quality AlInGaP to act as an efficient window, or even to spread the current effectively to the edges of the chip. By growing a thick GaP layer on top of the active device structure, an efficient window is produced and the sheet resistance of the p layers is reduced enough to promote adequate current spreading.

Vapor phase epitaxy (VPE) and liquid phase epitaxy (LPE) are the commonly used techniques for the mass production of LED materials. GaAsP is best grown using the VPE method, and AlGaAs and GaP are grown using the LPE method. Neither of these techniques works well for the growth of AlInGaP. A third technique called organometallic vapor phase epitaxy (OMVPE) does work well. OMVPE is similar to conventional VPE in which the reactant materials are transported in vapor form to the heated substrate where the epitaxial growth takes place. The main difference is that instead of using metallic chlorides as the source materials ($GaCl_3$ or $InCl_3$, for example), OMVPE uses organometallic molecules. The materials used in the case of AlInGaP are trimethylaluminum, trimethylgallium, and trimethylindium. Other similar organometallic compounds are sometimes used as well. As in VPE, phosphine gas is used as the source of phosphorus. By controlling the ratio of constituent gases within the reactor, virtually any composition of AlInGaP can be grown. The reactor is designed in such a way that the thicknesses of the epitaxial layers can be precisely controlled.

The schematic diagram in Fig. 3 shows a typical research-scale OMVPE reactor. In this example, the substrate sits flat on a horizontal graphite slab inside a quartz tube. Outside the tube and surrounding the graphite is a metal coil connected to a multikilowatt radio frequency generator. The graphite is heated to around 700 to 800°C by RF induction.

There are many variations on the design of the reactor chamber. For example, in some existing commercial OMVPE systems, the wafers sit on a horizontal platter and rotate either slowly or at high speed to achieve uniform growth across the wafer. Other systems use a barrel-type susceptor inside a large bell jar, similar to VPE and silicon epitaxy reactors. The method for heating the substrates can be RF induction, resistance heaters, or infrared lamps. Whatever the configuration, the conceptual nature of the growth process remains essentially the same.

The organometallic sources under normal room temperature conditions are either high-purity liquids or crystalline solids and are contained in small stainless-steel cylinders measuring about eight inches long by two inches in diameter. (Because they are pyrophoric, these materials are never exposed to air and require careful handling.) The cylinders are equipped with an inlet port connected to a dip tube, and an exit port. Hydrogen gas flowing through the dip tube and up through the organometallic liquid or solid becomes saturated with organometallic vapors. (This type of container is commonly called a "bubbler," referring to the action of the hydrogen bubbling through the liquid.) The mixture of hydrogen and vapor flows out of the cylinder and to the reactor chamber. The exact amount of organometallic vapor transported to the reactor is controlled by the temperature of the bubbler, which determines the vapor pressure of the organometallic material, and by the flow of hydrogen. The temperature of the bubblers is controlled by immersion in a fluid bath in which the temperature is regulated within ±0.1°C or better. Special regulators called mass flow controllers precisely meter the flow of hydrogen to each bubbler.

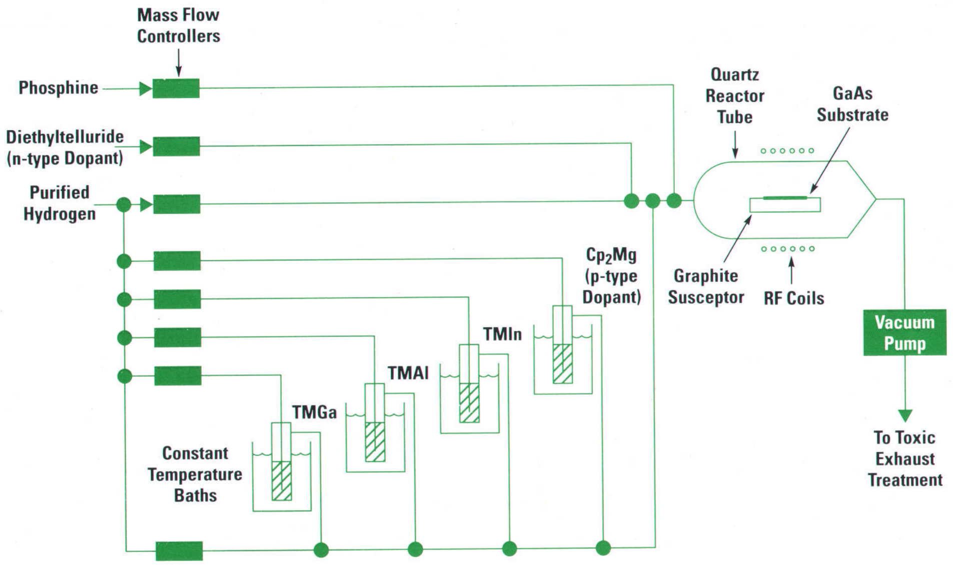


Fig. 3. Simplified schematic diagram of an organometallic vapor phase epitaxy (OVMPE) reactor.

At the entrance to the reactor chamber, the reactant gases are mixed. These gases consist of phosphine, a mixture of hydrogen and the organometallic vapors, dopant gases, and additional hydrogen added as a diluent. As the gases pass over the hot substrate, decomposition of the phosphine, organometallics, and dopant sources occurs. If all the conditions are correct, proper crystal growth takes place in an orderly atomic layer-by-layer process. Hydrogen, unreacted phosphine and organometallics, and reaction by-products such as methane are then drawn out of the reactor and through the vacuum pump for treatment as toxic exhaust waste.

The growth of III-V epitaxial materials is typically complex, and the successful production of high-quality films is dependent on many factors. The growth of AlInGaP is definitely no exception. Since this is a quaternary material system and is not automatically lattice matched to the substrate (unlike AlGaAs), the composition of the crystal lattice must be carefully controlled during the growth process. This means that each layer in the double heterostructure has to have the proper proportions of aluminum, indium, and gallium. Furthermore, the transition from one layer composition to the next often requires special consideration to avoid introducing defects into the lattice. Other factors, such as substrate temperature, total gas flow through the reactor, and dopant concentrations require careful optimization to achieve the best final device properties. Even after years of research with OMVPE, there is still a certain amount of art involved in its practice.

AlInGaP Device Structure

As mentioned previously, the high-efficiency AlInGaP LED is a double-heterostructure device. Fig. 4 shows a cross-section of a Hewlett-Packard LED with the individual epitaxial layers revealed. The light-producing part of the structure consists of a lower confining layer of n-type AlInP, a nominally undoped AlInGaP active layer, and an upper confining layer of p-type AlInP. Light is generated in the active layer through the recombination of carriers injected from the p-n junction. The confining layers enhance minority carrier injection and spatially confine the electrons and holes within the active layer, increasing the probability for band-to-band recombination. For such a structure, the internal quantum efficiency (number of radiative recombinations per total number of recombinations) can be very high, even approaching 100% for the best-quality materials.

On top of the double heterostructure is grown another layer, which serves two functions. First, it reduces the sheet resistance of the p-type layers, promoting current spreading throughout the chip, and second, it acts as a window layer to enhance coupling of the light out of the chip. Early in the development phase of the AlInGaP LEDs it was discovered that the thin upper confining layer of AlInP, ideal for confining electrons and holes in the active layer, is resistive and by itself prevents current from the central ohmic contact (shown in Fig. 4) from spreading out to the edges of the chip. In fact, with only AlInP as the top layer, virtually all of the current flows straight down, and light generation occurs only beneath the contact and is blocked from escaping the chip by the contact itself. With the addition of a thick conductive window, such as GaP, the current is able to spread out, and light generation occurs across the entire chip. Additionally, because the index of refraction of semiconductors is high (typically around 3.5), without the window much of the light produced is trapped inside the chip by total internal reflection and is eventually absorbed by the substrate. Using Snell's law and geometric optics, it can be shown that the thick window layer increases the amount of light that can escape the chip by a factor of three.[9]

Conceptually, any transparent and conductive epitaxial material could serve as the window material. From a practical standpoint, however, there are few epitaxial materials that can be grown on the AlInGaP layers that satisfy the requirements of transparency and electrical conductivity. The two best materials are AlGaAs and GaP. AlGaAs is a lattice matched material with good epitaxial growth characteristics and acceptable conductivity. However, it is transparent only in the red and orange spectral range. At wavelengths below about 610 nm, AlGaAs begins to absorb significantly. GaP, on the other hand, although mismatched to the AlInGaP lattice by 4%, is highly conductive and transparent in the spectral region from red to green, which is perfect for the spectral range of AlInGaP.

From an epitaxial standpoint, the successful abrupt growth of lattice mismatched GaP on an AlInGaP heterostructure is an interesting phenomenon. Normally, one would not expect GaP to grow as a single crystal layer directly on a mismatched "substrate" such as an AlInGaP heterostructure. It usually takes special growth techniques, such as alloy grading from one composition to the other to achieve a gradual change from the substrate lattice constant to that of the desired layer. (This is the common technique used for GaAsP epitaxy on GaAs and GaP substrates. The grading takes place over a distance of tens of micrometers of epitaxial material.) We have developed a technique for growing the GaP window directly on the AlInGaP heterostructure. The GaP at the interface with the AlInP contains a dense network of crystal defects (dislocations) caused by the lattice mismatch. The defect-rich layer is only a few hundred nanometers thick. It appears to have no effect on the transparency or conductivity of the window and the defects do not propagate down into the high-quality heterostructure where the light is generated.

Instead of growing the thick GaP window using the OMVPE technique, after the heterostructure growth is completed the wafers are removed from the OMVPE reactor and transferred to a conventional hydride VPE reactor where a 45-micrometer layer of GaP is deposited to complete the structure. The reason for the two-step growth process is to save time and cost. Organometallic sources are expensive, whereas hydride VPE requires only metallic gallium as a source. Also, the crystal growth rate using VPE can easily be ten times higher than with OMVPE, which is desirable for the growth of thick layers.

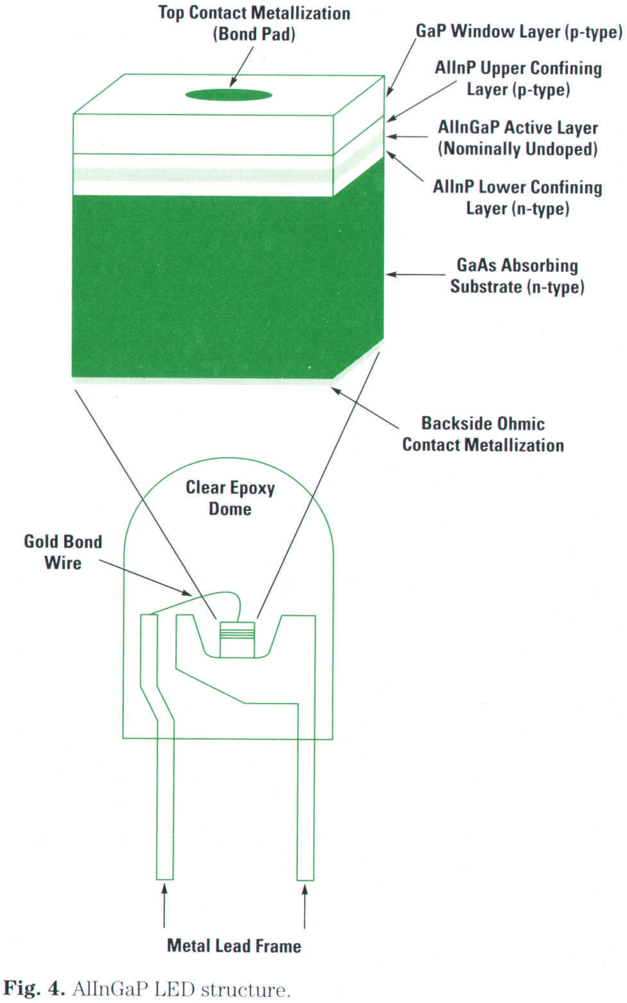

Top Contact Metallization
(Bond Pad)
GaP Window Layer (p-type)
AlInP Upper Confining Layer (p-type)
AlInGaP Active Layer (Nominally Undoped)
AlInP Lower Confining Layer (n-type)
GaAs Absorbing Substrate (n-type)
Backside Ohmic Contact Metallization
Clear Epoxy Dome
Gold Bond Wire
Metal Lead Frame

Fig. 4. AlInGaP LED structure.

Device Fabrication

The fabrication of LED chips is relatively simple compared to IC chip technologies. There is generally no high-resolution photolithography involved, and often there is no multilayer processing. The main problems arise because of the inherent difficulties in working with III-V semiconductor materials. These processes are notorious for working one day and not working the next, often without a clear explanation for the change. Processing operations, such as premetallization cleaning, metal etching, contact alloying conditions, and dicing-saw cut quality are constantly monitored and adjusted for optimum device performance.

In its simplest form, the process for making AlInGaP chips involves a metallization for the anode front contact pattern (usually a circular dot with or without fingers to promote current spreading), mechanical and/or chemical thinning of the wafer to achieve the proper die thickness, metallization on the back of the substrate for the cathode contact, and sawing the wafer into individual dice. The dice are assembled into the various lamp or display packages using automated pick-and-place machines. Conductive silver epoxy is used to attach the die to its leadframe, and gold-wire thermosonic bonding is used to bond to the top dot contact. In the case of a lamp package, the manufacturing process is completed by casting an epoxy dome around the leadframe. A cross-sectional view of a chip in a lamp package is shown in Fig. 4. Every device is tested to check the electrical characteristics, including the forward voltage at a specified current (usually 20 mA) and the reverse breakdown voltage at a specified current (usually –50 μA). Optical performance is also measured to check for light output flux, on-axis intensity, and dominant wavelength.

AlInGaP Performance

The operating characteristics of AlInGaP devices have already been briefly described, especially their high light output performance compared to other technologies. A more detailed analysis of AlInGaP performance is shown in Figs. 5 and 6. Fig. 5 shows the external quantum efficiency for AlInGaP T-1¾ lamps as a function of emission wavelength from about 555 nm to 625 nm. (These LEDs have the same double-heterostructure configuration except for the composition of the active layer which is adjusted to vary the emission wavelength.) Other types of T-1¾ LED lamps are included for comparison. Drive current is 20 mA in all cases. External quantum efficiency is a measure of the number of photons emitted from the device per electron crossing the p-n junction and is dependent on the efficiency of the semiconductor device at producing photons (the internal quantum efficiency) and on the ability to get those photons out of the chip and out of the lamp package (package efficiency). If every electron-hole pair produced a photon and every photon were extracted from the device and measured, the external quantum efficiency would be 100%.

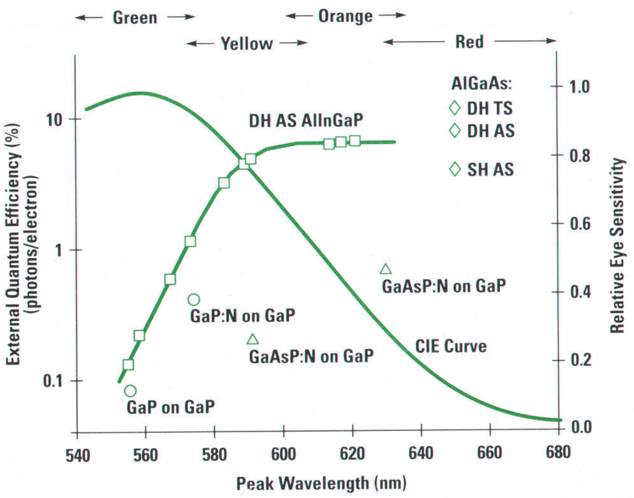


Fig. 5. External quantum efficiency of T-1¾ AlInGaP lamps compared to other technologies. Also shown is the CIE human-eye response curve.

Internal quantum efficiency is limited by the crystalline quality of the semiconductor, by the bandgap properties of the semiconductor, and by the device structure (homojunction or heterojunction). In the spectral range between 625 and 600 nm, the efficiency is almost flat. Here, crystalline quality is good, and the bandgap of the active layer is direct and well away from the indirect crossover. Also, the bandgap difference between the active layer and the upper and lower confining layers is large, providing adequate trapping of electrons and holes within the active layer and efficient radiative recombination.

As the wavelength is reduced by increasing the aluminum-to-gallium ratio in the active layer, several effects begin to lower the overall internal quantum efficiency. First, as the direct/indirect-bandgap crossover is approached, there is a greater probability for indirect-bandgap nonradiative transitions. This effect increases dramatically as the wavelength is reduced. Second, because aluminum is such a highly reactive atomic species, it has the tendency to bring undesirable contaminants, especially oxygen, into the crystal lattice with it. These impurities act as nonradiative recombination centers for electrons and holes. Consequently, as the proportion of aluminum in the active layer is increased to reduce the emission wavelength, more nonradiative recombination occurs. Finally, as the bandgap of the active layer is increased, the upper and lower confining layers become less efficient at keeping electrons and holes contained within the active layer before they recombine.

The relative importance of these three effects is still being investigated. Models describing direct/indirect-bandgap effects, defect-related nonradiative recombination, and confining layer efficiency exist. However, these models are dependent on an accurate knowledge of the bandgap of the material. For AlInGaP, there is still uncertainty about the exact bandgap properties, notably the exact location of the direct/indirect crossover. It is commonly believed that higher efficiencies at the short wavelengths should be achieved with improved epitaxial growth techniques, possibly by improving the purity of the organometallic source materials.

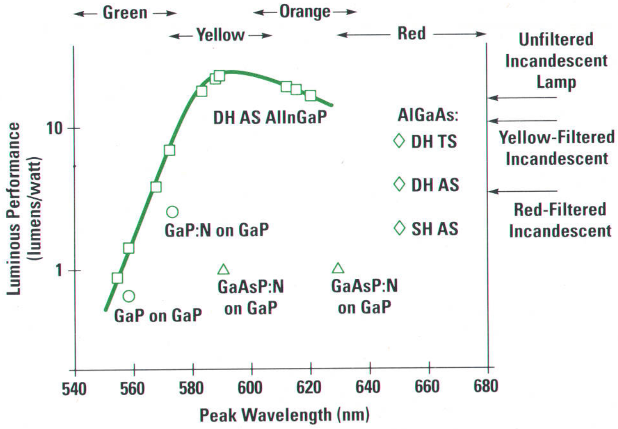


Fig. 6. LED luminous performance for AlInGaP compared to other technologies. Luminous performance is the product of power efficiency (roughly equal to quantum efficiency, Fig. 5) and the eye's response.

Once the light is produced in the active layer the task becomes one of getting the light out of the chip. Because the index of refraction of semiconductors such as AlInGaP is high (n = 3.5, approximately), most of the generated light that strikes the sidewalls of the chip is trapped within the chip either because of total internal reflection or because of Fresnel reflection. In the case of an absorbing substrate chip, such as the present AlInGaP device, reflected rays generally are lost to absorption in the substrate. We have minimized the losses from total internal reflection with the addition of the thick GaP window layer. Nevertheless, even the best external quantum efficiency theoretically possible for a cubic-shaped double-heterostructure absorbing substrate chip in air is only about 2%.

The effects of total internal reflection and Fresnel reflection are mitigated by encapsulating the chip within clear epoxy plastic shaped with a hemispherical dome (the typical LED lamp package configuration). The plastic acts as an index-matching medium between the semiconductor and the air, reducing the effects of total internal reflection and Fresnel reflection. The hemispherical shape of the plastic eliminates total internal reflection within the plastic itself and acts to focus the light from the chip. Generally, the external quantum efficiency of an encapsulated chip is increased by a factor of three, bringing the theoretical maximum external quantum efficiency to between 6% and 7% for an absorbing substrate chip.

From Fig. 5 it can be seen that at the longer wavelengths, the external quantum efficiency of AlInGaP is about 6%, comparing favorably with absorbing substrate DH AlGaAs at 7%. Only TS AlGaAs has a higher external quantum efficiency owing to the lack of absorption by the substrate. All other LED materials are less efficient than AlInGaP from 625 to 555 nm. In the yellow-to-orange wavelength range, this difference is an order of magnitude or more.

Included in Fig. 5 is the CIE relative eye sensitivity curve which shows that the eye is most sensitive to green photons and much less so to red photons. This curve is used to convert external quantum efficiency data to the luminous performance data in Fig. 6. Fig. 6 shows lumens of visible light

	Lamp Type	Dominant Color (nm)	Viewing Angle	Typical Intensity (on-axis, millicandelas)	Typical V_f at 20 mA	Typical V_r at $-100\ \mu A$
HLMA-BL00	T-4, Clear	592	3°	8400	1.9V	25V
HLMA-CL00	T-1 3/4, Clear	592	7°	2600	1.9V	25V
HLMA-DL00	T-1 3/4, Clear	592	30°	1000	1.9V	25V
HLMA-KL00	T-1, Clear	592	45°	200	1.9V	25V
HLMA-CH00	T-1 3/4, Clear	615	7°	2600	1.9V	25V
HLMA-DH00	T-1 3/4, Clear	615	30°	600	1.9V	25V
HLMA-KH00	T-1, Clear	615	45°	200	1.9V	25V
HLMA-DG00	T-1 3/4, Clear	622	30°	600	1.9V	25V

Fig. 7. Hewlett-Packard AlInGaP lamp products.

emitted from the LED lamp per watt of power applied to the diode (y axis) as a function of emission wavelength (x axis). This data is representative of how the eye actually responds to various types of LEDs. The effect of the CIE curve is to depress performance in the red part of the spectrum, resulting in a dramatic increase in apparent performance of the AlInGaP lamps compared to even TS AlGaAs lamps. It should be pointed out that the AlInGaP data shown in Figs. 5 and 6 represents the best reported results, whereas the data for the other technologies shows typical production values. Production performance values for AlInGaP are not yet established. Initially the performance will be lower than the data shown here but is expected to increase and surpass this data as the technology evolves and matures.

Also indicated in Fig. 6 are the luminous performance levels for automotive incandescent lamps, both filtered and unfiltered. These benchmarks are useful because of the interest in using LEDs instead of incandescent lamps for tail lights, brake lights, turn signals, and side marker lights on automobiles and trucks. The high efficiency of AlGaAs and AlInGaP LEDs and their long lifetimes make them attractive alternatives to incandescent light bulbs in the automotive industry. Because LEDs can be assembled into a smaller package than an incandescent bulb, automotive design can be more flexible and overall manufacturing costs lower.

The reliability of AlInGaP LEDs is generally good compared to other types of LEDs. Stress tests in which devices are driven at currents up to 50 mA at ambient temperatures ranging from –40 to +55°C show good light output and electrical stability beyond 1000 hours. Since AlInGaP LEDs have not existed for very long, device lifetime data as long as 10,000 hours is scarce. However, indications are that there are no inherent reliability problems associated specifically with AlInGaP.

For some stress conditions, AlInGaP performs significantly better than other products. For example, in high-temperature, high-humidity conditions, AlGaAs LEDs fail rapidly because of corrosion of the high-aluminum-content epitaxial layers. Since the overall aluminum content of AlInGaP devices is less than for AlGaAs, this corrosion problem does not appear, and AlInGaP LEDs perform very well in high-humidity conditions. Also, it is well-known that standard yellow GaAsP LEDs exhibit serious light output degradation when operated at low temperatures. AlInGaP LEDs demonstrate excellent low-temperature stability.

Of course, good LED device performance and reliability do not happen automatically. There are many conditions that occur during the growth of the epitaxial material and during device processing that affect initial light output, electrical characteristics, and device longevity. In fact, many factors affecting performance are not completely understood at this time. With ongoing analysis of the problems that occur, additional insight into the properties of AlInGaP epitaxial growth and device design will follow.

HP AlInGaP Products

The proliferation of AlInGaP chips into various LED packages will be an ongoing process over the next few years. Initial market demands are for T-1¾ lamp packages for moving message signs, highway warning markers, and automotive and truck lighting applications. As of this writing, several AlInGaP lamp packages are available in three colors from amber to red-orange. These products are listed in Fig. 7.

Conclusion

We have attempted to provide a general description and understanding of HP's new family of LEDs made from AlInGaP. We have compared the performance and production of AlInGaP devices with other LED technologies. We have also tried to give the reader a general understanding of LEDs and the III-V processes necessary for their manufacture.

HP's AlInGaP devices represent the brightest visible LEDs that have ever been made. Interest in them is quickly growing as manufacturers come up with new applications for them. Although comparably bright red AlGaAs LEDs have been available for several years, the appearance of bright orange and yellow lamps has made possible total LED replacements in applications where low-wattage filament lamps have been used exclusively. The benefits of LEDs include long lifetime, performance reliability under a broad range of operating conditions, and overall cost savings over traditional incandescent lamps.

Acknowledgments

The development of the AlInGaP LEDs took a number of years, starting from the initial R&D phase when we struggled to grow even single layers of not-very-good epitaxial material. Since then we have come a long way towards bringing AlInGaP out of the laboratory and into the product line. The authors wish to thank Chris Lardizabal, who has worked on processing AlInGaP wafers and testing devices from the very start of the project, and Tia Patterakis, Susan Wu, Anna Vigil, and Charlotte Balassa for processing the wafers, helping with epi growth, and endless testing of AlInGaP chips and lamps. Other people who deserve recognition for helping to develop and understand AlInGaP LEDs include Doug Shire, Dan Steigerwald, and Frank Steranka. Finally, we would like to thank our R&D manager, George Craford, for his continuous support and encouragement.

References

1. N. Holonyak, Jr., and S.F. Bevacqua, "Coherent (Visible) Light Emission from GaAsP Junctions," *Applied Physics Letters*, Vol. 1, 1962, p. 82.
2. R.A. Logan, H.G. White, and W. Wiegmann, "Efficient Green Electroluminescence in Nitrogen-Doped GaP p-n Junctions," *Applied Physics Letters*, Vol. 13, 1968, p. 139.
3. W.O. Groves, A.J. Herzog, and M.G. Craford, "The Effect of Nitrogen Doping on GaAsP Electroluminescent Diodes," *Applied Physics Letters*, Vol. 19, 1971, p. 184.
4. M.G. Craford, R.W. Shaw, W.O. Groves, and A.H. Herzog, "Radiative Recombination Mechanisms in GaAsP Diodes with and without Nitrogen Doping," *Journal of Applied Physics*, Vol. 43, 1972, p. 4075.
5. J. Nishizawa and K. Suto, "Minority-Carrier Lifetime Measurements of Efficient GaAlAs p-n Heterojunctions," *Journal of Applied Physics*, Vol. 48, 1977, p. 3484.
6. F.M. Steranka, et al, "Red AlGaAs Light-Emitting Diodes," *Hewlett-Packard Journal*, Vol. 39, no. 8, August 1988, pp. 84-87.
7. C.P. Kuo, et al, "High Performance AlGaInP Visible Light-Emitting Diodes," *Applied Physics Letters*, Vol. 57, 1990, p. 2937.
8. R.M. Fletcher, et al, "The Growth and Properties of High-Performance AlGaInP Emitters Using a Lattice Mismatched GaP Window Layer," *Journal of Electronic Materials*, Vol. 20, 1991, p. 1125.
9. K.H. Huang, et al, "Twofold Efficiency Improvement in High-Performance AlGaInP Light-Emitting Diodes in the 555-to-620-nm Spectral Region Using a Thick GaP Window Layer," *Applied Physics Letters*, Vol. 61, 1992, p. 1045.

HP Task Broker: A Tool for Distributing Computational Tasks

Intelligent distribution of computation tasks, collective computing, load balancing, and heterogeneity are some of the features provided in the Task Broker tool to help make existing hardware more efficient and software developers more productive.

by Terrence P. Graf, Renato G. Assini, John M. Lewis, Edward J. Sharpe, James J. Turner, and Michael C. Ward

HP Task Broker is a software tool that enables efficient distribution of computational tasks among heterogeneous computer systems running UNIX*-system-based operating systems. Task Broker performs its computational distribution without requiring any changes to the application. Task Broker relocates a job and its data according to rules set up at Task Broker initialization. The other capabilities provided by Task Broker include:

- Load balancing. Task Broker can be used to balance the computation load among a group of computer systems. Since Task Broker has the ability to find the most available server for a computation task transparently, it can effectively level the load on a compute group, thus helping to make existing hardware more efficient.
- Intelligent targeting. Task Broker can transparently target specific servers most appropriate for a specialized task. For example, a graphics simulation application may be more efficiently executed on a machine with a graphics accelerator or fast floating-point capability. These targeting characteristics can be built into the Task Broker group definition without requiring the user to have any machine-specific knowledge. Thus, expensive resources don't need to be duplicated in a network.
- Collective computing. Task Broker allows a network of workstations to form a computational cluster that can replace a far more expensive mainframe or supercomputer. This approach offers multiple advantages over the single compute server model. Some of these advantages include increased availability (no single point of failure), improved scalability (ease of upgrade), and reduced costs. See "HP Task Broker and Computational Clusters," on page 16.
- Heterogeneity. Task Broker can be used to create a heterogeneous cluster, allowing a network of machines from multiple vendors to interoperate in a completely transparent fashion. Task Broker will run on several different workstation platforms, all of which can interoperate as servers and clients.
- DCE Interoperability. Task Broker is able to take advantage of many of the services provided by HP's DCE (Distributed Computing Environment) developer's environment. See "Task Broker and DCE Interoperability," on page 19.

HP Task Broker runs on HP 9000 Series 300, 400, 600, 700, and 800 computers running the HP-UX* operating system, and the HP Apollo workstations DN2500, DN3500, DN4500, DN5500, and DN10000 running Domain/OS. In addition, Scientific Applications International Corporation (SAIC) has ported Task Broker to the Sun3, Sun4, and SPARCstation platforms.

Automated Remote Access

The need to access remote computer resources has existed ever since computers were tied together by local area networks. Remote access gives the user a means of increasing productivity by allowing access to more powerful or specialized computer resources.

To access a remote resource, computer users have had to rely on guesswork for determining optimal placement and have been saddled with the tedious activity of manually moving files to and from a resource.

Task Broker effectively automates the manual tasks required for distributing computations by:

- Gathering machine-specific knowledge from the end user
- Analyzing machine-specific information and selecting the most available server
- Connecting to a selected server via telnet, remsh (remote shell), or crp (create remote process)
- Copying program and data files to a selected server via ftp (file transfer protocol) or NFS (Network File System)
- Invoking applications over the network
- Copying the resulting data files back from the server via ftp or NFS.

Each of the above steps is done automatically by Task Broker without the user needing to be aware of, or having to deal with, the details of server selection and data movement.

Server selection is one of the most significant contributions provided by Task Broker. For the user to determine the most appropriate server for a job manually, all of the dynamic variables of server availability would have to be captured before every job submittal. Because this is a time-consuming, cumbersome process, developers trying to run a job would spend very little time selecting an appropriate server.

Instead, developers would revert to using either their own machine for compute jobs or just a few popular machines, overloading those machines and underloading others. In addition, having to manage several network connections

HP Task Broker and Computational Clusters

A computational cluster is a group of workstations networked together and used as a single virtual computational resource. This notion is an extension of the Task Broker cluster concept, since it is based on the idea that a cluster of workstations can actually replace a mainframe.

The motivation behind this concept comes from customers who are downsizing from a single compute server, such as a mainframe or supercomputer, or customers who have computationally intensive tasks that can execute more effectively on a cluster of workstations.

The advantages of the computational cluster over the resource that it is intended to replace are several:

- The cluster can be considerably less expensive then a mainframe.
- The cluster is modular and therefore more easily upgradable.
- The cluster can consist of workstations that may already exist in the environment.

Task Broker has an obvious role in this area of computing, since the computational cluster is really a special case of the Task Broker solution. However, it is important to note that, in terms of distributing computations, only a portion of the mainframe replacement solution would be provided by Task Broker in its current form.

Task Broker represents the class of solutions that provide a mechanism for coarse-grained parallelism (i.e., giving the user the ability to run multiple tasks or applications in parallel). The goal of this type of solution is to achieve parallelism without impacting the application, or to maximize the use of hardware.

A finer level of parallelism can be provided by tools that can break up an application into subtasks and run them in parallel. The subtasks can be procedures, loops, or even instructions. The goal of these solutions is to have an application complete in the minimum time possible, as opposed to those of the coarse-grained alternative.

This area of computing is obviously more involved then can be covered here. The point to be made is that customers are in need of new ways of optimizing their use of hardware, and Task Broker can, in its current form, provide a solution. Task Broker can provide parallelism at the application level, which is a major portion of the computational cluster solution.

simultaneously to try to balance the workload is also cumbersome, and tends to lead to the same result. The end result is increased frustration and decreased productivity.

Task Broker automates these services, which most developers find difficult to manage manually.

Bidding and Execution

A machine running Task Broker can act as a client, a server, or both. A Task Broker client is a submitter of jobs into the compute group, and a Task Broker server is a machine that provides services for clients. A single instance of Task Broker, called the Task Broker daemon, resides on each client and server.

Each server provides one or more services for the work group, each of which represents a specific compute job. Servers can provide any number of services, and services can be provided by one or more servers (which would be necessary to load balance the compute group).

Task Broker clients and servers interact to distribute and execute jobs in the following manner:

1. A user submits a request for a service to the local Task Broker daemon (client daemon).

2. The client daemon sends a message to the group of servers, requesting bids to service the submitted job.

3. The servers compute their bids, or *affinity values*, for the requested service, based on their availability to accept the job. The bids are returned to the client.

4. The client waits a preset amount of time for the servers to return their bids and selects the server with the highest bid.

5. The client transmits the necessary files (if necessary) to the selected server.

6. The server executes the job according to instructions in the local execution script.

7. At job completion, the server returns the output files to the client, which are then placed in the user's working directory.

Since every job submitted to the work group involves bidding before acceptance by a server, and the bids can be computed dynamically based on the server's availability at that time, the jobs are automatically serviced by the most appropriate machine. A failing machine will automatically be avoided by this bidding mechanism, increasing the fault tolerance of the group. The basis for the bids or affinity values is described later.

If there are no available servers when bids are requested, or if the returned bids do not exceed a preset threshold because the servers are all being heavily used, the job will be put into a local queue. The jobs in the local queue will be resubmitted for bids after a preset time limit or by receiving a callback from a newly available server. In addition, the job may execute locally if the submitting machine can also provide the requested service.

Each daemon maintains a log file that is used to record daemon activity. These can be used to analyze the machine use in the work group and can be the basis for fine tuning the Task Broker installation.

Task Broker Setup

Task Broker setup takes place when the product is installed. Installation and setup are performed by a Task Broker administrator. The Task Broker administrator is a user with the appropriate permissions to initialize and modify the Task Broker installation of daemons and setup files.

When hardware changes are needed in the network the administrator needs to make sure the Task Broker setup files are kept current. In addition, the administrator can make changes to the daemon's setup files to fine tune the installation. To assist the administrator in this analysis, Task Broker can collect information about daemon and service activity through the use of its logging feature or its accounting file. Administrator duties are given in reference 1.

Each machine running a Task Broker daemon needs some or all of the following files to operate as either a client or a server:

Configuration File. This file specifies what services are provided, when the services are available, and who has access to these services. It also specifies how services are to be provided, and under what conditions (see Fig. 1). The contents of a configuration file are divided into the following categories:

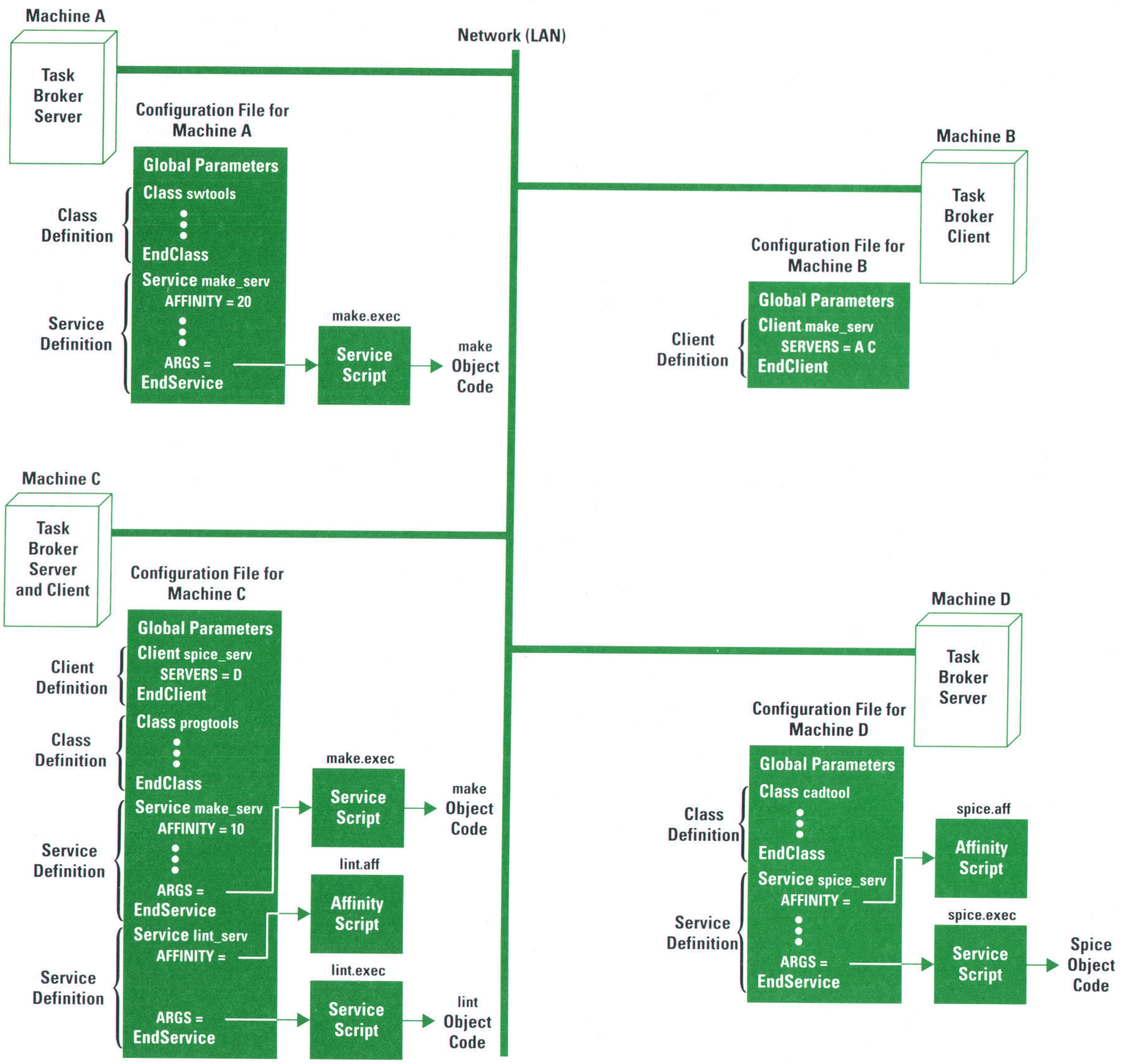


Fig. 1. An overview of Task Broker configuration files.

- Global parameters. These parameters specify changes to Task Broker default values that govern the global conditions on the local Task Broker computer. The parameter that governs the waiting period for the task placement process and the parameter that specifies whether to record CPU time used by local tasks are examples of global parameters.
- Class definition. This definition specifies the maximum number of services belonging to a named class that can run on the local server at one time. Every service specified must be a member of the specified class. For example, Spice might be a member of a class specified as cadtools.
- Client definition. This definition specifies the servers that can provide service to a client.
- Service definition. This definition specifies items such as the local server's ability to provide a particular service, how the service will be processed, the affinity value or affinity script, and a list clients that have access to the service.

Service Script. This is a shell script that defines how each service being provided by the server is carried out. This script typically invokes an application that provides the requested service. This script is specified by the ARGS parameter in the service definition portion of the configuration file.

Affinity Script. This script defines the algorithm to be used by the server to compute the affinity value when a job is bid on. If a constant is used to define the affinity value, this script is not needed.

Submit Script. This script, which is invoked from a Task Broker client, submits a service request for a Task Broker service. A service request contains information such as optional parameters or data files that cause the service to be run in a specific manner.

Affinity Value

The affinity value is an integer from 0 to 999 that quantifies a Task Broker server's ability to provide a specific service. The value may reflect the availability of certain computer resources such as disk space or other factors essential to perform the service.

Affinity values can either be hard-coded into the service script, which resides in each server's configuration file, or can be calculated before each bid submittal through the use of an affinity script. For example, the following script uses a hard-coded affinity value.

```
# Task Broker Service Definition
#
Service foo
   CLASS = service_tasks
   MAX_NUMBER = 2
   ALLOW = (12.34.567.*)
   MIN_FREESPACE = 30000
   AFFINITY = 10
   .
   .
   .
Endservice
```

In the above case, when the server daemon receives a service request for the foo service, it checks the service definition in its configuration file. In this case, the daemon checks several parameters for each service request it receives. Some of these checks ask the following questions:

- Is the number of tasks running less than the maximum (MAX_NUMBER)?
- Is the requester allowed to run the service here (ALLOW)?
- Are there 30M bytes of free disk space (MIN_FREESPACE)?

If the answer to all the above questions is "yes," the server daemon sends the affinity value of 10 as its bid for the requested service. If any of the answers is "no," no bid is returned to the requesting client.

The service definition can also invoke an affinity script as in the following example.

```
# Task Broker Service Definition
#
Service foo
   CLASS = service_tasks
   AFFINITY = "/users/tbroker/lib/foo.aff"
   .
   .
   .
Endservice
```

The shell script foo.aff could possibly include the parameters specified in the first example's service definition such as MAX_NUMBER, ALLOW, and MIN_FREESPACE. It could also include checks on the machine or user submitting the request and checks on whether the data to be accessed is locally resident. The result is that depending on the outcome of the checks, the script will or will not send an affinity value to a requesting client.

For load balancing to take place properly, the affinity scripts should be identical on every computer in the compute group. Since the affinity values returned by the server daemons directly affect the placement of jobs in a work group, proper parameter selection in the affinity scripts is the key to optimal server selection.

Example: A Distributed Make Facility

This example will show how Task Broker can be used to create a distributed make facility, enabling compilations to be distributed to different workstations on the network so that they can execute concurrently, resulting in linked binaries when everything is completed successfully. The procedure is summarized in Fig 2.

The process begins with the user on the client machine creating C program source files (ⓐ in Fig. 2) and placing them in the source file directory. At ⓑ compiles are initiated at the client by executing a makefile, which in turn invokes a submit script (tbmake in this example). The submit script

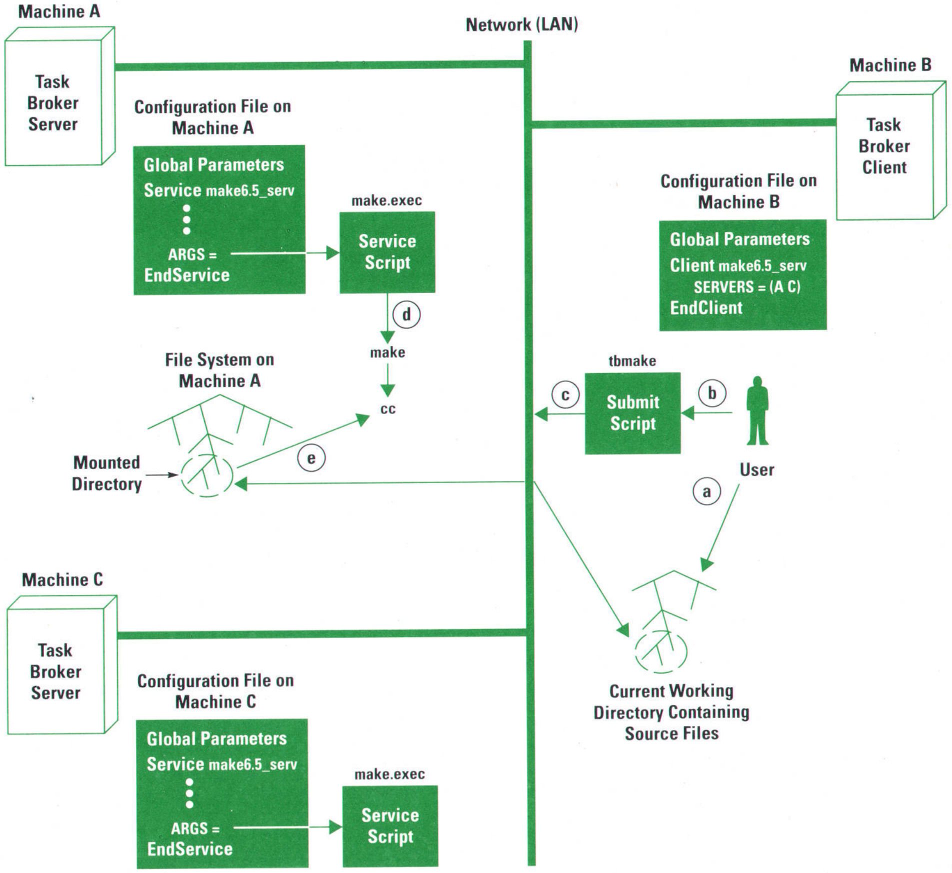


Fig. 2. The flow of activities during a remote program compilation.

submits a compile request to the local daemon, and at ⓒ the client daemon submits a make6.5_serv request, including information about the directory containing the source files. The servers each bid on the compile request and when the selected server is available (machine A in this case) its daemon accepts the compile request and invokes its local cc service script (ⓓ in Fig. 2). The service script can access the client's file system via NFS with Task Broker performing the file system mounts if necessary (ⓔ in Fig. 2).

The client's submit script is written such that it will wait for successful completion of all compiles before requesting bids for the link service. When the server accepts the link request, the compiled code is linked to create an executable program. Finally, the file system containing the source files and the executable program file is unmounted.

This example demonstrates several key features of Task Broker:

- Multiple instances of existing applications can be executed concurrently in a work group with very little effort.
- Task Broker provides a flexible way of delaying the execution of an application until conditions necessary for its execution are in place. In this case, the link operation was delayed until the distributed compiles completed successfully.
- The service scripts can be written to access remote data via mechanisms such as NFS mounts of client file systems.

Configuration Strategies

The following two examples of Task Broker configurations will demonstrate different philosophies of its use.

Task Broker with a Mainframe. The first example, which is shown in Fig. 3, illustrates how a group of Task Broker daemons can have their services augmented by a mainframe.

Task Broker and DCE Interoperability

The HP DCE (Distributed Computing Environment) developer's environment provides a common standards-based framework for distributed administration, application development, and execution in a network of heterogeneous computer systems. Designed to support the HP 9000 Series 700 and 800 computer systems running the HP-UX 9.0 operating system, HP's DCE developer's environment is an implementation of the Open Software Foundation (OSF) DCE developer's services with additional tools for DCE-based application development.

The DCE core services include security service, remote procedure call (RPC), directory services, time service, and threads. Extended services such as the distributed file system are also provided.

The current Task Broker already benefits from, and can make use of many of these DCE services. Since DCE was designed to provide benefits without necessarily requiring changes to existing applications, Task Broker can invoke applications that explicitly use some DCE services without modifications to Task Broker or the applications invoked by Task Broker. These DCE services include:

- Remote Procedure Call (RPC). An application written using RPC can be distributed in a work group by Task Broker.
- Time Service. The host machines in a compute group can use the time service to keep their clocks synchronized. This can greatly simplify the management of a Task Broker installation because items such as the Task Broker daemon log files will have their time stamps synchronized.
- Directory Services. Applications that make use of directory services can be managed by Task Broker without restriction.
- Threads. As with RPC, a multithreaded application can be distributed by Task Broker without modification.
- Distributed File System. This feature is not only compatible with Task Broker, but will greatly simplify distributed access in the Task Broker work group.
- Diskless Support. Task Broker will operate on diskless machines without modification.

For Task Broker to take advantage of other DCE services such as the security service will require internal changes to Task Broker.

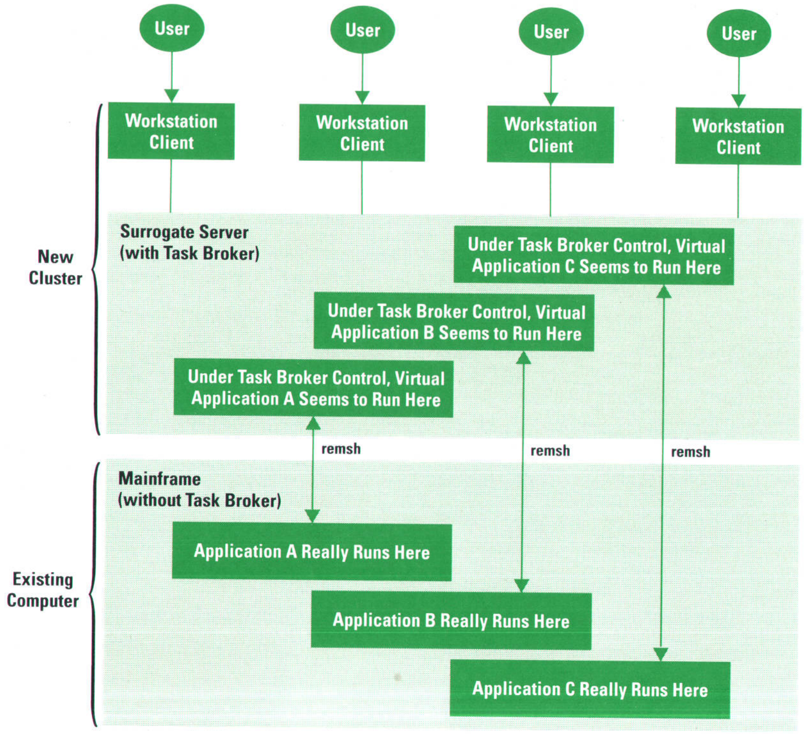

User
User
User
User
Workstation Client
Workstation Client
Workstation Client
Workstation Client
New Cluster
Surrogate Server (with Task Broker)
Under Task Broker Control, Virtual Application C Seems to Run Here
Under Task Broker Control, Virtual Application B Seems to Run Here
Under Task Broker Control, Virtual Application A Seems to Run Here
remsh
remsh
remsh
Mainframe (without Task Broker)
Existing Computer
Application A Really Runs Here
Application B Really Runs Here
Application C Really Runs Here

Fig. 3. A group of Task Broker daemons having their services augmented by a remote mainframe.

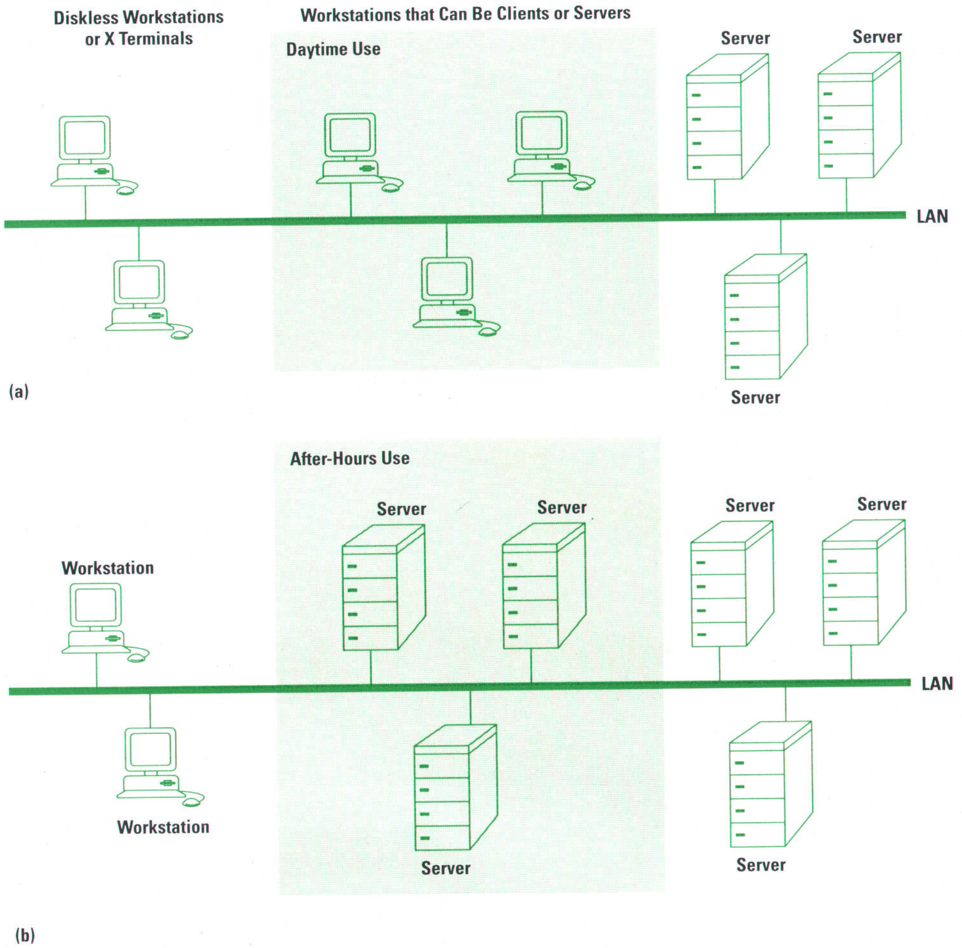


Fig. 4. A flexible Task Broker work group in which certain workstations are configured to be either clients only or servers only depending on the time of day. Of course these systems have the software and hardware capabilities to be clients or servers. (a) During the daytime these dual-role systems are configured to be clients only. (b) In the after hours the systems are used as servers only.

Each of the Task Broker daemons acts as a server representing a mainframe service in the work group. The bids made by the daemons indicate the ability of the mainframe to take on additional work. Using Task Broker to combine a group of workstations with a mainframe in this way has several key advantages:

- The mainframe resources can become transparently and seamlessly included in the work group without porting any of its applications.
- The workstation users can gain access to mainframe resources without machine-specific knowledge, or even any knowledge that the mainframe is being accessed in their calculations.
- A Task Broker daemon does not need to be present on every host in a work group because a host can have a surrogate server in the group acting on its behalf.

The result in this example is that Task Broker allows overall hardware use to increase along with the group's productivity with minimal impact on either hardware or software and little added expense.

Flexible Work Group. The second example of a Task Broker configuration demonstrates how Task Broker can be used to create a flexible work group. During the day the clients shown in Fig. 4 access a dedicated server group, and during the evening hours, when most users have gone home, some of the clients become servers.

This example makes use of Task Broker's ability to delay the submittal or acceptance of jobs until after a certain time of day has passed. This can be done either in the submit script, delaying the time when clients request bids for the service, or by setting the "time-of-day" parameter in the affinity script, delaying the time when certain server daemons will begin generating bids for any service.

Using Task Broker to implement this form of flexible configuration can contribute to a group's productivity in several ways:

- Workstation users can access dedicated compute services during the day (in this case the server pool) and can have their machine automatically added to the server pool after work hours.
- Large jobs requiring a large amount of compute power can be queued to execute after hours to take advantage of the increased size of the server pool.
- Once the Task Broker work group has been set up as described, no intervention is needed to maintain a flexible configuration. If a user wishes to remove a machine from the server pool, a quick change to its affinity script is all that is necessary.

These two examples are intended to show that Task Broker can be used to add flexibility to an existing network as well as increase access to computer resources that were previously inaccessible.

Task Broker and Other Alternatives

The strategy behind the Task Broker design is that in most cases the user is interested in:

- Having a job placed and executed as efficiently as possible and not in controlling the placement of a job
- Distributing tasks at the application level rather than the procedure level
- Having a tool that will require no changes to the application to perform its function.

Job Placement. Although Task Broker provides the user with the ability to target specific machines for specialized tasks, its primary emphasis is to free the user from concerns about job placement. In environments using scarce resources, such as a single supercomputer, there is a similar need for a tool to provide a way of preventing users from monopolizing that resource.

For example, suppose some installation has a tool that controls job queues on a mainframe. In this case, the user submits a request to one of several queues along with a set of options specifying execution limits, priority, and so on. The tool then accepts or rejects the queued job based on resource limits and other factors. If accepted, and there are available slots for immediate execution, the tool removes the request from the head of the queue and the request is serviced. The request will execute concurrently with other accepted jobs, based on an administrative limit.

Task Broker provides a more general solution to this problem. It views the entire network of machines as a scarce resource, and by load balancing the resources, it prevents any one machine in the group from being monopolized, or any user from monopolizing too many resources. Thus, Task Broker will not forward a job to a server unless one is sufficiently available.

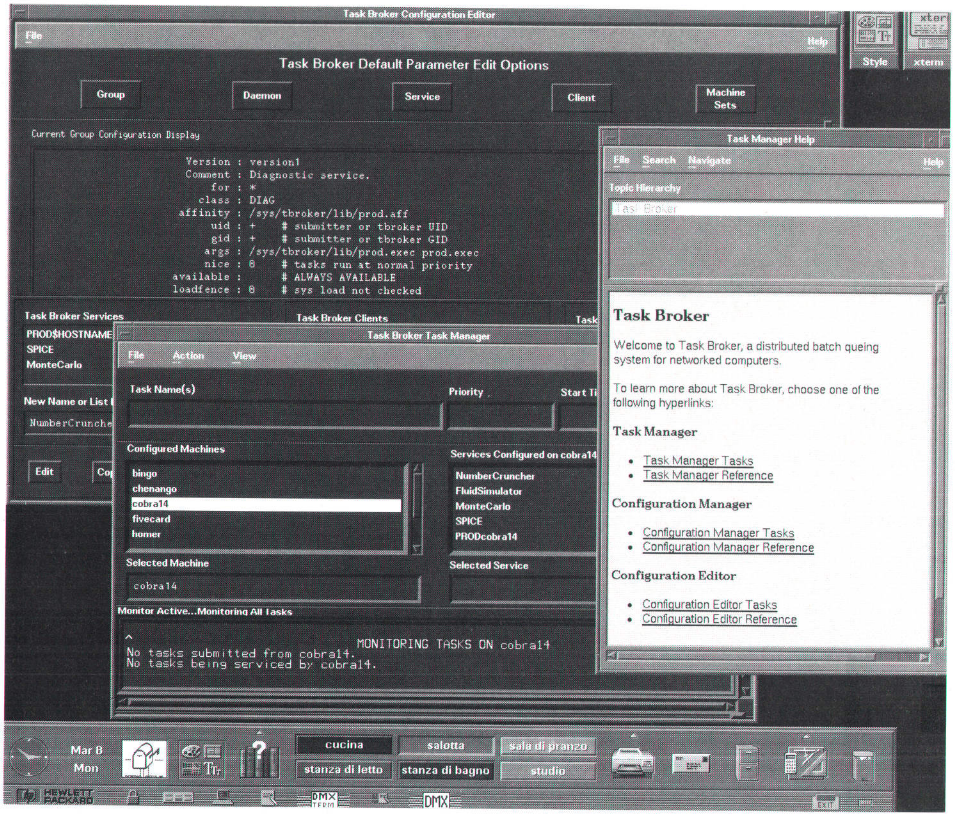
In addition, Task Broker provides mechanisms such as file transfers, remote file system mounts, and affinity calculations based on configuration that obviate the need for concerns about job placement.

Granularity of Distribution. Task Broker distributes tasks at the application level. Alternate strategies of distributed computation, such as remote procedure call (RPC), provide remote placement at the procedure level.

HP Task Broker Version 1.1

The accompanying article describes the features provided in the first version of Task Broker. The new version of Task Broker contains all the features contained in Version 1.02 and adds the following features:

- A graphical user interface (GUI) has been added to improve the product's ease of use. The GUI provides a visual interface to most of the Task Broker's command set and configuration information. Fig. 1 shows some of the windows provided in this new GUI for configuration management.
- Centralized configuration management has been added to allow the entire Task Broker installation to be initialized using a single group configuration and to be administered from any single machine site. What this means is that the data in the configuration files described in the accompanying article can be located at one machine site.
- An integrated forms-based configuration editor is provided. The configuration syntax is simpler and checking is done during the editing session.
- Finally, an online, context-sensitive help subsystem has been added.



Task Broker Default Parameter Edit Options

Current Group Configuration Display

```
  Version : version1
  Comment : Diagnostic service.
      for : *
    class : DIAG
 affinity : /sys/tbroker/lib/prod.aff
      uid : +    # submitter or tbroker UID
      gid : +    # submitter or tbroker GID
     args : /sys/tbroker/lib/prod.exec prod.exec
     nice : 0    # tasks run at normal priority
available :      # ALWAYS AVAILABLE
loadfence : 0    # sys load not checked
```

Task Broker Services: PROD$HOSTNAME, SPICE, MonteCarlo

New Name or List: NumberCrunche

Configured Machines: bingo, chenango, cobra14, fivecard, homer

Services Configured on cobra14: NumberCruncher, FluidSimulator, MonteCarlo, SPICE, PRODcobra14

Selected Machine: cobra14

Selected Service:

Monitor Active...Monitoring All Tasks

```
^
                    MONITORING TASKS ON cobra14
No tasks submitted from cobra14.
No tasks being serviced by cobra14.
```

Task Broker

Welcome to Task Broker, a distributed batch queing system for networked computers.

To learn more about Task Broker, choose one of the following hyperlinks:

Task Manager

- Task Manager Tasks
- Task Manager Reference

Configuration Manager

- Configuration Manager Tasks
- Configuration Manager Reference

Configuration Editor

- Configuration Editor Tasks
- Configuration Editor Reference

Fig. 1. The new Task Broker graphical user interface.

The difference represents a trade-off of computational control versus ease of implementation. RPC requires procedure calls in an application to be replaced by call stubs in an intermediate definition language. These stubs handle the remote placement of the actual procedure call. As such, RPC requires customized application source code, most of which must be redesigned and reimplemented if not originally implemented using RPC.

With RPC the procedure is usually located on a centralized server, or replicated in several places (requiring the servers to keep the replicas synchronized). While the server side of the application is executing, the client side is not, reflecting the synchronous nature of procedure calls.

In summary, Task Broker is nonintrusive to application source code (satisfying the third user interest above) and allows the execution of the applications it distributes to take place concurrently. It does, however, limit the user to remote placement at the application level. RPC gives a finer level of computational control, but requires source code changes and does not provide a mechanism for concurrent execution.

Conclusion

Task Broker can provide many benefits to an organization with a network of computers. Because of its flexibility, Task Broker can easily be tailored to provide a simple distributed solution to many additional types of situations. As a tool for distributing computation tasks, Task Broker can provide a way to make existing hardware more efficient by increasing its level of use, and software developers more productive by providing a way to access an expanded set of computing resources.

Acknowledgments

The author wishes to acknowledge the contribution of the following people in HP's User Interface Technology Division in Fort Collins, Colorado who preceded us as the caretakers and spokespersons for Task Broker: Ken Sandberg, John Metzner, David Wright, Stewart Mayott, Gary Thundquist, Sean Tracy, Mark Ostendorf. In addition, Gary Kessler and Bob Murphy, of HP's Chelmsford Systems Software Lab and the Workstation Group respectively, have contributed to this article.

Reference

1. *Task Broker Administrator's Guide*, HP Part Number B1731-90003.

The HP-RT Real-Time Operating System

An operating system that is compatible with the HP-UX* operating system through compliance with the POSIX industry standards uses a multi-threaded kernel and other mechanisms to provide guaranteed real-time response to high-priority operations.

by Kevin D. Morgan

HP-RT† is Hewlett-Packard's real-time operating system for PA-RISC computers. It is a run-time-oriented product (as opposed to a program-development-oriented product) based on industry standard software and hardware interfaces. HP-RT is intended to be used as a real-time data acquisition and system control operating system. It is designed around the real-time system principles of determinism (predictable behavior), responsiveness, user control, reliability, and fail-soft operation. These characteristics distinguish a real-time operating system from a nonreal-time operating system. This article reviews some of these characteristics of HP-RT and discusses the specific designs used to provide these features.

HP-RT runs on the HP 9000 Model 742rt VMEbus board-level computer, which is based on HP's PA-RISC 7100 technology (see Fig. 1). The 742rt is designed to fit into a VMEbus card cage or an HP 9000 Model 747i industrial workstation cabinet.[1]

The HP-RT kernel is compatible with the HP-UX operating system through compliance with the following industry standards:

- POSIX (Portable Operating System Interface) 1003.1, which defines a standard set of programmatic interfaces for basic operating system facilities
- POSIX 1003.4 draft 9, which defines the standards for real-time extensions
- POSIX 1003.4a draft 4, which defines the standards for process-level threads.

HP-RT also supports C/ANSI C, C++, PA-RISC assembly language, and many SVID/BSD (System V Interface Definition/Berkeley Software Distribution) commands and functions.

HP-RT Software

The HP-RT software is divided into two main categories: the HP-RT kernel and the optional HP-RT services (see Fig. 2).

HP-RT Services. The optional HP-RT services include the following components:

- Network services including the Network File System (NFS), TCP/IP, Berkeley sockets, and ARPA/Berkeley networking services
- Libraries for developing OSF/Motif graphical user interfaces and X clients
- Development tools to help users create applications to run in the HP-RT environment
- Cross debuggers hosted on an HP-UX development workstation for debugging the HP-RT kernel or applications running on an HP-RT target system.

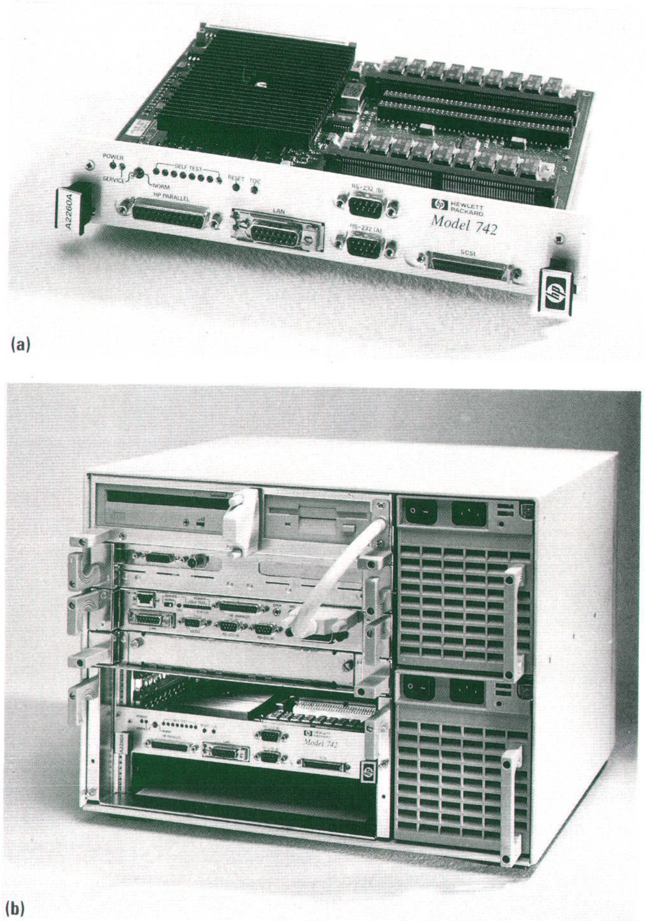


Fig. 1. (a) The HP 9000 Model 742rt board-level computer. (b) An HP 9000 Model 747i industrial workstation with a Model 742rt loaded in.

† HP-RT is derived from a third-party operating system called LynxOS from Lynx Real-Time Systems Inc. All kernel-level algorithms and data structures described in this paper are based on LynxOS features.

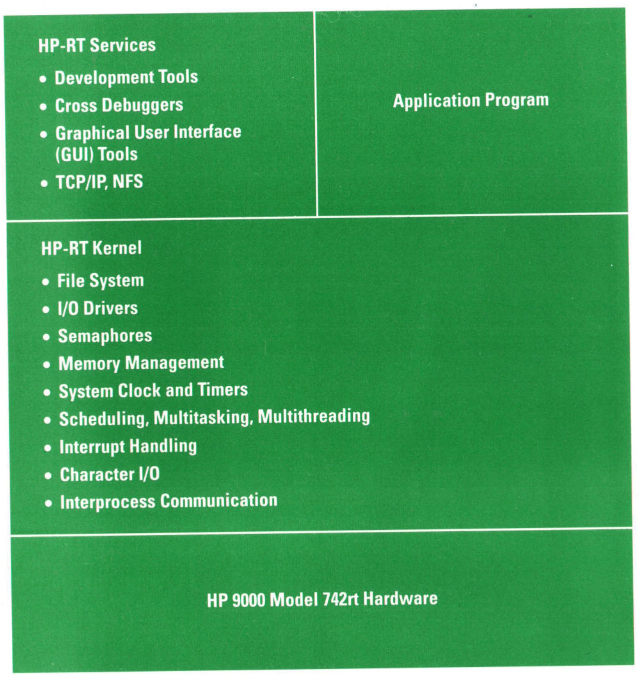


Fig. 2. The HP-RT kernel and services.

Kernel Software. The HP-RT kernel is designed so that it can be scaled to balance memory and performance requirements. It is small to reduce overhead. The kernel components include:

- A counting semaphore mechanism for process synchronization and to help ensure atomicity around critical sections of code.
- A system clock that generates time interrupts every 10 milliseconds. Thus, time events using standard software interfaces have a 10-millisecond resolution. For higher timing accuracies, drivers and user processes can access the hardware timers on the Model 742rt. These timers have 1-μs resolutions and are 16 and 32 bits wide.
- I/O drivers for Ethernet, SCSI II, RS-232-C, and parallel I/O for the Model 742rt computer, and guidelines for writing VMEbus drivers
- Standard operating system services such as:
  - ○ Scheduling, multitasking, and multithreading
  - ○ Memory management
  - ○ Interrupt handling
  - ○ Character I/O
  - ○ Interprocess communication
  - ○ POSIX 1003.1, .4, and .4a kernel services.

Many of these components are described in more detail later in this article.

HP-RT Development Environment. The development environment for HP-RT is shown in Fig. 3. Programs created to run on the Model 742rt in the HP-RT environment are developed (using PA-RISC compilers and linkers) on an HP 9000 Series 700 or 800 HP-UX system. The executable programs can be downloaded via LAN to a local disk on the target system (Model 742rt), or implicitly downloaded when the program is executed via NFS mounting between the HP-RT and HP-UX systems. The user can debug the downloaded program from the host system via the RS-232-C and LAN connections between the two systems. Users can customize the SoftBench software development environment[2] on the development host to launch programs to a remote HP-RT system and to launch the correct program debugger for HP-RT program debugging.

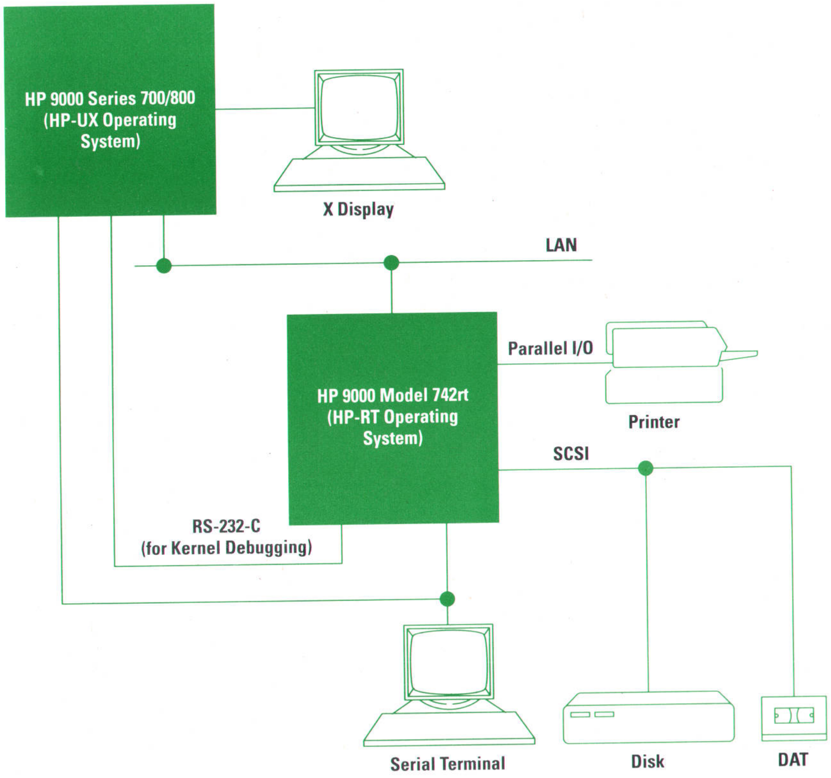

HP 9000 Series 700/800 (HP-UX Operating System)
X Display
LAN
Parallel I/O
Printer
HP 9000 Model 742rt (HP-RT Operating System)
SCSI
RS-232-C (for Kernel Debugging)
Serial Terminal
Disk
DAT

Fig. 3. The HP-RT development environment.

The items that come with the HP-RT development toolkit include:

- Libraries for building HP-RT kernels and user programs
- Include files for compiling user programs and I/O drivers for executing in an HP-RT operating environment
- Installation and user program compilation scripts
- A pair of source-level debuggers: one for user program debugging and one for I/O driver and kernel-level debugging.

The two remote debuggers included with the HP-RT development kit are derived from the standard xdb debugger product provided with the HP-UX operating system. The debugger used for user program debugging is capable of debugging multithreaded user processes and communicating with the target HP-RT system using a TCP (Transmission Control Protocol) virtual circuit socket. The kernel debugger is for kernel-level and I/O driver debugging and communicates with the target HP-RT system via a dedicated RS-232-C serial communication link. Using a dedicated communication link allows the kernel debugger to operate without interfering with the normal operation of the target operating system.

A set of user commands, a bootable kernel, and miscellaneous files are included with the HP-RT system. These items can be installed via LAN on a disk connected to the target system. The HP-RT kernel can also be booted across a LAN and commands and user programs can either reside in RAM memory (via a RAM disk facility) or be accessed across the network via NFS mount points. The command set on the HP-RT target system is oriented around run-time operations and system administration. Commands related to program development (such as cc and the rcs and sccs tools) are not supported and can only be used on the host.

HP-RT Hardware

The hardware that supports execution of the HP-RT operating system is the HP 9000 Model 742rt VMEbus board computer shown in Fig. 1. This system consumes consumes two slots of a VMEbus backplane. The system processing unit and onboard I/O features of the Model 742rt include:

- PA-RISC 7100 processor, which has a clock frequency of 50 MHz and is capable of executing 61 MIPS
- 8M bytes of ECC (error correction code) RAM for main memory, which can be upgraded to 64M bytes of ECC RAM (The ECC RAM comes in a pair of SIMMs and provides single-bit error correction and multiple-bit error detection.)
- 64K-byte external instruction cache and 64K-byte external data cache
- Onboard I/O ports for one SCSI II interface (up to seven devices), two serial RS-232-C interfaces, one parallel interface, and one Ethernet LAN interface
- VMEbus D64 interface, which provides an asynchronous, 32-bit data bus that is capable of transfer rates of up to 40 Mbytes/s.

The Real-Time Kernel

The HP-RT kernel and I/O drivers are designed for real-time response and determinism at a level never before accomplished in a Hewlett-Packard operating system product. The HP-RT kernel ensures that the highest-priority operations are serviced within 50 to 110 microseconds in the worst case and typically much faster depending on the specific operation. To accomplish this, the HP-RT kernel uses a fully reentrant and interruptable design and makes extensive use of full kernel support for threads for user and kernel processes.

Multithreaded Kernel

The fundamental unit of an executing task in HP-RT is the concept and structure of a thread. A thread contains a program counter (next instruction pointer) and a stack for recording local subroutine variables and calling sequence parameters. Threads do not own a specific address space or a specific set of code. Threads typically share address space (data area) and code with other threads. The concept of a process is simply a combination of a single thread, a code segment, and a data area (see Fig. 4a). HP-RT extends this concept by allowing a single process to create multiple threads (see Fig. 4b). These additional threads execute code in the same process code area and have identical access rights to all data areas in the process. See "An Overview of Threads," on page 27 for a brief tutorial on threads.

HP-RT also implements the concept of a kernel thread. A kernel thread is a thread of execution that only executes kernel code at a kernel privilege level. Kernel threads are used in HP-RT to provide kernel services asynchronously for any specific user process or thread with each service executing at a user-specified priority.

Reentrancy and Interruptability

The HP-RT kernel's general model is to execute on behalf of a thread of execution with interrupts enabled and context switching allowed. The specific thread executing may be a thread associated with a user process or a kernel thread. All threads, regardless of type, have their own user-specified priority, scheduling policy (time-sliced versus run-to-completion), and system level.

The system level is a specification of the mode in which a thread is executing. At system level zero, a thread runs in user mode, with user-level privileges. Kernel threads by definition never use this system level. At system level one, a thread executes kernel code with kernel-level privileges and with all interrupts enabled and context switching allowed. At system level two, a thread executes kernel code with context switching disabled, but interrupts enabled. Finally, at system level three, a thread executes kernel-level code with both context switching and interrupts disabled. Table I summarizes these system levels and execution modes.

Context switching and interrupt handling in HP-RT are described in more detail in the article on page 31.

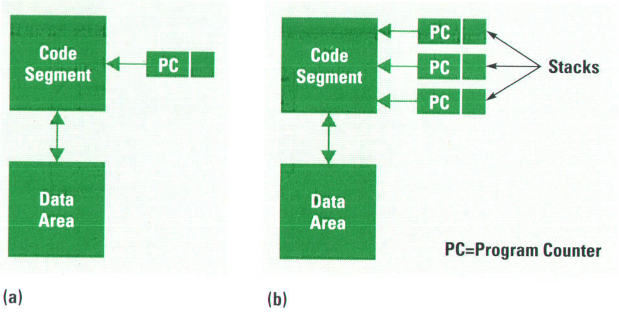


Fig. 4. Thread configurations. (a) A typical single-thread process. (b) A multiple-thread process.

Table I
System Levels and Execution Modes

System Level	Execution Mode	Context Switching	Interrupts
Zero	User	Allowed	Enabled
One	Kernel	Allowed	Enabled
Two	Kernel	Disallowed	Enabled
Three	Kernel	Disallowed	Disabled

The HP-RT system supports one nonthread mode of execution, which is based on execution using a single interrupt stack. However, unlike timesharing systems and many real-time systems, HP-RT makes very limited use of interrupt-stack-based execution because this mode of execution is always at a higher priority than thread execution. Execution using an interrupt stack means that a full thread context is not established, which means that a context switch to a thread cannot be allowed until the interrupt-stack-based execution is complete. Most interrupt service routines, such as the handlers for the SCSI bus and LAN interrupts, are instead handled by a specific kernel thread. These threads are scheduled when their corresponding interrupt occurs at their specific priority and are not executed until all higher-priority thread execution is complete.

Because of the general reentrancy of HP-RT, explicit calls are used in kernel code and I/O drivers for managing reentrancy.† The macros sdisable(), srestore(), disable(), and restore() are used to move a process to system levels two (context switch disabled) or three (both context switching and interrupts disabled) and back to the premove system level. Turning context switching off guarantees atomicity with respect to the execution of other threads. Turning off interrupts guarantees atomicity with respect to execution of both threads and interrupt-stack-based handlers.

Data structures used by the kernel are generally global to the entire kernel and nonreentrant operations must be properly protected. A simple example of this is the use_count field of the in-core inode†† data structure. The use_count field indicates the number of instances of a particular file that are active (e.g., open). When a new process accesses an inode, the equivalent of the code statement inode_ptr–>in_use++ (increment use_count) must be executed. On PA-RISC (and most RISC processors), this code translates to a sequence of instructions that loads the use_count value, increments it, and then stores the value to the memory location it came from. Interleaving such operations, which can easily happen because of a context switch from one thread to another, will cause the use_count to miss an increment, producing devastating long-term results.

For example, Fig. 5 shows what can happen when a thread is interrupted before finishing incrementing the use_count field for a particular inode. The use_count field is represented by the variable X, which is initially equal to one (i.e., some other thread or process is accessing the same file). At ⓐ Thread 1 begins executing the instructions to increment X, but just before storing the result in X, Thread 2 interrupts at ⓑ and the scheduler hands control over to Thread 2. Thread 2 increments the same use_count field. When Thread 2 is finished, X = 2 and the scheduler returns control back to Thread 1 at ⓒ. At ⓓ Thread 1 finishes its work on the use_count field by storing the value it computed before being interrupted into X. At this point X should be equal to three, but because Thread 1 was interrupted before it finished its critical section, X = 2.

The need for atomic increment and decrement operations is so pervasive in the HP-RT kernel that special macros called ATOMIC_INC() and ATOMIC_DEC() are used. These macros generate inline assembly code that disables interrupts, performs the increment or decrement operation, and reenables interrupts.

Use of an interrupt disable versus a context switch disable is a key design decision for every critical section of HP-RT kernel code. The main question asked in arriving at a decision is whether the operation is critical relative to execution of code that can run on the interrupt stack. Since very little code in HP-RT executes on the interrupt stack, a context switch disable usually suffices for protection. However, a context switch disable is a more expensive operation than an interrupt disable operation. A context switch requires memory access and an interrupt disable only requires execution of an inline assembly statement which turns off the interrupt enable bit in the PA-RISC processor status word. Thus, very short operations are better protected with interrupt disables.

This raises the question of how HP-RT solves the problem of long critical sections for which a context switch or an interrupt disable last too long. In the analysis of customer requirements and competitive systems, it was determined that context switch off times should be held to as close to 100 microseconds as possible, and ideally less, and interrupt disables should be held as close to 50 microseconds as possible, and ideally less. Longer critical sections are managed using kernel-level semaphores.

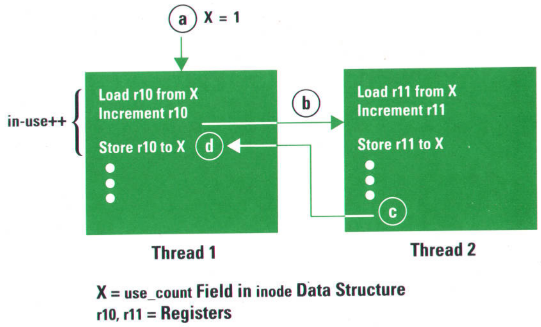


Fig. 5. What can happen when a thread is context switched in the middle of a critical operation. Thread 1 is interrupted and context switched just before it is about to increment the use_count value. As a result, when Thread 1 is finally able to finish its operation, the wrong value is stored in use_count.

† A reentrant process consists of logically separate code and data segments and a private stack. Multiple instances of a reentrant process can share the same code segment but each instance has its own data segment and stack.

†† An inode is the internal representation of a file in a UNIX*-system-based operating system. An in-core inode is one that resides in main memory.

An Overview of Threads

When a process is running it executes a sequence of instructions stored in its address space in memory. This execution of a sequence of instructions is called a thread of execution, or simply a thread. The execution of a thread requires that it have its own program counter to point to the next instruction in the sequence, some registers to hold variables, and a stack to keep track of local variables and procedure call information. Although threads have some of the same characteristics as a regular process, they are sometimes called a "lightweight" process because they don't carry around the overhead (or extra weight) of regular processes. Table I lists some typical items associated with each thread and each process.

Fig. 1 models processes and threads running in a computer. The processes in Fig. 1a have one thread of execution each. They also have their own address spaces making them independent of each other. To communicate with each other (for example, to share resources) they must do so through the system's interprocess communication primitives, such as semaphores, monitors, or messages. In Fig. 1b the three threads are in one process. Thus they share the same address space and have access to all the per-process items listed in Table I.

One of the reasons threads were invented was to provide a degree of quasiparallel execution to be combined with sequential execution and blocking system calls. For example, consider a file server that must block occasionally to wait for the disk. In a single-process situation the server would get a request and service it to completion before moving on to the next request. Thus, no other requests would be serviced while the server is waiting on the disk. If the machine is a dedicated file server, the CPU is also idle while the server process is waiting on the disk.

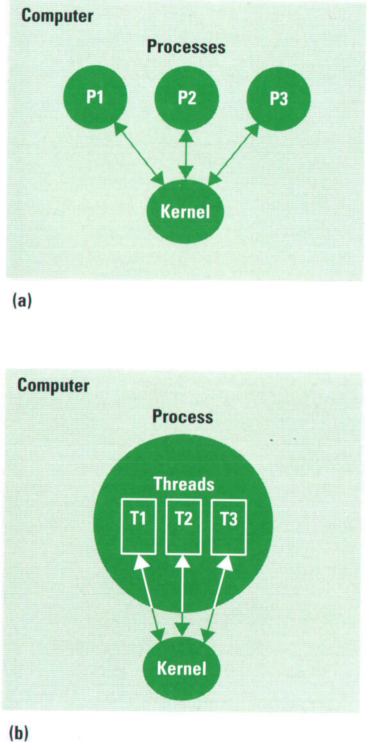


Fig. 1. Models of processes and threads running in a computer. (a) Multiple processes. (b) Multiple threads in one process.

Table I
Items Associated with Threads and Processes

Per-Thread Items*	Per-Process Items
Program counter	Address space
Stack	Global variables
Registers	Files
	Child processes
	Signals
	Semaphores

* All per-thread items are also per-process items.

If the server is a multithreaded process, one thread could be responsible for reading and examining incoming requests and then passing the request to a thread that will do the work. When a thread must block waiting on the disk, the scheduling thread can get another request and invoke another thread to run. The result of using threads in this case would be higher throughput because the CPU would not sit idle, and better performance because it is much faster to switch threads than to switch processes.

In a real-time system where a quick response to interrupts and other events is critical, threads offer some definite advantages, especially if one considers context switching between processes versus switching between threads. Table II summarizes some of the main differences between threads and processes.

Table II
Differences between Threads and Processes

Processes	Threads
Program-sized	Function-sized
Context switch may be slower	Context switch may be faster
Difficult to share data	Easy to share data
Owns resources such as files and memory	Owns stack space and registers only

Bibliography

1. T. Anderson, et al, "The Performance of Thread Management Alternative for Shared-Memory Multiprocessors," *IEEE Transactions on Computers,* Vol. 38, no. 12, December 1989, pp. 1631-1644.

2. A. S. Tanenbaum, *Modern Operating Systems,* Prentice-Hall, 1992, pp. 507-523.

3. P. Dasgupta, et al, "The Clouds Distributed Operating System," *IEEE Computer,* Vol. 24, no. 11, November 1991, pp. 34-44.

4. R. Lafore and P. Norton, *Peter Norton's Inside OS/2,* Simon & Schuster, Inc., 1988, pp. 134-174.

Kernel Semaphores and Priority Inheritance

An example of an extended critical section is the manipulation of an in-core inode. Critical inode operations such as the addition of a file to the directory data of a directory inode must be performed atomically. Each inode holds a semaphore which is locked and unlocked around these critical operations.

The HP-RT kernel uses the simple semaphore primitives swait() and ssignal() (corresponding to Dijkstra's P and V operations)[3] for process synchronization, mutual exclusion, and atomic resource management. A single 32-bit integer is used as a kernel semaphore data structure. This data structure supports two semaphore types: counting semaphores and priority-inheritance semaphores. With an additional

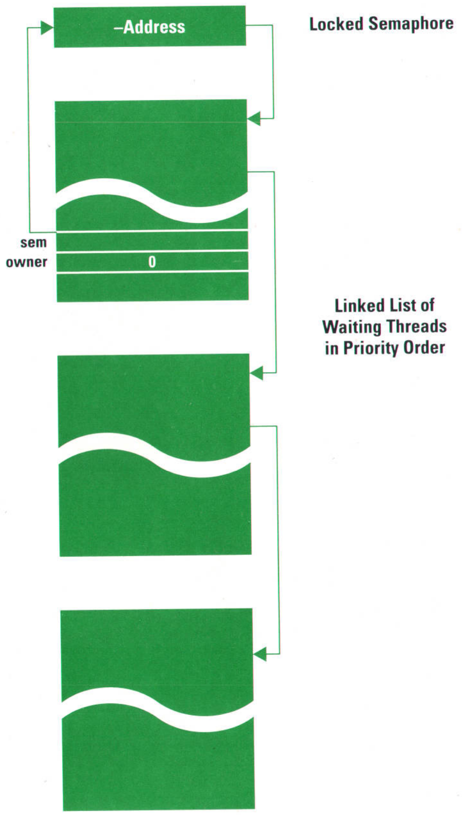


Fig. 6. A locked counting semaphore and waiting threads.

level of lock and unlock code and using a separate integer as a counter, priority-inheritance semaphores can also be used as the basis for counting semaphores. Priority-inheritance semaphores are described later in this paper.

The semaphore primitives ssignal and swait have the code to interpret the contents of the kernel semaphore data structure and are able to differentiate between counting and priority-inheritance semaphores.

A counting semaphore in HP-RT holds a positive count value when the semaphore is unlocked and a resource is available. An swait() operation on a positive-valued semaphore causes the semaphore to be atomically decremented, and the calling thread continues execution. An swait() on a zero or negative-valued semaphore (the resource is not available) causes the thread to block (suspend) on the semaphore.

When one or more threads are blocked on a counting semaphore, the threads are placed into a priority-ordered linked list with the semaphore heading the list. To identify a semaphore that is locked and has one or more waiting threads, the semaphore is set to the negative address of the first waiting thread (see Fig. 6). The sem and owner fields shown in Fig. 6 are described below.

An ssignal() on an unlocked or locked-with-no-waiters counting semaphore merely causes the nonnegative value of the semaphore to be atomically incremented. An ssignal() on a locked semaphore with one or more waiters (one that holds a negative thread structure address) causes the first (highest-priority) waiting process to be unlinked and scheduled. Table II summarizes the different states of HP-RT counting semaphores.

Table II
Different States of Counting Semaphores

State	Meaning
0	Locked with no waiters
–Address	Locked with waiters (The address points to the first thread in the list of waiting threads.)
≥1	Unlocked

One drawback of this semaphore methodology is that there is no clear ownership of a locked semaphore. The second drawback is the risk of priority inversion.

Priority Inversion

In most real-time operating systems, a priority-driven preemptive scheduling approach is used. This scheduling method works well when a higher-priority process (or thread) can preempt a lower-priority process with no delays. One important problem that sometimes hampers the effectiveness of this scheduling algorithm is the problem of blocking caused by the synchronization of processes that share physical or logical resources.

The most common situation occurs when two processes attempt to access shared data. In a normal situation, if the higher-priority process gains access to the resource first, then good priority order is maintained. However, if a higher-priority process tries to gain access to a shared resource after a lower-priority process has already gained access to the resource, then a priority inversion condition takes place because the higher-priority process is required to wait for a lower-priority process to complete.

The following example, which is loosely based on an example first described by Lampson and Redell,[4] shows how a priority inversion can occur. Although the term process is used in the following example, the executing entity could just as well be a thread.

Let P1, P2, and P3 be three processes arranged in descending order of priority. Let processes P1 and P3 share a common data structure which is guarded by the binary semaphore X. Fig. 7 and the following sequence shows the events that can lead to a priority inversion:

1. P3 locks X and enters its critical section.

2. P1 arrives, preempts P3 and begins its processing.

3. P1 tries to lock X, but because X belongs to P3, P1 is blocked.

4. P3 again attempts to finish its critical section.

5. P2 arrives and preempts P3 before it finishes its critical section.

6. Assuming there are no more preemptions at some point P2 finishes, then P3 finishes, and P1 finally is unblocked on resource X and allowed to finish its critical section.

In this scenario the duration of P1's blocking is unpredictable because other processes can show up before P3 finishes its critical section and is able to release X.

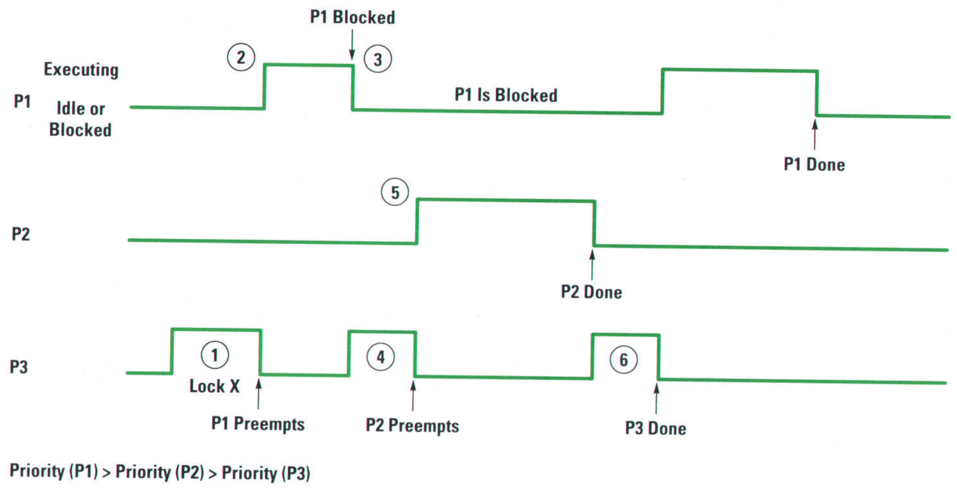


Fig. 7. A time line illustrating priority inversion.

Priority Inheritance

The methodology used in HP-RT to avoid the priority inversion problem employs priority-inheritance semaphores. The basic concept of priority-inheritance semaphores is that when process P blocks a higher-priority process, it executes its critical section at the highest priority level of all of the blocked jobs. Process P returns to its original priority level when it completes its critical section, which then allows the highest-priority blocked process to execute.

From the example above if P1 is blocked by P3 then according to the priority-inheritance concept, P3 inherits the same priority as P1 while it executes in its critical section. When process P2 arrives (while P3 is in its critical section) it would not be able to preempt process P3 because P3 would be running at a higher priority than P2. Thus, process P2 will not begin execution. When P3 finishes its critical section, process P1 can preempt P3 and run to completion. Then process P2 can begin execution.

Priority-inheritance semaphores can become quite complex when nested semaphore locks are allowed as they are in the HP-RT kernel. Not only must the current owner and all waiters for a semaphore be known, but given the owner of a particular semaphore, the highest-priority waiters of all semaphores currently owned by that owner must be known. This allows the system to manipulate priority properly as semaphores are released. The priority must revert to the priority of the current highest-priority waiter of all still-owned semaphores.

To manage this complexity and yet retain a single interface and data structure for semaphore operations, HP-RT uses the semaphore value –1 to indicate unlocked for a priority-inheritance semaphore. A value of one is not a possible thread structure address, so this value cannot be confused with the negative address of the first waiter of a counting semaphore.

Two fields in the thread structure are used to differentiate between the various states of priority-inheritance and counting semaphores when they are locked. A counting semaphore that is locked and has waiters will have the sem field in the first waiter thread holding the address of the semaphore and an owner field containing zero (see Fig. 6). A priority-inheritance semaphore that is locked and has no waiters will hold the negative address of the owner thread, which has a sem field with a value of zero (see Fig. 8a). Lastly, a locked priority-inheritance semaphore that has waiters will hold the negative address of the highest-priority waiting thread. This thread structure has a sem field holding the address of the semaphore and an owner field holding the address of the owning thread (see Fig. 8b).

To keep track of the highest-priority waiters for all owned priority-inheritance semaphores, a doubly linked list containing the highest-priority waiters for each owned semaphore is attached to the thread structure of each semaphore owner.

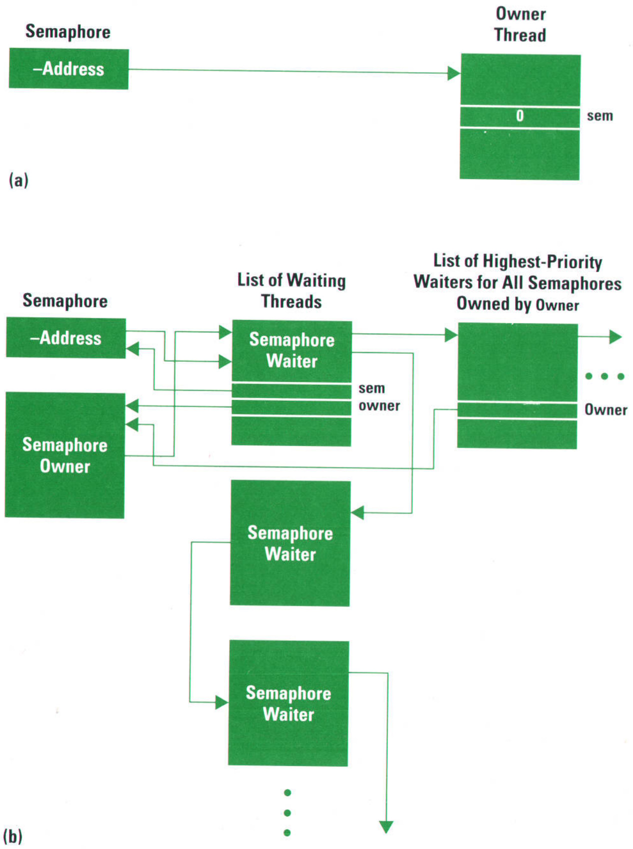


Fig. 8. Data structures associated with priority-inheritance semaphores. (a) A locked semaphore with no waiting threads. (b) A locked semaphore with waiting threads.

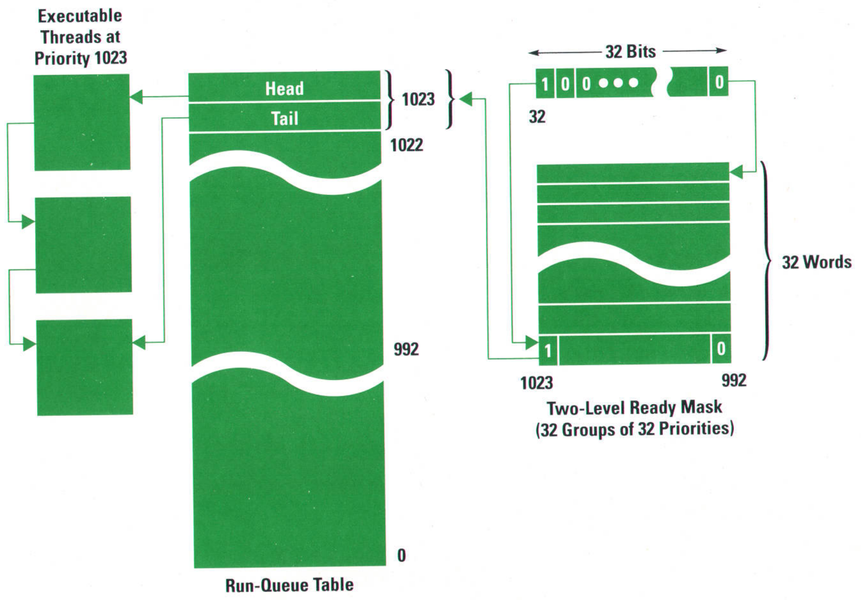

Executable Threads at Priority 1023
Head
Tail
1023
1022
992
0
Run-Queue Table
32 Bits
1 0 0 ••• 0
32
32 Words
1 0
1023
992
Two-Level Ready Mask (32 Groups of 32 Priorities)

Fig. 9. Data structures for process scheduling in HP-RT.

The different states of priority-inheritance semaphores are summarized in Table III.

Table III
Different States of Priority-Inheritance Semaphores

State	Meaning
–1	Unlocked
–Address of thread owner	Locked without waiters (sem field in thread owner = 0)
–Address of highest-priority waiting thread (sem field in highest-priority waiting thread = semaphore address and owner field = thread owner address)	Locked with waiters

Process Scheduling

HP-RT currently uses 64 distinct priority levels with the ability to extend support to 1024 distinct priority levels. Half of all HP-RT priorities are reserved for use by kernel management software. There is no explicit user program interface provided for placing a priority at these reserved levels. The reserved priorities are interleaved with the user priorities and are considered a "priority boost" on a user priority. Thus, between any two user priorities N and N + 1 lies a priority N + boost, which is more important than priority N and less important than priority N + 1. Boosted priorities are used by kernel service threads to provide service just above the priority of the highest-priority requesting process, but not at the next highest user priority which may be in use by the system user. Priority boosting is also used for temporary elevation of the priority of processes blocking on I/O operations to maximize throughput. This type of algorithm is only used in a user-specified portion of the overall priority range.

The HP-RT kernel internally manages priorities by converting from the user priority plus a possible boost value to a run queue table index by using the formula:

$$\text{Internal Priority} = (\text{user priority}) \times 2 + \text{boost},$$

where boost is either zero or one. Hence, if user priorities range from zero to 127, the internal priorities range from zero to 255.

HP-RT maintains a run-queue table with one entry per internal priority. Each entry holds a ready thread list head and a list tail pointer (see Fig. 9). To determine quickly the highest priority for which there is a runnable thread, HP-RT uses a two-level bit mask called a ready mask in which a set bit indicates a runnable thread. The top level of the ready mask is 32-bit word. Each bit in this word indicates that within a set of 32 priorities, at least one thread is executable. Thus, if as shown in Fig. 9 the high-order bit of the first word of the ready mask is set, then there is at least one thread in the internal priority range of 1023 to 992 that is executable. The second level of the ready mask holds up to 32 32-bit entries each of which indicates which of these 32 priorities holds executable threads.

By using high-speed assembly language code to find the first set bit in the ready mask, the highest-priority thread in the nonempty run queue can be quickly determined.

References

1. B. Clements, "Mechanical Considerations for an Industrial Workstation," *Hewlett-Packard Journal*, this issue, p. 62.
2. *Hewlett-Packard Journal*, Vol. 41, no. 3, pp. 36-58.
3. E. W. Dijkstra, "Co-operating Sequential Processes," *Programming Languages*, F. Genuys, Editor, London: Academic Press, 1965.
4. B. W. Lampson and D. D. Redell, "Experiences with Processes and Monitors in Mesa," *Communications of the ACM*, Vol. 23, no. 2, February 1980, pp. 104-117.

HP-UX is based on and is compatible with UNIX System Laboratories' UNIX* operating system. It also complies with X/Open's* XPG3, POSIX 1003.1 and SVID2 interface specifications.

UNIX is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and other countries.

X/Open is a trademark of X/Open Company Limited in the UK and other countries.

Managing PA-RISC Machines for Real-Time Systems

In the HP-RT operating system, the interrupt-handling architecture is especially constructed to manage the high-performance timing requirements of real-time systems.

by George A. Anzinger

The task of an operating system is to manage the computer system's resources. This management should be done so as to give the best possible performance to user tasks or jobs presented to the system. How this performance is measured and valued differs depending on the task or mission of the system. The three major classes of tasks or missions presented to an operating system are timeshare, batch, and real time. The important aspects of performance of these three classifications differ, and, because they differ, require the operating system to use different algorithms to manage system resources.

Timeshare

Timeshare systems are usually designed to share system resources with all contending processes. The major resource to be shared is CPU time, which is usually sliced into small units (called time slices) and allocated to all runnable processes in a "fair" way. Various notions of fair exist and have been used, but in general, runnable processes contend at the same level or priority for CPU time. Some (or even most) systems modify this notion of fair to give more time to a process that blocks often and less to a process that is compute bound. Some systems may also have preferred priorities for processes that run on behalf of the system. Such processes may be handling printers, communication lines, or other things that are shared with several processes.

Batch

Batch systems are usually designed to maximize the throughput of the system. That is to say, they attempt to get the most work done in a given period of time. Such systems will not usually use a timeshare scheduling algorithm because it introduces overhead that does not add to the desired result—throughput. To help achieve maximum throughput, one popular batch scheduling algorithm is to run the job that has the least amount of time left to run. The point is that batch systems typically do not need to make any attempt to share CPU time.

Real Time

Real-time systems, unlike timeshare or batch systems, are usually designed to run the most important process that is ready. Importance is assigned by the user or designer of the system, and the operating system has little or nothing to say about it. The system designer (i.e., the user who sets up the system) decides the order of process importance and assigns priorities for all processes on the system. The operating system's job then is very simple: give the CPU to the highest-priority process that is ready. The performance of a real-time system is usually measured by how fast it can respond to events that change the identity of the highest-priority ready process. Such events are usually external and come to the system in the form of interrupts, but can also be internal in the form of processes that promote other processes to higher priorities (or demote themselves to lower priorities). Another major event that real-time systems must respond to is the passage of time. The indication of the passage of time also comes to the system in the form of an external interrupt.

From this discussion, it is apparent that one major measure of a real-time system is how quickly it can respond to an interrupt. A response consists of:

- Recognizing that the interrupt is pending
- Processing the interrupt (i.e., deciding what to do about it)
- Taking the indicated action.

Usually the indicated action will be to switch context to the process that is to handle the interrupt. Context switching encompasses the actions taken when control or execution moves from one process to another as a result of an interrupt or some other event (see "Context Switching in HP-RT" on page 32 for more about context switching).

From a system's point of view the response (or response time) is the time it takes the whole system† to do something that changes the environment it is monitoring or controlling. From an operating system vendor's point of view the response stops when the user code gets control and the operating system's responsiveness is no longer key to system performance.

While the system is dealing with one interrupt and preparing a response, it may need to contend with other interrupts that are less urgent. The system must take the time to determine this.

It is also possible that, at the time an interrupt arrives, the system is in a state in which the interrupt system or context switching is off. The system needs to go into these states to protect shared data from corruption by contending processes (see "Protecting Shared Data Structures," on page 33). Some systems protect themselves and their shared data by turning off context switching whenever they are in system code.

† This includes the operating system, the user application, and the external devices.

Context Switching in HP-RT

Context switching can be defined as moving abruptly from one area of code to another as the direct result of some influence outside of the program or programs being switched to or from. Usually the context switch is the direct result of an interrupt or trap (a trap is an internal interrupt caused by some program activity such as divide by zero or illegal memory access). A context switch can also occur as a result of a program or thread blocking. In this case the operating system will context switch to a program or thread that is not blocked. These two different ways of generating a context switch have different overhead costs as will be explained below. One of the measures of a real-time system is how fast it context switches. When used in this way the reference is to how fast one user process can be suspended and another user process restarted.

To context switch, the operating system must save the **from** process's state. The state consists of all the machine registers that the program may depend on. After saving the **from** process state, the **to** process's state must be restored. As a result of this save and restore, both the **to** and **from** processes have their view of the world preserved and restored respectively even if they are suspended for a very long time.

For example, consider the case of a user program that has asked for some device input. The program will be suspended or blocked on the device driver waiting for the device to respond with the desired data. While waiting, the operating system will find some other program that is ready to run and switch to it. When the desired data arrives, the processor will be interrupted and the operating system will switch control of the processor to the waiting program.

As an example of a context switch that is strictly the result of an external interrupt, consider the case in which a time slice is exhausted. In this case, both the program being switched from and the one being switched to are interrupted as opposed to having to block and wait for a resource.

From a system overhead point of view there are four different types of context switch:

- Both the **from** and the **to** processes enter the blocked state programmatically
- The **from** process blocks programmatically and the **to** process is interrupted
- The **from** process is interrupted and the **to** process is blocked programmatically
- Both processes are interrupted.

Because of calling sequence conventions, processes that are interrupted incur additional overhead to save and restore caller registers.

To take advantage of the savings possible when processes block programmatically, HP-RT uses a context switch routine based on this type of block. The extra work required when processes are interrupted is performed by code in the system interrupt handler.

This is not reasonable for a high-performance real-time system that is trying to switch contexts in less than 50 µs. For these systems it is necessary to recognize and process interrupts in the 25-µs range. This implies that the interrupt off time plus the interrupt processing time must be kept below 25 µs.

This paper will explore the problems a PA-RISC architecture presents to real-time processing. These problems revolve around the need for fast context switching, interrupt handling, and repeatability. Next, possible solutions to these problems will be discussed, detailing the solutions used in the HP-RT (real-time) operating system, which runs on the HP 9000 Model 742rt VMEbus board computer. The hardware and software components of the Model 742rt are described in the article on page 23.

PA-RISC Architecture

The RISC architecture is used to speed up CPUs by designing them so that each instruction is simple and can be executed quickly. The goal is usually to have each instruction take the same amount of time to execute and to design the machine so that several instructions can be pipelined. To get all instructions to execute in the same time requires that no one instruction can be complex. Operations that are complex and require more than one instruction time are either handled by subroutines or by coprocessors. Coprocessors are designed to run independently allowing the main processor to do other useful work while the coprocessor does its work. For example, HP's PA-RISC machines use coprocessors to do floating-point math.

In HP's PA-RISC processors, the following characteristics are important for real-time applications:

- Memory reference instructions either load or store and do nothing else. This means that there is no read-modify-write instruction.
- Memory reference instructions may stall if the data is not available. To help in this regard, a cache memory is used to speed up the average access to memory.
- Since memory accesses are potential roadblocks, 32 general-purpose registers are available as well as 27 control registers and 32 64-bit-wide floating-point registers. This allows the processor to keep most of the variables of interest in registers, avoiding slow memory access operations.
- All interrupt context is kept in control registers.

Real Time and HP's PA-RISC

From a real-time perspective, the characteristics of HP's PA-RISC that are of concern are those that limit performance in the real-time sense. As discussed above, a real-time system must be able to change its mind (context switch) quickly. This implies that the large context associated with a process can be a problem. Also, while changing context, as well as doing other things, the system needs to be even more responsive to interrupts. This means we must not turn the interrupt system off for long times. In particular, we must not turn it off for the duration of a context switch.

HP-RT is the result of porting a third-party operating system† to the HP 9000 Model 742rt board level real-time computer.

As such, the porting team was constrained to work with the conventions existing in the system being ported. Likewise, the porting team was not empowered to change any of the language or hardware conventions that exist in HP's PA-RISC machines and the HP-UX* host operating system.

To take advantage of the best of HP's PA-RISC processors, the port team decided to restrict the system to PA-RISC 1.1 architectures. The 1.1 architecture provides shadow registers that allow system interrupt code to be run without saving any context (see "The Shadow Register Environment," on page 34).

On examining the way the system we were porting recommends that drivers be written we found the following:

† LynxOS from Lynx Real-Time Systems Inc.

- After an interrupt, the system enters the interrupt service routine. The routine should be written in C and should make a call to the operating system function ssignal and then return.
- The function ssignal increments a counting semaphore, and if the result is 0, the interrupt service thread is put in the ready list (execution threads and counting semaphores used in the HP-RT operating system are described in the article on page 23).
- If the new entry in the ready list has a higher priority than the current process, a flag is set indicating that a context switch is needed. (Context cannot be switched while in an interrupt handler.)
- When the driver's interrupt service routine returns, the system notices whether a context switch is pending and if so takes the required action. If not, the system just returns to the point of the interrupt.

The problem with this picture is that to call the interrupt service routine the system has to save most of the system state. This is a lot of overhead for only one function call and return.

The team decided that a better way to handle interrupt servicing would be to code a companion ssignal function. The new ssignal runs using only the shadow registers and still does everything the original ssignal did. This scheme allows the whole ssignal call to be made without establishing a C context, which involves saving and restoring the C environment (see "C Environment," on page 35). However, some restrictions are placed on I/O drivers in that they have to make their semaphores known to the operating system.

In some cases, calling the ssignal function is almost all that an interrupt service routine will do. It is also possible that a few lines of assembly code might be required to complete the interrupt service routine. Such code might move a byte of incoming data from the I/O device to an internal buffer. For applications that have these kinds of interrupts, the system provides the ability to call an assembly language interrupt service routine. To keep overhead low, the assembly language interrupt routine is restricted to using the shadow registers and no system resources. The system interrupt dispatcher calls the ssignal function if the assembly language routine returns a nonzero semaphore.

Some I/O devices and drivers require full C-code interrupt handlers. For these interrupts, the system establishes a C context on an interrupt control stack. In this context interrupts of higher priority are turned on while the interrupt is processed. These routines can also call a limited number of system functions. For example, the system time base interrupt is handled by a C interrupt handler.

With three different possible interrupt handling situations, the operating system needs to have the ability to decide quickly which interrupt service routine to use. Usually this is done by either a table index, in which the system determines the method to use via a number that is an index into a table of routines to call, or a case statement, in which the indicated method, again expressed as a number, is used to indicate which code to execute. A much quicker method than these two is to put the address of the interrupt service routine in the driver's table structure. This also allows the system to be expanded easily to handle other interrupt handler environments.

Protecting Shared Data Structures

Shared data structures are needed in any operating system to keep track of the resources that the system is sharing among several processes. For example, each process will need memory for its code and data. This memory is a shared resource and the management structures must be accessed in a way that will not allow the system to lose parts of the resource. One method of keeping track of a resource like memory is to keep free pages of memory in a free list. When a page of memory is needed, the page at the head of the free list is removed from the list and given to the requesting process. This removal (and its subsequent return) must be done in an atomic operation with respect to the contending processes. By this we mean that, as far as any process cares, the removal of a page from the free list happens as one indivisible operation. Otherwise, a contending process could get control and possibly get the same page.

The importance of maintaining atomicity in dealing with a shared resource such as memory on a free list is illustrated in the following example. The process of removing page A from the free list involves:

1. Picking up the pointer to page A from the list head

2. Using the resulting pointer to get the pointer to page B, which is in the first word of page A

3. Storing the pointer to page B in the list head.

If the removal is interrupted after step 1 but before step 3, and the interrupting process also tries to remove a page from the free list, both processes will get the same page and most likely the system will fail. Similar problems on returning of pages to the free list can result in lost pages or even circular free lists.

The solution to these problems is to make a sensitive operation atomic with respect to contenders. If only processes can contend, it is sufficient to prevent context switches for these periods of time. If one or more of the contenders runs on an interrupt, then interrupts must be disabled to achieve the required atomic operation.

The HP-RT system supports three levels of contention protection:

- Interrupts disabled
- Context switch disabled
- Semaphore locking.

From an overhead point of view, the cost is lowest for the interrupt disable and highest for the semaphore lock. From an impact on performance point of view, interrupts should be disabled only for short periods of time, context switch disabled only for slightly longer times, and semaphores held as long as needed.

For short operations, such as the list removal operation described above, the interrupt disable method is the best to use (even if the atomic test does not require this level of protection) because the disable time is short and the overhead of interrupt disable protection is the lowest of the three methods.

A New Interrupt Environment

The need to deal with the three interrupt handling situations described above and the requirement to handle interrupts from the VMEbus meant that we had to design and implement a new interrupt handling environment. Fig. 1 shows a simplified view of the logical I/O architecture that the HP-RT interrupt handling subsystem is designed to service.

The nature of the VMEbus requires a second level of interrupt dispatch. This is necessary because VMEbus interrupts come into the PA-RISC processor via one of seven lines or PA-RISC interrupt levels. As shown in Fig. 1, each of these lines can handle several independent devices, which implies several interrupts.

The VMEbus standard specifies that a device requesting an interrupt must assert its request on the interrupt line it is

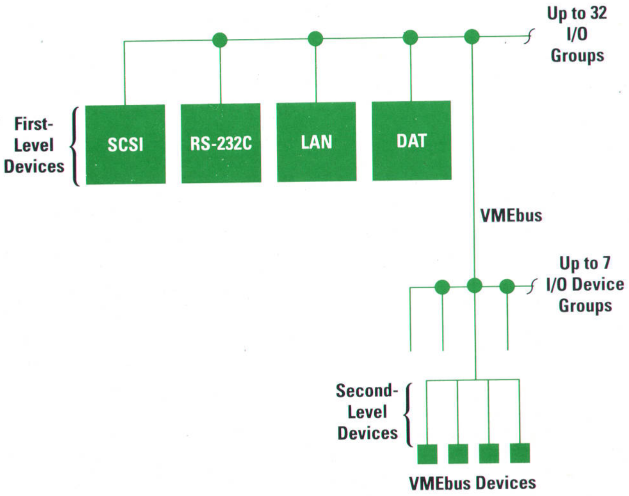
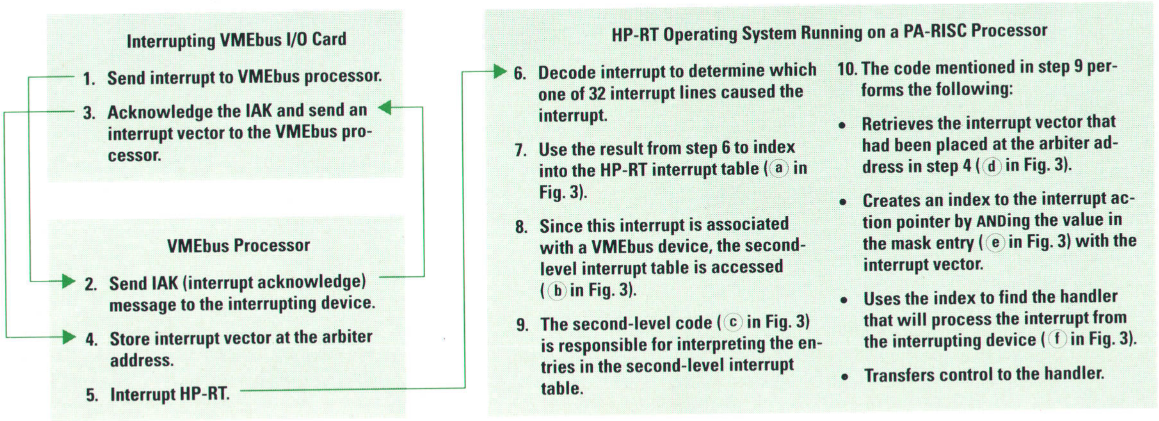


Fig. 1. A logical view of the I/O architecture the HP-RT operating system is designed to work with.

using. The interrupt responder sees the request and sends back an interrupt acknowledgment for that interrupt line. Each device using the same line blocks the acknowledgment signal from being seen by devices farther away from slot 0† while it has an interrupt request pending. When a device with an interrupt pending sees an interrupt acknowledge it responds by sending back an interrupt vector. The interrupt vector is a data element (byte or word) that identifies the interrupting device and is used by the interrupt responder to dispatch the interrupt.

The original plan for the Model 742rt hardware was to interrupt the PA-RISC processor when a VMEbus interrupt request was asserted and to do the interrupt acknowledgment when the processor attempted to read the interrupt vector. This plan required the operating system to stall in the interrupt handler with the interrupt system off for an unspecified length of time because VMEbus devices are not required to yield the bus to a requester, making the actual time required to do an operation on the bus open-ended. To solve this problem, the HP-RT team decided that the interrupt vector should be prefetched by the hardware before interrupting the PA-RISC processor. In this way a VMEbus interrupt can be dispatched without the PA-RISC processor having to wait for the VMEbus processor to fetch the interrupt vector. The current hardware always does the interrupt acknowledge as soon as possible but has the option of asserting the processor interrupt either immediately or on completion of the interrupt acknowledgment.

Fig. 2 shows the steps involved in handling a VMEbus interrupt and Fig. 3 shows a portion of the system interrupt table which is used for handling second-level VMEbus interrupts



Interrupting VMEbus I/O Card

1. Send interrupt to VMEbus processor.
3. Acknowledge the IAK and send an interrupt vector to the VMEbus processor.

VMEbus Processor

2. Send IAK (interrupt acknowledge) message to the interrupting device.
4. Store interrupt vector at the arbiter address.
5. Interrupt HP-RT.

HP-RT Operating System Running on a PA-RISC Processor

6. Decode interrupt to determine which one of 32 interrupt lines caused the interrupt.
7. Use the result from step 6 to index into the HP-RT interrupt table (ⓐ in Fig. 3).
8. Since this interrupt is associated with a VMEbus device, the second-level interrupt table is accessed (ⓑ in Fig. 3).
9. The second-level code (ⓒ in Fig. 3) is responsible for interpreting the entries in the second-level interrupt table.
10. The code mentioned in step 9 performs the following:
    - Retrieves the interrupt vector that had been placed at the arbiter address in step 4 (ⓓ in Fig. 3).
    - Creates an index to the interrupt action pointer by ANDing the value in the mask entry (ⓔ in Fig. 3) with the interrupt vector.
    - Uses the index to find the handler that will process the interrupt from the interrupting device (ⓕ in Fig. 3).
    - Transfers control to the handler.

Fig. 2. An example of the VMEbus interrupt handling process.

The Shadow Register Environment

The PA-RISC 1.1 implementation added shadow registers to the basic machine architecture. Shadow registers are seven registers into which the contents of GRs (general registers) 1, 8, 9, 16, 17, 24, and 25 are copied upon interruption. The contents of these general registers are restored from their shadow registers when an RFIR (return from interruption and restore) instruction is executed.

The shadow register environment includes code that executes between a processor interrupt and the following RFIR instruction. This code is executed in HP-RT using only the shadow registers. It is important to note that the nature of this environment is further defined by the nature of the processor's behavior on interrupt. When an interrupt occurs the processor transfers control to the interrupt code with the following state:

- Interrupt system off
- Interrupt state collection disabled
- Virtual memory system (both code and data) off
- All access protection off.

Since the virtual memory system is off, all memory for both code and data must reside in and be accessed by physical addresses. Usually an operating system will put the interrupt handling code in an area of memory that is "equivalently mapped." This means that the physical and virtual addresses are the same. This also means that code running in the shadow register environment cannot access memory with virtual addresses that are not equivalent since to do so would require the hardware to map the address using its TLB (translation lookaside buffer).† The hazard here is that the required entry may not be in the TLB, which would cause a trap to the TLB miss handler. Since traps are a form of interrupt, the miss handler would not be provided with the interrupt state (because the interrupt state collection is disabled) and thus would not know how to return to the point of the trap.

On the plus side, if the whole interrupt can be processed in the shadow register environment, the RFIR instruction is all that is needed to return to the point of interruption.

† The translation lookaside buffer or TLB is a hardware address translation table. The TLB speeds up virtual-to-real address translations by acting as a cache for recent translations.

† Slot 0 in a VMEbus cardcage typically houses the card or cards that contain the VMEbus system controller and other resources.

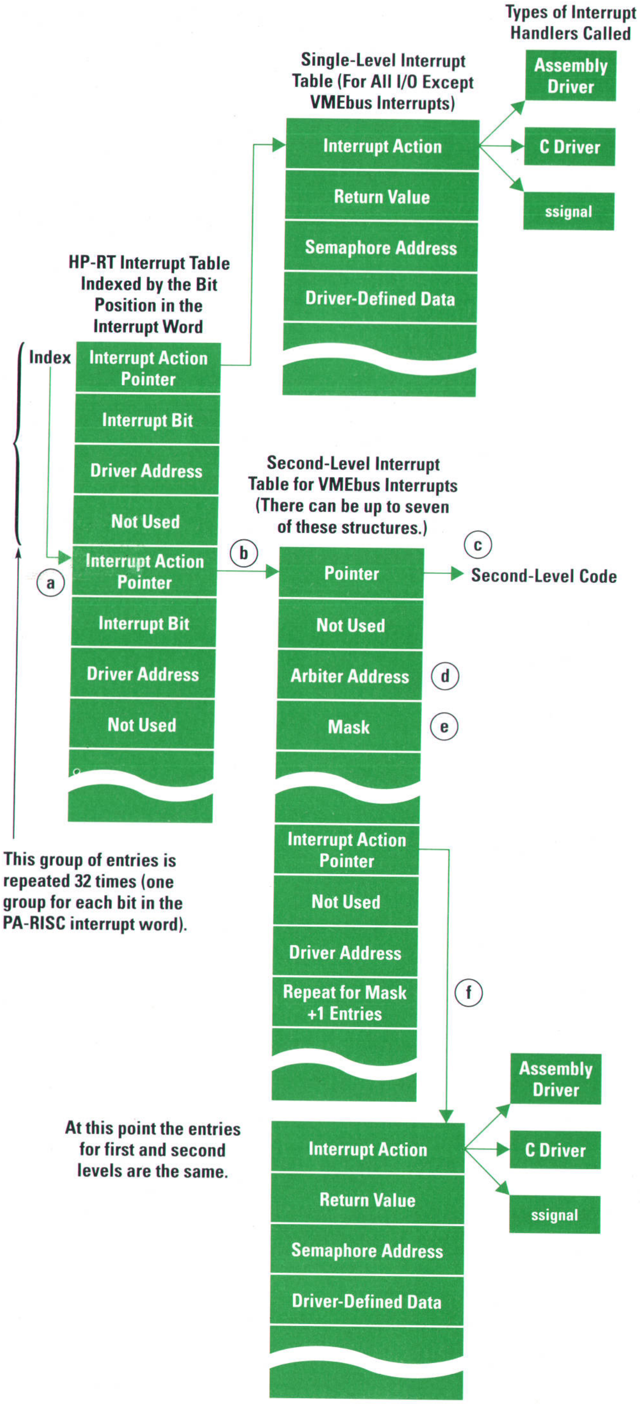

Types of Interrupt Handlers Called
Single-Level Interrupt Table (For All I/O Except VMEbus Interrupts)
Assembly Driver
Interrupt Action
C Driver
Return Value
ssignal
Semaphore Address
Driver-Defined Data
HP-RT Interrupt Table Indexed by the Bit Position in the Interrupt Word
Index
Interrupt Action Pointer
Interrupt Bit
Driver Address
Not Used
Second-Level Interrupt Table for VMEbus Interrupts (There can be up to seven of these structures.)
a
Interrupt Action Pointer
b
Pointer
c
Second-Level Code
Interrupt Bit
Not Used
Driver Address
Arbiter Address
d
Not Used
Mask
e
This group of entries is repeated 32 times (one group for each bit in the PA-RISC interrupt word).
Interrupt Action Pointer
Not Used
Driver Address
Repeat for Mask +1 Entries
f
Assembly Driver
At this point the entries for first and second levels are the same.
Interrupt Action
C Driver
Return Value
ssignal
Semaphore Address
Driver-Defined Data

Fig. 3. The HP-RT interrupt table structure.

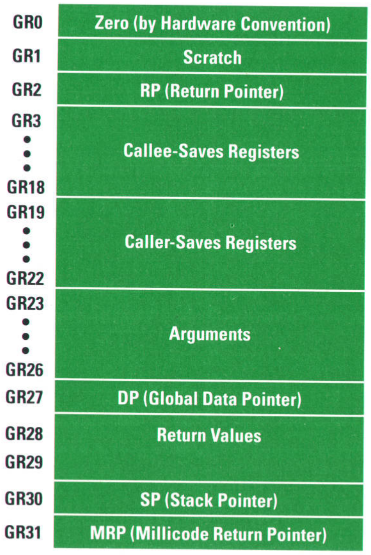
C Environment

C environment refers to the implied machine state when executing in a C language program. This machine state is really a set of register use conventions that are defined in the software architecture for the PA-RISC processors (see Fig. 1). Some of the basic assumptions made in C about these registers include:

- Register 30 is the stack pointer and points at the first available double word on the stack. The stack grows with increasing addresses.
- Just below the current stack pointer is a standard stack frame with room for the return address to be saved (if the callee needs to save it) and room for each of the call parameters to be saved.
- Registers 26, 25, 24, and 23 (as needed) contain the call arguments. If more than four arguments are passed, those above the first four arguments are stored in the stack frame.
- Register 27 is the global data register and is used to address any global data needed by the procedure.
- Register 2 contains the address to return to when the procedure is done.
- Registers 28 and if needed 29 are to contain the function result when the function returns.
- Registers 3 through 18 (the callee-saves registers) can be used only if they are saved and restored before returning to the caller.
- Registers 19 through 22 (the caller-saves registers) and registers 1 and 31 are available to use as scratch registers.

There are other conventions for floating-point and space registers which are usually not important in operating system code.

The shadow register environment, which consists of registers 1, 8, 9, 16, 17, 24, and 25, is not compatible with the C environment.



GR0	Zero (by Hardware Convention)
GR1	Scratch
GR2	RP (Return Pointer)
GR3 • • • GR18	Callee-Saves Registers
GR19 • • • GR22	Caller-Saves Registers
GR23 • • • GR26	Arguments
GR27	DP (Global Data Pointer)
GR28 GR29	Return Values
GR30	SP (Stack Pointer)
GR31	MRP (Millicode Return Pointer)

Fig. 1. Register use conventions in the C environment.

and non-VMEbus interrupts. Note the correspondence between the interrupt table structure and logical I/O architecture shown in Fig. 1. The three different interrupt handling situations mentioned above are taken care of by allowing one of the three types of interrupt routines to be specified in the table (see the interrupt action entry in Fig. 3).

Second-level VMEbus interrupts are handled by reading the returned interrupt vector, masking it, and using the result to index to the interrupt action that will handle the interrupt (ⓕ in Fig. 3). The masking is done to prevent indexing to a location outside of the table and to allow the interrupting device to pass back status information in the high part of the word. The mask is computed at system configuration time from the user's specification of the high number to be returned on a given interrupt line. This number is rounded up to the nearest power of two (2^n). For example, if the highest number to be returned on a particular interrupt line is 12 then n is four because 2^4 provides the nearest power of two greater than 12.† This results in a table that is larger than needed but eliminates the need to check if the masked number is too large. Unused entries in both the first-level and second-level interrupt tables are filled with entries that

† The largest value for n is 256.

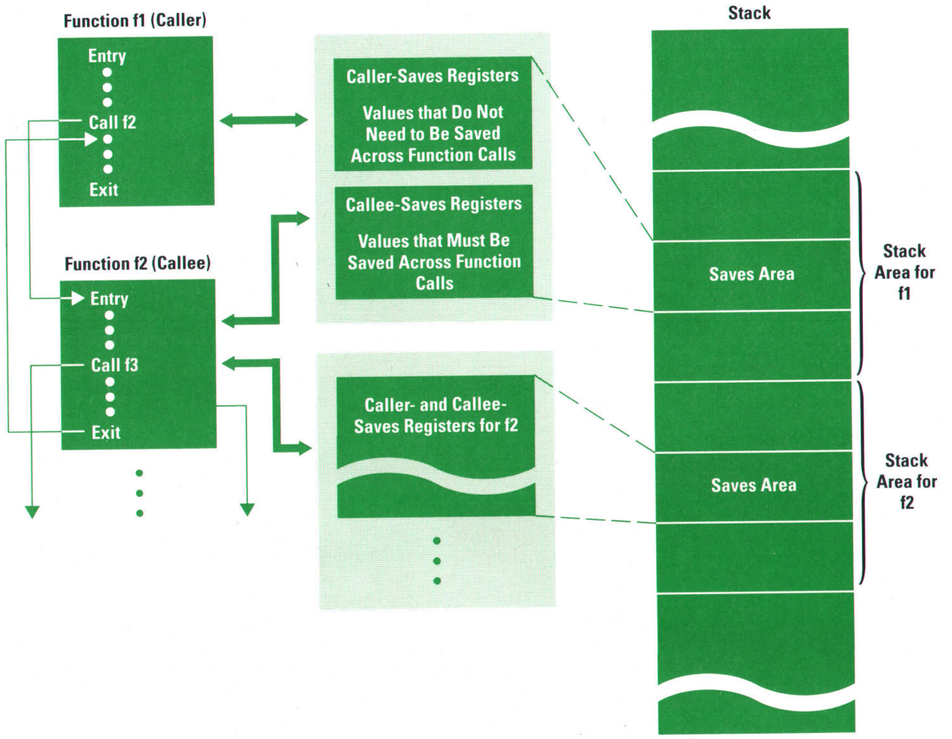


Fig. 4. The relationships between function (or procedure) calls, the caller- and callee-saves registers, and the stack area. The caller puts data it wants to preserve in the callee-saves registers before making a call. If the called routine (callee) needs to use any of the callee-saves registers, it saves the value contained in the register and restores the value back into the register before returning to the caller.

result in system illegal interrupt messages should such an interrupt ever happen.

Initially, the HP-RT team wanted the interrupt handler and the interrupt off times to be "blind" to interrupts for a maximum of 100 instruction times, including any stall states minus cache misses. The notion of blind to interrupts was introduced to cover the case in which the system keeps the interrupt system off, but still processes the interrupt in a timely fashion. This occurs in the interrupt handler, for example, when after it processes an interrupt it looks at the pending interrupts and if it finds one, processes it without turning on the interrupt system. The operating system interrupt dispatching code met the 100-instruction time limit.

Handling Large Contexts

The PA-RISC architecture divides a program's context into two register sets: *caller-saves* and *callee-saves* registers. The caller-saves registers consist of registers that are expected to contain values that do not need to be preserved across a procedure call, that is, values the calling function does not care about. Therefore, these registers are available for use as scratch registers or for parameter passing by the called routine. The callee-saves registers are used for values that must be preserved across a procedure call. Thus, if the called routine wants to use a callee-saves register, it must first save it and then restore it before it returns. The PA-RISC architecture also specifies where these registers must be saved on the call stack (see Fig. 4). This caller-saves and callee-saves convention is used by the PA-RISC compilers so that the system can depend on it.

HP-RT depends on the caller-saves and callee-saves division to keep context management code to a minimum. In particular, on system calls the system saves only the user's (caller's) return address, global register, and stack pointer. The system call handler then calls the requested system call function depending on that function to save and restore any callee-saves registers it may want to use. Likewise, on interrupts or traps where control must be transferred to the kernel stack, only the caller-saves registers need to be saved because HP-RT depends on callee-saves registers to be saved by any function called. Therefore, since the context switch code is called as a function, all it has to save are the callee-saves registers. By saving only what needs to be saved at each step, the system keeps the overhead low for register saves and restores.

HP-RT also takes advantage of the fact that the floating-point coprocessor is enabled by setting bits in a control register. If the coprocessor is not enabled, the system will generate an emulation trap when a process attempts to use any floating-point instructions. Processes start with the floating-point coprocessor disabled. When a process attempts to use floating-point instructions, the code in the emulation trap handler saves the old process's floating-point registers and loads the current process's floating-point registers. In this way, the overhead of floating-point context switching is limited to only the times when it is needed.

In deference to maintaining a low interrupt-off time, the system checks for pending interrupts once it has stored the old process's floating-point registers. If any external interrupts are pending at this time, the system will set the floating-point ownership flags to show that the coprocessor is not owned and then handle the interrupt. The current process will be redispatched still not owning the floating-point coprocessor, but will immediately end up in the emulation trap which will finish the context switch. Of course the interrupt could cause the current process to lose the CPU, possibly even to the process whose state the system just saved. For this reason, a flag is kept to show that the registers were not changed so the process may proceed with only a quick pass through the emulation code to get the coprocessor bits set again.

Setjmp and Longjmp Solutions

On rare occasions the operating system is required to abort a system call. This occurs when the user sets up a signal handler and the signal handler is specified as requiring the termination of any system call that is pending when the signal is delivered. As mentioned above, the system takes advantage of the fact that functions called on a system call will restore the callee-saves registers. These registers are saved on the stack by each function in the call chain, starting from the system call handler to the code that delivers the signal to the user. The problem then is how to recover these registers so the user code will have the correct register set when control is returned to it. The normal way to handle this kind of situation is to do a setjmp call to save the callee-saves registers in a local buffer and then do a longjmp call (which restores the saved registers) from the signal delivering code. The porting team decided that the overhead of a setjmp on every system call was too high.

One solution that was considered was to identify all possible places in the kernel where such a signal could be delivered. Code could then be put in place to do a setjmp only when the signal delivery was possible. This approach was abandoned when it was found that these calls could come from user-written drivers. The solution used is to unwind the stack, picking up each of the saved registers until the stack is back to the system call handler. This solution takes more time in the rare case of a call being aborted, but does not put overhead in the path of all system calls.

Hardware Help

It was mentioned above that the VMEbus hardware holds off interrupts until the information needed to process the interrupt is available. The HP-RT team also requested and received a real-time mode in the interrupt convention for onboard I/O device interrupts. The normal convention was that all onboard device interrupts were collected into one bit (each bit corresponds to one interrupt line). Under this convention the software interrupt handler would first decode the interrupt source to this bit and then read an I/O space register that contained a bit map of all the onboard devices requesting interrupt service. The hardware convention used was to clear this register when it was read. This required the software to keep track of all the bits that were set and to call the handler for each bit. The software management task for this convention would have been fairly high because the real-time system wants the interrupt system on most of the time, which means that it is possible for another interrupt to be received from another onboard device before the current interrupt is completely processed. At the same time, the rest of the main processor's interrupt register would not be in use.

The HP-RT team asked for an interrupt mode in which each onboard device has its own interrupt bit on which it can interrupt the main processor. This convention not only eliminates the need to remember which bits were set, but also eliminates a level of decoding in the interrupt path.

Conclusion

One of the main goals of the HP-RT project was to minimize the time to handle interrupts. Table I, which shows the results of these efforts, is a task response time line that shows the time consumed by each activity in the path from an interrupt to the task (e.g., user code) that does something to respond to the interrupt. For cases in which an interrupt is handled by an interrupt service routine in the operating system and not user code, the interrupts disabled and dispatch interrupts times shown in Table I are the only times involved in determining the total task response time. Their worst-case times in this situation are 80 μs and 6 μs respectively, giving a total task response time of 86 μs. The 80 μs time is rare and work is continuing to reduce this time.

Table I
Time Line for HP-RT Running on the HP 9000 Model 742rt

Tasks Performed After an External Event	Task Response	
	Best Case	Worst Case
Interrupts disabled	0	0
Dispatch interrupts	3 μs	6 μs
Other interrupts	0	9 μs†
Context switch off	0	166 μs††
Scheduling and switching	27 μs	45 μs
Return from system call	1.2 μs	4.6 μs
Total time	31.2 μs	230.6 μs

† Three interrupts

†† This time is rare and is in code other than the interrupt and context switch code. Work is continuing to reduce this time.

HP-UX is based on and is compatible with UNIX System Laboratories' UNIX* operating system. It also complies with X/Open's* XPG3, POSIX 1003.1 and SVID2 interface specifications.

UNIX is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and other countries.

X/Open is a trademark of X/Open Company Limited in the UK and other countries.

The HP Tsutsuji Logic Synthesis System

A new logic synthesis system has reduced the time to design ASICs by a factor of ten.

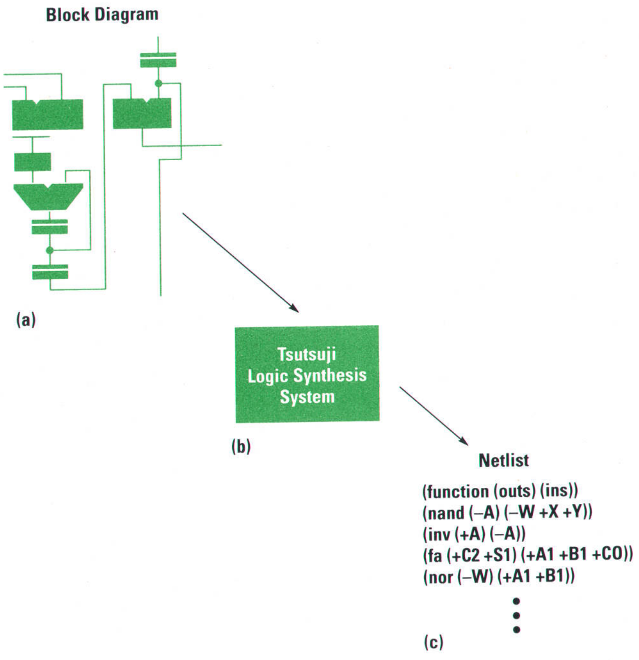
by W. Bruce Culbertson, Toshiki Osame, Yoshisuke Otsuru, J. Barry Shackleford, and Motoo Tanaka

Logic synthesis assists and automates the process of refining digital designs from high-level, abstract conceptions to low-level, concrete specifications for physical implementation. The HP Tsutsuji logic synthesis system, a software package that runs on HP 9000 Series 700 workstations, was jointly developed by Hewlett-Packard Laboratories in Palo Alto, California and Yokagawa-Hewlett-Packard (YHP) Design Systems Laboratory (YSL) in Kurume, Japan. Tsutsuji, the Japanese word for azalea, was adopted as the name of the product because at the inception of the project, Kurume was hosting the World Azalea Congress.

Input to the Tsutsuji logic synthesis system is expressed as block diagrams composed of adders, multiplexers, shifters, register files, and so forth. Tsutsuji transforms these block diagrams into efficient, electrically and functionally correct netlists,† which can be implemented in various technologies (see Fig. 1).

The most obvious benefit of logic synthesis is that it reduces the time needed to develop a new product. In a competitive market, the time needed to develop a product often has a greater influence on profitability than the product's performance or factory cost because of its effect on the technology potential in the product (see Fig. 2). In addition to shortening the design phase of the development schedule, logic synthesis can also reduce the debugging and testing phases by eliminating the errors that inevitably occur when a gate-level design is produced manually.

A disadvantage of the traditional digital design process is that designs are not captured precisely until they have been refined to too low a level of abstraction (Fig. 3a). At this point, technological dependencies have been introduced and high-level functions (Fig. 3b) have been obscured. Experience has shown that these designs can almost never be reused to take advantage of faster and cheaper technologies when they become available. In contrast, Tsutsuji accepts a high-level, technology-independent design and automatically maps it to the target technology. Reusing an old design can be as simple as rerunning the synthesis tools. Freed from



```
(function (outs) (ins))
(nand (–A) (–W +X +Y))
(inv (+A) (–A))
(fa (+C2 +S1) (+A1 +B1 +CO))
(nor (–W) (+A1 +B1))
```

Fig. 1. Tsutsuji is a high-performance logic synthesis system. Designs are expressed as block diagrams, which are transformed by Tsutsuji into a netlist file that can be used by gate array manufacturers to produce an application-specific integrated circuit (ASIC).

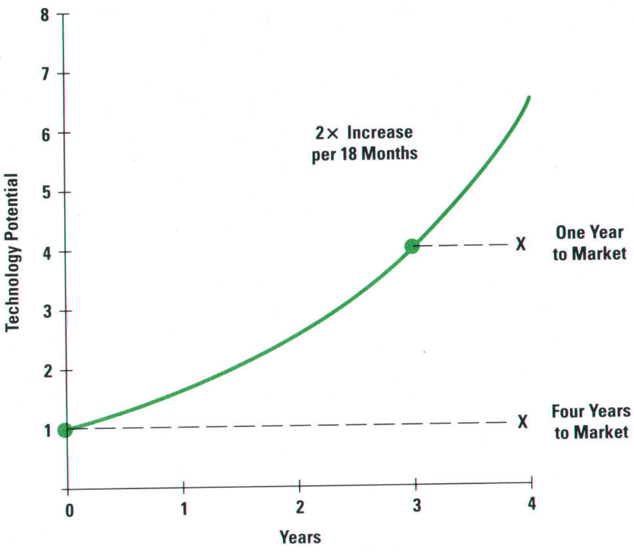


Fig. 2. Assume that the technology potential, which includes chip cost, speed, and density, grows exponentially. Then a project that can make it to market in one year will be implemented with a technology that will have four times the potential of a project that takes four years to market.

† A netlist is a list of logic gates and the interconnections, called nets, between them.

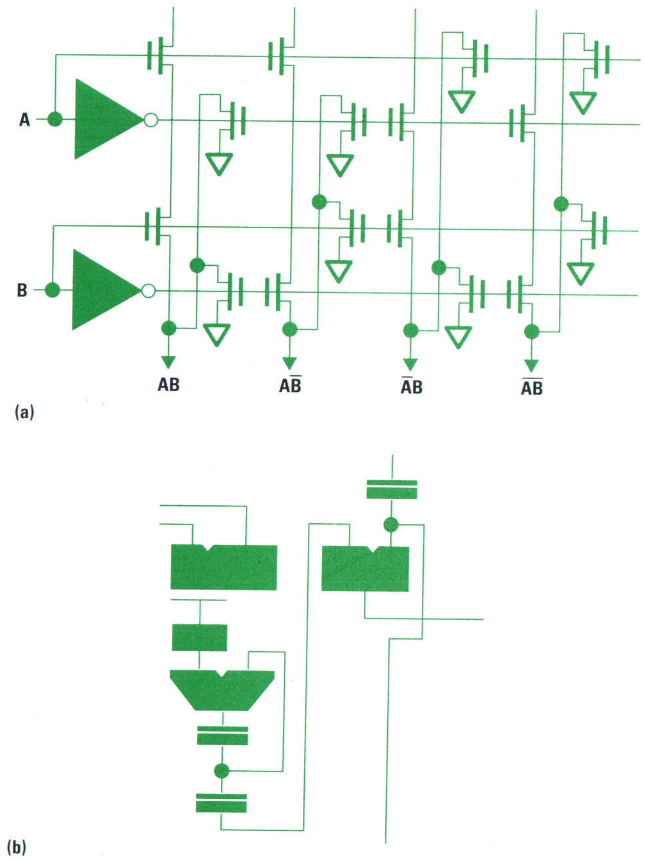


Fig. 3. Designs that are expressed directly in the technology of implementation (a) are often difficult or impossible to remap effectively. Conversely, designs that are created by logic synthesis from high-level modules (b) are inherently easy to remap.

the tedious low-level design tasks, a designer can devote increased time to the more profound system-level design issues, which can more significantly influence performance. Research into the art of implementing a specific function, for example a multiplier, needs to be performed only once to embed it into the logic synthesis system, after which it becomes available to all users of the system.

Logic Synthesis

The initial focus of Tsutsuji is to assist the design of application-specific integrated circuits (ASICs). ASIC vendors typically provide low-level tools for placement and routing, rule checking, and so forth. Tsutsuji is intended to complement and augment such tools rather than duplicate them. Thus, the output of Tsutsuji is the set of files a vendor needs to produce an integrated circuit.

After it is entered with a graphical editor, the block diagram describing the circuit is translated to a technology-specific netlist in two steps. In the first step, module generators, driven by parameters supplied from the block diagram, expand the blocks into a generic netlist of simple gates. At this stage, the gates have no restrictions on fan-in and fan-out and are essentially equivalent to logic equations. However, some modules such as multipliers can take advantage of higher-level primitives like full adders. If it is known at this stage that the target technology contains these higher-level primitives, then the modules can be instructed to emit them rather than the lower-level logic gates. This makes the task of technology mapping substantially easier and quicker.

During the second step, a technology backend manipulates the generic netlist into a new netlist that satisfies the design rules of the target technology (such as fan-in and output drive restrictions) and exploits the technology's special features.

Module Generators

The heart of Tsutsuji is a library of module generators, each of which can translate blocks of a single functional type into a collection of simple generic gates. The library contains module generators for all of the kinds of blocks that are typically used to construct computer data paths and control logic. There are currently about fifty module generators, including:

Adders
Comparators
Dividers
Majority Logic
Random Logic
Selectors
ALUs
Decrementers
Encoders
Multipliers
Registers
Shifters
Counters
Decoders
Incrementers
Multiplexers
Register Files
State Machines

It is important to stress that the library is not composed of fixed designs, as are standard cell and gate array libraries. Instead, it is composed of generators that can produce an endless variety of fixed designs. For example, blocks are synthesized with exactly the desired operand lengths. By adjusting the parameters given to the module generators, the designer tunes the synthesized circuit to achieve the project's cost and performance objectives. The speed of the synthesis process permits many design choices to be tried, with actual cost and performance data gathered for each. To produce a product upgrade, the current design can be reused, with blocks regenerated using synthesis parameters that yield higher performance. The new product is functionally equivalent to the first; consequently, the need for simulation and testing is reduced.

Extensive literature exists describing the implementation of data path and control logic functions, and much of this knowledge has been incorporated into the generators. Often there exist several algorithms that can be used to implement a given function. For example, the module library includes ripple-carry adders, carry-lookahead adders, and conditional-sum adders. Multipliers can be synthesized using iterative cellular arrays or carry-save adder arrays. Best of all, the designer needs little understanding of the alternatives since all are functionally the same and since fast synthesis provides a quick comparison of cost and performance.

Example: Shifter

Once an algorithm is chosen, there often remain a number of structural choices that can influence cost and performance. As an example, a 16-bit unidirectional shifter will be considered in detail. The shifter has 16-bit input and output data buses. There are also four weighted shift-amount inputs and a shift-in input.

In the case of the shifter, the library has only one algorithm—the shifter will be implemented as a collection of n-to-1 multiplexers. On the other hand, there are many possible structural arrangements of the multiplexers that will produce the

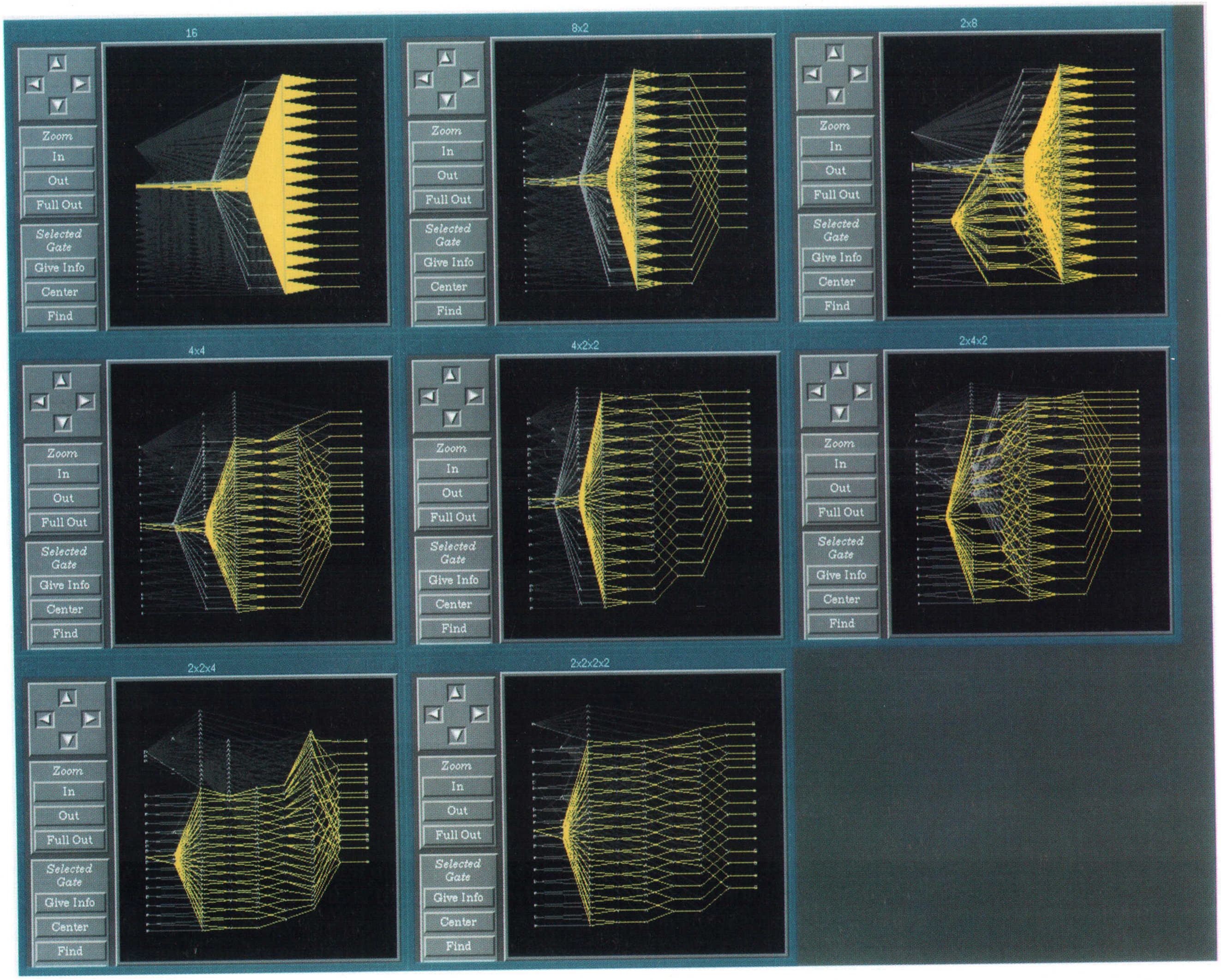


Fig. 4. All eight possible organizations for a 16-bit unidirectional shifter are shown in this topology graph mosaic. The organizations derive from the factorizations of 16, the bit width. The factorization (2 4 2) results in a shifter composed of a level of 2-to-1 multiplexers followed by a level of 4-to-1 multiplexers and finally followed by a level of 2-to-1 multiplexers.

desired shifter. For example, the shifter could be structured as one level of sixteen 16-to-1 multiplexers, or two levels, each composed of sixteen 4-to-1 multiplexers. Each factorization of the number sixteen yields a different way to structure the shifter. For example, the factorization (2 8) corresponds to a shifter with a level of 2-to-1 multiplexers and a level of 8-to-1 multiplexers. Fig. 4 shows topology graphs of the first-level (generic gates) implementations of all eight possible organizations of a 16-bit unidirectional shifter. For an explanation of topology graphs see "Netlist Topology Visualization" on page 44.

Table I contains data for a selection of structures for the shifter. The speed advantage of the (16) structure, which is significant in the technology-independent (generic gates) form, is not very pronounced after the CMOS technology backend corrects the excessive fan-in and fan-out. Good compromises between gate count and speed are offered by both (2 2 2 2) and (4 4); (2 2 2 2) may be favored in a technology providing only two-input gates. The organization of the shifter is specified on the module's tuning page. The tuning page is made visible by selecting the module in the block diagram and then clicking on the tuning page button to the left of the drawing area. Note in Table I that the (4 4) organization of the CMOS shifter is only about four percent slower than the (16) organization and requires only 41 percent as many cells for implementation.

To summarize, module generators provide designers with custom-produced functional blocks with exactly the required operand sizes. Designers can choose from a large number of functions. Given a function, a number of algorithmic and structural choices are usually available.

Technology Backends

The technology backends perform two functions: optimization and mapping. Optimization improves the cost and performance of a circuit. Mapping converts the netlist of generic gates produced by the module generators into an electrically correct netlist of gates that can be implemented in the target technology. Mapping is necessary because the module generators use gates chosen from a fixed set of functions, which may be different from those available in the target technology. Also, the module generators assume gates may have

Table I
16-Bit Unidirectional Shifter

	Organization				
	(16)	(2 8)	(4 4)	(2 4 2)	(2 2 2 2)
Generic Gates:					
Fan-in_{avg}	2.6	2.3	2.2	2.0	1.9
Fan-in_{max}	16	8	4	4	2
Fan-out_{avg}	1.9	2.0	1.9	1.9	1.8
Fan-out_{max}	16	16	16	16	16
Levels	5	6	6	7	9
Gates	300	209	172	186	200
CMOS Gates:					
Fan-in_{avg}	2.1	2.0	2.0	2.0	1.8
Fan-in_{max}	10	6	4	4	2
Fan-out_{avg}	2.0	1.9	1.8	1.8	1.7
Fan-out_{max}	9	8	8	8	8
Levels	9	9	7	7	9
Gates	474	334	257	244	251
Cells	575	345	236	219	212
Relative Delay	1.000	1.037	1.094	1.068	1.087

unlimited fan-in and fan-out. The technology backends allow Tsutsuji to realize an important goal: the ability to implement one design efficiently in multiple technologies.

Our experience with Tsutsuji has shown that relatively simple backends are most effective. We have tried other systems with far more sophisticated optimization features. These systems can considerably improve a poor design, although often the result still leaves much to be desired. For example, no system we have seen will convert a ripple-carry adder to a carry-lookahead adder. However, if the design is nearly optimal to begin with, then the best optimizers can improve it very little. Furthermore, these systems are so slow, even working on small circuits, that they discourage the experimentation and iterative design approach that we wish to promote.

Tsutsuji designs are, in fact, nearly optimal before they reach the technology backends. Because the implementation of data-path structures like adders has evolved to a very high art, and because our module generators have captured that art, circuits produced by the generators typically have excellent fan-in, fan-out, and cost-performance characteristics. For control logic, Tsutsuji uses generators that include their own optimization algorithms. Since these blocks typically contain relatively few gates, the optimization performed by the module generators is quick and effective.

The mapping and optimization applied by the backends involve only small numbers of adjacent gates at a time. These transformations, called *peephole optimizations*, can be performed far more quickly than the global optimizations used in some other systems. Most of the transformations can be specified as rules, each of which is a pair of patterns. The design is searched for collections of gates that match the first pattern in a rule. The collection of gates is then replaced with the second pattern in the rule (see Fig. 5).

A gate with excessive fan-in must be replaced by the Tsutsuji backends with trees of low-fan-in gates that implement the same function. To avoid increasing delay, nets on the critical path should enter the fan-in tree at its root while nets with plenty of slack† can enter the tree at a deeper level. Fixing excessive fan-out is analogous: the net with too many loads is replaced with a tree of buffers plus the original driver, which serves as the root. In this case, loads should be driven

† Slack is a measure of how critical the timing is at a gate or net, with zero slack being most critical. It is defined as the difference between the length of the longest path through the gate or net and the length of the critical path.

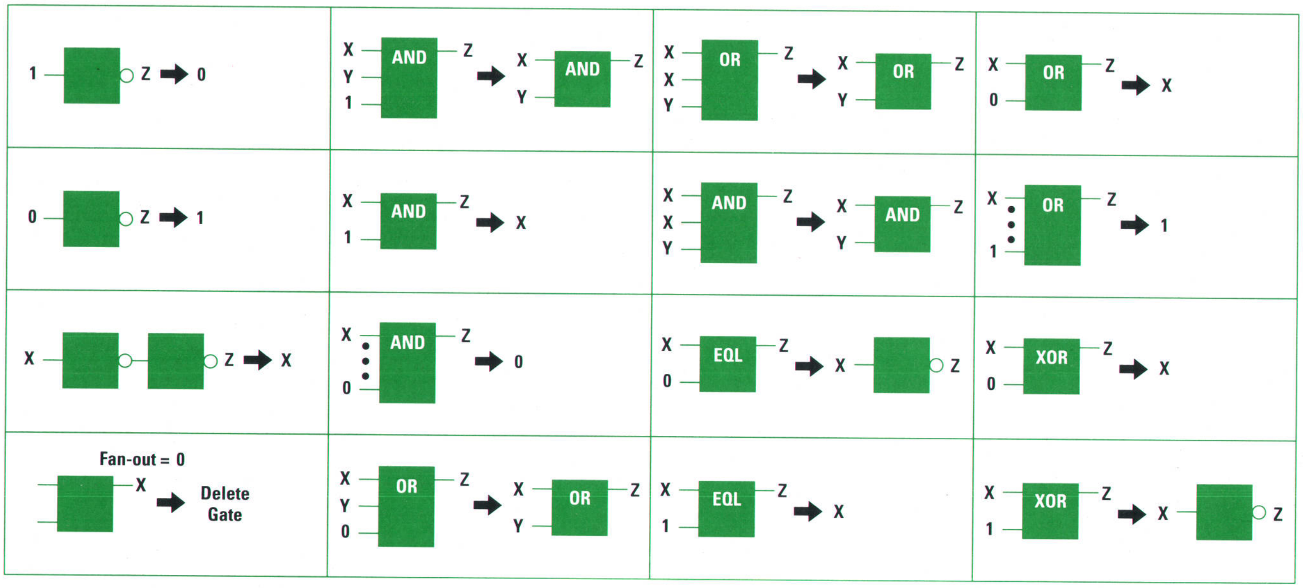


Fig. 5. Optimization rules are pairs of patterns. If the first pattern of a rule matches a fragment of the design, the fragment is replaced by the second pattern. (EQL = equal, the opposite of XOR.)

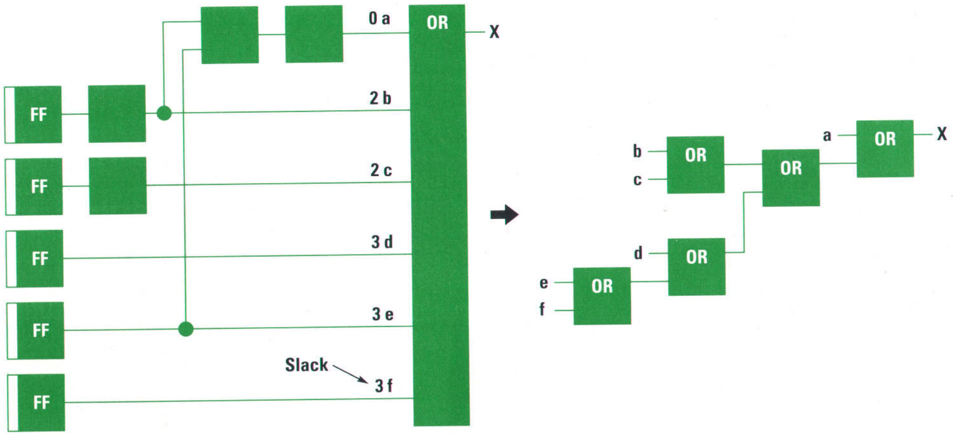


Fig. 6. A single gate with excessive fan-in is replaced by a tree of gates by the technology backend. In this example the fan-in limit is two. The shape of the tree and the points where the nets enter it are carefully chosen to avoid increasing delay.

by gates at a tree depth not greater than the slack through the load. An algorithm has been developed that builds optimal fan-in and fan-out trees. Optimal in this case means that no tree can be found that has less impact on the critical path. Fig. 6 illustrates the construction of an optimal fan-in tree.

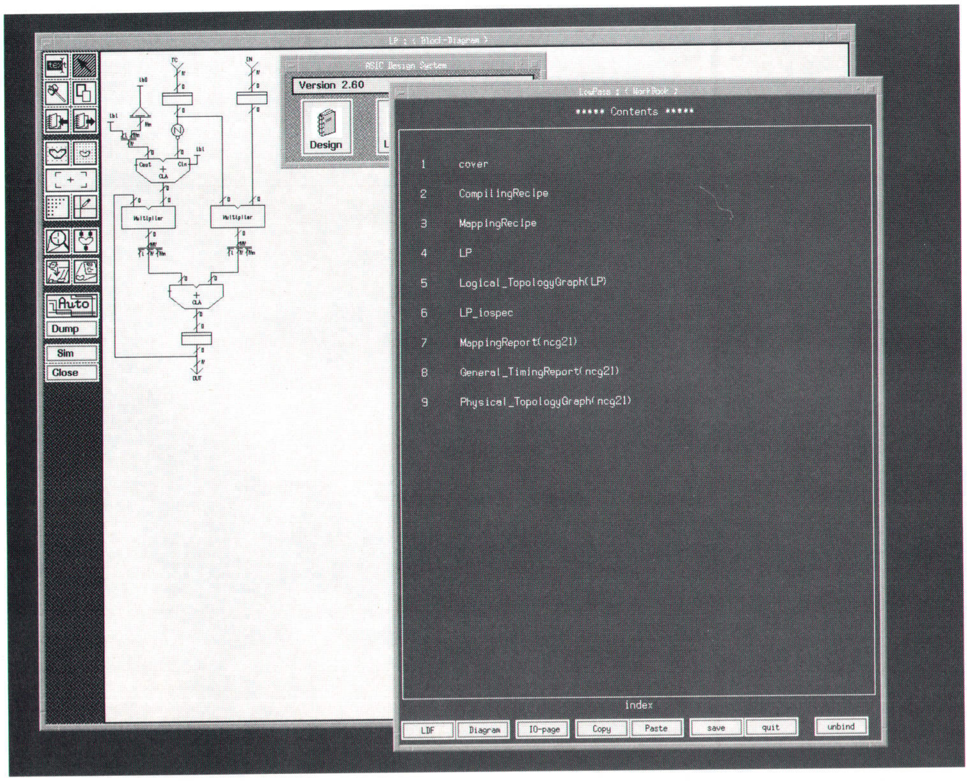
Human Interface

As massive VLSI becomes more prevalent, a way must be found to manage the complexity of million-gate systems on a chip. We wish to elevate the designer's perspective by encouraging optimization at the system level rather than at the gate or transistor level.

A great deal of effort was put into creating a system that would both encourage system-level thinking and synthesize and map designs rapidly. To complement this system, we wished to design a human interface that would evoke the intuition and even the playfulness of the designer. Our intent was that the designer would read the instructions after using the system.

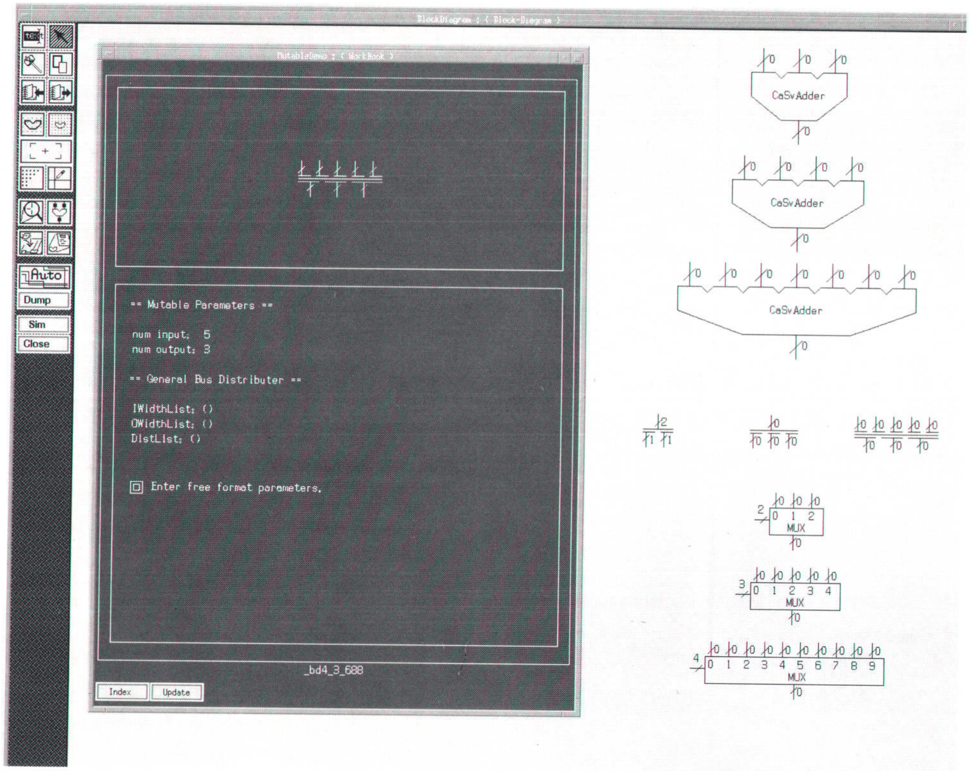
The analogy that the YSL design team chose for the Tsutsuji human interface was that of the engineer's design notebook (see Fig. 7). At a level above this is the concept of the library, which is simply a collection of notebooks and component catalogs that can be used in any design.

Design notebooks are broken down into pages. The first page is the index page, by which all other pages can be accessed. As the design progresses, pages are automatically added to the design notebook. For example, in a hierarchical design, a number of lower-level components would be created. Each of these components along with the top-level design would then automatically be added and appear in the index. Subsequent pages would be added to reflect the results of technology mapping, timing, and topological analysis.



```
***** Contents *****

1    cover
2    CompilingRecipe
3    MappingRecipe
4    LP
5    Logical_TopologyGraph(LP)
6    LP_iospec
7    MappingReport(ncg21)
8    General_TimingReport(ncg21)
9    Physical_TopologyGraph(ncg21)
```

Fig. 7. Tsutsuji presents the design as an engineer's design notebook. At the level above the notebook is a library consisting of other notebooks and component catalogs.



```
== Mutable Parameters ==

num input:  5
num output: 3

== General Bus Distributer ==

IWidthList: ()
OWidthList: ()
DistList: ()

□ Enter free format parameters.
```

Fig. 8. The tuning page for a module is accessed by selecting a module with the mouse and then clicking on the tuning page button to the left of the drawing area. Some symbols such as the carry-save adder automatically change their form as a function of the tuning parameters.

Block Diagram Design Entry

Nearly all substantial designs start out as block diagrams. We have chosen this natural form of expression as the principal form of design specification within Tsutsuji.

The design is entered by means of a block diagram drawing editor. The editor allows the designer to create, copy, move, delete, and connect graphical block diagram objects freely. A block diagram object can be a wire or bus connecting two or more modules, a module, or even a list of logic equations. Objects can be readily copied from other block diagrams in other design notebooks. To connect modules, the designer need only point at the appropriate connection points and Tsutsuji will automatically route the line. Modules that are already connected can be moved and Tsutsuji will automatically reroute the connections to the module.

Hierarchical designs can be created by entering a design in the normal manner and then putting the design in the design book where it can accessed via the Tsutsuji index page. Tsutsuji automatically constructs a symbol for the user. However, fastidious users who want a more distinctive symbol can use the drawing editor to alter the symbol shape.

Tuning parameters for modules are specified by first selecting the module with the mouse and then clicking on the tuning page button to the left of the drawing area. A special page for the selected module will appear and then the parameters can be entered (see Fig. 8).

Certain modules such as bus distributors, carry-save adders, and multiplexers require a different symbol depending upon their configuration. Rather than force the user to specify the shape for each configuration, Tsutsuji has a class of symbols that are mutable—the form changes as a function of the tuning parameters (see Fig. 8).

Textual Design Entry

Usually the data-path portion of a design is most naturally expressed graphically. Text is sometimes appropriate, however, for specifying the control portion of a design. *Logical Description Format*, or LDF, is the Tsutsuji language for specifying designs textually. LDF is similar to the C programming language, so it looks familiar to many users.

To use LDF, the user places a box in the graphical design and connects signals to it. With the mouse, the user then executes a command that causes an editor window to appear. By typing LDF text into the window, the designer specifies the function of the box.

The first two examples in Fig. 9 both specify the same function, an adder, but do so using two different features of LDF: random logic and truth tables.

The first line in Fig. 9a lists the four signals that connect the adder subdesign to the rest of the design. The last two of those signals are prefixed with an ampersand to indicate that they are outputs; the first two are inputs. The third line, which begins with the word net, creates and names two wires, which will be internal to the subdesign. Other internal signals will be created automatically if they are needed to implement the random logic expressions. The line carry = a & b;, as one might expect, creates an AND gate, connects its output to the signal carry, and connects its two inputs to the signals a and b.

Fig. 9b shows a truth table for an adder plus LDF text that implements the truth table. The truthtable feature in LDF is merely a textual structure for expressing a truth table.

The automaton structure in LDF allows the user to specify a state machine. It consists of a list of states. For each state, expressions are given for the outputs, and conditions are

Random Logic

```
adder (a, b, &sum, &carry)
{
   net local_net_1, local_net_2;

   carry = a & b;
   local_net_1 = ~a & b;
   local_net_2 = a & ~b;
   sum = local_net_1 | local_net_2;

}
```

(a)

inputs		outputs	
a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
adder (a, b, &sum, &carry)
{
   truthtable a, b: sum, carry
   {
      case 0, 1: 1, 0;
      case 1, 0: 1, 0;
      case 1, 1: 0, 1;
      default: 0, 0;
   }
}
```

(b)

```
StateMachine (a, b, next, &Out)
{
   automaton {
      STATE_0:
         Out = 0;
         if (next) goto STATE_1;
      STATE_1:
         Out = a | b;
         goto STATE_0;
      }
}
```

(c)

Fig. 9. Most design entry is done graphically in Tsutsuji. However, some portions of designs are more naturally expressed using text. Tsutsuji provides the LDF language for this purpose. This figure gives three examples of LDF: (a) specifies some combinational logic to implement an adder, (b) describes the same adder using a truth table, and (c) specifies a simple state machine.

given for changing to other states. Fig. 9c illustrates a simple state machine with two states.

Netlist Topology Visualization

The topology graph is a new means we developed for viewing a gate-level design. Unlike a traditional schematic, a topology graph can display a large design in a single window and can make the performance characteristics of the circuit easy to understand. The topology graph also makes it easy to trace the automatically generated gates back to modules in the user's high-level design.

Fig. 10 is an example of a topology graph. Circuit inputs are placed in a column on the left side of the graph. The horizontal coordinate of a gate is set to be proportional to the delay of the longest path from the inputs to the gate. Registers appear twice on the diagram. They are drawn first in the input column with only the register outputs shown (register outputs are inputs to the logic gates). They appear again with only the register inputs drawn at the right-hand endpoint of one or more paths through the circuit. Circuit outputs also appear at the right end of paths.

A straight line is drawn between two gates if an output of one of the gates drives an input of the other. The brightest colors are used to show connections with the lowest slack. For example, the critical path in Fig. 10 is drawn in yellow. This emphasizes the part of the circuit that limits the speed, which is usually the part of the circuit the designer most wants to see. Because delay information is inherently graph-oriented, we have found this graphical presentation of delay information to be an enormous improvement over the traditional textual delay report.

Tsutsuji users typically make their high-level designs functionally correct before they bother to examine their designs at the gate level. Once the design is functionally correct, there rarely is any need to look at the gate-level design in detail. Nevertheless, the topology graph program includes features for scrolling to any part of the design and zooming to any desired level of detail.

A particular gate can be selected by clicking with the mouse or typing the name of the gate. The green circle in Fig. 10 indicates a selected gate. Once selected, the gate can be brought to the center of the screen and magnified. A pop-up window of information about the gate can be requested; it gives information like the type and name of the gate, the gate's fan-in and fan-out, the slack at the gate, and so forth. The tree of signals driving the selected gate and the tree of signals driven by the selected gate can be highlighted, as shown by the red portion of Fig. 10.

Once a gate has been selected, it is possible to request a pop-up window showing the names of the gates that drive and are driven by the selected gate. Clicking on one of the names causes the corresponding gate to become the selected gate. This makes it easy to navigate through the design, following the circuit's interconnections.

When the user types a name into a module selection window, the named module is then highlighted, as shown in red in Fig. 11. This allows the designer to correlate blocks in the high-level design with gates in the gate-level design. The user can also request a pop-up window of information about the selected module.

The ability to see a particular high-level module within the topology graph of the entire circuit is invaluable for setting module tuning parameters. For example, the designer might use the mouse to select a gate on the critical path. From the gate information window, the user would learn the module from which the gate was synthesized. Then the user would select that module to highlight it on the topology graph. If the module were contributing significant delay to the design, the user might retune the module for higher performance. In another scenario, the user might select a module that was not on the critical path and retune it for a slower but cheaper implementation.

Simulation

To achieve our project goal of substantially increasing designer productivity, it was imperative to develop a fast simulator for Tsutsuji. Traditionally, simulation has been a process for verifying designs that were nearly complete.

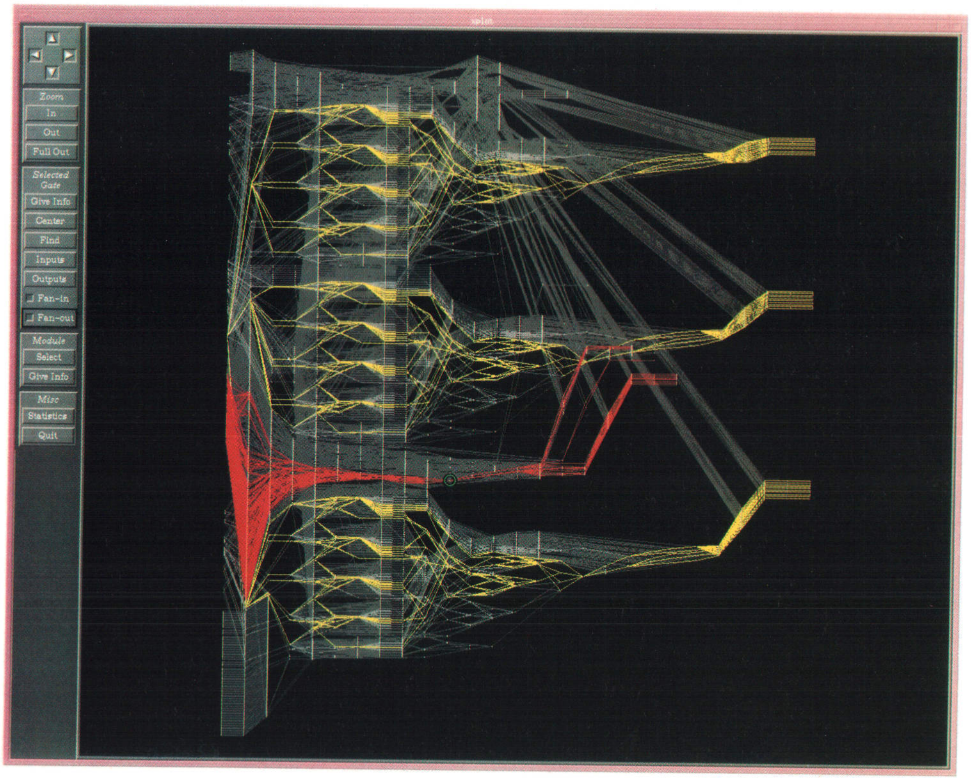
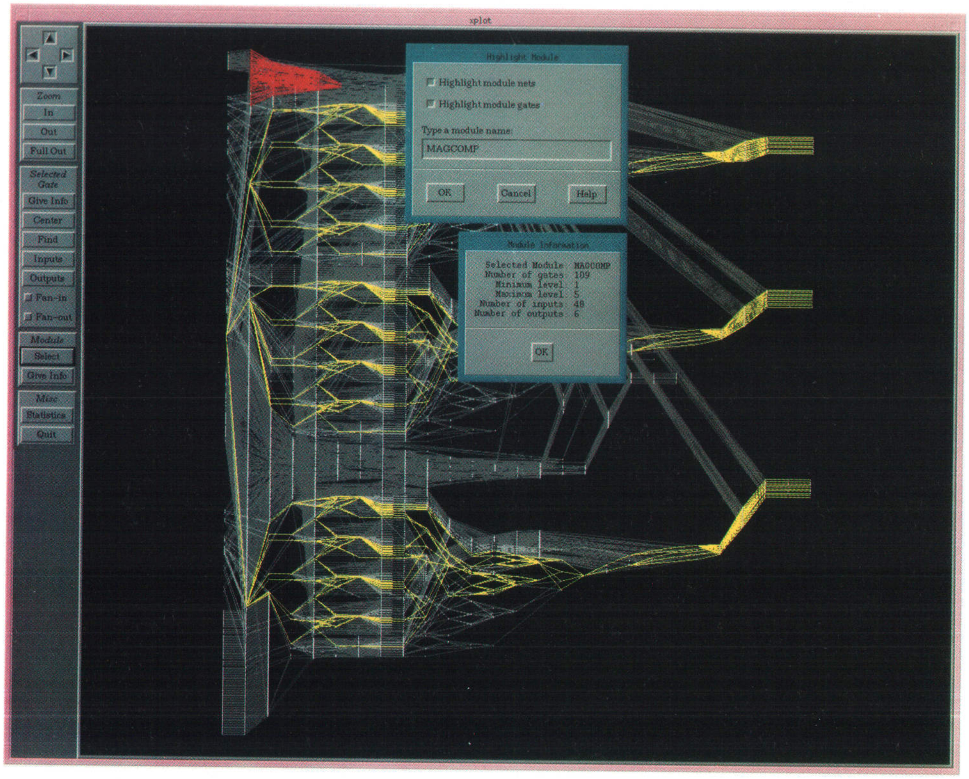


Fig. 10. The topology graph lets a Tsutsuji user view an entire gate-level design on a single screen. Signals flow from left to right in the diagram and the longest paths horizontally have the most delay. The critical paths are colored yellow. A gate has been selected as indicated by the green circle. Many features of the topology graph program relate to the selected gate; in this case, the fan-in and fan-out trees for the selected gate have been highlighted in red.

Computers were left unattended for hours or days while simulations ran and produced reams of paper. We wanted the Tsutsuji simulator to aid the early phases of design, producing results in real time and presenting them in the context of the application. We wanted designers to be able to experiment with significant design changes and see the effects instantaneously.

A previous YSL product included a simulator that evaluated about a thousand gates per second. The Tsutsuji simulator has achieved simulation rates as high as twenty-three million gate evaluations per second.† Some of this increase is the

† This was measured while simulating a 5000-gate floating-point multiplier using an HP 9000 Model 730 computer.



□ Highlight module nets

□ Highlight module gates

Type a module name:

MAGCOMP

```
 Selected Module  MAGCOMP
 Number of gates  109
   Minimum level  1
   Maximum level  5
Number of inputs  48
Number of outputs 6
```

Fig. 11. In this topology graph, a name has been typed into the Highlight Module pop-up window. This causes all the gates and interconnections within the named module to be highlighted in red. A Module Information window displays information about the highlighted module. The ability to see how one module is situated within the entire topology graph is useful for setting the module's tuning parameters. For example, if the critical path flows through the module, the designer may want to tune it for higher speed. Conversely, if no critical paths flow through the module, the designer may want to tune it for lower cost.

result of an impressive leap in workstation performance that occurred between the releases of the two products. Several other factors also contributed.

The most significant factor was the development of a special-purpose compiler that produces efficient simulation code. Much of the work performed on each simulation cycle by the previous simulator is now performed once during the compiling phase. Also, Tsutsuji produces circuits that adhere to strict design rules that make it possible to simulate the circuits accurately with a much simpler simulation strategy. For example, the delay in the circuits can be completely and quickly characterized by a separate static timing analyzer program; hence, the simulator can ignore all timing issues. Since Tsutsuji circuits use simple clocking and two-state Boolean logic, each gate only needs to be evaluated once per simulation cycle. A gate typically can be evaluated by a single, simple, machine-level instruction on the host computer.

When the user wishes to simulate a design, Tsutsuji displays the graphical simulation window. The user can choose buses to observe and can specify virtual instruments for driving or viewing values on the buses. Tsutsuji then automatically runs the simulation compiler and starts the simulation and the virtual instruments. The simulation program and the virtual instruments run as separate UNIX* processes that pass vectors through UNIX interprocess communication channels. This approach provides a flexible means for users to add new virtual instruments. To do so, the user needs UNIX programming skill but does not need to know anything about the internal structure of Tsutsuji.

Simulator Register Allocation

One of the interesting algorithms developed for the simulation compiler is the register allocation strategy. Computers store data in memory and registers. Registers are scarce and fast; memory is abundant and slow. Register allocation attempts to minimize the movement of data between memory and registers and to maximize the amount of calculation that is done in registers.

One of the first things the compiler does is transform the netlist into a list of instructions for a simple, idealized computer. These instructions are similar in function to instructions executed by real computers and are simplified mostly in the way they refer to data. Many optimizations that are complex to perform on real computer instructions can be performed easily and effectively on the simplified instructions. The compiler removes the simplifications in several stages until, finally, the simplified instructions become real computer instructions.

Typical ICs have at most several hundred input and output signals but have thousands of internal signals. In the simulation program, the values of the internal signals are stored in temporary variables. In the list of instructions, there is a point where a temporary variable first appears and another point where it is last used. The number of instructions between these points is called the *lifetime* of the variable. Storage (memory or registers) can be used for multiple variables if their lifetimes do not overlap.

A temporary variable is often used in many instructions. The first few instructions calculate the value of the variable, while the rest use the value to calculate other values. The number of instructions that use a variable is called the *reference count* of the variable.

A variable's lifetime and reference count can be used to measure the desirability of storing the variable in one of the scarce registers. If the lifetime is long and the variable is in a register, then many other variables are prevented from using the register. Hence, a long lifetime argues against putting a variable in a register. If a variable has a high reference count and is stored in a register, then many time-consuming memory references are avoided. Thus, a high reference count argues in favor of storing a variable in a register. Combining these ideas, we define the *cost* of putting a variable in a register to be the variable's lifetime divided by its reference count.

Our register allocation algorithm attempts to store low-cost variables in registers. During register allocation, the compiler passes sequentially over the instruction list. When a variable appears for the first time, it is assigned a register if its cost is low and a register is available; otherwise, it is assigned a location in memory. After a temporary variable appears for the last time, its storage becomes available again.

One question remains: how should low cost be defined? Rather than try to choose a specific threshold to separate high and low cost, we use an adaptive strategy. Whenever the compiler tries to allocate a register to a low-cost variable but finds none available, the threshold is lowered. Whenever a high-cost variable is assigned to memory and registers are available, the threshold is raised.

Our register allocation algorithm produces simulation code that runs almost four times faster than code that keeps all variables in memory. Yet, it is simple and requires minimal time and memory while compiling.

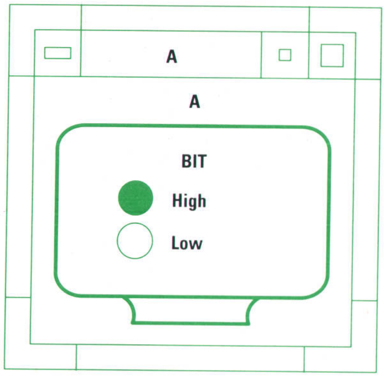
Virtual Instruments

By providing a set of versatile virtual instruments, we hope to move the designer closer to the application domain and away from the Boolean logic domain. Presently, Tsutsuji includes benchtop accessories and instruments that range in complexity from a simple on/off switch to a network analyzer. These are all instruments that the user can interact with in a real-time fashion as the simulation is progressing. The high speed of the simulator makes the concept of virtual instruments practical and allows the designer to participate in an interactive environment.

Probe. Probes are automatically attached to all primary input and output nodes when Tsutsuji is placed into simulation mode. The user can optionally connect probes to internal circuit nodes to aid in monitoring and debugging.

Switch. The switch (see Fig. 12) is a simple one-bit input port. It provides a convenient way for designers to interact with the logic simulation.

Constant Generator. The constant generator (see Fig. 13) is the equivalent of a potentiometer connected across a fixed voltage source and feeding an analog-to-digital converter. The degree of quantization of the constant generator is automatically determined by the width of the bus to which it is connected. Just like a laboratory potentiometer, the constant generator has coarse and fine adjustments: the outer ring on the knob is the coarse setting and the inner ring acts as a


A

A

BIT

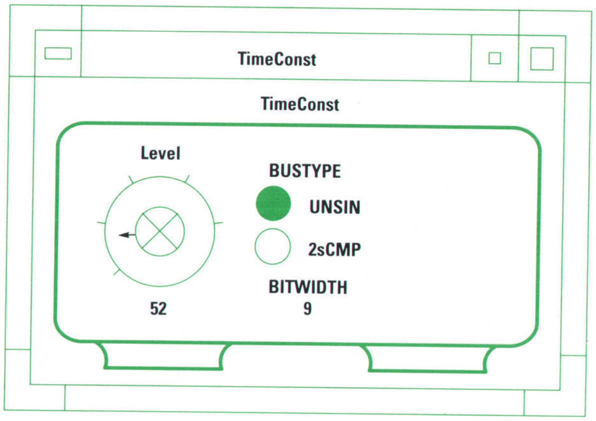
● High

○ Low

Fig. 12. Switches are a simple way for the user to interact with and control the simulation. The switch is activated by use of the mouse. The name of the input port becomes the title displayed on the switch panel.

vernier. For exact setting, the user can click on the displayed value with the mouse and then type the value from the keyboard. The output can be changed between two's complement and unsigned by clicking on the selector button.

Function Generator. The function generator (see Fig. 14) is a means of applying stimuli to the simulator. It is modeled after a conventional analog signal generator. Multiple variable-period, variable-amplitude waveforms are available (e.g., sine, triangle, square, ramp). Data can also be read directly from a file. The function generator's output bus width (i.e., quantization) is determined automatically by the width of the bus to which it is connected. The binary output of the function generator can be presented in either unsigned or twos complement form. An additional useful feature is that the output of one generator can be used to modulate a second one to create complex waveforms. The modulation includes amplitude, frequency, phase, and simple summation.


TimeConst

TimeConst

Level

52

BUSTYPE

● UNSIN

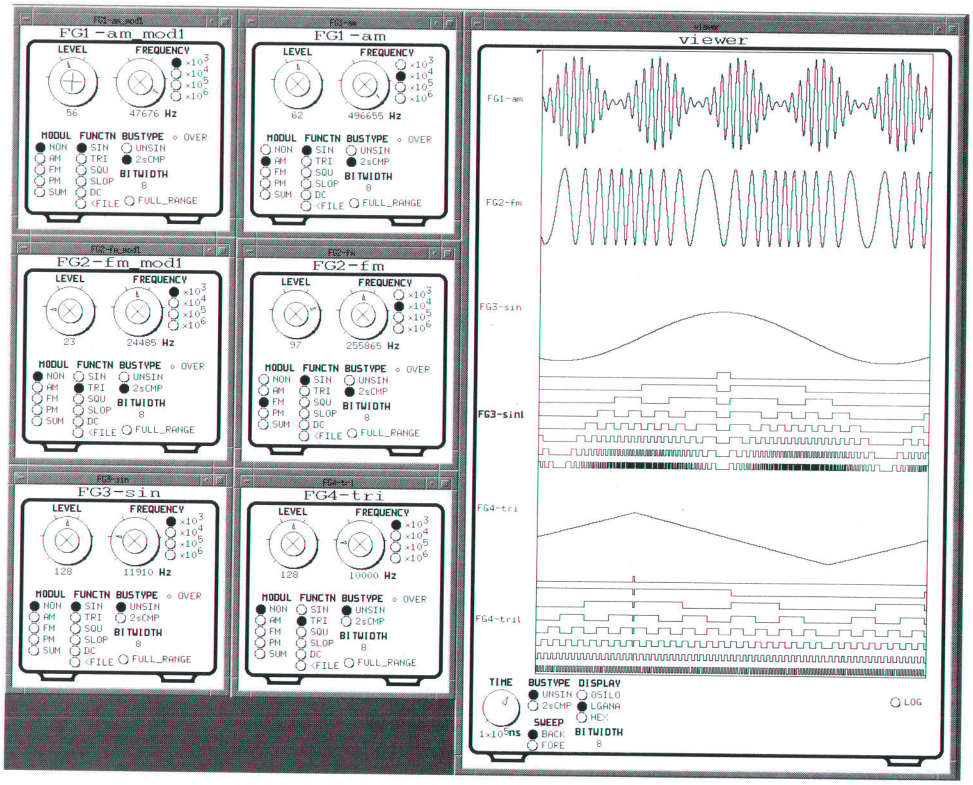
○ 2sCMP

BITWIDTH

9

Fig. 13. The constant generator provides a means for the user to vary inputs to the circuit while the simulation is under way by simply turning a knob. The resolution of the output is automatically determined by the width of the bus to which it is connected. The output can be presented in either twos complement or unsigned integer form.

Data Viewer. The data viewer (see Fig. 14) is a multimode, multichannel data display instrument. Each channel can be individually configured to display data as a conventional logic analyzer, as an oscilloscope, or in hexadecimal format. Each channel can represent the data as twos complement or unsigned. The trace speed is variable and can be optionally controlled by an external sync pulse. The data viewer automatically increases the number of display channels as more input buses are connected to the instrument. Changing the size of the window automatically rescales the data display.


FG1-am_mod1

LEVEL 56

FREQUENCY 47676 Hz ● ×10^3 ○ ×10^4 ○ ×10^5 ○ ×10^6

MODUL ● NON ○ AM ○ FM ○ PM ○ SUM

FUNCTN ● SIN ○ TRI ○ SQU ○ SLOP ○ DC ○ <FILE

BUSTYPE ○ UNSIN ● 2sCMP ∘ OVER

BITWIDTH 8 ○ FULL_RANGE

FG1-am

LEVEL 62

FREQUENCY 496655 Hz ○ ×10^3 ● ×10^4 ○ ×10^5 ○ ×10^6

MODUL ○ NON ● AM ○ FM ○ PM ○ SUM

FUNCTN ● SIN ○ TRI ○ SQU ○ SLOP ○ DC ○ <FILE

BUSTYPE ○ UNSIN ● 2sCMP ∘ OVER

BITWIDTH 8 ○ FULL_RANGE

FG2-fm_mod1

LEVEL 23

FREQUENCY 24485 Hz ● ×10^3 ○ ×10^4 ○ ×10^5 ○ ×10^6

MODUL ● NON ○ AM ○ FM ○ PM ○ SUM

FUNCTN ○ SIN ● TRI ○ SQU ○ SLOP ○ DC ○ <FILE

BUSTYPE ○ UNSIN ● 2sCMP ∘ OVER

BITWIDTH 8 ○ FULL_RANGE

FG2-fm

LEVEL 97

FREQUENCY 255865 Hz ○ ×10^3 ● ×10^4 ○ ×10^5 ○ ×10^6

MODUL ○ NON ○ AM ● FM ○ PM ○ SUM

FUNCTN ● SIN ○ TRI ○ SQU ○ SLOP ○ DC ○ <FILE

BUSTYPE ○ UNSIN ● 2sCMP ∘ OVER

BITWIDTH 8 ○ FULL_RANGE

FG3-sin

LEVEL 128

FREQUENCY 11910 Hz ● ×10^3 ○ ×10^4 ○ ×10^5 ○ ×10^6

MODUL ● NON ○ AM ○ FM ○ PM ○ SUM

FUNCTN ● SIN ○ TRI ○ SQU ○ SLOP ○ DC ○ <FILE

BUSTYPE ● UNSIN ○ 2sCMP ∘ OVER

BITWIDTH 8 ○ FULL_RANGE

FG4-tri

LEVEL 128

FREQUENCY 10000 Hz ● ×10^3 ○ ×10^4 ○ ×10^5 ○ ×10^6

MODUL ● NON ○ AM ○ FM ○ PM ○ SUM

FUNCTN ○ SIN ● TRI ○ SQU ○ SLOP ○ DC ○ <FILE

BUSTYPE ● UNSIN ○ 2sCMP ∘ OVER

BITWIDTH 8 ○ FULL_RANGE

viewer

TIME 1×10^5ns

BUSTYPE ● UNSIN ○ 2sCMP

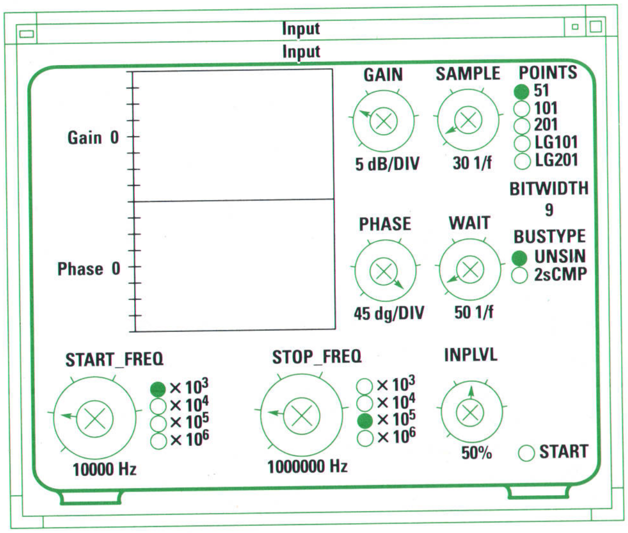
DISPLAY ○ OSILO ● LGANA ○ HEX

SWEEP ● BACK ○ FORE

BITWIDTH 8

○ LOG

Fig. 14. Several function generators connected to a data viewer. The top trace shows an amplitude modulated waveform supplied by the top two function generators. The function generator at the left is supplying the modulation signal for the generator to its right. The second trace is a frequency modulated waveform supplied by the two function generators in the center. The next four traces show sine and triangle waves in both oscilloscope mode and logic analyzer mode.


Input

Input

Gain 0

Phase 0

GAIN 5 dB/DIV

SAMPLE 30 1/f

POINTS ● 51 ○ 101 ○ 201 ○ LG101 ○ LG201

BITWIDTH 9

PHASE 45 dg/DIV

WAIT 50 1/f

BUSTYPE ● UNSIN ○ 2sCMP

START_FREQ ● × 10^3 ○ × 10^4 ○ × 10^5 ○ × 10^6 10000 Hz

STOP_FREQ ○ × 10^3 ○ × 10^4 ● × 10^5 ○ × 10^6 1000000 Hz

INPLVL 50%

○ START

Fig. 15. The network analyzer provides a swept-frequency signal to analyze a circuit's frequency response with respect to both phase and gain.

Network Analyzer. The network analyzer (see Fig. 15) automatically analyzes a circuit's frequency response in terms of both phase and gain. The instrument provides a signal whose frequency is swept between the start and stop frequencies as indicated on the front panel. The scale of the display can be varied, as can the nature of the sweep (linear or logarithmic) and the number of samples to be taken at each step.

Pixel Viewer. The pixel viewer (see Fig. 16) provides the user with a virtual color CRT that can be configured to any geometry and pixel size. There are a number of types of pixel viewers, but they fall primarily into two classes: those that accept a stream of pixels to be written in raster fashion and those that allow individual pixels to be addressed and written.

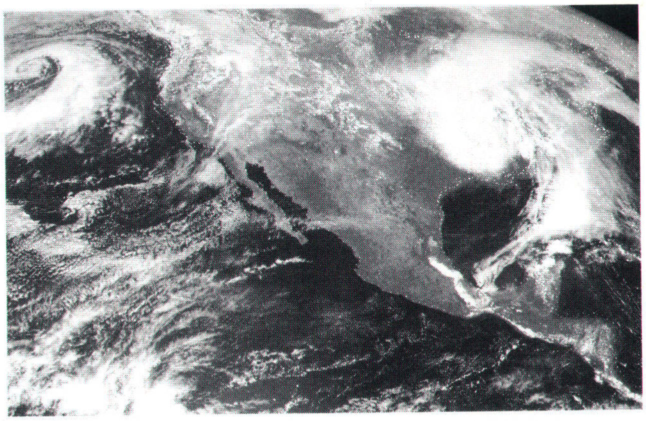
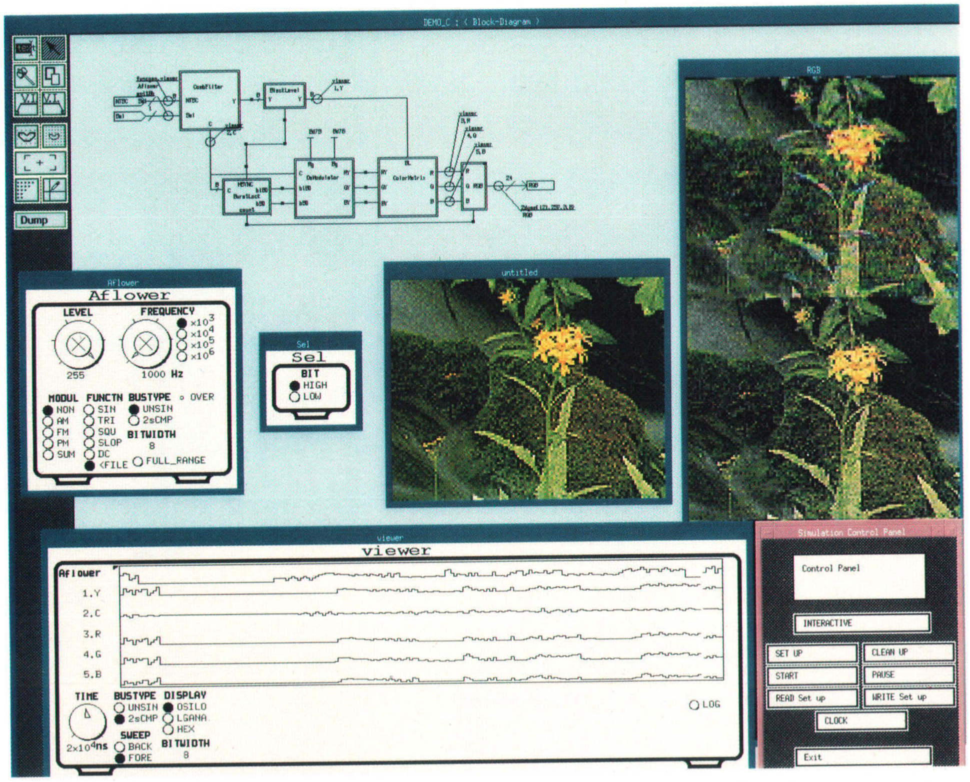


Fig. 16. The pixel viewer provides the user with a virtual color CRT that can be configured to any geometry and pixel size.

Examples

Tsutsuji is now being sold in the Japanese market by YHP. Customers have used Tsutsuji to implement a wide variety of ASICs ranging from digital signal processors to controllers to digital TV systems. The largest design to date has 170,000 gates, although Tsutsuji can easily handle designs of one-half million gates or more. The following examples illustrate how Tsutsuji readily involves the user in the domain of the application.

Television Decoding Filter. Many Tsutsuji customers are in the business of designing television receivers. Fig. 17 illustrates how Tsutsuji can be used to make fundamental design decisions during the earliest stages of design. The example shows an experiment to compare two TV decoder filters. One filter is less expensive to build than the other but produces lower-quality results. Whether the less-expensive filter would be good enough is an aesthetic question that is almost



Aflower

LEVEL 255

FREQUENCY ● ×10^3 ○ ×10^4 ○ ×10^5 ○ ×10^6 1000 Hz

MODUL ● NON ○ AM ○ FM ○ PM ○ SUM

FUNCTN ○ SIN ○ TRI ○ SQU ○ SLOP ○ DC ● <FILE

BUSTYPE ● UNSIN ○ 2sCMP ○ OVER

BITWIDTH 8

○ FULL_RANGE

Sel

BIT ● HIGH ○ LOW

viewer

Aflower

1.Y

2.C

3.R

4.G

5.B

TIME 2×10^4ns

BUSTYPE ○ UNSIN ● 2sCMP

SWEEP ○ BACK ● FORE

DISPLAY ● OSILO ○ LGANA ○ HEX

BITWIDTH 8

○ LOG

Simulation Control Panel

Control Panel

INTERACTIVE

SET UP

CLEAN UP

START

PAUSE

READ Set up

WRITE Set up

CLOCK

Exit

Fig. 17. In this example, Tsutsuji was used to compare two television decoder filters. A design was created that included both filters. During simulation, both filters decoded the same image, producing the two images on the right side of the screen. The designer could then compare them with the original image, in the center of the screen, and choose the most appropriate filter.

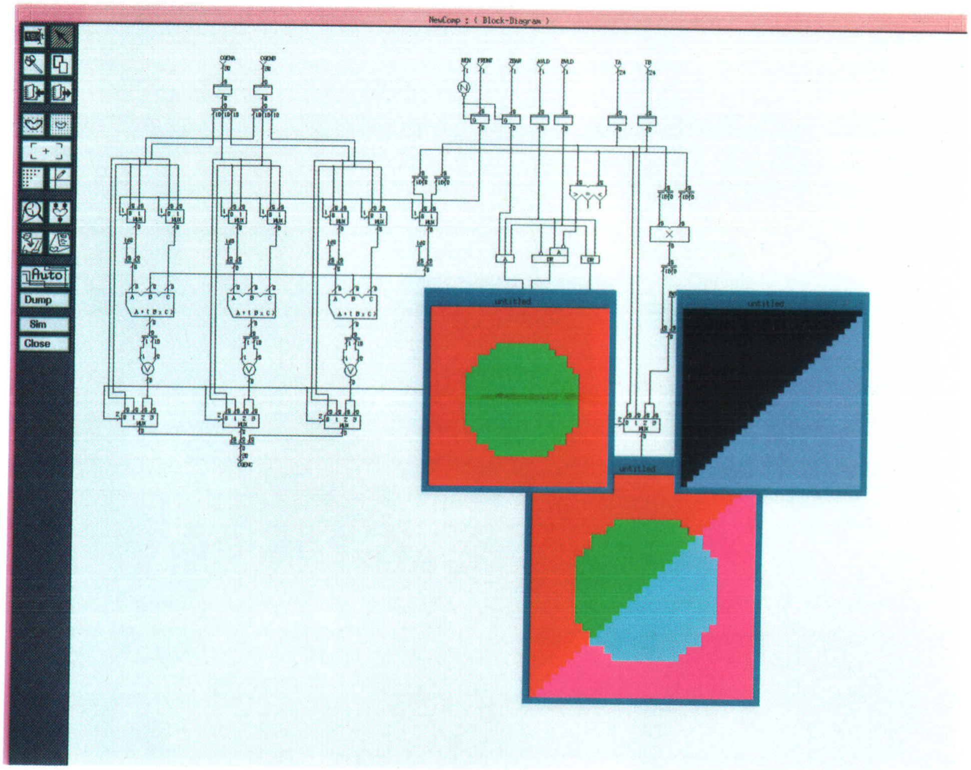


Fig. 18. This screen shows a design for an image processing chip that was designed with Tsutsuji. The designer spent two hours entering the design, which was then automatically synthesized into 8596 CMOS gate array cells in less than a minute. Actual images, appropriate in the image processing domain, were used when simulating the design. The upper two pixel-viewer virtual instruments show the input images; the lower viewer shows the blended image produced by the simulation.

impossible to answer without looking at the images the filter would produce. Tsutsuji, with its friendly simulation environment, provided an ideal means for answering that question.

A design was entered into Tsutsuji that included both filters. A switch, labeled Sel, lets the user switch between the two filters during simulation. The function generator to the left of the switch in this example merely reads the original image from a file and feeds it to the simulation. The image in the center of the screen is the original image before encoding and decoding. The tall pixel viewer on the right displays the output of the simulation. The instrument labeled viewer has been placed in oscilloscope mode and shows the input NTSC television signal, the signal after it is decoded into chroma and luminance (C and Y), and the signal again after it is decoded into red, green, and blue.

The simulation was started with the switch set to select the low-quality filter. The decoded image began filling the output pixel viewer. Once an entire image had been simulated, the high-quality filter was selected and another image was drawn. Once both decoded images were complete, the user could compare them with the original and make a well-informed decision about which filter to build.

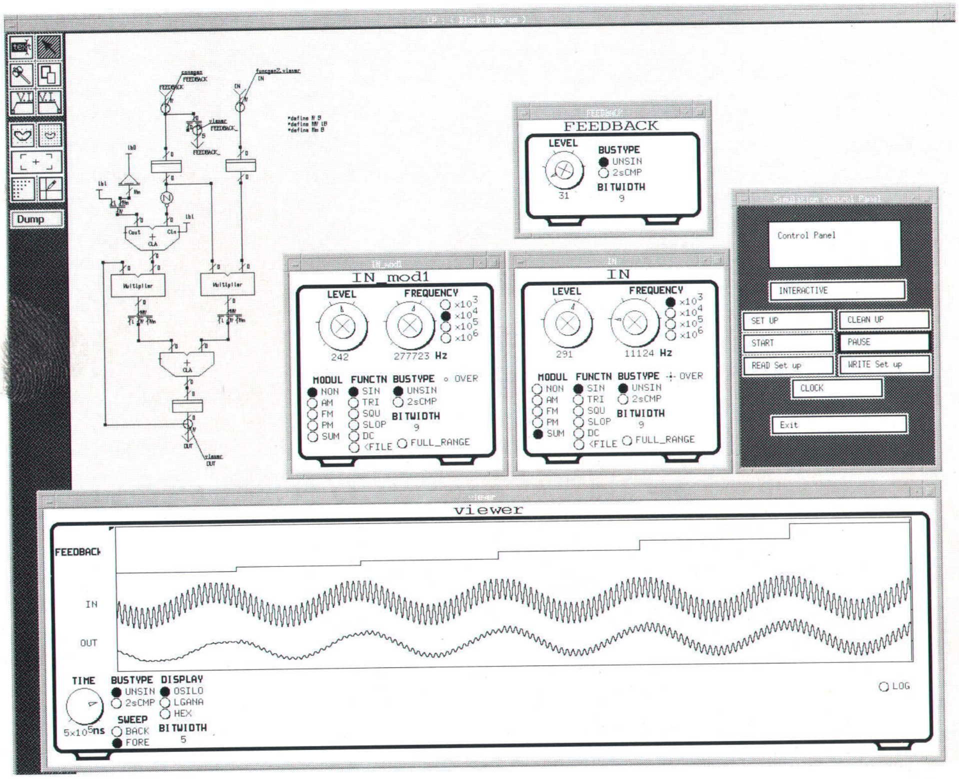
Image Processing ASIC. Fig. 18 shows an image compositor ASIC that was designed using Tsutsuji for an image processing system. The compositor ASIC merges two input images, producing one output image. The images are merged using one of two modes. In the first mode, the input images are treated as though they were transparent, and the output image is a blend of the two images. In the second mode, the input images are considered to be opaque. If two objects in the two input images overlap, then the object that is closest to the viewer is shown in the output image. The image processing system includes a tree of identical compositor ASICs. The tree has much the same function as an individual compositor ASIC except that it combines many images (not just two) into a single image.

Fig. 18 shows part of the compositor design and the result of simulating the design in its blending mode. The simulation inputs and outputs are viewed as images so that the designer will neither waste time interpreting the simulation nor risk misinterpreting it. Three pixel-viewer virtual instruments can be seen. The two upper viewers show the input images and the third viewer shows the blended result. The simulation, which required evaluation of about 5000 gates for each of the 900 pixels in the output image, was completed in less than a second.

The compositor design was entered into Tsutsuji by an inexperienced designer in two hours. The design consisted of approximately thirty high-level modules. The high-level design was synthesized into a design at the generic gate level in twelve seconds. It took an additional thirty-eight seconds to accomplish the following: the design was mapped into a commercial CMOS gate array library, the mapped design was translated into the file format that the gate array vendor accepts, and an exhaustive delay analysis was performed on the circuit. The resulting design uses 8596 gate array cells and 169 I/O pads.

Low-Pass Filter. Fig. 19 illustrates a logic synthesis session that has progressed to the point of logic simulation. The example is that of a simple low-pass filter. Instead of the streams of ones and zeros that are normally associated with logic simulation, we see waveforms—an appropriate form in which to view the input and output of a digital filter.

The illustrated low-pass filter takes a percentage of the previous input and sums it with one minus that percentage



FEEDBACK

LEVEL 31

BUSTYPE ● UNSIN ○ 2sCMP

BITWIDTH 9

IN_mod1

LEVEL 242

FREQUENCY 277723 Hz ○ ×10^3 ● ×10^4 ○ ×10^5 ○ ×10^6

MODUL ● NON ○ AM ○ FM ○ PM ○ SUM

FUNCTN ● SIN ○ TRI ○ SQU ○ SLOP ○ DC ○ <FILE

BUSTYPE ● UNSIN ○ 2sCMP ○ OVER

BITWIDTH 9

○ FULL_RANGE

IN

LEVEL 291

FREQUENCY 11124 Hz ● ×10^3 ○ ×10^4 ○ ×10^5 ○ ×10^6

MODUL ○ NON ○ AM ○ FM ○ PM ● SUM

FUNCTN ● SIN ○ TRI ○ SQU ○ SLOP ○ DC ○ <FILE

BUSTYPE ● UNSIN ○ 2sCMP OVER

BITWIDTH 9

○ FULL_RANGE

Control Panel

INTERACTIVE

SET UP | CLEAN UP

START | PAUSE

READ Set up | WRITE Set up

CLOCK

Exit

viewer

TIME 5×10^5 ns

BUSTYPE ● UNSIN ○ 2sCMP

SWEEP ○ BACK ● FORE

DISPLAY ● OSILO ○ LGANA ○ HEX

BITWIDTH 5

○ LOG

Fig. 19. A simple low-pass filter design example. The function generator on the left provides a high-frequency signal to be added to the low-frequency signal of the function generator on the right. The filter will remove varying amounts of this high-frequency signal as a function of the percentage feedback, which is controlled by the constant generator. The logic simulation is performed at the gate level so the real circuit will perform exactly as observed on the data viewer.

times the current output to form the next output. We can see that the major design parameters are indeed parameters, so the designer can, for example, explore the effects of quantization by changing the input bus width parameter and then resynthesizing the design—a process that takes less than a minute. The other major parameter is the percentage of the previous output used to compute the next output. Rather than laboriously type the constant and one minus the constant for each trial, the designer has added hardware to the circuit to compute these two values. The constant can then be applied from a constant generator and varied in real time while the simulation is progressing.

After the design has been simulated to satisfaction, the final synthesis can be performed. Here, the actual binary constant selected during simulation will be entered. There is no need to remove the superfluous adder. Since both of its inputs are now constants, all of its gates will be removed by the optimizer. The multipliers will also be affected by the optimizer, since each multiplier has a constant as one of its inputs. The final task is then to select a particular technology-specific library and perform the technology mapping. For this example, a nine-bit filter, the initial synthesis resulted in a design of 1401 gates. After mapping and optimization, the design was reduced to 649 gates.

In this example, a designer who was familiar with filter design (but not necessarily familiar with multiplier design) was able to enter and synthesize a design for a low-pass digital filter in about ten minutes. Subsequently, different bit-width designs were explored by simply changing the bus width parameter. To observe the effect of the feedback constant in real time, extra hardware was added to the design to save the designer's time. This hardware did not penalize the design because it was later completely removed by the optimizer. In an hour the designer was able to intuitively explore literally dozens of designs without becoming enmeshed in the intricacies of gate-level design. Essentially all of the designer's creativity and intuition was focused in the application domain.

Conclusion

Tsutsuji is a product from YHP in Japan that provides a set of fast and efficient tools for logic synthesis, simulation, and design visualization. The graphical nature of the human interface allows designs to be expressed quickly by the designer. Rapid synthesis and mapping encourage the designer to explore the design space interactively in search of an optimum system configuration. Applying creativity where it will have the greatest impact, the designer remains focused in the application domain, knowing that optimization and mapping into the chosen technology will be automatic. Designs produced by Tsutsuji are inherently reusable.

Acknowledgments

Without the vision, dedication and hard work of the YSL staff, Tsutsuji would have never been more than a lab curiosity. Listed all too briefly here are their names and principal contributions. Takako Inoue: database maintenance and development of the cell library editing tool. Yoshimasa Jimba: early development work on Tsutsuji. Hiroyuki Kitayama: worked on the simulator compiler and many real designs including DLX-CPU and the H261. Masaya Khono: worked on the parser language operation with Toshiki Osame, participated in the design of the database language and developed the symbol editor. Kazuhiro Maehara: key member who turned the prototype editor into a real product and was leader of the user interface team. Koji Marume: at HP Laboratories for one year, created some high-level modules and participated in the module compiler development. Yoshihiro Matsuda: helped with the early Tsutsuji concept design and designed the first ASIC with Tsutsuji. Satoshi Naganawa: at HP Laboratories for three years, created the module compiler and converted innumerable modules from LISP prototypes to C++ products. Hideki Nonoshita: developed the path analyzer. Yasufumi Otsuru: original member of the YSL team, worked in all aspects of Tsutsuji, including interprocess communication and logic minimization. Miki Ohyama: conveyor of concepts through graphic arts. Takashi Okuzono: developed the text editing portion of the block diagram editor and contributed to the aesthetic presentation of Tsutsuji. Ichirou Sasaki: was in charge of the simulation environment. Norio Tanaka: his clear descriptions made it possible for other people to use Tsutsuji. Yasumasa Teshima: developed the physical mapper. Koji Yamamoto: worked on the VHDL front-end prototype.

At HP Laboratories we would like to thank John Norris for allowing a project to start without a project number, David Horine for obtaining a project number, Ralph Patterson for signing a last-minute trip request to Kurume, and for all these years, a special note of gratitude is owed Dick Lampman for his vision, guidance, and support.

UNIX is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and other countries.

Designing a Scanner with Color Vision

The challenge for personal computer imaging today is to duplicate human color vision, not only in scanners but also in monitors and printers so that colors look the same in all media. The HP ScanJet IIc scanner uses a proprietary color separator design to provide fast, single-scan, 400-dpi, 24-bit color image scanning.

by K. Douglas Gennetten and Michael J. Steinle

The function of a desktop scanner is to digitize an image or a document and send the information to a computer, a facsimile card, or a printer. This allows the digital information to be processed, printed, and stored for archival purposes. A desktop scanner can be used for many different job functions and must be able to scan various types of documents, photographs, line-art drawings, and three-dimensional objects that may be placed on the scanner platen. The wide variety of material that can be scanned presents challenges for the scanning device.

HP ScanJet IIc Scanner

The HP ScanJet IIc scanner is a 400-dot-per-inch (dpi) flatbed scanner with black and white, color, and optical character recognition (OCR) capabilities. It is compatible with PCs and Apple Macintosh computers and with desktop publishing, presentation, and text recognition applications. It offers fast single-pass scanning, easy-to-use software, print path calibration, a legal-sized platen, HP AccuPage technology for text scanning, and low cost. Print path calibration optimizes the quality of the final output by compensating for differences in output devices and software applications. HP AccuPage technology, when combined with a software application that supports it (such as Caere's OmniPage Professional 2.0), uses special page recognition techniques and automatically sets the intensity to improve accuracy on text with nonwhite backgrounds. AccuPage also includes logic that joins broken characters.

The ScanJet IIc provides 8-bit grayscale and 24-bit color scanning capabilities. It uses an SCSI (Small Computer System Interface) for Macintosh computers and a dedicated SCSI adapter for PC-compatibles and MicroChannel PCs. Optimum brightness and contrast settings are selected automatically. Custom scaling is available in one-percent increments. Online help provides reference and tutorial information. An optional document feeder handles up to 50 pages automatically.

HP DeskScan II, the image scanning software included with the HP ScanJet IIc scanner, has a layered user interface for both beginning and expert users. Advanced functions are easily accessed as pull-down menus or floating tools. Image editing software is included, and a live preview feature shows the results of changes immediately on the screen.

Color Science

The experience of color is universal, transcending cultures and oceans. This experience always has one common thread: there are three elements in the experience of color vision. The first element is a source of illumination, the second is an object being illuminated, and the third is a detector to measure the reflected illumination from the object.

Illumination

Humans and many but not all animals see electromagnetic energy falling between 400 and 700 nanometers as *visible light*. Any energy within this range radiating from an object will influence its color appearance. Sources of illumination, whether natural or man-made, are characterized by their *spectral power distribution*, that is, their strength along the electromagnetic energy spectrum between 400 and 700 nanometers. The nature of this spectral distribution can profoundly effect the color of an illuminated object. A common illustration of this is the color shifts that occur under tungsten street lights. An extreme example would be laser light: all objects that are not black are red when illuminated in red laser light. To have a good color observation environment, the source of illumination must be *broadband*, that is, it must contain a relatively flat and broad spectrum of energy over the range of visible light. If any areas of the spectrum are weak or missing, it will not be possible to illuminate those portions of an object's *spectral reflectance* characteristic. The fluorescent bulb in the HP ScanJet IIc is designed with a mixture of phosphors to produce a broad spectrum of light energy.

The Object

Photons from the source of illumination arriving at the object can be affected in one of three ways. They can be transmitted through the object, reflected from the object, or absorbed within the object (and reradiated as heat or, in the case of fluorescence, reradiated as light of a different wavelength). Reflection is most relevant to the human experience of color. Colored objects are characterized by their *spectral reflectance distribution*. A vast variety of spectral reflectance distributions are found in the natural world.

Objects viewed with scanners such as the HP ScanJet IIc are usually in the form of documents. (In the case of the HP ScanJet IIc, a noteworthy exception is three-dimensional objects. The ScanJet IIc's illumination, optics, and single-pass color separation make it unusually capable as a three-dimensional object scanner.) Colors found on documents are usually generated with offset-press inks or photographic dyes. These colorants come in four varieties: cyan, magenta, yellow, and black. With only these four colors to work with, very few of the spectral reflectance curves found in nature can be even approximately reproduced. Fortunately, because of a phenomenon in human vision called *metamerism*, this is not necessary. Without metamerism, any picture containing grass would have to be created with chlorophyll to provide a matching color.

The Detector

In the case of human vision, all of the infinite degrees of freedom found in an object's spectral reflectance distribution are reduced to only three dimensions. This is the root of the phenomenon of metamerism. Because of this, colors can always be described with just three numbers. For example, a color can be described by three numbers representing amounts of red, green, and blue. The same color can be just as precisely and unambiguously described by numbers representing its hue, saturation, and lightness. Any of several other three-dimensional color systems could be used as well.

Like the human vision system, the human hearing system is a spectral waveform processor. Unlike the vision system, however, the hearing system retains all of the spectral content of audible sound all the way to the brain. This provides a very important capacity: when one listens to a chord played on a piano, one can easily discern the individual notes composing the chord. Also, from the character of the sound, it is obviously a piano chord rather than an organ or flute chord played from the same notes. An expert ear can even tell the brand and sometimes the vintage of the piano! In stark contrast, the eye cannot see chords. A white paper illuminated with a yellow light can appear exactly the same as the same paper illuminated with a mixture of green and red light. The spectral content, observable with a scientific instrument, can be radically different while the appearance is identical to a human. It is this mammoth simplification (loss) of information that allows us to reproduce the color of grass green exactly with only four inks or dyes. Unfortunately, there is a catch. This exact match is, strictly speaking, guaranteed under one and only one type of illumination. More on this later.

From Man to Machine

Scanners like the HP ScanJet IIc bring the gift of sight to computers. Producing any color image capture device such as this requires a partial duplication of the human vision system in the form of electronics and optics. The central task in this effort is the accurate description of the human vision system's method of converting spectral energy into three dimensions of color. This was done many years ago. Around 1930, primarily for the incipient color television industry, a group of people were tested for their sensitivity to monochromatic wavelengths over the visible spectrum. Each person adjusted the intensity of three lights until a match of the test wavelength was achieved. A series of such matches produced a set of three curves called the color matching functions. An averaged set became the international standard called the CIE standard observer (see Fig. 1). These curves form the basis of color television and the HP ScanJet IIc.

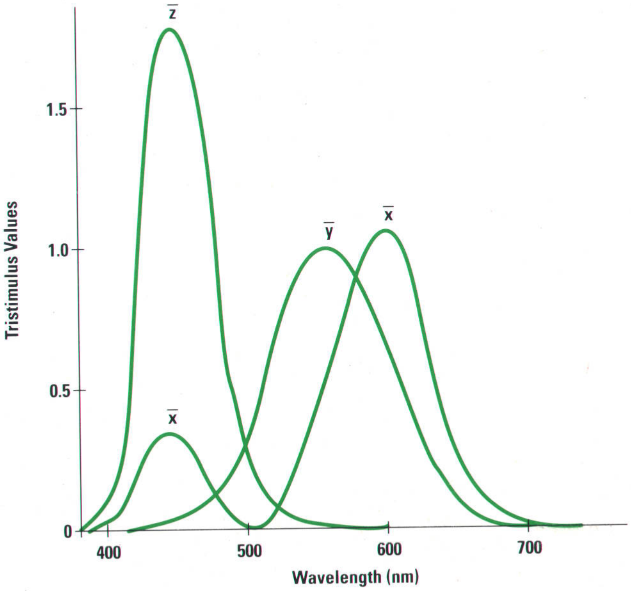


Fig. 1. CIE standard observer color matching curves.

The color matching functions of the standard observer can be converted into a new and equally valid set of three curves by multiplying the original curves by a 3-by-3 matrix. The U.S. National Television Standards Committee (NTSC) adopted one such set of curves for use in color television (see Fig. 2). This NTSC standard is used frequently by the computer graphics industry and was chosen for the design of the HP ScanJet IIc. To achieve a spectral sensitivity matching the NTSC curves, a combination of the spectral characteristics of all the optical elements must be considered. For the ScanJet IIc this includes the document glass platen, the lamp, the lens, three color separation filters, three mirrors, and the photosensitive charge-coupled device (CCD) detector. To duplicate the human color separation process, the net combination of all these elements must produce three color channels that are directly related to the standard observer through a 3-by-3 matrix operation.

The curves shown in Fig. 2 illustrate the ideal camera sensitivities for NTSC color television. Note the presence of several negative lobes. Because of these lobes, a perfect camera would require more than three detectors (adding one for each negative lobe) and in fact the very high-end broadcast cameras often have five or six detectors instead of the three found in home video cameras. The inability to include negative lobes slightly diminishes the accuracy of the color separation process. This degradation also exists for color film. The result is "instrument metamerism": some colors that match when viewed by a human observer do not match when viewed by the instrument, and vice versa.

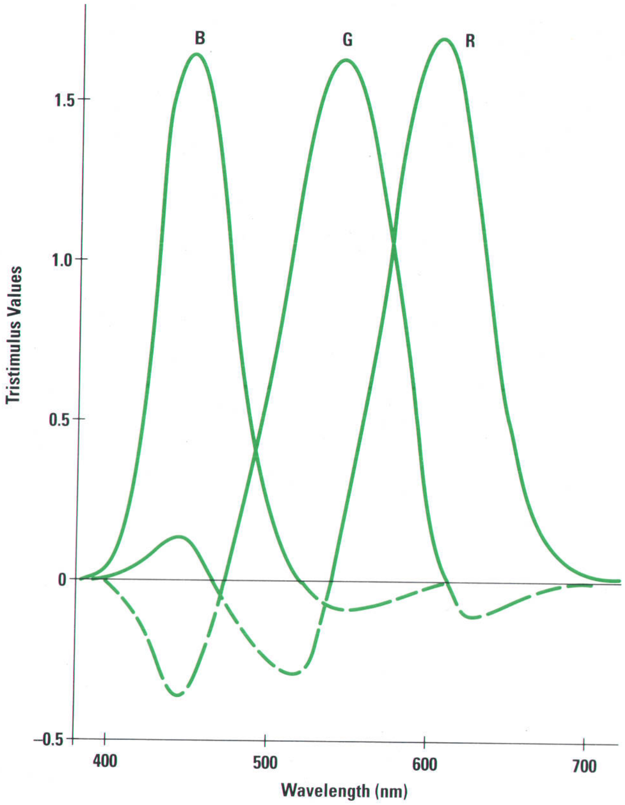


Fig. 2. NTSC color matching curves.

HP ScanJet IIc Color Separation

Of all of the elements along the optical path of the HP ScanJet IIc, the lamp and the filters have the most conveniently alterable spectral behavior, and in the case of the dichroic filters, this is very restricted. Because of the color separation method used (see "Color Separator Design," page 55), each color channel has access to three mutually exclusive bands of the color spectrum. The curves in Fig. 2 (and any other set of color matching curves) contain a great deal of overlap. Some wavelengths are visible to more than one channel. Only a small amount of overlap is possible with the method used in the ScanJet IIc, resulting in a slight degradation of the color performance. However, this color separation configuration has strong advantages in scanning speed and single-pass operation. Fortunately, the degradation made unavoidable by this configuration is small and is minimized through the optimization process described in the next section. The HP ScanJet IIc's color performance competes well with the other desktop scanners in the marketplace.

Measuring and Optimizing

The lamps in the HP ScanJet IIc are fluorescent. They are produced with a custom mixture of phosphors that are specifically designed to aid in the recreation of the NTSC spectral sensitivities. This ability to create custom spectral characteristics (see Fig. 3) helps offset the limitations of the filters. The color separation filters are a dichroic design (see "Color Separator Design," page 55). Their spectral characteristics can be altered primarily by moving the crossover frequencies. They have a fairly square passband performance that does not match the shapes of the NTSC curves very well. However, the combination of the filters and the lamp produces a much closer approximation of the desired result. Extensive measurement and characterization of the scanner was performed using a spreadsheet model of all the spectral characteristics throughout the optical path. This model was used to optimize the choice of lamps and filter crossovers. Additional optimization was achieved through the selection of a carefully determined default 3-by-3 matrix which is applied to all scanned pixels. This 3-by-3 matrix provides a closer approximation of NTSC color.

Color Matching

The low-cost color scanners and printers available today contribute to a growing demand for accurate color image reproduction. Users of desktop systems having color image capture, display, and printing capabilities are demanding better color image reproduction fidelity. Many factors contribute to the challenges of color matching.

Scanner Limitations. Scanner inaccuracies are most commonly caused by imperfect color matching functions in the color separation process. Another less obvious source of error is that typical document scanners provide their own light source. Any color scan from such a device can only give color measurement data for documents viewed under that particular light. Once the original document's spectral reflectance is reduced to the three dimensions of color, it cannot be reversed. The necessary information required for accurately determining the document's color under a different light source is irretrievably lost. This is true even for a scanner with perfect human-like vision and is unavoidable without increasing the number of dimensions (sensor colors) within the scanner. The result is that all color matches are conditional. They may, and often do, fail when the viewing conditions are changed. The only way to produce an invariant match—one that holds regardless of viewing conditions—is to capture and reproduce not the color of the original but its spectral reflectance. Scanners and color printers are not capable of this today.

Monitor Limitations. Color monitors produce a wide range of colors by mixing three different colored phosphors. Especially in a well-lighted office, these monitors are limited in their ability to recreate the range of visible colors. First, a three-gun monitor, no matter how perfect, can never recreate the colors of the rainbow or any of a large region of other saturated colors. Second, because of the surrounding

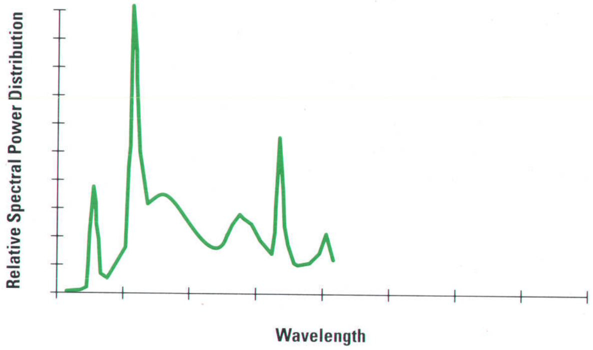


Fig. 3. Spectrum of the lamp in the HP ScanJet IIc scanner.

light, a typical monitor cannot produce a good black. Third, such a monitor has difficulty producing a pure, bright white. These last two points can easily be illustrated with a computer monitor and a laser-printed page. When the printed page is held near the monitor, it will typically appear brighter and whiter than the monitor's white. If the monitor is turned off (to reveal its blackest black) the black toner on the page will typically be much darker than the black of the monitor. An accurate reproduction of a monitor display of a white page with a black and yellow square would produce a printed page with many dots in the "white" areas, magenta dots in the "yellow" areas, and white dots in the "black" areas. This is rarely the desired result. WYSIWYG (what you see is what you get) is definitely not desired. Instead, it's "what you want is what you get" that is desired.

Printer Limitations. Further compounding the problems of color matching is the color gamut limitations of low-cost color printers (a printer's color gamut is the set of all of the colors it can print). Many displayable and scannable colors fall outside of the capabilities of most printers. Areas of images that contain these colors must be modified to accommodate the printer limits. Once again, the most accurate reproduction is often not the most desirable.

Managing all of these color matching issues and limitations is a very complex task. However, advancements continue to be made, and there is reason to hope for steady improvement in the disquieting situation that exists today on the PC.

Color Separator Design

The objective of a scanner is to digitize exactly what is on the document that is being scanned. However, this is not a realistic goal because it would require a CCD (charge coupled device) detector with an infinite number of pixels and a lens with a modulation transfer function (MTF) equal to 1.0, which does not exist. (Modulation transfer function is a measure of the resolving power or image sharpness of the optical system. It is analogous to a visual test that an optometrist would use to determine a human eye's resolving power.) Most important, the scanner user does not require an exact reproduction of the original because the human eye does not have infinite resolving power. The HP ScanJet IIc scanner is designed to obtain very fine-detailed images for a variety of color and black and white documents and objects that are typically scanned.

To design a high-performance, low-cost desktop scanner required a team effort involving the disciplines of optical, mechanical, electrical, firmware, and software engineering. Some key decisions that affected the design architecture were resolution (dots per inch), gray level depth, optical scanning resolution, scan time, product size, image quality, and product cost.

After the product was defined, a color separation technique was decided upon. This affected all the engineering disciplines involved in the product design. Various color separation techniques are used in the image reproduction industry. A few of the common techniques are:

- Colored dyes deposited on the CCD substrate. Used in camcorders, scanners, and color copiers.

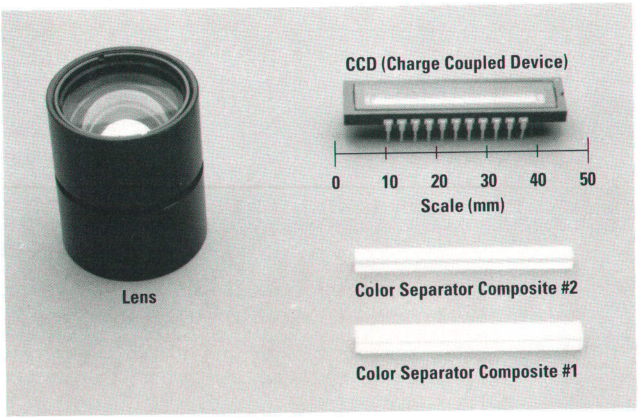

CCD (Charge Coupled Device)
0
10
20
30
40
50
Scale (mm)
Lens
Color Separator Composite #2
Color Separator Composite #1

Fig. 4. Lens, CCD detector, and color separator composites.

- Rotating or translating red, green, and blue filters. Used in scanners.
- Red, green, and blue flashing lamps. Used in scanners.
- Beam-splitting prisms with multiple CCD sensors. Used in scanners.

To meet the performance and cost goals for the HP ScanJet IIc, a new HP proprietary color separation method was developed and implemented. The initial development was done at HP Laboratories in Palo Alto, California and the technology was transferred to the Greeley Hardcopy Division in Colorado for continued development and implementation.

The color separation system consists of a lens, two color separators, and a CCD detector as shown in the photograph, Fig. 4. Each color separator is a laminated assembly as shown in Fig. 5. Each assembly is made of three glass plates that are bonded to each other with a thin layer of optical adhesive. Red, green, and blue reflective coatings are deposited on the glass before lamination. Specifically, dichroic coatings (2 to 3 μm total thickness) are deposited onto the glass substrates. Good spectral performance is obtained using dichroic coatings, resulting in an accurate colorimetric device.

The distance between colors at the CCD detector (see Fig. 5) depends on the thicknesses, index of refraction, and angles of the glass plates separating the red, green, and blue reflectors. The plates are thin glass substrates that have tightly controlled flatness, thickness, and angle tolerances. The thin plates are laminated to a thick baseplate, which provides mechanical rigidity and flatness. During the multilayer dichroic coating process the thin plates are distorted, but laminating them to the thick plate restores the flatness of the reflective surfaces. The first laminated plate has the color order of blue, green, red while the second plate has the order of red, green, blue. This configuration equalizes the optical path lengths to ensure simultaneous focus for all three colors. The order of coatings was selected to maximize spectral efficiency and simplify the coating process.

Each color component is focused onto a CCD row, each row consisting of 3400 imaging pixels (additional pixels are available and are used for light monitor control and dark voltage correction). The CCD generates a voltage signal that is proportional to the amount of light incident on the detector. This signal is processed and then digitized. Having a

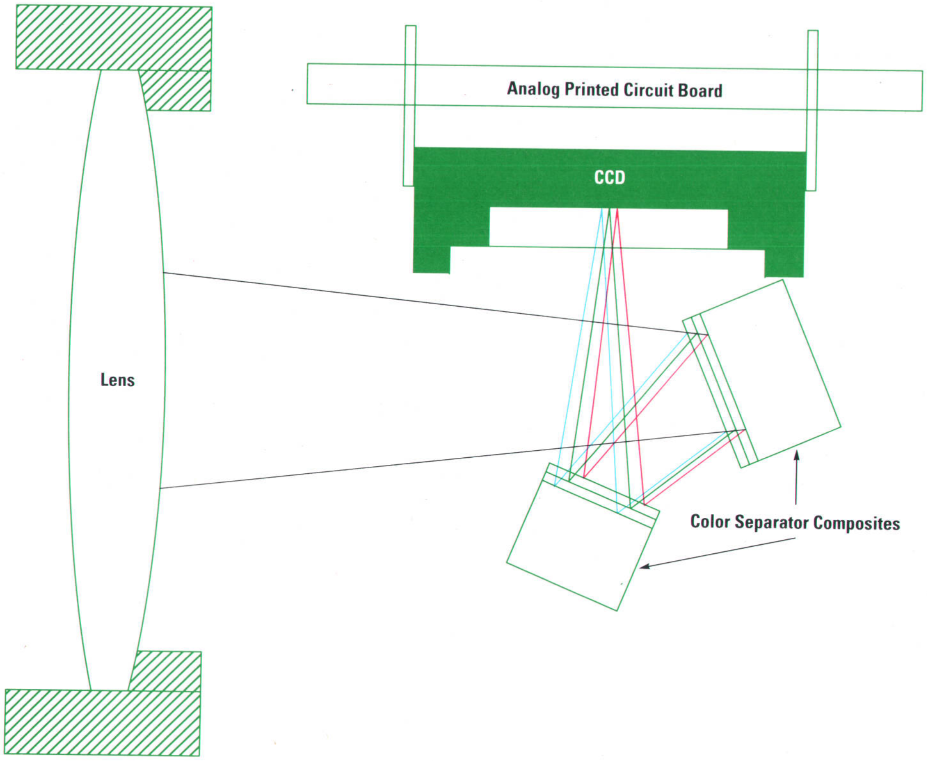


Fig. 5. HP ScanJet IIc color separation method.

CCD that integrates all three rows and senses all three colors simultaneously yields a single-pass scanner with excellent image quality. This color separation method also provides high-performance scanning capability in a small integrated package that is cost-effective and manufacturable at high volumes.

A layout of the optical system showing the light path is shown in Figs. 6 and 7. Fig. 6 also shows the solids model of the carriage, which includes the dual lamp assembly, three mirrors, the lens, the color separator, and the CCD assembly. The carriage is translated along the length of the document glass platen by a stepper motor drive system and a belt that is connected to the carriage. In Fig. 7 the light path is drawn for several rays from the scanned region. The lens is a six-element double Gauss design that yields a very good MTF.

The optical system was designed and evaluated using a commercially available optical design program. Unlike many other engineering disciplines such as finite element analysis,

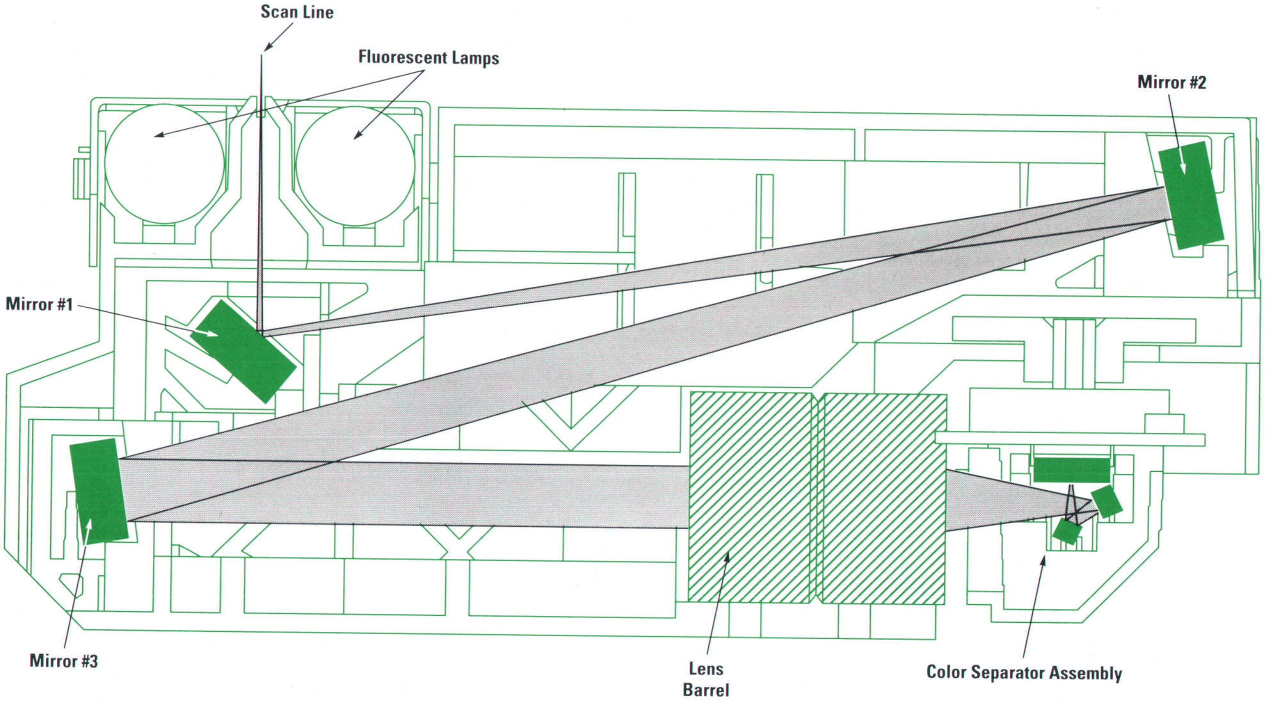


Fig. 6. Solids model of the HP ScanJet IIc optical path and carriage.

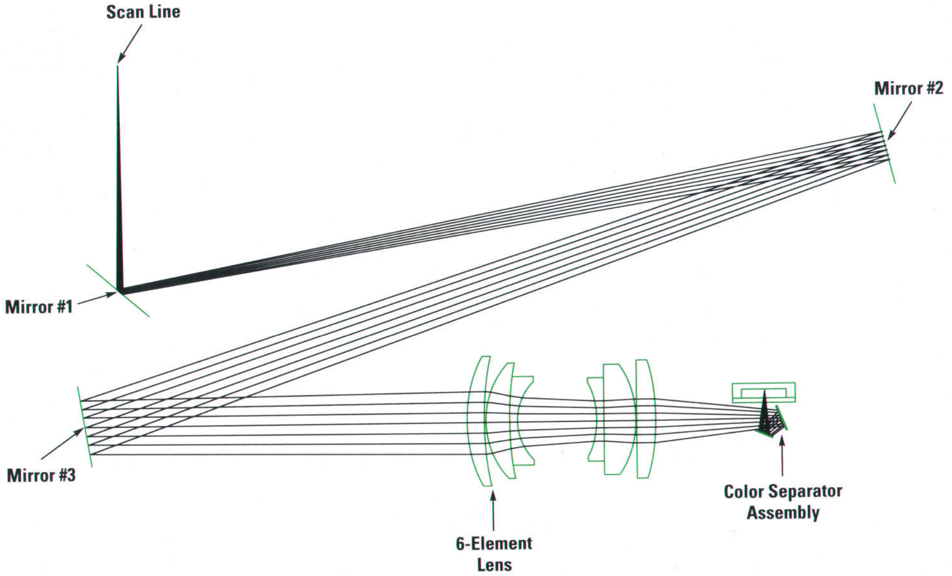


Fig. 7. Ray trace of the optical path (one color only).

for which it is more difficult to predict accurately how a fabricated prototype will perform, the performance of an optical system can be calculated very accurately. The effects of tolerances on the optical system were also modeled to ensure that the product could be manufactured at high volumes. Modulation transfer function (image sharpness) was evaluated for tolerances such as lens centering, tilt, accuracy of lens radii, index of refraction, and color separator flatness and thickness. A typical plot of modulation at 105 line pairs per inch (object side of the lens) as a function of position across the page is shown in Fig. 8. Modulation is the sharpness of the image at a specific line pair frequency, whereas MTF is the sharpness of the image as a function of line pair frequency. Fig. 8 demonstrates that the resolving power of the scanner varies only slightly with the location on the glass platen. This data includes the effect of the CCD's modulation:

$$\text{Modulation} = \text{Modulation}_{\text{Optics}} \times \text{Modulation}_{\text{CCD}}.$$

For fabricated optics tested on an optical bench, the measured through-focus data agreed closely with the calculated results.

To achieve precise optical alignment, custom tooling was designed and fabricated to meet production goals. Translational alignment of ±10 μm is required for focus and for centering the light path on the CCD. The alignment tools, consisting of translational and rotational stages, are controlled with an HP Vectra 386 computer and software that consistently gives optimized optical alignment.

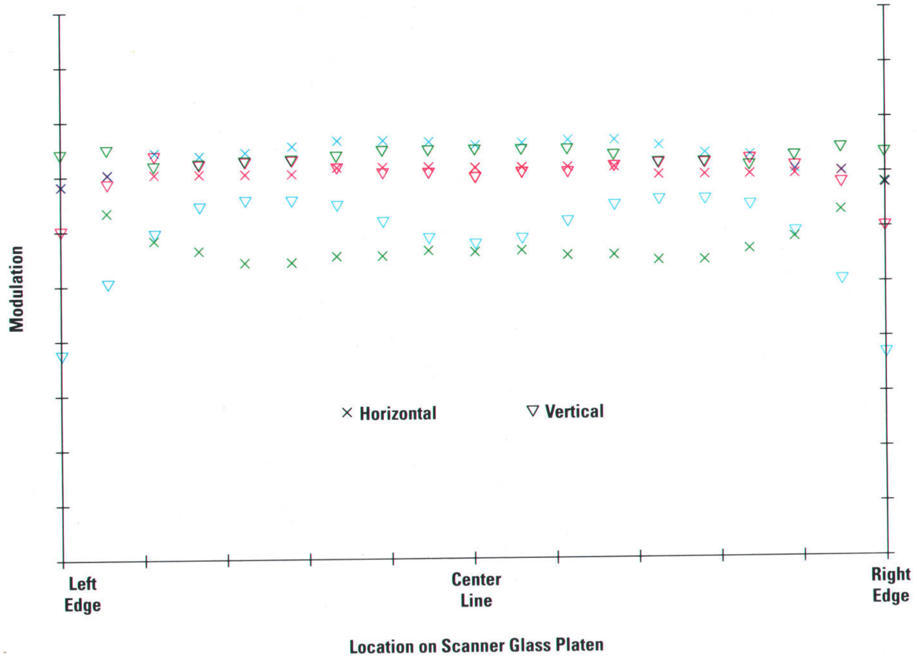


Fig. 8. Modulation for horizontal and vertical lines at 105 line pairs per inch (object side of lens) as a function of platen position for red, green, and blue.

Acknowledgments

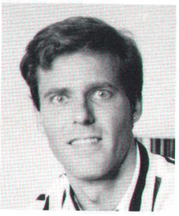
The development of the HP ScanJet IIc involved many engineering disciplines. Only the people who contributed to the scanner colorimetry and optics are mentioned here. Apologies to anyone who was inadvertently left off the list. Project managers: Corry Meyer and Jerry Bybee. Section manager: Dean Buck. R&D manager: Duncan Terry. HP Laboratories: Kent Vincent, Hans Neumann, and Ricardo Motta. Manufacturing optical alignment tooling and software: Bill Elder, Duane Starner, Kurt Spears, and Ray Cole. R&D: Bob Sobol, Steve Webb, and Dave Boyd. Materials engineering: Bruce Zignego, Walt Auyer, Robert Marquez, and Dave Bohn. Manufacturing process: Tom Hacker. Optical testing: John Wunderlich and Kevin Martin. Optical performance evaluation: Bob Gann. Typing manuscript: Erin Hsu. Custom assembly of optics: several production assembly operators. Independent optics consultant: Dr. Walter Mandler. Model shop: Chuck Fenwick, Dean Geilenkirchen, Greg Wolf, and Roger Lucas.

Authors

August 1993

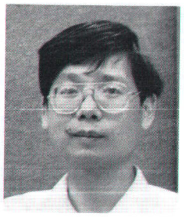
6 High-Efficiency LEDs

Robert M. Fletcher



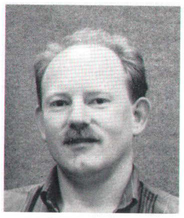
R&D engineer Bob Fletcher has been with HP's Optoelectronics Division since 1985 and has worked on aluminum indium gallium phosphide LEDs for most of that time. His responsibilities included developing wafer and die fabrication processes and transferring those processes from R&D to manufacturing. He is named as an inventor in three patents on LED device structures and is coauthor of 17 papers related to lasers, LEDs, and materials growth and characterization. Bob was born in Knoxville, Tennessee and attended Rice University, from which he received a BSEE degree in 1978. Continuing his studies at Cornell University, he received an MSEE degree in 1981 and a PhD in the same subject in 1985. Bob's outside interests include running, bicycling, playing classical guitar, and remodeling his house.

Chihping Kuo



C.P. Kuo is an R&D engineer and specialist in III-V crystal growth and characterization. He was born in Taipei, Taiwan and attended the University of Utah, receiving a PhD degree in electrical engineering in 1985. He joined HP's Optoelectronics Division the same year, and his work has focused on high-brightness LEDs. He's the author of 25 papers related to epitaxial growth of III-V materials and is named as inventor in three patents on AlInGaP LEDs. C.P. and his wife have one child. His leisure activities include music and bridge.

Timothy D. Osentowski



An R&D engineer at HP's optoelectronics division, Tim Osentowski is a specialist in epitaxial growth of materials for high-brightness LEDs. He was born in San Jose, California and completed work for his BSEE degree in 1981. With HP since 1978, he is the coauthor of more than ten articles on epitaxial growth of III-V materials and is named as an inventor in three patents related to AlInGaP LEDs. Tim is married and has two children. He's active in Little League baseball and likes golf and bowling.

Jiann Gwo Yu

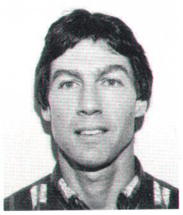
Jiann Yu has been with the Optoelectronics Division since 1981. His duties as an R&D engineer have included process development for various types of materials for LED displays. Jiann studied physics at Chengkung University in Taiwan (BS 1967) and at the University of Wyoming (MS 1972) before completing work for a PhD degree in materials science from the University of California at Los Angeles in 1978. Before coming to HP, he was a researcher at Northrop, Jet Propulsion Laboratories, and Tektronix. Jiann enjoys hiking and bicycling.

Virginia M. Robbins

Born in Camden, New Jersey, Virginia Robbins has a BEE degree from the University of Delaware (1981), and MSEE and PhD degrees from the University of Illinois (1983 and 1987). She joined HP Laboratories upon graduating from Illinois and has worked on crystal growth of AlGaInP by organometallic vapor phase epitaxy. She's currently involved in crystal growth of III-V semiconductors. Her work on LED transparent windows has resulted in two patents and she's the author or coauthor of several papers on III-V semiconductor materials and properties. Virginia is married and lists hiking, camping, and running as leisure activities.

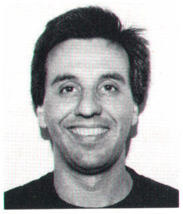
15 HP Task Broker

Terrence P. Graf



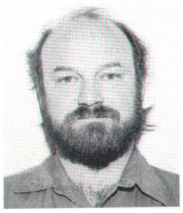
A project manager at HP's Open Systems Software Division, Terry Graf has been with HP since the Apollo Computer acquisition in 1989. Past HP projects include work on the Apollo Domain operating system and the OSF/1 operating system from the Open Systems Foundation. He was the project manager for the HP Task Broker development team. Terry has a BS degree in electrical engineering from Stevens Institute of Technology (1975) and an MBA degree from Northeastern University (1987). His other professional experience includes work on operating systems at Wang Laboratories. He's a member of the IEEE.

Renato G. Assini



A native of Boston, Massachusetts, Ron Assini has been with HP since 1989, when HP acquired Apollo Computer. A software engineer at HP's Open Systems Software Division, he developed the graphical user interface for HP Task Broker. On past projects, he was a peripheral diagnostic engineer and contributed to the development of the Apollo Domain operating system and to OSF/1 from the Open Systems Foundation. His other professional experience includes a stint at Honeywell/Bull as a test technician and diagnostic engineer. Ron received a BS degree in computer technology from Northeastern University in 1989. He is married, has two children, and is active in youth sports.

Edward J. Sharpe

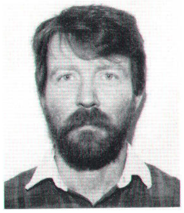


Ed Sharpe studied computer science at the State University of New York at Buffalo (BA 1977) and at the University of Southwestern Louisiana (MS 1980). He has been with HP since 1989 and has worked on several operating systems, including Apollo Domain, Mach, and OSF/1. He's currently an engineer at the Open Systems Software Division and was responsible for centralized configuration development for HP Task Broker. Ed is a member of the ACM and the IEEE.

John M. Lewis

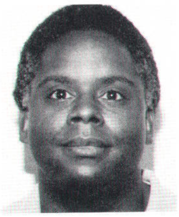
Software engineer John Lewis was born in Springfield, Massachusetts and attended Iowa Wesleyan College (BS mathematics 1974) and the University of Iowa (MS computer science 1977). He was a software engineer for Data General and Avatar Technologies before joining Apollo Computer. He worked on personal computer integration products at Apollo both before and after the HP acquisition in 1989. John is now in the Open Systems Software Division and contributed to software development and design for HP Task Broker. His outside interests include fly-fishing and skiing. A certified professional ski instructor, he conducts instructor training clinics and is a supervisor at a local ski area.

James J. Turner



Engineer James Turner helped develop the graphical user interface and internal library for HP Task Broker. He received a BS degree in computer science from Boston University in 1981 before joining Apollo Computer, where he was a hardware designer at the time of the HP acquisition in 1989. He is now a member of the Open Systems Software Division. He's named as the inventor in a patent related to high-speed memory access. James enjoys skiing, golf, and playing ice hockey.

Michael C. Ward



Mike Ward is an R&D specialist at the Open Systems Software Division and has been with HP since 1989. For the HP Task Broker project, he was involved in the development of the hardware, software, and network configuration required to support the product quality plan. Previous HP projects include installing, maintaining, and repairing R&D hardware and network equipment, installing and maintaining the Apollo Domain operating system, and testing operating systems before product release. In previous professional positions, he was an engineering and production assistant and manager. Mike has a BA degree in theater from the University of New Hampshire (1976) and an AA degree in computer science from Hesser College (1987).

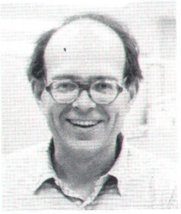
23 HP-RT Operating System

Kevin D. Morgan

Section manager Kevin Morgan joined HP in 1980 and has worked on several versions of the HP real-time executive (RTE) operating system. He was a member of the marketing team for HP RISC-based computers at the time of their introduction, and was a project manager for a real-time interface card for RISC systems. He's currently working in the Measurement Control Systems Division. Born in Portland, Oregon, he studied computer science at the University of California at Santa Barbara (BS 1980) and the University of California at Berkeley (MS 1984). He's the author of a magazine article on real-time systems. Outside of work, he's a musician, playing guitar in a rock and blues band, and also enjoys surfing. Kevin is married.

31 HP-RT Operating System

George A. Anzinger



With HP since 1969, George Anzinger is a systems software specialist for the HP-RT operating system in HP's Measurement Control Systems Division. He was born in Waukegan, Illinois and received a BSEE degree from the University of Wisconsin at Madison in 1968 and an MSEE degree from Stanford University in 1969. While at HP, he has worked on DACE, a data acquisition and control executive system, and on various functions of RTE, an HP real-time executive system. For the HP-RT project, he managed the development of the interrupt system and driver code and services. George is married and has two daughters. In his spare time he helps his wife run a video store, and he's also interested in electric automobiles and plans to build one.

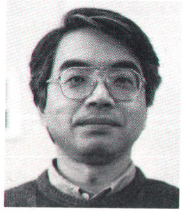
38 Logic Synthesis System

W. Bruce Culbertson

A member of the technical staff at HP Laboratories, Bruce Culbertson came to HP in 1983 and has worked on PC software products, on an experimental real-time computer network, and on ICs and design software for an HP PA-RISC processor. For the Tsutsuji project, his contributions include developing key algorithms and data structures and writing the simulation compiler and topology plotter programs. Born in Oakland, California, he studied mathematics at the University of California at Davis (BS 1973) and at the University of California at San

Diego (MA 1976). He also completed work for an MS degree in computer science from Dartmouth College in 1983. His work on the emulation of three-dimensional objects on a two-dimensional computer screen resulted in a patent, and he's a member of the IEEE and the ACM. Bruce likes skiing, especially cross-country ski racing, playing and collecting ethnic music, mountaineering, and bicycling.

Toshiki Osame

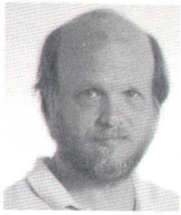


Born in Kagoshima, Japan, Toshiki Osame received a BS degree in physics from Kouchi University in 1977 and an MS degree in the same subject from Osaka University in 1979. He joined the YHP Systems Laboratory in 1986 and now is an R&D engineer at YHP at Kurume. He contributed to the development of a PLD design system and for the Tsutsuji system designed compiler architectures and developed the LDF language parser, the interface for the module generator, and the C simulation model. Toshiki is married and has three daughters.

Yoshisuke Otsuru

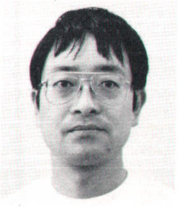
Yoshisuke Otsuru was project manager for the Tsutsuji project at Kurume Systems Laboratory in Japan. Born in Kurume, Fukuoka, Japan, he earned a degree in mechanical engineering from Kurume Technical College in 1969, and joined HP in 1985. He's the author of a series of review articles on ASIC design. Yoshisuke is married and has two children. His leisure activities include watching movies and playing golf.

J. Barry Shackleford



Barry Shackleford is a principal project scientist at HP Laboratories. He initiated the project that resulted in the Tsutsuji logic synthesis system and was the R&D project leader. He is presently investigating new computer structures for compilable hardware. Born in Atlanta, Georgia, he completed work for a BSEE degree from Auburn University in 1971 and for an MSEE degree from the University of Southern California in 1975. Before coming to HP in 1981, he worked for Hughes Aircraft and Amdahl. He has worked on a variety of other projects at HP, including a Kanji computer terminal. He is named as an inventor in two patents related to chip scan methods and cellular arrays and in four pending patents on cellular-arrayed computation structures. He's also the author of two articles on neural networks and a member of the IEEE. Barry speaks Japanese and enjoys Japanese food and culture. He hikes in the hills near HP every day and is currently spending most of his free time remodeling his home in Portola Valley, California.

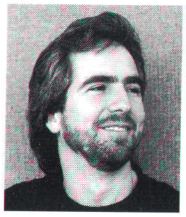
Motoo Tanaka



Motoo Tanaka joined Yokogawa-Hewlett-Packard in 1984 and has held several R&D positions. His past projects include contributions to a PLD design system and a PLD link for an electronic design system. For the Tsutsuji system, he was project manager of the user interface team and did the conceptual design and prototyping of the user interface and circuit editor. Currently, he is a technical customer support specialist. Motoo was born in Tokyo, Japan and received a bachelor's degree in electronic engineering in 1984 from the University of Electric Communications. He's married and has a son and daughter. His outside interests include fishing and classical guitar—he's a member of the Kurume Guitar Ensemble.

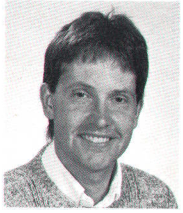
52 Color Scanner

K. Douglas Gennetten



Douglas Gennetten started at HP in 1978 as an electrical engineer. He has worked on several magnetic disk and tape products, earning two patents on data separator and phase-locked loop design. In 1989, he completed work for a degree in imaging science and printing technology from the Rochester Institute of Technology. He contributed to the design of the HP ScanJet IIc from its original conception. In his spare time, Douglas dreams of working part-time at HP to become a "starving artist." He has developed an aesthetically pleasing computer-generated sundial that maintains accurate local time to within 30 seconds year-round. He transfers the design to bronze or stone and hopes to have installations around the globe someday.

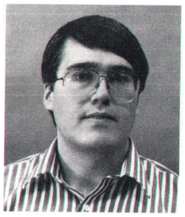
Michael J. Steinle



An R&D engineer at the Greeley Hardcopy Division, Mike Steinle joined HP in 1984. He designed and developed the imaging optics and illumination system for the HP ScanJet IIc and collaborated with the manufacturing staff on the carriage assembly. His current assignment is to design and develop a next-generation optical system. Mike was born in Galena, Illinois and attended Augustana College, from which he received a BA degree in physics in 1981. At Purdue University, he studied mechanical engineering and received a BSME in 1982 and an MSME in 1984. He's the coauthor of four articles related to the work he did on blood flow through a heart valve prosthesis while he was at Purdue. His work has resulted in four patents on color separator design and scanner optical systems. Mike is married and has two daughters. He volunteers at a local youth center and is active in his church. His leisure activities include tennis, bicycling, hiking, attending sports events, and socializing with family and friends.

62 Mechanical Design

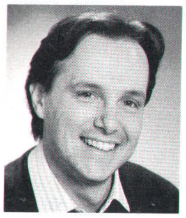
Brad Clements



Brad Clements received his BSME and MSME degrees from Brigham Young University in 1978 and 1979. After joining HP's Desktop Computer Division in 1979, he worked in materials engineering and manufacturing before moving to R&D. He has designed mechanical packaging for the HP 9000 Model 217, the company's first HP-UX workstation, for HP-HIL products, for the HP 9000 Series 300 workstations, and for the HP 9000 Models 745i and 747i industrial workstations. Brad is named as an inventor in three patents related to these products. Born in Idaho Falls, Idaho, he is married and has five children.

68 Laser Control Algorithm

Franco A. Canestri



Franco Canestri was born in Novi Ligure, Italy and received an undergraduate diploma in science from the Scientific College in Genoa in 1974. He completed work for a PhD degree in biophysics from the State University of Genoa in 1979 and taught mathematics and physics in secondary schools before serving in the Italian army. For several years, he was an IBM systems engineer and at the same time was an assistant fellow at the National Cancer Institute of Milan. In 1984, he joined HP's Böblingen Computer Division as a business manager for the HP 1000 computer family, then transferred a year later to the Medical Products Group Europe as a critically ill patient monitoring specialist. He is currently an application and technical support specialist for cardiology products in Europe. He focuses on surgical applications in his research on medical lasers. He is the author of ten papers on lasers in medicine and biophysics. Franco is married and has two children. He enjoys skiing, playing squash, languages, travel, music, and literature.

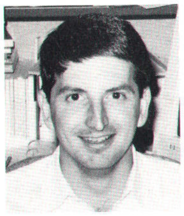
73 Online Defect Management

David A. Keefer

A software engineer, Dave Keefer has contributed to software tool development for the software quality department at HP's Boise Printer Division. He contributed to the implementation of the HP defect management system tool and to design enhancements and is now developing a next-generation DMS user interface. Dave was born in Reading, Pennsylvania and completed work for his BS degree in mathematics from Boise State University in 1988. He's now doing graduate-level work for a degree in software engineering from National Technical

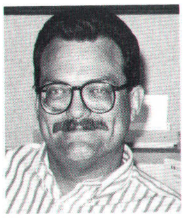
University. With HP since 1980, he has also worked at Fairchild Semiconductor. Dave is married and has two sons. Before becoming a software engineer Dave was a radio announcer, an endeavor he continues today as a hobby by providing narration for films such as HP training videos on disk products and doing some local radio commercials.

Brian E. Hoffmann



Brian Hoffmann was the architect and implementer of the original defect management system developed at HP's Boise Printer and Network Printer Divisions. Previous projects at HP include software test and software quality tools development for HP LaserJet printers. Brian was born in Detroit, Michigan and attended the University of Michigan, from which he received a BS degree in chemical engineering with a process modeling emphasis in 1982 and an MBA degree in information systems in 1988. Before joining HP in 1989, he was an engineer at Ford Motor Company, did design work at Apple Computer, and developed a UNIX user interface and other tools for Andersen Consulting. He's currently developing firmware for a future product. His professional interests include database architecture and numerical analysis. Brian is married and has two daughters. He enjoys camping, hiking, and woodworking.

Douglas K. Howell



Doug Howell is the software quality engineering manager at the Boise Printer Division. He was the project manager for the DMS project and manages a team of software quality engineers who are responsible for developing and supporting test process automation and software quality management tools for HP LaserJet products. He joined HP in 1982 at the Desktop Computer Division in Fort Collins, where he was a software quality engineer and manager. Doug was born in Rushville, Indiana and graduated from Indiana University in 1979 with a triple major in mathematics, chemistry, and German. He completed work for an MS degree in statistics from Colorado State University in 1983 and also did graduate-level work in mathematics at the Eberhard-Karls-Universität in Tübingen, Germany. He is named a certified quality engineer with the American Society for Quality Control and is a member of the American Statistical Association and the IEEE. In addition, he's the author of several papers and training classes on software metrics, software quality engineering, statistical quality control, and methods for customer surveys. Doug is married and his outside interests include restoring old sports cars, collecting antique guns, and gourmet cooking.

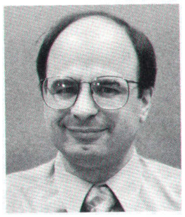
85 Productivity Gains with C++

Timothy C. O'Konski

Born in Oakland, California, Tim O'Konski attended the University of California at Berkeley, from which he received a BA degree in computer science in 1976. He joined HP's Santa Clara Division in 1984 and is now a software designer at Tandem Computers. His HP experience includes developing user interface software for the Santa Clara division, supporting C++ tool development for Corporate Engineering, and working on the design and development of C++ SoftBench in the Software Engineering Systems Division. Previously, he worked on operating system development at Texas Instruments and application software at Apple Computer. He's the author of a Byte magazine article on reusable functions and a member of the IEEE and ACM. Tim is married and has two sons. He coaches youth soccer teams and is active in his church. An avid rose gardener, he also enjoys photography, sailing, and family-related activities.

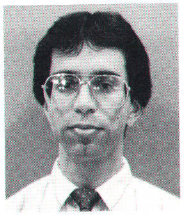
90 Real-Time Design Tool

Joseph M. Luszcz



A project manager in HP's Imaging Systems Business Unit, Joe Luszcz was born in Ware, Massachusetts and studied electrical engineering and computer science at Worcester Polytechnic Institute and at Northeastern University (BSEE 1973 and MSEE 1983). With HP since 1973, he has worked on a number of electrocardiography and ultrasound imaging products, and has managed software development projects for the HP SONOS 1000 cardiovascular imaging system. He is currently leading a software engineering team that is developing a reusable software architecture for imaging products. Joe is active in youth sports and a local cub scout pack. He is married and has five children and says family life occupies most of his time, but also enjoys softball and is learning GUI programming on his home computer.

Daniel G. Maier



A software engineer at the Imaging Systems Division, Dan Maier joined HP in 1989. He was born in Rochester, New York and is a graduate of Rensselaer Polytechnic Institute (BS computer science, 1987). For his first HP project, he helped develop an on-line quantitative analysis package for the HP SONOS 1000 ultrasound system. He later worked on the team that developed the software architecture for ultrasound systems and wrote end-user applications. Previously, he developed software for testing graphical display device drivers at the CalComp Company. Outside HP, Dan tutors grade school children in mathematics. His other outside interests include softball, basketball, skiing, golf, and music.

Mechanical Considerations for an Industrial Workstation

Besides being a compute and data processing engine, a workstation in an industrial and measurement environment must be mechanically designed to handle the special requirements of these environments.

by Brad Clements

The HP 9000 Models 745i and 747i are entry-level industrial workstations. These systems are designed for test and measurement, industrial process control, and electronic testing applications. Both machines are based on HP's PA-RISC version 1.1 architecture,[1] and they both run the HP-UX* 9.0 operating system. Except for dimensions and EISA and VME slots, both machines provide the same features. Fig. 1 shows a rear view of the Model 745i and 747i workstations.

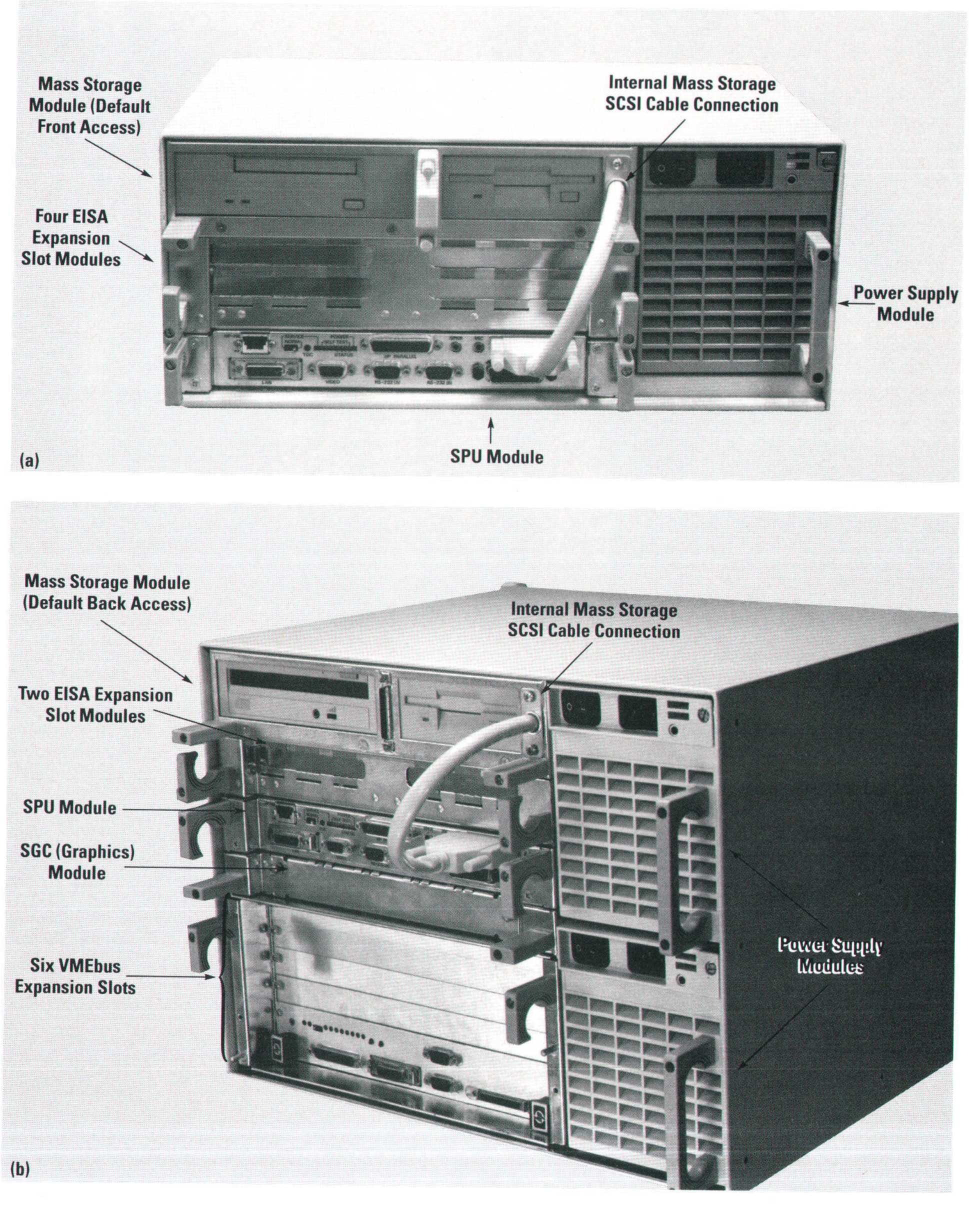

Mass Storage Module (Default Front Access)
Internal Mass Storage SCSI Cable Connection
Four EISA Expansion Slot Modules
Power Supply Module
SPU Module
(a)
Mass Storage Module (Default Back Access)
Internal Mass Storage SCSI Cable Connection
Two EISA Expansion Slot Modules
SPU Module
SGC (Graphics) Module
Six VMEbus Expansion Slots
Power Supply Modules
(b)

Fig. 1. Rear views of HP 9000 Series 700i industrial workstations. (a) Model 745i. Overall size 176.75 mm high by 425.45 mm wide by 412.6 mm deep (6.97 inches by 16.75 inches by 16.2 inches). (b) Wallmounted Model 747i. Overall size 310.15 mm high by 425.45 mm wide by 412.6 mm deep (12.21 inches by 16.75 inches by 16.2 inches).

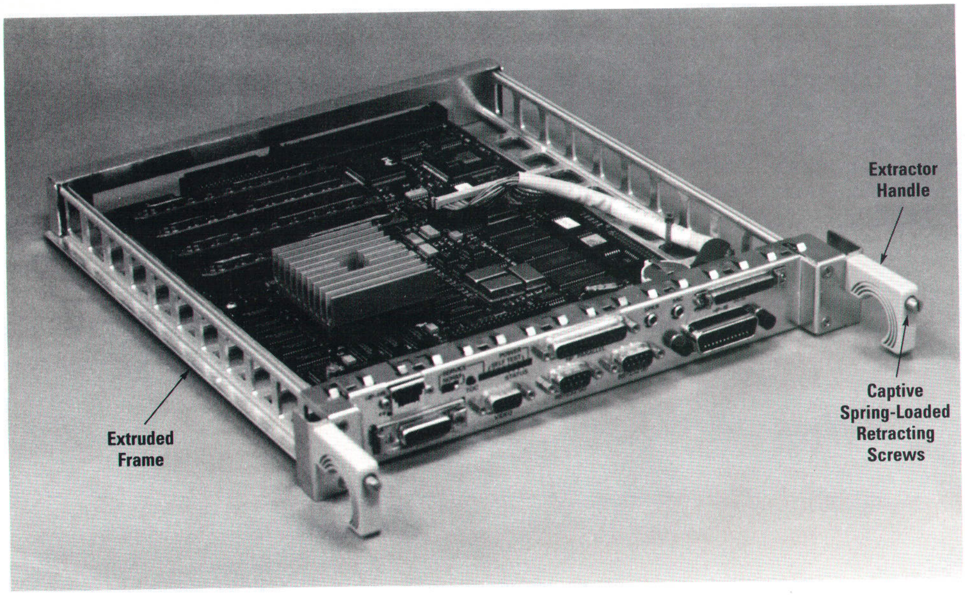


Fig. 2. CPU brick showing the extractor handle.

Background

At the beginning of the investigation phase for the industrial workstation project, a team from R&D and marketing set out to answer the question "what makes an industrial workstation different from a standard workstation?"† Dozens of customers in the measurement and industrial automation markets were visited to help us understand their needs that go beyond the features provided in HP's line of standard workstations. This article addresses the mechanical design aspects of the differences between standard and industrial workstations, and the design strategy we used to meet the needs of customers in the industrial marketplace who use or could use engineering workstations.

Serviceability

Unlike standard workstations, industrial workstations are intended to be incorporated in large, very complex manufacturing processes that produce products worth extremely large amounts of money per hour. The cost of downtime demands the highest level of serviceability. Trade-offs for cost that compromise serviceability cannot be made. Our goal was to provide access to all service-level components in less than three or four minutes.

All service-level components in the Model 745i and 747i industrial workstations including the backplane can be removed and replaced from the cable end of the computer while the computer chassis remains mounted in the rack. This feature sets a new standard for serviceability in this industry. To make the serviceable modules, or *bricks*,†† easy to remove, an extractor handle was developed which holds a captive spring-loaded retracting screw (see Fig. 2). The handle provides a trigger grip for the index finger and a fulcrum surface for the thumb when removing adjacent bricks. The handle also provides a surface to push on while seating the bricks. Regulatory compliance dictated the use of a tool to remove all bricks. The captive screw, which is housed in the handle, visually pops forward to indicate to the operator that the brick is unfastened. Once the bricks are removed an internal wall (see Fig. 3) swings up to unlatch so that it can be taken out of the cabinet to allow the customer to remove the backplane by undoing a single captive fastener located on the backplane.

Connectivity

In addition to the robust core I/O capabilities offered by HP's standard workstations, the Models 745i and 747i provide an HP-IB interface as part of the core I/O. To provide I/O functionality that goes beyond that offered as core I/O, expansion slots are provided. The number of slots requested for industrial workstations is not only greater than for standard workstations, but the types of I/O slots are mixed. Besides the core I/O, the current HP standard workstations only provide EISA slots, which support several I/O protocols.[2] In addition to supporting EISA slots, the Model 747i

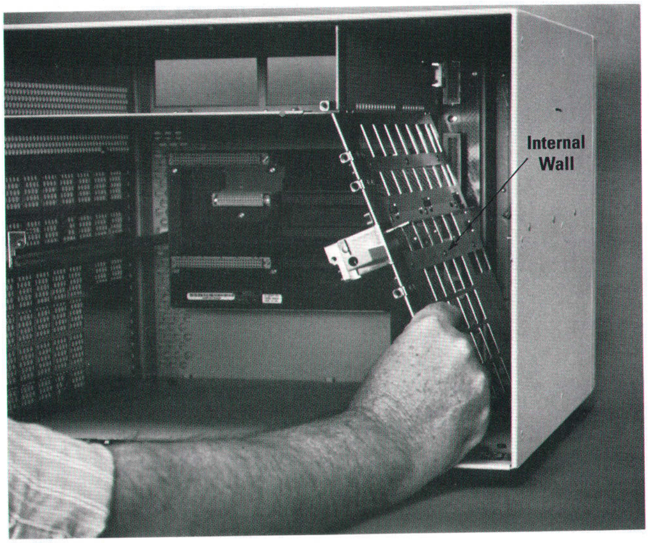


Fig. 3. Gaining access to the Model 747i backplane by removing the internal wall.

† A standard workstation is one that is typically used for program development or running application programs (e.g., CAD/CAM, desktop publishing, etc.).

†† A brick is the term we use for all the modules designed for the Model 745i and 747i workstations.

also supports VMEbus. The package for these machines was designed to be large enough to be able to house the larger cards such as VXIbus cards.†

Support Life

Support life is a very important consideration to the industrial automation customer. Once an industrial workstation has been designed and installed into a factory process it is rarely replaced or upgraded for reasons other than loss of support. Support life is not something that is designed in, but rather a promise or commitment made to customers by HP. The current standard workstations are supported for five years while the Models 745i and 747i carry a 10-year commitment. To reflect a long support life, the industrial design of the Models 745i and 747i has a much plainer and timeless look (see Fig. 4) than the new line of standard workstations.

Reliability

In many standard workstation applications the hardware becomes obsolete long before physically wearing out because of reasons such as the availability of lower-cost machines or machines with faster graphics engines. With industrial workstations this may not be the case because certain items like the fan may not have the same 10-year or even 20-year life that a factory installation may have. For example, extensive testing was done on fan bearing systems to select the best fan for the Models 745i and 747i, but the life expectancy of the fan is still not greater than the service life of the workstation. Thus, the power supply carries a fan-tachometer signal and an overtemperature signal, and is serviceable. More details relating to fan and airflow reliability are discussed later in this article.

† As of this writing VXIbus cards are not yet supported in the HP 9000 Series 700i machines.

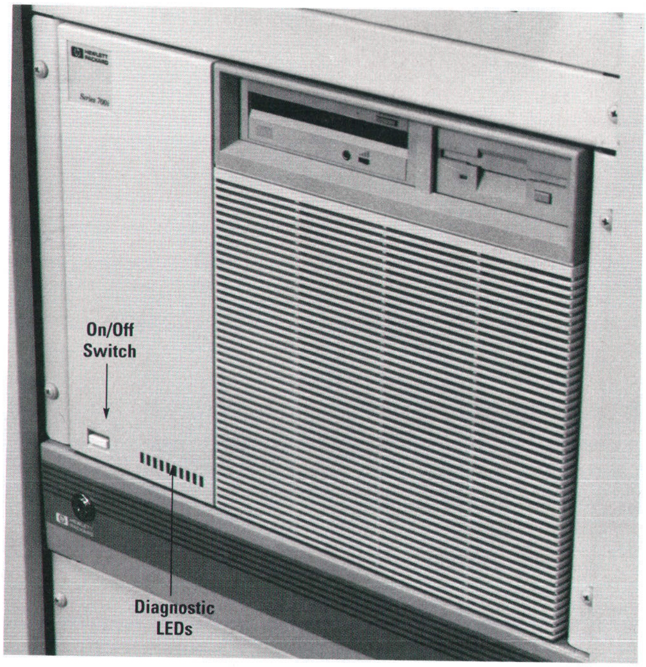


Fig. 4. Rackmounted Model 747i with noncable end out.

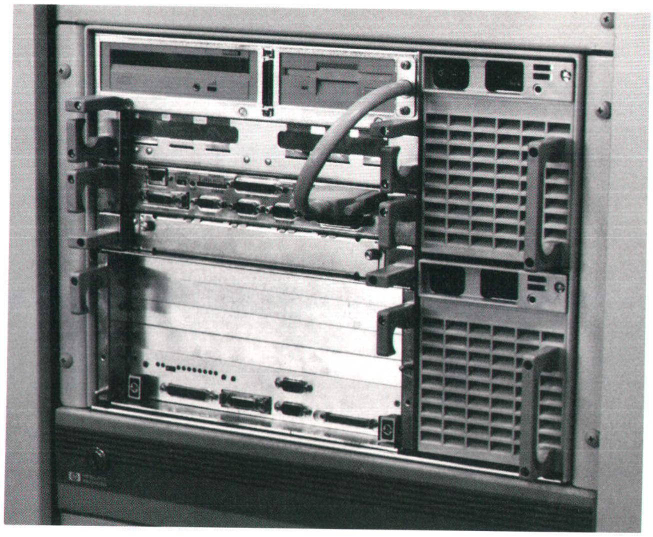


Fig. 5. Rackmounted Model 747i with cable end out.

Graphics

In a typical standard workstation configuration only one large color display needs to be supported because the user is able to access multiple applications using windows. However, in some industrial automation environments, industrial workstations are required to support several large graphics displays. For example, in a control room application large monitors are used to replace walls full of critical instrument gauges. The user or control room operator needs to monitor more gauge images than can be seen on one monitor screen without paging through windows. Windows are still needed for less critical gauges and other operations.

Front-to-Back Reversibility

For measurement automation customers, the business end or user interface end of the Models 745i and 747i is the noncable end of the package (see Fig. 4). All the cables and clutter are hidden in the rear of the machine inside of the rackmount cabinet, which has an access door in the back. User-accessible mass storage bays, an on/off switch, and diagnostic LEDs are located at the front end of the machine, which is the most cosmetic surface of the product.

On the other hand, the industrial automation customer typically wants the cable end of the machine to be the user interface end of the product, with the diagnostic LEDs, on/off switch, and user-accessible mass storage bay also located at the cable end of the machine (see Fig. 5). The Models 745i and 747i were designed to allow HP manufacturing to configure the computer to meet the needs of both the measurement automation and the industrial automation customer. Front-to-back reversibility is provided by redundant on/off switches, redundant diagnostic LEDs, and a mass storage brick that allows user-accessible devices to be located at either end of the product.

Mounting Options

Standard workstations are designed to live in an office environment with the workstation cabinet sitting under a monitor on a desktop or as a minitower on the floor beside the desk. The industrial workstation is required to live in rackmount and other mounted environments. The Models 745i

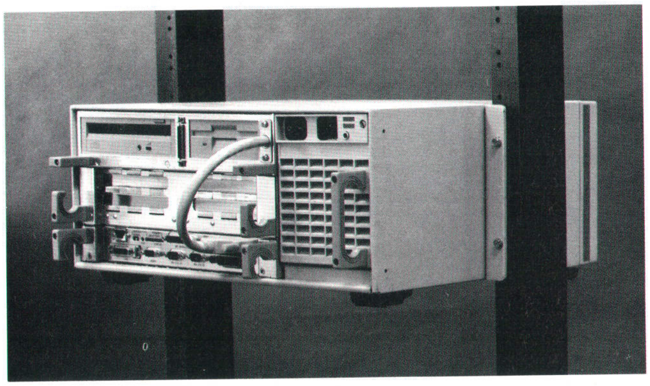


Fig. 6. Mastmounted Model 745i.

and 747i can be mounted in a variety of different configurations. They can be rackmounted from the cable end, rackmounted from the noncable end, stacked on a bench with other HP products, wallmounted with cables facing out from the wall, or mastmounted close to the center of mass of the product (see Fig. 6).

Package Form Factor

In a rackmount environment, package height is always important to instrument and measurement automation customers, but perhaps more important to industrial automation customers is the package depth. The Models 745i and 747i are designed to fit inside a 450-mm (17.7-in) deep wallmounted cabinet with the door closed. With the front bezel removed the distance from the mounting wall to the I/O connector surface is 355 mm, leaving a 95-mm depth for cables. The height of the package was driven by the nearest even number of rackmount units that a 120-mm fan and line filter stack would fit in. With feet removed the Model 745i is four EIA (Electronics Industries Association) standard rack units (177 mm) high and the Model 747i is seven EIA standard rack units (310.4 mm) high. The width of the package is 425 mm to allow nonrackmounted stacking on a lab bench with other standard HP 425-mm-wide instruments.

Airflow Management and Acoustics

The HP acoustic noise goal for office environment products is 50 dBa maximum sound power level. Standard workstations struggle to meet this goal while not making thermal compromises. Industrial workstations can be found in control room or factory-floor environments which can be warmer than a typical office. The variety of mounting options provided by the Models 745i and 747i introduce airflow inlet constraints not required of standard workstations. To provide more thermal margin at higher temperatures with constrained airflow inlets, the 50 dBa goal was compromised. The Model 745i noise level is about 54 dBa and the Model 747i noise level with two fans is about 57 dBa.

The Models 745i and 747i incorporate a negative pressure airflow design. Unlike a positive pressure airflow design, which allows airborne particulates to be filtered out through an inlet filter, the negative pressure system has no filter. Small inlet filters fill with airborne particulates in a relatively short time, greatly reducing the volume of air that moves through the product. Experience has shown that these small filters do not get cleaned as often as required and lead to system reliability problems. Rather than filtering dust, the negative pressure design passes most dust through the product. The dust that does collect over time inside the product is far less detrimental than a clogged filter. For extremely dusty environments the product should be housed in an enclosure that provides air filtering on a scale that can adequately and reliably filter airborne particulates. The negative pressure approach offers some additional benefits. First, a much larger inlet area is possible which reduces total airflow impedance through the product. Second, an uninterrupted airflow zone in front of the fan introduces more laminar airflow to the fan blades, which reduces acoustic noise. Finally, airflow is more uniform. Having more options for inlet locations provides better airflow rationing throughout the product.

When viewed from the cable end of the product, the main air inlet is on the left side of the product (see Fig. 7). In an industrial automation installation the left side typically has far

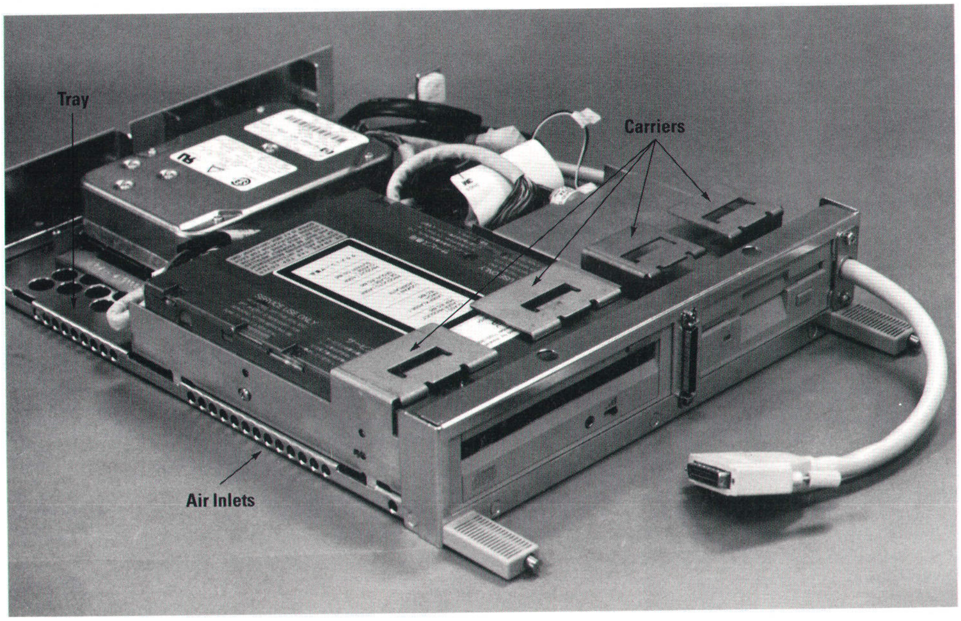


Fig. 7. Mass storage brick with user-accessible devices located at the cable end. Also shown are the air inlets and the carriers that hold the mass storage devices in place.

fewer cables than the right side. This relatively small number of cables on the left side of the product creates little airflow impedance.

In addition to the inlet holes on the left side, inlet holes are provided on the front of the product. The front holes are redundant, allowing the air inlet on the left side to be partly restricted as in a very tight rack installation with little plenum space on the sides. Air flows across the bricks and into the power supply. In an industrial automation installation, the cables that come into the system rack and lie along the right side of the product can be so numerous that airflow through them can be difficult. Therefore, the air exhaust designed into the Models 745i and 747i is out the cable end of the product through the power supply (see Fig. 8).

The power supply is equipped with a temperature sensor that is located near the exhaust fan. This sensor controls the fan speed and is located downstream in the airflow path so that the fan will speed up when the system is heavily loaded, the ambient air is relatively warm, or the inlet is partially restricted. The airflow through the Model 745i is a generous 56 ft^3/min at low speed and 70 ft^3/min at high speed. The Model 747i with two fans moves 105 ft^3/min of air at low speed and 132 ft^3/min at high speed.

In the Model 747i, which has two power supplies and one sensor for each supply, each sensor can also sense when the fan associated with one of the power supplies is not operating properly. When this happens, the operating fan will be sped up, pulling air through the power supply with the defective fan. This should extend the life of the power supply with the defective fan until the controlled process can be shut down in a graceful and less disastrous manner or until control can be passed to a redundant computer.

Brick Strategy

The wide range of measurement and industrial automation customer needs could not be met with just one product. Therefore, we had to develop a strategy to offer a high degree of flexibility for product features. In an ideal world, the best approach to providing different product features would be to design a family of subassemblies, or bricks, which could be mixed and matched in many different configurations. Each brick would adhere to standard size constraints such as width, depth, incremental height units, and electrical interconnect standards. Conceptually the OEM customer would be able to select the number, type, and mix of I/O slots, the number and type of graphics display interfaces, the number and type of mass storage devices, and the number and type of CPU options.

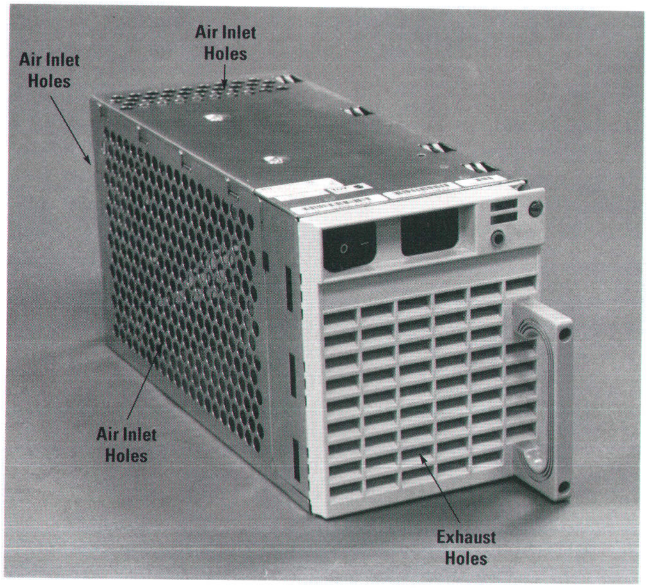


Fig. 8. Power supply module.

For the Model 745i and 747i workstations, the width of the standard brick was driven by the width of two EISA cards laid side by side. The maximum depth of a brick was driven by the length of an EISA card. The standard brick height increment concept was abandoned to allow the products to fit into a smaller package while adhering to EIA standard rackmount increments. The electrical interconnect standard was also abandoned because of physical connector space, connector cost, and high insertion forces. Flexibility for future upgrades was traded off for greater serviceability and lower cost.

Industrial and measurement automation customers rarely upgrade a system after it is installed. Therefore, rather than designing a standard package with optional expanders that carry the added cost of box-to-box interconnect and make the removal of the backplane in the rack impossible, an approach of using standard bricks housed in a variety of different sized chassis was implemented. Each brick has the same backplane.

The Model 745i uses a 4U (four EIA instrument rack units) high box and holds a CPU brick, a four-slot EISA brick, a mass storage brick, and a power supply brick (see Fig. 1a). The Model 747i uses a 7U package which holds a CPU brick, a two-slot EISA brick, an SGC (standard graphic connect) brick, a six-slot VMEbus brick, and two power supplies (see Fig. 1b). The boxes contain two internal walls that support the card guides, and a structure to support the bricks. These walls can be separated from the chassis, making it possible to design other versions of walls quickly. This feature allows different versions of the industrial workstations to be designed for OEM customers. The versatility offered by the walls allows a shorter time to market for future products and reduces the development cost of redesigning an entire package. The backplane, which provides power and bus signals between bricks, is unique for each product developed.

CPU Brick. The HP PA-RISC processor delivers more than enough processing for the vast majority of customers in the industrial and measurement markets. However, customers do want HP PA-RISC machines for the expected support life. Standard 16M bytes of SIMM ECC (error correction code) RAM with optional configurations up to 128M bytes is supported. The core I/O includes HP-HIL, parallel, two serial ports, audio in and out, SCSI, AUI (access unit interface) LAN, HP-IB, and onboard 1280-by-1024-pixel graphics memory. The CPU brick is housed in an aluminum extruded frame to provide additional mechanical board support during insertion, to protect surface mount components on the underside when outside the product, and to offer a rugged industrial appearance and feel. Fig. 2 shows the CPU brick.

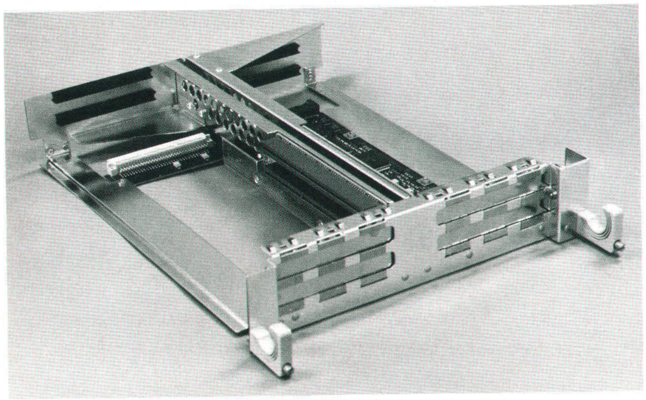


Fig. 9. EISA card brick with four slots.

EISA Brick. To save space in the product the EISA I/O cards are oriented horizontally (see Fig. 9). The structure that supports the cards along with the converter circuits is easily removed for service or upgrades. Easy access to EISA I/O cards is a feature that adds to the competitiveness of our workstations in the industrial marketplace. Almost all of the PCs used in the industrial marketplace require the user to remove the workstation cabinet from the rack and then open a clamshell case to service or upgrade I/O cards.

Mass Storage Brick. The removable tray that holds the mass storage devices is structurally reinforced so that the mechanical vibration frequency response is high. The tray is firmly supported at one end by three tight-toleranced pins and at the other by two captive threaded fasteners. This solid foundation approach required no additional vibration mounts beyond those designed into the individual mass storage devices by the manufacturer. This approach not only is lower in cost for the majority of customers, but provides a significantly more rugged system. However, customers with systems that are vehicle mounted will require very soft vibration isolators and thus a larger shock zone around the disk, both of which lead to higher costs and a physically larger product. Fig. 7 shows a mass storage brick.

The individual mass storage devices are held in place by carriers that were leveraged from the high-volume HP 9000 Model 425e workstation. These carriers, which are shown in Fig. 7, can be oriented towards either the cable end or the noncable end of the tray by means of interlock details located in different places on the tray.

The SCSI interface to the mass storage devices is provided by an external shielded cable which comes from a filtered connector on the CPU brick. This approach was leveraged from the design used in the HP 9000 Models 720 and 730. Besides providing excellent EMI and ESD performance, this design allows the user to connect to an external mass storage device rather than the devices on the mass storage tray. This capability is useful for diagnostics.

VMEbus Brick. The VMEbus brick, shown in Fig. 10, provides six VMEbus slots. The entire brick, which includes the VMEbus cardcage, backplane, and translation circuit, is removable as one piece. Customers are delighted to have the ability to remove the VMEbus brick and take it to a lab bench to work on. With the brick removed, access to the P2 connector† is convenient. A cable passage slot allows easy passage of ribbon cable from the rear of the backplane to inside the cardcage.

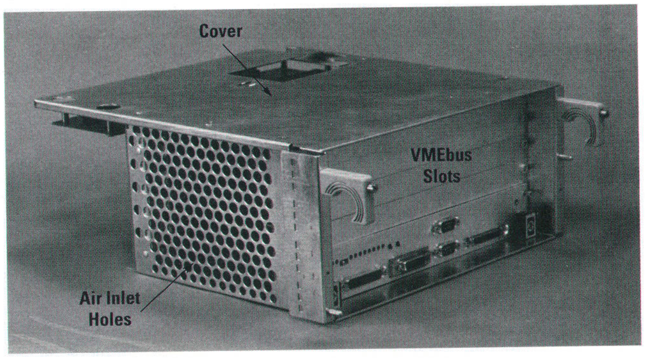


Fig. 10. VMEbus brick with six VMEbus slots. The first two slots are occupied by a two-slot VMEbus module.

The cover shown in Fig. 10 is required to provide RFI regulatory compliance. The customer can modify the cover to add the desired bulkhead-style connector hole patterns and provide cables with service loops as required for each different configuration. Most customers elect to eliminate this part when it is not required.

SGC Brick. Standard graphic connect, or SGC, allows access to HP graphics and is a standard feature of HP 9000 Series 700 workstations.

Power Supply

The power supply delivers up to 300 watts. Once the power supply is removed, the 120-mm fan housed inside the power supply is accessible by removing only two screws. A floating connector system prevents damage from mechanical shock. The power supply is wrapped in metal. Besides protecting the user from electrical shock, this reduces EMI between the power supply and the CPU or other EMI-sensitive bricks.

Acknowledgments

I would like to thank the following individuals: Paul Febvre who as project manager led the team to complete the project on time within cost goals, Ron Dean who contributed to the design of the CPU brick, EISA bricks, SGC brick, and backplane, Dave Merrill who contributed to the mass storage brick and VMEbus brick, plastic parts development designs, and plastic mold vendor interface, and Mo Khovaylo who contributed to the industrial design of the product.

References

1. *Hewlett-Packard Journal*, Vol. 43, no. 4, August 1992, pp. 6-63.
2. *Hewlett-Packard Journal*, Vol. 43, no. 6, December 1992, pp. 78-108.
3. L. A. DesJardin, "VXIbus: A Standard for Test and Measurement System Architecture," *Hewlett-Packard Journal*, Vol. 43, no. 2, April 1992, pp. 6-14.

HP-UX is based on and is compatible with UNIX System Laboratories' UNIX* operating system. It also complies with X/Open's* XPG3, POSIX 1003.1 and SVID2 interface specifications.

† The P2 connector is one of three VMEbus backplane connectors.

Online CO_2 Laser Beam Real-Time Control Algorithm for Orthopedic Surgical Applications

New data obtained from treating polymethylmethacrylate (PMMA) with a nonmoving, CW, 10-watt, CO_2 laser beam is presented. Guidelines based on this data can be used during precision laser surgery in orthopedics to avoid unnecessary mechanical and thermal trauma to healthy bone tissue. A computerized algorithm incorporating these guidelines can be implemented on an HP 9000 workstation connected to a central database for multiple-operating-room data collection, online consultation, and analysis.

by Franco A. Canestri

The work described in this article was done to confirm in greater detail the conclusions published in 1983[1] on treating polymethylmethacrylate (PMMA) with a nonmoving, CW, 10-watt CO_2 laser beam and to investigate any possible additional relationship among the ablated methacrylate volume, the surface crater radius $R(t_e)$, and its depth $Z(t_e)$, where t_e is the beam exposure time in seconds. Because of the very close thermodynamic similarity between PMMA and bone tissue (see Table I), these results may be valuable in orthopedic surgery, where the procedures of cutting bone and removal of bone cement (a methacrylate polymer) are well-known sources of complications. Carbon dioxide lasers have been used in continuous and pulsed modes in both cases, but bone carbonization, thermal injury, and debris result very frequently in inflammatory response with a retarded rate of bone healing. Therefore, a method for clean removal of bone cement and precise osteotomy without mechanical and thermal trauma would have distinct advantages over existing techniques.

In this article, equations for $R(t_e)$ and $Z(t_e)$ for each focal length are presented. A very interesting relation was identified between the ablated volume for a given focal length and the values of R and Z integrated between $t_e = 0$ and $t_e = 2$ seconds. The most important result is confirmation of the very close relationship between the areas under the R and Z curves and the volume. With a simple equation (equation 3, discussed later), it is possible to compare the characteristics of craters obtained with moving and nonmoving laser beams at different operative conditions between 0 and 2 seconds, a time interval that covers the majority of combinations of output powers, scanning speeds, and focal lengths reported in the literature.[3-11] The close thermodynamic similarities between PMMA and compact bone tissue have been demonstrated, except for the water content (Table I, bottom), which strongly influences CO_2 laser beam absorption. Therefore, a correction factor must be applied to the main equation to calculate the ablated volume in bone tissue.

Table I
PMMA versus Bone Thermodynamic Parameters in the Near to Mid-Infrared Wavelength Laser Beam Region (800 nm to 10.6 μm)

	PMMA		Bone Tissue	
Density $\left(\frac{g}{cm^3}\right)$	1.1	[19]	0.8 to 1.3	[5]
Specific Heat $\left(\frac{J}{g\ °C}\right)$	1.38	[7]	1.3 to 23.1	[5]
Thermal Conductivity $\left(\frac{J}{s \cdot cm \cdot °C} \times 10^{-2}\right)$	0.17	[7]	0.16 to 0.34	[5]
Thermal Diffusivity $\left(\frac{cm^2}{s} \times 10^{-3}\right)$	1.06	[1]	1.0 to 2.2	[5]
Fluence Ablation Threshold $\left(\frac{J}{cm^2}\right)$	9.6	[7]	2.1 to 3.4 8.0 to 18.0	[14] [10]
Latent Heat of Ablation $\left(\frac{J}{cm^3} \times 10^3\right)$	3.85	[7]	3.7 to 13.0	[11]
Ablation Energy $\left(\frac{J}{g} \times 10^3\right)$	3.5	[7]	3.0 to 14.0	[3]
Water Content (%)	0.3 immersed 24h @ 23 °C	[19]	10.0	[14]

The numbers in square brackets indicate references listed on page 72.

Portions of this article were originally published in the *International Journal of Clinical Monitoring and Computing.*[2] © Copyright 1992 Kluwer Academic Publishers. Reprinted with permission.

$$\Psi = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ & \cdot & & & & & & \\ & (FS_m, \lambda_n) & & & & & & \\ & & \cdot & & & & & \\ & & & (FS_p, \lambda_q) & & & & \\ & & & & \cdot & & & \\ & & & & & & \vdots & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \vdots & \cdot \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & . \\ 1.5 & 3 & 4.5 & 6 & 7.5 & 9 & 10.5 & . \\ 2.5 & 5 & 7.5 & 10 & 12.5 & 15 & 17.5 & . \\ 3.5 & 7 & 10.5 & 14 & 17.5 & 21 & 24.5 & . \\ 4.5 & 9 & 13.5 & 18 & 22.5 & 27 & 31.5 & . \\ . & . & . & . & . & . & . & . \end{bmatrix}$$

Fig. 1. Focal sequence matrix.

Equipment and Symbols

As described in reference 1, our group at the National Cancer Institute of Milan obtained results for laser wavelengths of $\lambda_1 = 2.5$ in, $\lambda_2 = 5$ in, $\lambda_3 = 7.5$ in, and $\lambda_{6+} = 400$ mm = 15.75 in, using a commercial Valfivre CO_2 laser with a nominal output of 10 watts on the beam spot. The transverse beam mode was TEM_{11*} and the focusing head was kept steady over well-polished cubes of ester methacrylate (Vedril C from Montedison) measuring 3 by 3 by 2 cm. The exposure intervals of the nonmoving CW CO_2 laser beam were set to 0.4, 0.7, 1, 1.3, 1.6, and 1.9 seconds.[12,13] A nitrogen flow helped remove powder and steam during irradiation. Knowing that $\lambda_2 = 2\,\lambda_1$, $\lambda_3 = 3\,\lambda_1$, and $\lambda_{6+} = 6.3\,\lambda_1$, a working matrix Ψ (Fig. 1) was defined in which the elements of each row represent focal lengths $n\lambda_b$, where n = 1, 2, 3, 4,… and λ_b is the basic focal length, varying between 1 inch and an arbitrary maximum in steps of 0.5 inch (first column). The matrix Ψ represents a comprehensive set of commonly used focal lengths[3-11,14] structured to allow quick access to the operational data on a given focal length. Each row of Ψ defines the concept of a focal sequence FS_b of a given basic focal length λ_b.

Results

All of the existing experimental trials performed using a CW CO_2 laser beam with exposure times ranging between 0.4 and 2 seconds on PMMA samples show clearly the strong focal-length-related ablative beam effects.[1,15-18] The data points $R(t_e)$ and $Z(t_e)$ measured in this study can be expressed for t_e between 0 and 2 seconds by the equations shown in Fig. 2. The following empirical equation can forecast the ablated volumes in PMMA for focal lengths of 2.5 in, 5 in, 7.5 in, and 15.75 in (400 mm):

$$V\left(t_e, \lambda_k, FS_b\right) = L\left(\lambda_b, \lambda_k\right) \cdot C\left(t_e, \lambda_k\right) \cdot V_b\left(\lambda_b\right)$$

$$L\left(\lambda_b, \lambda_k\right) \equiv \sum_{n=1}^{k} \frac{\lambda_n}{\lambda_b} \tag{1}$$

$$C\left(t_e, \lambda_k\right) \equiv \left. \frac{\displaystyle\int_o^{t_e} Z(t)dt}{\displaystyle\int_o^{t_e} R(t)dt} \right|_{\lambda = \lambda_k}$$

In this equation, $V(t_e, \lambda_k, FS_b)$ is the ablated PMMA volume after t_e seconds of λ_k-focused laser beam irradiation. $V_b(\lambda_b)$ is constant for each FS_b. n is an integer multiple of $\lambda_b = j + 0.5$ in, where j = 0.5, 1, 2, 3,…

Recent investigations have shown that equation 1 can be written in a more analytical form for exposure times t_e of 0.4 and 2 seconds as follows:

For $t_e = 0.4$ s,

$$V\left(\lambda_k\right) = \left(0.1068 + 0.5581\,\lambda_k - 0.0296\,\lambda_k^2\right)^2$$

For $t_e = 2$ s, (2)

$$V\left(\lambda_k\right) = \exp\left\{\frac{-57.564 + 49.307\,\lambda_k}{1 + 14.741\,\lambda_k + 0.251\,\lambda_k^2}\right\}$$

Fig. 3 compares experimental data for these two exposure times with plots of equations 2. For $t_e = 0.4$ s, $r^2 = 0.996$ and for $t_e = 2$ s, $r^2 = 0.987$, where r^2 is a measure of how well a given analytical curve fits the experimental data ($r^2 = 1$ for a perfect match). These equations can also be used to study

$\lambda = 2.5$ in = 63.5 mm

$$Z = 35 - \frac{21.2}{t_e^{0.2767}} \qquad 0.55 \le t_e \le 2$$

$$Z = 12 - t_e^{0.4306} \qquad 0 < t_e \le 0.55$$

$$\Delta Z = \pm\ 0.34$$

$$2R = 0.258 + 0.642\ t_e^{0.0886} \qquad 0 < t_e \le 1$$

$$2R = 0.9 \qquad t_e \ge 1$$

$$\Delta 2R = \pm\ 0.2$$

$\lambda = 5$ in = 127 mm

$$Z = 31.5 - \frac{22.5}{t_e^{0.4475}} \qquad t_e \ge 1$$

$$Z = 11\ t_e^{0.9777} \qquad 0 < t_e \le 1$$

$$\Delta Z = \pm\ 0.27$$

$$2R = 0.516 + 0.784\ t_e^{0.1519} \qquad 0.2 \le t_e \le 1$$

$$2R = 1.3 \qquad t_e \ge 1$$

$$\Delta 2R = \pm\ 0.42$$

$\lambda = 7.5$ in = 190.5 mm

$$Z = 14.13\ \ln(1 + t_e) \qquad 0 < t_e \le 1.5$$

$$Z = 18 - \frac{8.0172}{t_e^{1.1551}} \qquad t_e \ge 1.5$$

$$\Delta Z = \pm\ 0.3$$

$$2R = 0.774 + 0.706\ t_e^{0.1484} \qquad 0 < t_e \le 1$$

$$2R = 1.48 \qquad t_e \ge 1$$

$$\Delta 2R = \pm\ 0.38$$

$\lambda = 15.75$ in = 400 mm

$$Z = 2.5\ t_e \qquad 0 < t_e \le 2$$

$$\Delta Z = \pm\ 0.11$$

$$2R = 1.625 + 0.575\ t_e^{0.7391} \qquad 0 < t_e \le 1$$

$$2R = 2.2 \qquad t_e \ge 1$$

$$\Delta 2R = \pm\ 0.4$$

Fig. 2. Experimental best-fit equations for $R(t_e)$ and $Z(t_e)$ for nonmoving, 10-watt, CW laser beams at focal lengths of 2.5, 5, 7.5, and 15.75 inches. R and Z are in mm. t_e is in seconds. The transverse beam mode was TEM_{11*}.

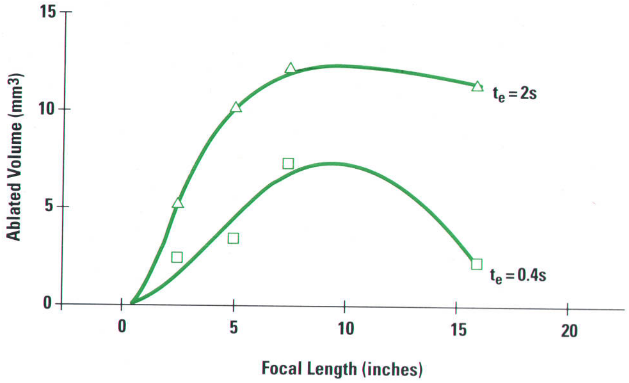


Fig. 3. Best-fit curves of average ablated volumes in PMMA for a 10-watt, nonmoving, CW, TEM_{11*} laser beam.

the effects of changing focal lengths, exposure times, and ablated volumes. It is important to notice that there is a maximum ablated volume for each exposure time t_e, and that increasing the focal length does not correspond to a linear increase of the ablated volume.

LCA Algorithm: Preliminary Investigation and Proposal

Since PMMA[7,19] and compact bone tissue have similar thermodynamic characteristics except for their water content, the proposed equations can be used in a closed-loop computer-assisted algorithm for orthopedic surgical applications. The algorithm is named LCA after the two parameters L and C in equation 1.

The implementation of this algorithm on a HP 9000 HP-UX* workstation would provide the surgeon with an additional safety tool to reduce the risks of bone injury during laser irradiation, which often results in inflammatory response with a retarded rate of bone healing.[4,15,13] This happens quite frequently during general orthopedic surgery, especially because of incorrect settings of laser beam focal lengths and/or exposure times. Removal of bone cement (a PMMA-based polymer) that is in close contact with healthy native bone is the most critical operation in terms of potential bone damage.[7]

Operation of the LCA algorithm is as follows (see Fig. 4). The surgeon specifies the required crater diameter 2R and depth Z and the maximum tolerances Δ2R and ΔZ, and chooses parameters λ_k, t_e, W, v that are likely to produce the desired ablation. (W is the output power of the laser and v is the scanning speed of the laser beam.) The computer program checks whether λ_k in FS_b is also included in $FS_{2.5}$. The focal sequence $FS_{2.5}$ is known experimentally and is therefore always used as the primary reference.

In parallel, the maximum ablation volume V_{max} is calculated and stored as described in reference 15, using the specified values of R and Z. The values of V_b, L, and C are calculated using equation 1. If λ_k is not an element of $FS_{2.5}$, the algorithm interpolates between the two closest focal lengths belonging to $FS_{2.5}$.

Equation 1 has to be corrected for a laser beam that is moving with respect to the operating table and to take into consideration the different CO_2 laser beam absorption modalities of PMMA and bone tissue because of their different water content. A nonmoving laser beam has the same cutting capabilities of a moving beam if the former has an equivalent exposure time (t_{eqv}) given by the equation:[2]

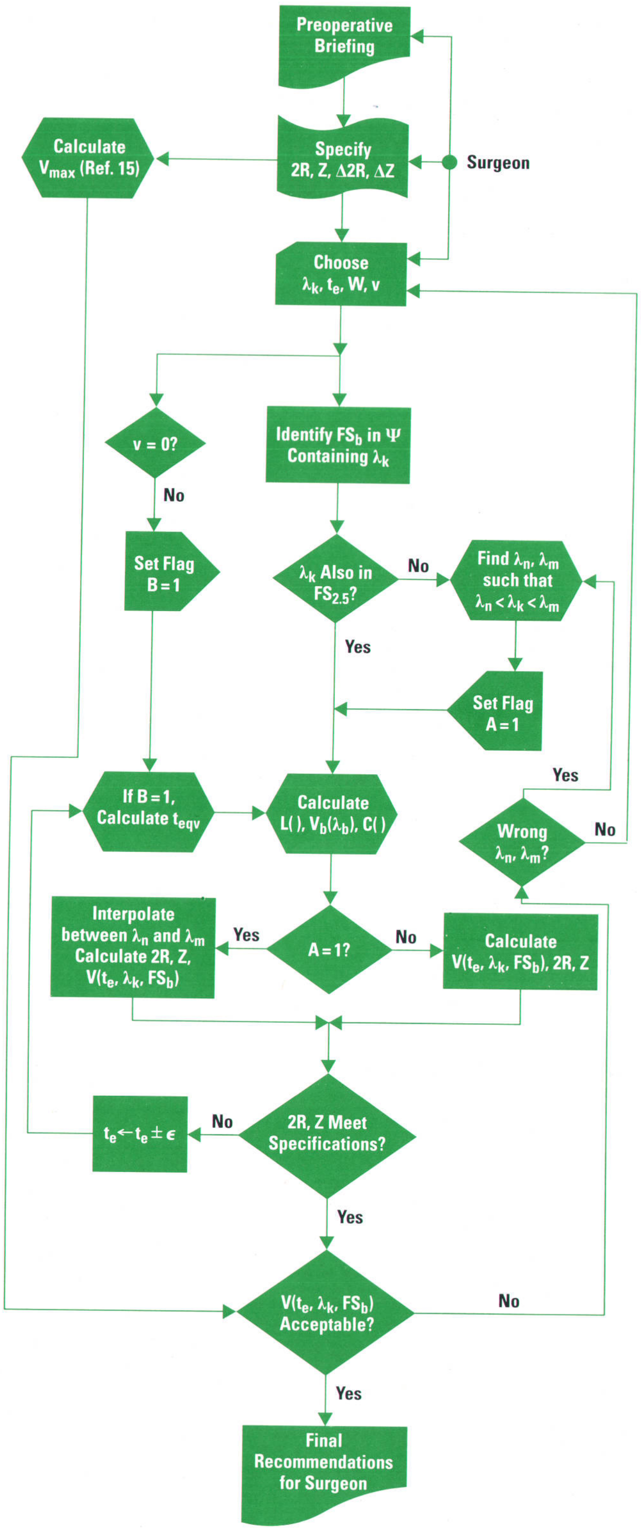

Preoperative Briefing
Calculate V_{max} (Ref. 15)
Specify 2R, Z, Δ2R, ΔZ
Surgeon
Choose λ_k, t_e, W, v
v = 0?
No
Identify FS_b in Ψ Containing λ_k
Set Flag B = 1
λ_k Also in $FS_{2.5}$?
No
Find λ_n, λ_m such that $\lambda_n < \lambda_k < \lambda_m$
Yes
Set Flag A = 1
Yes
If B = 1, Calculate t_{eqv}
Calculate L( ), $V_b(\lambda_b)$, C( )
Wrong λ_n, λ_m?
No
Interpolate between λ_n and λ_m Calculate 2R, Z, $V(t_e, \lambda_k, FS_b)$
Yes
A = 1?
No
Calculate $V(t_e, \lambda_k, FS_b)$, 2R, Z
$t_e \leftarrow t_e \pm \epsilon$
No
2R, Z Meet Specifications?
Yes
$V(t_e, \lambda_k, FS_b)$ Acceptable?
No
Yes
Final Recommendations for Surgeon

Fig. 4. Flowchart of the LCA algorithm.

$$t_{eqv} = \frac{W}{W_{eqv}} \frac{\pi R_s}{2v}, \qquad (3)$$

where R_s is the surface radius of the beam spot, v is the scanning speed of the laser beam, W is the output power of the moving laser, and W_{eqv} is the output power of the nonmoving laser.

In the case of a moving laser beam ($v \neq 0$), equation 3 is used to determine t_{eqv}. The crater diameter 2R and depth Z are then calculated using the equations in Fig. 2. Their values are compared with the specified values using the tolerances $\Delta 2R$ and ΔZ supplied by the surgeon as input data. The two calculated values are corrected by adjusting the exposure time t_e until they are within the specified tolerances. Finally, the data 2R, Z, t_e and $V(t_e, \lambda_k, FS_b)$ are proposed for validation after an additional safety check between the volume V_{max} and $V(t_e, \lambda_k, FS_b)$. This last step is necessary to prevent the ablation volume from exceeding the value V_{max} calculated at the beginning, which is a "not-to-exceed" ablation volume. This can happen if the wrong λ_k and t_e are selected at the beginning of the LCA simulation.

In case of a dangerous situation, a warning message appears and a new focal length is suggested even if it belongs to a different FS_b in $\mathbf{\Psi}$. At the end of the simulation, a comprehensive final report is printed out for the surgeon's convenience. In parallel, a central data base is automatically updated for later review. Reports and statistics can be requested either online for direct support in a specific case that needs more attention or later for teaching and research activities. Video images stored during the actual operation can also be recalled, printed, and attached to the report for the complete documentation of each case.

For $t_e = 0.4$ s and $t_e = 2$ s, equation 2 is used instead of equation 1. This allows a faster determination of the final total ablated volume for a given λ_k.

System Design

By implementing a workstation-based design, each operating room can be equipped with a CO_2 laser mainframe which can be interfaced to an HP-UX workstation able to perform several tasks simultaneously in real time (Fig. 5). For example, one task is the general supervision of the laser beam following the guidelines proposed, analyzed, and validated through the LCA algorithm. This can be achieved by using a laser control interface for dynamic adjustment of the laser's output parameters and by a multiplexer which physically checks that the laser performs as requested. This is done by using an optical device connected to the laser output focusing head, which is also responsible for changing the laser's focal length and the beam mode.

A second important task is network communication among several similarly equipped operating rooms. Each independent node can send LCA simulations, intraoperative data, video sequences, and other results directly to a main database over a multiple-user local area network. The network also allows mutual point-to-point communication so that operating room X can exchange data with operating room Y for consultation. The database and the HP-UX operating system are resident on a network server. The LCA application software together with the related check routines is loaded on each operating room's workstation, which is physically installed in a reserved area close to the operating room but not in the patient's vicinity.

This method can increase the productivity of the operating room suite of a hospital. It also offers the possibility of building a reference center for laser applications in surgery, using a network concept that can be extended to other institutions.

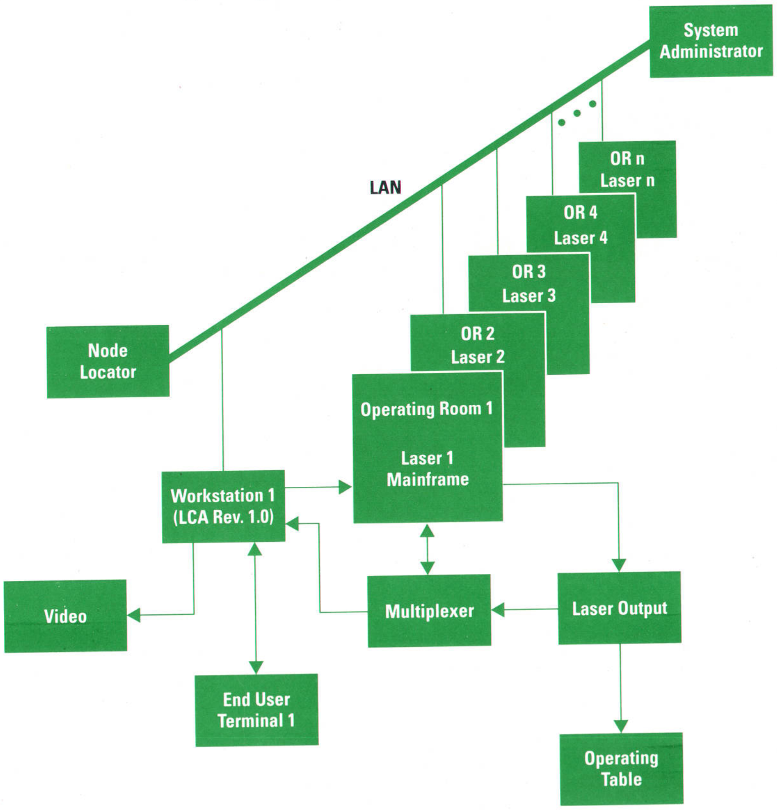


Fig. 5. System design.

Conclusions

The LCA method suggests a global but detailed set of guidelines to be followed during orthopedic surgery using a continuous wave CO_2 laser beam at different operating conditions. Critical cases can be simulated on PMMA samples first and then transferred to bone tissue. It has also been shown how to transfer preliminary test results from PMMA to bone samples for moving or nonmoving CW laser beams. A computerized system can store and control in real time the operative procedures and a convenient database can be built for later consultation. Additional investigation is needed to test the validity of this method over a large variety of hard tissues and during the use of pulsed and superpulsed laser beams.

Acknowledgments

This paper is dedicated to my wife Britta and my son Fabrizio for their continuous support and understanding.

References

1. G. Fava, R. Marchesini, F. Canestri, et al, "CO_2 Lasers: Beam Patterns in Relation to Surgical Use," *Lasers in Surgery and Medicine*, Vol. 2, 1983. pp. 331-341.

2. F. Canestri, "Proposal of a Computerized Algorithm for Continuous Wave CO_2 Laser On-Line Control during Orthopaedic Surgery. Phase I: Theoretical Introduction and First In-Vitro Trials," *International Journal of Clinical Monitoring and Computing*, Vol. 9, 1992, pp. 31-44.

3. S. Biyikli and M.F. Modest, "Energy Requirements for Osteotomy of Femora and Tibiae with a Moving CW CO_2 Laser," *Lasers in Surgery and Medicine*, Vol. 7, 1987, pp. 512-519.

4. L. Clayman, T. Fuller, and H. Beckman, "Healing of Continuous Wave and Rapid Superpulsed Carbon Dioxide Laser-Induced Bone Defects," *Journal of Oral Surgery*, Vol. 36, 1978, pp. 932-937.

5. S. Biyikli, M.F. Modest, and R. Tarr, "Measurements of Thermal Properties for Human Femora," *Journal of Biomedical Materials Research*, Vol. 20, 1986, pp. 1335-1345.

6. S.J. Nelson, et al, "Ablation of Bone and Methacrylate by a Prototype Mid-Infrared Erbium:YAG Laser," *Lasers in Surgery and Medicine*, Vol. 8, 1988, pp. 494-500.

7. C. Scholz, et al, "Die Knochenzemententfernung mit dem Laser," *Biomedizinische Technik*, Vol. 36, no. 5, 1991, pp. 120-128.

8. C. Clauser, "Comparison of Depth and Profile of Osteotomies Performed by Rapid Superpulsed and Continuous Wave CO_2 Laser Beams at High Power Output," *Journal of Oral Surgery*, Vol. 44, 1986, pp. 425-430.

9. R.C. McCord, et al, "CO_2 Laser Osteotomy, Technical Aspects" in I. Kaplan, editor, *Laser Surgery: Proceedings of the Second International Symposium on Laser Surgery*, Dallas, 1977, Jerusalem Academic Press, 1978.

10. R.C. Nuss, C.A. Puliafito, et al, "Infrared Laser Bone Ablation," *Lasers in Surgery and Medicine*, Vol. 8, 1988, pp. 381-391.

11. A. Charlton, A.J. Freemont, et al, "Erb:YAG and Hol:YAG Laser Ablation of Bone," *Lasers in Medical Science*, Vol. 5, 1990, pp. 365-373.

12. K. Hishimoto and J. Rockwell, "Carbon Dioxide Laser Surgery—Biophysical Studies," *Proceedings of the Fourth Congress of the International Society for Laser Surgery*, Tokyo, November 3-8, 1981.

13. J.T Walsh, T.J. Flotte, and T.F. Deutsch, "Erb:YAG Laser Ablation of Tissue: Effect of Pulse Duration and Tissue Type on Thermal Damage," *Lasers in Surgery and Medicine*, Vol. 9, 1989, pp. 314-326.

14. J.T Walsh and T.F. Deutsch, "Erb:YAG Laser Ablation of Tissue: Measurement of Ablation Rates," *Lasers in Surgery and Medicine*, Vol. 9, 1989, pp. 327-337.

15. F. Canestri, "A Proposed Clinical Application of a Model of CO_2 Laser Radiation-Induced Damage Craters," *Journal of Medical Engineering and Technology*, Vol. 12, no. 3, 1988, pp. 112-117.

16. F. Canestri, "Control of CO_2 Lasers during Surgery," *Journal of Biomedical Engineering*, Vol. 9, 1987, p. 185.

17. R.J. Freiberg and A.S. Holsted, "Properties of Low-Order Transverse Modes in in Argon-Ion Lasers," *Applied Optics*, Vol. 8, no. 2, 1969, pp. 355-362.

18. E. Armon and G. Laufer, "New Techniques for Reducing Thermochemical Damage in the Course of Laser Surgery," *Lasers in Surgery and Medicine*, Vol. 7, 1987, pp. 162-168.

19. C. Tribastone and C. Teyssier, "Designing Plastic Optics for Manufacturing," *Photonics Spectra*, Vol. 25, no. 5, 1991, pp. 120-128.

Online Defect Management via a Client/Server Relational Database Management System

The ability to provide timely access to large volumes of data, ensure data and process integrity, and share defect data among related projects are the main features provided in this new defect management system.

by Brian E. Hoffmann, David A. Keefer, and Douglas K. Howell

The defect management system, or DMS, described in this article is an online transaction processing system for managing defects found during software and firmware development and test. It was developed to enable HP's Boise Printer and Network Printer Divisions to manage shared defects in leveraged and concurrent products and to increase data integrity and reduce overall defect processing time. The DMS application is based on an off-the-shelf relational database management system, which employs a client-server architecture running on an HP 9000 workstation. The development team employed an evolutionary delivery process to ensure that the system met user needs and used proprietary 4GL (fourth-generation language) programming tools to maximize productivity. This paper summarizes the rationale for building DMS, details its implementation and design, and evaluates the system and its development process.

Background

Since the introduction of the first HP LaserJet printer in 1984, increasing customer demand for LaserJet products has kept HP's printer divisions on a steady growth curve for years. Market demand for new products with increased capability has continually challenged the R&D and quality assurance organizations to scale up their activities, while improving overall product quality and reliability. Furthermore, competitive pressures for increased frequency of product introductions with shorter development times have challenged development teams to drive out process inefficiencies so they can develop more complex products in less time.

One of the responsibilities of the software quality organization is to provide extensive defect tracking and software process measurement services which enable R&D management to gauge software quality and product schedule accuracy. This also entails maintaining all historical defect data on LaserJet and related products, which provides management information about historical product quality and past and present project schedule trends.

As R&D activities continued to expand, we found that our ability to support the existing defect tracking system became limited. Foreseeing an inability to manage defect information and software metrics at this scale with existing tools, we set out to develop a defect tracking system that could operate under these demands as well as tackle some of the more difficult defect tracking challenges.

Existing Problems

Many features required for the divisions' defect tracking process were not supported by the old defect tracking software. Over time our process had evolved into a largely manual system with limited electronic assistance. Fig. 1 gives an overview of the key elements of our old defect tracking system.

Three physical elements were required for defect submittal: a paper submit form, defective hardcopy (if applicable), and source files (if applicable). Since the existing (pre-DMS) defect tracking software was unable to translate some of these elements into an acceptable electronic form, manual translation and filing processes became necessary. This mixture of human and electronic processes created problems in the following areas:

- Volume sensitivity
- Tracking defects through concurrent projects and code leverages
- Data and process integrity
- Timeliness.

Volume Sensitivity. Among all the problems with the previous defect tracking system, volume sensitivity was the most notable. Because of the serial nature of the old process and its requirement for extensive human assistance to move defects through the system, bottlenecks would occur under any serious load. Many steps required manual intervention by engineers and administrative assistants to drive a defect through its complete cycle. As a result, the labor demands imposed by the defect tracking system became a tremendous burden as the number of defects submitted by projects increased.

Concurrent Projects and Code Leverages. Nearly all the R&D projects that tracked defects were code leverage efforts rather than new code development efforts. In addition, many leverages of similar code were occurring simultaneously among projects at multiple sites. However, no utility or process in the defect tracking system dealt directly with the problem of tracking defects in leveraged code. The problem

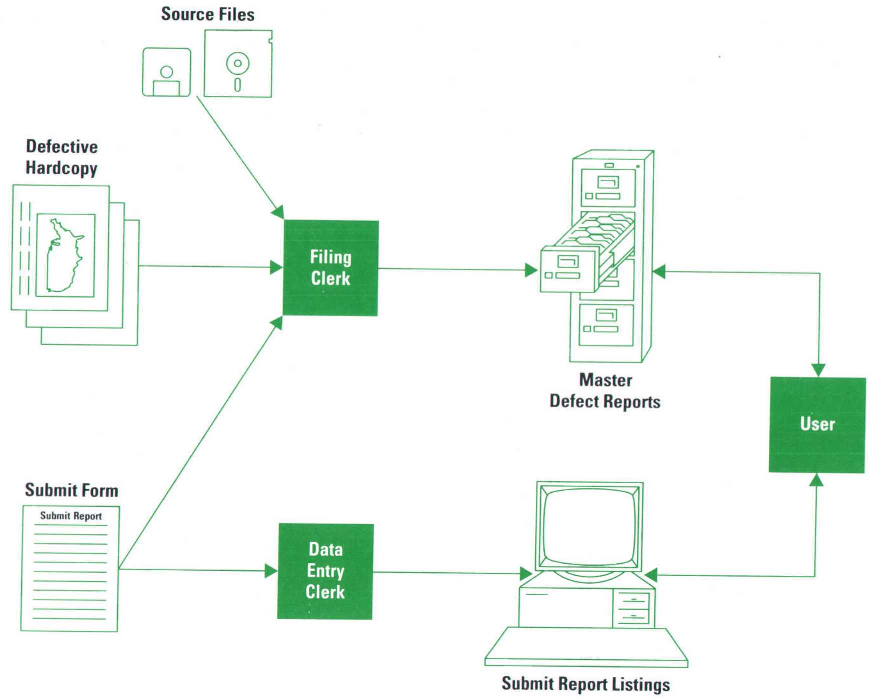


Fig. 1. The original (pre-DMS) defect tracking system.

was particularly noticeable at the beginning of a leveraged project when R&D engineers were required to read entire databases of defects to identify unresolved problems in inherited code. In addition, no formal notification mechanism existed that would notify R&D engineers, for example, that the code they inherited last week received a new defect today.

Data and Process Integrity. An unfortunate and costly byproduct of a system without data and process integrity checks is data corruption and unknown data states. Our old defect tracking system was no exception to this rule. Given the relatively open flat-file data structures and often unreliable e-mail-based transaction schemes of this system, we often scrambled to recover or reconstruct a lost or broken defect record—an activity that often consumed all the time of the defect tracking system administrator.

Timeliness. A final weakness of the old defect tracking system was its inability to provide timely access to accurate defect information and project metrics. Since portions of the process were distributed among various people and tools, instantaneous information was not always available. Even simple requests for defect information might require assistance from the defect tracking administrator or specialized tools. This serial process and its patchwork of components effectively inhibited the free flow of defect information to R&D.

DMS Features

Implementation Guidelines

To maintain focus during the implementation of DMS, the following guidelines were established to assess whether DMS would achieve its design objectives:

- DMS must seamlessly and automatically encapsulate our old defect process model which has proven itself in the past.
- DMS must rely on a client-server architecture to deliver its capabilities via a network to as much of the development community as possible, while centrally maintaining and ensuring 24-hour, seven-day continuous operation.
- Given the rate at which R&D and quality assurance processes are adapting to keep pace with market demands, DMS must be able to adapt and embrace additional process refinements as they evolve.

DMS Process Encapsulation

On the surface, DMS is an online database application that offers engineers and managers full electronic access to defect information. DMS is much more than just a data collection and reporting tool, it is also an electronic mechanism that supports the defect tracking process that we have proven and refined over time.

DMS functionality is divided into six core and six auxiliary functions (see Fig. 2). The core functions were identified as the minimal set required for an operational system. Auxiliary functions were added incrementally in subsequent releases of the tool. The core functions are Submit, Receive, Resolve, Modify/Delete, Update, and Verify. The auxiliary functions are Screen, Screen Resolve, Screen Update, Unreceive, Unresolve, and Unverify. With the exception of Update, each of these functions causes a defect to move from one state to another. The states, which are represented by rectangles in Fig. 2, are Unscreened, Rejected Unscreened, Unreceived, Open, Unscreened Resolve, Unverified Resolve, and Verified Resolve. Each state represents the status of a defect record in the DMS database.

DMS functions are accessed by the user from the menu items presented by the initial DMS screen (see Fig. 3). The Submit, Modify/Delete, and Receive functions are accessed through the Submit menu item, the Resolve and Screen Resolve functions are accessed through the Resolve menu item, and the Verify function is accessed through the Verify menu item. The auxiliary functions are accessed through the Update menu item. Users navigate DMS forms either through the

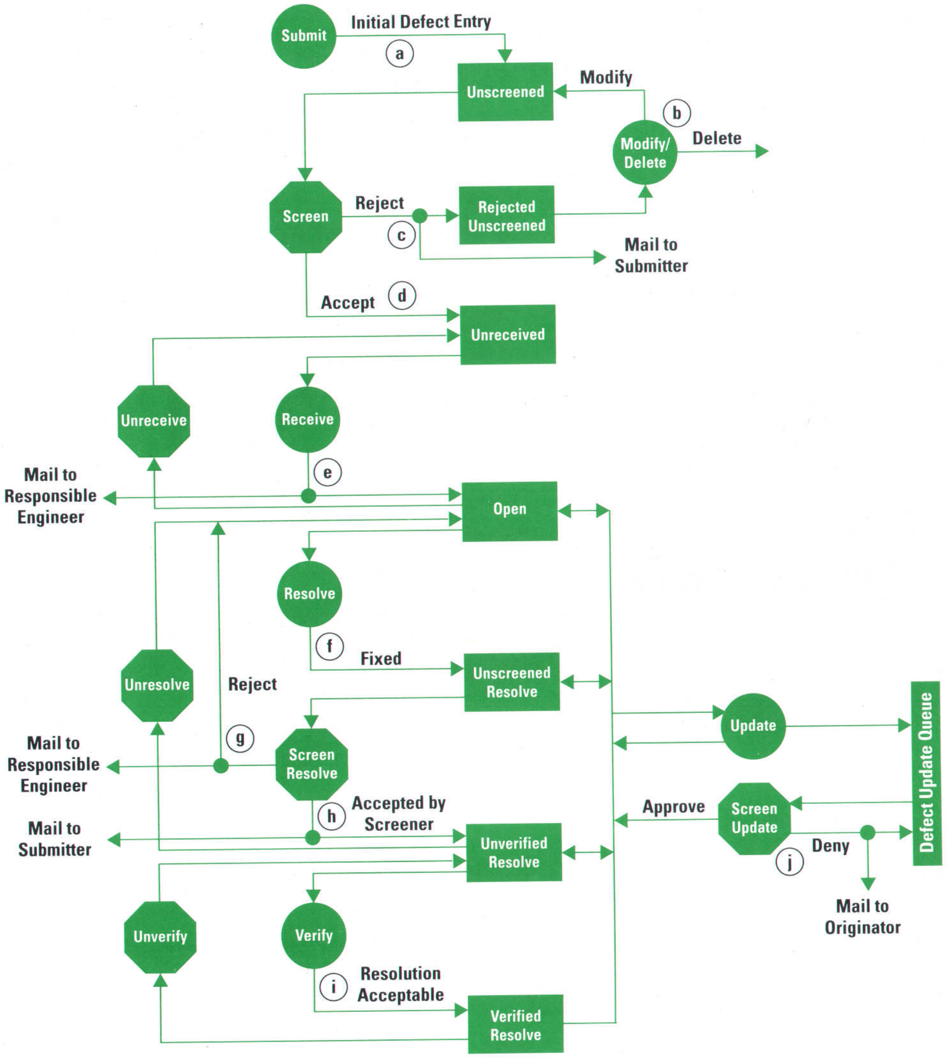


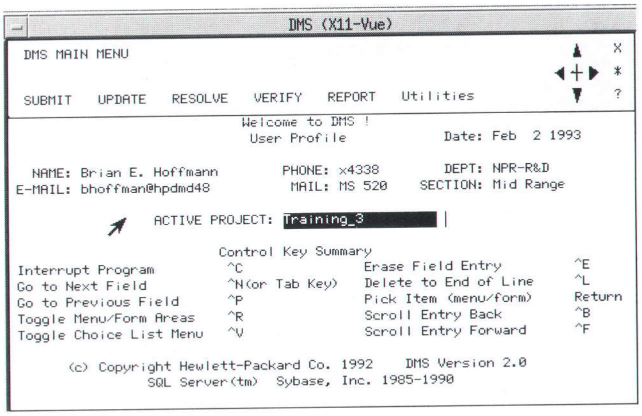
Fig. 2. Functions and states in DMS. The circles represent core functions, the octagons represent auxiliary functions, and the rectangles represent states.

control keys shown in Fig. 3 or by selecting the desired item with a mouse.

Three roles are played by DMS users, and each role has a different permission level. The roles are user (lowest permission level), screener, and manager (highest permission level). Users typically perform the Submit, Receive, Resolve, and Verify functions. Screeners typically perform the Screen and Screen Resolve functions. The manager permission level is reserved for individuals who are responsible for adding and configuring projects in DMS.

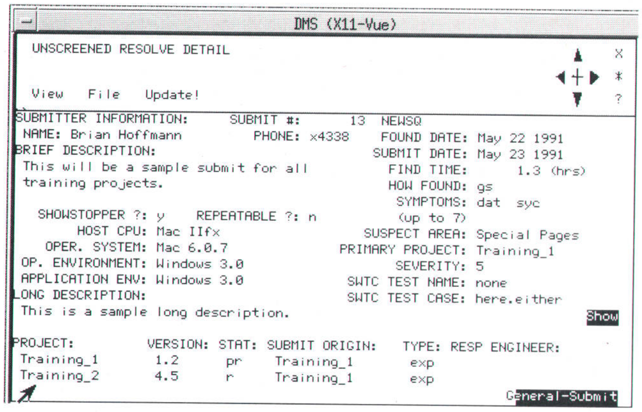
Submit Function. This is where defect information is initially entered into the DMS database, resulting in the defect being placed in the Unscreened state (ⓐ in Fig. 2). The user is required to enter a minimal set of information relating to the defect. The user also has the opportunity to add optional information at this time. Submitters have the ability to attach both text and object files to defects. These files may be of particular use to engineers attempting to reproduce and repair defects. Once a defect is submitted, the user can continue to add additional information via the Modify/Delete function ⓑ until the defect is screened.

Screen Function. This function is typically performed by a person associated with the development team who has intimate knowledge of the product or test process. The Screen function is the point at which the defect information is examined for completeness and correctness, and the defect severity is added to the defect record. If insufficient information is provided by the submitter then the screener may reject the defect, placing it in the Rejected Unscreened state ⓒ. The act of rejecting a defect causes electronic mail to be sent to notify the submitter that the defect was not accepted. The



```
DMS (X11-Vue)

DMS MAIN MENU

SUBMIT   UPDATE   RESOLVE   VERIFY   REPORT   Utilities

                          Welcome to DMS !
                           User Profile            Date: Feb  2 1993

  NAME: Brian E. Hoffmann       PHONE: x4338        DEPT: NPR-R&D
E-MAIL: bhoffman@hpdmd48         MAIL: MS 520    SECTION: Mid Range

           ACTIVE PROJECT: Training_3

                        Control Key Summary
Interrupt Program          ^C               Erase Field Entry        ^E
Go to Next Field           ^N(or Tab Key)   Delete to End of Line    ^L
Go to Previous Field       ^P               Pick Item (menu/form)    Return
Toggle Menu/Form Areas     ^R               Scroll Entry Back        ^B
Toggle Choice List Menu    ^V               Scroll Entry Forward     ^F

      (c) Copyright Hewlett-Packard Co. 1992    DMS Version 2.0
               SQL Server(tm)  Sybase, Inc. 1985-1990
```

Fig. 3. Initial DMS screen. DMS functions are accessed from this screen.



```
UNSCREENED RESOLVE DETAIL

SUBMITTER INFORMATION:      SUBMIT #:       13  NEWSQ
 NAME: Brian Hoffmann          PHONE: x4338    FOUND DATE: May 22 1991
BRIEF DESCRIPTION:                            SUBMIT DATE: May 23 1991
 This will be a sample submit for all           FIND TIME:      1.3 (hrs)
 training projects.                             HOW FOUND: gs
                                                 SYMPTOMS: dat  syc
  SHOWSTOPPER ?: y    REPEATABLE ?: n            (up to 7)
       HOST CPU: Mac IIfx                    SUSPECT AREA: Special Pages
    OPER. SYSTEM: Mac 6.0.7               PRIMARY PROJECT: Training_1
 OP. ENVIRONMENT: Windows 3.0                    SEVERITY: 5
 APPLICATION ENV: Windows 3.0              SWTC TEST NAME: none
LONG DESCRIPTION:                          SWTC TEST CASE: here.either
 This is a sample long description.

PROJECT:         VERSION: STAT: SUBMIT ORIGIN:   TYPE: RESP ENGINEER:
 Training_1        1.2     pr     Training_1      exp
 Training_2        4.5     r      Training_1      exp
```

Fig. 4. One of four pages of information presented to the user performing a Screen Resolve function.

submitter then has the option of deleting the defect or modifying and returning the defect to the Unscreened state via the Modify/Delete function ⓑ. It is important to note that this is the only point in the DMS process where a defect can be removed from the database. Once a defect has been screened it is assigned an identifying number and made part of the permanent project database upon its entry into the Unreceived state ⓓ.

Receive Function. A defect is routed to the person who will most likely be responsible for defect repair via the Receive function. The name of the responsible engineer is obtained from a database list of all possible names for a given project. When a defect is received, the defect is moved into the Open state and an electronic mail notice containing the defect number and brief details about the defect is sent to the person named as the responsible engineer ⓔ.

Resolve Function. When a defect is repaired, the Resolve function is used to add fix information to the defect record and promote the defect to the Unscreened Resolve state ⓕ. This function is usually performed by the engineer who enters the resolution code, the description of the resolution, and any relevant files. Depending on the resolution code, other information may be required as well. For example, if the defect is resolved as a CC (code change) then the user is required to add the name of at least one module that was changed.

Screen Resolve Function. This function allows the project screener to scan the resolved defect to make sure that the resolution information is as complete and correct as possible. No additional information is added to the defect record by this function. Screen Resolve allows each project screener to make sure that the resolution information meets the standards set by each project team. If the resolution is rejected by the screener, the defect is returned to the Open state, and electronic mail is sent to the responsible engineer stating that the resolution has been rejected ⓖ. If the resolution is accepted by the screener, the defect is promoted to the Unverified Resolve state and the submitter is sent electronic mail stating that the defect is ready for verification ⓗ.

Fig. 4 shows one of four pages of information presented to the user performing the Screen Resolve function. The user can switch between pages with the View menu item. The File menu item is used to move files between the HP-UX* file system and the defect record. The Update menu item allows users access to the Update function. The bottom two lines of the form show that the defect is shared between two projects. The defect is in an Unscreened Resolve state for project Training_1 (pr = Unscreened Resolve) and in an Unverified Resolve (r = resolve) state for project Training_2.

Verify Function. The submitter uses the Verify function to determine if the defect is fixed. If the resolution is acceptable to the submitter, a verification code is added to the defect and it is promoted to the Verified Resolve state (ⓘ in Fig. 2). If the submitter decides that the defect is not repaired then the project screener is notified. The screener has the capability to return the defect to the Open state via the Unresolve function.

Undo Functions. Screeners also have the ability to move defects from the Verified Resolve state to the Unverified Resolve state via the Unverify function and from the Open state to the Unreceived state via the Unreceive function. When a defect is moved to a previous state all information that was added by the previous function is lost. For example, when a defect is unresolved the information added by the Resolve function is lost.

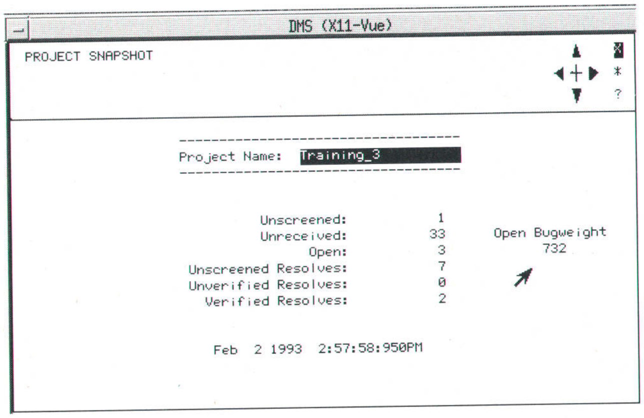
Update Function. When a defect is in the Unreceived state and beyond, changes are made to the defect via the Update function. Changes made to defects by this function are either applied directly to the defect record or placed in an update queue based on the following criteria:

- If the person performing the update is a screener or is listed as a responsible engineer for the project that owns the defect then the update is applied to the defect record.
- If the person performing the update does not meet the above criteria then the modified defect record is placed in the update queue where a screener must approve the modifications before they are applied to the defect database.

There is also a set of configurable rules that may force an update into the update queue. If the screener rejects the update, electronic mail is sent to the originator of the update about the rejection (ⓙ in Fig. 2).

There is an additional process step not shown in Fig. 2. It was pointed out to the DMS developers that in the early stages of a project, engineers frequently find and fix a great many defects in a very short period of time. The engineers found it very time-consuming to submit a defect and wait for another individual (perhaps two) to screen and receive a defect so that it could be resolved. In this case the DMS process was seen as a deterrent to collecting complete defect history. The process model was modified in the third release of DMS to allow the responsible engineer to move a defect from the Unscreened state to the Unscreened Resolve state via the Resolve Unscreened function. This function can be performed only if the submitter and resolver are the same person, and the resolver is the responsible engineer for the project that owns the defect.

Users can readily determine the distribution of defects for a project through the project snapshot screen (see Fig. 5). Accessed from the Report menu item shown in Fig. 3, the snapshot shows the number of defects in each DMS state and the bugweight. The bugweight metric is calculated as the sum of severities squared for all defects in the Open and Unreceived states.



```
DMS (X11-Vue)

PROJECT SNAPSHOT

            ------------------------------------
            Project Name:   Training_3
            ------------------------------------

                  Unscreened:          1
                  Unreceived:         33      Open Bugweight
                        Open:          3           732
        Unscreened Resolves:           7
        Unverified Resolves:           0
          Verified Resolves:           2

            Feb  2 1993  2:57:58:950PM
```

Fig. 5. Project snapshot screen.

DMS fully encapsulates this six-step process and subjects all incoming defect information to rigorous process checks. Users can rely on the knowledge that defects are in predictable states and benefit from the valuable process metrics that are derived by measuring transitions from state to state.

Client/Server Architecture

Another DMS characteristic that provides the foundation for many DMS services is the fact that it is built on a commercially available client/server RDBMS (relational database management system). Among the many benefits of a client/server database architecture are distributed processing, heterogeneous operating environments, and elimination of many configuration management problems. An additional database feature that enables DMS to guarantee data and process integrity is transaction management. Since DMS uses the OLTP (online transaction processing) capabilities of the underlying database, all operations that modify data in DMS are nested in real-time transactions. Modification requests that fail as a result of invalid data or hardware failure, for example, do not corrupt existing data. Bad transactions are automatically rolled back to known previous states. See "Client/Server Database Architecture," on page 78.

Defect Sharing. One of DMS's most important implementation details is the relational structure of a defect record. By making use of traditional relational design guidelines, defect record implementation in DMS enables information for one project to be shared easily with a record from another project. As Fig. 6 indicates, the pre-DMS implementation of a defect record consisted of project-dependent submit and resolve information, which could not be easily shared between projects.

In DMS, relational structuring has been used to separate submit information from project-specific resolve data. One submit record can be shared among many different projects, with each project having a potentially unique resolution record. This model is shown in Fig. 7. The major benefits of this structure are twofold. First, all projects charged with the same defect automatically share project independent submit information. Second, the structure provides an automatic communication path for project dependent resolve information to all the projects that share the defect. Thus, each project can quickly obtain another project's status information for a shared defect.

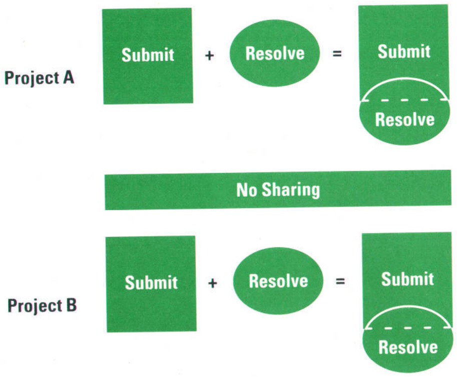


Fig. 6. A pre-DMS defect record consisted of project dependent submit and resolve information tied together in one record structure. Sharing defect information between projects was difficult.

Flexible Architecture. DMS has been able to derive several benefits from its relational implementation. Of these benefits, structural extensibility is perhaps the most important. Given that the system was designed and implemented in an evolutionary delivery model, a relational architecture proved to be an ideal complement. Existing structures can be easily reused and new structures can be added without massive structural rewrites. For example, elaborate user configurability was not supported in early releases. For subsequent releases, however, it was a simple matter to add user configuration tables to the schema without altering existing defect record structures.

Connectivity. DMS's client/server architecture has also enhanced various aspects of connectivity. The physical separation of data manipulation code (back end) from user interface code (front end) maximizes modularity, resulting in more readable, less error-prone code. As a side benefit of the separation, incremental functionality can be added to the front or back end of the code while online without affecting either end. From a maintenance perspective, the client/server architecture eliminates many traditional configuration management headaches. Software distribution problems, for example, are eliminated since client interface code is exported to users on the network from one central location via NFS. From a network perspective, a heterogeneous client environment is fully supportable. Since the server requires no knowledge of client type, multiple client platforms have equal access to DMS.

Robust Operation. Transaction management capabilities round out the list of major DMS features. Given that DMS was a migration to a multiuser online transaction processing system, full transaction arbitration became a must. During

(continued on page 80)

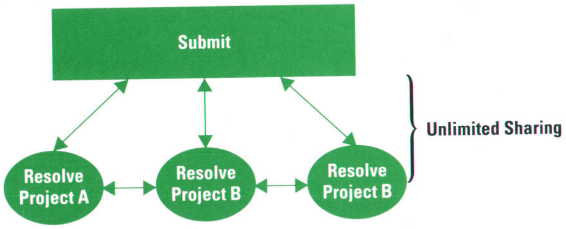


Fig. 7. The relational structuring of DMS records allows submit information to be shared between different projects.

Client/Server Database Architecture

Database management systems implemented under client/server computing models have enjoyed increasing popularity in recent years. Advanced networking capabilities coupled with powerful minicomputer and microcomputer systems connected to networks have favored client/server database architectures over more traditional centralized database management systems. While the specific benefits of implementations vary, client/server databases typically distinguish themselves in four general categories:

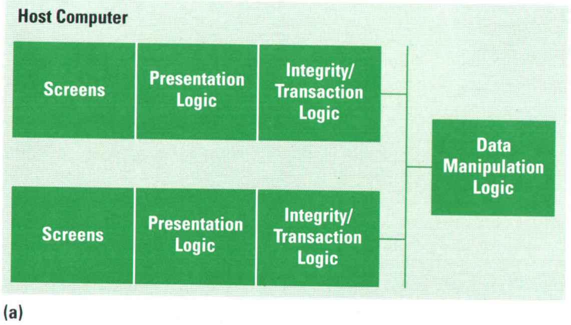
- Separation of presentation services from data manipulation services
- Scalable high performance
- Server-enforced integrity and security
- Heterogeneity and distribution autonomy.

Separation of Presentation and Data Manipulation Services

Unlike traditional centralized databases, client/server database environments cleanly separate presentation (user interface) services from data manipulation services. A database client performs all application or user-specific services necessary to convey information to and from a user. The data server focuses all services on the efficient and secure manipulation of data conveyed to it from the client. Fig. 1 illustrates the labor division in traditional and client/server architectures.

The net result of this separation is an optimum division of labor: data management and transaction functions are managed independently from user interface and presentation functions. The benefits of this approach are twofold. First, distribution of client services to each client CPU enables the server to maintain a respectable response time performance advantage over traditional databases. Fig. 2 shows typical response curves for traditional versus client/server databases. Since client/server databases are less demanding of operating system overhead, they tend to perform better under load.

The second major benefit of the client/server separation is the leverage of existing CPUs on the network. Client/server architectures stretch an organization's overall CPU investment by ensuring that overall processing loads are properly balanced between client and server processing.



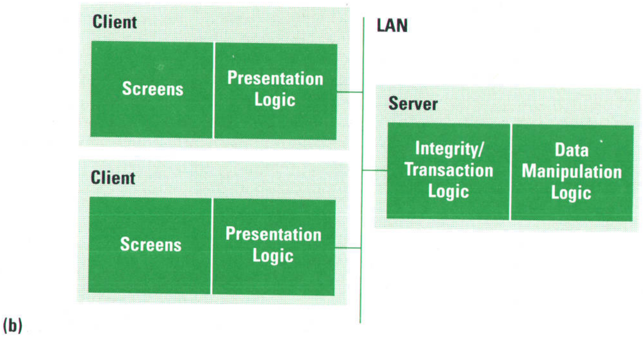


Fig. 1. Division of processing labor in traditional versus client/server architectures. (a) Traditional architecture. (b) Client/server architecture.

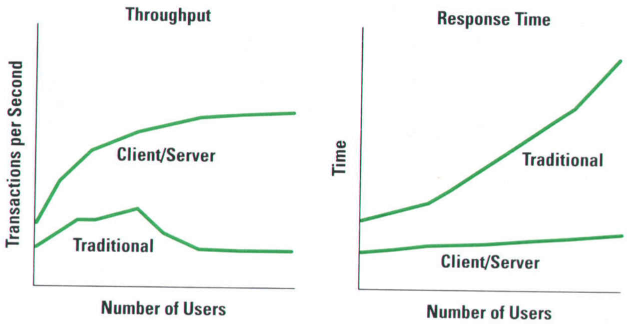


Fig. 2. Throughput and response time performance in traditional and client/server architectures.

Scalable High Performance

A unique performance trait of most client/server database architectures is their ability to enhance server performance much more than traditional database architectures. As seen in Fig. 2, response time as a function of user load tends to scale more linearly under client/server architectures. This performance advantage is often rooted in the following design fundamentals:

- Use of stored database procedures, which often include control flow extensions to the data manipulation language
- Use of remote procedure calls (RPCs) for server-to-server communication
- Implementation of the server as a single multithreaded process in the operating system.

Stored procedures are the hallmark of client/server databases. Typically, they exist as specialized database executables. These executables are constructed by compiling source statements from the same data manipulation statements (e.g., SQL) used with the server in an interactive fashion. Once a procedure is compiled and stored in the database's data dictionary, an application can issue a run-time call to the procedure. The procedure then executes the same data manipulation or query that was defined by the source statements that built the procedure.

Stored procedures offer many major performance benefits. First, network communications are dramatically reduced since one procedure call replaces many individual data manipulation statements. Second, since stored procedures are already compiled at run time, performance measurements indicate that they can process data manipulation statements five to ten times faster than a sequence of single data manipulation statements. Fig. 3 illustrates the execution differences between stored procedures and traditional database servers.

Third, stored procedures often possess the ability to control logic flow in database operations. Control constructs such as branching and looping combined with the ability to declare local variables and create temporary database objects, such as tables, enable stored procedures to perform complex data manipulation sequences. Stored procedures can also be nested to invoke a series of database events with a single function call.

Another feature high-performance servers possess is the implementation of the server as a single operating system process while accommodating multiple simultaneous client processes. Fig. 4 illustrates the architectural differences between single-threaded and multithreaded server implementations.

A multithreaded server design frees the database from nearly all of the operating system overhead that limits traditional database architectures. For example, the amount of memory required for a database user connection in a multithreaded implementation is around 50K bytes. In contrast, traditional single-threaded servers can require up to 2M bytes of memory per user connection, operating system

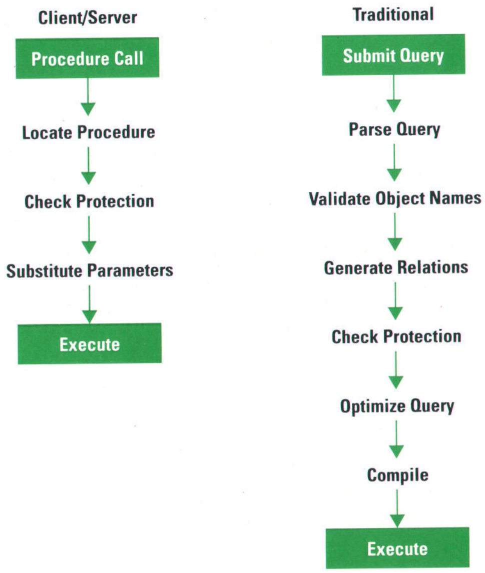


Fig. 3. Execution profiles in traditional and client/server architectures.

overhead included. Hence, multithreaded server implementations make more memory available for disk caching and other applications.

Server-Enforced Integrity and Security

The client/server approach also has many advantages in preserving the integrity of information in a database. Unlike traditional approaches to maintaining data and process integrity, business rules and data transaction checks in a client/server database environment are exclusively enforced by the server. Opportunities for data corruption resulting from maintenance efforts are significantly reduced, since business rules and transactions only have to be modified in the server instead of in every client application using them.

A specific mechanism often employed by a client/server database for enforcing integrity constraints is known as the trigger. A trigger is a special type of stored procedure that is attached to a table and automatically called, or triggered, by an attempt to insert, delete, or update data in a table. Since triggers reside in the server with the database, they are particularly effective as integrity mechanisms since they adopt a data- and business-rule-driven approach to integrity, as opposed to an application-controlled integrity approach. Trigger code is written only once, instead of many times in multiple client applications. An application cannot avoid firing a trigger when it attempts to modify data in a table.

Another common use of triggers is for the maintenance of internal database consistency, or referential integrity. For example, duplicate data rows in related tables can be prevented by a uniqueness constraint defined in an insert trigger of either or both tables to guarantee the one-to-one unique relationship that exists between two tables. Since client applications cannot be relied on to maintain the consistency of a database, triggers prove to be the ideal mechanism for this task.

Some integrity mechanisms seen in client/server database environments impose data constraints on single data fields directly. These mechanisms include rules, defaults, and user-defined data types. A rule is a programmable mechanism for performing conditional data checks such as data range checks or conditional comparisons as well as structural checks on data syntax. Defaults simply provide a user-specified value on inserts in the event that one is not provided with the insert statement. User-defined data types provide integrity on values that are at higher levels of abstraction than numerical types alone provide. Some higher-level user-defined data types might include money, color, or postal code.

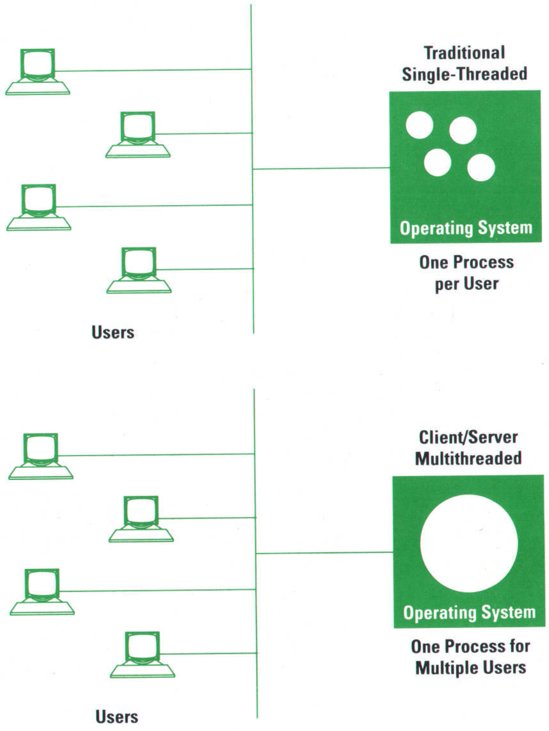

Traditional Single-Threaded
Operating System
One Process per User
Users
Client/Server Multithreaded
Operating System
One Process for Multiple Users
Users

Fig. 4. Server designs. (a) Traditional, single-threaded design with one process per user. (b) Client/server, multithreaded design with one process for multiple users.

Heterogeneity and Distribution Autonomy

Client/server databases deliver an open architecture that facilitates portability to and management of the multivendor components within a networked computing environment. Hardware independence is much more easily achieved in client/server situations since the architecture can cleanly divide client hardware from server hardware. Furthermore, most commercial vendors of client/server database environments offer mechanisms for easily linking different operating platforms together. Thus, existing and new hardware resources have a much higher compatibility likelihood and can therefore be used more efficiently. Additionally, hardware budgets can be spread further and take full advantage of downsizing to workstations and supermicrocomputer systems.

In addition to supporting hardware independence in networked database environments, client/server interfaces also permit open communication in heterogeneous software environments. The same formalized software interfaces that connect a client to a server can be leveraged by other applications or software environments. Foreign data sources and applications can be seamlessly integrated into a client/server database environment.

Although the client/server approach breaks down the traditional barriers that prevent data distribution, it simultaneously creates potential for excessive complexity in distributed database processing. As the risk of data corruption increases with the complexity of distribution schemes, the old "centralize versus decentralize" debate becomes justifiably fueled. Fortunately, implementation under a client/server approach does not require an all or nothing approach to distribution of the database system. Developers are free to choose the level to which the database system is to be distributed, thus retaining a high degree of autonomy on the issue of distribution. Furthermore, the client/server approach allows developers to evolve the system incrementally toward more or less distribution as required by the application. In traditional database architectures, the choice is fixed, with evolution to more distributed and heterogeneous environments made virtually impossible.

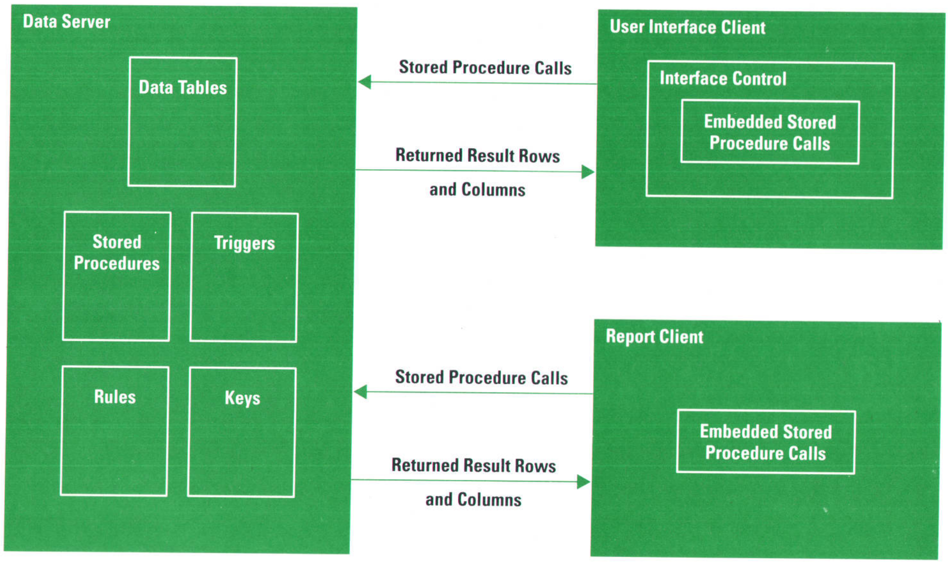


Fig. 8. Software components of the client/server architecture used in DMS.

heavy operation, the server reliably conducts more than 50 simultaneous client sessions. A beneficial side effect of transaction management for DMS is that the integrity of all process steps can be maintained. Any process change that fails will simply be rolled back. Transaction management also provides the capability to assign user permissions to process transactions. With transaction rules and user roles fully defined in DMS, all process rules can be enforced through user permissions.

Physical Overview

DMS is implemented in two distinct components: a data server and a user interface client (see Fig. 8). In addition, mail clients and report clients exist that can also interact with the server. The data server is a commercially available relational database that contains the procedures, triggers, rules, and keys that control data manipulation. These elements are stored in the server as part of the DMS build process.

Stored Procedures. Stored procedures are database routines that are constructed from individual data manipulation statements. They reside in the data server as database executable routines. Similar to function calls in programming languages, stored procedures can accept input arguments and return an exit status. However, unlike traditional functions, stored procedures can return a variable-length stream of data, which is organized into rows and columns. Columns typically represent data fields, which are placeholders for information in a database table. Rows usually represent the data records that are stored in a database table. Results are typically returned in ASCII tab-delimited format for presentation or postprocessing.

A stored procedure may contain one or more batches of SQL (Structured Query Language) statements. The SQL statements make use of input arguments (if any), perform some data manipulation operation, and return a stream of formatted data (if any) to the client application that made the call. In addition, stored procedures can make use of control flow constructs such as if...else, while, and so on to perform complex data manipulation tasks. Stored procedures can also call other stored procedures performing a series of data manipulation operations with a single function call. In DMS stored procedures are used extensively. For example, the entire DMS process shown in Fig. 2 is supported as a series of stored procedures.

DMS clients are able to call stored procedures and receive data returned from them via a bidirectional communications protocol. In DMS, this protocol is fully supported by the database manufacturer in the form of a series of libraries that link the user interface forms code with the database manipulation language (e.g., SQL).

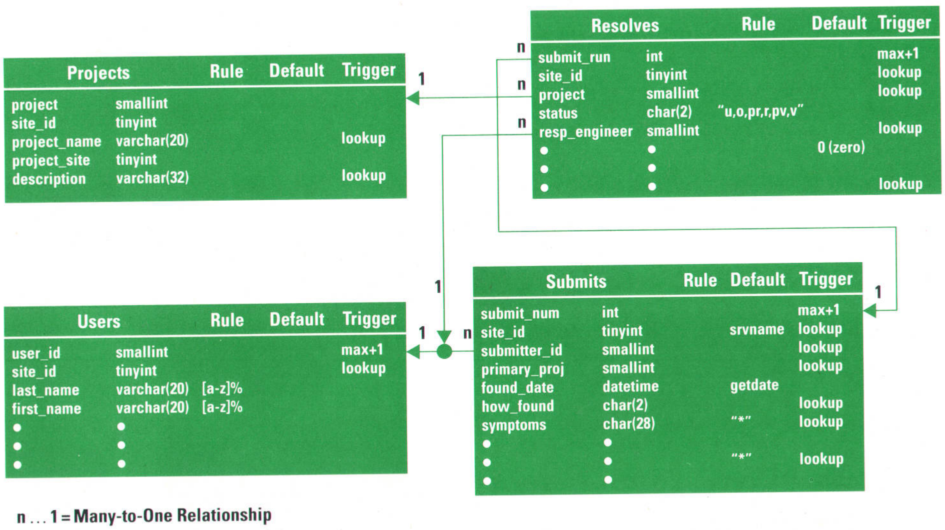
Triggers. Operations performed by stored procedures that modify data may cause the execution of a server trigger. A trigger is a special type of stored procedure installed on the server to ensure the relational integrity of the data in the DMS database. Any operation on a table or column that modifies data will cause a trigger to be executed. Triggers are used in DMS to ensure that all operations that modify data are carried out consistently throughout the database. For example, a trigger will prevent a user from being deleted from the DMS database if that user is referenced in an open defect.

Rules. Rules are another database object used in the DMS process to enforce integrity constraints that go beyond those implied by a particular data type. Rules are applied to individual columns of tables to ensure that the values applied by insert or update operations conform to a particular set or range of possible values. For example, the table column that denotes the state of a defect is a character data type of up to two characters. The rule applied to this column ensures that a prospective insert or update will not succeed unless the new value corresponds to one of the seven states shown in Fig. 2. The following example illustrates this rule:

```
create rule status_rule
as @status in ("n ", "nr", "u ", "o ", "pr ", "r ", "v ")
```

The states "n" and "nr" refer to the Unscreened and Rejected Unscreened states respectively. The remaining states represent (in order) Unreceived, Open, Unscreened Resolve, Unverified Resolve, and Verified Resolve.

Tables. The DMS database is constructed of a number of tables. The tables serve to gather data items into logically related groups. Separate tables exist to contain information relating to the submit and resolve portions of defects. Other



Projects		Rule	Default	Trigger
project	smallint			
site_id	tinyint			
project_name	varchar(20)			lookup
project_site	tinyint			
description	varchar(32)			lookup

Resolves		Rule	Default	Trigger
submit_run	int			max+1
site_id	tinyint			lookup
project	smallint			lookup
status	char(2)	"u,o,pr,r,pv,v"		
resp_engineer	smallint			lookup
●	●		0 (zero)	
●	●			
●	●			lookup

Users		Rule	Default	Trigger
user_id	smallint			max+1
site_id	tinyint			lookup
last_name	varchar(20)	[a-z]%		
first_name	varchar(20)	[a-z]%		
●	●			
●	●			
●	●			

Submits		Rule	Default	Trigger
submit_num	int			max+1
site_id	tinyint		srvname	lookup
submitter_id	smallint			lookup
primary_proj	smallint			lookup
found_date	datetime		getdate	
how_found	char(2)			lookup
symptoms	char(28)		"*"	lookup
●	●			
●	●		"*"	lookup
●	●			

Fig. 9. A portion of the relational table structure hierarchy in DMS.

tables exist to maintain information about fixed modules, attached files, and auxiliary information. There is another group of tables that maintain information about projects, users, and DMS server sites. These tables are related to one another via server keys which serve to map the relationships between tables in the database and help ensure relational integrity (see Fig. 9). When a stored procedure that returns rows from more than one table is executed, keys are used to make sure that the information is joined properly and that the data returned does not contain any unwanted rows.

User Interface Client. The user interface client is constructed from a commercially available 4GL forms language and a custom C language run-time executive. The forms language development environment allows the rapid construction and evaluation of groups of atomic user interface objects like input fields, pull-down menus, and form decoration. Stored procedure calls are triggered by these atomic objects causing the server to generate rows of result data. The forms language is designed to manage returned data rows efficiently with a minimal amount of client coding. The C language run-time executive provides the interface client with access to HP-UX* commands and custom C functions. Mail generated by DMS originates from the client interface through the mailx(1) command. Files that are attached to defects are loaded into the server via a call to a C function by the interface client.

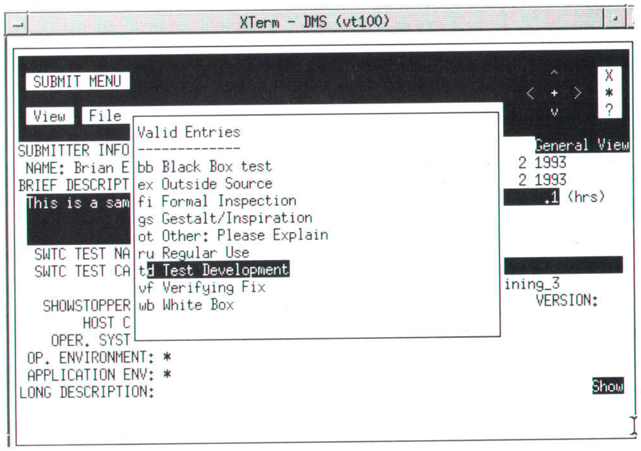
The user interface client can be compiled to execute on a number of hardware platforms. In addition to HP 9000 Series workstations, users routinely execute the tool from networked Macintosh computers running Mac-X and PC-compatible computers running X server software or VT100 terminal emulation. The latter is particularly convenient for users who wish to run the tool from home via modem. Fig. 10 shows the DMS interface as seen from a VT100 terminal emulator. The example in Fig. 10 shows the lookup choices given to the user for the "how found" field of the submit function. The main difference between this and the X11 presentation is that the user must navigate the forms via control keys instead of a mouse.

Utility Clients. Other types of clients, typically report clients, can be created using C language libraries purchased from the database manufacturer. Programs generated using these libraries differ from other types of host language interfaces that rely on embedded SQL. The libraries do not require a host language precompiler to process the source code into some intermediate form. These libraries have been used to generate custom reporting tools which can be executed from any suitably configured workstation as illustrated below:

```
$ subnum –file parms –project SUNFLOWER –sort submitter |
   extract | csv_report > report_file
```

The client subnum generates the key values for every defect belonging to project SUNFLOWER which meets the criteria contained in file parms ordered by the name of the submitter. These key values are piped to the client extract which pulls information about each defect from the database. This information is then piped to a text processing script and deposited in a file. Using this scheme, data can be extracted from the DMS database and readily formatted for spreadsheet applications as well as plain ASCII text reports. Additionally, there are a number of third-party software vendors that provide products that are designed to interact with DMS provided that the PC has network access to the data server.

Execution Environments. DMS data servers are typically set up on dedicated hosts. The server at our division server is currently installed on an HP 9000 Model 710 computer configured to connect with up to 200 simultaneous clients (see



```
XTerm - DMS (vt100)

 SUBMIT MENU                                                 ^    X
                                                           < + >  *
 View  File                                                  v    ?
                  Valid Entries
SUBMITTER INFO    -------------                       General View
 NAME: Brian E    bb Black Box test                    2 1993
BRIEF DESCRIPT    ex Outside Source                    2 1993
 This is a sam    fi Formal Inspection                      .1 (hrs)
                  gs Gestalt/Inspiration
                  ot Other: Please Explain
  SWTC TEST NA    ru Regular Use
  SWTC TEST CA    td Test Development
                  vf Verifying Fix                    ining_3
    SHOWSTOPPER   wb White Box                           VERSION:
         HOST C
     OPER. SYST
 OP. ENVIRONMENT: *
 APPLICATION ENV: *
LONG DESCRIPTION:                                             Show
```

Fig. 10. DMS interface as seen from a VT100 terminal emulator screen.

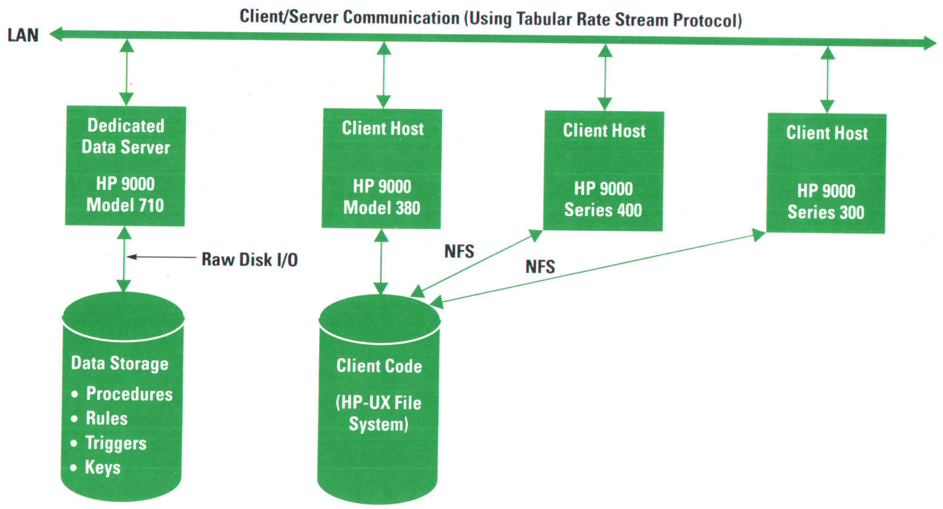
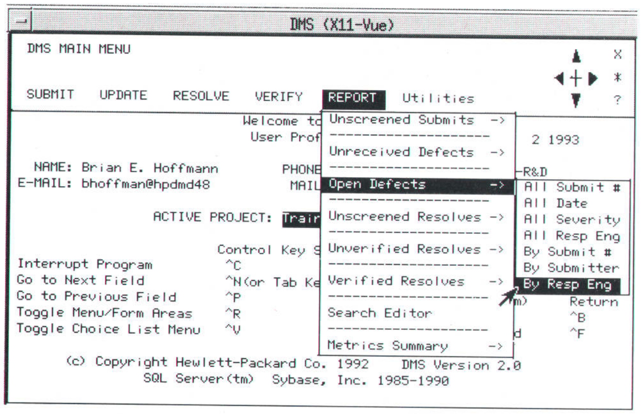


Fig. 11. The hardware components of the DMS client/server architecture.

Fig. 11). Client code may reside on any suitably configured workstation. For convenience and maintainability we have placed the user interface client code on a single HP 9000 Model 380 server which distributes the code to other client hosts via NFS.

Reporting and Searching. Reports can be generated from DMS via the Report menu item from the top-level form shown in Fig. 3. A variety of canned reports are available as an ad-hoc query mechanism. Fig. 12 shows how a user might generate a report of all open defects for a particular engineer. When initiated, the interface prompts the user for the name of a responsible engineer from a list of responsible engineers who have open defects for a given project. Users can readily generate reports of open defects ordered by submit number, submit date, severity, and others. Fig. 13 shows a summary list of open defects ordered by submit number. These reports can be printed or saved to a file in a number of formats.

The ad hoc query mechanism allows customized summary reports to be generated based on any combination of selection criteria. This utility can be used to produce, for example, a list of resolved defects for a given project with a particular submit version, resolution code, and fix time greater than eight hours. These customized reports are generated by the Search Editor selection shown in Fig. 12.



```
                                DMS (X11-Vue)
 DMS MAIN MENU

 SUBMIT    UPDATE    RESOLVE    VERIFY   REPORT    Utilities
                         Welcome to| Unscreened Submits  ->
                          User Prof| -------------------      2 1993
                                   | Unreceived Defects  ->
  NAME: Brian E. Hoffmann    PHONE | -------------------   -R&D
E-MAIL: bhoffman@hpdmd48      MAIL | Open Defects        -> All Submit #
                                   | -------------------    All Date
            ACTIVE PROJECT: Train  | Unscreened Resolves -> All Severity
                                   | -------------------    All Resp Eng
                   Control Key S   | Unverified Resolves -> By Submit #
Interrupt Program         ^C       | -------------------    By Submitter
Go to Next Field          ^N(or Tab Ke Verified Resolves  -> By Resp Eng
Go to Previous Field      ^P       | -------------------  m)   Return
Toggle Menu/Form Areas    ^R       | Search Editor               ^B
Toggle Choice List Menu   ^V       | -------------------  d      ^F
                                   | Metrics Summary     ->
       (c) Copyright Hewlett-Packard Co. 1992    DMS Version 2.0
                 SQL Server(tm)  Sybase, Inc. 1985-1990
```

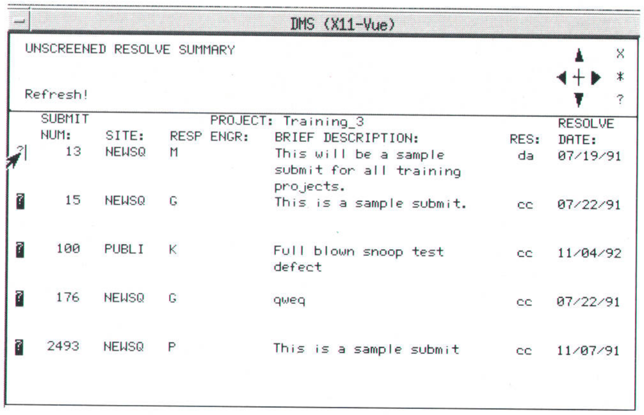
Fig. 12. A screen for generating a list of defects belonging to a particular engineer.

We use PC-based and HP-UX-based spreadsheet packages to produce custom graphic reports for project management. These reports are generated and distributed on a weekly basis. The reports can also be generated on a demand basis by the individual project teams. Figs. 14 to 18 show samples of some of these reports.

Current Status

DMS is currently in its fourth release after two years of development. The latest release, version 2.0, contains all of the original target functionality and a significant number of user-requested enhancements. This latest release contains features that allow online project configuration, user configuration, and defect modification. These features, along with other new features, reduced the amount of system administration time to expected levels. Enhancements in version 2.0 include changes to make similar operations exhibit more consistent behavior throughout the tool and changes to the defect management process to satisfy the demonstrated needs of project teams. Finally, the enhancements provided in version 2.0 of DMS that have evolved over several releases reflect the maturity of the product and the relative stability of the feature set.

Currently DMS is in use at six HP divisions on four sites. More than seven hundred users in R&D, QA, and technical



```
                                DMS (X11-Vue)
 UNSCREENED RESOLVE SUMMARY

 Refresh!
   SUBMIT                  PROJECT: Training_3                       RESOLVE
   NUM:    SITE:   RESP ENGR:  BRIEF DESCRIPTION:              RES:  DATE:
 ?    13   NEWSQ   M           This will be a sample            da   07/19/91
                               submit for all training
                               projects.
 ?    15   NEWSQ   G           This is a sample submit.         cc   07/22/91

 ?   100   PUBLI   K           Full blown snoop test            cc   11/04/92
                               defect

 ?   176   NEWSQ   G           qweq                             cc   07/22/91

 ?  2493   NEWSQ   P           This is a sample submit          cc   11/07/91
```

Fig. 13. Summary list of open defects.

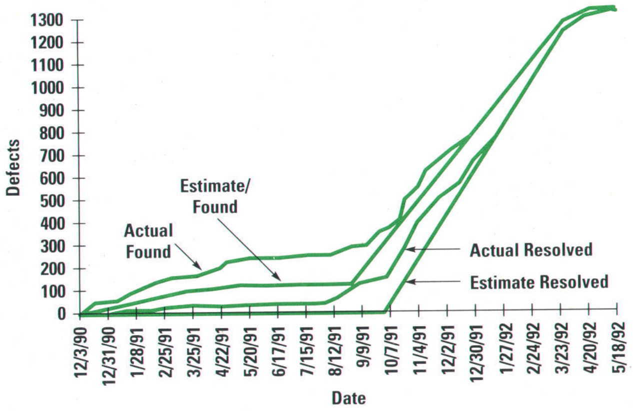


Fig. 14. This report shows how the ratio of defect find rate to defect fix rate can be used to track ongoing project progress.

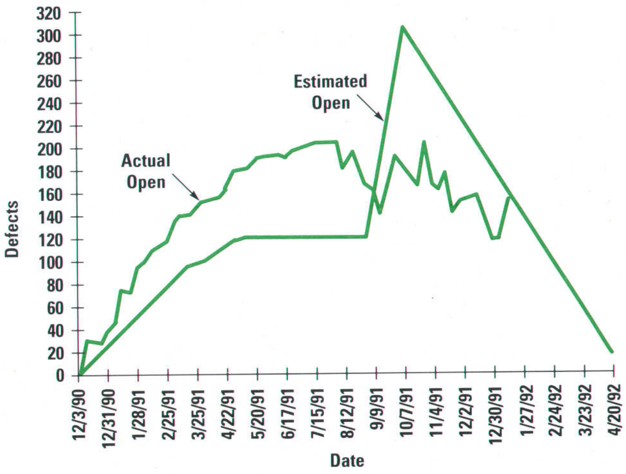


Fig. 15. This report tracks open defect counts against estimates of projected open defects necessary to meet a scheduled completion deadline.

support have logged over 7500 defects against 25 major projects in 18 months of use.

Lessons Learned

As anticipated in early design sessions, choosing an interface toolset with incomplete graphical user interface capabilities proved to be a significant hurdle for many users. In its current form, DMS employs a character-based windowing scheme that runs in both X11 and ASCII keyboard environments. While this interface style maximizes connectivity, allowing virtually anyone with LAN access to use DMS, it has proven to be a tough sell to R&D users who expect tools to exhibit an OSF/Motif look and feel. As a result of the decision to trade off connectivity for X11 and OSF/Motif support, more non-X11 users have access to DMS at the expense of X11 users who are inconvenienced by a more primitive interface.

While an evolutionary delivery can be used to deliver just-in-time functionality to users, one cannot underestimate the importance of user involvement in making design decisions and prioritizing implementation tasks. The users group and steering committee proved to be successful tools in guiding the evolution of DMS. The users group is an open forum that allows communication between users and developers. The steering committee consists of a group of expert users who ensure that the tool evolves in a consistent direction.

Conclusion

DMS has achieved its objective as an "industrial-strength" defect management solution. Since its introduction, it has been used to manage defect information for many flagship products over many divisions. It has proven itself as a 24-hour-a-day workhorse, serving as many as 40 to 50 simultaneous users during normal business hours. In fact, the real-time reliance on DMS has necessitated scheduled maintenance during late night and weekend hours. DMS enabled us to extend its contribution into the R&D community by providing the services of a self-contained software process tool with minimum administration. As a result, DMS has increased the scale at which we can provide defect tracking services without incurring significant personnel increases.

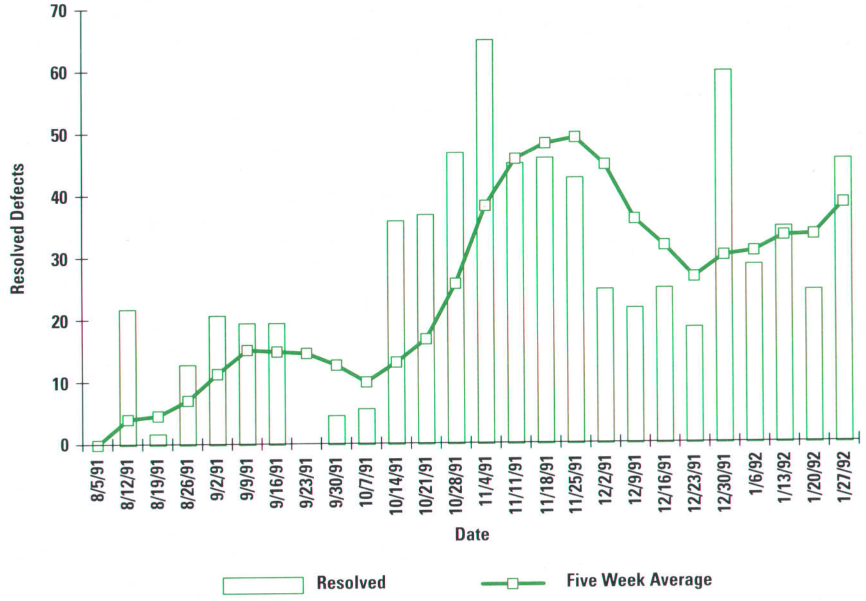


Fig. 16. This report shows overall project weekly defect fix rates as well as a moving average for the same.

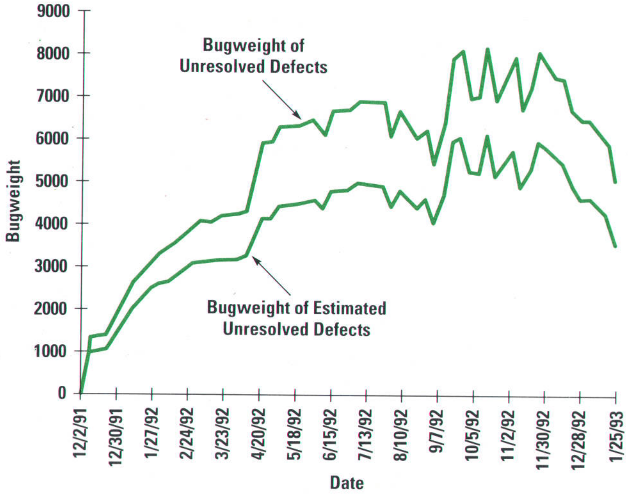


Fig. 17. This report gives a measure of overall software quality (bugweight) over time.

Another major success for DMS is the degree to which related projects can now share defect information. Based on the immediate acceptance and use of the defect-sharing capabilities of DMS, users now actively treat defect information as sharable, and are conscientiously communicating with other projects via this mechanism. For multisite operation, DMS successfully demonstrated the ability to cross-submit and cross-track over multiple sites transparently. By making use of passive server-to-server communication mechanisms, DMS servers at different sites can easily be configured to communicate defect information.

An area where DMS has greatly assisted R&D management is in metrics analysis. Using SQL, raw defect data is much more readily analyzed and converted into a digestible form. The rate at which more extensive ad hoc analyses can be delivered has greatly increased. In addition, protecting process and data integrity has enhanced the accuracy and reliability of all queries, ad hoc or standard.

Timeliness has been another major success of DMS. As can be seen in Fig. 19, DMS has succeeded in shortening much of the defect life cycle time from an average of days to hours.

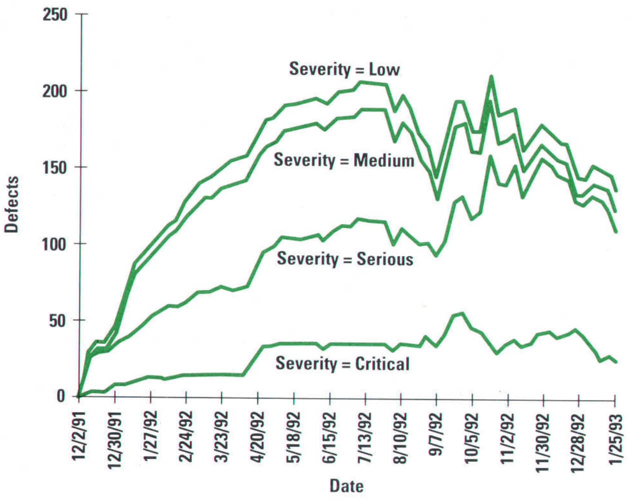


Fig. 18. This report provides a cumulative segmented view of unresolved defects by severity.

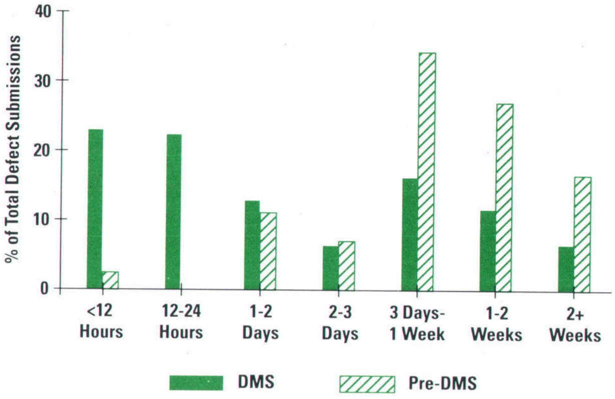


Fig. 19. The difference between the defect life cycles before and after DMS. This shows the elapsed time between defect find date and engineer assign date.

In addition to the direct benefits DMS has provided, we have observed some interesting cultural shifts in the R&D and test communities in the printer divisions at our site. Project team members have come to place great reliance in the ability to get instantaneous defect information. The rigorous process imposed by DMS ensures that all defects contain the minimally required set of information and that all of the information has been validated against centrally maintained lookup tables. Users have come to appreciate the ease with which defect information can be located and manipulated 24 hours a day, seven days a week.

DMS has also empowered users to manage defects within a proven process model. Given that the number of projects and the defects they generate will continue to increase, it is clear that the number of individuals needed to move a defect through the process needed to be minimized. DMS has been successful in that it has encapsulated a defect management process known to work for the laser printer firmware development process and has decreased the number of individuals needed to manage defect information. DMS allows project teams to manage all aspects of the defect process without the need for the intervention of defect tracking experts or other outside agencies.

In an unanticipated use, DMS has allowed us to share defects with third-party software vendors and still maintain security of internal defect information through the passive server data exchange mechanism. We have successfully used this mechanism to import and track defects that originated from a public DMS database server that was accessible to third-party engineers. This capability has generated much interest from project teams that use DMS and have extensive interaction with third-party software vendors.

Acknowledgments

We would like to thank all members of the HP Boise Printer Division and Network Printer Division teams for their assistance and perseverance in helping us refine DMS. Additional thanks to Craig Denson, Rick Smith, Virginia Pollack, Larry Allen-Tonar, and all members of the R&D teams in the San Diego hardcopy divisions.

HP-UX is based on and is compatible with UNIX System Laboratories' UNIX* operating system. It also complies with X/Open's* XPG3, POSIX 1003.1 and SVID2 interface specifications.

UNIX is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and other countries.

X/Open is a trademark of X/Open Company Limited in the UK and other countries.

Realizing Productivity Gains with C++

Although C++ contains many features for supporting highly productive software development, some characteristics of this object-oriented programming language tend to slow the realization of these productivity gains.

by Timothy C. O'Konski

In many cases there is a long delay between starting to work in the C++ language and realizing the potential productivity gains of the object-oriented paradigm, including code reuse. Before this delay can be shortened or eliminated entirely, practical issues relating to the multiparadigm nature of C++ and its C ancestry must be understood.

The object-oriented benefits of data abstraction and inheritance coupled with type checking give C++ a natural advantage when attempting to build both system and application software. Additional productivity gains can be obtained by reusing a class library if the following considerations can be met:

- Programming mechanisms contained within the class library must be understood by the programmer before they can be expanded and reused correctly.
- When selecting a C++ library class or class template, the size, performance, and quality characteristics of each class or class template component must be apparent to the programmer.
- Appropriate class or class template definitions must first be properly located by the programmer so they can be incorporated into the program currently under development.
- The time taken by the programmer to learn how to use the library correctly must be much less than the time necessary for the programmer to create new code. Otherwise, the programmer might attempt to rewrite the C++ library, inhibiting the productivity gain.

This paper describes our experiences with developing new C++ software and modifying existing C++ libraries. It also looks at possible uses of templates and exception handling defined in the new emerging ANSI C++ standard X3J16.

Mixing Programming Paradigms

The following discussion describes some standard C++ programming paradigms and their associated problems.

Concrete Data Types. Concrete data types are the representation of new user-defined data types. These user-defined data types supplement the C++ built-in data types such as integers and characters to provide new atomic building blocks for a C++ program. All the operations (i.e., member functions) essential for the support of a user-defined data type are provided in the concrete class definition. For example, types such as complex, date, and character strings could all be concrete data types which (by definition) could be used as building blocks to create objects in the user's application. The following code shows portions of a concrete class called date, which is responsible for constructing the basic data structure for the object date.

```
typedef boolean int;
#define TRUE 1
#define FALSE 0

class date {
public:
        date (int month, int day, int year); //Constructor
        ~date();                             //Destructor

        boolean set_date(int month, int day, int year);

        // Additional member functions could go here...

private:
        int year;
        int numerical_date;

        // Additional data members could go here...

};
```

Designers of concrete data types must ensure that users of this class will not want to add functionality to the class through derivation. Otherwise, the class must be designed to handle incremental additions in advance. Failing to do so could cause an ill-defined set of functions (for example, a missing assignment or copy constructor[1]) which in turn would cause a defect to be uncovered by unsuspecting users of the concrete data type.

Abstract Data Types. Abstract data types represent the interface to more than one implementation of a common, usually complicated concept. Because an abstract data type is a base class to more than one derived class, it must contain at least one pure virtual function. Objects of this type can only be created through derivation in which the pure virtual function implementation is filled in by the derived classes.

The following is an example of an abstract base class:

```
class polygon {
public:

        // constructor, destructor and other member functions
        // could go here...
        virtual void rotate (int i) = 0; //a pure virtual function
        // other functions go here...

};
```

Glossary

Although the terminology associated with object-oriented programming and C++ has become reasonably standardized, some object-oriented terms may be slightly different depending on the implementation. Therefore, brief definitions of some of the terminology used in this paper are given below. For more information on these terms see the references in the accompanying article.

Base Class. To reuse the member functions and member data structures of an existing class, C++ provides a technique called class derivation in which a new class can derive the functions and data representation from an old class. The old class is referred to as a base class since it is a foundation (or base) for other classes, and the new class is called a derived class. Equivalent terminology refers to the base class as the superclass and the derived class as the subclass.

Catch Block. One (or more) catch statements follow a try block and provide exception-handling code to be executed when one (or more) exceptions are thrown. Caught exceptions can be rethrown via another throw statement within the catch block.

Class. A class is a user-defined type that specifies the type and structure of the information needed to create an object (or instance) of the class.

Constructors. A constructor creates an object, performing initialization on both stack-based and free-storage allocated objects. Constructors can be overloaded, but they cannot be virtual or static. C++ constructors cannot specify a return type, not even void.

Derived Class. A class that is derived from one (or more) base classes.

Destructors. A destructor effectively turns an object back into raw memory. A destructor takes no arguments, and no return type can be specified (not even void). However, destructors can be virtual.

Exception Handling. Exception handling, which is a feature defined in the ANSI X3J16 Draft and implemented in HP's 3.0 C++ compiler, provides language support for synchronous event handling. This feature is not the same as existing asynchronous mechanisms such as signals which are supported by the underlying environment. The C++ exception handling mechanism is supported by the throw statement, try blocks, and catch blocks.

Member Functions. Member functions are associated with a specific object of a class. That is, they operate on the data members of an object. Member functions are always declared within a class declaration. Member functions are sometimes referred to as methods.

Multiple Inheritance. A derived class can be derived directly from one or more base classes. Any member function ambiguities are resolved at compile time.

Object. Objects are created from a particular class definition and many objects can be associated with a particular class. The objects associated with a class are sometimes called instances of the class. Each object is an independent object with its own data and state. However, an object has the same data structure (but each object has its own copy of the data) and shares the same member functions as all other objects of the same class and exhibits similar behavior. For example, all the objects of a class that draws circles will draw circles when requested to do so, but because of differences in the data in each object's data structures, the circles may be drawn in different sizes, colors, and locations depending on the state of the data members for that particular object.

Template. A class template provides a mechanism for indicating those types that need to change with each class instance. The generic algorithm associated with the class remains invariant. Later in the class instantiation, these types are bound to built-in or user-defined types.

Throw Statement. A throw statement is part of the C++ exception handling mechanism. A throw statement transfers control from the point of the program anomaly to an exception handler. The exception handler catches the exception. A throw statement takes place from within a try block, or from a function in the try block.

Try Block. A try block defines a section of code in which an exception may be thrown. A try block is always followed one or more catch statements. Exceptions may also be thrown by functions called within the try block.

Virtual Functions. A virtual function enables the programmer to declare member functions in a base class that can be redefined by each derived class. Virtual functions provide dynamic (i.e., run-time) binding depending on the type of object.

Other classes, such as square, triangle, and trapezoid, can be derived from polygon, and the rotate function can be filled in and defined in any of these derived classes. Note that polygon objects cannot be constructed. The C++ compiler will prevent this from happening because there is at least one pure virtual member function not yet defined.

Abstract data types sometimes suffer from too many functions being declared virtual. This adds both size and some slight overhead to the program's speed of execution. Inlining will usually compensate for the speed overhead incurred by a virtual function, but will add even more to the size of the program or library object file.

Node Classes. Node classes are viewed as a foundation class component upon which derived classes can be built. Designed to be part of a hierarchy, a node class relies on services from other base classes and provides some unique services itself. A node class defines any virtual functions necessary to change the object's behavior or fill in any pure virtual function definitions still left undefined in the derived class. Additional functions are also added by a node class to widen the behavior of an object. Node classes, by their very nature, will not suffer the fate of misconstrued concrete data types described above, but may suffer from some programming errors.

Common problems in declaring node classes stem from the fact that they are designed to be sources of object derivation. Because of this, the presence of any virtual functions (in either the base or any derived classes of the node class) will require the presence of a virtual destructor to ensure proper class cleanup. Because one cannot determine if and when a virtual function might be added by a class derivation, it is better to be safe and declare the destructor virtual in the base class. This is because the "virtualness" of the destructor cannot be added in any derived class. It must be part of the base class destructor declaration.

An additional problem common to a node class is improper verification of protected data members. Because a derived class can modify or change protected data members, they could be invalidated by any derived class. Adding assert statements to a special "debug" version of the node class that validates the protected data can detect this type of programming error.

Interface Classes. Interface classes are the most humble but important and overlooked of all classes. The purest form of an interface class doesn't cause any code to be generated. Consider the following unsafe class called List, which is wrapped by the class template SafeList:

```
template<class T>
class SafeList : private List<void*> {
public:
  void insert(T* p) { List<void*>::insert(p); }
  void append(T* p) { List<void*>::append(p); }
  T* get() { return (T*) List<void*>::get(); }
};
```

Here, a class template called SafeList is used to convert an unsafe generic list of void* pointers into a more useful family of type-safe list classes.[2] Type-safe means that the compiler checks for correct pointer types instead of allowing any pointer (e.g., void*) to be used within a list template. The very nature of a void* pointer is that it may contain a pointer to any object. By adding the SafeList template, we are ensuring that a List template can only contain pointers to classes that we have defined for use with a List template.

Interface classes are used to adjust an interface, provide robustness with a greater degree of type safety, or prevent member function names from two different class hierarchies from clashing.

Handle Classes. Handle classes provide an effective separation between an object interface and its implementation. Handle classes provide a level of indirection via pointers. Additional benefits include an interface to memory management and the ability to rebind iterators for a class representation. An iterator is a function that returns the next element in a list, array, string, or any collection of items each time it is called.

A handle class is a good candidate for a class template:

```
template <class T> class handle_class {
  T* representation;
public:
  T* operator–>() { return representation;}
  // ...
};
```

This code fragment shows how a handle class is used to manipulate pointers to objects of type T, instead of actual user-defined class representations of objects of type T. A problem with handle classes is that they require cooperation between the class being handled and the handle class itself.

C Roots

The fact that C++ is based on the C programming language is evident throughout the language. C is retained as almost a subset of C++ and so is the emphasis on C facilities that are low-level enough to deal with the most demanding system programming tasks.

Once a class definition has been agreed upon by a programming team, the programmers have the ability to proceed with implementation by using member function code stubs whenever necessary. This practice of filling in stubs with real function code when necessary in conjunction with C++'s static type checking enables a form of rapid prototyping via incremental development.[3] C++ allows iterative design and implementation to proceed in parallel, facilitating a more rigorous design methodology than conventional C programming.[4]

Because of the C roots of C++, most or all of the low-level programming tasks that are within the range of C are still within the scope of C++. However, some of the problems that have plagued C programmers also effect C++ programmers. The problems encountered in this regard include: uninitialized pointers, data narrowing, memory leaks, and conflicting #defines, typedefs, and enums.

Uninitialized Pointers. An uninitialized pointer might contain a garbage address, and if used in its uninitialized state may cause the program to abort.

```
int *pint;

// other code (*pint is not initialized to any address)

*pint = 9; // may result in a "Bus error(coredump)"
```

Data Narrowing. On a system where sizeof(short) is two, the code:

```
unsigned short s = 65535;
signed int i = s;
```

will silently change the value of s to be –1 when i is printed. This is because in the unsigned version, the high bit is used to increase the value of the unsigned short, while in the signed (i.e., int i) version, the sign bit is used to signify a negative number. When the unsigned value s is assigned to the signed int value i, the number changes from a large unsigned value to a small negative value because its high-order bit is interpreted in a different manner.

Memory Leaks. When a location that contains a pointer to memory is deallocated, a "memory leak" occurs (just as in C). This means that the location that contained the pointer to memory allocated out of free storage is no longer valid. Thus, the allocated memory cannot be accessed for the duration of the program. For example, in the sequence:

```
{
  char *s = new char[10];
  // some code here...
}
// the variable s is no longer accessible
```

the pointer s is out of scope and a memory leak will occur immediately after this code segment if a delete s operation is not performed before the end of this code fragment.

#defines, typedefs, and enums. Problems with these declarations occur on a per-program basis when declared at file scoping. For example,

```
// header file #1:

typedef int boolean;
// ...

<eof>
```

and then:

```
// header file #2:

typedef unsigned boolean;
// ...

<eof>
```

will cause the linker to issue an error and abort because of conflicting typedef declarations in two different source files.

Typical Problems with Libraries

The ANSI C++ Committee X3J16 and a parallel ISO (International Standards Organization) committee are currently standardizing the C++ language. Over the past six years the C++ language has continued to evolve through five major releases. This moving target has resulted in libraries and programs that typically have upgrades that accommodate the new language features without taking full advantage of them.[5] This means that the programmer must make decisions regarding which feature is the correct one to use with each new release of a class library.

Requiring C++ library users to be conversant with both the previous and current C++ versions is a hardship on the C++ programmer. As a result some programmers have completely avoided new versions of C++ and stayed with the C++ release upon which their product is based. This problem will subside significantly when the X3J16 committee work becomes solidified into a draft standard.

The traditional object-oriented approach of using class derivation (i.e., inheritance) to reuse existing functionality is not necessarily the best way to make use of C++ classes to provide a *has-a* relationship as opposed to the traditional inheritance use to provide an *is-a* relationship. Is-a relationships are provided for by C++ via inheritance, which is commonly known as a class derivation. For example, if class B is derived from class A, B has all the characteristics of A plus some of its own, and we can safely say that B is-a kind of A. Has-a relationships are supported by composition rather than by inheritance. Composition is implemented by making one class a member of another class.[6] For example, we have a has-a relationship if B is contained in A.

It is still not clear whether to use multiple inheritance to combine the features of two different class libraries (i.e., both via is-a relationships) into a new class. One school of thought argues that multiple inheritance gives the class designer much more flexibility than single-inheritance class relationships.[5,7,8]

Classes that incorporate the new exception handling mechanism (described below) and also reside in multiple libraries do not yet exist on the marketplace. Therefore, conclusive evidence regarding the utility of multiple inheritance as a language feature to be used when combining classes from multiple libraries cannot be constructed until such C++ libraries exist and are successfully reused.

Templates

Templates provide a type-safe way of creating what is essentially a macro-like textual substitution mechanism and manipulating different types in a generic fashion. Templates provide a way to define those types that need to change with each class instance. Templates are created by parameterizing types in a class definition. These parameters act as placeholders until they are bound to actual types such as int, double, short, and char. For example, in the following code fragment, which is a template for an array class, Alphanum is the parameterized type.

```
// ...

const int arraysize = 16;
template <class Alphanum>
class array {
public:
  array(int sz=arraysize)
      {size=sz; indx=new Alphanum [size]; }
  virtual ~array() {delete indx; }

  // ...

private:
  int size;
  Alphanum *indx;

  // ...
};
```

When this template is used, objects of the array class might look like:

```
main ()
{
    array <int> intx (2); // integer objects...
    array <double> doublex (2); // double objects...
    array <char> charx (2); // character objects...
}
```

This shows that the actual type is substituted for the generic Alphanum defined in the template.

Using template classes creates a need for specific configuration management and tool support.[6] Additionally, template syntax is complicated and makes the code more difficult for others to understand. Tool support is needed to help cover the template syntax issues and for manipulating the interactions between templates, classes, and exceptions.

Exceptions

An exception is an event that occurs during program execution that the program is typically not prepared to handle. This event usually results in the program transferring control to another part of the program (exception handler) that can handle the event. Exception handling is necessary for robust, reusable libraries. Since exceptions may cause resources to be released in an unexpected manner, acquisition of resources and appropriate cleanup is a new requirement on class libraries. The typical mechanism of acquisition and release of files can easily be handled by using object constructors and destructors as shown in the following example.

```
class FilePtr {
  FILE* p;        // declare pointer to a file...
public:
  FilePtr(const char* n, const char* a) //class constructor
  { p = fopen(n, a); }

  FilePtr(FILE* pp) { p = pp; }
  ~FilePtr() { fclose(p); }              // class destructor closes file...

  operator FILE* () { return p; }
};
```

With this object class, a file pointer p can be constructed with either a File* or the arguments required for an fopen(). In either case FilePtr will be destroyed at the end of its scope and the destructor will close the file. A simple example of this programming technique would look like:

```
void use_file(const char* name)
{
  FilePtr f(name, "r");
  // use f...
}
```

The destructor will be called regardless of whether the function is exited normally or an exception occurs.

Other C++ language issues that need to be considered when reusing C++ libraries that incorporate exceptions include:

- Converting existing libraries to handle exceptions properly
- Remapping unexpected() and terminate() functions
- Combining the exceptions of one library with those of another library in a single program
- Handling asynchronous events (e.g., signals) and synchronous C++ exceptions simultaneously.

The following example program shows how multiple threads of control, which are represented by HP-UX* asynchronous signals and synchronous C++ exceptions, do not work together simultaneously. The second throw statement (rethrow) in the myhandler portion of the code, which tries to transfer control outside the exception handler, will not work at this time. The compiler cannot detect this condition because of the possibility of separate compilation of the signal-handler code and the code that traps to the signal handler.

```
#include <unistd.h>
#include <stream.h>
#include <signal.h>

/*
 * types needed below (used to be in <signal.h>, but were removed in
 * HP-UX 8.0)
 *
 * Note: This program (and the other code in this article) was compiled on an
 * HP 9000 Model 730 running HP-UX A.08.07 using HP C++ A.03.00
 */

typedef void SIG_FUNC_TYP(int); /* for UNIX* System V compatibility */
typedef SIG_FUNC_TYP *SIG_TYP;
#define SIG_PF SIG_TYP

int i = 0;

/* The function myhandler is called when the SIGINT is detected by the
 * program; after which a “sleep” and then a “throw” is performed (i.e., in
 * a synchronous manner). PLEASE NOTE: This signal handler could reside
 * in a separate compilation unit, making it impossible for a compiler to
 * check for this error condition.
 */

void myhandler()
{
     try {
       (void) signal (SIGINT, (SIG_PF) myhandler);
       cout << “in myhandler now...\n” << flush;
       sleep(1);
       throw i; // error: NO throws allowed in signal handlers if they
                 // are not caught in the signal handler
     }
     catch (int i) {
     cout << “catch inside myhandler now...\n” << flush;
     throw;  //this is an error because rethrows (or throws)
             // are not allowed to propagate outside a signal handler
       }
}

/* This main program waits for a SIG_PF, (i.e., usually CTRL/C)
 * which causes a core dump because of the throw propagation restriction
 * mentioned above. This mixture of asynchronous signals and the
 * synchronous exception handling causes C++ to exhibit a routine failure.
 * Therefore, this construct should be avoided.
 */

int main ()
{
       cout << “starting the program..\n” << flush;

       // Arm our signal handler...
       (void) signal (SIGINT, (SIG_PF) myhandler);

       // forEVER loop...
       for (;;)
       {
         try { // Now that we are in a try block, let's throw something...
           throw i;
         }

         catch (int i)
         { // Now we're in the catch block, so let us notify the user and
           // sleep for a moment...

           cout << “in main catch now...\n” << flush;
           sleep(1);

         }
       }

     // we'll never get here, but for completeness...
     return 0;
}
```

Conclusion

C++ is an effective language for promoting both incremental development and code reuse. The additional capabilities of templates and exceptions need to have more idioms formalized for their proper use. Because of C++'s increasing complexity, stronger environmental support is critical for the continuation of the language's success.

Acknowledgments

I would like to thank Tony Fanning, Walter Underwood, Marv Watkins, Dan Hoffman, and Susan Meade for their help and guidance in making this paper a reality.

References

1. J. O. Coplien, *Advanced C++ Programming Styles and Idioms*, Addison Wesley, 1992.
2. B. Stroustrup, *The C++ Programming Language, Second Edition*, Addison Wesley, 1991.
3. G. Booch, *Object-Oriented Design*, Benjamin Cummings, 1991.
4. T. O'Konski, D. Durham, B. Lin, J. Schofield and Pau-Ling Yen, *Object-Oriented Languages Evaluation Report*, HP Internal Document, 1986.
5. J. Waldo, “Controversy: The Case for Multiple Inheritance in C++,” *Computing Systems*, Vol. 4, no. 2, 1991.
6. S. B. Lippman, *C++ Primer, Second Edition*, Addison Wesley, 1991.
7. T.A. Cargill, “Controversy: The Case Against Multiple Inheritance in C++,” *Computing Systems*, Vol. 4, no. 1, 1991.
8. S. C. Dewhurst and K. T. Stark, *Programming in C++*, Prentice Hall, 1989.

HP-UX is based on and is compatible with UNIX System Laboratories' UNIX* operating system. It also complies with X/Open's* XPG3, POSIX 1003.1 and SVID2 interface specifications.

Bridging the Gap between Structured Analysis and Structured Design for Real-Time Systems

A real-time software design technique has been applied to the design of the software architecture for ultrasound imaging products.

by Joseph M. Luszcz and Daniel G. Maier

Structured analysis (SA) and structured design (SD) are two widely used methodologies for software development.[1,2] Structured analysis specifies the functionality to be implemented by a software system, and structured design is used for partitioning a single task into a set of functional modules. See "Structured Analysis and Structured Design Refresher" on page 92 for a brief review of the structured analysis and structured design notation and terminology used in this paper.

When designing and implementing a software system represented by a structured analysis model, it is usually necessary to partition the functionality among a number of concurrent tasks to meet the timing constraints placed on the software system. In addition, to achieve a design with the characteristics of low coupling and high cohesion, it is desirable to partition the functionality into objects or packages for data hiding.

Although structured techniques provide designers with a methodology for partitioning a complex system into manageable pieces for analysis and design, there are some problems in making the transformation from SA to SD for real-time systems design. For example, the transformation from SA to SD does not easily support concurrency. Processes need to be grouped into concurrent tasks before detailed design. Another example is related to object-oriented design. SA and SD do not strongly support producing well-encapsulated objects.

Because of these problems a software developer, after specifying a real-time system using SA techniques, is often not sure how to proceed to a design and typically resorts to ad hoc design techniques. A methodology is needed to help the designer bridge the gap between SA and SD.

The ADARTS Solution

ADARTS (Ada-based Design Approach to Real-Time Systems)* is a high-level design methodology that effectively fills the gap between SA and SD by providing a systematic means (called process steps) for partitioning an SA specification model into a set of tasks, packages (objects), and communication links, which can then be designed using SD. The deliverables from ADARTS include a set of high-level architectural diagrams and a set of specifications called component interface specifications for each task and package. These deliverables are described later in this article. Fig. 1 shows a simple overview of how ADARTS fits into the software development process with SA and SD. The notation and graphic symbols for the ADARTS diagram shown in Fig. 1 are described in more detail later in this paper.

We have been using ADARTS for embedded software development at Imaging Systems Division (ISY) since early 1990. ADARTS helped us deal with the complexity inherent in the design of the ISY shared software architecture, and we found it indispensable in turning SA models into realizable designs.

While the ADARTS technique supports the synchronizing constructs inherent to the Ada programming language, it is not necessary to program in Ada to derive the majority of the benefits from ADARTS. We easily adapted the methodology and its diagramming terminology to a more conventional operating environment consisting of high-level language programs running under the control of a real-time operating system. At ISY, we develop software in the C language running under the pSOS-68K operating system. However, our approach to using ADARTS works with any language.

The diagramming notation used in ADARTS is based on the Buhr notation,[4] which is used to represent the tasks, packages, and communication paths resulting from the design decisions made in ADARTS (see Fig. 2). This diagramming notation is supported in commercially available CASE tools.

The rest of this paper gives a brief overview of the ADARTS methodology (defining the notation shown in Fig. 2 along the way) and then presents an example of our experience with using ADARTS for software architecture design.

ADARTS Process Steps

The ADARTS process involves following the steps listed below to create the ADARTS deliverables mentioned above and then using the deliverables to design and implement the system.

1. Develop a real-time structured analysis specification, and level the data flow diagrams to create a single (flat) diagram from the hierarchical set of data flow diagrams in the original model.

* Although ADARTS uses Ada constructs, its use is not limited to Ada. ADARTS was created by a group of companies called the Software Productivity Consortium (SPC). Two versions of the ADARTS specification have been produced by the consortium. This paper is based on the first version.[3]

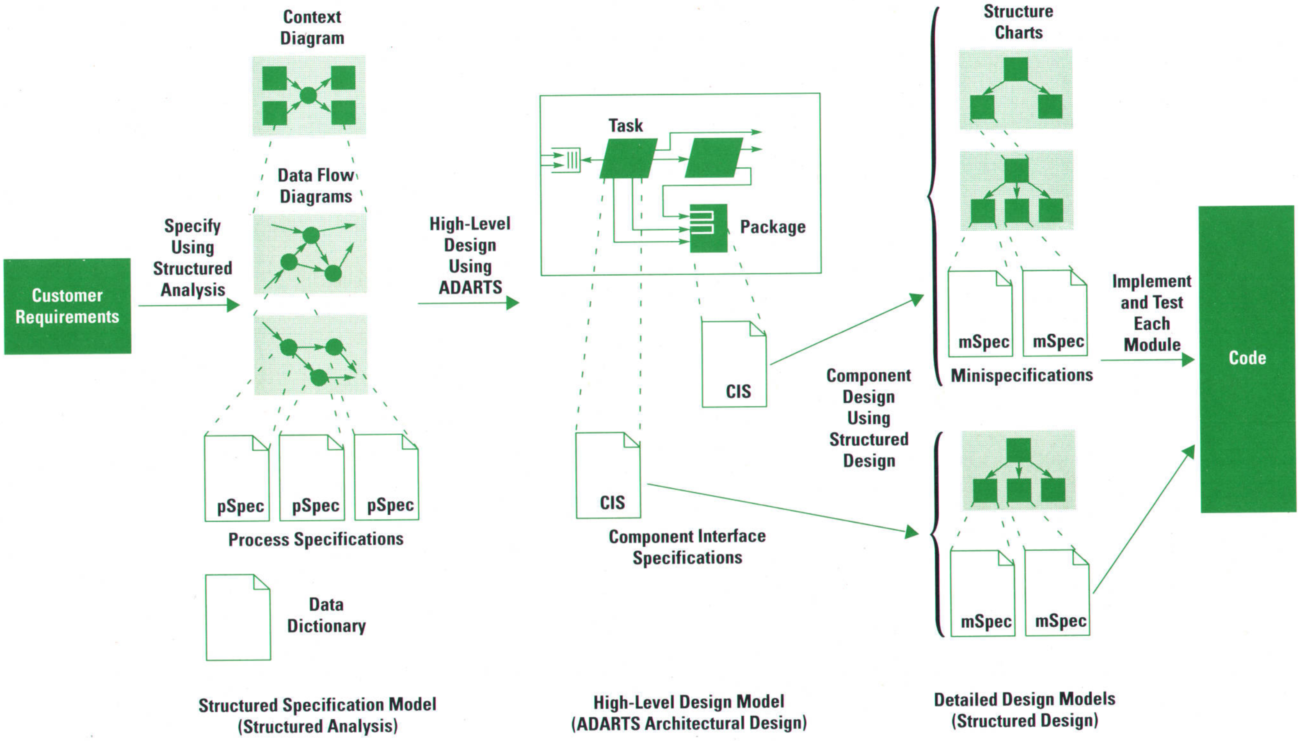

Context Diagram
Data Flow Diagrams
Customer Requirements
Specify Using Structured Analysis
pSpec
pSpec
pSpec
Process Specifications
Data Dictionary
High-Level Design Using ADARTS
Task
Package
CIS
CIS
Component Interface Specifications
Component Design Using Structured Design
Structure Charts
mSpec
mSpec
Minispecifications
mSpec
mSpec
Implement and Test Each Module
Code
Structured Specification Model (Structured Analysis)
High-Level Design Model (ADARTS Architectural Design)
Detailed Design Models (Structured Design)

Fig. 1. The role of ADARTS in the software development process.

2. Identify concurrent tasks by applying task structuring criteria, and determine the kind of communication and synchronization mechanisms for the data and events passed between tasks. Task structuring involves combining those processes that should be combined or keeping separate those processes that must be separate.

3. Identify packages (objects) by applying the package structuring criteria to produce well-encapsulated software objects.

4. Add support tasks to provide required synchronization and message buffering services. An example of a support task is a task that synchronizes access to a data store that is accessed by multiple tasks.

5. Package the tasks (we skipped this Ada-specific requirement).

6. Develop an NRL (Naval Research Laboratory method) module hierarchy (we skipped this step).

7. Define component interface specifications.

8. Perform the detailed design of task and package internals using structured design or an alternative methodology.

9. Implement (code) the tasks and packages.

10. Store the completed components and design documentation in a (reuse) library.

Note that the ADARTS process steps are all-inclusive, covering the development of the software system from conception to delivery. Steps 2 through 7 are the contributions of ADARTS that are above and beyond SA and SD.

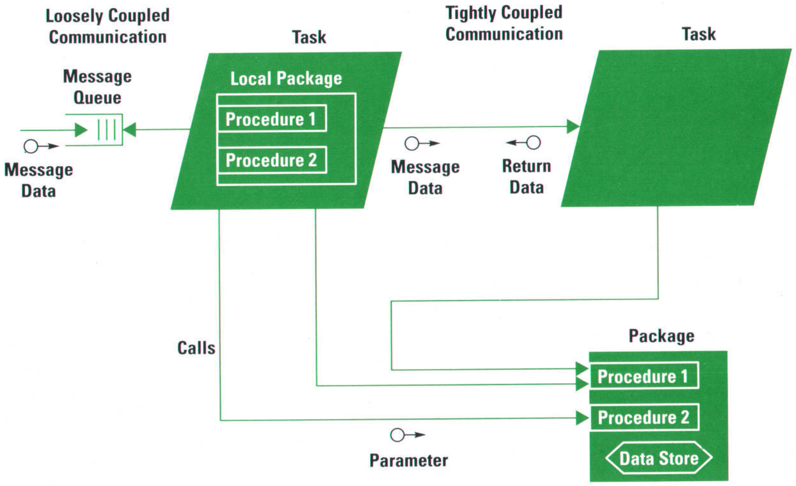


Fig. 2. ADARTS notation.

Structured Analysis and Structured Design Refresher

Structured analysis and structured design concepts have been in use for several years now, and thus the concepts, terminology, and graphic symbols are fairly well known. The following are some very brief definitions of the SA and SD graphics symbols shown in Figs. 1 and 2 and used in the accompanying article. Fig. 3 shows the process flow from customer requirements to code when SA and SD are used in software development. References 1 and 2 in the accompanying article contain much more information about SA and SD techniques.

Structured Analysis Notation

Structured analysis notation and methodology provide:

- A graphic and concise representation of software functionality
- A technique for partitioning a problem into manageable pieces
- A way to represent software functionality in a nonredundant fashion
- A way to create a logical model of what the software system does rather than how to do it.

The following definitions are associated with the notation in Fig. 1.

Data Flow Diagram. A data flow diagram is a network of related functions or processes and the corresponding data interfaces between these components. The notations shown in Fig. 1 that are used to depict a data flow diagram consist of:

- Labeled arrows, which show the data flows (information flow) between processes
- Circles (or bubbles), which represent the processes or transformations being modeled by the system
- Two parallel lines, which represent data stores, or places where information can stored
- Rectangles, which represent the data sources and destinations (sinks) in the system.

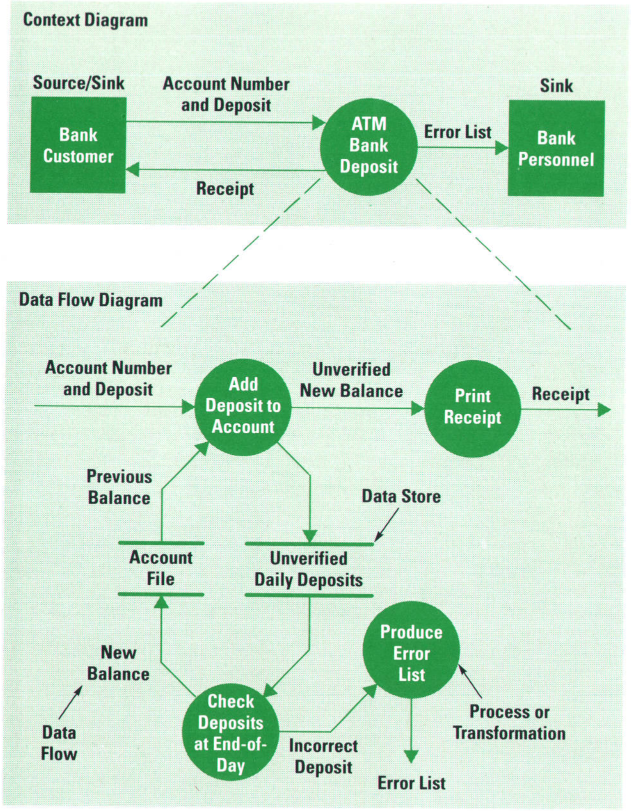


Fig. 1. The essential elements of structured analysis notation using bank transactions as an example.

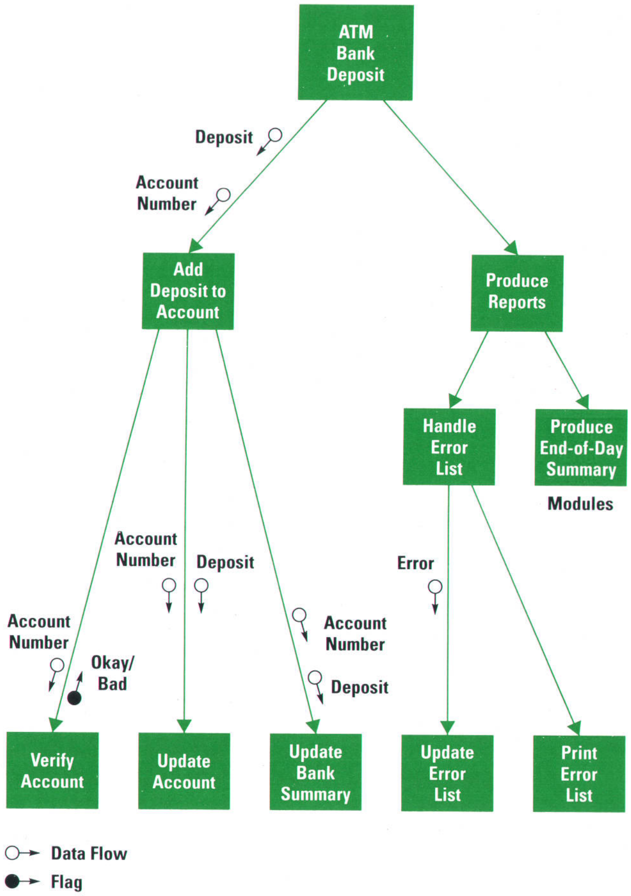


Fig. 2. A structure chart showing the essential elements of structured design notation for the bank example.

Task Structuring Criteria

Task structuring criteria are rules that guide the designer in combining SA process transformations (process bubbles or pSpecs) and control transformations to form concurrent tasks, while separating those transformations that need to be separate into independent tasks. These criteria reflect the same reasoning that an experienced real-time system designer might use when deciding on a concurrent task structure. ADARTS organizes these criteria for systematic application to software design.

The following are ADARTS task structuring criteria for combining transformations to create concurrent tasks:

- Sequential cohesion. Combine transformations that execute in sequence with other transformations, such as a state machine and the processing that occurs on a state transition (see Fig. 3).
- Temporal cohesion. Combine transformations that must run at the same time as other transformations, such as transformations that must respond to the same event (interrupt) or the same time tick. Fig. 4 shows the transformations that take place when a sensor monitoring patient temperature senses an out-of-limits temperature.
- Functional cohesion. Combine transformations that perform one or more related functions. These functions typically operate on the same data structure or same I/O device. For

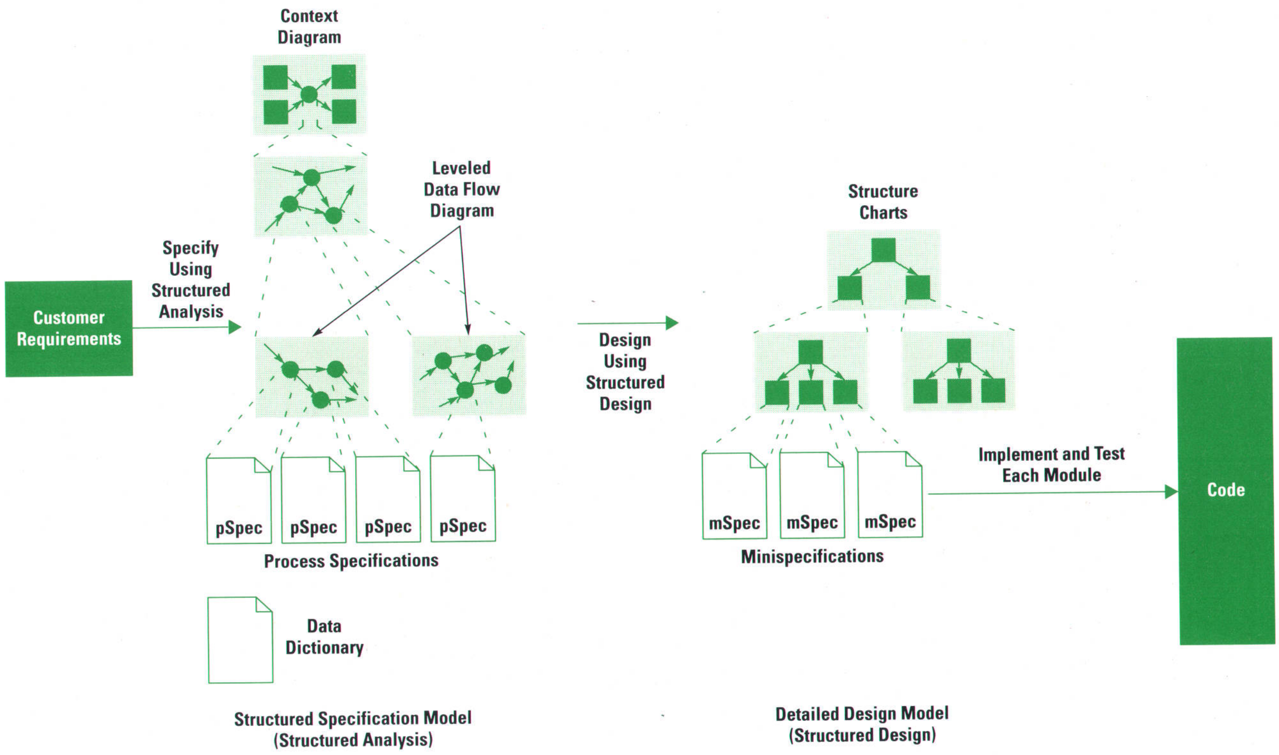

Context Diagram
Leveled Data Flow Diagram
Specify Using Structured Analysis
Customer Requirements
pSpec
pSpec
pSpec
pSpec
Process Specifications
Data Dictionary
Structured Specification Model (Structured Analysis)
Design Using Structured Design
Structure Charts
mSpec
mSpec
mSpec
Minispecifications
Implement and Test Each Module
Code
Detailed Design Model (Structured Design)

Fig. 3. The role of structured analysis and structured design in the software development process.

Context Diagram. This is the top-level diagram which shows the environment in which the software system being designed is supposed to work. The context diagram consists of one circle and the sources and sinks in the system's operating environment. This diagram forms the basis from which the rest of the system is designed. One or more levels of data flow diagrams can be derived from the context diagram. Multiple levels are created by partitioning the processes in the data flow diagrams.

Data Dictionary. The data dictionary is used to define all data flows and components of data flows and data stores. It provides a single place to record information that is necessary to understand the data in the data flow diagram.

Process Specifications. When a process can no longer be partitioned, the resulting processes are called primitive processes. Process specifications, or pSpecs, are used to describe these primitive processes. The notations commonly used in pSpecs include structured English, decision trees, and decision tables.

A data dictionary and process specifications are symbolically represented in Fig. 3.

Structured Design

Structured design is the process of refining the output from the structured analysis phase to design the module structure that will lead to a particular implementation of the software system. The steps in the process include:

- Derive a representation of the program structure with a structure chart. While the structure chart can be created from any system specification, it is typically created from a flattened data flow diagram. The structure chart consists of three basic features: boxes representing modules, arrows connecting the modules, and short arrows with circular tails representing data passed from one module to another (see Fig. 2).
- Expand the high-level definition by identifying lower-level modules needed to carry out the higher-level functions.
- Improve the representation by employing the design principles of cohesion and coupling. Coupling measures the degree to which modules depend on each other, and cohesion measures the degree to which elements within a single module are related to each other.
- Complete the detailed module design by employing a procedural logic description such as a minispecification (mSpec). An mSpec is similar to a pSpec for processes in structure analysis, except this time each module is being documented.

example, a process that computes trip statistics contains functions that access a database of collected trip data and then compute the required statistics (see Fig. 5).

The criteria for separating transformations into independent tasks include:

- Event dependency. Use a separate task for each transformation or group of transformations dependent on:
  - ○ Device I/O constraints such as responding to asynchronous I/O requests
  - ○ User interface constraints such as independent users or user interface sequential activities such as windows
  - ○ Periodic events (events that initiate transformations at regular time intervals).
- Control Transformation. Use a separate task for each independent control transformation such as a state machine controller or a transformation that is enabled and disabled by a control transformation.
- Task Priority. Use separate tasks for time-critical or computation-intensive activities.
- Multiprocessing. Use separate tasks for transformations that must execute on separate physical processors.

ADARTS specifies the order in which task structuring criteria should be applied so that the first criterion assigned to a transformation is usually the predominant one. However, subsequent criteria may contradict the original classification, and when that happens the original decision should be

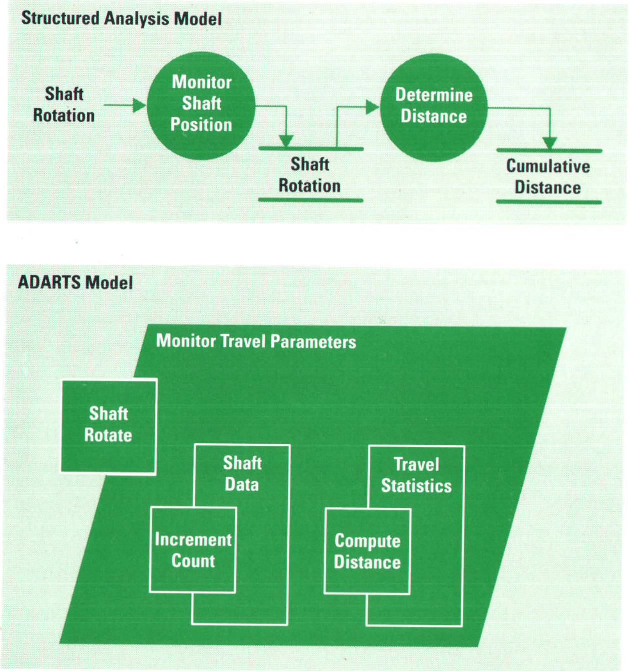


Fig. 3. An example of sequential cohesion in which two transformations that occur in sequence are combined into one task.

reconsidered using good engineering judgment to decide which criterion is dominant.

Intertask Communication and Synchronization

Once the task structure has been defined, the data and event interfaces between tasks must be determined. The ADARTS process provides guidelines for choosing how each data and event flow will be passed between tasks:

- Tightly coupled communication. This type of communication involves sending messages or events and then waiting for a response. A model of tightly coupled communication is shown in Fig. 6a.
- Loosely coupled communication. This type of communication is implemented by a message or event queue with no response. In the producer-consumer model, the producer would continue to send messages to a queue without waiting for a response from the consumer, which extracts messages from the queue at its own pace (see Fig. 6b).
- Loosely coupled communication with multiple producers. This communication style is implemented by a FIFO buffer or a priority queue. This is the case in which many producers might try to communicate with one consumer at the same time (see Fig. 6c).

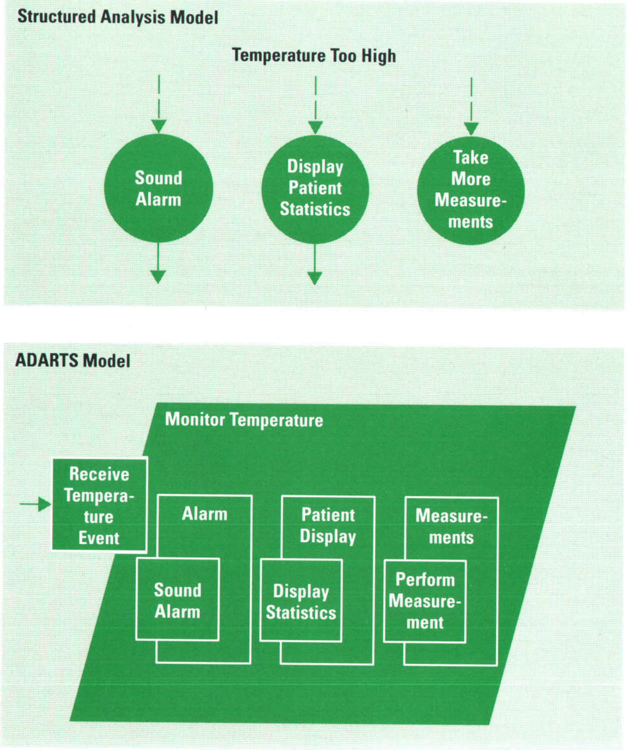


Fig. 4. An example of temporal cohesion in which three processes activated by the same event are combined into a single task.

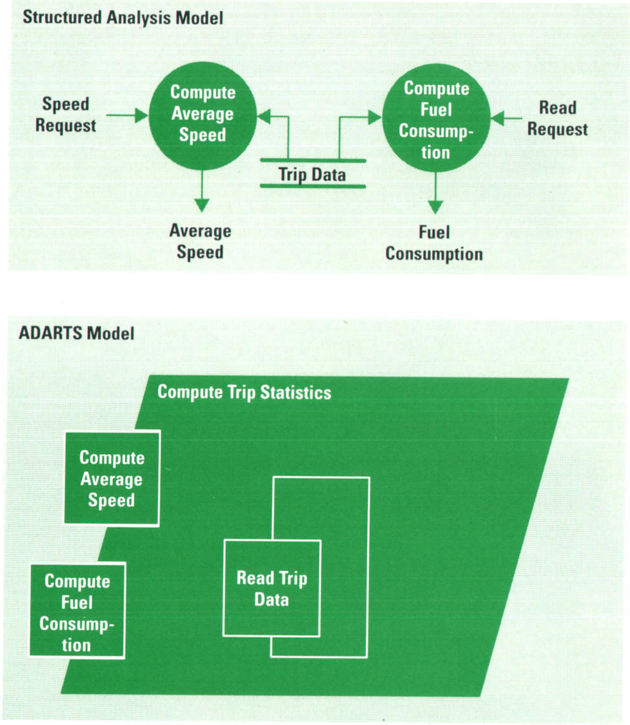


Fig. 5. An example of functional cohesion in which two transformations performing functions related to trip data are combined into a single task.

Each communication or synchronization flow is represented in ADARTS architecture diagrams by a distinguishing symbol, and each type of flow is implemented by a specific mechanism within the run-time environment of the system being designed.

Package Structuring Criteria

The package structuring criteria are rules for creating packages, or objects. The application of these criteria produces well-encapsulated software objects using the concept of information hiding. The ADARTS package structuring process does not contain many original ideas, but represents a compilation of existing ideas and strategies applied to a new domain (real-time systems). These rules fall into one of the following categories:

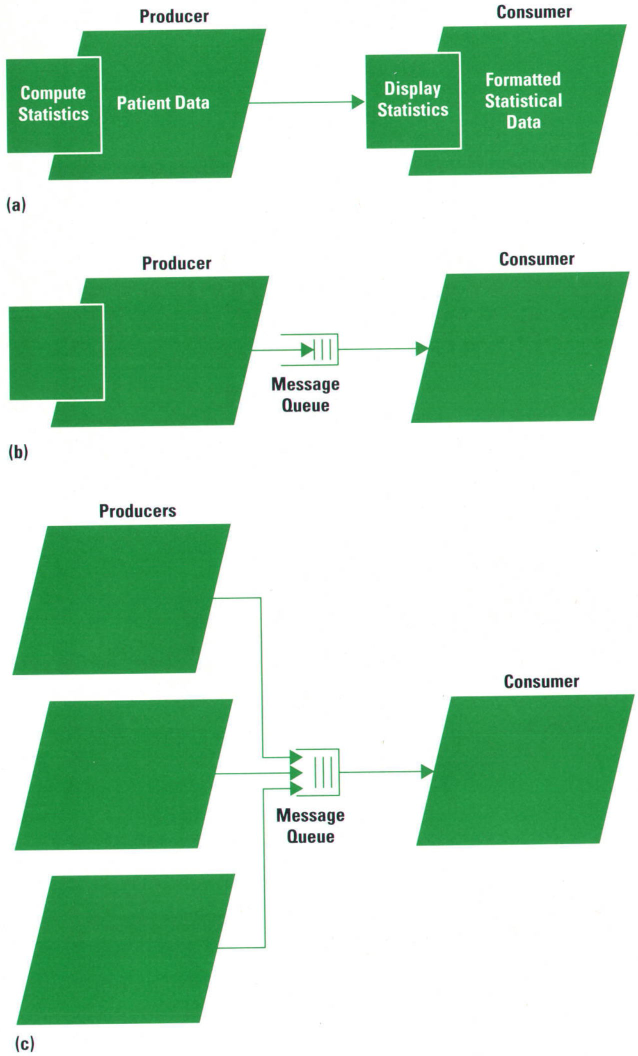


Fig. 6. ADARTS diagrams modeling intertask communication and synchronization. (a) Tightly coupled communication. (b) Loosely coupled communication. (c) Loosely coupled communication with multiple producers.

- Hardware hiding modules. These modules are used to encapsulate parts of the virtual machine such as the operating system or communications mechanisms or interfaces (e.g., device drivers) to particular I/O devices (see Fig. 7).
- Data abstraction packages. Each structured analysis data store becomes the basis for a data abstraction package, which hides system behavior requirements or software design decisions associated with data (see Fig. 8).
- Servers. Servers are passive modules that provide services for other tasks. Files servers and print servers are examples of these types of modules.

Component Interface Specifications

Component interface specifications (CIS) are textual descriptions of each ADARTS task and package containing information that is needed to inspect the high-level design and move to the detailed design and implementation of that component. Each component interface specification contains the name and type of the component, what the component does and when it does it, the operations provided by the component (including individual access procedures or functions

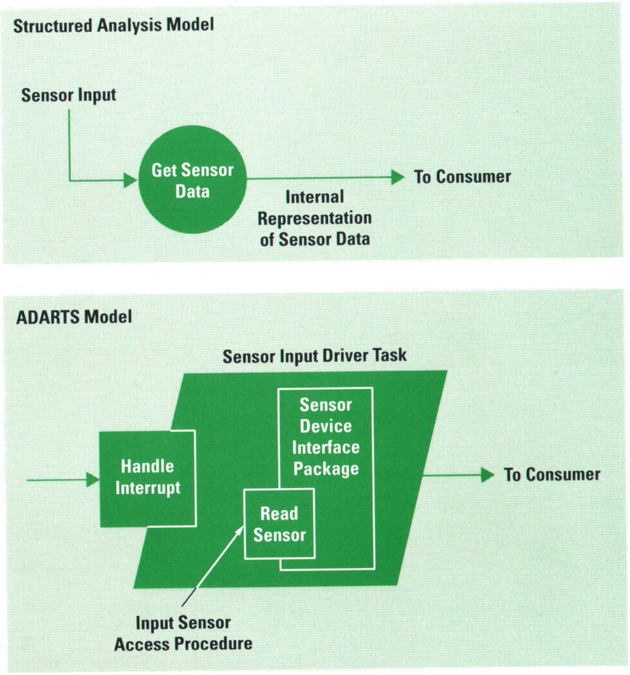

Structured Analysis Model
Sensor Input
Get Sensor Data
To Consumer
Internal Representation of Sensor Data
ADARTS Model
Sensor Input Driver Task
Handle Interrupt
Sensor Device Interface Package
Read Sensor
To Consumer
Input Sensor Access Procedure

Fig. 7. An example of a hardware hiding module showing the interface to an input sensor.

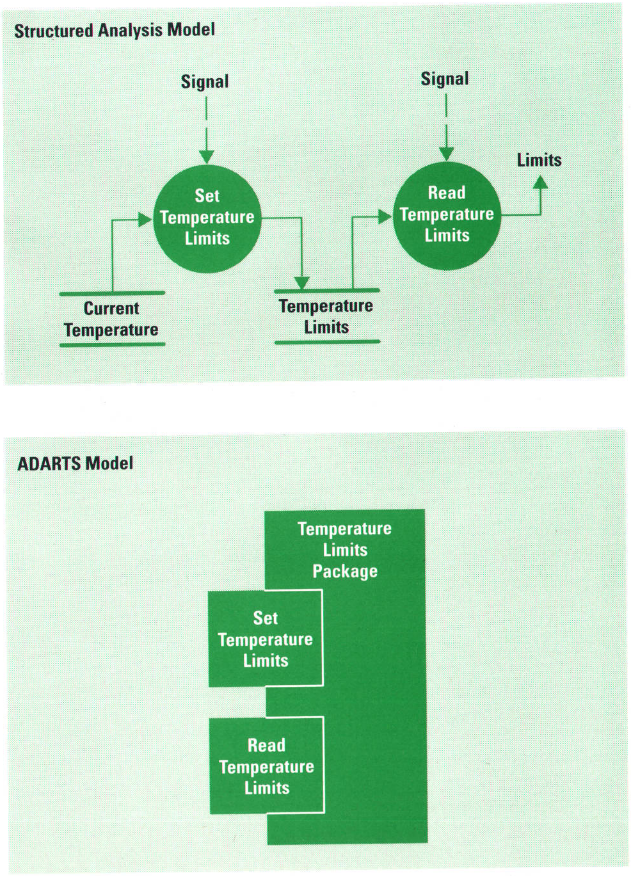


Fig. 8. A model of a data abstraction package.

NAME: Scanning_state_object

DESCRIPTION:

The scanning_state_object takes a state stimulus as input and produces the new scanning state (or an error code) as output (in the form of the scanning-state events). The object also has the ability to go explicitly to a particular state.

Errors will be explicitly listed in the state tables. This will make the state tables a better reflection of system operation.

A script will be written to compile descriptions of the scanning-state tables into ROM data tables. The scanning-state tables for various system configurations should all be explicitly present in the source files, then compiled by the script to produce the C source for the state tables.

DATA STORES:

Scanning-state transition table.
Initial scanning state – for use at power on/reset.

OPERATIONS:

Scnst_initialize( )

This entry point should be called at power-on/reset to set up the state transition table and initialize scanning state variables.

Scnst_chg_state (stimulus)

"Stimulus" is one of an enumeration of possible stimuli. This entry point chooses a new scan state based on the stimulus or produces an error code if the stimulus is not allowed in the current scan state. If the scan state is changed the new scan state is output as an event.

Scnst_goto_state (state)

"State" is one of the possible scanning states.
This is meant to be used by internal applications. It causes the scanning state to be changed to the indicated state.

Fig. 9. An example of a component interface specification.

with parameter definitions), and the effects of the component's operations. Fig. 9 shows an example of a component interface specification.

Experience with ADARTS

Architecture Design

We used ADARTS in the design of a software architecture for new ultrasound imaging products. This project was divided into two parts. The first part dealt with the development of the system software that provides a framework for application development and all generic services required in the application domain. The second part of the project dealt with the development of the software applications that provide the specific functionality required by a target system. Each of these two parts of the project used SA and ADARTS, but in slightly different ways.

First, the architectural structure and system components were developed. The high-level functions of the architecture were specified using structured analysis. This specification treated all functionality in terms of the general processing flow required for any application, without defining the specifics of any particular application. The specification model was validated by walking through test cases derived from a number of representative applications.

After the SA specification was complete, the ADARTS technique was applied to design the task and package structure and identify the communication mechanisms to be used. Component interface specifications were created, detailing the interfaces and functionality of each system component, followed by the detailed design and implementation of each component. Fig. 10 illustrates a key part of the ADARTS process, in which functionality is grouped into coherent tasks and packages using the appropriate structuring criteria. The sacks drawn around the various groups of data flow diagram elements in Fig. 10 show the application of the task and package structuring and intertask synchronization criteria described above. After several iterations these sacks were transformed into the simplified ADARTS architectural diagram shown in Fig. 11. The letters in Figs. 10 and 11 show the correspondence between the two system representations.

After the architecture was specified, designed, and implemented, attention was turned to the second part of the project—development of applications to run within the new architecture. Once again, structured analysis was used for specification of the software. However, we enhanced the design step by adding supplemental criteria to guide the designers in allocating application functionality to previously designed architectural components. For example, within certain architecture components places were left open to plug in application software that:

- Processes a keystroke (see vk (virtual key) functions in Fig. 11)
- Defines an application-specific parameter (see the agents data structure in Fig. 11)
- Adjusts an application parameter value when other parameters it depends on change (see the check routines function in Fig. 11).

The supplemental criteria helped the application designers determine where each aspect of the application functionality should reside within the architecture. The ADARTS methodology was thus used as a template for creating a more specific high-level design method. Detailed design for each component of the application then proceeded in the usual way.

Package Design

The degree to which the package structuring procedure of ADARTS was used during the project varied significantly. In some cases, the structuring criteria were applied rigorously. For example, in the continuous loop review application, which is an ultrasound application that supports acquiring video images into memory and playing them back as continuous loops, the criteria were applied to a leveled data flow diagram, leading to a highly modular ADARTS design consisting of objects with cohesive operations.

In many cases, ADARTS was used simply as a notation to show an object representation of a system's functionality. Some ADARTS designs were derived from a complete and leveled SA and others were derived from a high-level or abbreviated SA. Although the package structuring criteria were not explicitly used here, designers still applied the information hiding concepts recommended by ADARTS. For example, the Interpret Stimulus component shown in Fig. 11, which encapsulates the user interface functionality of the system, is quite complex. The ADARTS design for this component, although not derived from a complete SA, is very useful for showing the interactions between the packages. The component interface specifications for this component made it easier to understand the individual packages contained in the component.

Similarly, while the criteria developed to guide engineers in allocating specification functionality to architecture components were not always used explicitly, they communicated

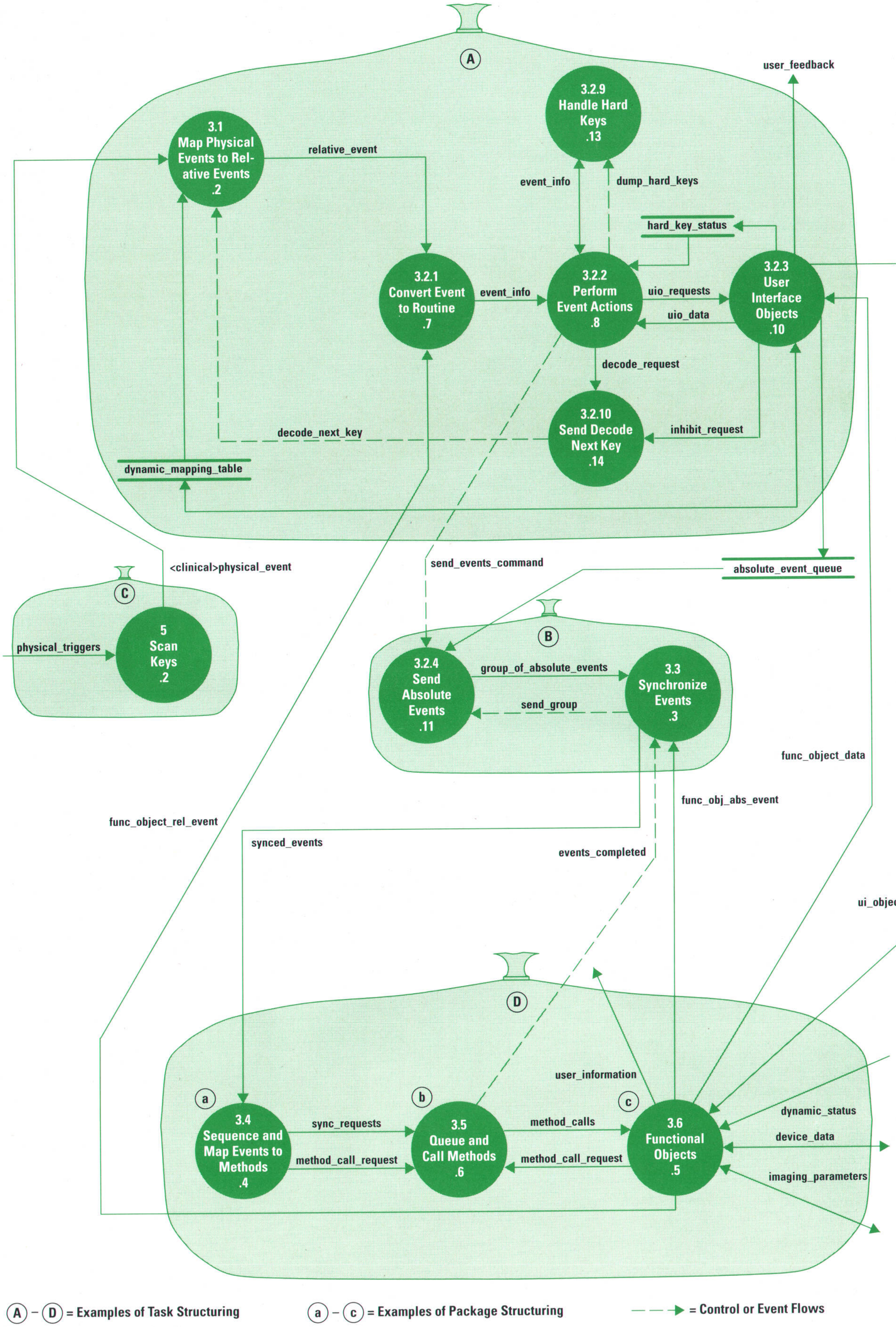
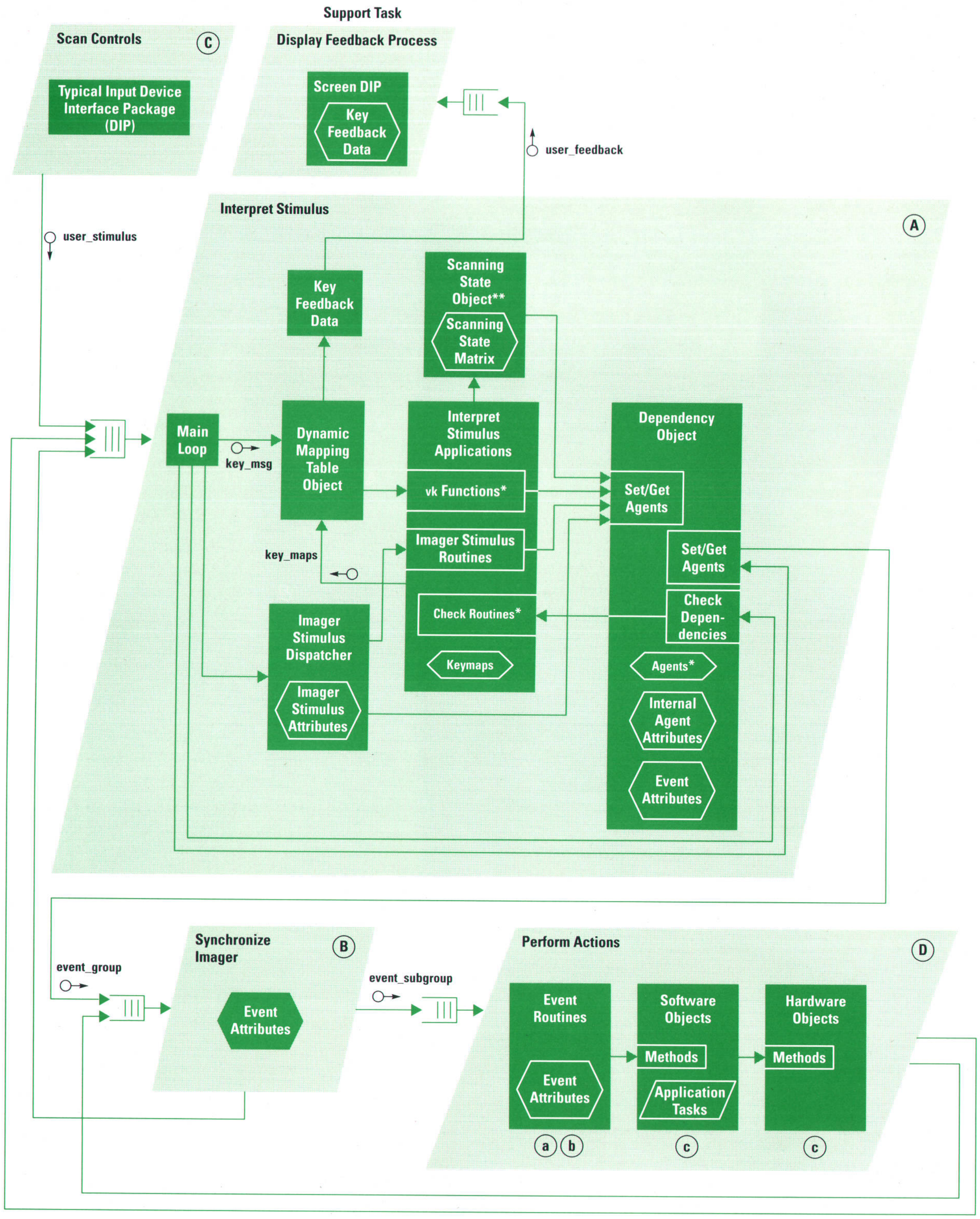


Fig. 10. A leveled structured analysis representation of the software architecture for the ultrasound imaging products.



* Some of the places left open in packages for plugging in application software.
** The component interface specification shown in Fig. 9 is for this package.

Fig. 11. The ADARTS architectural diagram obtained, after several design iterations, from the structured analysis diagram shown in Fig. 10.

to designers the choices that had to be considered when assigning the functionality of the application being designed to the appropriate components.

Tools and Techniques

We used a commercially available CASE product to generate the ADARTS diagrams. Since the product was targeted for Ada users, we had to deal with several drawbacks in using the tool because we were using the C language and a real-time operating system. For example, the product provided no direct support for the message queue symbol (used to show loosely coupled communication between processes), so we constructed the required symbol from primitive line structures. In addition, there was no integrated mechanism for creating component interface specifications and tying them to ADARTS components. Also lacking was a traceability mechanism for tracing system requirements from SA to ADARTS to SD.

Since the architecture software is a key part of the imaging product, we paid extra attention to using the development process to attain high-quality software. We adopted a general software development framework for the project. The steps in our process and the deliverables are summarized in Table I.

Table I
Development Process Using ADARTS

Phase	Deliverable
Requirements Generation	Software Requirements Specification
System Specification	Structured Analysis
Architecture Design	ADARTS
Detailed Design	Structured Design
Implementation	Source Code

Each step in this process was usually followed by an inspection by the appropriate individual or group.

Summary

The following is a list of the strengths and weaknesses we found by using this version of ADARTS in our environment.

Strengths. Some of the contributions and positive aspects of the ADARTS technique include:

- Continuity and task structure. ADARTS provides a linkage between an SA model and the detailed design of individual software modules by partitioning the specification into the optimal set of concurrent tasks and the appropriate communication mechanisms between them. SA and SD alone do not aid in the design of the overall concurrent task structure.
- Package structure. ADARTS, through its package structuring criteria, provides a method for achieving a reasonable object structure for the functionality represented by the SA model. SD alone, through its transform analysis and transaction analysis techniques,* is not effective for building encapsulated objects. Encapsulation is the predominant object-oriented design concept applied to our software development activities, and ADARTS supports this design aspect very well.
- Visibility. ADARTS design deliverables (architecture diagrams and component interface specifications) make a software design more visible, promoting more effective design inspections and making design concepts clear to other engineers who have a need to understand or maintain the software.
- Systematic approach. The steps used in ADARTS provide a systematic approach to system-level design, reducing the thrashing that can occur when following unstructured or ad hoc system design methods.
- Intuitive. ADARTS is easy to understand for new and experienced software engineers and intuitive to those familiar with real-time software design.
- Acceptance. ADARTS was accepted by the engineers using it at ISY, although their reasons varied widely. Each of the strengths stated above was cited by one or more engineers as the most valuable contribution of ADARTS.

Weaknesses. Just as we found many strengths in the ADARTS technique, we also found some weaknesses in using ADARTS in our environment. These weaknesses include:

- Object orientation. ADARTS falls short in its support for several of the currently accepted object-oriented design characteristics. For example, there is no provision for defining object classes or inheritance.
- Tools. Manipulating the architecture diagrams used in ADARTS (as well as the data flow diagrams and structure charts used in SA and SD) with the currently available CASE tools is time-consuming and has been a frequent complaint from engineers using these methods.
- Ada. We didn't have a need for the Ada-specific structures discussed in the ADARTS paper, and therefore we did not gain the full benefit inherent in the ADARTS methodology.

Conclusion

We found ADARTS to be an extremely effective technique for bridging the gap between a structured analysis specification and the structured design of the software modules that make up a software system. By providing a path between the two techniques, it makes both far more valuable than they would be otherwise. For structured analysis, the contribution to the definition of concurrent tasks and communication mechanisms is indispensable, but even if there is no concurrency required, ADARTS helps in identifying an object structure before applying the next detailed design step. Even if ADARTS is used on an SA specification that requires neither concurrency nor objects, it produces the trivial-case high-level design consisting of a single task in a single package which can then be constructed using SD. Thus, there is no harm in applying the technique to all designs.

Acknowledgments

The dedicated efforts of the members of the software architecture project team were key to the successful application of ADARTS within HP's Imaging Systems Division (ISY): Stephen Agyepong, Janice Bisson, Risa Bobroff, Marc Davidson, Paula Hart, Bill Harte, Peter Kelley, Martin Moynihan, and Kris Rovell-Rixx. We also thank Joe Kobrenski and Paul Kelly for their contributions in promoting the use of ADARTS within the project during its formative stages, and Bill Koppes and Arthur Dickey for supporting the group in its efforts to improve our development process during the project. Finally, we thank Dr. Charles Butler of Colorado State University for his guidance in the proper use of structured methods at ISY.

References

1. T. DeMarco, *Structured Analysis and System Specification*, Yourdon Press, 1978.
2. M.P. Jones, *The Practical Guide to Structured Systems Design*, Yourdon Press, 1980.
3. H. Gomaa, *ADARTS—An Ada-based Design Approach for Real Time Systems*, SPC-TR-88-021 Version 1.0, Software Productivity Consortium, August 1988
4. R. J. A. Buhr, *System Design with Ada*, Prentice-Hall, Inc., 1984.

* Transaction analysis is a design strategy based on a study of the transactions the system must process. Transform analysis is a design strategy based on the study of the data flows in a system and the transformations performed on that data.

Hewlett-Packard Company, P.O. Box 51827
Palo Alto, CA 94303-0724

ADDRESS CORRECTION REQUESTED

Bulk Rate
U.S. Postage
Paid
Permit No. 3913
Portland, OR

HEWLETT-PACKARD
JOURNAL

August 1993 Volume 44 • Number 4

Technical Information from the Laboratories of Hewlett-Packard Company

Hewlett-Packard Company, P.O. Box 51827
Palo Alto, California, 94303-0724 U.S.A.

Yokogawa-Hewlett-Packard Ltd., Suginami-Ku Tokyo 168 Japan

CHANGE OF ADDRESS: To subscribe, change your address, or delete your name from our mailing list, send your request to Hewlett-Packard Journal, P.O. Box 51827, Palo Alto, CA 94303-0724 U.S.A. Include your old address label, if any. Allow 60 days.

5091-8156E