HEWLETT-PACKARD
JOURNAL

[illegible]

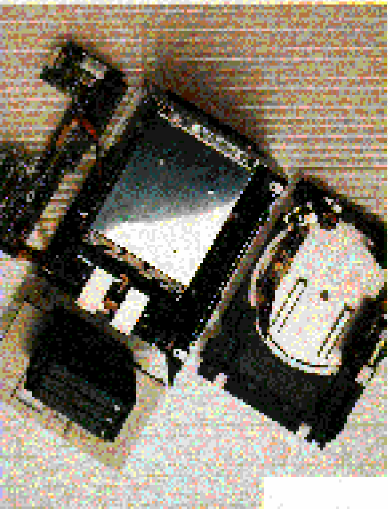


table of contents

December 1994,
Volume 45, Issue 6

Articles

10 Enterprise Modeling and Simulation: Complex Dynamic Behavior of a Simple Model of Manufacturing

by M. Shahid Mujtaba

Fast DDS-2 Digital Audio Tape Drive

Running at a data transfer rate of 510 kbytes/s, the HP C1533A tape drive can record a full 4-Gbyte DDS-2 cartridge in just over two hours, almost an hour less than typical DDS-2 drives. Its development required improvements in tape material, length, and thickness, new read and write heads, a new drum design, and new methods for linearity measurement and adjustment.

by Damon R. Ujvarosy

Like all aspects of computing today, the face of mass storage is changing rapidly. Only a few short years ago, gigabytes of disk storage was the domain of large computer systems, housed in computer rooms with dedicated staff to look after the equipment. Personal computers that had more than 100 megabytes of disk storage were a rarity, and networks were just coming of age. The individual computer user rarely considered backup. Critical data was kept on large computer systems and backup to tape was handled by the MIS department. The odd file on the PC that was important could be saved on a diskette.

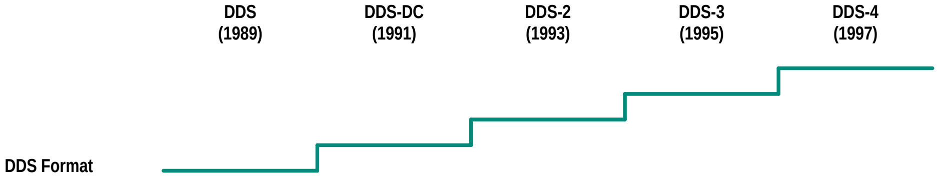
Times have changed rapidly. Individual PCs with several hundred megabytes of disk storage are common. Network servers for PCs and workstations have multiple gigabytes of disk storage. The data on these disks is critical to the company's business. High-performance, high-capacity backup solutions that fit the needs of today's computer systems are essential.

The same technologies that are propelling disk drive capacity and performance are also being applied to tape drives. Tape drives that meet the backup needs of the individual PC user are available using the DC2000 minicartridge—for example, the HP Colorado Memory Systems Jumbo 250 and Jumbo 700 tape drives. However, the backup needs of the network server are far greater. These larger backups also require improved performance to complete the data backup in a reasonable time.

The Digital Data Storage (DDS) format standard was developed by HP and Sony in the late 1980s to establish a capacity and performance point that would serve the emerging network server market as well as the more established small multiuser systems market. The DDS format standard is based on the Digital Audio Tape (DAT) standard, which uses 4-mm-wide tape. Tape drives that employ the DDS format standard are therefore often referred to as DAT or 4-mm tape drives.

In HP DAT drives, lossless data compression was added later to the DDS format standard using an HP-developed method called DCLZ,[1] an implementation of a technique known as Lempel-Ziv data compression.[2] DCLZ effectively doubled the capacity and performance of the tape drive, and at the same time, longer tapes were added. Four gigabytes could then be stored on a single DDS tape. Further extensions to the DDS format standard that will allow the DDS format to serve the backup needs of network servers through the end of the 1990s have been agreed to by the DDS manufacturers group (see Fig. 1).

The HP C1533A DDS-2 tape drive (Fig. 2), introduced in 1993, stores eight gigabytes of data (typical capacity



	DDS (1989)	DDS-DC (1991)	DDS-2 (1993)	DDS-3 (1995)	DDS-4 (1997)
DDS Format					
Enabling Technologies	• Audio DAT • Media Specifications • Format • Error Correction	• 90-m Tape • DCLZ Compression	• MP+ Media • 120-m Tape • Thinner Tracks	• MP ++ Media • Increased bpi • New Format • PRML Channel (Partial Response Maximum Likelihood)	• ME Media • 180-m Tape • Thinner Tracks
Uncompressed Capacity	• 1.3 GBytes	• 2.0 GBytes	• 4.0 GBytes	• 12.0 GBytes	• 24.0 GBytes
Uncompressed Transfer Rate	• 183 kBytes/s	• 183 kBytes/s	• 360 kBytes/s to 750 kBytes/s	• 500 kBytes/s to 1.5 MBytes/s	• 500 kBytes/s to 3 MBytes/s

Fig. 1. Evolution of the DDS tape format.

Fig. 2. The HP C1533A DDS-2 tape drive has a native-mode data transfer rate of 510 kbytes/s, 40% faster than other designs. It records on high-capacity 4-Gbyte DDS-2 cartridges or on earlier DDS media.

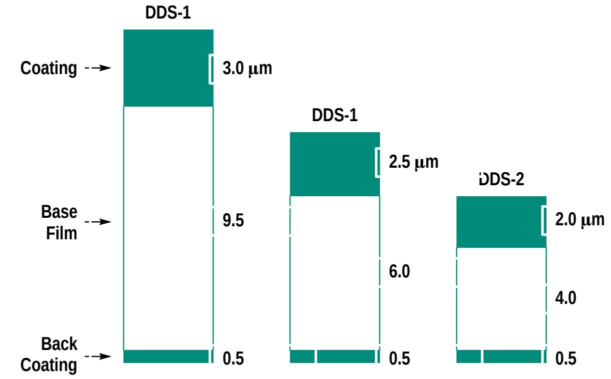
achieved using the DCLZ data compression standard) and has a data transfer rate more than 2.5 times that previously available on DDS tape drives. The HP C1533A not only reads and writes tapes based on the DDS-2 format standard, but is also able to read and write tapes based on the original DDS format standard to provide compatibility with the large installed base of DDS tape drives already in existence.

The DDS-2 format standard calls for the data to be written on tracks that are nominally 9.1 µm wide as opposed to the previous DDS format standard, which used a 13.6-µm track width. The DDS-2 format standard also makes use of tapes that are 120 m long rather than the previous 90-m and 60-m tapes. These changes, defined by the DDS-2 format standard, along with a data transfer rate increase to 510 kbytes/s from 183 kbytes/s (the data transfer rate seen by the user is effectively doubled to over 1 Mbyte/s by the use of data compression) required numerous technical developments.

Media

The media used for DDS-2 are an enhancement of the existing DDS 60-m and 90-m media. Physically, all three cartridges look similar; they use the same cartridge shell and are all designed to meet the same environmental and data reliability specifications. What differentiates them, apart from the packaging, is the length of tape in the cartridge shell and the signal characteristics or recording properties of the media. The tape drive differentiates between the cartridge types by the use of recognition holes in the cartridge shell.

The longer tape length is achieved by reducing the total thickness of the tape so that the tape pack volume is the same for 120 m as for 60 m and 90 m. Fig. 3 shows the relative thicknesses of the three DDS media types and a simplified view of the construction of the tape. To provide optimal head-to-tape contact when switching between the different tape thicknesses (the drive needs to read and write DDS-1 tapes as well as DDS-2 tapes) the stiffness of the tapes needs to be matched as closely as possible. However, because the relationship between tape thickness and stiffness is a cubed law, use of the same base film material is not possible. A new base film material, polyamide or PA, has been developed. It has high stiffness, and by careful design of the heads and drum, the ability to switch from one tape type to another can be optimized.



Length	60 m	90 m	120 m
Thickness	13 µm	9 µm	6.5 µm
Width	3.81 mm	3.81 mm	3.81 mm
Base Film	PET	PET or PEN	PA
Coating	MP	MP	MP+

Fig. 3. DDS tape types. MP = metal particle. PET = polyethylene terathalate. PEN = polyethylene naphtalate. PA = polyamide.

The 120-m tape, at 6.5-µm total thickness, offers the thinnest media currently in use in the data recording industry. This has necessitated improvements in the accuracy of the tape guidance system within the tape mechanism and in the tape motion tension control servo of the mechanism to prevent media damage.

The use of thinner tracks reduces the overall system signal-to-noise ratio and the tape was called upon to make a contribution to reducing the deficit. An increase of +3 dB over the existing DDS media was needed. Existing 60-m and 90-m media use metal particle (MP) coatings. To meet the additional signal requirements an enhanced MP tape has been developed and has been designated MP+. By using a combination of magnetic particle size reduction, increased coercivity, increased remanence, and reduced surface roughness of the media, the additional signal requirements were achieved.

The higher head-to-tape speed of the HP C1533A DDS-2 tape drive compared to previous DDS drives places additional constraints on the media. Without efficient lubrication the surface of the media is damaged reducing the life of the media. There is also the possibility of the heads becoming clogged with debris from the media. To reduce this effect the lubrication has been modified for DDS-2.

Heads

The changes needed for DDS-2 also called for modifications to the heads. A DDS tape drive has four heads mounted on a rotating drum. Two heads are used for writing and two for reading. The tape is wrapped around the drum over an angle that is nominally 90 degrees (see Fig. 4) so that only one head is in contact with the tape at any given time.

During a write, the first write head (designated the A write head) contacts the tape and data is written with an azimuth angle of +20 degrees. The first read head (designated the A read head) contacts the tape next to verify the previously

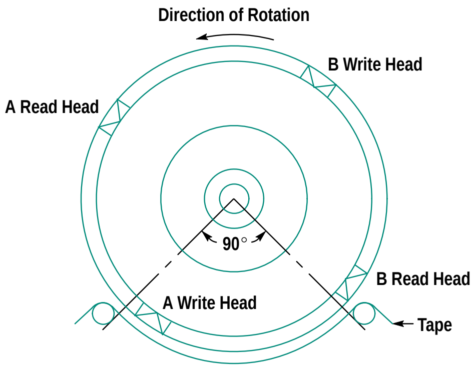


Fig. 4. The DDS tape drive has four heads mounted on a rotating drum. Only one head at a time is in contact with the media.

written data. The second write head (designated the B write head) contacts the tape next to write data with an azimuth angle of −20 degrees. The second read head (designated the B read head) contacts the tape next to verify the data previously written by the B write head. During this process the tape is moved forward to produce data on the tape as shown in Fig. 5.

To accommodate the higher coercivity of the DDS-2 tape media the write head was changed from a Sendust†-based head to a metal-in-gap (MIG) style ferrite-based head (see Fig. 6). The MIG head ensures that the magnetic coating on the tape is fully saturated during the write process. Sendust is still used in the head, but only for the gap metal and not as the bulk material.

The read head required several changes to meet the needs of the HP C1533A. The first was to change from a Sendust-based head to a ferrite-based head. This was necessary to maintain the nominal head life specification of 6000 hours. Since the HP C1533A has a data transfer rate that is 2.87 times the previous generation of DDS tape drives, the head is in contact with the tape media 2.87 times more during that 6000 hours. The ferrite-based head is harder than the Sendust-based head and meets the life requirements.

The second change to the read head was to the width. The nominal width of the read head for the DDS format standard was 20.4 μm. A read head of this width on a 9.1-μm wide DDS-2 track would allow too much adjacent track noise to be picked up, thereby reducing the signal-to-noise ratio below an acceptable level. At first glance it would seem that a read head width of 9.1 μm (equal to the nominal written track width) would be optimum (maximum on-track signal pickup with minimal adjacent-track noise pickup). This would be valid if the tracks were all perfectly straight. However, the DDS-2 standard calls for the tracks to be straight within ±2.5 μm over the length of the track (called linearity) to allow for mechanical tolerances in the tape drive. While a 9.1-μm-wide read head would provide an excellent signal-to-noise ratio on a perfectly straight DDS-2 written track, it would not be able to read a worst-case DDS-2 written track that was written by a different drive (see Fig. 7). In this case, the read head would have a large adjacent-track noise pickup with a relatively small on-track signal pickup. Since the ability to interchange tapes between tape drives was an important consideration in the design of the HP C1533A tape

† Sendust is an alloy of 85% Fe, 6% Al, and 9% Si. It was developed at the University of Sendai, Japan.

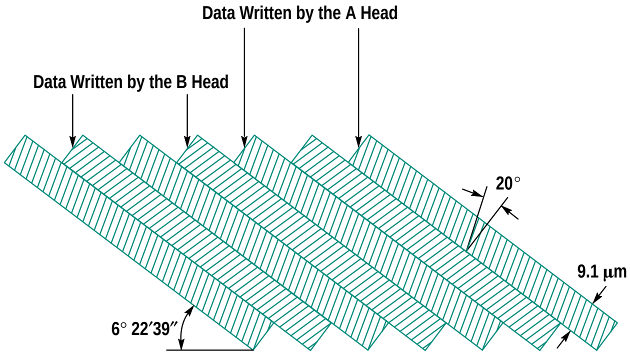


Fig. 5. DDS-2 track format on tape.

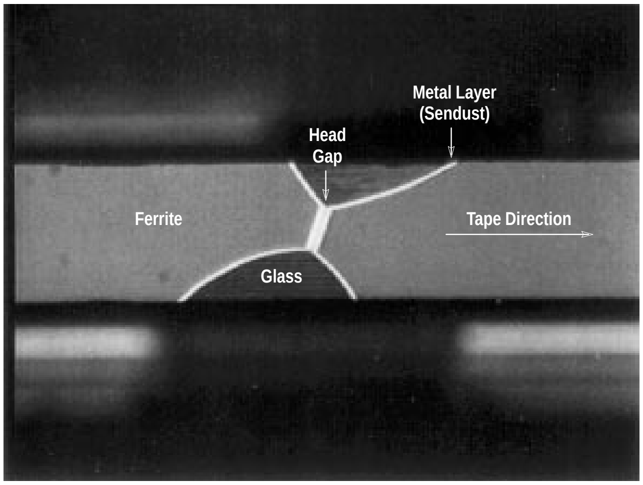


Fig. 6. The HP C1533A tape drive has metal-in-gap ferrite-based write heads.

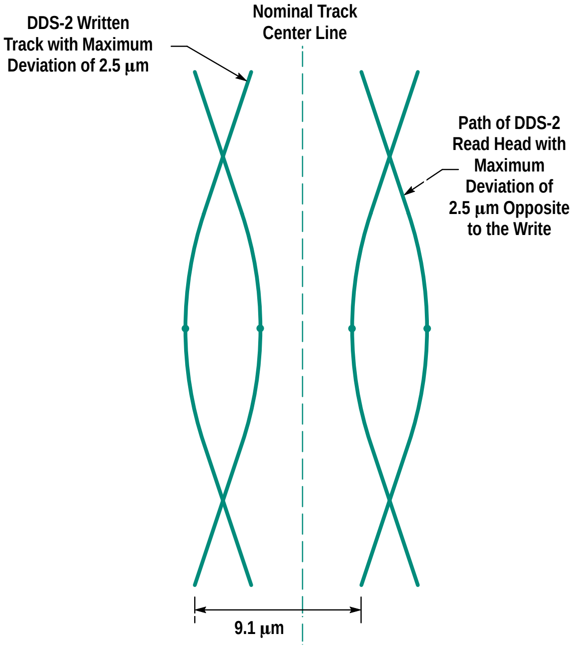

DDS-2 Written Track with Maximum Deviation of 2.5 μm
Nominal Track Center Line
Path of DDS-2 Read Head with Maximum Deviation of 2.5 μm Opposite to the Write
9.1 μm

Fig. 7. Effects of tape and drive nonlinearity when a track written by one drive is read by a different drive.

drive, a balance needed to be struck to achieve the optimum read head width. A 12-μm read head width was specified through the use of computer modeling of the effect of adjacent-track noise on signal-to-noise ratio. Experiments verified the performance of the 12-μm head.

Drum Design

The next major challenge came in the drum design. The increase in the data transfer rate to 510 kbytes/s from 183 kbytes/s required an increase in the drum rotation speed, to 5737 r/min from 2000 r/min. The major issues to contend with were drum bearing life, acoustic noise at the higher rotation speed, and excessive air between the drum and the tape. Excessive air would force the tape too far away from the drum, resulting in reduced signal levels because of loss of contact between the head and the tape.

A great deal of work has already gone into bearing life and acoustic noise in high-rotation-rate spindle motors for disk drives. The key to bearing life is the proper choice of lubricants. By using the same high-performance lubricants that are found in disk drive spindle motors, we were quickly able to meet the bearing life requirements of the HP C1533A tape drive.

The acoustic noise generated by the drum is largely a function of the control system. The control algorithm used in the HP C1533A tape drive reduces the high-frequency content of the control signals so that the acoustic noise of the HP C1533A is comparable to previous-generation DDS tape drives.

The problem of excess air between the drum and the tape required the development of techniques to bleed away the excess air to ensure that proper head-to-tape contact was maintained. Several techniques were prototyped and carefully measured by HP Laboratories for their impact on tape deformation. Among the techniques prototyped was a "windowless" drum, in which there is a gap between the lower, stationary section of the drum and the upper, rotating section of the drum. The gap provides a path for the air to bleed away and eliminates the need for a "window" around the head. A second technique prototyped was a standard window style drum with a small chamfer along the bottom edge of the upper rotating section of the drum to provide the necessary air bleed (see Fig. 8).

In the end, the window style drum using a chamfer on the lower edge of the rotating section of the drum was found to be the best solution, ensuring that the tape was not damaged while providing an easily manufacturable solution to the problem of excess air between the drum and the tape.

Linearity

As previously mentioned, the DDS-2 specification calls for a maximum deviation from a straight line of ±2.5 μm for a written track. This specification is referred to as linearity. The linearity of the previous generation of DDS tape drives was measured and found to have a mean value of 3.7 μm and a standard deviation of 0.74 μm. To meet the DDS-2 specifications, an intensive research activity was undertaken at HP Laboratories. That research determined that the linearity measurement and adjustment process would have to be changed to meet the DDS-2 specifications consistently. Previously, linearity was measured by writing a tape, physically cutting out a section of that tape, developing the tape using ferrofluids to be able to see the written tracks, and then measuring the tracks under a microscope. This technique suffers from two problems. First, the measurement error associated with the technique was found to be up to 2 μm. Second, the technique did not allow the linearity to be measured in real time while adjustments to the guides were being made on the production line.

The problem of measurement error was tackled by developing an automated optical measurement system. Using optical pattern recognition software, the system automatically finds special written patterns on the tape. A precision coordinate measurement system measures the track position relative to the edge of the tape. The system is calibrated with a chrome optical standard and a measurement accuracy of ±0.15 μm is achieved.

This optical measurement system is used to measure the absolute linearity of tapes. These tapes are then used as a reference for a real-time measurement system in production. Special patterns written on the tape are used by the tape drive under test to measure its own deviation along the track relative to the tape in the drive. By subtracting out the measured linearity deviation of the tape in software, an accurate real-time measurement of the linearity of the drive under test is achieved. It is then possible for a production operator to adjust the tape guides for minimum linearity deviation as part of the standard production process. The use of these linearity measurement and adjustment methods has allowed HP to reduce the mean linearity deviation in production to 1.4 μm with a standard deviation of 0.33 μm, well within the DDS-2 specifications.

Performance

The network server market that the HP C1533A tape drive is designed to serve requires high performance as well as high capacity. The data transfer rate of 510 kbytes/s is an important factor in the performance of the HP C1533A tape drive since it defines the maximum rate at which data (after compression) can be written to or read from the tape. The actual

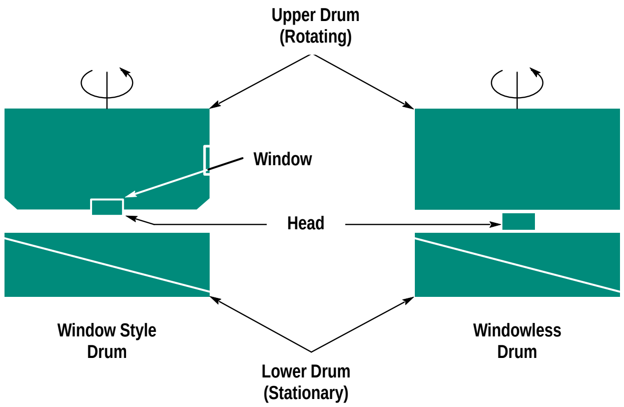


Fig. 8. Two styles of drums with gaps to bleed off excess air between the drum and the tape. In each case, four heads are spaced evenly around the drum, but only the one nearest the observer is shown in this drawing. The diagonal line on the lower drum is the path of the lower edge of the tape in this helical scan system.

performance the user will see in a system is a function of at least thirteen factors. Some of these factors are a function of the tape drive, but many are dictated by the system using the tape drive. The major performance factors, along with the corresponding controlling functions, are listed below.

Performance Factor	**Controlling Function**
Data transfer rate	Tape drive if desired data transfer rate is greater than maximum tape drive transfer rate Computer system if desired data transfer rate is less than maximum tape drive transfer rate
Maximum data compression ratio that maintains maximum tape drive transfer rate	Tape drive
Data compression ratio	Data
Main buffer size	Tape drive
SCSI transfer rate	Limited to the maximum SCSI transfer rate that both the tape drive and the computer system can achieve
Data transfer size	Computer system

The architecture of the HP C1533A controller is outlined in Fig. 9.

An examination of the write process is instructive in understanding the potential performance limiters. The following are the major steps in the write process.

1. Computer system negotiates SCSI transfer rate with the tape drive.

2. Computer system establishes transfer size.

3. Computer system transfers data to the tape drive.

4. Tape drive compresses the data through the data compression processor and moves the data into the main buffer.

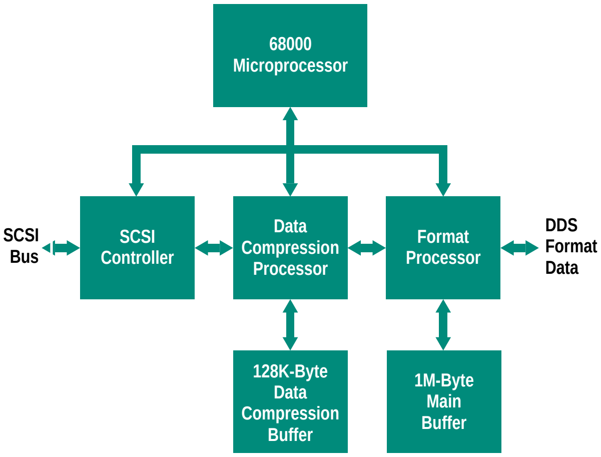


Fig. 9. Block diagram of the HP C1533A tape drive controller.

5. Format processor divides the data into DDS format standard groups and adds access and indexing data to enable high-speed search and retrieval and error correction fields to maintain data integrity on reads.

6. DDS format data is written onto the tape.

To achieve maximum performance, it is essential that DDS format data is available to write onto the tape at all times. If no data is available, the tape drive will have to stop writing data and wait until data is available before restarting the write process. The tape will also have to be repositioned before the next write can begin. The format processor must of course have the ability to take the data from the main buffer, convert the data into the DDS format and compute the error correction values in real time or the tape drive will never achieve its full performance.

The main buffer performs a speed matching function, giving the data compression processor the necessary freedom to output data at a varying rate while keeping the tape drive streaming. Modeling demonstrated that a buffer size of 1M bytes is sufficient to maintain the performance of the HP C1533A tape drive in most applications.

The maximum speed at which the data compression processor can take uncompressed input data and output it as compressed data is another factor in the performance picture. The more compressible the data, the faster the data compression processor needs to be to maintain an output rate that is fast enough to keep the tape drive streaming. An average compression ratio of 2 to 1 is the accepted industry norm for typical computer data. However, within a large backup, the compression ratio of individual files will vary tremendously. By studying a large number of backups, we were able to establish that the data compression processor needs to be capable of about 4-to-1 compression at full output rate to maintain an overall 2-to-1 compression rate. We therefore designed the data compression processor for the HP C1533A to meet this requirement.

The last major piece in the performance picture has to do with the ability of the computer system to move data to the tape drive. To get the maximum performance from the tape drive, it is important that the computer system provide the data as fast as the tape drive needs it. The maximum rate at which the data can then be transferred to the tape drive is determined by the negotiated SCSI transfer rate and the transfer size that the computer system establishes. The SCSI transfer rate is the lower of the computer system and tape drive maximum rates. If the computer system's maximum SCSI transfer rate is less than the tape drive's maximum SCSI transfer rate, the SCSI transfer rate used will be that of the computer system. The slower the transfer rate, the less time there is to cover any system overhead. Additionally, if the computer system establishes a small transfer size, the data transfer will occur in small increments. Since each transfer has overhead associated with it, small transfer sizes will have relatively more overhead which will likely reduce the performance of the tape backup (see Fig. 10).

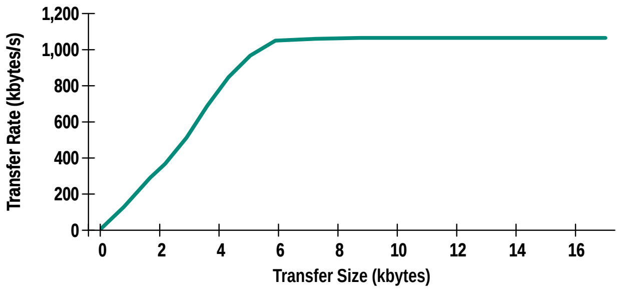


Fig. 10. Maximum data transfer rate of the HP C1533A tape drive at 2:1 data compression ratio (variable mode, writing from memory on an HP 9000 Model 720 computer).

Acknowledgments

I would like to acknowledge Hugh Rattray for mechanism and media development management, Peter Steven for tape path development, John Hardwick and Phil Connor for head and drum development, Ian Russell for media development, Simon Gittens for servo and data channel development management, Rob Morling for data channel development, Ben Willcocks for servo code development, Julian Potter for prototype and tool development, Tom Conway for servo IC development, Steve Krupa for controller development management, Tim Phipps, Andy Hana, Steve Langford, Pete Walsh, and Dave Dewar for controller IC development, Richard Bickers, Paul Bartlett, Simon Rae, and Jon Buckingham for controller firmware development, Pete Bramhall for technology and standardization management, Chris Williams and Simon Chandler for data channel modeling, Bryan Magain for interchange test development, Richard Vincent for integration management, Dave Tuckett, Simon Southwell, and Jon Rushton for integration engineering, Greg Trezise and John Rich for printed circuit board development, Gary Marriner for project coordination, Mike Padfield, Malcolm Grimwood, and Jonathan Lord for testing, Carl Tausig of HP Labs, Palo Alto for linearity measurement tools, Albert Jeans of HP Labs, Palo Alto for tape path modeling, and many others. I would like to thank Ian Russell, Peter Steven, and Richard Bickers for their help in preparing the material for this article. I would also like to thank Jo Dursley for her efforts in typing and editing the article.

References

1. M.J. Bianchi, et al, "Data Compression in a Half-Inch Reel-to-Reel Tape Drive," *Hewlett-Packard Journal,* Vol. 40, no. 3, June 1989, pp. 26-31. This paper discusses the HP-DC algorithm, a precursor of the DCLZ algorithm.

2. J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE TRansactions on Information Theory,* Vol. IT-23, no. 3, May 1977, pp. 337-343.

DDS-2 Tape Autoloader: High-Capacity Data Storage in a 5¼-Inch Form Factor

The autoloader holds six 4-gigabyte cartridges. With data compression, it can back up typically 48 Gbytes of data overnight or 8 Gbytes every day for six days, unattended.

by Steven A. Dimond

The trend from centralized computing data centers to PCs and client-server networks has led to increased local or semilocal data storage and backup. As discussed in the preceding article, DDS tape drives are designed to meet these requirements. However, as network server disk capacities increase above the capacity of a single DDS tape (8 gigabytes for DDS-2, compressed), or if manipulation of the backup tapes becomes a chore, then there is a requirement for a larger storage device.

The type of person who carries out the backup has also changed with these trends in computing. The centralized data center had trained, full-time operators, whereas the network administrator or workstation user may not be formally trained and will want to spend the minimum time and effort completing the backup.

These requirements for storage capacity and ease of use have led to a need for an automated, easy-to-use, large-capacity tape device.

One way to add significant capacity at modest cost is to use a changer mechanism (robot) to select a tape from a library of tapes and put it into the tape drive. The changer mechanism may only double or quadruple the cost of the tape drive unit, but the capacity can increase many times more than this. There are tape libraries that have from 10 to 120 tapes and one or two built-in tape drives. The access times to select a tape are acceptable for a backup or library type application.

Given the emerging network requirements, there is an even bigger need for a small device that fits the standard "5¼-inch" peripheral slots. These are approximately 146 mm wide by 83 mm high by 203 mm deep (5.75 in by 3.25 in by 8 in). This is enough volume to hold a smaller peripheral-size tape drive and a changer mechanism.

These smaller devices that perform unattended backup are typically called autoloaders. At HP's Computer Peripherals Bristol division, we investigated this growing need. This investigation led to the development of the HP C1553A DDS-2 digital audio tape autoloader, Fig. 1.

The HP C1553A autoloader incorporates the HP C1533A DDS-2 tape drive described in the article on page 6. It holds six DDS-2 cartridges, each having a native capacity of four gigabytes. With data compression, each cartridge can hold typically eight gigabytes, giving the autoloader the ability to back up typically 48 gigabytes without operator intervention.

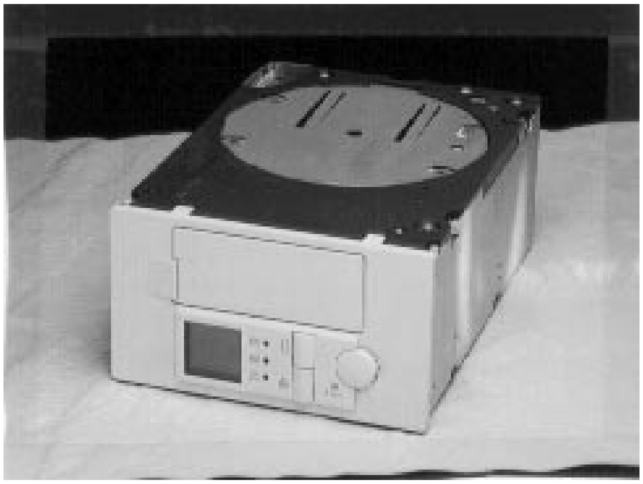


Fig. 1. The HP C1553A DDS-2 digital audio tape autoloader contains a DDS-2 tape drive and a cartridge changing mechanism that selects one of six cartridges from a magazine and loads it into the drive. Cartridges are changed automatically under software control.

Two standalone versions are available: The HP 6400 Model 48AL for HP 9000 workstations and the HP SureStoreTape 1200e for Novell Netware and Windows NT systems.

Design Objectives

The basic definition for the HP C1553A autoloader was very simple. It was to be a DDS tape autoloader, fit into a standard 5¼-inch peripheral enclosure, use a standard HP DDS drive with minimum (or no) modifications, use the drive's SCSI II interface, hold as many tapes as possible, and be reliable and ergonomic.

During the investigation phase for this product there were prototypes of similar products available. To keep the investment low we considered procuring one of these designs. However they fell short of some of the requirements, so the decision was made to produce our own design.

Given the small size of the product and the desire to accommodate as many tapes as possible, the interior space was at a premium. Despite frequent questioning by the engineers, the outside dimensions could not be increased if we were to be sure of satisfying the maximum number of customers. One possibility was to have a "power bulge" on the rear of the unit, but this was rejected because it might obstruct some customers' installations.

It was decided that the new HP C1533A DDS-2 tape drive was to be fitted inside the autoloader. For reasons of manufacturing simplicity and cost, the drive had to be used with minimum modification. We were also aware that the autoloader would be a platform for future DDS drives, so easy integration was important for future generations as well. The price of the autoloader could perhaps be only double that of the DDS-2 drive for several times the capacity.

The SCSI II interface of the built-in drive can be used to pass on control commands to the changer mechanism. Thus the customer need only use one SCSI bus ID where some libraries require two.

The more tapes the device can hold, the more attractive a product it will be. Competing autoloaders with four cartridges were known to be under development, so our goal was to match this number or exceed it.

We saw the magazine holding the tapes as a simple storage solution, that is, inexpensive and capable of being stored like a video tape. This allows the user to treat a magazine as a big single backup tape, rather than having to manipulate a lot of single tapes.

For reliability and ease of use, use models and metrics were developed. In the early stages of development, user tests were conducted on possible design concepts.

Physical Architecture

The physical architecture is dominated by the lack of space. Fitting the 3½-inch C1533A tape drive into the volume allowed for the autoloader accounts for much of the difficulty. The drive is placed at the rear of the autoloader volume because it has its interface connections at the rear and the tape loading at the front. There is just enough space for a tape in front of the drive, but unfortunately no front-to-rear room for a mechanism. However, removing the front panel (bezel) of the drive, which is not required, created a vital few extra millimeters.

The construction of the front of the DDS-2 drive is such that the tape can overlap the drive when it is ejected, that is, the tape can extend into the drive about 11 mm. This means that a tape must be moved vertically above the drive to remove it; it cannot move down through the drive mechanism. This allows just over 10 mm from the front of any tape to the autoloader front panel. This configuration also places the drive at the rear bottom of the autoloader so that access to other tapes is from above the drive.

This configuration does have some benefits. The drive connectors and option switches on the rear and bottom are directly accessible at the exterior of the autoloader. Thermally this is also the best arrangement because the drive base is exposed to the exterior air. The drive base is an important heat dissipation area just below the main controller printed circuit assembly. Finally, for integration and repair we were able to design the autoloader to accept the drive with a simple bracket arrangement and a single connecting cable.

The changer mechanism, controller printed circuit assembly, magazine, and door and front panel parts have to fit in the rest of the available space. At this point we checked again whether we could exceed the form factor, and were again asked to look inward rather than outward.

Autoloader Control Electronics

The autoloader control electronics were designed with the aim of linking the mechanism and firmware elements with low-risk, proven technology at low cost. The main controller printed circuit assembly is a through-hole design with a shape to suit the space available. The space envelope for the printed circuit assembly was derived directly from the mechanical CAD model because it was so tight. Control of the mechanism is managed by a Hitachi H8/325 microcontroller with on-board one-time-programmable (OTP) and RAM memories, nearly 90% of the pins being available for I/O. Logic-level pulse width modulation signals generated by the microcontroller control the dc motors and their integral gearboxes. Two-level motor current sensing is used to detect mechanical jams or excessive motor loading. The four motors and the picker solenoid are powered from the 12V supply available in 5¼-inch peripheral slots.

The state of the mechanism is determined by optical means. For each motion, a mechanical part has a rib that is made to pass through slotted optical switches. The rib has slots at datum positions in the motion that are detected by the optoswitch. The width of each slot is calculated to reflect the mechanical tolerance of the particular motion so that the firmware can guarantee a particular mechanical position as long as the optical switch is open. An important philosophy here was to position each slotted rib (comb) at the “point of action” (the farthest point from the motor drive) so that backlash does not compromise the accuracy of the position detection.

By using relatively large (and inexpensive) motor drive ICs operating well within their thermal specifications and mounting the printed circuit assembly vertically for optimum convection cooling, thermal problems were avoided. For the picking action, an oversized solenoid is operated conservatively so that it does not get too hot. The solenoid delivers a relatively large force for the picker fingers but for short durations (less than two seconds).

The front-panel printed circuit assembly is connected to the main controller printed circuit assembly by a flexible circuit. Mounted on the front-panel printed circuit assembly are the three front-panel switches, the door open optoswitch, three LEDs, and the LCD. The LCD is a custom design procured with a standard driver IC on its flexible circuit, which is soldered to the front-panel printed circuit assembly. All of these components were physically modeled in the HP ME30 CAD system to integrate the electrical and mechanical designs.

An important design goal was ease of access to the printed circuit assembly since the firmware is stored in an OTP device that must be replaced if firmware upgrades are necessary. To this end the board is fully connectorized and fixed by a single screw. No adjustments or calibrations to the printed circuit assembly are needed, so complete printed circuit assemblies can be swapped in if necessary. The layout of the printed circuit board is heavily influenced by the flexible circuit designs and a lot of time in the early stages of the project was spent on the topography of the flexible circuits, their routing, and the effect of the positions and direction of entry on the printed circuit assembly layout. In particular, to keep the cost of the flexible circuits as low as possible they were all designed as single-sided circuits. This dictated pin ordering on the printed circuit assembly, which also needed to have minimum layers to keep it low in cost. The final printed circuit assembly has just four layers including power and ground planes covering 90% of the board area. There are four flexible circuits connecting the motions and the front-panel printed circuit assembly. Their physical layouts were modeled on the HP ME30 system and also using paper mock-ups to check for control of their positions when moving, as well as track layout.

Greg K. Trezise
Development Engineer
Computer Peripherals Bristol

Ideas were tried in outline form to maximize the number of cartridges with the simplest mechanism. It quickly became clear that the size of the DDS cartridge imposed limitations that dictated the capacity and mechanism design. For example, even with minimal thicknesses in between, there is only room for three cartridges above the drive. To achieve a capacity of four with a simpler mechanism the cartridges would have to move up and down in front of the drive, but this violated the form factor.

Using only mockups and cartridges, we arrived at the possibility of rotating the column of three tapes above the drive. This accommodated six cartridges and would make us the market leader in capacity. There is just space, but how could it be implemented? The vertical height of the components had to be pared down to the minimum. For example, there is 1.5 mm between cartridges, 1 mm for a shelf in the magazine, and 0.5 mm clearance (all dimensions nominal). The volume swept by the six tapes rotating is very large, taking up about half the height of the autoloader and most of the width, leaving only about 8 mm on each side. We decided that the changer mechanism would have to occupy some of the swept volume when the tapes were not rotating.

We used the HP ME30 mechanical CAD software to plan our air space and implement the concept. At this point three mechanical engineers developed the design, sharing the same file system and conventions so that we could easily load the subassemblies of our colleagues to check for clashes and interfaces.

Autoloader Design

The autoloader can be broken into several physical subassemblies, as follows:

- Magazine
- Mechanism motions—X, Y, Z, R
- Front panel and user interface
- Control electronics and motors
- Mechanism firmware
- Drive firmware, SCSI interface, and link to the mechanism.

Magazine. The magazine is made from polycarbonate plastic moldings (see Fig. 2). It is simply a container for the six cartridges, three on each end. The main molding is a box structure into which some shelves are fixed. The thin (1 mm) wall sections and usability features made this an extremely challenging design. Required features were retention of the cartridges and acceptance of the correct orientation only. In other words, you can only put the cartridges into the magazine in the correct way, and then they stay in, even when you shake them by hand or apply shock and vibration to the unit. The cartridges were not designed with these features for autoloader use, so we had to be creative. In addition there are regulatory requirements (UL94-V2 flame resistance) and a need for reasonable robustness, that is, the cartridges should not be damaged if the magazine is dropped on the floor. The magazine also has areas for a label and indications to help ease of use. There is a gear form along one side for automatic loading. The magazine is shaped so that it can only be inserted the correct way into the autoloader.

A semitransparent polypropylene library case is supplied with the magazine. This allows the user to store the magazine neatly with some protection. It is similar to the library cases for commercial videotapes.

X Motion. Looking from the left side of the unit, the X motion moves horizontally (front to back) in the autoloader (see Fig. 3). This motion moves the cartridge horizontally. The

(continued on page 16)

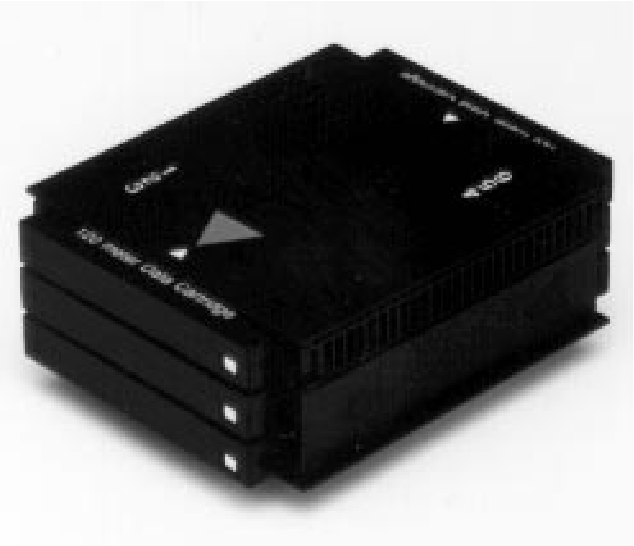


Fig. 2. The six-cartridge magazine is designed to be handled like one large tape cartridge.

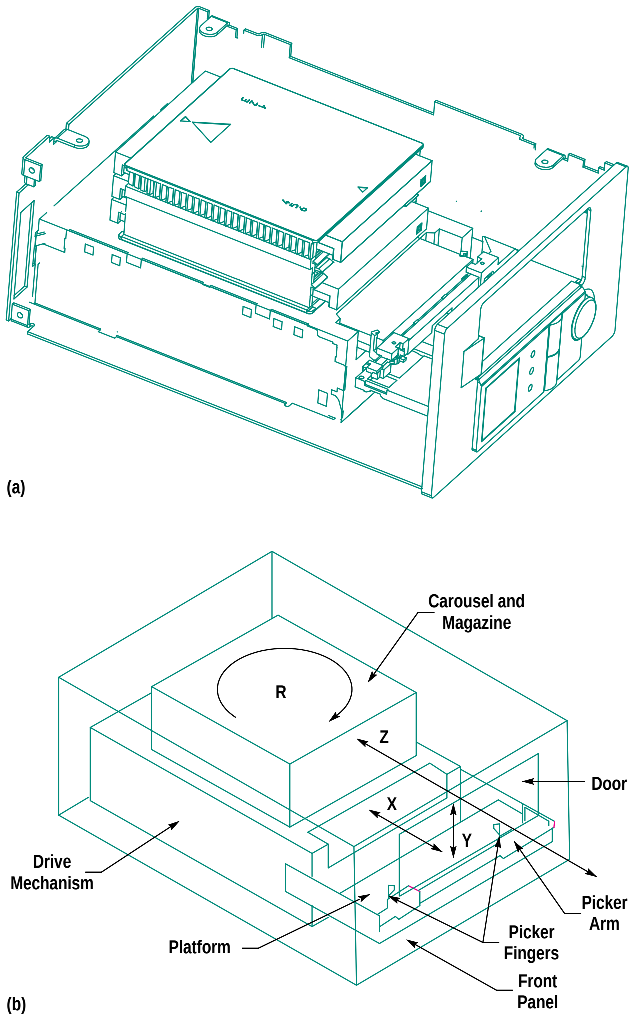


Fig. 3. (a) Interior layout of the HP C1553A autoloader. (b) Motions of the cartridge changing mechanism.

Autoloader Firmware Design

When the HP C1553A DDS-2 autoloader was designed, one of the primary goals was to add the autoloader mechanism to a standard DDS drive with no hardware modifications to the drive and minimum firmware modifications. The resulting architecture allows the autoloader mechanism to be independent of drive hardware and to be used with different drive products.

Originally, HP's DDS tape drive product line was not designed to be used with an autoloader. However, the requirement for a low-cost autoloader mechanism was realized and steps were taken to add an autoloader to the current product line. This required three basic steps:

- Design of the drive/autoloader interface, both hardware and software, requiring no hardware changes to the drive
- Addition of the loader command set to the drive firmware
- Design of the autoloader electronics and firmware to enable the required communications with the drive.

Drive/Autoloader Interface

Since the drive had not been designed with the intention of interfacing to an autoloader, there was very little hardware available to create an interface to the autoloader mechanism. It was decided that a port that was used for debugging purposes in manufacturing test could be used to communicate with the autoloader, since it had no use outside the manufacturing line.

The port has four data lines and a single address line along with the required handshake lines for the drive's 68000 processor. This allows a total of four registers, two write-only and two read-only. It was decided that these should be 8-bit registers accessed by two successive 4-bit operations. The four registers are the drive status register, the autoloader status register, the drive autoloader command register, and the autoloader drive report register.

Two registers are used as a command report mechanism to allow the drive to send commands to the autoloader. This is the basis for the control of the autoloader. When the drive receives an SCSI command that requires autoloader operation, it writes the appropriate single-byte command code into the drive autoloader command register as two four-bit writes. When the autoloader has completed the operation it places the single-byte report in the autoloader drive report register and asserts an interrupt signal to the drive to indicate that the register should be read.

Commands that require parameters are preceded by a push parameter command. This is a single-byte command that has the top bit set. All other commands have the top bit clear. This allows the remaining seven bits to be pushed onto a parameter stack. Successive push parameter commands allow more than seven bits to be pushed.

The autoloader status and drive status registers are used for handling the front panel. Since the autoloader's front panel is completely different from the standalone drive's, it is accessed via the autoloader processor. However, since the drive processor has the responsibility for telling the autoloader what to do, the front-panel switches are read and interpreted by the drive via the autoloader status register. This allows maximum flexibility of operation and configuration.

To maximize the usability of the autoloader, it was decided to use a character-based LCD display to give messages to the operator. Since most of the status information comes from the drive, the drive status register is used to pass status codes to the autoloader to display status messages on the display. The text for the messages is stored in the autoloader processor ROM. While it would have been more flexible to store the messages in the drive, there was insufficient space.

Drive Firmware Architecture

The changes required to the drive firmware to implement the drive/autoloader interface relate to two distinct areas. The architecture of the firmware for the autoloader is shown in Fig. 1.

First, the normal front-panel handling task within the firmware is replaced by a new version, which communicates drive status to the autoloader. This receives status information from both the SCSI task and the drive task on the drive. This status information is passed over the drive/autoloader interface rather than being displayed on the drive front panel. The SCSI task is changed to read the buttons on the new front panel as well as the eject button on the drive. The drive eject button is left active so that a tape can still be recovered from a drive in an autoloader even if the autoloader hardware is not working.

Secondly, the SCSI task required the addition of the functionality to handle the SCSI medium changer command set. This involved adding new functionality to the task to interpret a new class of commands and pass them on to the autoloader

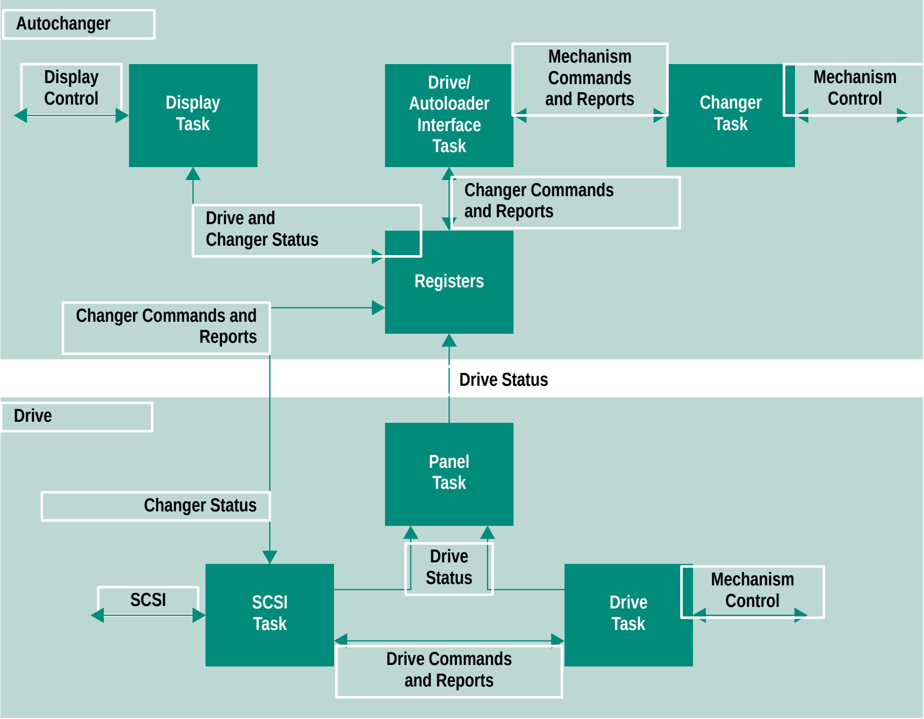


Fig. 1. Autoloader firmware architecture.

mechanism. Parsing the commands on the drive allows the drive to remain in control of the whole autoloader and minimizes the risk of conflicting commands going to the drive and autoloader. It also avoids duplicating the software to parse SCSI commands for both the drive and the autoloader.

Autoloader Firmware Architecture

The autoloader firmware consists of three distinct functions. These are communicating over the interface to the drive, handling the display, and controlling the autoloader mechanism.

These three functions are implemented in three separate tasks running in a round-robin fashion with a 1-ms time slice for each function. This made development of the software easier and allowed the separate functions to be implemented with minimal risk of interference with one another.

To save hardware costs, the drive/autoloader interface registers are not implemented as hardware latches. Instead, the I/O lines from the drive are wired directly into the ports of the H8/325 microcontroller used to control the autoloader mechanism. The H8 has a set of internal memory locations that mirror the imaginary hardware registers. When the select line from the 68000 is asserted, indicating an access to the drive/autoloader interface registers, this causes an interrupt to the H8. The H8 reads its I/O lines and handshakes the data to or from the 68000. This gives the appearance to the 68000 of slow hardware latches. The tasks running on the H8 merely need to access the internal memory locations as if they were the registers.

The drive status register is treated slightly differently from the other registers in the drive/autoloader interface. Because the drive can send repeated status values to this register faster than the display task on the H8 can read them, the values are queued within the H8 to be read in sequence. This ensures that an important status code is not lost behind a less important one. In addition, certain status codes cause flags to be set within the H8 that determine whether the drive is in a certain state. This allows tracking of the state of the drive.

Mark Simms
Development Engineer
Computer Peripherals Bristol

cartridge is gripped by metal fingers (mounted on a picker arm) on an edge near where a human grips the cartridge. The fingers are sprung shut, gripping a cartridge in case of a power failure, and are opened by a solenoid. The fingers are mounted on an arm, which pushes and pulls the cartridge. The arm can pull a cartridge out of the magazine and push it into the drive. The picker arm is moved by a belt, which is driven by a dc gear motor.

The cartridge is not designed for manipulation by a mechanism, so the choice of features that were suitable for gripping and alignment was limited. The obvious edges were not specified in the cartridge standard, and it took two years to have some of these features added to the standard.

Y Motion. The Y motion moves the cartridge and picker arm up and down. The picker arm is mounted on a platform that is lifted or lowered by two cams. The picker arm runs on a shaft, allowing the X motion. The shaft and the other X-motion parts including the gear motor and belt are all mounted on the platform. Connections are made to these parts by a flexible circuit. The platform has three pins, one on the left and two on the right, which project out the sides (looking from the front of the unit) into cams, one on each side of the unit. The pins run in tracks that resemble escalator shapes, that is, they have 50-degree slopes with horizontal portions. The pins can only move vertically because they also run in slots in metal plates. The cams with the shaped slots move backwards and forwards and drive the pins and the platform up and down (see Fig. 4). Both cams are driven by one dc gear motor. The one on the right has a molded rack and is driven directly by a gear on the gear motor. The left cam is connected by a lever across the bottom front of the unit to the right cam, and is driven by the same Y gear motor. This cam arrangement tolerates inaccurate positioning of the Y gear motor and cams. The height of the cam components themselves determines the height accuracy of the platform, which early calculations showed is adequate. The pins can be anywhere on the horizontal portions of the cam tracks, which are about 5-mm long. The flat plate arrangement of the cams and tracks fits neatly into the unit on either side of the platform. The left cam extends into the magazine rotation area making extra use of the space when the magazine is not rotating.

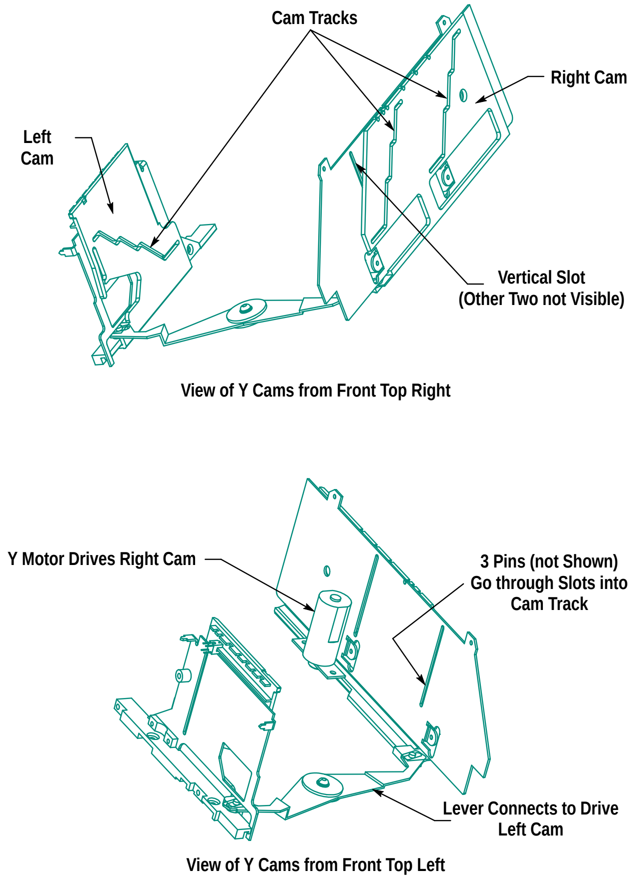

Cam Tracks
Right Cam
Left Cam
Vertical Slot (Other Two not Visible)
View of Y Cams from Front Top Right
Y Motor Drives Right Cam
3 Pins (not Shown) Go through Slots into Cam Track
Lever Connects to Drive Left Cam
View of Y Cams from Front Top Left

Fig. 4. Y-motion cam design. (a) View of the Y cams from the front top right. (b) View of the Y cams from the front top left.

R Motion. The R motion is the rotation of the magazine. This is achieved by a large disk in the top of the unit. The magazine sits on top of the drive. The usual drive lid (top) is replaced with one that has the front edge cut away to allow the cartridge to be lifted straight up with the 11-mm overlap. The rotating disk in the top of the unit has two moldings attached to it that hang down on either side of the magazine, allowing it to be located and turned around. Originally the rotating disk in the top of the unit was going to be inside the unit. However, because of the extreme vertical space problems, the disk is actually part of the exterior surface of the unit. The disk rotates 180 degrees and is driven by a dc gear motor through a clutch. The clutch allows the disk to be driven into an end stop for accuracy while preventing damage to the gear motor, which was found in early prototypes. The clutch is a custom design and drives a gear form on the disk.

Z Motion. The Z motion is the movement of the magazine in and out of the autoloader. The magazine has a large rack (gear form) on one side. This engages with a gear in the unit, which is driven, through a custom clutch, by the Z dc gear motor. A microswitch (Z switch) activated by a rocker arm indicates when a magazine has been inserted by the user. The insert and eject mechanism (Z motion) is deliberately designed to mimic the familiar home video recorder type of action. User tests showed that this action was familiar and intuitive. The action of the user is to push the magazine into the autoloader through the door, which is sprung shut. The Z switch detects the magazine and the Z gear motor starts. When the magazine is pushed a little farther the gear engages and pulls the magazine from the user. On entry the magazine compresses a spring-loaded pusher arm, which is used on ejection. The Z switch also detects when the magazine has reached the fully home position, that is, fully into the unit. On ejection the Z gear motor pushes the magazine out through the opened door. As the gear disengages, the sprung pusher completes the ejection. The magazine is caught by a small sprung plastic part to ensure a consistent eject distance. The distance is over 22 mm, which allows handicapped users to grip and remove the magazine.

The lid assembly, which contains the R and Z motions, was one of the later subassemblies designed and proved very difficult to finalize. The physical space restrictions and the desire to get the right feel for the user meant that several iterations of design had to be prototyped and tested.

Front Panel. The front panel assembly provides the controls (three buttons), displays, and door (see Fig. 5). A printed circuit board mounted behind the front-panel plastic molding has the display and switch components on it. Once again space was at premium and the whole assembly had to be thin to miss other mechanism parts. The layout was determined by the user tests and the position of the door. The door is central in the upper portion. The entry has a keying feature so that the magazine cannot be inserted the wrong way. Combined with the magazine features, this means that the user is guided into correctly inserting the cartridges and prevented from making mistakes. This means that the correct orientation of the cartridge is ensured when it arrives inside the autoloader. This is not always the case in other autoloader systems, which have to check the orientation.

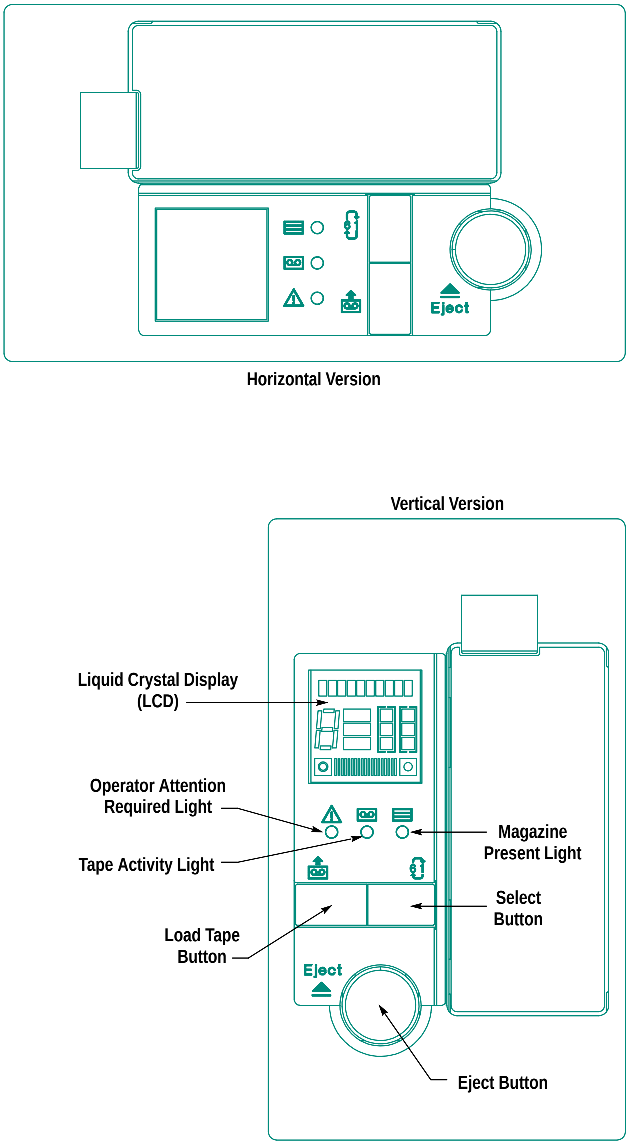


Fig. 5. Front panel layout.

The door opens inwards and is sprung shut, but for magazine ejection the door must be opened by the unit to allow the magazine to eject. The door is locked when a magazine is inside the unit to prevent tampering or injury to the user. It is unlocked when there is no magazine inside to allow the user to insert one. The door is pushed open and locked by the movement of the left-hand cam. The cam travels are extended to allow this operation. At the normal resting Y height (bottom of travel in front of the tape drive), the cams move farther to unlock the door, then farther to push open the door. This allows the extra function with no extra gear motor or mechanism. An optoswitch sensor detects whether the door is open; its main purpose is for safety, so that the unit can be stopped in case of a faulty lock. In this case, the unit will display an error message, "Close Door," and wait.

The appearance and position of the front-panel displays and controls were determined by the industrial designer within the mechanical design limitations and were heavily influenced by the user test results. Early on, some mockups were tested because there is potentially a lot of information that could be displayed to the user, such as error messages and many status messages, about 60 total. The most basic button and the one that all users require is the **Eject** button. This was made large and obvious, the shape making it look pushable so that minimum or no text is needed for users to choose it. The second button selects a starting tape, of the ones loaded, and the final button starts a manual backup. Alternatively, the unit can be controlled by SCSI commands from the host computer.

The displays are grouped into two types: those giving overall status that can be seen from a distance, such as idle, backing

Network Backup with the HP C1553A DDS Autoloader

The four main applications for the HP C1553A autoloader described in the accompanying article are:

- Single large backup
- Centralized network backup
- Fully automated backup
- Near-line data storage.

The backup of a large amount of data in a single session is a clear application for the autoloader. Today there are many servers with a disk capacity exceeding that of a single DDS-2 cartridge, which is typically 8 gigabytes with 2:1 data compression. The system administrator with a single tape drive must either manually insert new tapes into the drive when doing full backups, or must settle for incremental backups that only back up the data that has changed since the last full backup. These two options present some difficulties. Backups are typically carried out at night when server use is lower, so tape changing is inconvenient at best. A restore based on an incremental backup routine can be complicated since it will involve using several unrelated tapes. The autoloader, with its six cartridges, enables the system administrator to protect up to 48 gigabytes of data in one single unattended session.

Most of today's local area networks consist of several servers and many clients. Centralized network backup involves backing up all servers and clients across the network onto one high-capacity drive such as the autoloader. This is a cheaper alternative to having a separate tape drive on each server. Other benefits include having only one tape drive and one software package to administer and enhance security by having all removable data in one location, which can be physically secured. This is the same rationale that has been employed for centralizing network printing on one high-duty-cycle printer. For additional flexibility each of the the autoloader's six cartridges can be configured to hold data from a specific source. For example, each server can be backed up to its own cartridge and all clients to one of the other cartridges. Alternatively, all servers and clients on a segment of the network can be backed up to a specific cartridge. The exact choice will depend on restore and disaster recovery considerations.

Fully automated backup relieves the busy system administrator from another task previously taken for granted in the days of central mainframes: tape rotation. Methods such as "grandfather-father-son" and "tower of Hanoi" were developed to prevent overuse and wearout of media and to make available several differently aged versions of data when restoring. These methods involve backing up to a different cartridge every day. For the system administrator with a single tape drive this means manually changing the tape in the drive every day. If for some reason this does not happen most software packages will abort the backup, meaning that the system is unprotected. More significant, when the time comes for a restore, the system administrator must be on hand to retrieve the correct cartridge from secure storage and manually load it into the drive. These tasks can now be automated by making use of the autoloader's multiple-cartridge capacity. A simple routine with five data cartridges and a cleaning cartridge could be configured to perform a full backup every weekday to a new tape. This weekly cycle could be repeated over an extended period of time. A routine giving a longer file history would involve performing a full backup on the first day of the week, followed by daily incremental backups to the same tape. The magazine would then provide five weeks of protection for a server of up to five gigabytes. Using a tower of Hanoi rotation scheme, sixteen weeks of protection can be achieved with a single magazine. In all cases, the only manual intervention would be periodic magazine rotation to a fireproof safe or offsite location. Restores no longer need to involve the system administrator, either. With all of the cartridges available by random access in the magazine, the backup software can give users the ability to restore their own files with overall access rights controlled by the system administrator.

Prolonged operation of tape drives without any tape head cleaning can result in a media warning that causes the backup software to abort the backup. This need not be the case with the DDS-2 drive, which has a has a self-diagnostic capability that senses the write error rate. When this increases beyond a conservative threshold, the drive sends a message to the backup software, which can respond with the initiation of a head cleaning cycle using the cleaning cartridge included in the magazine. This will typically occur every twenty-five hours of use and ensures a long period of error-free operation without system administrator intervention.

up, or fault condition, and those giving further detailed information that is required when the user is near the unit, such as "Insert Mag" or "Clean Me." A custom LCD was selected for the detailed information, the custom icons and layout enabling a lot of information to be conveyed at a glance without long text messages. Ten characters of text are available to display messages (see Fig. 6). The printed icons beside the three buttons and three LEDs were also developed and tested for intuitive use by users and to avoid any text that would require localization. The exception to this is that the word "Eject" was required by the English speakers tested because the universal eject icon, seen on almost every VCR and cassette player, was not recognized! The word "Eject" is acceptable internationally.

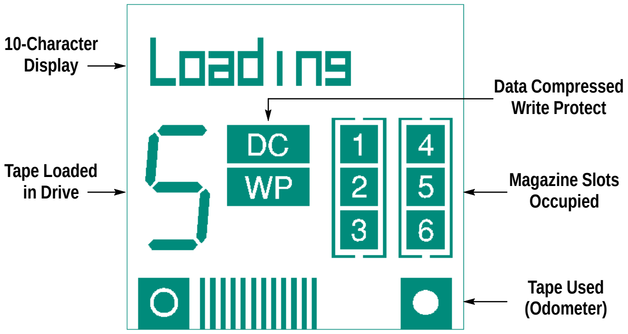


Fig. 6. Liquid crystal display layout.

Control Electronics. This development effort is described in "Autoloader Control Electronics" on page 13.

Mechanism and Drive Firmware. This development effort is described in "Autoloader Firmware Design" on page 15.

Mechanical Design Methods

ME30, HP's 3D computer-aided design tool, was used by the mechanical and manufacturing engineers. It was run on HP 9000 Series 700 workstations that had a common disk mounted with directories structured and named like the product subassemblies. The designers were responsible for maintaining the latest revision of their parts in these shared areas, and for providing a macro that would simply load all the subassembly parts. The motors, the flexible circuits, and the main components on the controller printed circuit assembly were all modeled. Thus the latest design was available to the designers for cross-checking, and to the procurement and manufacturing engineers for process planning. HP ME30 proved very valuable at speeding up the checking and success of prototype assembly builds. Better visualization allowed the designers to spot problems early, thereby avoiding embarrassing problems on prototype builds. On one occasion, we discovered that we were about to design a gear motor assembly through the center of the H8 microprocessor!

aaaxsNear-line data storage is an evolving application for multiple storage devices based on using intelligent data management software known as Hierarchical Storage Management software. The size of hard disk mass storage on servers is increasing all the time. Today's average network server has a disk capacity of 9 gigabytes and this is projected to rise to 40 gigabytes within the next five years. The system administrator faces a constant challenge to ensure economic and efficient use of disk space by users. The reason for not adding hard disk drive capacity at will is cost. Hard disks are a very expensive data storage medium, but are necessary to ensure fast access to data; access time is of the order of 10 milliseconds. Magnetic tape, by contrast, has a cost per megabyte of data one-hundredth that of hard disks, but the access time of a tape drive is on the order of 30 seconds. An analysis of the use pattern of files on a typical server shows that some current files are accessed frequently, but that the majority are older files used infrequently. Hierarchical Storage Management software tracks file access and automatically migrates infrequently used files to a lower-cost storage medium. Although the file is stored on another device, a phantom file of zero size is left in the original directory. As far as the user is concerned the file is still present. When access is required the Hierarchical Storage Management software retrieves it automatically. The small delay in retrieving the file from the slower device is acceptable on an occasional basis. All of the activity is transparent to the users, who effectively see a virtually unlimited amount of disk space. The data migration is triggered at typically 80% of disk capacity. The system administrator therefore never needs to be concerned about running out of disk space. The autoloader with its six cartridges can provide up to 48 gigabytes of near-line data storage at a fraction of the equivalent hard disk storage cost.

Performance Considerations

The autoloader's large capacity is well-matched by the DDS-2 drive's high data transfer rate. However, backup is a resource intensive operation that uses all of the components of the computer and network, not just the tape drive. Careful selection of all of the hardware is required to balance the throughput and ensure that there are no bottlenecks. The exact configuration will depend on the type of backup being performed.

For server-based backup, the limiting component is typically the hard disk drive. Backup involves randomly accessing all files on the hard disk. The disk can spend more time seeking data than actually reading it. The limitation can be reduced by "spanning" the data to be read over several disks. While one or more disks are seeking data one of the other disks can be reading data. This spanning can usually be implemented in the operating system but is more commonly implemented in hardware in the form of a redundant array of inexpensive disks (known as a RAID disk array). Here a dedicated controller card takes care of all the data input and output for all of the disks and frees the operating system from that overhead. In most cases an HP C1533A DDS-2 tape drive backing up data from a RAID disk array with five disks will approach its maximum native transfer rate of 510 kilobytes per second which is equivalent to 60 megabytes per minute with data compression.

For centralized network backup, the limiting component is typically the network. Today's most popular network topology is Ethernet, operating at a bandwidth of 10 megabits per second. During backup all the data must travel across the network, along with all of the disk access commands. This results in most cases in a transfer rate of about half of the maximum the DDS-2 drive can achieve. This can be reduced further if the amount of traffic on the network is high enough to result in packet collisions, so backups should be run at night when traffic is low. To achieve higher transfer rates over the network, faster topologies must be used. FDDI over fiber-optic cable and 100 Base-VG both operate at 100 megabits per second, ten times faster than Ethernet. Implementation of these technologies is becoming more widespread for reasons other than backup, such as multimedia and video. For existing installations it is not necessary to recable the entire network. The majority of data transferred will be from server to server, so these can be connected together with a dedicated high-speed backbone. This will result in a backup speed close to the DDS-2 drive's maximum.

When using intelligent data management software, an apparent increase in performance can be seen. This is because the infrequently accessed files, after having been backed up several times, are considered stable and are no longer backed up. A full backup therefore involves fewer files and can be completed in a shorter time. This performance gain is achieved with software without any changes to the hardware.

Michael G. Bertagne
Technical Marketing Engineer
Mass Storage Europe

Although valuable, the latest HP mechanical CAD software does not fully simulate all the mechanical motions. In this respect it is less mature than EE CAD. So the traditional design, build, test, redesign cycle is still the major way to achieve reliability improvements in the design. Anything that can be done to shorten this cycle, rapid prototyping for example, can save weeks or months of time to market. The 3D models allowed us to use the latest fast prototyping methods, including stereo lithography and CNC milling of parts.

Design Margin Analysis

Because of the desire to prove as much of the design as possible before commitment to tooling, we adopted an unusual approach to the mechanical tolerances. Typically designers will approach key areas and perform a tolerance analysis. They will add up the tolerances for the parts using data that they hope is representative of the production parts. Because of the lack of space, interdependency of all the subassemblies, and simply pressure of work on the designers, we decided to adopt another approach. A consultant from Cranfield Institute of Technology, one of the leading teaching and consultancy groups in manufacturing technology in the UK, was enlisted to help analyze the tolerances. We started from the outside in, deciding on the key areas of functionality, then building up spreadsheets of the systems with the tolerances. Our procurement engineers provided capability study data for the manufacturing processes of similar parts. Starting out with simple arithmetic, we built this analysis into a design margin index.[1] The managers and designers were then able to have a single number for the "goodness" of the design in each of six key areas. This was obtained by the root-sum-of-squares (RSS) method of calculating tolerances.[2] The squares of the tolerances are all added, and then the square root of the sum gives the variance of the assembly. If we had simply taken worst-case tolerances the design would not have worked because the worst-case numbers were too large. The RSS method assumes that the parts have a typical Gaussian distribution of sizes. Statistically, taking the RSS only excludes 0.27% of the possible cases. This was not considered completely satisfactory, so our goal was to have 1.5 times the RSS figure as the minimum design criterion for each key area. A simple scoring method of comparing the actual RSS figure for the key area with the goal of 1.5RSS gave us the "goodness" figure. Because the key areas are not independent, this allowed us to view the overall capability of the design and prevented conflicts such as improvements by a designer in one area increasing margin at the expense of another key area.

The use of a consultant to check the design and the use of the design margin index strengthened our execution of the mechanical design, forcing us to change the design and production processes of some parts.

A Change of Direction

As the design approached completion and we were ready to start production prototypes in manufacturing, the division reassessed the manufacturing strategy. It was clear that the product would consume more resources than the division was willing to commit. The increase in volumes of the DDS-1 products and the launch of the DDS-2 drive at nearly the same time would cause a bulge in resource requirements. The decision was made to find a partner to manufacture the autoloader mechanism, while HP supplied the DDS-2 drive.

Büro-und DatenTechnik (BDT) GmBH was selected and from November 1992 worked closely with us to take the autoloader into manufacturing. BDT was selected for their manufacturing expertise and quality in producing similar products, including paper sheet feeders for printers and another autoloader. The partnership has proved extremely successful. Their engineers have looked critically at the design and improved it. They have developed it and taken it successfully into manufacturing. HP was able to redeploy people on other projects, leaving only a core team to work with BDT.

Applications

To ensure the availability of software solutions that fully support the automation features of the HP C1553A, HP has developed the LABS (Low-Admin Backup for Servers) standard guidelines for software developers. The LABS guidelines define a set of software attributes that virtually eliminate human operator involvement in the backup process. The HP SureStoreTape 1200e product is available as a bundle with LABS software developed by Palindrome Corporation. In addition, several other solutions are available for different operating systems, such as Cheyenne ARCSERVE and Palindrome Backup Director for Novell networks, Arcada Backup Exec for Windows NT, and Legato NetWorker for UNIX® systems and Novell.

The autoloader can work in two ways, depending on the application software: random mode or sequential mode. In random mode the software issues SCSI medium changer commands to load a specific cartridge number. The software therefore has complete control over the operation. In sequential mode the user starts the backup from the chosen cartridge, say cartridge 1, and the host writes until the tape is full. The host then issues an SCSI unload command and the autoloader replaces the cartridge with the next one automatically. This allows easier integration into older systems that do not support the SCSI medium changer command set.

See page 18 for more about network backup applications.

Acknowledgments

The core team of designers deserves a lot of credit for achieving so much with so little: a small volume in the product, and a small team compared to the size of the job. Everyone contributed a lot. Thanks to Ray Dixon and Phil Williams for mechanical design, Mark Simms for mechanism firmware, Nigel Evans for displays, and Kevin Jones for drive and SCSI firmware. Bill Meikle, R&D project manager, guided us and contributed many important ideas. Jim Dow did the industrial design and he and Tom Cocklin, both from the HP Greeley facility, heavily influenced the ergonomics. Jeff Blanchard tried to break the prototypes to the best effect and guided us toward a reliable design. Outside this team of designers was the strength of our procurement and manufacturing engineers, especially Phil Catt, Dave Rush, and Roy Bradford, who enabled us to turn prototypes around and also contributed to the design. Dr. John Adie (design margin index) and Dave Moseley from Cranfield Institute of Technology provided an injection of ideas and analysis.

We must give a special mention to the BDT engineers and production personnel who took our basic design, criticized and improved it. They put in much hard effort and good faith to get the product into manufacturing.

References

1. J. Adie, "Optimising the Design Process," *Professional Engineering,* February 16, 1994.

2. J.R. Milner, "The Toleranced Design of the Model 520 Computer," *Hewlett-Packard Journal,* Vol. 35, no. 5, May 1984, pp. 10-11.

Automatic State Table Generation

The HP C1553A DDS tape autoloader requires a complex sequence of simple operations to carry out mechanical retries. These sequences are defined in tables. Cadre's Teamwork was used for input and an automatic tool was used to generate the tables to go in ROM.

by Mark J. Simms

The autoloader firmware for the HP C1553A digital audio tape autoloader was written in the C programming language to run on a Hitachi H8/325 processor. This processor is an embedded system microcontroller with built in ROM, RAM, I/O ports, timers, and serial ports. This allows a very low-cost implementation of the autochanger controller in which most of the functions can be carried out in a single chip. However, the largest ROM size available on the H8 series of processors was 32K bytes. This means that the complex retry algorithms required for controlling such a mechanical device needed to be implemented in as compact a manner as possible.

Our laboratory has a large amount of experience in producing table-driven systems. All of our products have had some form of table-driven control structures in some part of their firmware. However, experience had shown that there can be severe problems maintaining table-driven code because of the difficulty of maintaining the tables. This derives from the lack of readability of software written in C or assembly language that merely defines the contents of data structures. A lot of documentation needs to be added to the source code to explain the meaning of the entries. If this is not maintained, then the declarations rapidly become unreadable. This greatly increases both the time needed to implement changes and the risk of errors.

The designers of the HP 9145A cartridge tape drive and the HP 35470A DDS tape drive attempted to improve this situation by defining state machine languages that can be translated into C source code automatically. These languages offered powerful constructs for defining the tables in terms of state machines. The software would remain in one state until an event was detected. Then a set of actions would be carried out and a new state entered. This approach made the table definition much more readable than the basic data declarations and greatly alleviated the maintenance problems. However, the state machine languages suffered from many of the problems that are characteristic of "unstructured" programming techniques. There was no observable flow in the source code since transitions were permitted between any two states. This made it very difficult to follow the flow of the program and determine what sequence of actions had occurred.

To aid in documenting these state machines, the Teamwork structured analysis tool from Cadre Technologies was adopted. This allowed the initial problem analysis to be carried out graphically. A state transition diagram was produced to document the desired solution (see Fig. 1). This was then implemented using the state machine language. However, as the state machine language description was modified, the diagram gradually became more and more out of date and was updated periodically. This meant that while the diagram could be used to gain an initial familiarity with the software, it could never be guaranteed to be completely accurate.

With the HP C1553A autoloader, these problems became more serious. The state tables were to be used very extensively for mechanical control. Also, there was a very strong need to communicate the control algorithms to mechanical engineers and the product test team. This required that good accurate documentation be available to all within the division. It was felt that any manual system for maintaining such documentation would prove unusable in real situations. As a result, a decision was made to generate the state tables directly from the Teamwork diagrams. This would ensure that the diagrams were always accurate reflections of the software.

Design Implications

Analysis of the HP C1553A motor control software showed that the software divides into two major sections. The first is a number of routines that handle the low-level control of the mechanism. These routines control the motors and solenoids, read sensors, and track the position of the mechanism. They track control information and map that onto the control signals required to operate the motors in the correct direction at the correct power level. They debounce input readings and map them into mechanism position information. These routines are implemented in C and use global variables to interface with the rest of the software. The routines are called in sequence to carry out all the necessary interfacing to the mechanism.

The second section is the control sequencing. This contains a number of state machines. Some of these are directly linked to the individual mechanism parts. Others sequence individual mechanism operations together in response to a single command. These state machines interface with the low-level routines by means of the control information and mechanism position global variables.

Since several state machines are running concurrently with the low-level routines, this concurrency must be reflected in the software. Each state machine, when called, checks to see if a transition needs to be made. If not, control is returned immediately. If a transition is needed, that transition is executed and control is returned. This ensures that all the state

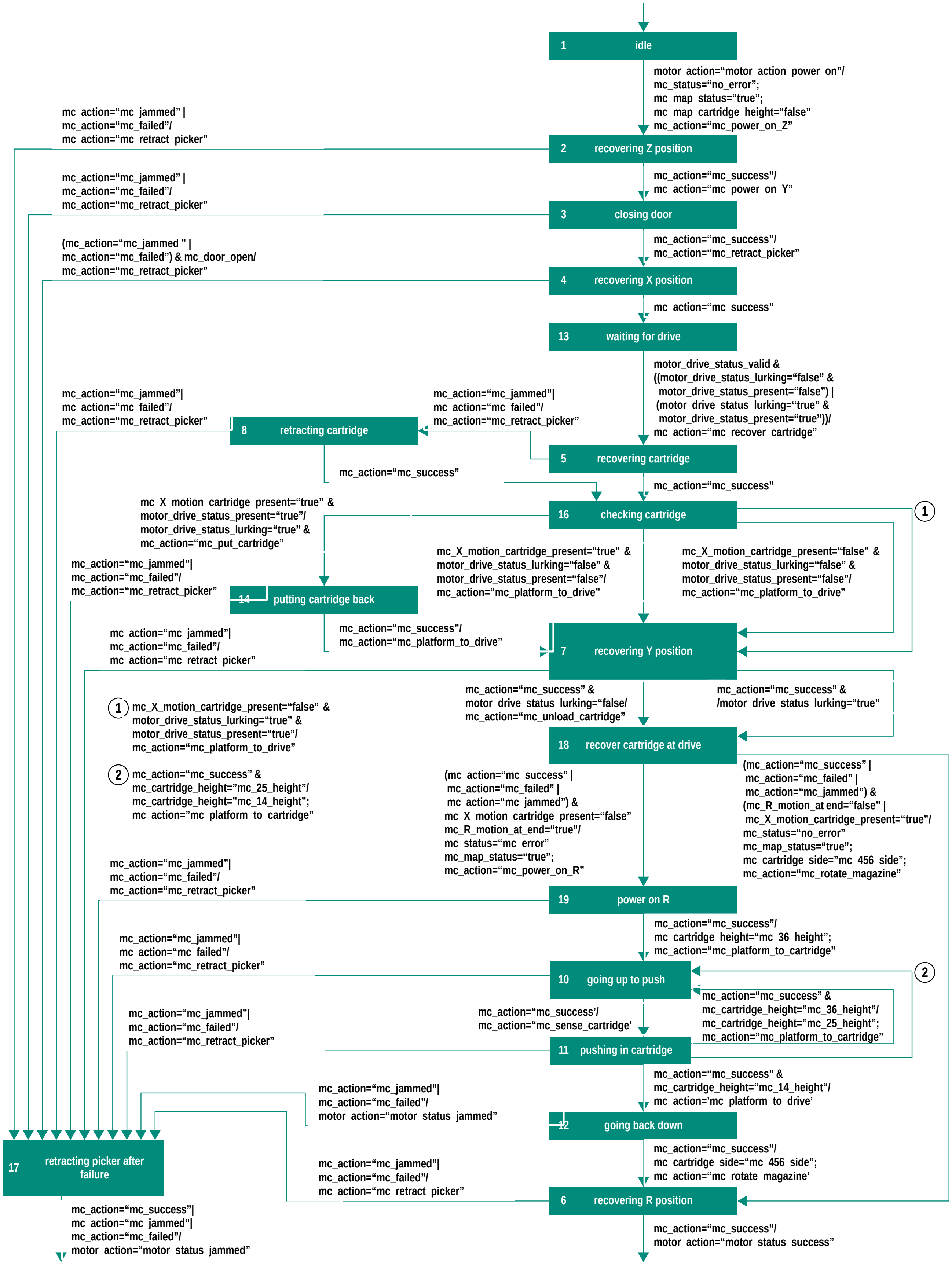

1 idle
motor_action="motor_action_power_on"/
mc_status="no_error";
mc_map_status="true";
mc_map_cartridge_height="false"
mc_action="mc_power_on_Z"
mc_action="mc_jammed" |
mc_action="mc_failed"/
mc_action="mc_retract_picker"
2 recovering Z position
mc_action="mc_success"/
mc_action="mc_power_on_Y"
mc_action="mc_jammed" |
mc_action="mc_failed"/
mc_action="mc_retract_picker"
3 closing door
mc_action="mc_success"/
mc_action="mc_retract_picker"
(mc_action="mc_jammed " |
mc_action="mc_failed") & mc_door_open/
mc_action="mc_retract_picker"
4 recovering X position
mc_action="mc_success"
13 waiting for drive
motor_drive_status_valid &
((motor_drive_status_lurking="false" &
motor_drive_status_present="false") |
(motor_drive_status_lurking="true" &
motor_drive_status_present="true"))/
mc_action="mc_recover_cartridge"
mc_action="mc_jammed"|
mc_action="mc_failed"/
mc_action="mc_retract_picker"
8 retracting cartridge
mc_action="mc_jammed"|
mc_action="mc_failed"/
mc_action="mc_retract_picker"
5 recovering cartridge
mc_action="mc_success"
mc_action="mc_success"
mc_X_motion_cartridge_present="true" &
motor_drive_status_present="true"/
motor_drive_status_lurking="true" &
mc_action="mc_put_cartridge"
16 checking cartridge
1
mc_action="mc_jammed"|
mc_action="mc_failed"/
mc_action="mc_retract_picker"
14 putting cartridge back
mc_X_motion_cartridge_present="true" &
motor_drive_status_lurking="false" &
motor_drive_status_present="false"/
mc_action="mc_platform_to_drive"
mc_X_motion_cartridge_present="false" &
motor_drive_status_lurking="false" &
motor_drive_status_present="false"/
mc_action="mc_platform_to_drive"
mc_action="mc_jammed"|
mc_action="mc_failed"/
mc_action="mc_retract_picker"
mc_action="mc_success"/
mc_action="mc_platform_to_drive"
7 recovering Y position
mc_action="mc_success" &
motor_drive_status_lurking="false/
mc_action="mc_unload_cartridge"
mc_action="mc_success" &
/motor_drive_status_lurking="true"
1 mc_X_motion_cartridge_present="false" &
motor_drive_status_lurking="true" &
motor_drive_status_present="true"/
mc_action="mc_platform_to_drive"
2 mc_action="mc_success" &
mc_cartridge_height="mc_25_height"/
mc_cartridge_height="mc_14_height";
mc_action="mc_platform_to_cartridge"
18 recover cartridge at drive
(mc_action="mc_success" |
mc_action="mc_failed" |
mc_action="mc_jammed") &
mc_X_motion_cartridge_present="false"
mc_R_motion_at_end="true"/
mc_status="mc_error"
mc_map_status="true";
mc_action="mc_power_on_R"
(mc_action="mc_success" |
mc_action="mc_failed" |
mc_action="mc_jammed") &
(mc_R_motion_at end="false" |
mc_X_motion_cartridge_present="true"/
mc_status="no_error"
mc_map_status="true";
mc_cartridge_side="mc_456_side";
mc_action="mc_rotate_magazine"
mc_action="mc_jammed"|
mc_action="mc_failed"/
mc_action="mc_retract_picker"
19 power on R
mc_action="mc_success"/
mc_cartridge_height="mc_36_height";
mc_action="mc_platform_to_cartridge"
mc_action="mc_jammed"|
mc_action="mc_failed"/
mc_action="mc_retract_picker"
10 going up to push
2
mc_action="mc_success" &
mc_cartridge_height="mc_36_height"/
mc_cartridge_height="mc_25_height";
mc_action="mc_platform_to_cartridge"
mc_action="mc_jammed"|
mc_action="mc_failed"/
mc_action="mc_retract_picker"
mc_action="mc_success'/
mc_action="mc_sense_cartridge'
11 pushing in cartridge
mc_action="mc_success" &
mc_cartridge_height="mc_14_height"/
mc_action='mc_platform_to_drive'
mc_action="mc_jammed"|
mc_action="mc_failed"/
motor_action="motor_status_jammed"
12 going back down
17 retracting picker after failure
mc_action="mc_success"/
mc_cartridge_side="mc_456_side";
mc_action="mc_rotate_magazine'
mc_action="mc_jammed"|
mc_action="mc_failed"/
mc_action="mc_retract_picker"
6 recovering R position
mc_action="mc_success"|
mc_action="mc_jammed"|
mc_action="mc_failed"/
motor_action="motor_status_jammed"
mc_action="mc_success"/
motor_action="motor_status_success"

Fig. 1. State transition diagram.

machines and low-level routines can be called in sequence, thereby maintaining the required concurrency.

From this analysis, the following design criteria were derived for the state machine implementation:

- The state machines must be able to respond to the values of mechanism position variables and execute transitions accordingly.
- The state machines must be able to set mechanism control variables when a transition occurs.
- A timeout mechanism is required that can handle times up to 30 s with a resolution of 1 ms.
- Each state machine must execute at most one transition when executed.
- Each transition must be executed as a complete unit to lessen the risk of data contention problems.
- The state machine implementation must use the minimum space possible.
- The ability to store a history of trace information must be provided for debugging purposes.

Interpreting Teamwork/RT

The Teamwork/RT product provides a graphical state machine editor that has the following features:

- There is a set of states, each of which has a unique number and a unique name.
- There is an initial state, indicated by a single initial transition.
- Transitions out of states may enter other states or may indicate that the state machine has exited.
- Transition information indicates the condition under which the transition occurs and may give actions that are to be carried out on that transition.
- Each transition condition is a logical expression consisting of a number of comparisons of variables with values.
- Each transition action is an assignment of a value to a variable.

The full syntax of this state machine description is supported by the code generator tools. This gives a fairly rich design environment in which to work. It has the additional advantage that if the diagram is correct according to the Teamwork syntax checker, it should generate code correctly and that code should compile.

To parse the state machine diagrams, a program was written to access the Teamwork database. This retrieves the data structures for a complete diagram, parses them, and generates the required code table. The program connects to the Teamwork database, retrieves the state transition diagram, and follows the linked list of states. For each state, it identifies each transition that exits that state. For each transition, it parses the associated text and generates the required data structures for its condition, actions, and next state.

Finding the start and end states of a transition and finding the transition information that is bound to a given transition involves the concept of an instance number. Each item in a Teamwork diagram is given a number to identify it. This number is unique across all items in the given diagram, whether an item is a text block, a transition, or a state.

The instance numbers of the initial and final states and the text block associated with a given transition are stored in that transition's entry in the linked list of transitions. The state can be identified by searching through the linked list of states to find the one with the same instance number as in the transition entry. The text block holding the transition information can be identified by searching through the linked list of text blocks to find the one with the same instance number.

To increase the performance of the code generator, an array of pointers to text blocks and states is set up at the start of the program. The list of text blocks is searched and the entry in the array indexed by the instance number of a given block of text is set to point at that block of text. The list of states is searched and the entry in the array indexed by the instance number of a given state is set to point to that state's entry. This allows the start state, end state, and text block associated with a given transition to be found by a single table look-up each.

Parsing Transition Information

The transition information associated with a transition consists of an event or an event, a / character, and a semicolon-separated list of actions.

The event is a logical expression consisting of a number of comparisons. Comparisons are linked with logical OR operations indicated by the | character and logical AND operations indicated by the & character. The logical NOT operator, indicated by the ~ character, can be used to invert an expression or comparison. Order of execution can be indicated by the use of parentheses. Each comparison consists of either just a variable or a variable, a comparator symbol, and a constant value in double quotation marks. The comparators can be equal =, greater than >, less than <, not equal ~=, less than or equal to <=, or greater than or equal to >=.

Each action consists of the name of a variable, an equals sign =, and the value to be assigned to that variable in double quotation marks.

This gives a text block of the form:

```
mc_timed_out |
( mc_jam_sense &
  mc_picker_state=”mc_picker_open” ) /
mc_X_motion_direction=”mc_X_brake”;
mc_action=”mc_jammed”
```

This should be read as follows: If mc_timed_out is true, or both mc_jam_sense is true and mc_picker_state is mc_picker_open, then set mc_X_motion_direction to the constant value mc_X_brake and set mc_action to the constant value mc_jammed and transition to the next state.

This text is parsed using a parser written using the yacc and lex tools provided with the HP-UX* operating system. this generates a reverse Polish form for the logical expression along with the values required for the actions.

Generating the State Table

The major issue with the state table design was to implement the full syntax supplied by the Teamwork/RT state transition diagrams in as compact a form as possible. Performance was not an issue. The maximum response time required of the firmware was of the order of 3 ms. This was easily achievable with the designs chosen.

To produce a machine readable form of the transition information, each part is taken in turn.

To reference variables, the H8 processor uses 16-bit addressing. This can be truncated to 10 bits to limit the address space to the 1024 bytes of RAM available on the processor. Since there are over 3000 references to less than 50 variables, an index table is far more efficient. In the transition information, a six-bit index value is stored. This is used to look up the address of the variable in an array of pointers.

Since every variable in the logical expression is associated with a comparator, it would be useful to store the information indicating the comparator in the spare two bits with the variable index. Unfortunately, there are six comparators, which would require at least three bits to store. However, the comparators form pairs of inverses. Equal is the opposite of not equal, greater than is the opposite of less than or equal to and less than is the opposite of greater than or equal to. As such, the logical NOT operation is used so that only the three basic comparators have to be used in the state table, allowing the comparator to fit in the two spare bits of the variable index. Since most comparisons are equalities, this saves a great deal of space while preserving the simple table format with no items crossing byte boundaries. All comparisons are with 8-bit values. These are stored in the byte after the variable index and operator.

The logical operators are stored in single-byte values. This is wasteful since only a couple of bits are really required for these. However, it maintains the simplicity of the generator and interpreter by avoiding the necessity of table items crossing byte boundaries.

The expression is stored in the table in reverse Polish format with the comparisons as values. The expression is terminated with a zero byte to indicate where calculation should stop.

Each action consists of the six-bit index for the variable stored in one byte and the single-byte value to be assigned stored in the next byte. The list of actions is terminated by a zero value to indicate the end of the list.

While most of the actions involve assignments of a single byte, the setting up of timeouts requires that a 16-bit value for the timeout in milliseconds be set. This variable is considered a special case by the state machine generator and two assignments are generated. This maintains the readability of the state machine diagram without adding complexity to the state machine table to handle 16-bit values.

The final part of the transition information is the next state. This is given as a single-byte value.

The state table entries for the transition text example given earlier are:

```
( mc_timed_out_index | 128 ), 1, /* == */
( mc_jam_sense_index | 128 ), 1, /* == */
( mc_picker_state_index | 128 ), mc_picker_open, /* == */
1, /* AND */
2, /* OR */
0,
mc_X_motion_direction_index, mc_X_brake,
mc_action_index, mc_jammed,
0,
0,
```

The transition data for each state is generated in turn. At the end of the transitions out of a given state, a single byte of value 255 is inserted to indicate the end of the transitions associated with that state.

To access the transitions associated with a given state, a table of pointers is used. The pointer indexed by a given state number points to the first transition from that state.

A single byte of data is used to hold the number of the current state. This is the same value as appears on the Teamwork/RT diagram.

Finally, a structure is generated that holds a pointer to the state pointer table, a pointer to the index variable, a pointer to the state variable, and a code to be used for logging.

The complete data structures for the state table are shown in Fig. 2.

State Table Interpreter

To execute the state table an interpreter routine is used. This reads the state table and carries out the actions required.

The interpreter routine is passed a pointer to the header structure for the state machine to be executed. It uses the pointer to the state variable to get the current state. It uses this as an index into the state pointer table to get a pointer to the transitions for the current state. It then scans the transition information.

The first thing in the transition information must be an expression, terminated by a zero byte. The interpreter routine checks each byte in turn to see if it is a comparison, an operation, or the terminating zero.

If either or both of the top bits are set, then it is a comparison and the next byte is the value to be compared. The bottom six bits of the byte are used as an index into the index table to get the pointer to the variable to be checked. This is then used to get the variable's value. The first two bits are used to determine whether the comparison should be for equality, greater than, or less than. The next byte is read, the comparison is executed, and the result is placed on a stack. The current byte is then incremented past both bytes of the comparison.

If neither of the top two bits is set but the value is not zero, then the byte indicates a logical operator. If the operator is AND or OR, then the top two values are removed from the stack, the operation is carried out on them, and the result is pushed back on the stack. If the operator is NOT, then the top value is pulled off the stack, inverted, and pushed back on. The current byte is then incremented past the operator.

This is repeated until a zero byte is found as the current byte. Then the top value is pulled off the stack to indicate whether the expression evaluated to TRUE or not.

If the expression is TRUE, the actions are read in turn as byte pairs until a zero byte is found as the first byte. For each byte pair, the value in the second byte is assigned to the variable pointed to by the pointer in the index table indicated by the index in the first byte. When the terminating zero byte is found, the next byte is assigned to the state variable and the interpreter routine exits, having completed a transition.

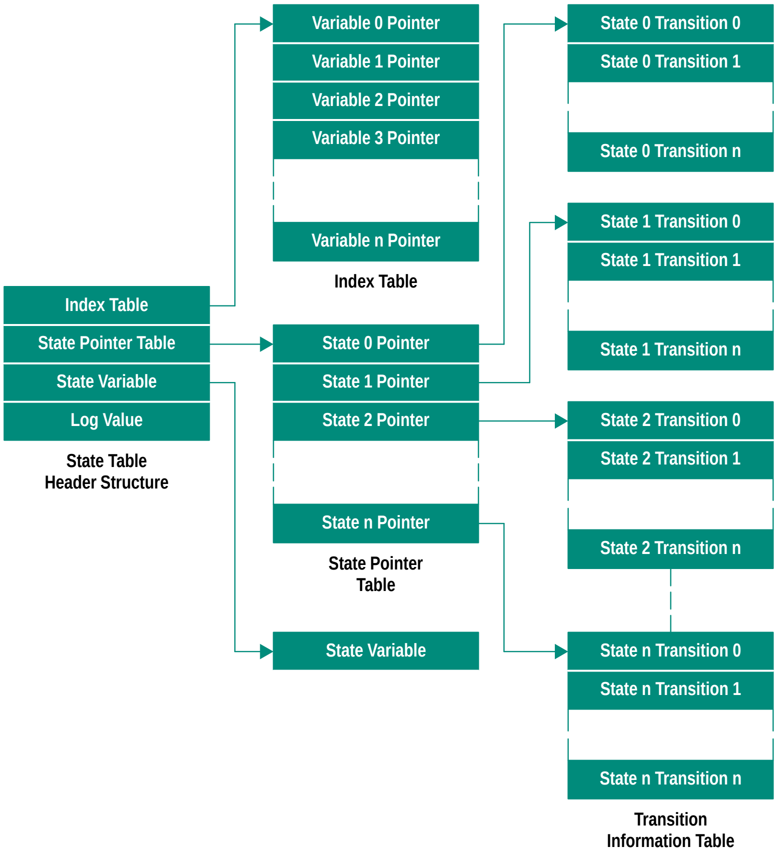


Fig. 2. State table data structures.

If the expression evaluates to FALSE, the actions are skipped over until a zero index is found. The next byte is skipped as well, since this is the next state value. The subsequent byte is checked to see if there are any more transitions to process. If this byte is not 255, then another transition follows and it is processed in the same manner. If this byte is 255, then all the transitions have been processed and none have conditions that are TRUE. The interpreter routine exits without carrying out any actions or altering the state variable.

Initialization and Exception Conditions

The state variable must be initialized at startup for the initial state of the system. Rather than provide a mechanism for the initialization routines to know the initial state, an additional state 0 is added to the state machine. Any transitions on the diagram that connect off-page are viewed as connecting to this additional state. Since the initial transition that identifies the initial state is the only one that can come from off-page, this allows the initial state to be set merely by setting it to zero. As soon as the state machine is executed, it will transition into the initial state on the diagram.

When an exception occurs, it is useful to be able to restart the state machine to initialize those areas of the mechanism under its control. Rather than connect all the transitions that handle exception conditions to the initial state, these are allowed to run off-page. In the Teamwork/RT notation, this means that the state machine has exited and that it should be restarted from the initial state on its next invocation. Since these off-page connectors connect to the additional state 0, this has exactly the desired effect. The exception transition can then be made to restart the state machine without the clutter of unnecessary connections on the diagram.

Debugging and Trace Logging

To be able to debug problems with a real-time control system, it is important to be able to get trace information back from the unit after a failure to determine what the system was doing at the time of failure. With a system that is largely implemented as state tables, it is possible to follow the flow of actions in terms of the state changes involved. As such, the state table interpreter was designed with a tracing function built in.

Whenever a state change occurs, the state table interpreter logs the current time as a 16-bit rolling clock counting in milliseconds, the 8-bit log value in the state table header structure that identifies which state machine is being executed, and the 8-bit value of the new state. The resultant 32-bit value is stored in an internal rolling buffer and is also transmitted on one of the H8 processor's built-in serial lines.

To decode this trace information, host-based interpreter programs are used. These decode the information in the trace log to identify exactly which state table and state within the table are involved. These are then printed out using the names on the Teamwork/RT state transition diagrams. This enables the changes in state to be followed on the diagrams merely by following the names given. The time each state was entered is listed alongside the name of the state to facilitate the interpretation of the mechanical factors that may have caused the state change. The use of milliseconds throughout, both for timeouts and for trace logging, allows engineers debugging failures to work in real-world values rather than units that are solely dependent on the software design.

During product test, the serial output from the microprocessor is monitored and the data is interpreted in real time. This allows both real-time debugging of failures and the gathering of a complete history of a test. In the field, the rolling buffer is returned from the autochanger via its SCSI interface. This only allows a recent history to be returned, but does not require additional hardware to monitor the serial port.

HP-UX is based on and is compatible with Novell's UNIX® operating system. It also complies with X/Open's* XPG4, POSIX 1003.1, 1003.2, FIPS 151-1, and SVID2 interface specifications.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

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

Using State Machines as a Design and Coding Tool

The wide acceptance of real-time extensions to structured analysis techniques have led to the use of state machine descriptions for the specification of systems in which state or sequence is a vital part. However, the techniques for implementing these specifications have remained poorly understood and haphazard, leading to implementations that are difficult to verify against the specification. This paper examines different approaches to the use of state machines and explores their advantages and disadvantages.

by Mark J. Simms

In the theory of state machines, two types of state machine model are defined: the Mealy Model and the Moore model. In the Mealy model, outputs are associated with transitions. When a transition happens, the output is generated. The format of this type of state machine as implemented in Cadre's Teamwork software is shown in Fig. 1. In the Moore model, outputs are associated with states. When a state is entered, the output is generated. The format of this type of state machine as implemented in Teamwork is shown in Fig. 2.

Traditionally, the Mealy state machine model has been used for software systems. Teamwork versions prior to 4.0 did not support the Moore state machine model. This is because software state-based systems typically only take action in response to an event. An output is set on the transition, although it may remain set indefinitely. This is sometimes not the optimum way of approaching a state-based model, but tends to reflect the way software engineers think.

As a result, this paper will concentrate on the Mealy model and references to state machines may be taken to assume this model unless there is a specific reference to the contrary. All of the concepts can be applied to Moore-model state machines because any Moore state machine can be implemented as a Mealy state machine, although the converse is not true.

When the original concepts of structured analysis were proposed by Tom DeMarco,[1] no concepts of state or sequence were used. This led to difficulty in modeling a large class of problems, including real-time control systems that are largely based around state and sequence and have little data flow

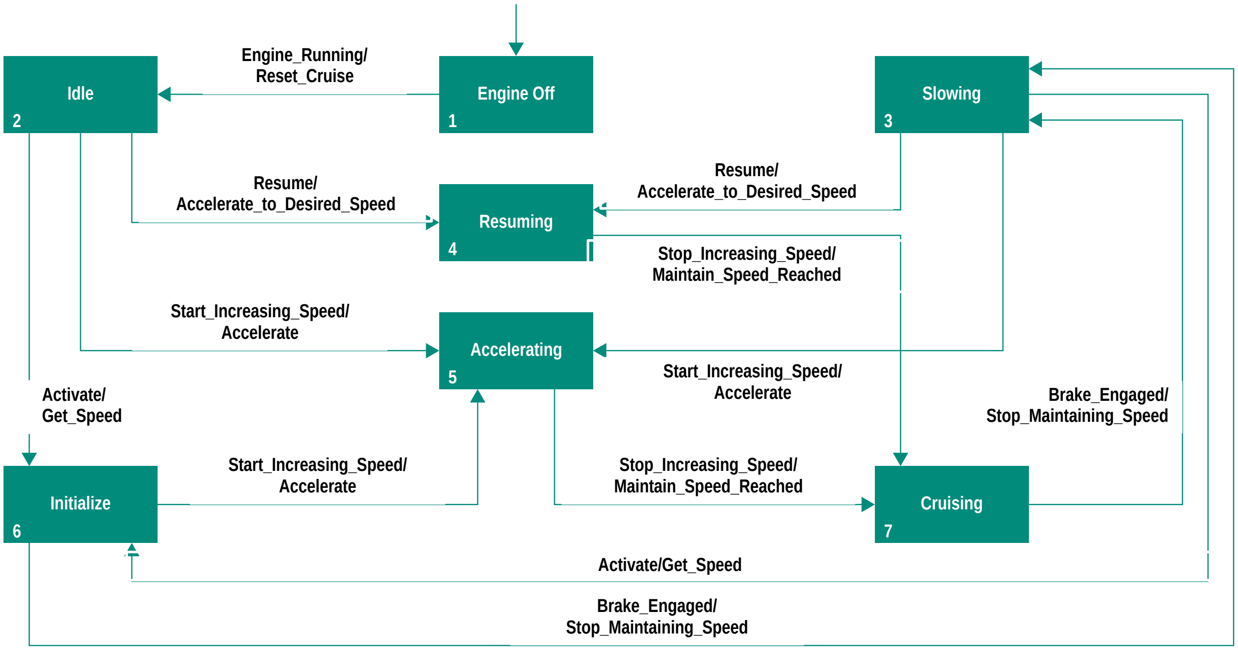


Fig. 1. Mealy state machine.

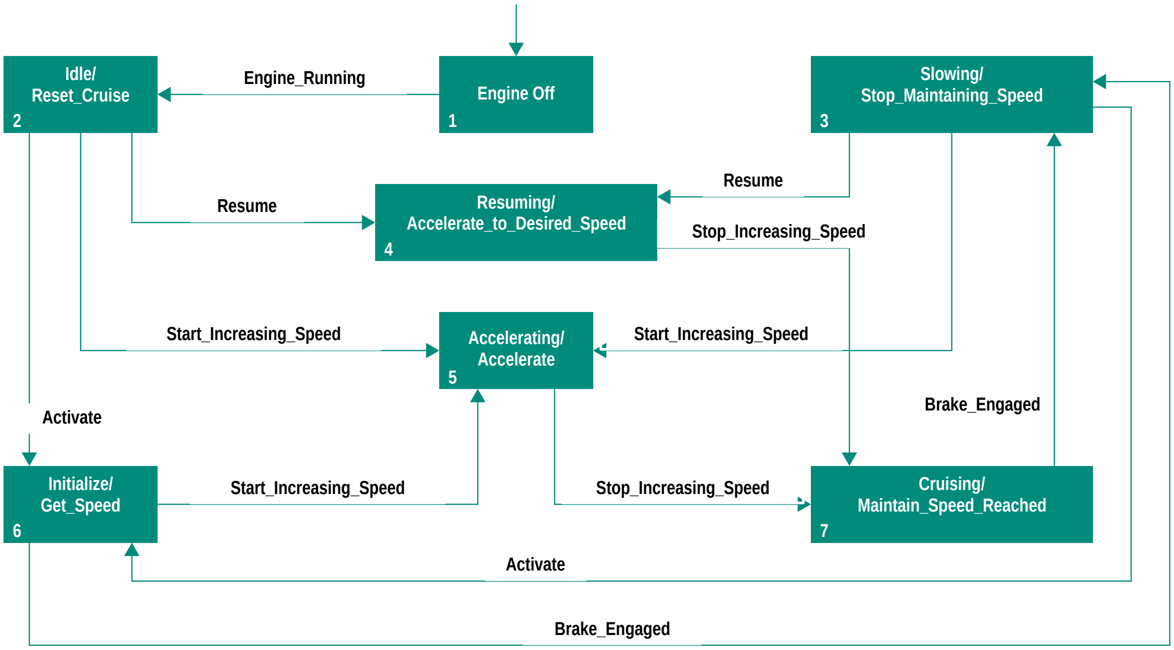


Fig. 2. Moore state machine.

content. As a result, two proposals were made for extensions to the structured analysis notation that would enable this type of problem to be modeled.

The first proposal came from Paul Ward and Steve Mellor,[2] who introduced a concept of signals to the structured analysis notation. Signals differ from data flows in that they carry timing information but no data. There are two types of signals: events and prompts. Events are generated by state machines and by data transformations in response to changes in the environment and cause state transitions within the state machine. Prompts are generated by state machines to control data transformations. Ward and Mellor proposed three types of control that could be exercised on data transformations: enable, disable, and trigger. Teamwork/SIM adds the kill prompt. The format of a Ward-Mellor state machine is shown in Fig. 3.

The second proposal came from Derek Hatley and Imtiaz Pirbhai,[3] who use the concept of a control flow. Control flows have the same properties as data flows, but are used for control purposes. They carry data and are continuously valid. The only way to pass timing information is to change the value of a control flow in response to an external event. Logical expressions involving input control flow values are used to determine when transitions of a state machine take

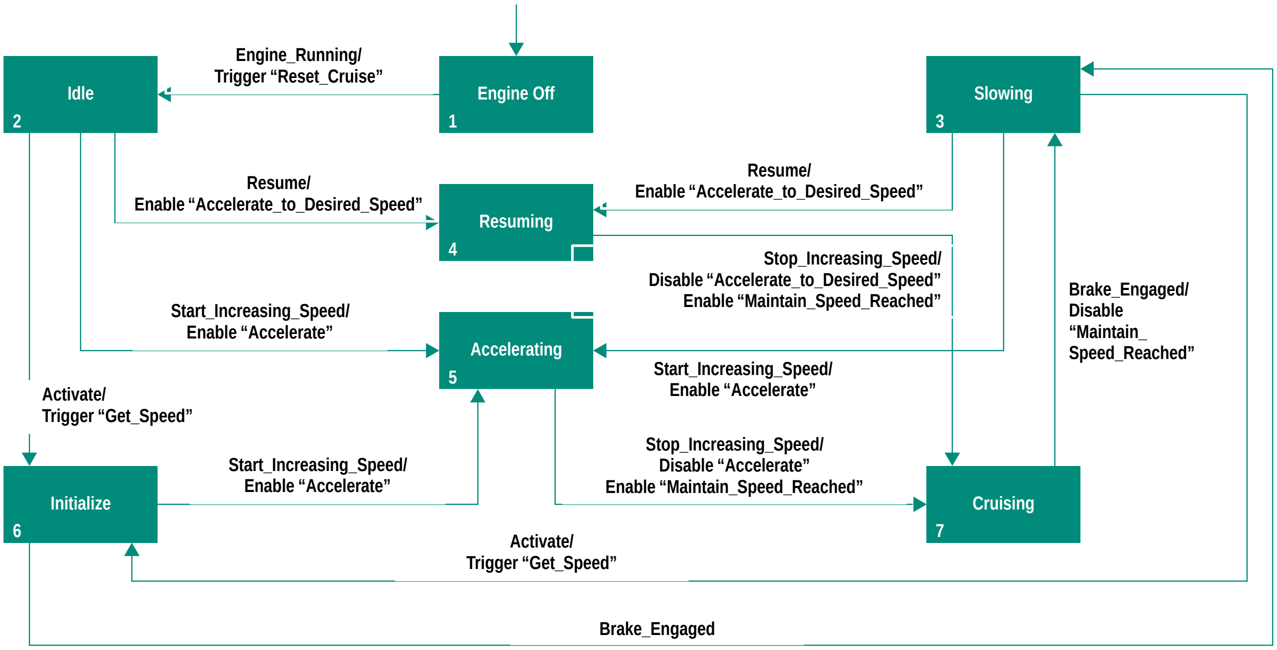


Fig. 3. Ward-Mellor state machine.

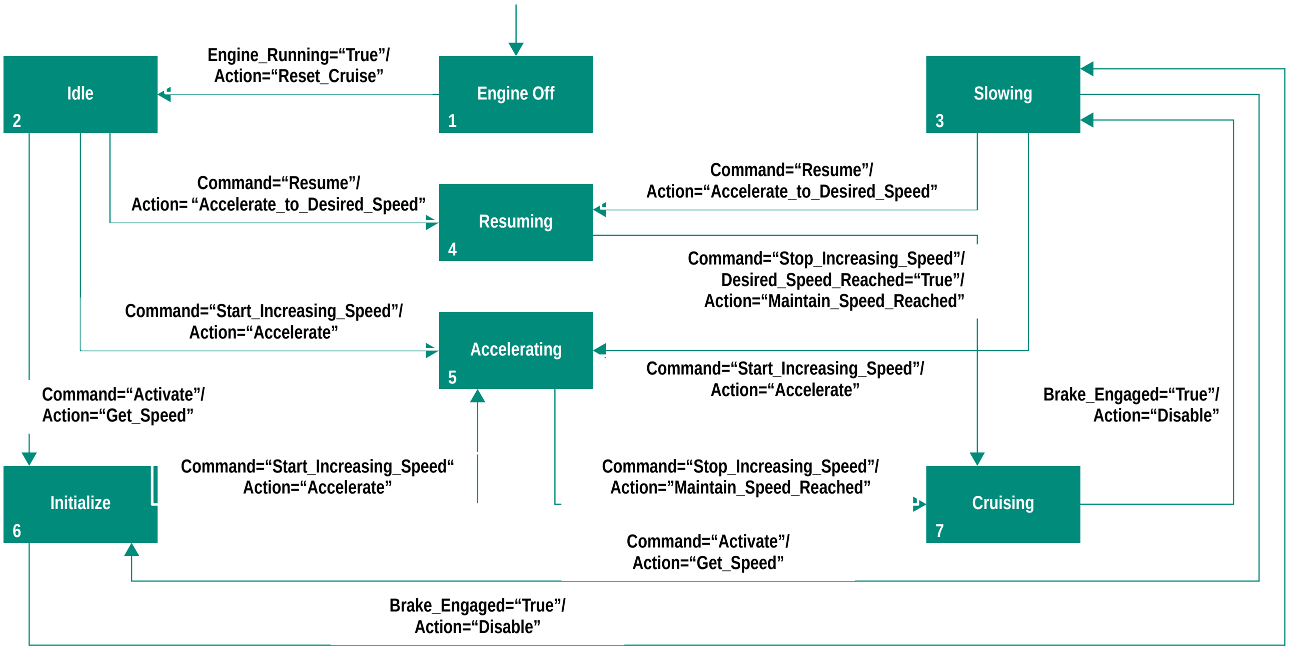


Fig. 4. Hatley-Pirbhai state machine.

place. Assignments are used in a state machine to control the values of output control flows. The format of a Hatley-Pirbhai state machine is shown in Fig. 4.

Ward and Mellor's signals can only be used in a Mealy state machine since they have an instantaneous quality and it makes little sense to have an instantaneous signal associated with a state. This could be interpreted as the signal being sent whenever the state is entered, but this is really associating it with all the transitions into the state.

Hatley and Pirbhai's control flows can be used with either state machine model. If the setting of control flows is associated with transitions in a Mealy state machine, they are assumed to be set until actively reset on another transition. If the setting of control flows is associated with states in a Moore state machine, then the control flow is deemed to be undefined if the state machine is in a state that does not actively output that control flow. This leads to a clearer definition of what is happening with a Mealy state machine and therefore a tendency to use this model.

Design Criteria

When using state machines as a design tool rather than an analysis tool, the method of implementing the state machine must be considered to give the design a rigid definition. In particular, the means of passing the external inputs to the state machine and the way the state machine interfaces to the procedural flow of the rest of the code must be well-defined.

Ward and Mellor recommend that the analysis be partitioned into tasks according to the number of state machines in the system. In addition to the state machine, the data transformations that it prompts and the event recognizers that supply it with events may be in the same task. This leads to a very simple interface to the rest of the code. The state machine is the top-level module of the task. It calls an event recognizer to get an event from the outside world. The event recognizer returns an event to the state machine if one is available or returns a code indicating no event if no event is available.

The state machine then executes the transition by flagging data transformations as enabled, disabled, or triggered and changing to the next state. It executes any enabled or triggered data transformations by calling the relevant procedures. Finally, it sets the state of triggered data transformations to disabled before calling the event recognizer again.

This implementation approach has a few drawbacks. First, if there are a large number of state machines, then the number of tasks can become very large. For systems that have to be implemented on hardware with limited power, this can be very wasteful. To get around this problem, a system by which multiple state machines can be implemented in a single task is required. This can be done by making the state machine return control to the calling procedure after each loop. This allows multiple state machines to be called in sequence. This also allows other data transformation modules to be called in the same sequence, imitating the parallel nature of the analysis.

Secondly, in systems that are capable of suspending tasks, the Ward-Mellor approach is very wasteful of processor time. In this type of system, the event passing mechanism should be built into the operating system in such a manner that tasks can block until an event is generated. This works well if triggers are the only prompts required. If data transformations must be enabled and disabled, the operating system must be used to do this. However, this might produce excessive operating system overhead.

The most efficient implementations based on this type of design require that only triggers be used, but multiple state machines can be executed in the same task. With these restrictions, the Ward-Mellor design approach can be highly efficient for large and medium-sized systems. A real-time operating system handles the scheduling of the tasks and can block a task waiting for a semaphore to be set. The semaphore can be set either by another task or by an interrupt service routine responding to an external event. A counting semaphore can be used to allow multiple events to activate a single task. Alternatively, a message queue mechanism can be used to implement a similar technique. If events are passed between state machines in the same task, then this can be implemented by flags that are checked by the event recognizer before suspending.

The top-level module in each task is responsible for calling the event recognizer routine. This blocks until the task's semaphore is set. It then determines what event occurred and returns the corresponding event code to the top-level module. The appropriate state machine is called based on the event code and passed the code as a parameter. The state machine determines the data transformation modules to be triggered and executes them in turn. It assigns the new state and returns. The event recognizer is then called again.

Hatley and Pirbhai offer far less guidance than Ward and Mellor on how to convert structured analysis state machines into designs. However, the meaning of the state machine appears more obvious at first sight, leading to a fairly obvious design.

Since the control flows are simple data values, these are easily implemented as static variables. The state machine needs only to read the variables, evaluate the logical expressions on the transitions in sequence until one is found to be true, execute the assignments on that transition and assign the new state. This continues with the transitions from the new state.

This approach has two problems. First, the order in which the transitions are processed is arbitrary. This means that the actions performed by the implementation are not necessarily uniquely determined by the state machine description. This is, however, a problem with the underlying analysis method. It is the responsibility of the analyst to ensure either that no two transitions become active simultaneously or that the system will operate within specification even if they do.

Second and more serious, the state machine offers no obvious way of interacting with the environment other than through the static variables. This means that the state machine would ideally run in its own task interfacing with the rest of the system via the variables representing the control flows. This is a highly inefficient implementation. This can be improved by implementing the state machine in such a manner that it returns control to the module that called it after each cycle. This allows several state machine modules and associated data transformation modules to be placed in the same task. This improves efficiency somewhat, but still uses processing time when nothing is being done.

Because of the inefficiency of this sort of implementation it can only be used where there is sufficient processor time to spare. However, it does have advantages when there is no operating system or where the operating system only offers basic functionality. Since the state machine operates on global variables, some simple data transformations can be incorporated into the state machines. This can produce a design that is very easy to learn and to follow and that can be implemented very easily.

It is sometimes advantageous to add some of the features of control-flow-based state machines to signal-based state machines. This involves adding states that control the flow of the state machine based on the value of variables. There should always be at least one exit transition active whenever such a state is entered. These states are not true states since the state machine never waits in the state. Instead they are merely decision points that affect the future flow through the state machine. Under certain circumstances, this can greatly simplify the state machine.

Implementation Techniques

There are two major approaches to implementing state machines in software. The first is to generate inline code that executes the state machine logic directly. This is the faster solution, but uses a large amount of code space. This approach is typically used on large systems where there is a lot of memory and on systems where response time is very important.

The second approach is to generate a table that encodes the state machine logic in a compact manner that is then interpreted by a separate state machine interpreter. This produces very compact code that is suitable for systems where code space is low and response time is not critical.

Both signal-driven and control-flow-driven state machines can be implemented either as inline code or as tables. Various different coding schemes can be used depending on the complexity of the syntax used for the state machine. In systems where the state machine module never returns, the program counter can be used to determine the state. More usually, however, a static state variable is used to maintain the state between calls of the state machine module.

For a Hatley-Pirbhai type state machine, this leads to an implementation of the form shown in Fig. 5. The state machine is implemented as a single C function. A static variable within the function holds the state. This variable is initialized to the initial state of the state machine when the task is started. The function is structured as a switch statement in which each case limb is the processing for a given state. The

```
void state_machine( void )
{
  static state = Engine_Off;
  switch ( state ) {
  case Engine_Off:
      if ( Engine_Running == True ) {
         Action = Reset_Cruise;
         state = Idle;
      }
      break;
  case Idle:
      if ( Command == Resume ) {
         Action = Accelerate_to_Desired_Speed;
         state = resuming;
      } else if ( Command == Start_Increasing_Speed ) {
         Action = Accelerate;
         state = accelerating;
      } else if ( Command == Activate ) {
         Action = Get_Speed;
         state = Initialize;
      }
      break;
  case ( Slowing ):
      .
      .
      .
  break;
  }
}
```

Fig. 5. Code for Hatley-Pirbhai state machine.

```
#define number_of_enabled_functions 3
#define Accelerate_to_Desired_Speed_INDEX 0
#define Accelerate_INDEX 1
#define Maintain_Speed_Reached_INDEX 2

void (*function_array)()[ number_of_enabled_functions ] =
  {     accelerate_to_desired_speed,
        accelerate,
        maintain_speed_reached
  };

int enabled_array[ number_of_enabled_functions ]
  = { FALSE, FALSE, FALSE };

void state_machine( int event )
{
  static int state = Engine_Off;
  int i
  switch ( state ) {
  case Engine_Off:
        if ( event == Engine_Running ) {
              Reset_Cruise();
              state = Idle;
        }
        break;
  case Idle:
        if ( event == Resume ) {
              enabled_array[
                    Accelerate_to_Desired_Speed_INDEX ] =
                         TRUE;
              state = Accelerating;
        } else if ( event == Start_Increasing_Speed ) {
              enabled_array[ Accelerate_INDEX ] = TRUE;
              state = Accelerating;
        } else if ( event == Activate ) {
              Get_Speed();
              state = Initialize;
        }
        break;
  case Slowing:
        .
        .
        .
  break;
  }
  for ( i = 0; i < number_of_enabled_functions; i++ ) {
        if ( enabled_array[ i ] ) {
              function_array[ i ]();
        }
  }
}
```

Fig. 6. Ward-Mellor state machine code.

case limb corresponding to the currently active state is executed when the function is called. This tests the exit conditions for the state in a series of if statements. If one of these is qualified, then the actions are carried out in the statements attached to the if statement and the new state is assigned. The function is then exited to allow other processing to be carried out in the same task.

For a Ward-Mellor type state machine, this approach leads to an implementation of the form given in Fig. 6. Two arrays are used. The first holds pointers to all the functions that are enabled and disabled. The second holds a flag indicating whether the function is currently enabled or disabled. The function that implements the state machine has many similarities to that for the Hatley-Pirbhai state machine. It is structured as a switch statement with a case limb for each state. Each case limb consists of if statements that determine if the corresponding actions should be executed. The conditions used are far more restrictive than in the Hatley-Pirbhai case. They check whether the event code passed to the function as a parameter is the value corresponding to the required event. The actions are either triggers, which simply call the function that corresponds to the required action, or enables and disables, which set and clear flags in the array.

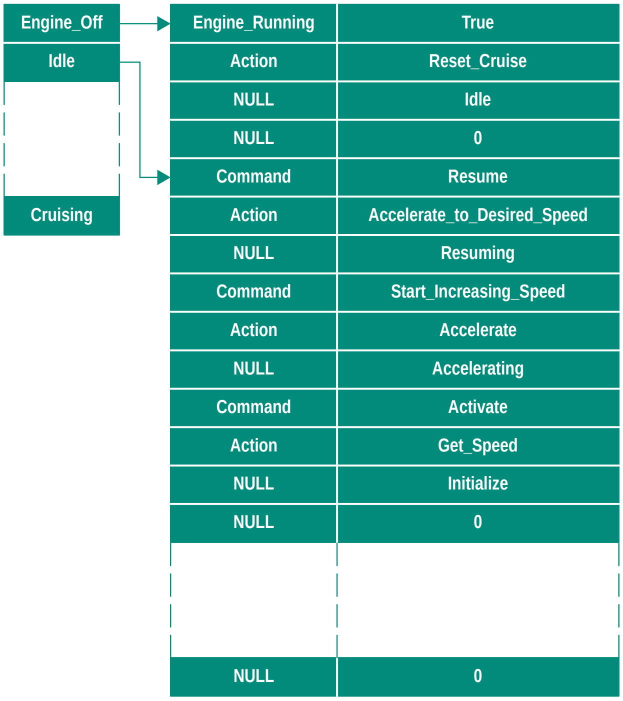
At the end of the function, the array of flags is searched and the corresponding functions are called via the pointers in the second array.

This design makes a few assumptions about the calling environment. First, the event recognizer functionality is not in the state machine itself. The event recognizer is executed in the calling environment and the event code is passed as a parameter to the function. Secondly, the calling environment must not block because this would prevent the enabled modules from being executed. Since the enabled modules are called as functions from the state machine, the state machine function must be executed at least as often as the enabled modules need to be called.

Since the content of these state machines is fairly simple and well-defined, machine code is a somewhat inefficient way of storing it. As a result, in systems where code space is at a premium, it may be advantageous to implement the description of the state machine as a table that is interpreted by a separate routine. The following paragraphs describe one possible way of doing this for a Hatley-Pirbhai state machine.

The state machine consists of an array of pointers and a state variable. The state variable is used as an index into the array to get the address of an array of structures containing a pointer and a value. Each transition consists of a single condition structure followed by a series of action structures and then a structure with a null pointer and the destination state indicating the end of the transition. The end of the transitions for a given state is indicated by another structure with a null pointer. This is depicted in Fig. 7.

To execute this state machine, an interpreter function is given the state variable and the list of pointers. Using the state variable as an index into the table, it uses the corresponding pointer to find the first structure corresponding to



Engine_Off
Idle
Cruising

Engine_Running	True
Action	Reset_Cruise
NULL	Idle
NULL	0
Command	Resume
Action	Accelerate_to_Desired_Speed
NULL	Resuming
Command	Start_Increasing_Speed
Action	Accelerate
NULL	Accelerating
Command	Activate
Action	Get_Speed
NULL	Initialize
NULL	0
NULL	0

Fig. 7. State tables for a table-driven state machine.

a transition from the current state. It then checks to see if the condition is true by comparing the variable pointed to by the pointer with the value stored in the structure. If they are different, the table is scanned until a structure with a null pointer is found indicating the end of that transition. The procedure is then repeated until either a true condition is found or a condition with a null pointer is found. If a condition with a null pointer is found, all the conditions have been tested and the interpreter function returns to its calling environment.

If a condition is found to be true, the interpreter function scans the subsequent structures and assigns each value in the structure to the variable indicated by the corresponding pointer. It continues to do this until a structure with a null pointer is encountered, indicating the end of the transition. It assigns the value in this structure to the state variable to cause a change of state. It then returns to its calling environment.

A similar approach can be used for Ward-Mellor state machines. Event codes and pointers to functions to be enabled or disabled are encoded in the tables. This requires a slightly more flexible table format, but the principles are the same.

Automatic Generation

Once a rigorous mapping has been defined between the state machine design and the code to be produced from it, it is theoretically possible to design tools that can translate state machine descriptions directly to the source code for the final software. With the extensive use of graphical state machine editors for analysis, this gives the potential for a graphical form of source code that is easy to follow and easy to modify, removing some of the major problems of software maintenance.

Analysis tools are not designed with direct code generation in mind. As a result, the mappings from state machine description to code must be defined by the engineers on the project using the tools. This allows a lot of flexibility for experienced engineers to produce mappings that are highly tuned to the application concerned. It does mean that there is a requirement for anyone wishing to learn about the code to determine what the mapping is and why it was chosen. Once this is understood, the functionality of the code can be followed from the state machine diagrams.

For a structured analysis tool to be able to fulfill this role, it must have a number of features. The following are some of the most important:

- The tool must support the state machine features required by the implementers. This includes Mealy and Moore state machines, types of conditions that cause transitions, types of actions that can be placed on transitions or in states, size and complexity of diagrams, and so on.
- It must be possible to integrate the tool into the configuration management system. Since the state machine diagrams are now source code, it is vital that they be treated with a high degree of care and attention.
- The tool must be able to access the diagrams. If the data is stored in any sort of database system, the appropriate access routines must be supplied.
- The tool must keep diagrams in a documented format that does not change between revisions. Continually modifying code generation programs to track the format of the diagrams is unacceptable.

Once these features have been established, code generation becomes a simple task of defining the mapping between the state machines and the code and then designing the translator program and any interpreter routines for table-driven state machines. The rest of the code can then be designed from the remainder of the structured analysis with the state machine implementation in mind.

There have been a number of dedicated code generator programs on the market for some time, many of which use state machines as part or all of their input tools. These systems come with rigorously defined semantics for their diagrams so that users of the system can rapidly understand designs with which they are unfamiliar. These programs also come with a defined mapping of the diagrams to code.

The biggest problem with this sort of system is that the features of the state machines and the mapping to code supplied by the vendor may not be ideal for the implementation that is required for a given problem. Few systems currently available offer any ways of tuning the implementation for a given set of design criteria. This is one of the major features to look for in such a system.

A hybrid approach is possible in which a code generator tool is used, but a custom front end is included to tune the resulting code from the generator. This might be done either by treating the tool in the same way as a generic structured analysis tool and accessing the diagrams and generating code directly from them or by postprocessing the resultant code to optimize it for the given situation.

The hybrid approach is probably harder than designing a generator for a generic structured analysis tool, but the results could be better. The added rigor of code generators means that it is easier to use standardized semantics for the system. The semantics are more likely to be complete than those of a structured analysis tool.

Summary

The usefulness of state machines for specifying control applications has been well-proven. Their use in design and implementation is also showing a great deal of promise. It has shown major advantages in the following areas:

- Rapid code implementation because of the very close mapping of analysis, design, and code
- Ease of maintenance because of the availability of easy-to-read code in the form of state machine diagrams
- Compact implementation of a large proportion of the functionality of a problem because of the use of table-based state machines.

These advantages make the investment in tools for this technique well worth the effort and expense involved.

References

1. T. DeMarco, *Structured Analysis and System Specification,* Prentice Hall, 1978.

2. P.T. Ward and S.J. Mellor, *Structured Development for Real-Time Systems,* Prentice Hall, 1985.

3. D.J. Hatley and I.A. Pirbhai, *Strategies for Real-Time System Specification,* Dorset House Publishing, 1987.

An Event-Based, Retargetable Debugger

Remote and event-based debugging capability, a sophisticated graphical user interface, and adaptability to different languages and target platforms are some of the features provided in this debugger.

by Arun K. Iyengar, Thaddeus S. Grzesik, Valerie J. Ho-Gibson, Tracy A. Hoover, and John R. Vasta

Software developers rely heavily on debuggers for both fixing bugs and analyzing programs. The information obtained by a debugger can be used to add new features and improve performance. Event-based debugging[1] is a powerful method for examining program behavior. In event-based debugging, the user instructs the debugger to respond to events that occur during program execution. The proper selection of events allows programs to be debugged at a higher level than would otherwise be possible.

This article describes the HP Distributed Debugging Environment (HP DDE).[2] HP DDE is distributed because it is capable of debugging programs executing on remote hosts. HP DDE has been ported and retargeted to several platforms and can be used to debug programs written in C, C++, FORTRAN, Pascal, and various assembly languages. The platforms that HP DDE can run on include the HP-UX* operating system, Domain/OS 68K, Domain/OS Prism, the SPARC implementation of Solaris, and the PA-RISC implementation of the OSF/1 operating system. HP DDE has a sophisticated graphical user interface that provides the user with point-and-click access to commands and program state. Finally, HP DDE has many features that support event-based debugging and debugging optimized code.

The modular architecture of HP DDE enhances its portability and has been a critical component in its success. HP DDE consists of a main debugger that communicates with several modules called *managers.* The main debugger contains support for generalized debugger functions, and the managers contain dependencies on specific languages, object code formats, target platforms, and user interfaces.

User-Visible Features

Event-Based Debugging in HP DDE

Almost all debuggers support a traditional debugging paradigm in which the user inserts breakpoints before critical points in the program and examines the state of the program after breakpoints are hit to find program errors. The disadvantage of this paradigm is that the user has to know where to insert breakpoints within the program. The user may also have to examine the state of the program at a number of breakpoints before obtaining the desired information.

In event-based debugging, the debugger does not return control to the user until an event defined by the user occurs. By choosing an appropriate event, the user can avoid stopping the program at points that are not critical. Event-based debugging allows the user to identify what is going on in a program without spending large amounts of time inserting breakpoints at crucial points.

Event-based debugging is achieved by monitoring the executing program at an interval or granularity level specified by the user. Whenever the debugger determines that a desired event has occurred, the debugger returns control to the user. A lower level of granularity, corresponding to frequent monitoring, allows the user to determine more accurately the location where a particular event occurs. However, a low granularity level can cause execution time to increase substantially. HP DDE provides several intervals for monitoring program execution, including every machine instruction, every source statement, and entry to or exit from each procedure. Monitoring can be restricted to specific procedures within a program. Typically, a user would use procedure-entry-and-exit-level granularity to narrow an event to a specific procedure. After the faulty procedure is located, the user can use source-statement-level or machine-instruction-level granularity within the procedure to determine the exact location of an event.

HP DDE contains many features for event-based debugging including conditional breakpoints, execution traces, data watchpoints, and event intercepts. Conditional breakpoints allow the user to halt execution at a given point in a program only if a set of conditions are met. Execution traces suspend program execution at intervals specified by the user. Data watchpoints allow the user to monitor a variable or memory range (program execution is suspended when a value corresponding to a watched variable or memory range changes). Event intercepts allow the user to regain control of the program when events such as program exceptions, shared library loading and unloading, thread creation and termination, and signals from the operating system occur.

The HP DDE command language allows the user to declare variables for use in a debugging session and contains loops, conditionals, and assignment statements to allow a wider range of event specification.

Examples

Suppose that the user desires to suspend execution every time a variable x becomes zero. This can be accomplished with the following HP DDE command:†

```
dde> watchpoint  x –do [if (x != 0) –then (go)]
```

The behavior when no monitoring interval is specified in the watchpoint command (as in this example) is to monitor the expression after every source statement.

As another example, suppose the user wishes to break upon entry to function foo only if argument x is nonnegative. This can be accomplished using a conditional breakpoint:

```
dde> breakpoint foo –do [if (x < 0) –then (go)]
```

It is possible for the user to examine data structures while a program is suspended. The following HP DDE commands will print all positive elements of a 20-element array A:

```
dde> declare int dde_var
dde> set dde_var=0
dde> while dde_var < 20 –loop [if A[dde_var] > 0 –then \
        (print A[dde_var]);\
        set dde_var = dde_var + 1]
```

The declare command allocates space for new variables that can be used in debugging sessions, and the backslash (\) indicates that an HP DDE command continues on the next line.

The following commands will suspend execution whenever the sum of the 20 elements of A is greater than 100:

```
dde> declare int i
dde> declare int sum
dde> watchpoint A –do \
      [set i=0; set sum = 0; \
      while i < 20 –loop \
         [set sum = sum + A[i]; set i = i + 1]; \
      if sum <= 100 –then (go)]
```

Sequences of commands such as these can be stored in a file and used as input to HP DDE. In addition, macro expansion allows a user to define a single command that is automatically expanded by the command interpreter into multiple commands.

As a final example, the following commands will execute a program and print the number of times a multithreaded application switches between different threads:

```
dde> declare int ts_count
dde> set ts_count = 0
dde> intercept thread_switch –do [set ts_count = ts_count + 1; go]
dde> go; print ts_count
```

The User Interface

Several user interfaces have been designed for HP DDE. The most sophisticated is a graphical user interface based on the X Window System and OSF/Motif.[3] It features multiple windows, context-sensitive pop-up menus, and online help. The user can customize menus, command buttons, and key bindings. A typical HP DDE debugging session with four windows is illustrated in Fig. 1. The graphical user interface can manipulate up to six different types of windows:

- Target program window. This is the window from which HP DDE is invoked. All data to and from the program being debugged is directed to this window. The main parameter to the debugger is the name of the program to be debugged.
- Transcript display window. This is the debugger's main window. It has four components. The command entry line component at the bottom of the window is for keyboard entry of debugger commands. All debugger commands can be entered from the keyboard in this area. Above the command entry line is the transcript area, which displays input to the debugger (including commands accessed from the mouse) and debugger output. The buttons along the left side of this window provide quick access to common debugger commands. For example, the user can single-step the program (execute a single source statement) by clicking on the button labeled Step. These buttons can be reconfigured by the user to represent other commands. The pull-down menus across the top of the transcript display window allow access to all debugger commands via the mouse.
- Source code display window. This window displays the source code for the program being debugged.
- Assembly display window. This window displays the disassembled machine instructions for the program being debugged.
- Traceback display window. This window displays the current dynamic call chain of the target program.
- Variable display window. This window displays the values of variables or memory ranges for which watchpoints have been set. When a watched value changes, HP DDE suspends execution, notifies the user of the change in value, and updates the variable display window with the new value.

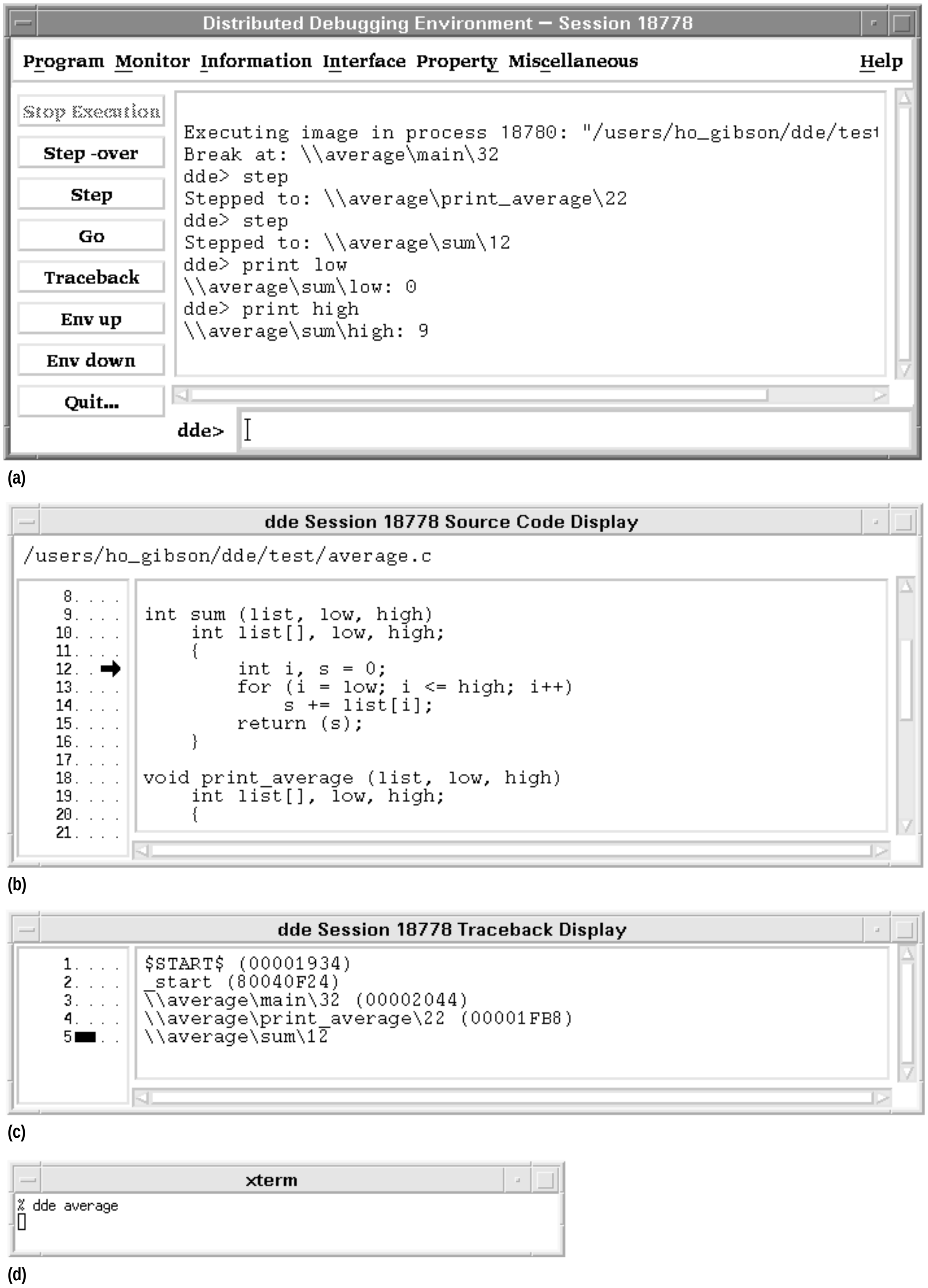
Context-sensitive pop-up menus allow the user to access common debugger commands with the mouse. Menus appear when the mouse is clicked. The menu that appears depends on the position of the mouse within a particular window at the time the mouse is clicked. For example, clicking on the mouse with the cursor positioned in the source code region brings up the menu displayed in Fig. 2. The cursor in Fig. 2 was positioned at the opening bracket of the expression list[i] at the time the mouse was clicked. The menu was thus seeded with the expression list[i]. If the cursor had been positioned over list, the menu would have been seeded with list instead of list[i]. The mouse can be positioned to print complex expressions in source programs without requiring the user to type in the expressions.

For even quicker access, double-clicking causes HP DDE to execute the first menu item instead of displaying the menu. The default pop-up menus contain actions that are commonly performed by users. However, pop-up menus can be easily reconfigured by users to contain actions that are common to a particular application.

Debugging Optimized Code

At a high level, optimization can be described as modifying instructions and memory references to improve program performance. Multiple source statements may map to the same machine instruction. In some cases source statements may not correspond to any machine instructions. Variables may be moved from memory into a register or from one register to another to eliminate costly memory references. Unless a debugger has some way of knowing how source statements and machine instructions have been rearranged and where a variable is located, it can be very difficult to

† HP DDE debugger commands can be entered in the command entry line area in the transcript display window described in the next section.


Distributed Debugging Environment — Session 18778
Program Monitor Information Interface Property Miscellaneous Help
Stop Execution
Step -over
Step
Go
Traceback
Env up
Env down
Quit...
(a)
dde Session 18778 Source Code Display
(b)
dde Session 18778 Traceback Display
(c)
xterm
(d)

```
Executing image in process 18780: "/users/ho_gibson/dde/tes
Break at: \\average\main\32
dde> step
Stepped to: \\average\print_average\22
dde> step
Stepped to: \\average\sum\12
dde> print low
\\average\sum\low: 0
dde> print high
\\average\sum\high: 9
dde>
```

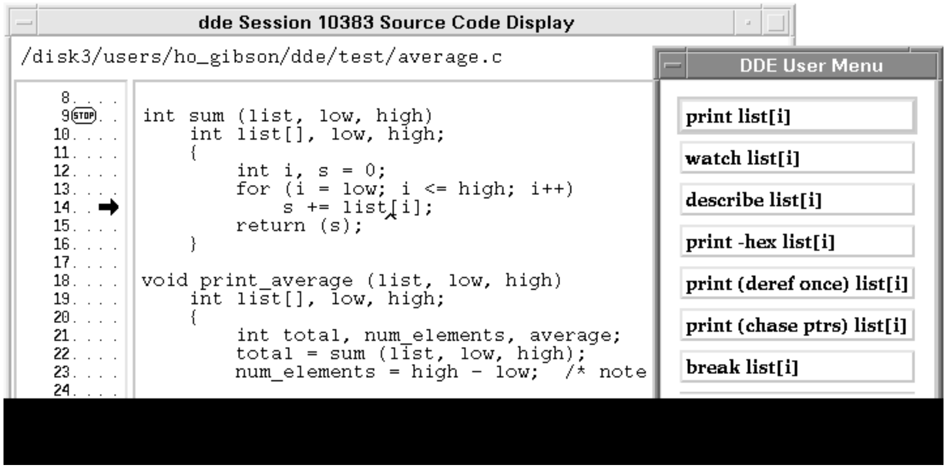
```
/users/ho_gibson/dde/test/average.c

 8. . . .
 9. . . .  int sum (list, low, high)
10. . . .      int list[], low, high;
11. . . .      {
12. . ➡        int i, s = 0;
13. . . .          for (i = low; i <= high; i++)
14. . . .              s += list[i];
15. . . .          return (s);
16. . . .      }
17. . . .
18. . . .  void print_average (list, low, high)
19. . . .      int list[], low, high;
20. . . .      {
21. . . .
```

```
1. . . .  $START$ (00001934)
2. . . .  _start (80040F24)
3. . . .  \\average\main\32 (00002044)
4. . . .  \\average\print_average\22 (00001FB8)
5■. .     \\average\sum\12
```

```
% dde average
```

Fig. 1. Some of the windows involved in a typical HP DDE debugging session. (a) Transcript display window. (b) Source code display window. (c) Traceback display window. (d) Target program window.


dde Session 10383 Source Code Display
DDE User Menu
print list[i]
watch list[i]
describe list[i]
print -hex list[i]
print (deref once) list[i]
print (chase ptrs) list[i]
break list[i]

```
/disk3/users/ho_gibson/dde/test/average.c

 8. . . .
 9 STOP . .  int sum (list, low, high)
10. . . .      int list[], low, high;
11. . . .      {
12. . . .          int i, s = 0;
13. . . .          for (i = low; i <= high; i++)
14. . ➡              s += list[i];
15. . . .          return (s);
16. . . .      }
17. . . .
18. . . .  void print_average (list, low, high)
19. . . .      int list[], low, high;
20. . . .      {
21. . . .          int total, num_elements, average;
22. . . .          total = sum (list, low, high);
23. . . .          num_elements = high - low;  /* note
24. . . .
```

Fig. 2. A context-sensitive pop-up menu. This menu was obtained by positioning the mouse cursor (denoted by the caret on line 14) in the source code display window and clicking on a mouse button.

Compiler Optimizations and Debugging

The object code produced by a compiler using straightforward compiling techniques is often not very efficient. In many cases, the object code can be made to run faster or take up less space through program transformations known as optimizations. Compilers that improve performance through code transformations are known as optimizing compilers.

In an unoptimized program, there is generally a one-to-one correspondence between a source statement and a group of one or more machine instructions. Optimizing compilers must preserve the correctness of a program, but after optimization, this one-to-one correspondence no longer exists in many cases, and the program may no longer execute in the order implied by the source code. This complicates debugging. Programmers must often resort to debugging assembly code to figure out how an optimized program is behaving.

In general, users should debug the unoptimized version of a program before using the optimizer. However, there are a number reasons why the ability to debug optimized code is desirable:

- A program may run correctly when compiled without optimization and fail when compiled with optimization. This can happen even if the optimizer is working correctly. For example, reordering statements may result in arithmetic overflow or underflow. An uninitialized variable or an out-of-bounds memory reference might cause a problem in the optimized version but not in the unoptimized version.
- The time or space requirements of an unoptimized program might be too big to allow adequate testing.
- Production code is often optimized. A customer may submit a bug report with a core file produced from an optimized program. It would be desirable to have the ability to analyze the core file.
- Optimizing compilers can be written more easily with good tools for debugging optimized code.
- The compiler may not have the ability to generate unoptimized code.
- The programmer may have mistakenly supplied explicit assertions, directives, or options that caused the optimizing compiler to generate incorrect code.
- Optimizing compilers may have bugs.

A Short Primer on Debugger Internals

When using a debugger, the user typically wants to be able to stop the program at various times during execution, print the value of a program symbol, follow the program flow of control by stepping through the source lines in a function, or examine the program execution stack to determine how execution ended up at a particular location.

To enable the user to perform many of these operations, a debugger must have access to a lot of information, much of it related to program blocks, symbols, data types, and source lines. Many debuggers read this information from the object code file and store it in an internal symbol table for easy lookup and traversal.

A debugger uses the information in the symbol table to translate a symbolic location specified by the user into a virtual address. Once a debugger has a virtual address for a program location, it can set a program monitor (such as a breakpoint, trace, or watchpoint) at that location. Once the virtual address for a program symbol is known, a debugger can find its value and display the value according to the symbol's type.

To determine the type and value of a language expression specified by the user, which may range from a simple variable reference to a complex expression, a debugger needs some way to determine if the expression is correct. One way to implement this functionality is with a parser that translates the expression into some sort of abstract syntax or intermediate language tree made up of operator nodes and operand leaves. This tree is then evaluated to determine the type and result value of the expression.

The contents of the symbol table are important to a debugger, but it also needs access to the address space of the target program. On many UNIX systems, debuggers control the target program with the ptrace() system call. With ptrace() a debugger can read from and write to the program's address space, execute the program for one or many instructions, and send a signal to the program. For example, once a debugger has translated a program location into a virtual address, it uses ptrace() to write a special instruction into the instruction stream of the target program. Execution of the special instruction causes the target program to stop and the debugger to regain control of the target program.

†

Wavelet Analysis: Theory and Applications

Wavelet analysis has attracted attention for its ability to analyze rapidly changing transient signals. Any application using the Fourier transform can be formulated using wavelets to provide more accurately localized temporal and frequency information. This paper gives an overview of wavelet analysis and describes a software toolbox created by HP Laboratories Japan to aid in the development of wavelet applications.

by Daniel T.L. Lee and Akio Yamamoto

Wavelet analysis (also called wavelet theory, or just wavelets) has attracted much attention recently in signal processing. It has been successfully applied in many applications such as transient signal analysis, image analysis, communications systems, and other signal processing applications. It is not a new theory in the sense that many of the ideas and techniques involved in wavelets (subband coding, quadrature mirror filters, etc.) were developed independently in various signal processing applications and have been known for some time. What is new is the development of recent results on the mathematical foundations of wavelets that provide a unified framework for the subject.

Within this framework a common link is established between the many diversified problems that are of interest to different fields, including electrical engineering (signal processing, data compression), mathematical analysis (harmonic analysis, operator theory), and physics (fractals, quantum field theory). Wavelet theory has become an active area of research in these fields. There are opportunities for further development of both the mathematical understanding of wavelets and a wide range of applications in science and engineering.

Like Fourier analysis, wavelet analysis deals with expansion of functions in terms of a set of basis functions. Unlike Fourier analysis, wavelet analysis expands functions not in terms of trigonometric polynomials but in terms of *wavelets,* which are generated in the form of translations and dilations of a fixed function called the *mother wavelet.* The wavelets obtained in this way have special scaling properties. They are localized in time and frequency, permitting a closer connection between the function being represented and their coefficients. Greater numerical stability in reconstruction and manipulation is ensured.

The objective of wavelet analysis is to define these powerful wavelet basis functions and find efficient methods for their computation. It can be shown that every application using the fast Fourier transform (FFT) can be formulated using wavelets to provide more localized temporal (or spatial) and frequency information. Thus, instead of a frequency spectrum, for example, one gets a wavelet spectrum. In signal processing, wavelets are very useful for processing nonstationary signals.

Wavelets have created much excitement in the mathematics community (perhaps more so than in engineering) because the mathematical development has followed a very interesting path. The recent developments can be viewed as resolving some of the difficulties inherent in Fourier analysis. For example, a typical question is how to relate the Fourier coefficients to the global or local behavior of a function. The development of wavelet analysis can be considered an outgrowth of the Littlewood-Paley theory[1] (first published in 1931), which sought a new approach to answer some of these difficulties. Again, it is the unifying framework made possible by recent results in wavelet theory related to problems of harmonic analysis (also to similar problems in operator theory called the Calderón-Zygmund theory[1]) that has generated much of the excitement.

In electrical engineering, there have been independent developments in the analysis of nonstationary signals, specifically in the form of the short-term Fourier transform, a variation of which called the Gabor transform was first published in 1946.[2] A major advance in wavelet theory was the discovery of smooth mother wavelets whose set of discrete translations and dilations forms an orthonormal basis for $L^2(\mathbf{R})$, where **R** is the real numbers and L^2 is the set of all functions, f, that have bounded energy, that is, functions for which

$$\int_{-\infty}^{\infty} |f(t)|^2 dt < \infty .$$

This is a main difference from the Gabor transform. In the Gabor case, no orthonormal basis can be generated from smooth wavelets. Thus the unifying framework brought about a better understanding and a new approach that overcomes the difficulties in the short-term Fourier transform methods.

In the next section we give an overview of the main features of wavelet analysis and then turn to a software toolbox that

HP Laboratories Japan has developed to help in the development of wavelet applications.

For an excellent tutorial introduction to the subject, see Rioul and Vetterli[2] and the references therein (it lists 106 references). Daubechies' book[1] is a standard reference on the subject.

Fundamentals of Wavelet Theory

This section gives a quick overview of the main formulas.

The Analyzing Wavelet

Consider a complex-valued function ψ satisfying the following conditions:

$$\int_{-\infty}^{\infty} |\psi(t)|^2 dt < \infty \tag{1}$$

$$c_\psi = 2\pi \int_{-\infty}^{\infty} \frac{|\Psi(\omega)|^2}{|\omega|} d\omega < \infty, \tag{2}$$

where Ψ is the Fourier transform of ψ. The first condition implies finite energy of the function ψ, and the second condition, the admissibility condition, implies that if $\Psi(\omega)$ is smooth then $\Psi\ (0) = 0$.

The function ψ is the mother wavelet.

Continuous Wavelet Transform

If ψ satisfies the conditions described above, then the *wavelet transform* of a real signal s(t) with respect to the wavelet function $\psi(t)$ is defined as:

$$S(b, a) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} \psi'\left(\frac{t - b}{a}\right) s(t) dt, \tag{3}$$

where ψ' denotes the complex conjugate of ψ, and this is defined on the open (b,a) half-plane ($b \in \mathbf{R}$, $a > 0$). The parameter b corresponds to the time shift and the parameter a corresponds to the scale of the analyzing wavelet.

If we define $\psi_{a,b}(t)$ as

$$\psi_{a,b}(t) = a^{-1/2}\psi\left(\frac{t - b}{a}\right), \tag{4}$$

which means rescaling by a and shifting by b, then equation 3 can be written as a scalar or inner product of the real signal s(t) with the function $\psi_{a,b}(t)$:

$$S(b, a) = \int_{-\infty}^{\infty} \psi'_{a,b}(t) s(t) dt. \tag{5}$$

When function $\psi(t)$ satisfies the admissibility condition, equation 2, the original signal s(t) can be obtained from the wavelet transform S(b,a) by the following inverse formula:

$$s(t) = \frac{1}{c_\psi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} S(b, a)\psi_{a,b}(t) \frac{dadb}{a^2}. \tag{6}$$

Discrete Wavelet Transform

In the discrete domain, the scale and shift parameters are discretized as $a = a_0^m$ and $b = nb_0$, and the analyzing wavelets are also discretized as follows:

$$\psi_{m,n}(t) = a_0^{-m/2}\psi\left(\frac{t - nb_0}{a_0^m}\right), \tag{7}$$

where m and n are integer values. The discrete wavelet transform and its inverse transform are defined as follows:

$$S_{m,n} = \int_{-\infty}^{\infty} \psi'_{m,n}(t) s(t) dt, \tag{8}$$

$$s(t) = k_\psi \sum_m \sum_n S_{m,n}\psi_{m,n}(t), \tag{9}$$

where k_ψ is a constant value for normalization.

The function $\psi_{m,n}(t)$ provides sampling points on the scale-time plane: linear sampling in the time (b-axis) direction but logarithmic in the scale (a-axis) direction.

The most common situation is that a_0 is chosen as:

$$a_0 = 2^{1/v}, \tag{10}$$

where v is an integer value, and that v pieces of $\psi_{m,n}(t)$ are processed as one group, which is called a *voice.* The integer v is the number of voices per octave; it defines a well-tempered scale in the sense of music. This is analogous to the use of a set of narrowband filters in conventional Fourier analysis.

Wavelet analysis is not limited to dyadic scale analysis. By using an appropriate number of voices per octave, wavelet analysis can effectively perform the 1/3-octave, 1/6-octave, or 1/12-octave analyses that are used in acoustics.

The main focus of current research is on finding optimal wavelet basis functions and efficient algorithms for computing the corresponding wavelet transforms. The wavelet basis function can be implemented as an FIR (finite impulse response) filter or an IIR (infinite impulse response) filter depending on the particular properties needed.

Graphical Representation

This section describes how to display complex-valued functions such as equations 3 and 8 so that useful information about the signal s(t) can be highlighted. There are two aspects to consider.

The open (b,a) half-plane on which the wavelet transform is defined can be mapped onto the full plane (b,–log(a)). This representation is indispensable if we want to display, in a single picture, information with a wide range of scale parameters. For example, for sound signals in the audible range, a spread of ten octaves is common. A disadvantage of this representation, on the other hand, is that straight lines on the open (b,a) half-plane become exponential curves in the logarithmic representation.

Expressions 3 and 8 depend on the choice of the analyzing wavelet ψ. To obtain full quantitative information about the signal s(t) from its transform S(b,a), we need to know the analyzing wavelet ψ. There are, however, many features of the signal that are independent of the choice of ψ. Such features involve the phase of the complex-valued functions. Therefore, it is useful to represent separately the modulus and the phase of the complex-valued function S(b,a) to be described.

Shown in Figs. 1 and 2 is an example of the wavelet transform of a localized pulse that approximates a delta function.

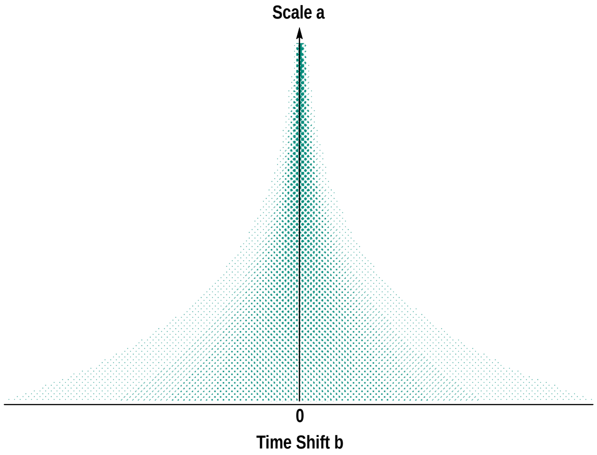


Fig. 1. Magnitude of the wavelet transform of a delta function.

The horizontal axis is time in both the magnitude picture, Fig. 1, and the phase picture, Fig. 2, and the vertical axis is scale, with small scale at the top.

In Fig. 1, the magnitude increases toward the top of the picture. The modulus or magnitude, $|S(b,a)|$, is converted to grayscale and is normalized to its maximum, that is, the plot shows x, where:

$$x = \frac{|S|}{|S_{max}|} \le 1. \tag{11}$$

The phase of S(b,a) is given by a grayscale picture in which a phase of 0 corresponds to white and a phase of 2π to black. This convention is quite useful in interpreting the resulting picture. When the phase reaches 2π, it is wrapped around to the value 0. The lines where the density drops abruptly to zero are clearly visible on the picture and play an important role in the interpretation as a visible line of constant phase. In Fig. 2, one can see the lines of constant phase pointing to the location of the delta function.

Examples of Wavelet Functions

Haar Wavelet. The Haar wavelet is the simplest kind of wavelet function. Suppose that $\phi(t)$ is a box function satisfying the following:

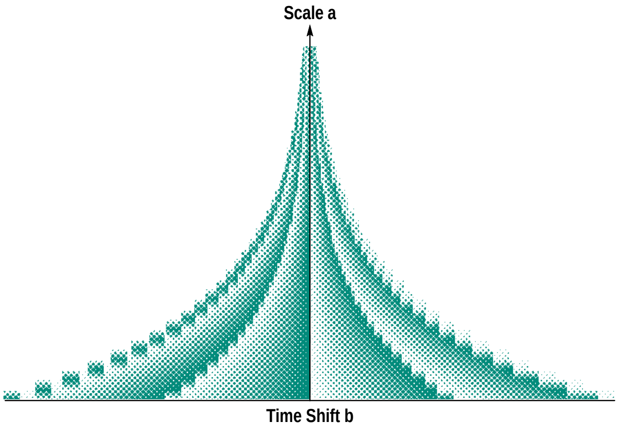


Fig. 2. Phase of the wavelet transform of a delta function.

$$\phi(t) = \begin{cases} 1 & \text{if } 0 \le t \le 1 \\ 0 & \text{otherwise.} \end{cases}$$

If we define the function $\psi(t)$ as $\psi(t) = \phi(2t) - \phi(2t{-}1)$, we can obtain the following function:

$$\psi(t) = \begin{cases} 1 & \text{if } 0 < t \le 1/2 \\ -1 & \text{if } 1/2 < t \le 1 \\ 0 & \text{otherwise.} \end{cases}$$

The function $\phi(t)$ is the Haar scaling function, and $\psi(t)$ is the Haar wavelet. This function is orthogonal to its own translations and dilations, that is, the family

$$\psi_{m,n}(t) = 2^{-m/2}\,\psi(2^{-m}t - n), \qquad m,n \in \mathbf{Z}, \tag{14}$$

where **Z** is the real integers, constitutes an orthonormal basis for $L^2(\mathbf{R})$. Historically the Haar function was the original wavelet. This wavelet is not continuous, and its Fourier transform $\Psi(\omega)$ decays only like $|\omega|^{-1}$, corresponding to bad frequency localization.

Meyer Wavelet. Yves Meyer constructed a smooth orthonormal wavelet basis as follows. First of all, define the Fourier transform $\Phi(\omega)$ of a scaling function $\phi(t)$ as:

$$\Phi(\omega) = \begin{cases} 1 & \text{if } |\omega| \le \frac{2}{3}\pi \\ \cos\left[\frac{\pi}{2}\nu\left(\frac{3}{4\pi}|\omega| - 1\right)\right] & \text{if } \frac{2}{3}\pi \le |\omega| \le \frac{4}{3} \\ 0 & \text{otherwise,} \end{cases} \tag{15}$$

where ν is a smooth function satisfying the following conditions:

$$\nu(t) = \begin{cases} 0 & \text{if } t \le 0 \\ 1 & \text{if } t \ge 1 \end{cases} \tag{16}$$

with the additional property

$$\nu(t) + \nu(1{-}t) = 1. \tag{17}$$

This function Φ is plotted in Fig. 3.

In this case, the wavelet function ψ can be found easily from Φ. First, we find the Fourier transform of ψ:

$$\Psi(\omega) = e^{i\omega/2}\sum_{l \in \mathbf{Z}} \Phi(\omega + 2\pi(2l + 1))\Phi(\omega/2) \tag{18}$$

$$= e^{i\omega/2}[\Phi(\omega + 2\pi) + \Phi(\omega - 2\pi)]\Phi(\omega/2). \tag{19}$$

The function Ψ is plotted in Fig. 4.

Now since Ψ is compactly supported (its duration is finite and nonzero) and $\Psi \in C_k$ where k is arbitrary and may be ∞ (i.e., Ψ has at least k derivatives), ψ can be obtained by taking the inverse Fourier transform. Fig. 5 shows a graph of the Meyer wavelet $\psi(t) \in C^4$.

Morlet Wavelet. This particular function was most often used by R. Kronland-Martinet and J. Morlet. Its Fourier transform is a shifted Gaussian, adjusted slightly so that $\Psi(0) = 0$:

$$\Psi(\omega) = e^{-(\omega-\omega_0)^2/2} - e^{-\omega^2/2}e^{-\omega_0^2/2} \tag{20}$$

$$\psi(t) = \left(e^{-i\omega_0 t} - e^{-\omega^2/2}\right)e^{-t^2/2}. \tag{21}$$

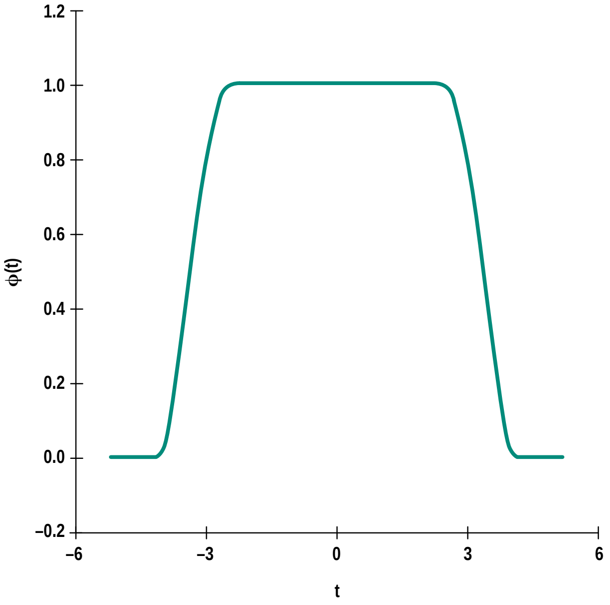


Fig. 3. Fourier transform of the scaling function for the Meyer basis.

Often ω_0 is chosen so that the ratio of the highest maximum of ψ to the second highest maximum is approximately 1/2, that is,

$$\omega_0 = (2/\ln 2)^{1/2} \simeq 5.3364\ldots \quad (22)$$

In practice one often takes $\omega_0 = 5$. For this value of ω_0, the second term in equation 20 is so small that it can be neglected in practice. Consequently, the Morlet wavelet can be considered as a modulated Gaussian waveform. Its real and imaginary parts for $\omega_0 = 5$ are shown in Figs. 6 and 7, respectively.

The Morlet wavelet is complex, even though most applications in which it is used involve only real signals. The wavelet transform of a real signal with this complex wavelet is plotted in modulus-phase form, that is, one plots $|\langle s, \psi_{m,n}\rangle|$ and $\tan^{-1}[\mathrm{Im}\langle s, \psi_{m,n}\rangle/\mathrm{Re}\langle s, \psi_{m,n}\rangle]$, where the brackets indicate the scalar or inner product of the signal waveform s with the basis function $\psi_{m,n}$, that is,

$$\langle s, \psi_{m,n}\rangle = \int_{-\infty}^{\infty} s(t)\psi'_{m,n}(t)dt.$$

The phase plot is particularly suited for the detection of singularities.

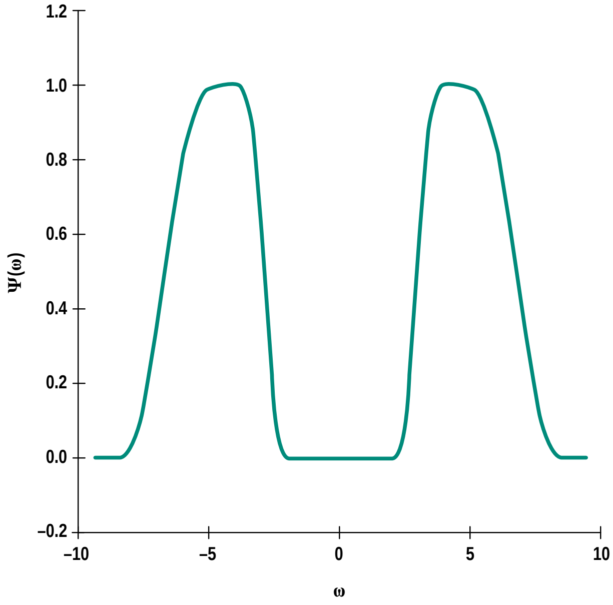


Fig. 4. Fourier transform of the Meyer wavelet.

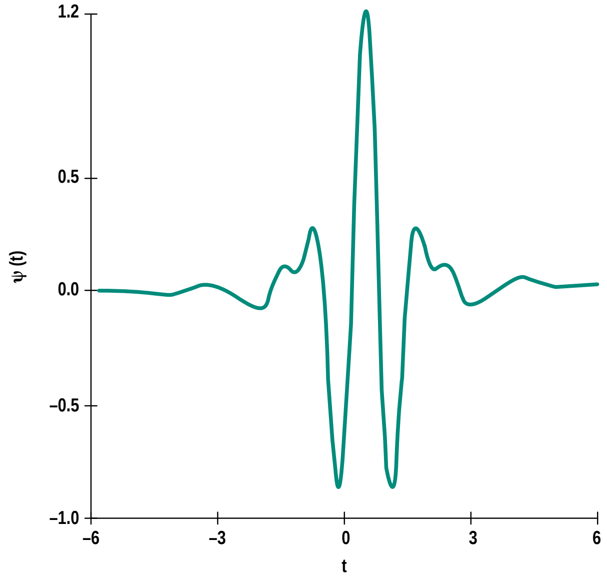


Fig. 5. The Meyer wavelet.

Daubechies Wavelet. Except for the Haar basis, all of the examples of orthonormal wavelet bases consist of infinitely supported functions. Ingrid Daubechies constructed an orthonormal wavelet in which ψ is compactly supported. The way to ensure compact support for the wavelet ψ is to choose a scaling function ϕ with compact support.

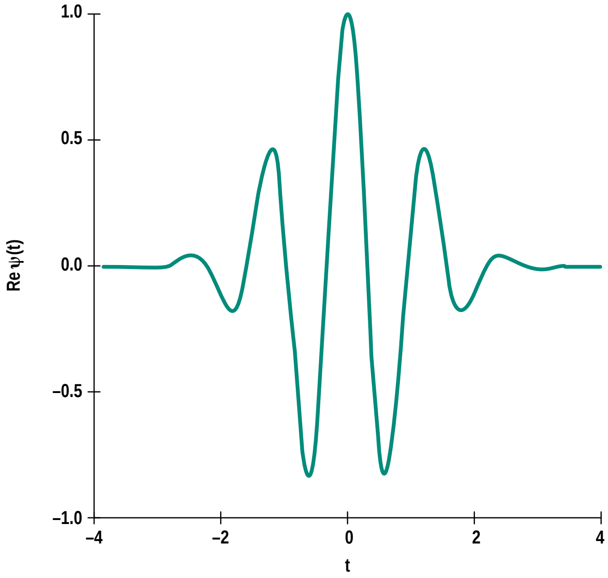


Fig. 6. Real part of the Morlet wavelet for $\omega_0 = 5$.

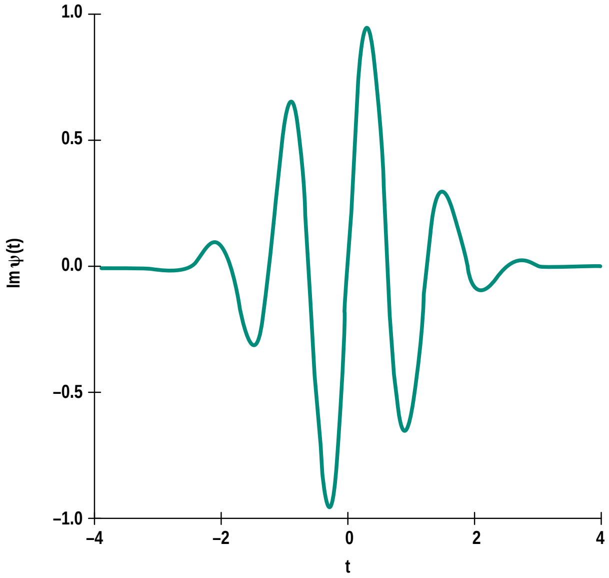


Fig. 7. Imaginary part of the Morlet wavelet for $\omega_0 = 5$.

First of all, find a progression $\{\alpha_k; k \in \mathbf{Z}\}$ satisfying the following four conditions for all integer $N \ge 2$:

$$\alpha_k = 0 \quad \text{if } k < 0 \text{ or } k > 2N \tag{23}$$

$$\sum_{k=-\infty}^{\infty} \alpha_k \alpha_{k+2m} = \delta_{0m} \quad \text{for all integer m} \tag{24}$$

$$\sum_{k=-\infty}^{\infty} \alpha_k = \sqrt{2} \tag{25}$$

$$\sum_{k=-\infty}^{\infty} \beta^k k^m = 0, \quad 0 \le m \le N-1, \tag{26}$$

where $\beta_k = (-1)^k \alpha_{-k+1}$.

If N = 1, then $\alpha_0 = \alpha_1 = 1$, corresponding to the Haar basis.

We can find a compactly supported scaling function $\phi(t)$ from the above progression $\{\alpha_k\}$. The function $\phi(t)$ is one solution of a functional equation:

$$\phi(t) = \sum_{k=-\infty}^{\infty} \alpha_k \sqrt{2}\, \phi(2t - k). \tag{27}$$

It is continuous and compactly supported and satisfies $\int \phi(t)dt = 1$ for integer N and the corresponding progression $\{\alpha_k\}$. The support of $\phi(t)$ is [0,2N–1].

Furthermore, if β_k is defined as the condition 26, the function $\psi(t)$ satisfying a functional equation

$$\psi(t) = \sum_{k=-\infty}^{\infty} \beta_k \sqrt{2}\, \phi(2t - k) \tag{28}$$

is compactly supported and fulfills the following:

- $\int \psi(t)t^m dt = 0$ for all integers $0 \le m \le N-1$.
- $\phi(t), \psi(t) \in C^{\lambda(N)}$ for Holder spaces $C^{\lambda(N)}$, where $\lambda(N)$ is an integer parameter and the elements of $C^{\lambda(N)}$ are functions that have $\lambda(N)$ derivatives.

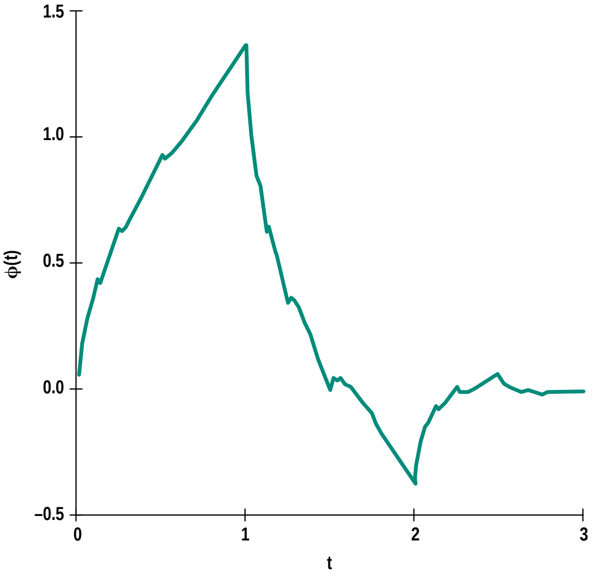


Fig. 8. The Daubechies scaling function for N = 2.

Figs. 8 and 9 show graphs of the Daubechies scaling function ϕ and the corresponding wavelet ψ for the value of N = 2.

Software Tools: Khoros System

The wavelet analysis software developed by HP Laboratories Japan is implemented as a toolbox in the Khoros system. The Khoros system is an integrated software development environment for information processing and visualization, based on the X Window System. It is distributed in the public domain and has been ported to the HP-UX* operating system.[3]

Khoros components include a visual programming language, code generators for extending the visual language and adding new application packages to the system, an interactive user interface editor, an interactive image display package, an

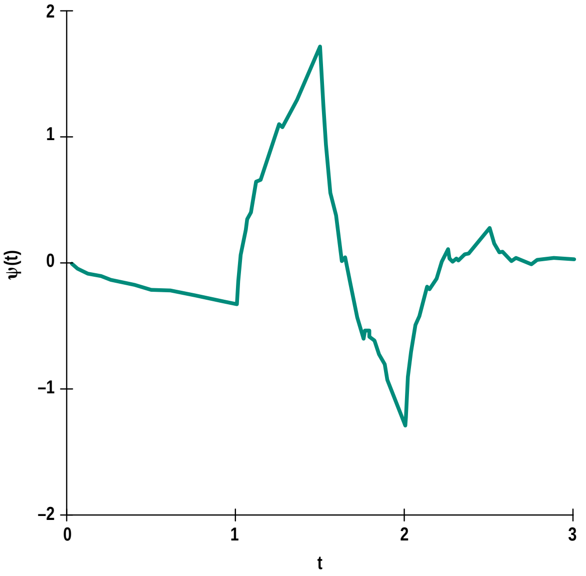


Fig. 9. The Daubechies wavelet for N = 2.

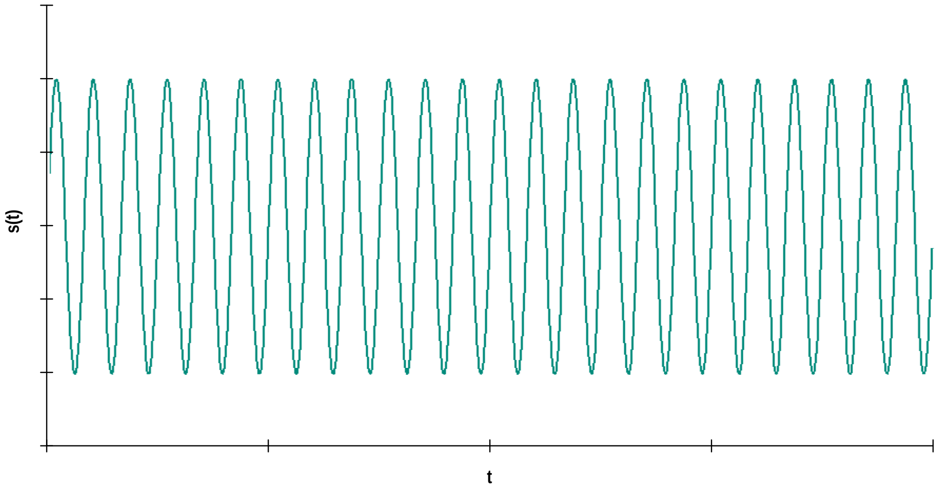


Fig. 10. Sinusoid with constant frequency.

extensive library of image processing, numerical analysis, and signal processing routines, and 2D/3D plotting packages.

The Khoros system also supports the toolbox update method for new routines created by another person or developed on another machine. A toolbox contributed by HP Laboratories Japan, the HPLJ Toolbox, contains wavelet application development tools, image data compression utilities, and other utilities.

Wavelet Analysis Examples

The following examples illustrate the advantages of the time-scale resolution properties of the wavelet transform and a related concept, the *chirplet transform,* for the analysis of various input signals, including a delta function, a step or box function, a differentially discontinuous function, a ramp function, sinusoidal functions, a chirp signal, and a sum of gliding tones.

Wavelet Analysis

This section gives several application examples of wavelet-based signal analysis, including both stationary and nonstationary signal analysis. These results were obtained with a Morlet wavelet, that is, a complex sinusoid windowed with a Gaussian envelope, expressed as follows:

$$\psi(t) = e^{ict}\exp\left(-\frac{1}{2}t^2\right), \qquad (29)$$

where c is a constant value of 5 so that the function $\psi(t)$ satisfies the admissibility condition.

As for the number of voices discussed earlier, we use v = 6 in the following three examples of synthesized data analysis.

Example 1. The first example gives the analysis of two sinusoids. Fig. 10 shows a sinusoid with a single constant frequency, and Figs. 11 and 12 represent its wavelet transform. The horizontal axis is in time in both the magnitude picture, Fig. 11, and the phase picture, Fig. 12. The vertical axis is scale, small scale at the top. Certain features of the signal are evident: horizontal strips of constant magnitude, and vertical lines in step with the phase of the signal.

Fig. 13 shows a sinusoid with linearly increasing frequency. The wavelet transform analysis results for this signal are shown in Figs. 14 and 15. Clearly visible is the upward slope corresponding to the increase of frequency.

Example 2. The second example is the analysis of the superposition of two delta functions and two sinusoids, as shown in Fig. 16. One delta function is larger than the sinusoidal signals and is visible in Fig. 16, but the other is much smaller and does not appear.

Figs. 17 and 18 show the wavelet transform representations. We can easily see the two peaks at smaller scale that correspond to the discontinuities contained in the input signal.

Example 3. This example shows the analysis of a sum of three sinusoids with different starting times. The input signal shown in Fig. 19 is not discontinuous, but its first derivative is.

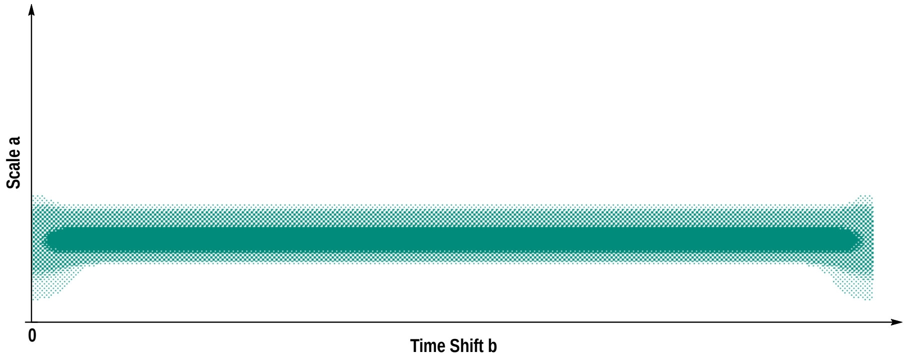


Fig. 11. Magnitude of the wavelet transform of a constant-frequency sinusoid.

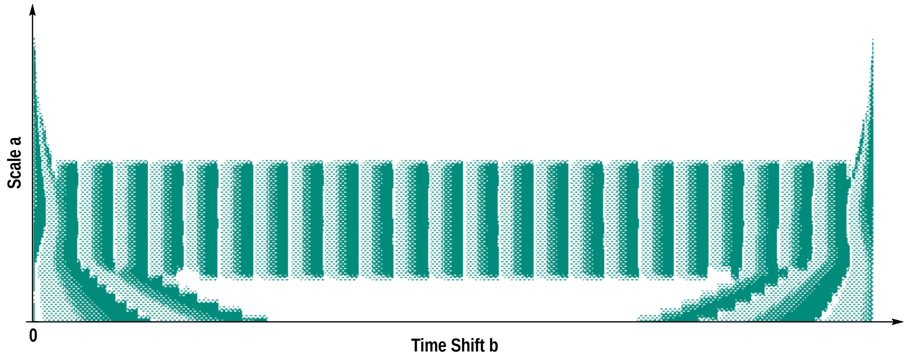


Fig. 12. Phase of the wavelet transform of a constant-frequency sinusoid.

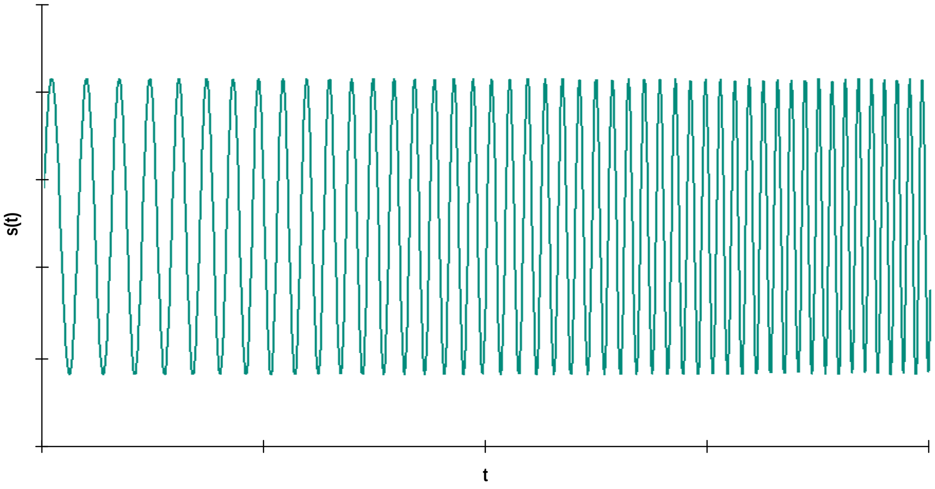


Fig. 13. Sinusoid with linearly increasing frequency.

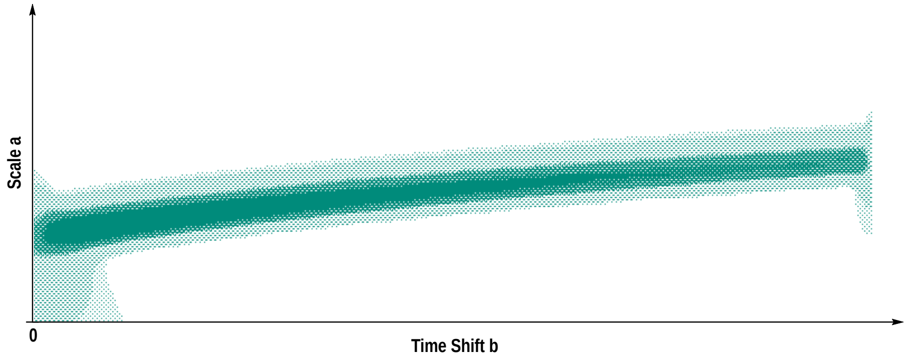


Fig. 14. Magnitude of the wavelet transform of a sinusoid with linearly increasing frequency.

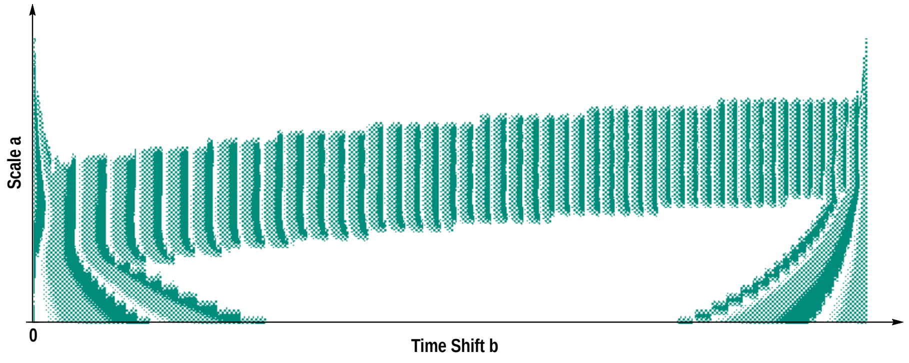


Fig. 15. Phase of the wavelet transform of a sinusoid with linearly increasing frequency.

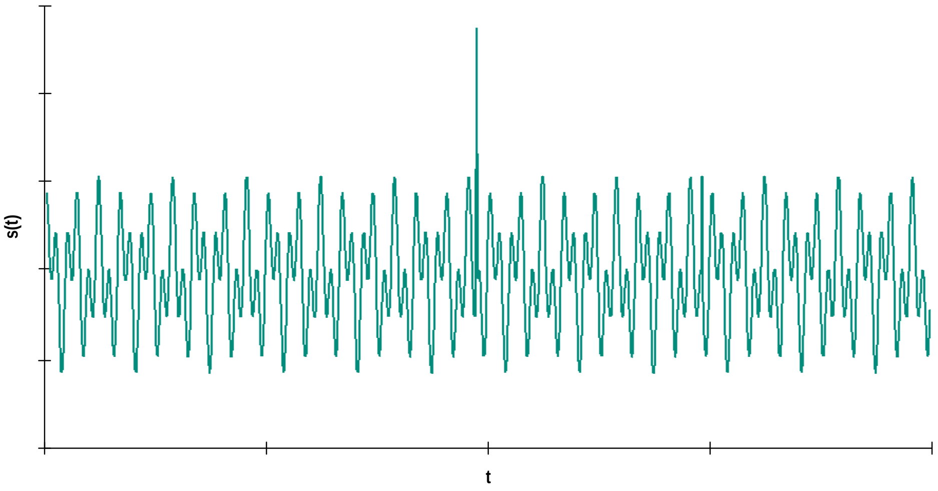


Fig. 16. Two delta functions and two sinusoids.

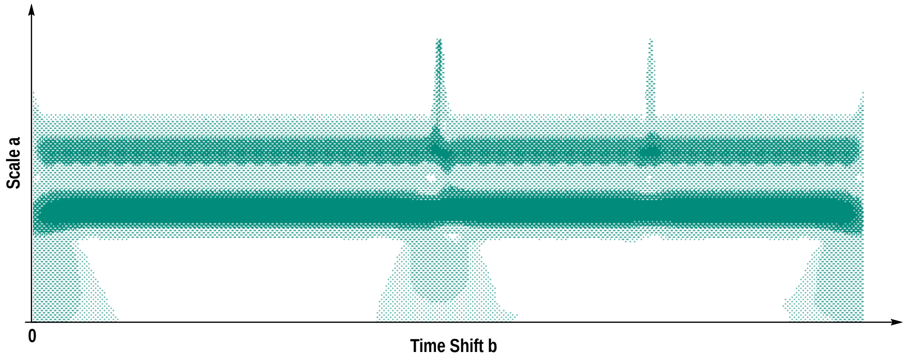


Fig. 17. Magnitude of the wavelet transform of the sum of two delta functions and two sinusoids.

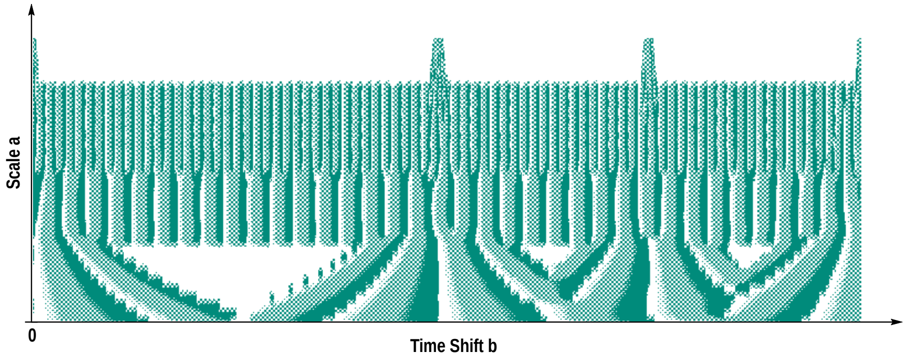


Fig. 18. Phase of the wavelet transform of the sum of two delta functions and two sinusoids.

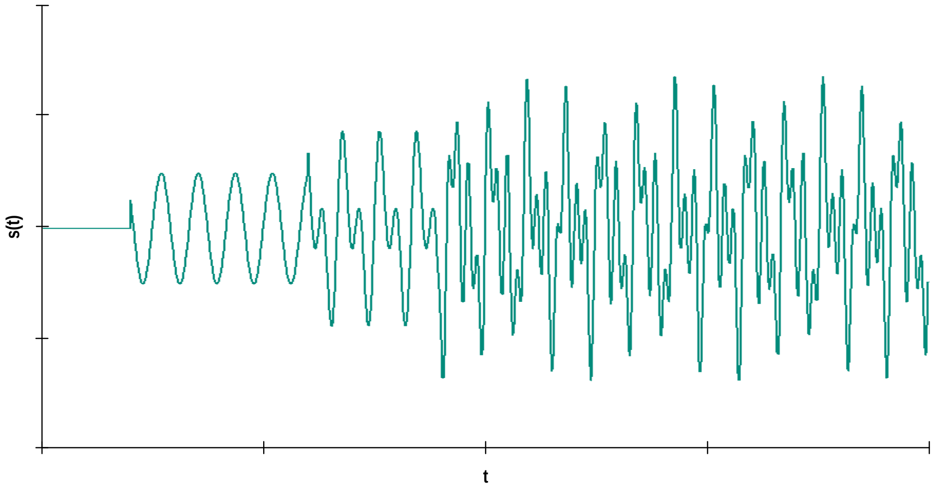


Fig. 19. Three sinusoids with different starting times.

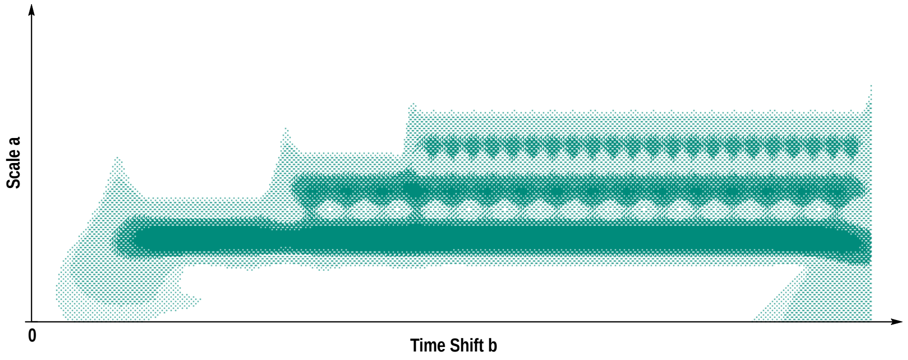


Fig. 20. Magnitude of the wavelet transform of three sinusoids with different starting times.

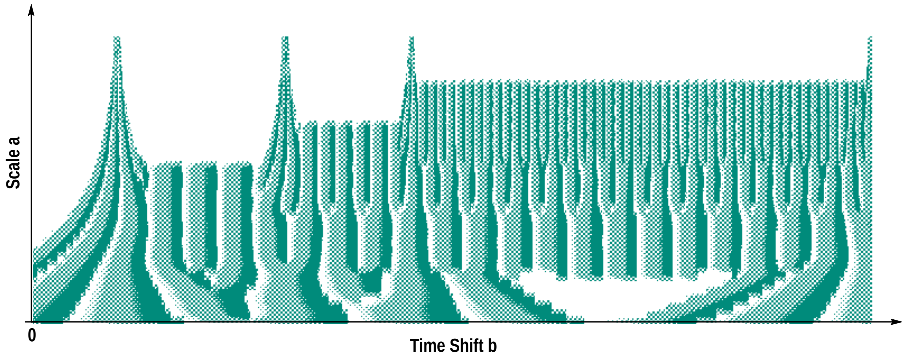


Fig. 21. Phase of the wavelet transform of three sinusoids with different starting times.

Both the frequencies and the beginnings of the components are very clearly visible in the wavelet transform pictures,

Figs. 20 and 21. A low-frequency sinusoid starts first, followed by a medium-frequency and a high-frequency sinusoid.

Real Data Analysis. This section shows the results of real data analysis. This data, provided by the Lake Stevens Instrument Division, is the transmitter turn-on data of a dual-band transceiver that was taken at a center frequency of 146.52 MHz with the measurement span set to 39.0625 kHz. In other words, the data is filtered to approximately a 40-kHz bandwidth. The time interval between points is 20 μs.

The input signal is plotted in Fig. 22. In this case, the transform was performed for the value of v = 12, and the magnitude and phase of the wavelet transform are shown in Figs. 23 and 24, respectively.

Chirplet Analysis

The wavelet transform has the effect of dissecting the time-scale plane into time-invariant cells with an aspect ratio dependent on the scale parameter. This property is important in spectral processing of signals but does not affect dynamic spectrum displays where time t is advanced by small increments relative to the cell width.

The representation of signals may benefit if the cell shape is not held time-invariant throughout. This time-dependent adjustment can be performed adaptively. Such a technique, called the *chirplet transform,* has been proposed. It uses oblique cells adapted to the local structure, permitting separation of the signal components.

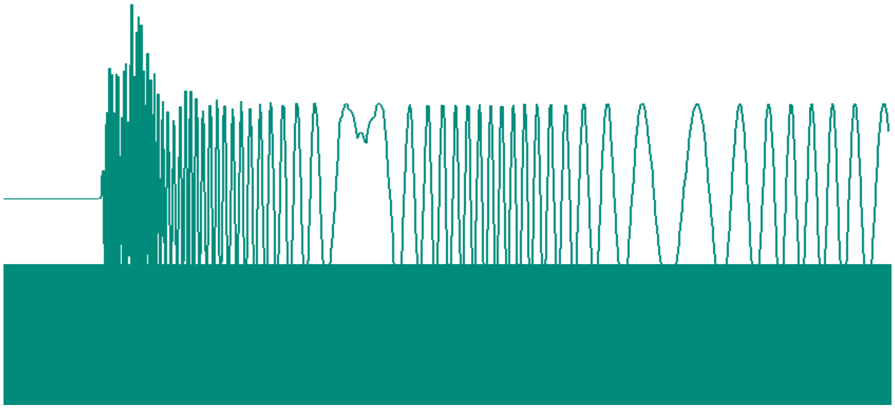


Fig. 22. Transmitter turn-on data.

Approaches to Verifying Operational Test Release Vectors

Five techniques are employed to minimize the time to develop the test vectors used to test manufactured parts on an IC component tester.

by Joy Xiao Han

In today's competitive computer industry, the ability to accelerate the development process from design to manufactured part in a timely manner is an important factor. At HP's Chelmsford systems laboratory one of the things we do is design and develop chips for HP 9000 Series 800 workstations. Among the activities performed during this development is the generation of a series of test vectors called operational test release (OTR) vectors, which are used on component testers to verify the correctness of the manufactured parts.

It typically takes six months to generate and verify operational test release vectors. With the techniques described in this article we have been able to reduce the time spent creating these vectors to four months.

Test Vector Development

The final product of our test vector development process is a set of vectors (also called test patterns) that can be loaded onto a tester to verify manufactured parts. Physical defects that appear on manufactured parts can be so varied that it is often impractical to try to detect them directly. Instead, automatic test generation, waveform creation, and verification tools are employed to deal with a logical model of a physical defect, which is known as a fault. The most widely used fault model is the stuck-at fault in which the input or the output of a logic element is stuck at logic 0 or 1. For example, an open trace, assuming positive logic, might exhibit itself as stuck at logic 0.

Each vector that tests a typical block of application logic* has at least two parts. One part is made up of data and/or instructions which are applied to the input of the chip. The other part, called "expected results," is used for comparison with the actual output of the application logic to detect any faults.

The first steps of our test vector development process are shown in Fig. 1. We start by using a program called ATPG (automatic test pattern generator) from Crosscheck Corporation to produce a file containing the test input patterns, the fault-free simulation output patterns (expected results), and the scan-in and scan-out** patterns for the test logic. ATPG uses a circuit model of the chip to determine the content of the patterns produced.

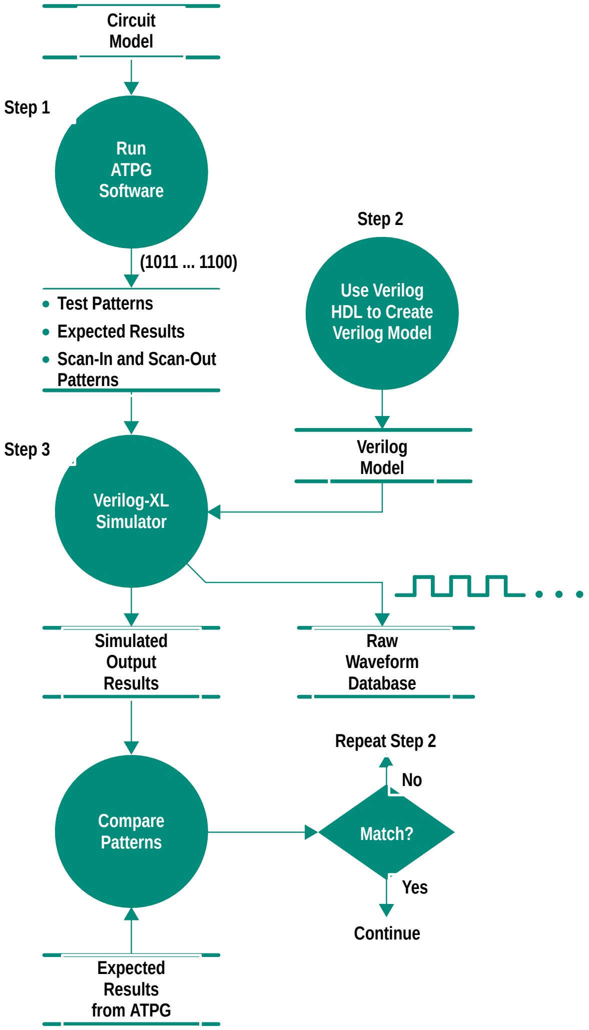


Fig. 1. The steps used to create a raw waveform data base.

The next step in our process is to use the Verilog hardware description language (Verilog HDL) to create a behavioral and structural model of the target hardware. The instructions in the program are structured to test the application logic via special test pins collectively called a test access port (TAP). The TAP provides access to test logic circuits that are built into a component to test the component itself and the interconnections between components. The TAP also provides access to circuits that allow control and observation of the

* We use the term application logic in this paper to refer to the logic on the chip that performs the intended functions of the component. The other logic on the chip is called test logic, which is included on the chip for testability.

** Scan-out patterns are also treated as expected results.

Overview of the Test Access Port

Since the emergence of surface mounted devices a great deal of concern and discussion has gone into determining how to test boards crammed with these high-density devices. In 1990 these concerns resulted in ANSI/IEEE Standard 1149.1-1990, Standard Access Port and Boundary-Scan Architecture. This standard defines test logic that can be included on an integrated circuit to provide standardized approaches to testing the component itself or the interconnections between components on a printed circuit board. The standard also allows for observing or controlling the behavior of a component during its normal operation. The test logic allows test instructions and test data to be fed to a component, and upon execution of an instruction, allows the results to be read out and observed. All instructions, test data, and results are communicated in serial format.

The test logic defined by the standard consists of a chain of boundary-scan cells and test support logic, which are accessed through the TAP inputs (see Fig. 1). A boundary-scan cell is a shift-register stage that is connected between each input or output pin on an IC and the application logic to which each pin is normally connected (see Fig. 2). The scan cell has two states of operation. One state allows a sequence of bits representing data and instructions to be shifted (scanned-in) into a chain of scan cells, resulting in latching each cell to the desired value. The scan-in and scan-out lines shown in Fig. 2 carry the bits from one cell to another. The logic specified in the standard is designed so that the serial movement of instruction data is not apparent to the circuits whose operation is controlled by the instruction.

The other state of operation for the scan cells involves testing the application logic. The test operation involves either receiving test data from the application logic via the signal-in line and then latching the output, or shifting test data into the application logic via the signal-out line. The test logic is specified such that the movement of test data has no effect on the instruction present in the test circuitry.

After the test state is done the scan mode can be invoked again to shift out the latched test results for comparison with the expected results.

The clock, shift, and mode lines shown in Fig. 2 are controlled by the TAP signals (described below). The TAP lines are responsible for sending the proper signal sequences to control the scanning or testing states. In addition, the mode line is controlled according to the type of pin it is connected to (e.g., input, output, bidirectional, tristate, etc.).

The IEEE standard defines a minimum of three input connections and one output connection (see Fig 1). An optional fourth input (TRST*) provides for asynchronous initialization of the test logic circuitry defined in the standard.

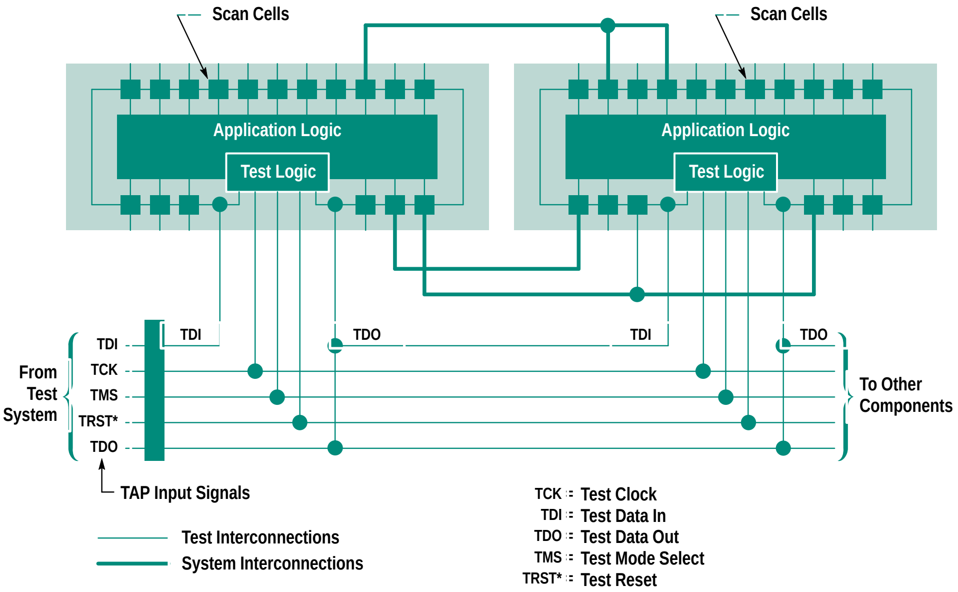


Fig. 1. A simplified block diagram of the test logic defined in ANSI/IEEE Standard 1149.1-1990 surrounding application logic.

application circuits during their normal operation. The specification for the TAP logic is given in IEEE Standard 1149.1-1990, and a brief overview is provided above. Also included in the Verilog HDL model are calls to a utility called $tds_monitor()* which associates timing information with the bit patterns sent to the device under test. These calls will create a raw waveform database containing timing and pattern information. It is called a raw waveform database because during simulation runs every change on the component's pins is included in the database. One of the techniques described below explains how this data is manipulated to produce a more refined waveform database.

The third step shown in Fig. 1 involves running the Verilog-XL logic simulator using the Verilog HDL model created in step 2 and the patterns created in step 1 as inputs. A waveform database and an output pattern file are created from this simulation. The output patterns from the simulation are compared to the expected output patterns generated by the ATPG software. If the patterns don't match, steps 2 and 3 are repeated until they do. If the patterns do match, we move on to prepare the waveform database to become our operational test release vectors.

A great deal of time can be spent going back and forth between steps 2 and 3. The rest of this paper describes some techniques that I have found to be helpful in getting through steps 2 and 3 quickly. These techniques verify that the TAP circuits are functioning properly.

* The $tds_monitor() runs as part of the Verilog-XL simulator.

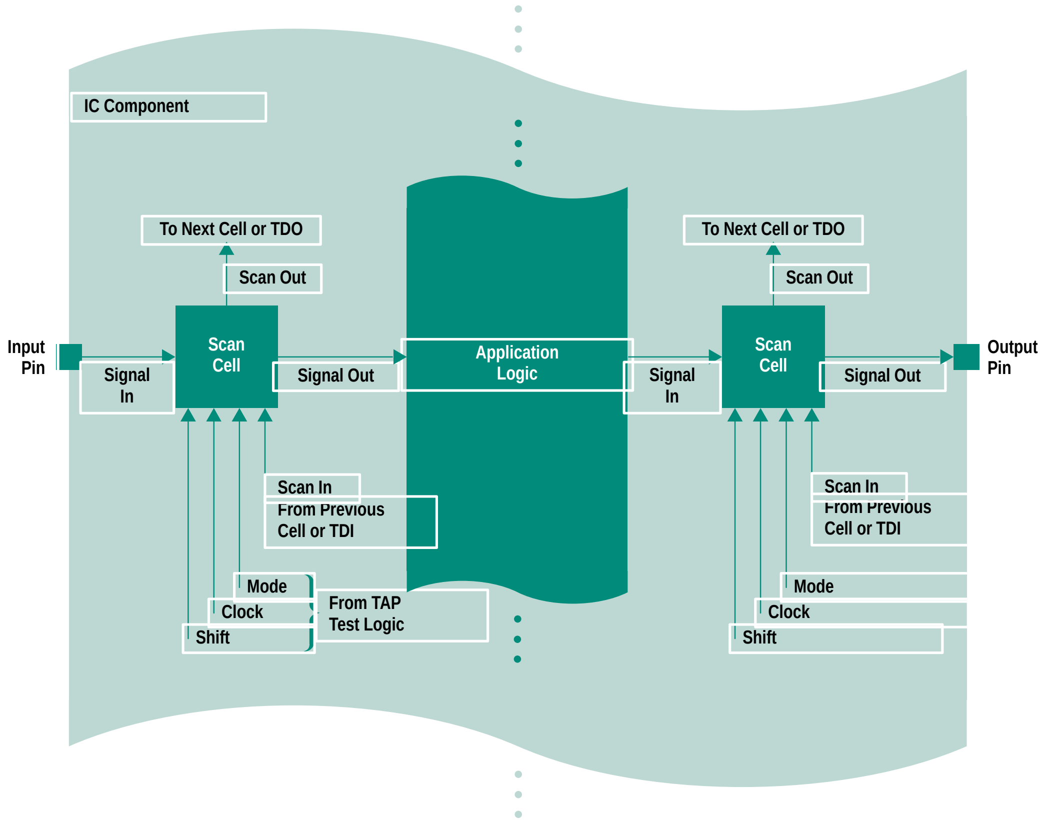


Fig. 2. The location of scan cells in relation chip I/O pins and the application logic. The mode, clock and shift signals are derived from the TAP input signals TMS, TCK, and TRST* respectively. The first scan-in signal and the last scan-out signal correspond to TDI and TDO respectively.

Test Clock. TCK is the test clock input that provides the clock for the test logic. This clock is provided so that scan cells surrounding the application logic can be controlled independently of system clocks. TCK allows shifting of test data concurrently with system operation (allowing online monitoring). It also ensures that test data can be moved to or from a chip without changing the state of the application logic.

Test Mode Select. TMS is the signal used by the TAP controller to control test operations. One use of TMS is to select whether the test circuitry is in the test state or the scan state. To guard against race conditions, the TMS signal like the TDI signal described below must be sampled on the rising edge of TCK.

Test Reset. The optional TRST* signal is included to allow for asynchronous reset of the TAP controller. The reset signal only affects the test logic and has no impact on the application logic.

Test I/O Lines. TDI and TDO are the test data input and output lines respectively. They provide for the serial movement of test data through the circuit. Data presented at TDI is clocked into the selected register on the rising edge of TCK, while output data appearing at TDO is clocked out on the falling edge of TCK. To simplify the operation of components that are compatible with the standard, data must be propagated from TDI to TDO without inversion.

Bibliography

1. *IEEE Standard Test Access Port and Boundary-Scan Architecture,* IEEE Std. 1149.1-1990, IEEE Standards Board, May 1990.

2. R. G. Bennets, *Design of Testable Logic Circuits,* Addison-Wesley, 1984.

3. V. D. Agrawal and S. C. Seth, *Test Generation for VLSI Chips,* IEEE Computer Society Press, 1988.

Technique 1

Check the scan chain* without a system clock. This step is mainly used to make sure that the scan chain on the chip is not broken. This test works by ensuring that whatever value is scanned in (SI in Fig. 2) should be exactly identical to the value scanned out (SQ). The scan chain is advanced by clock pulses. For example, a pulse from CLKA followed by a pulse from CLKB causes the data on SI to be propagated to SQ. SQ turns into the SI for the next scannable flip flop on the scan chain. If the chip passes this test, we know that the scan logic is set properly and that there is continuity in the scan chain.

Technique 2

Check the scan chain with a system clock. This check verifies that the combinational logic in the application logic portion of the chip works. Usually the master clock (MCLK) is opposite in state to the system clock (CLK) (i.e., when CLK is on, MCLK is off and vice versa). The only exception to this behavior occurs when we execute a double-strobe test to check the time margin on the chip.

The opposite clock states are verified by ensuring that the data on the D input in Fig. 2, which is the result of all previous combinational logic, is inverted at MQ when CLK is low. On the other hand, when CLK is turned on, the inverted value of MQ (the exact value of D) appears at Q. This is the same value we can monitor at SQ by advancing the scan chain with pulses from CLKA and CLKB in the correct sequence.

Technique 3

Check the TAP state sequence. Since the TAP logic is basically a state machine its current state is recorded in a mode register. I have found it necessary to pay attention to the

* A scan chain is a shift-register path through a circuit which is typically placed there to improve testability. See “Overview of the Test Access Port” on page 56.

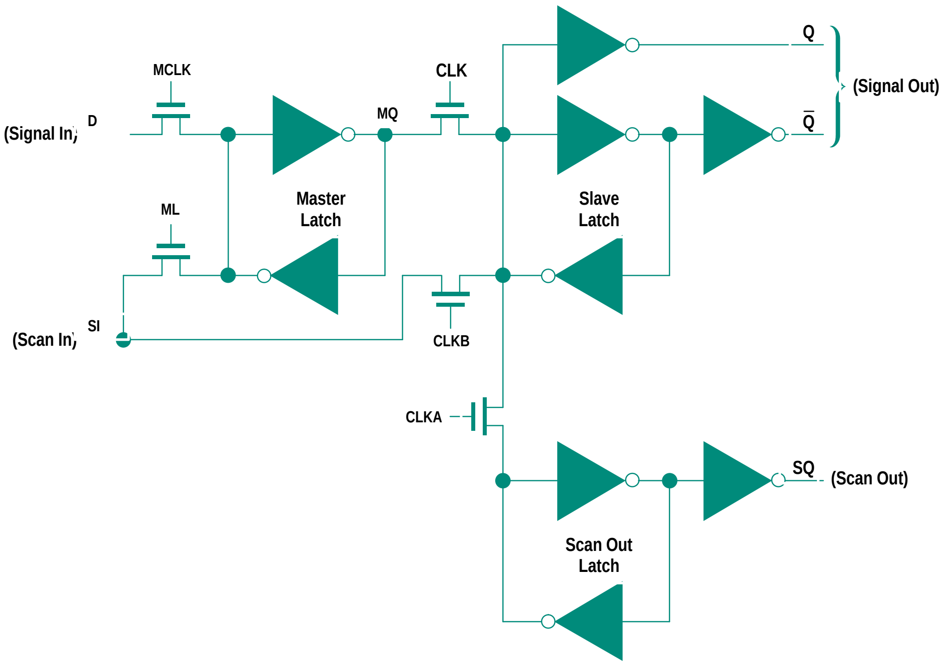


Fig. 2. Scannable flip-flop. This is a portion of our implementation of a scan cell. The MCLK, ML, CLKA, and CLKB signals are controlled by the TAP test logic and are derived from the TAP input signals TMS, TRST*, and TCK.

previous state of the TAP circuit by checking certain bits in the mode register. For example, one error that typically takes a long time to correct occurs when the bit in the mode register that controls I/O direction (PSCAN in Fig. 3) is not set properly during initialization. Forgetting to set this bit causes TSTDEN (test data enable) to go high during a scan. Later when data is supposed to be coming out of the chip (via the I/O pin) and the tester is driving a value into the chip, a bus fight occurs. We want TSTDEN to be low, which puts the gate in tristate mode when the tester is driving a signal onto the pin. Anytime the wrong data is output because of something that is done or not done early in the test cycle, it always takes a long time to debug. Debugging time can be saved if each state (bits in the mode register) is closely monitored.

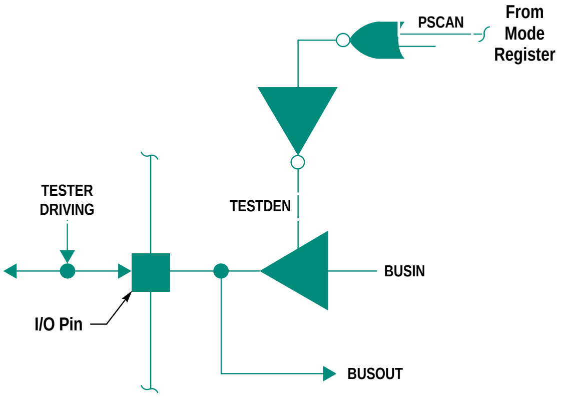
Technique 4

Check the value of each pin before each system clock. One bug that occurs frequently is that test vectors will run smoothly during simulation, but cause bus fights when they are run on the tester. From Fig. 3 we can see that if BUSIN and the tester drive an identical value onto the I/O pin at the same time, a problem occurs that can only be caught by the tester and not by simulation. This problem can be eliminated if we stop the simulation before each system clock and make sure that the I/O pin is driven by either BUSIN when the pin acts as an output (i.e., the pins drive the values to the BUSOUT), or by the tester when the pin acts as an input, but not both at one time. This task can be easily accomplished by using the Verilog-XL command $showvars.

Technique 5

Verify that the process of creating the operational test release (OTR) vectors for the tester is correct. Fig. 4 shows the additional steps we take to create and verify the OTR vectors. The first thing we do is take the raw waveform database described above and use the conditioners* ALIGN and WINDOW to create a more refined waveform database. We use the ALIGN conditioner to align each signal to be edge triggered. The WINDOW conditioner is used to select certain cycles or windows within a cycle during which output data is valid, which results in monitoring pins only at the times we care about. The next step is to verify that the windowed data is okay. This involves the same process we went through in steps 2 and 3 of Fig. 1. If everything is okay, the waveform database is converted to OTR vectors to run on the tester.

* Conditioners are functions that provide a way to modify waveform data generated via $tds_monitor. The ALIGN and WINDOW conditioners come from TSSI Inc.



TESTDEN = 0 when TESTER DRIVING Is Sending Input to I/O Pin
= 1 when BUSIN Is Sending Output to I/O Pin

Fig. 3. A simplified diagram of the circuitry around an I/O pin.

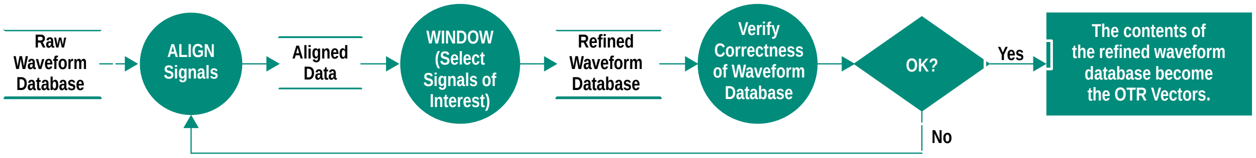

Raw Waveform Database
ALIGN Signals
Aligned Data
WINDOW (Select Signals of Interest)
Refined Waveform Database
Verify Correctness of Waveform Database
OK?
Yes
No
The contents of the refined waveform database become the OTR Vectors.

Fig. 4. The final steps in creating the OTR vectors.

Conclusion

These five techniques have proven to be a success in the Chelmsford systems lab by shortening the time it takes us to produce final test vectors. These techniques can also be applied to non-ATPG OTR vectors, since we can create vectors manually to meet different needs and put them into ATPG format. We characterized the entire analog circuitry embedded in one chip by controlling the proper bits on the scan chain.

Acknowledgments

I would like to acknowledge all those from the Chelmsford systems lab and the Corvallis Integrated Circuit Business Division lab who contributed to the slave memory controller's operational test release. Also, special thanks to my managers for their continuous recognition and support.

Estimating the Value of Inspections and Early Testing for Software Projects

A return-on-investment model is developed and applied to a typical software project to show the value of doing inspections and unit and module testing to reduce software defects.

by Louis A. Franz and Jonathan C. Shih

The software inspection process has become an important part of the software development cycle,[1,2,3,4] and has been used with varying levels of success within Hewlett-Packard.[4,5] One of the main reasons for its success is that detecting defects early has a big impact on reducing the cost of dealing with software defects later in the development cycle. One HP entity used metrics data from several software projects and an industry profit and loss model to show the high cost of finding and fixing defects late in the development cycle and during postrelease.[5]

This paper describes the methods we have used to integrate inspections and prerelease testing into the development of an information technology software project. The metrics collected and the tools we used to collect the metrics data on this project are described. Finally, we describe an approach to using the metrics data collected during inspections and testing to estimate the value (return on investment) of investing time and effort in early defect detection activities.

Background

The sales and inventory tracking (SIT) project evolved from separate initiatives by several different groups in HP. These initiatives had different objectives, but all relied on elements of the same data: computer dealer sales and inventory levels of HP products. To simplify the collection, processing, and storage of this data, it was decided to create a centralized system to house the data. All the applications that would need to use this information could access the central SIT system database. Fig. 1 shows the four major modules that make up the SIT system and the major data stores accessed by the system. These modules will be referenced throughout this paper. Table I provides a summary of the major attributes of the system.

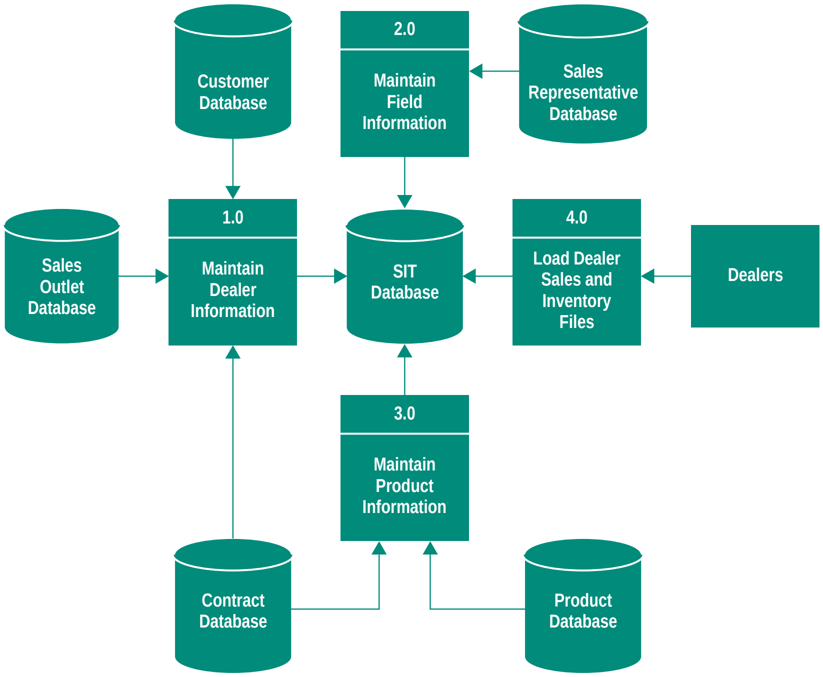

Customer Database
2.0
Maintain Field Information
Sales Representative Database
Sales Outlet Database
1.0
Maintain Dealer Information
SIT Database
4.0
Load Dealer Sales and Inventory Files
Dealers
3.0
Maintain Product Information
Contract Database
Product Database

Fig. 1. The major components of the sales and inventory tracking (SIT) system.

Table I
Summary of the Sales and Inventory Tracking System

System type	Batch
Hardware platform	HP 3000 Series 980 running the MPE/XL operating system
Language	COBOL
DBMS	HP AllBase/SQL (relational)
Data communications (used for the electronic transfer of data between the dealer and HP)	Electronic Data Interchange (EDI) ANSI standard transaction sets (file formats): 867 product transfer and resale report 846 inventory inquiry and advice
Data access and manipulation tools	Cognos, PowerHouse (Quiz), Supertool, KSAM, and VPLUS
Code	25K total lines of source code (13.6 KNCSS)*

* KNCSS = Thousands of noncomment source statements

We used the traditional HP software life cycle for our development process. The basic steps of this process include:

- Investigation
- Design
- Construction and Testing
- Implementation and Release
- Postimplementation Review.

The inspection and prerelease testing discussed in this paper occurred during the design, construction, and testing phases.

Inspection Process and Inspection Metrics

The objectives of the inspection process are to find errors early in the development cycle, check for consistency with coding standards, and ensure supportability of the final code. During the course of conducting inspections for the SIT project, we modified the HP inspection process to meet our specific needs. Table II compares the recommended HP inspection process with our modified process.

Step 3 was changed to Issue and Question Logging because we found that inspectors often only had questions about the document under inspection, and authors tended to feel more comfortable with the term issue rather than defect. The term defect seemed to put the author on the defensive and severely limited the effectiveness of the inspection process.

Table II
Comparison of Inspection Processes

Step	Our Inspection Process	HP Inspection Process
0	Planning	Planning
1	Kickoff	Kickoff
2	Preparation	Preparation
3	Issue and Question Logging	Defect Logging
4	Cause Brainstorming	Cause Brainstorming
5	Question and Answer	Rework
6	Rework	Follow-up
7	Follow-up	

The Question and Answer step was added because inspectors often had questions or wanted to discuss specific issues during the Issue and Question Logging session. These questions defocused the inspection and caused the process to take longer.

In the Planning step the author and the moderator plan the inspection, including the inspection goal, the composition of the inspection team, and the meeting time. The Kickoff step is used for training new inspectors, assigning the roles to the inspection team members, distributing inspection materials, and emphasizing the main focus of the inspection process. During the Preparation step, inspectors independently go through the inspection materials and identify as many defects as possible. In the Cause Brainstorming step, inspection team members brainstorm ideas about what kind of global issues might have caused the defects and submit suggestions on how to resolve these issues. During the Rework step, the author addresses or fixes every issue that was logged during step 3. Finally, in the Follow-up step, the moderator works with the author to determine whether every issue was addressed or fixed.

Along with the modified process, a one-page inspection process overview was generated as the training reference material for the project team. This overview was a very convenient and useful guideline for the project team because it helped to remind the team what they were supposed to do for each inspection.

Deciding What to Inspect

Because of time and resource constraints, not all of the project's 13 source programs and 29 job streams could be inspected. The project team decided to use the risk assessment done for the master test plan, which uses the operational importance and complexity of a module as a basis for deciding which programs and job streams to inspect. The risk assessment used for the master test plan is described later. Fig. 2 shows the results of this selection process in terms of inspection coverage and relative level of complexity of the programs and job streams.

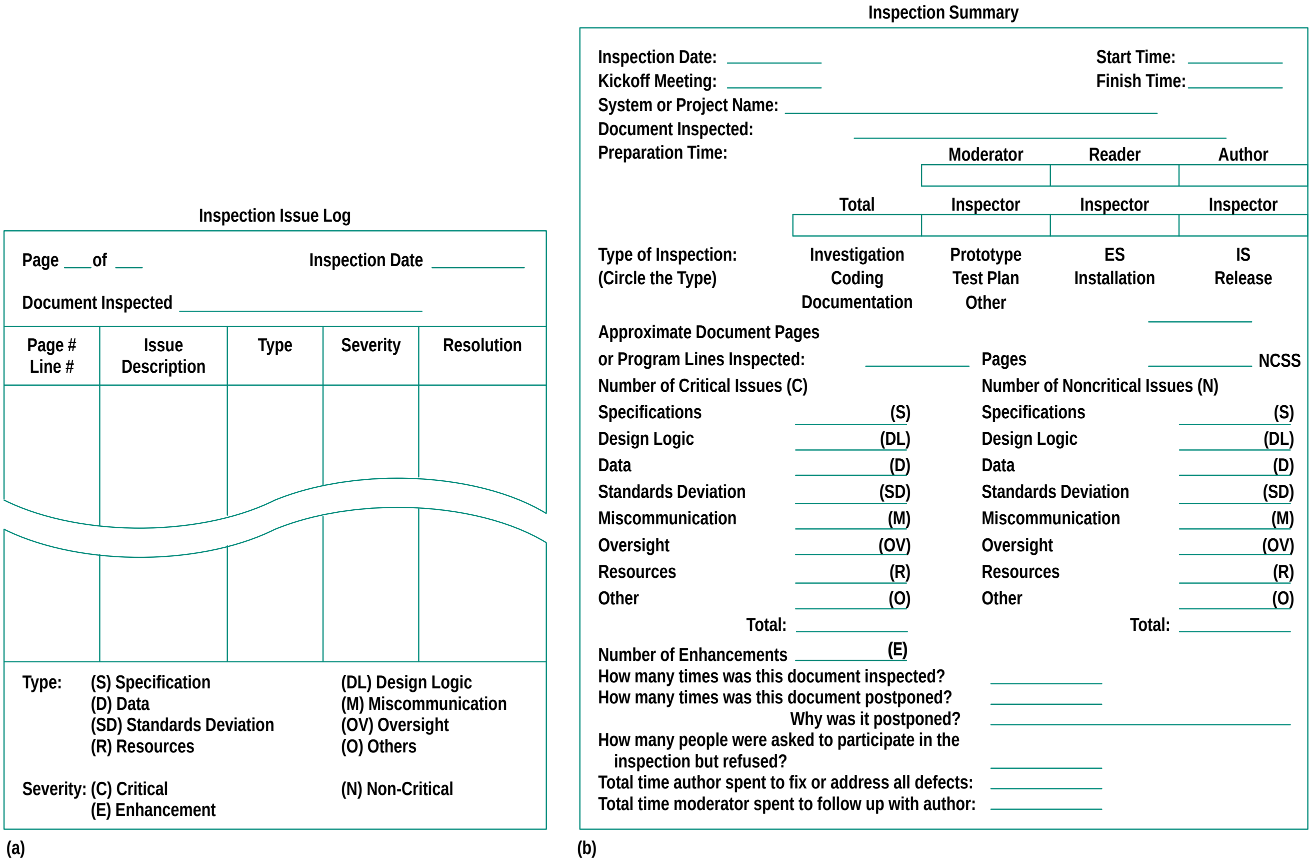
Inspection Metrics

Three forms were used to collect inspection metrics: the inspection issue log, the inspection summary form, and the inspection data summary and analysis form. Fig. 3 shows an inspection log and an inspection summary form.

Modules	Test Plan and Test Script Inspected	Percent of Programs Inspected	Percent of Job Streams (JCL) Inspected	Relative Complexity
1.0	Yes	67%	100%	Medium
2.0*	No	100%	100%	Low
3.0	Yes	60%	33.3%	Medium
4.0	Yes	100%	100%	High

* This module consisted of only one JCL and one program.

Fig. 2. Inspection coverage for the major modules in the SIT system based on the criteria used in the master plan.



Inspection Issue Log

Page ___of ___ Inspection Date __________

Document Inspected ________________________

Page # Line #	Issue Description	Type	Severity	Resolution

Type: (S) Specification (DL) Design Logic
(D) Data (M) Miscommunication
(SD) Standards Deviation (OV) Oversight
(R) Resources (O) Others

Severity: (C) Critical (N) Non-Critical
(E) Enhancement

(a)

Inspection Summary

Inspection Date: __________ Start Time: __________
Kickoff Meeting: __________ Finish Time: __________
System or Project Name: ______________________________
Document Inspected: ______________________________
Preparation Time:

	Moderator	Reader	Author

Total	Inspector	Inspector	Inspector

Type of Inspection: (Circle the Type)

Investigation	Prototype	ES	IS
Coding	Test Plan	Installation	Release
Documentation	Other		__________

Approximate Document Pages or Program Lines Inspected: __________ Pages __________ NCSS

Number of Critical Issues (C)		Number of Noncritical Issues (N)	
Specifications	______(S)	Specifications	______(S)
Design Logic	______(DL)	Design Logic	______(DL)
Data	______(D)	Data	______(D)
Standards Deviation	______(SD)	Standards Deviation	______(SD)
Miscommunication	______(M)	Miscommunication	______(M)
Oversight	______(OV)	Oversight	______(OV)
Resources	______(R)	Resources	______(R)
Other	______(O)	Other	______(O)
Total:	______	Total:	______

Number of Enhancements ______(E)
How many times was this document inspected? __________
How many times was this document postponed? __________
Why was it postponed? ______________________
How many people were asked to participate in the inspection but refused? __________
Total time author spent to fix or address all defects: __________
Total time moderator spent to follow up with author: __________

(b)

Fig. 3. (a) Inspection issue log. (b) Inspection summary form.

The inspection issue log is used for logging the issues observed by the inspectors during the Issue and Question Logging session. The inspection summary describes the document inspected, the inspector's preparation time, the type of inspection , the number of pages and lines inspected, the number and types of defects identified, and the total time used to fix or address all the defects. The inspection data summary and analysis form is a spreadsheet that was used to collect the data entries required to calculate inspection efficiency, inspection effectiveness, total time saved, and the return-on-investment value (described later). Table III lists the data collected in the data summary and analysis form for each item inspected.

We selected four key inspection metrics to measure our inspection effort: number of critical defects found and fixed, number of noncritical defects found and fixed, total time used by inspections, and total time saved by inspections.

Testing Process

Our testing process included test planning, unit testing, module testing, and system testing. Test planning involved creating a master test plan and doing a risk assessment to determine where to focus our testing time.

Master Test Plan. A master test plan was created during the design phase when the test strategy for the project was outlined (Fig. 4). The master test plan included the test plan design and the type of tests to be performed. For design purposes, the system was divided into logical modules with each module performing a specific function (see Fig. 1). The test design was also oriented around this division.

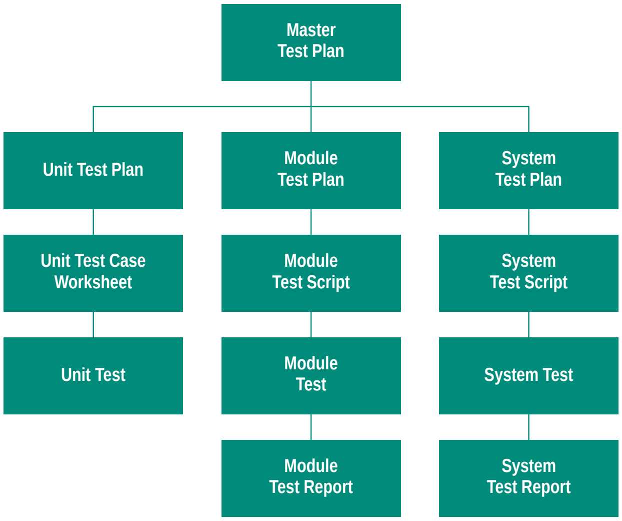


Fig. 4. Master test plan organization.

Table III
Inspection Summary and Analysis Data

Metric	Units or Source of Data
Inspection time	Preparation and meeting time in hours
Defects	Number of critical and noncritical defects
Documentation type	Code, requirements and design specifications, manuals, test plans, job streams (JCL), and other documents
Size of document	KNCSS for code and number of pages for other documents
Amount inspected	KNCSS or number of pages inspected
Preparation rate	= (number of pages) × (number of people)/preparation time
Logging rate	= (critical + noncritical defects)/(hours/number of people)
Moderator follow-up time	Hours
Time to fix a defect	Hours
Total time used	= inspection time + time to fix + follow-up time
Time saved on critical defects	*
Time saved on noncritical defects	*
Total time saved	*
Return on investment	*

* Defined later in this article.

The primary and secondary features to be tested were also included in the master test plan. The primary features were tested against the product specifications and the accuracy of the data output. Secondarily, testing was performed to ensure optimum speed and efficiency, ease of use (user interface), and system supportability.

Risk Assessment. A risk analysis was performed to assess the relative risk associated with each module and its components. This risk analysis was used to help drive the schedule and lower-level unit and integrated tests. Two factors were used in assessing risk: operational importance to overall system functionality and technical difficulty and complexity. Each module was divided into submodules and rated against each risk factor. For example, the proper execution of the logic in submodule 4.2 was critical to the success of the system as a whole, while submodule 1.4 merely supplied additional reference data to the database. Accordingly, module 4.2 received a higher operational importance rating. Similarly, submodule 4.2 also received a higher complexity rating because of the complexity of the coding task it entailed. Each rating was based on a scale of one to five with one being the lowest rating and five being the highest rating. Ratings for each risk factor were then combined to get an overall risk rating for each submodule (Fig. 5).

Module	Submodule	Operational Importance	Technical Difficulty	Overall Risk Rating
1.0	1.1	5	1	6
	1.2	5	3	8
	1.3	5	3	8
	1.4	1	1	2
	1.5	4	3	7
	1.6	3	2	5
2.0	2.1	5	2	7
	2.2	5	3	8
3.0	3.1	5	4	9
	3.2	5	2	7
	3.3	3	2	5
	3.4	5	2	7
	3.5	4	5	9
	3.6	1	1	2
4.0	4.1	5	3	8
	4.2	5	5	10
	4.3	5	4	9
	4.4	5	3	8
	4.5	3	2	5
	4.6	2	4	6
	4.7	1	3	4

Fig. 5. Risk ratings by submodule.

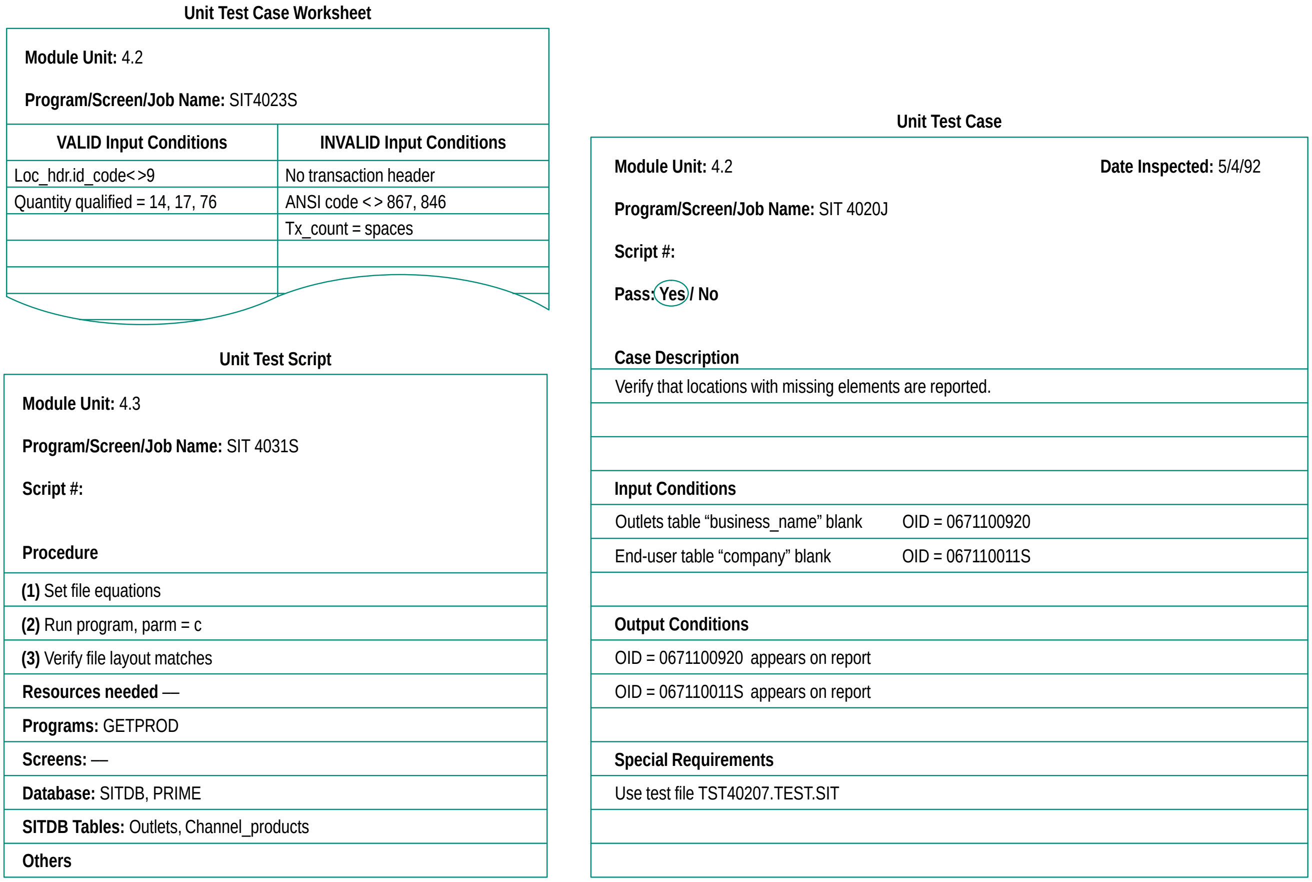
Unit Testing. Unit testing, as in most software projects, was performed for all programs and job streams. For the purpose of this project, each individual program and job stream was considered a unit. Since programs were often embedded in job streams, program unit tests were often synchronized with job stream unit tests to conserve time and effort.

Because of the small size of the project team, almost all tests were performed by the program or job stream author. To minimize the impact of this shortcoming, a simple testing review process was established. A series of standard forms were created to document each test and facilitate review by the designer, users, and the project lead at different points in the unit testing process (see Fig. 6).

Module Testing. An integrated test of all programs and job streams within each module was conducted to test the overall functionality of each of the four system modules. Since each program or job stream had already been tested during unit testing, the primary focus of module testing was on verifying that the units all functioned together properly and that the desired end result of the module's processing was achieved.

A brief integrated test plan document was created for each module. This test plan listed the features to be tested and outlined the approach to be used. In addition, completion criteria, test deliverables and required resources were documented (Fig. 7).

Detailed test scripts were used to facilitate each module test and make it easy to duplicate or rerun a particular test. Each



Unit Test Case Worksheet

Module Unit: 4.2

Program/Screen/Job Name: SIT4023S

VALID Input Conditions	**INVALID Input Conditions**
Loc_hdr.id_code<>9	No transaction header
Quantity qualified = 14, 17, 76	ANSI code < > 867, 846
	Tx_count = spaces

Unit Test Script

Module Unit: 4.3

Program/Screen/Job Name: SIT 4031S

Script #:

Procedure

(1) Set file equations

(2) Run program, parm = c

(3) Verify file layout matches

Resources needed —

Programs: GETPROD

Screens: —

Database: SITDB, PRIME

SITDB Tables: Outlets, Channel_products

Others

Unit Test Case

Module Unit: 4.2 **Date Inspected:** 5/4/92

Program/Screen/Job Name: SIT 4020J

Script #:

Pass: [✓ Yes] **/ No**

Case Description

Verify that locations with missing elements are reported.

Input Conditions

Outlets table "business_name" blank	OID = 0671100920
End-user table "company" blank	OID = 067110011S

Output Conditions

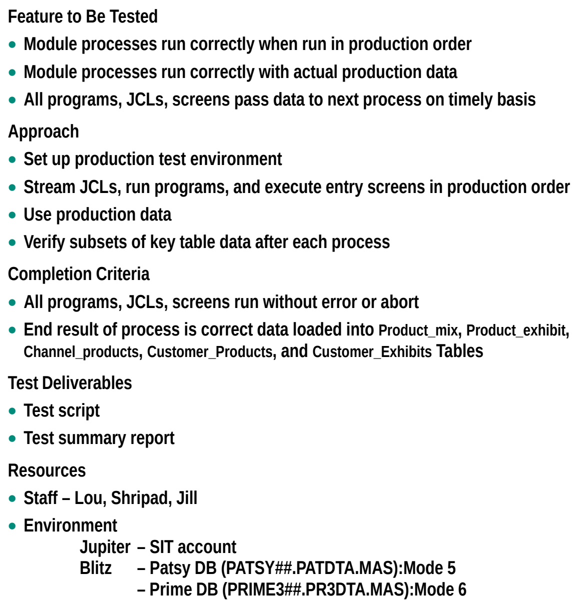
OID = 0671100920 appears on report

OID = 067110011S appears on report

Special Requirements

Use test file TST40207.TEST.SIT

Fig. 6. Unit test forms.



Feature to Be Tested

- **Module processes run correctly when run in production order**
- **Module processes run correctly with actual production data**
- **All programs, JCLs, screens pass data to next process on timely basis**

Approach

- **Set up production test environment**
- **Stream JCLs, run programs, and execute entry screens in production order**
- **Use production data**
- **Verify subsets of key table data after each process**

Completion Criteria

- **All programs, JCLs, screens run without error or abort**
- **End result of process is correct data loaded into Product_mix, Product_exhibit, Channel_products, Customer_Products, and Customer_Exhibits Tables**

Test Deliverables

- **Test script**
- **Test summary report**

Resources

- **Staff – Lou, Shripad, Jill**
- **Environment**

```
        Jupiter – SIT account
        Blitz   – Patsy DB (PATSY##.PATDTA.MAS):Mode 5
                – Prime DB (PRIME3##.PR3DTA.MAS):Mode 6
```

Fig. 7. A portion of an integrated test plan.

test script document included a listing of the resources needed for the test, such as supporting databases, SIT database tables, files, file equations, programs, and a step-by-step procedure for executing the test. Where appropriate, specific data to be entered into the system was included in the procedure. Also included were the expected results of each test step and the overall test.

Once the integrated module tests had been completed successfully (often this took several iterations), a report detailing the test results was created. The integrated test report documented the number of times the test was run, verified that the completion criteria were met, listed the number of critical and noncritical defects detected, and verified that each defect had been fixed.

System Testing. System testing was conducted in tandem with a pilot test at an HP dealer. Initial values were entered for the general-purpose database tables and for the dealer involved in the pilot test. Production schedules were used to control the job streaming that executed the system.

Testing Metrics. Two kinds of metrics were selected to measure our testing effort: total number of critical defects and total testing time for each test phase. Table IV summarizes our test metrics for unit and module testing. One critical defect was found during system testing and the total system test time was 19 hours.

Table IV
Testing Metrics

Module	Size*	Unit Test			Module Test		
		T_c	N_c	$(ATT)_c$	T_c	N_c	$(ATT)_c$
1.0	4489	73	4	18.25	41	7	5.86
2.0	639	26	0	0	**	**	**
3.0	3516	31.5	3	10.5	46	2	5.11
4.0	3738	35	21	1.67	36	13	2.77

* Noncomment source statements (NCSS)

** Not applicable

T_c = Total testing time for critical defects in hours

N_c = Total number of critical defects

$(ATT)_c$ = Average testing time per critical defect = T_c/N_c
$(ATT)_c$ is defined as zero when N_c is zero

Return-on-Investment Model

It is now generally accepted that inspections can help software projects find defects early in the development cycle. Similarly, the main purpose of unit and module testing is to detect defects before system or pilot testing. Questions that often come up regarding these defect finding efforts include how much project time they will consume, how effective they are, and how we can measure their value. Most of these issues have been addressed in different ways in the literature.[4,5]

In this section we present a return-on-investment (ROI) model we used to measure the value of inspections and early testing in terms of time saved (and early to market). The whole idea behind this kind of measurement is that it should take longer to find and fix a defect at system test than it does to find the same defect during inspection or unit testing. This means that for every defect found during an inspection or at an earlier stage of the testing phase, there should be a time savings realized at the system test phase.

First we define the Prerelease ROI as:

Prerelease ROI =
Total Time Saved / Total Time Used (1)

where:

Total Time Saved = Total Time Saved by Inspection + Total Time Saved by Unit and Module Testing

and

Total Time Used = Total Time Used by Inspection + Total Time Used by Unit and Module Testing.

We calculate the individual ROI for inspection and testing, respectively, as follows:

Inspection ROI = Total Time Saved by Inspection / Total Time Used by Inspection (2)

Testing ROI = Total Time Saved by Unit and Module Testing / Total Time Used by Unit and Module Testing. (3)

We wanted to measure not only how much time was being spent on inspection and testing but also how much time was being saved as a result of the defects found during inspections and unit and module testing.

The time used during an inspection includes the sum of the total inspection time spent by each team member, the time spent by the author on fixing the defects, and the time spent by the moderator following up on defect resolution with the author.

For inspections, we defined the total time saved and the total time used as:

Total Time Saved by Inspection = Time Saved on Critical Defects + Time Saved on Noncritical Defects

Total Time Used by Inspection = Inspection Time + Time to Fix and Follow up for Defect Resolution

where the critical defects are defects that affect functionality and performance and noncritical defects are all other defects.

The time spent finding and fixing a critical defect at system test is called BBT (black box testing time). Therefore, for every critical defect found before system test, the total time saved can be calculated as follows:

Time Saved on Critical Defects = BBT × Number of Critical Defects − Total Time Used.

The model we used to measure noncritical defects is based on the assumption that noncritical defects could be found by inspection but would not be detected by testing. The noncritical defects will become supportability issues after manufacturing release. We defined a new variable called MTTR* (mean total time to rework) to measure the time spent on noncritical defects.

MTTR = Time to Find Defect + Time to Fix Defect + Time to Release to Production.

Thus,

Time Saved on Noncritical Defects = MTTR × Number of Noncritical Defects.

For testing metrics we wanted to measure not only how much time was being spent on unit and module testing, but also how much time was being saved as a result of the defects found during these tests. Thus, we defined the total time saved and total time used for testing as:

Total Time Saved = Time Saved on Critical Defects

Total Time Used = Time to Design and Build a Test + Time to Execute + Time to Find and Fix a Defect.

* We used an average time of 6 hours for MTTR in our calculations.

The defect data and time data for our sales and inventory tracking project are summarized in Tables V and VI.

Table V
Defect Summary for the SIT Project

	During Inspection	During Testing	Total Prerelease Defects
Number of Critical Defects Found and Fixed	12	51	63
Number of Noncritical Defects Found and Fixed	78	0	78

With the code size equal to 13.6 KNCSS, prerelease defect density = 141/13.6 = 10.4 defects/KNCSS.

Using the model described above, we can calculate the ROI values shown in Table VI.

Total Time Saved by Inspection = (20 hours × 12) + (6 hours × 78) − 90 hours = 708 hours**

Total Time Saved by Testing = 20 hours × 51 − 310 =
710
hours*

From equations 1, 2, and 3:

ROI for Inspections = 708 / 90 = 787%

ROI for Testing = 710 / 310 = 229%

Prerelease Total ROI = 1418 / 400 = 355%.

Table VI
Time Data and Return on Investment Results

	Total Time Used (hours)	Total Time Saved (hours)	Return on Investment (%)
Inspection	90	708	787
Testing	310	710	229
Prerelease Total	400	1418	355

Results

Fig. 8 shows that with the exception of module SIT4.0 the average testing time per critical defect decreased from unit test to module test for the system's major modules. The reason that it took 19 hours per critical defect at system test is mainly the time it took to find and fix one defect that was overlooked during inspection. Had the project team not overlooked one particular issue related to product structure that resulted in this defect, the average testing time per critical defect at system test would have been significantly lower.

Module SIT4.0 went through the most thorough inspection including a design inspection since it is the most complex of the four modules. We believe our efforts paid off because it took less time at unit test and module test in terms of average testing time per critical defect for module SIT4.0 than for the other three modules.

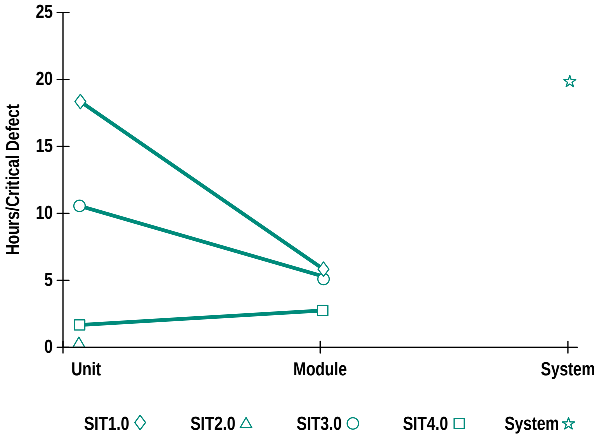


Fig. 8. Testing time by test phase and module.

Fig. 9 is a plot of the ROI column in Table VI. It shows that inspections have resulted in more than three times the ROI of testing. This reinforces the notion that a great deal of money and time can be saved by finding defects early in the software development cycle.

Lessons Learned

The inspection and testing processes we used for the SIT project are not very different from other software projects in HP. However, we did put more emphasis on early defect detection and collected a lot of metrics. The following are some of the lessons we learned from our efforts during this project.

Project Management. Some of the lessons we learned about project management include:

- Setting aside time for inspections and thorough testing does pay off in the long run. Management approval may be difficult to get, especially when under intense time pressure. One should get commitment to delivering a quality product, then present inspections and testing as part of delivering on this commitment.
- Keep high-level test plans short and simple while still providing enough direction for the lower-level plans. By keeping these plans short and simple, time will be saved and the project team can still get adequate direction.
- Adequate follow-up to inspection and testing activities is important to make sure all issues are resolved. The inspection log, testing error logs, and integration test reports

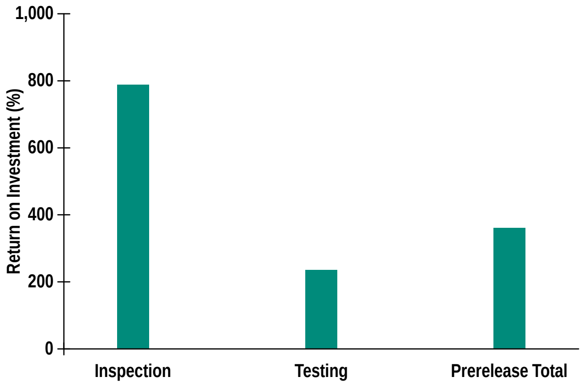


Fig. 9. Relative impact of inspections and testing.

* The time to find and fix a critical defect during system test at HP ranges from 4 to 20 hours. We used 20 hours in our ROI calculations.

helped the project lead keep up on the status of each issue and ensure that each issue was resolved.

- Establish coding standards at the beginning so that minimal time is spent during code inspections questioning points of style. The focus of code inspections should be code functionality.

Inspection. Since the inspection process is the most important tool for defect-free software, many adjustments were made here.

- Attitude is the key to effective inspections. No one writes error-free code, but many people think they do. Authors must realize that they make mistakes and take the attitude that they want inspectors to find these errors. The inspectors, on the other hand, can destroy the whole process by being too critical. The inspectors must keep the author's ego intact by remaining constructive. Perhaps the best way to keep people's attitudes in line is to make sure they know that they may be an inspector now, but at a later date, their role and the author's will be reversed. For this reason, implementing an inspection process for most or all of a project's code is likely to be much more effective than random inspection of a few programs.
- No managers should be involved in the inspection of code. Having a manager present tends to put the author on the defensive. Also, depending on the person, the inspector either goes on the offensive or withdraws from the process entirely.
- In addition to finding defects (or "issues"), which helps to save testing and rework time, the inspection process has other, more intangible, benefits:
  - ○ Increased teamwork. Inspections provide an excellent forum for the team to see each other's strengths and weaknesses and gain a new respect for each other's unique abilities. By adding the question and answer session to the inspection process, we provided a forum for the team to discuss issues and creatively solve them together.
  - ○ Support team education. Including members of the team that would eventually support the SIT system allowed these people to become familiar with the system and confident that it would be supportable.

Testing. The lessons learned from unit and module testing include the need for expanded participation in testing and the value of test scripts.

- Unit testing. On small project teams it is difficult to coordinate testing so that someone other than the author tests each unit. Establishing a process that includes the designer, other programmers, and users helps tremendously towards ensuring full test case coverage.
- Module testing. Integration test scripts are invaluable. The effort expended to create the scripts for the SIT project was significant, especially the first one. However, the reward, in terms of time saved and rework, more than justified the effort. Furthermore, these scripts have been very useful to the support team for performing regression testing when the programs or job streams are modified.

Success Factors. The SIT product was released to production in early March 1992. Since that time the product has been relatively defect-free. In reviewing what has been done, we observed some key factors that contributed to our success. These success factors can be summarized as following:

- Strong management support. We had very strong management support for the inspection and testing process and the time commitment involved. This was the most important and critical success factor for the implementation of inspections and metrics collection.
- Team acceptance. The SIT project team accepted the quality concept. We agreed on our quality goals and understood how the inspection and testing processes would help us to achieve those goals.
- Focus. The SIT project was selected as the pilot project to implement the inspection process. Our initial focus was on code inspection. After the project team felt comfortable with doing inspections, other documents such as test scripts and test plans were also inspected.

Acknowledgments

We would like to thank the following people for their help and support in developing this paper: Jennifer Hansen, Tadd Koziel, Bruce Morris, Patti Hutchison, John Robertson, Debra Vent, and Kelley Wood.

References

1. M.E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM System Journal,* Vol.15, no. 3, 1976, pp. 182-211.

2. M.E. Fagan, "Advances in Software Inspections," *IEEE Transactions on Software Engineering,* Vol. SE-12, no. 7, July 1986, pp. 744-751.

3. T. Gilb, *Principles of Software Engineering Management,* Addison-Wesley Publishing Co., 1988, Chapter 12.

4. F.W. Blakely and M.E. Boles, "A Case Study of Code Inspections," *Hewlett-Packard Journal,* Vol. 42, no. 4, October 1991, pp. 58-63.

5. W.T. Ward, "Calculating the Real Cost of Software Defects," *Hewlett-Packard Journal,* Vol. 42, no.4, October 1991, pp. 55-58.

Clock Design and Measurement Issues in Pentium™ Systems

Design difficulties in producing a statistically stable 66-MHz Pentium system are reviewed. The information is pertinent to many other new, high-speed processors as well. A new, more informed approach to designing well-timed systems in this performance class is proposed. Measurements that support this approach are examined, particularly those made with the HP 8133A pulse generator.

by Michael K. Williams and Andreas M.R. Pfaff

Clock rates in all classes of computational systems, from PCs to supercomputers, have been escalating exponentially for years. Computational systems formerly considered simpler have come to run at speeds that were previously found only in more complex and aggressive systems. Before this happened, systems at the simpler end of this spectrum (PCs and workstations) operated at clock rates that don't present very difficult clock distribution and reception problems.

Recent introductions of new processor types have given PC and workstation system designers new chips and chipsets that enable system designs that deliver much higher levels of performance.[1] Most of these devices employ internal structures that come directly from the world of mainframes and supercomputers: pipelining, 64-bit data buses, on-board floating-point units, instruction prefetching, and sophisticated caching schemes. Many of these processors are summarized in Table I. These new device families include Intel's Pentium processor, Digital's Alpha, the Apple/IBM/Motorola PowerPC, and others. These ICs have clock rates that range from tens to hundreds of MHz. Some are expected eventually to exceed 1 GHz.

Table I
Some New Processor Types and their Clock Rates

Manufacturer	Processor	Clock Rate
Intel	Pentium	60 and 66 MHz
Intel	P54C	99 MHz
Intel	486	66 and 99 MHz
Apple/IBM/Motorola	PowerPC	80 MHz
Digital Equipment Corp.	Alpha	>100 MHz
MIPS	R4400SC	150 MHz

With all of the sophisticated internal structures and faster operating speeds comes a price to be paid by the design team. Specifically, successful system design at these speeds requires very careful consideration of many mechanisms, such as timing and pulse fidelity, that are unimportant at lower speeds (16 to 33 MHz).

Pulse fidelity, sometimes referred to as signal integrity, is that part of high-speed digital design that is concerned with managing the analog effects that prevent signals from being reliably recognized at their destinations. This includes ensuring that edges arrive at their loads with proper edge speeds and proper shapes, and controlling the various types of noise (crosstalk, EMI, reflections, ground bounce, etc.) that can cause unreliable or false triggering. The extent to which these issues are important has increased dramatically in PC and workstation designs. Wider buses and faster clocks and edges (higher waveform spectral content) are the primary sources of these problems. The classic discussion of all of these analog effects can be found in reference 2.

Timing, or clock distribution and reception, is the other critical facet of design in these faster systems, and is possibly the most significant and least well-understood aspect of the design. Timing environment design is the process of specifying how the clock is to be distributed and received throughout the system such that the state architecture is reliably synchronized. Reliable means that synchronization is guaranteed on every cycle of every copy of the design that is manufactured, despite the presence of a variety of statistical tolerancing mechanisms (skew, jitter, etc.) which reduce the precision with which the clock can be delivered. These tolerancing mechanisms are described in detail on page 70. When reliable synchronization is ensured by sound design practices, the design is said to be statistically stable.

The question of exactly how to ensure this statistical stability is one that each design team must face as they adopt these new devices into their designs. Success at answering this question brings with it higher yields, fewer design turns, and the elimination of extremely subtle timing failures. Methods for doing this, while relatively new to designs at the workstation and PC level, have been commonplace in the design of higher capability systems (mainframes and supercomputers) for many years. A descriptive term for the approach that is common to all of these methods is informed design.

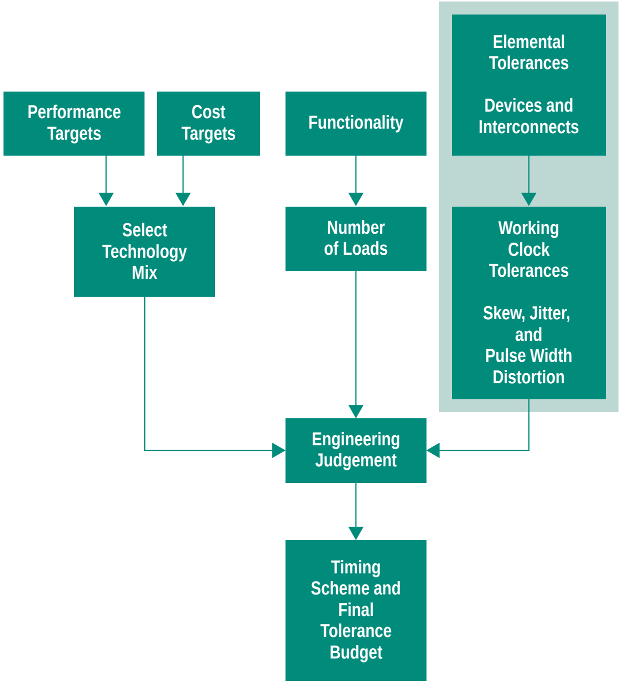


Fig. 1. Timing environment design process. Design-specific and/or unrated parametric information must be incorporated into the engineering decision making process from the outset.

Two results are produced by an informed approach to the design of a timing environment. The obvious one is a specification of a clock distribution scheme. Equally important, however, is a detailed knowledge of the tolerance on the arrival time of any clock waveform emerging from any output of any copy of that network, on any cycle of its operation. This knowledge, that is, the tolerance budget, is used by the timing verification software to determine if the rest of the system is correctly timed. Obviously the quality of this determination is a function of the quality of the tolerance budget. Informed design, as it applies to timing, can be viewed as the practice of ensuring that all of the mechanisms that contribute to the overall tolerancing of the clock have been accurately assessed.

Measurement is used to characterize devices and printed circuit board processes to see how they tolerance. This device-level tolerance data is used to compute the overall tolerance on the system clock. And this system-level tolerance is used within the timing verifier to ensure the creation of a statistically stable system. Fig. 1 illustrates where device-level parametric data fits into the overall decision making process.

In this article we examine some of the difficulties a designer will encounter in specifying, analyzing, and verifying a timing scheme for a 66-MHz Pentium system. This falls in the lower speed range for the new round of processors. However, ultratight timing specifications coupled with the currently available implementation technologies (clock buffers, printed circuit boards, etc.) make 66-MHz Pentium systems among the most difficult from a timing environment design perspective. We will see, for example, that the timing within the CPU complex (processor, cache controller, and cache RAMs) is very sensitive to clock jitter. This sensitivity, and others, make 66-MHz systems ideal for the informed design approach. Furthermore, the issues and methods presented are general and extend to other processor types as well.

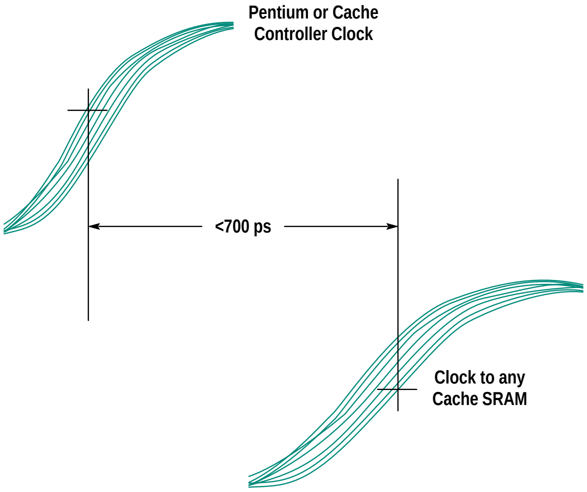


Fig. 2. The difference in arrival time between either the Pentium clock or the cache controller clock and the clock arriving at any SRAM must be less than 700 ps in every system on every cycle.

Pentium Characteristics and Requirements

An understanding of the difficulties of distributing a clock within a Pentium design must begin with an understanding of Pentium timing requirements. Our discussion of this aspect of the design will be in summary form, and the reader is referred to the Intel documentation[3-6] for a more complete discussion of requirements. Also, reference 7 discusses both the requirements and the various design decisions in much deeper detail than can be done here.

A variety of system configurations are supported by the Pentium processor. The clock rate can be either 60 or 66 MHz. The system can use either no second-level caching, or it can have 256K-byte or 512K-byte cache memories. Systems with 256K-byte caches can operate at either clock rate, while 512K-byte systems are limited to 60 MHz. A "typical" Pentium design is expected to operate at 66 MHz and have a 256K-byte second-level cache. For such systems, there are 12 clock loads within the CPU complex. Depending upon how the rest of the system is designed, the total number of clock loads will typically be in the range of 15 to 20, although in some server systems, this number can range an order of magnitude higher.

The Pentium specification dictates that the arrival times of the clock at the processor and at the cache controller never differ by more than 200 ps. It also states that the difference in arrival times between the processor and any cache memory, and the cache controller and any cache memory, can never exceed 700 ps (Fig. 2). These tolerance specifications must be met at 0.8, 1.5, and 2.0 volts. In any design, there will be other tolerance requirements that state how much difference in arrival time is permitted between clocks at loads within the CPU complex and clocks at loads external to it (external loads). These requirements will always be directly determined by the design itself. However, the overall tolerance budget will usually be driven by the timing within the CPU complex.

Tolerance Mechanisms in Clock Distribution Networks

As described in the accompanying article, we are attempting to guard against a number of statistical tolerancing mechanisms, such as skew and jitter, that reduce the precision with which a clock signal can be delivered. Here we present an overview of these mechanisms.[1] For the purpose of considering system timing issues, it is useful to separate the system state architecture into a timing environment and a computation environment (see Fig. 1). The boundary between these two parts of the system is composed of the system state devices. Except for segment delay times and communications locality, we don't address the details of the computation environment here. The timing environment can be further broken down into three sections: the clock or phase generator, the clock distribution network, and the memory elements.

The clock generator supplies the signal whose edges eventually dictate when switching occurs throughout the system. The clock generator determines the period, pulse width, number of phases, and relative phase separation of the clock waveform. The primary attributes of the generator to be specified at design time are the waveform period and stability or jitter. For systems that use a processor chip, the period is usually specified by the manufacturer of the processor. Instability (jitter) in the waveform emerging from the generator detracts from either performance or reliability. Beyond these, there are frequently secondary issues and features that contribute to system testability—frequency and duty cycle adjustability, overtone suppression, modes (burst, single-step, fast, and slow), scan-path drive and timing, and others.

The state devices are flip-flops, latches, or memory devices of some type. New devices with enhanced testability features are appearing more frequently. The state devices play an important role in determining the low-level timing constraints in that their setup, hold, and minimum pulse width requirements must be satisfied at full clock speed.

The clock distribution network is a network of buffers and interconnects that conveys the clock signal to the clock consumers. It is responsible for fanout amplification and is generally tree-structured. In simpler systems, all of the fanout can occur in a single buffer. In larger systems, thousands of copies of the clock can be produced, requiring many levels of buffering (12 to 15 levels in some supercomputers). From a timing perspective, the ideal situation is for all of the copies of the clock waveform to emerge from the leaves of the clock distribution network at the same moment. However, the devices (both buffers and interconnects) that make

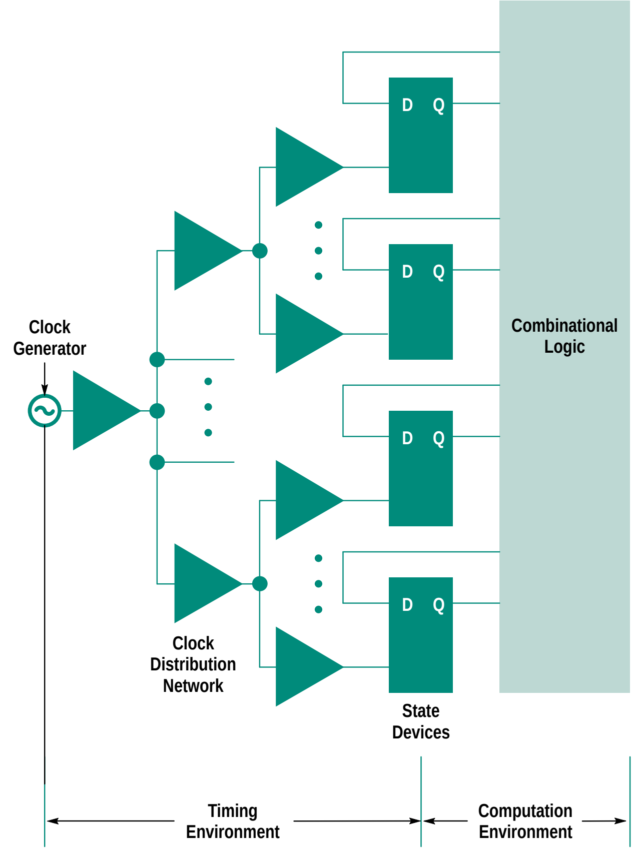
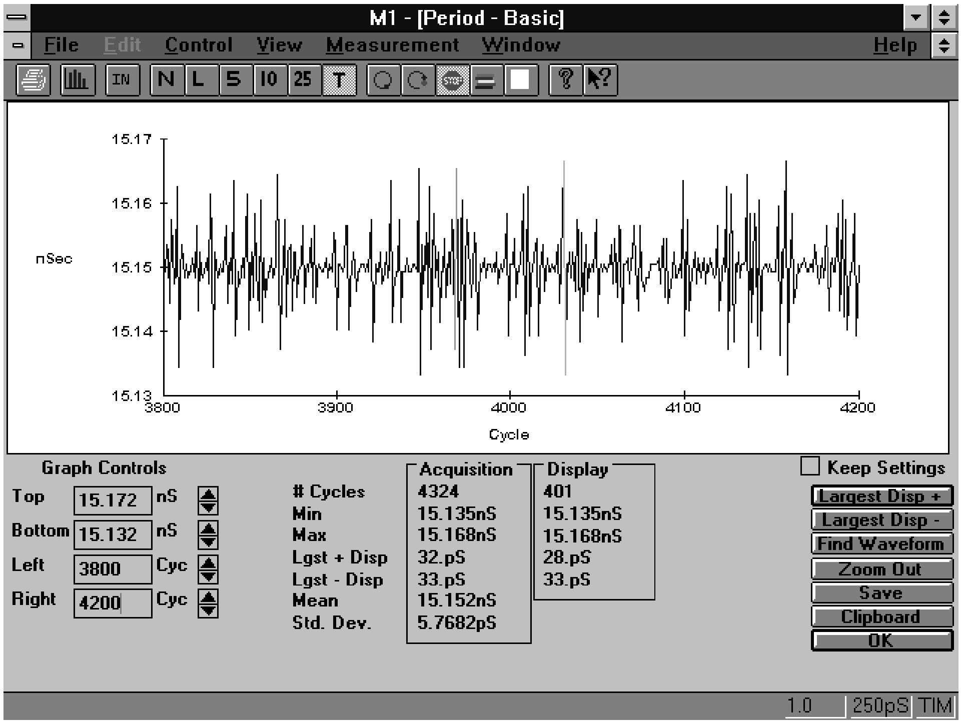


Fig. 1. State architecture model. Any synchronous digital system can be decomposed into a timing environment and a compute environment. The design issues specific to the timing environment are becoming critical in PC and workstation designs.



Graph Controls

Top 15.172 nS
Bottom 15.132 nS
Left 3800 Cyc
Right 4200 Cyc

	Acquisition	Display
# Cycles	4324	401
Min	15.135nS	15.135nS
Max	15.168nS	15.168nS
Lgst + Disp	32.pS	28.pS
Lgst - Disp	33.pS	33.pS
Mean	15.152nS	
Std. Dev.	5.7682pS	

□ Keep Settings

Fig. 2. There are a number of metrics of jitter. This measurement shows the cycle-to-cycle variation in the period of a 66-MHz clock. This was made using the Amherst Systems Associates M1 time interval measurement software, which analyzes digitized waveform data from an HP 54720D oscilloscope for jitter in a variety of ways.

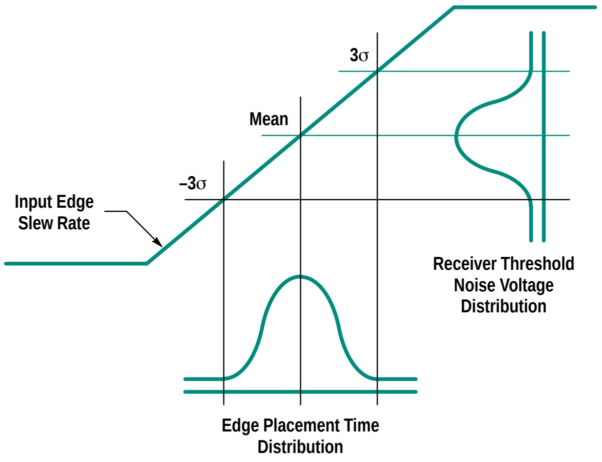

3σ
Mean
−3σ
Input Edge Slew Rate
Receiver Threshold Noise Voltage Distribution
Edge Placement Time Distribution

Fig. 3. Jitter, as it occurs in clock buffers, is generally the result of noise in the power environment (return currents, image currents, etc.) modulating the switching threshold of the buffer.

up the paths through the clock distribution network have a statistically distributed delay. These distributions can be time-invariant (static) or time-variant (dynamic). An example of a statically distributed tolerance is skew in clock buffers. This is the variation in delay either from pin to pin in a single package or from part to part. Interconnects can also exhibit tolerancing. This is most easily thought of as a variation in the propagation rate of several picoseconds per inch (10 to 40 ps/in). Interconnect tolerancing is frequently a source of unanticipated timing failures.

An example of a dynamically distributed tolerance is jitter. The placement in time of a waveform edge that has jitter varies from one cycle to another. It can be thought of as having a period that changes from one cycle to the next. Fig. 2 shows an example of this variation. Jitter can be added to the clock waveform in two places: at the generator or in the buffers. At the generator, jitter can occur through either internal noise or dynamic temperature or supply voltage instabilities. Jitter added in the clock buffers is caused primarily by noise in the power environment (return and image currents in power planes sweeping past power and ground pins, etc.) causing time-varying shifts in the device's switching threshold. This is illustrated in Fig. 3. Note that jitter (an expansion of the distribution of the edge placement) is increased when the noise voltage is increased or the edge rate of the signal arriving at the buffer is decreased. The management of jitter at consistent and acceptably low levels is perhaps the single greatest challenge for designers of systems that incorporate many of the new high-performance processors. A more in-depth discussion of jitter measurement and management can be found in references 1 and 2.

There are also statistical variations in how two identical parts are used. For example, one system may run a little warmer than another, another may have a little more noise in the power environment, and so on. Some of these tolerances are time-variant and some are not. As shown in Fig. 4, these device-level distributions can be statistically combined† to give a system-level distribution on the path delays in the clock distribution network. This system-level path delay distribution has a mean value that is sometimes called the nominal delay. By statistically combining the individual nominal delays along the path, one computes the nominal delay for that path.

When using the nominal delays, it is important to keep in mind that there is actually a delay distribution. This means that even if every path in the design is specified to be identical, when the product is manufactured there will be product-to-product variations in the propagation delay of any given path, there will be path-to-path variations within any given machine, and there will be cycle-to-cycle variations on a given path in a given machine. The result is that one must design the system in a manner that both suitably minimizes these tolerances and consciously considers the fact that the tolerances will always be nonzero. The design is said to be statistically stable when it has this characteristic.

When the tolerances in the system accumulate beyond the value anticipated by the designer, the design is said to be statistically unstable. In statistically unstable designs, some small fraction of the manufactured systems will experience timing failures despite the absence of any physical defects. In these systems, the clock can arrive at times other than the designer anticipated, and this can mean that one or more of the state device timing requirements (setup time, hold time, or minimum pulse width) will be violated.

Violations of any of the device-level timing requirements can result in statistically unreliable switching at the state devices. This can cause unpredictable deviations in normal system-level behavior. These faults can be extremely difficult and time-consuming to isolate. In fact, the failure modes exhibited by systems with internal timing problems are easily among the most difficult to diagnose using conventional troubleshooting methods. It is frequently necessary to employ an analytic approach to find these faults in any sort of efficient manner. These failure modes include:

- Intermittent or nonrepeating
- Low frequency of occurrence (minutes through weeks)
- Migration of the symptom location through the system
- Hibernation (failures occur as device parameters change slightly with age)
- Statistical.

References

1. M.K. Williams, "Distortion and Tolerance Mechanisms in High-Speed Clock Delivery," *Proceedings of the 1993 Hewlett-Packard High-Speed Digital Symposium,* pp. 4-1 to 4-41. Also available as Application Note ASA 93-1 from Amherst Systems Associates.

2. M.K. Williams, "Design Trade-offs in High-Speed Clock Distribution and Reception," *Proceedings of the 1993 Hewlett-Packard High-Speed Digital Symposium,* pp. 6-1 to 6-34. Also available as Application Note ASA 93-2 from Amherst Systems Associates.

† The combination of these subordinate distributions is more complicated than direct addition. It must also take into account correlations that occur in such tree-structured circuits, and other related mechanisms called tracking effects.

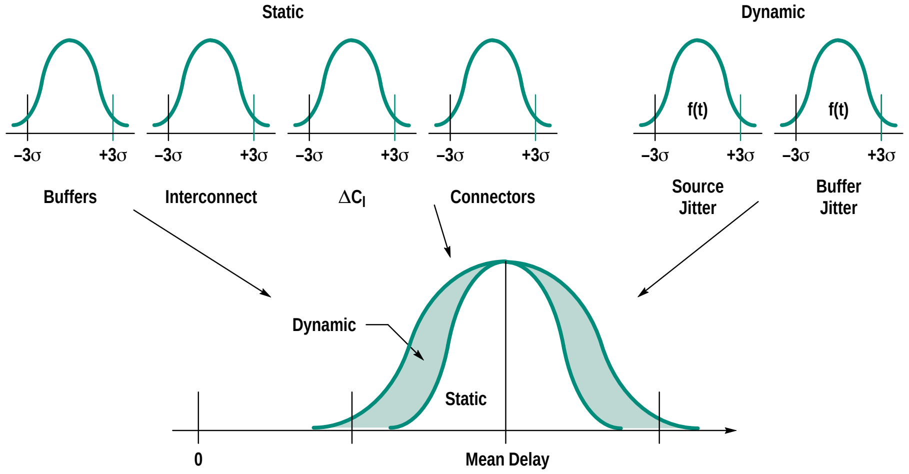


Fig. 4. A variety of tolerancing mechanisms contribute to the uncertainty in the arrival time of the clock edge at any clock load. Generally the only one that is available in catalogs or data sheets is the buffer tolerancing.

In general, the difficulty of any particular timing environment design can be estimated from two facets of the design: the number of clock loads and the amount of allowable clock tolerance, expressed as a fraction of the period. One threshold of difficulty occurs at about ten board-level clock loads† and tolerance budgets that are less than 10% or so of the period. For the typical 66-MHz system we have assumed that the loading (15 to 20 clock loads) ranks it as somewhat difficult. The tolerances within the CPU complex of 200 and 700 ps represent 1.3% and 4.7%, respectively, of the 15-ns cycle time. This represents a very challenging timing requirement. Table II summarizes Pentium clock tolerancing for various system configurations.

Table II
Clock Tolerancing and Loading within the Pentium CPU Complex

Clock Speed (MHz)	Cache Size (bytes)	Tolerance (ps)	Number of Loads
60 or 66	None	N/A	1 (CPU only)
66	256K	700	12 (CPU, cache control, 10 SRAM)
60	512K	800	20 (CPU, cache control, 18 SRAM)
60	256K	800	12 (CPU, cache control, 10 SRAM)

Design Example

In this section, we attempt to impart some insight as to where the tolerance budget comes from. We illustrate some of the aspects of the design that are major drivers of this budget. Our goal is to show the importance of having complete and accurate design information at every step of the process. However, the process of completely and precisely evaluating each component of that budget is complicated, and is beyond the scope of this article. The interested reader is directed to References 8 and 9 for a more in-depth discussion of the design decisions presented here.

Before describing the design, we encourage the reader to adopt the view that every design decision that pertains to clock paths should be made very carefully and considered from the perspective of how that decision impacts the tolerancing of the clock. It is a fact that every physical design decision (buffer selection, transmission line geometry and impedance, termination, grounding schemes, etc.) that relates to the clock paths impacts clock tolerancing.

Preliminary Decisions. Our example design here is a fully synchronous 66-MHz system with a 256K-byte second-level cache. It is based on the use of the Intel 82496 cache controller and the 82491 cache SRAMs. In this discussion, we make almost no assumptions†† about circuitry beyond the CPU complex, since the design challenge lies with the clocks within the complex. Beyond this, we assume the Intel suggested device placement (Fig. 3). Placement must be very carefully considered for these devices not only from a clock-distribution perspective, but also from the perspective of the times of flight of all of the data, address, and control signals. These times are very precisely specified in the Pentium specification.

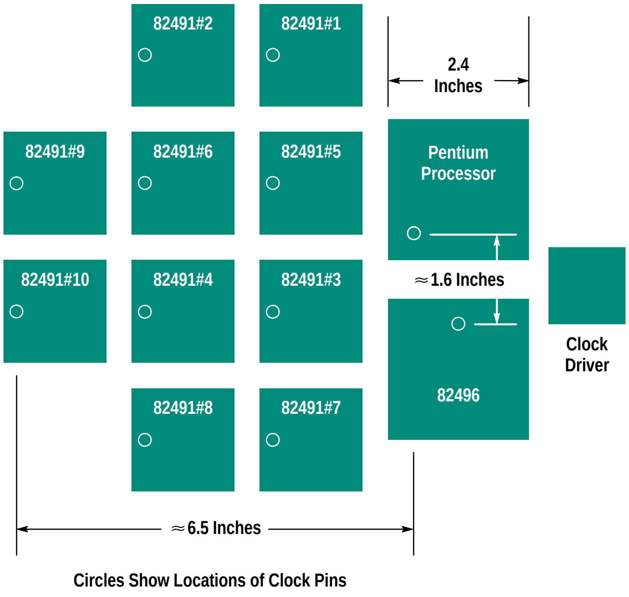


Fig. 3. Intel suggests this placement for use with their second-level cache chipset.

As stated earlier, the typical design is expected to have a total of 15 to 20 board-level clock loads. To minimize clock tolerancing caused by variations in the load capacitance, it is desirable to drive the system in a point-to-point fashion. This means one clock load per clock buffer pin. We have selected a 20-output static clock buffer for this role. It has a pin-to-pin tolerance (skew) of 500 ps.

The interconnect for the design being described here was also very carefully considered. It was decided to route all clocks in microstrip (typically less tolerancing than stripline because of a faster propagation rate). An interactive field solver was used to design the microstrip. The resulting propagation rate is 146.4 ps/in.

Predicting Actual Clock Tolerances. A good way to begin is to do an inventory of where the clock loads in the system are expected to be placed and get as much information as possible about what types of loading they will present. Intel provides very complete pi-models††† for all of the pins on the devices within the Pentium CPU complex (also known as the “optimized interface group”). These models provide minimum, maximum, and typical values. The minimum and

† Most clock buffers have 10 or fewer outputs. When the number of loads in a design exceeds this level, either multiple loads must be clustered on each output or a multichip solution is required. The former increases the load capacitance range ($C_{max} – C_{min}$) any output can see, which increases the difference in arrival time between the fastest and slowest conditions. The latter solution, using additional devices, increases cost and the length of the clock paths, which in turn increases the opportunity for tolerancing to occur in the clock.

†† We will make reference to worst-case external clock loading when we do the load/placement inventory.

††† A pi-model is a standard ac model of an input pin, consisting of a parallel inductor, a series capacitor, and another parallel inductor.

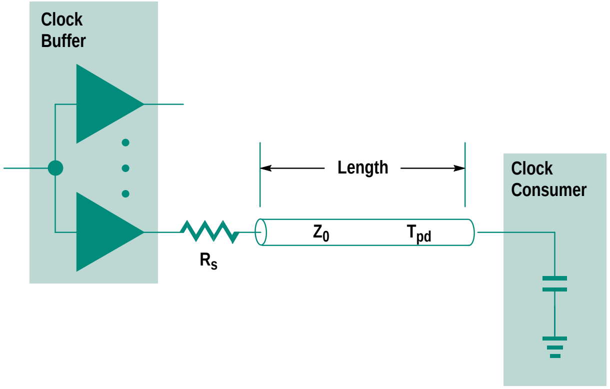


Fig. 4. Most of the clock nets in our design can be viewed as simple series-terminated transmission lines driving single capacitive loads.

maximum ratings permit an accurate determination of the range of distortion delay† that will occur at any pin type. Usually, the best a designer can hope for in terms of published parametric pin data is typical input capacitance values. This only permits estimation of the typical distortion delay, not the range.

When the clock load inventory is completed, the designer will know approximately how far most loads are from the clock buffer and which buffers are most heavily loaded (typical). This information lets the designer estimate how late the slowest load typically reaches threshold. From this value, the other clock paths can be adjusted (e.g., by serpentining) to align their typical delays with the slowest one in the system. For this design, the result of the inventory is that the largest mean path delay is 1586 ps. The delay ranges for all of the other paths in the system are centered on this value.

Since we used point-to-point distribution for most†† of the clock paths, the general structure of the clock nets is shown in Fig. 4. The general formula for computing the tolerancing at this point is:

$$\text{tolerance}_{\text{net}} = \text{skew}_{\text{int}} + \text{skew}_{\text{ext}} + \text{jitter},$$

Skew_{int} is the intrinsic skew, that is, the delay variation of the buffer (pin-to-pin in this case). For our buffer, this is 500 ps. Skew_{ext} is the extrinsic skew, that is, the delay variation along the net. Jitter is the peak value, rather than rms or some other statistical jitter metric.

Extrinsic skew is not a single mechanism. It can be broken down into two major components:

$$\text{skew}_{\text{ext}} \cong \Delta LT_{\text{pd}} + \text{tol}_{\text{mfg}},$$

where ΔLT_{pd} is the variation in the propagation delay of a signal down a loaded transmission line. It takes into account the range of loads seen at the end of a net. Tol_{mfg} is the manufacturing tolerance of the interconnect. It ranges from about 1 ps/in to about 50 ps/in times the length of the interconnect.

ΔLT_{pd} can be computed from:

$$\Delta LT_{\text{pd}} = LT_{\text{pd}}\left(\sqrt{1+\frac{C_{l\max}}{LC_0}}-\sqrt{1+\frac{C_{l\min}}{LC_0}}\right),$$

where L is the length of the net in inches and T_{pd} is the unloaded propagation rate in ps/in. $C_{l\max}$ and $C_{l\min}$ are the maximum and minimum values of load capacitance. To compute the difference in arrival times between two clock loads, these values will be from different pins. Equivalent values for C_l can be computed from pi-models. C_0 is the intrinsic capacitance of the net.

Following this general format for computing the tolerances, we can compute a worst-case difference in arrival times of the clocks to the cache controller and the cache RAMs.

$$700\text{ ps} \geq \text{skew}_{\text{int}} + \Delta LT_{\text{pd}} + \text{tol}_{\text{mfg}} + \text{jitter}.$$

Plugging in what we computed,

$$700\text{ ps} \geq 500 + 90 + 60 + \text{jitter},$$

which gives us the constraint on clock jitter:

$$50\text{ ps} \geq \text{jitter}.$$

The overall tolerance budget is summarized in Fig. 5. The jitter constraint is very aggressive for PC and workstation class computers. Normally, this constraint is a full order of magnitude higher. Keeping noise levels low enough to meet this constraint will present some unique measurement requirements, as we shall see in the next section.

Incorporating Measurement Information

We have, thus far, described a number of the more challenging issues that must be addressed in producing a 66-MHz Pentium design with statistically stable timing. We have attempted to emphasize the importance of employing informed design practices. The basic tenet of these practices is that important design decisions (e.g., timing verification) are based upon deliberately and accurately gathered design information. The better this design data is, the better the design decisions that are based upon this data. In the case of timing,

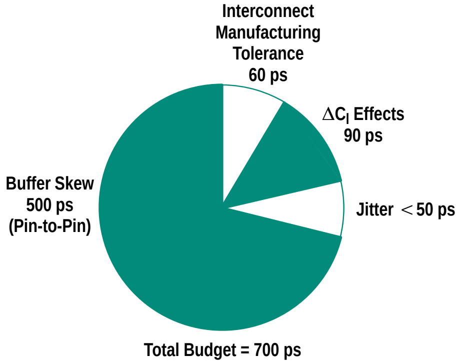


Fig. 5. After accounting for all of the tolerancing mechanisms we have little or no control over, our typical Pentium design can tolerate approximately 50 ps of jitter.

† Distortion delay is that component of the delay that a clock edge experiences as it arrives at the load and enters the die. The parametrics of the pin, as represented by the pi-model, act as a filter. The more the high-end spectral content of the edge is attenuated, the more the slope of the edge is reduced, adding delay in the amount of time it takes the waveform to climb to threshold. What is important with the clock is not absolute delay, but delay variation, so when the parametrics vary more widely, more variation (tolerance) can occur in the timing at the pin. This variation is often referred to simply as load capacitance variation.

†† Because of a 200-ps allowable difference in arrival time between the processor and the cache controller, these two loads are actually clustered at the end of a single clock net. This is discussed in much more detail in Reference 4.

we are talking about all of the low-level tolerance information required to compute an accurate tolerance budget.

As noted on page 70, the only significant component of the tolerance budget that can be found in data sheets is the buffer tolerance. All of the other low-level tolerance information must be determined through measurement. It cannot be generated for a design at one company and shared with others. The tolerance information is determined by the specific methods and devices employed in a particular design, and each design is unique in these regards.

Perhaps the most notable measurement information relates to the very tight jitter allowance. An upper limit of 50 ps will require exploration and experimentation of various design alternatives (device placement, bypass filtering, ground plane cuts, etc.) to determine their exact effect on jitter. Jitter caused by switching noise will be first-order sensitive to clock buffer placement. And this may involve some measurement activities that are very new to PC and workstation design activities.

Measurement is usually viewed as a stimulus and response process. Stimulus gear includes pulse and function generators and waveform synthesizers. Response gear includes oscilloscopes, time-interval analyzers, spectrum analyzers, and so on. Response is unquestionably important when the measurement of very low-amplitude jitter (10 to 50 ps) is being performed. However, one of the less well-understood facets of precision measurement relates to the specification of stimulus gear and methods for these measurements. In the high-speed PC and workstation designs we're discussing here, stimulus issues center primarily in two areas: characterization activities and applications calling for an alternate, adjustable clock source. As we shall see, the precision of the waveform submitted to a device under test has a significant impact on the quality of the design data that results from the measurement. In this section, we discuss a number of measurement methods that apply to these two areas.

Instrumentation Issues. For all of the measurements described in this section, the way the measurement is made and the quality of the instrumentation employed in the measurement are issues of genuine importance. The importance of precision cannot be underestimated. Any tolerance on the measurement must also be included in the final tolerance data. That, of course, means that measurement tolerance directly detracts from system performance.

The very low levels of jitter allowed in the systems we're discussing makes the measurements very challenging. For example, the waveform timing uncertainty or jitter of the source (pulse generator) must be much less than the jitter of the device under test (DUT). There are two reasons for this. The first is to avoid corrupting the measurement. A good rule of thumb is to try to keep stimulus jitter an order of magnitude below what you are expecting to measure. In that way, the majority of the jitter measured is what occurs within the DUT. The second reason for low source jitter is that the tolerance budget establishes an upper bound on the amount of jitter permitted on the clocks distributed to the loads, and the total jitter on those clock signals includes jitter from the signal source. Consider, for example, Fig. 6, which shows a clock buffer being driven by an external signal source. The buffer can be viewed as a "jitter mixer," that is, the total jitter transmitted to the clock loads is the sum of the jitter that the buffer adds because of noise (J2) and the jitter on the externally generated waveform that drives the buffer (J1). If J1 is significant with respect to J2, it will swamp the measurement. Furthermore, if J1 + J2 exceeds the jitter limit, the system will not function properly during the measurement. This brings up an interesting point. If you plan to make these sorts of measurements and use an external signal source, you must account for the jitter of whatever signal source you may use in the tolerance budget.

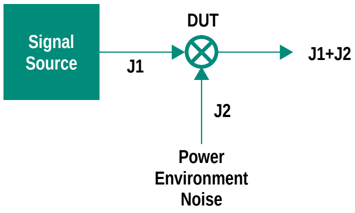


Fig. 6. When the device under test is a static clock buffer it acts as a jitter mixer, combining noise-induced jitter with jitter coming in from the signal source. For tight systems like Pentiums, it is clear that both the source jitter and the power environment jitter must be kept to a minimum to permit reliable testing and characterization.

In our Pentium design, our 50-ps allowance for jitter means that if we plan to use a signal source with 15 ps of jitter, we should limit jitter in the system to less than 35 ps. A 10-ps source will permit the design to work with 40 ps of in-system jitter. However, to use a source with much more than 15 ps of jitter means greater design difficulty in minimizing in-system jitter,† and increasing difficulty in interpreting system-level jitter measurements because of the difficulty of determining how much of the jitter is from the source and how much is from the system.

Substitute Clock Measurements. The most common reason for using an external source to drive the clock is to do system-level timing margin testing and verification. The fundamental question behind these measurements is how sensitive the system is to imperfect device timing. In other words, the sensitivity of the system to variations in parameters such as frequency, duty cycle, skew/jitter, or phase separation†† is being determined.

Fig. 7 illustrates a measurement setup for one type of margin testing. Specifically, the setup permits investigating how sensitive load number one is to various types of parametric tolerancing by controllably varying the parameters of the waveforms produced by the signal source. For example, by advancing the phase of the waveform to load 1 and noting where unreliable switching occurs, and then retreating the phase of load 1 and again noting where unreliable switching

† It is probably a useful rule of thumb that when the stability requirement of the clock in a mass-produced computer system exceeds the stability found in precision pulse generators, the requirement is perhaps too aggressive.

†† Phase separation is a parameter in systems with multiphase clocks. It is the minimum separation between an edge in one clock phase and an edge in another.

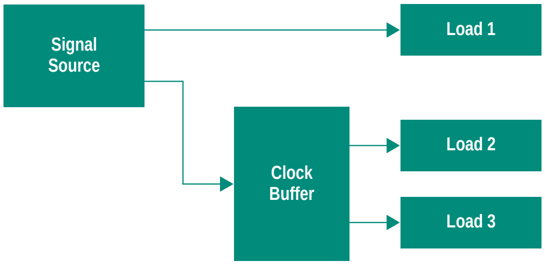


Fig. 7. The margin available at a specific load in a system can be examined by driving that load from a two-channel signal source and carefully adjusting the relative phase of the channels until unreliable switching is detected.

occurs, the operating limits of the load 1 clock can be estimated.† It is common during the course of a design to need to adjust or ascertain the tolerance at a specific point in the system (i.e., a point tolerance). For example, the clock to a particular point in the system may have to be forced to be earlier, or less toleranced than originally assumed, because some aspect of the segment bounded by that state device has changed.

Another critical verification is that the jitter that actually occurs in the final hardware is acceptably low. The designer starts with an assumption of what can be achieved. However, accurately predicting jitter is difficult, even with "representative" assessment boards and experiments. Front-end assessment of jitter is important, but only an estimate can be produced without final hardware. Only the final hardware will have the actual switching activity, the actual return and image currents, and the actual paths and obstacles that steer these currents. To verify jitter, it's necessary to measure it in a variety of locations and switching conditions.

One other significant application of an alternate, adjustable clock source occurs during debugging. The external clock can bypass either the clock source or paths through the clock distribution network to permit the investigation of a timing problem at the loads. The benefit of this, of course, is that a defective source or clock distribution network can be bypassed or the loads supplied with a clock with jitter reduced to below normal operational levels.

Characterization Measurements. The verification activities described in the previous section are intended to determine how sensitive the system is to imperfect timing. Device characterization measurements ask the question from the other side—how imperfect might the timing be? This class of measurements includes fixtured device measurements.

For example, phase-locked loop clock buffers are basically active signal sources. As such, they have jitter of their own (intrinsic jitter). To characterize various facets of this jitter (cycle-to-cycle deviation, phase noise, jitter spectrum, settling time, susceptibility to power supply noise, etc.) without corruption from some external effect, it is important to supply the device with a stable reference signal and a clean power environment (Fig. 8). Note that a measurement of the intrinsic jitter of a well-fixtured phase-locked loop clock buffer does not establish how the device will perform in the system. Instead, it establishes an upper bound on stability. The live system will have a noisier power environment and less stable reference signals. The spectrum of these disturbances will not likely be consistent in every application, nor will it be easily predicted. The behavior of the phase-locked loop is affected in a very complex way by the superposition of these various external processes.

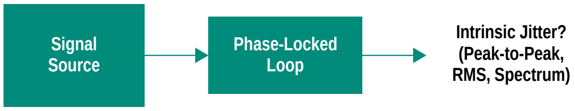


Fig. 8. A stable reference signal should be supplied while characterizing a phase-locked loop.

Another measurement process that involves the clock buffers is the determination of so-called "derating factors." The published tolerances for clock buffers include not only process and manufacturing variations, but also consideration that the parts may be operated across a range of temperatures, operating voltages, and loadings. The system designer has no control over how buffers vary because of process variations, but does have control over the range of temperature, voltage, and loading in the design, and may wish to attempt to remove that part of the buffer tolerance that is attributable to these margins on the operating ranges. A series of fixtured device measurements made while carefully varying some environmental variable can yield estimations of how much of the published tolerance is attributable to the environmental operating range.

There is also a role for board-level measurements. As stated earlier, the jitter of a clock buffer (phase-locked loop or static) is affected to a large extent by the level of noise in the power environment. More specifically, it is determined by the noise where the device power and ground pins attach to the power and ground planes. This noise is caused by image and return currents in those planes. There are places on the board where this noise is significantly higher than other places. Furthermore, the gradient of these changes can be fairly tight, with quiet points existing millimeters from points that carry high image currents. All this means that buffer placement and orientation on the board have an impact on clock jitter. It is possible to evaluate approximately where the quiet locations are on a "technology board." Fig. 9 shows such an experiment. The external signal source is used to drive a representative collection of switching gates.

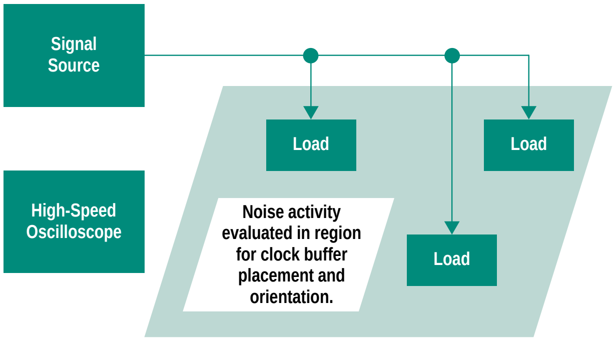

Signal Source
Load
Load
High-Speed Oscilloscope
Noise activity evaluated in region for clock buffer placement and orientation.
Load

Fig. 9. By examining the power environment noise in the region where the clock buffer is expected to be placed, the quietest power and ground connection points can be determined.

† Of course, this only shows the sensitivity of that particular system. However, that result can then be guardbanded to take into account what might happen across a larger population of systems.

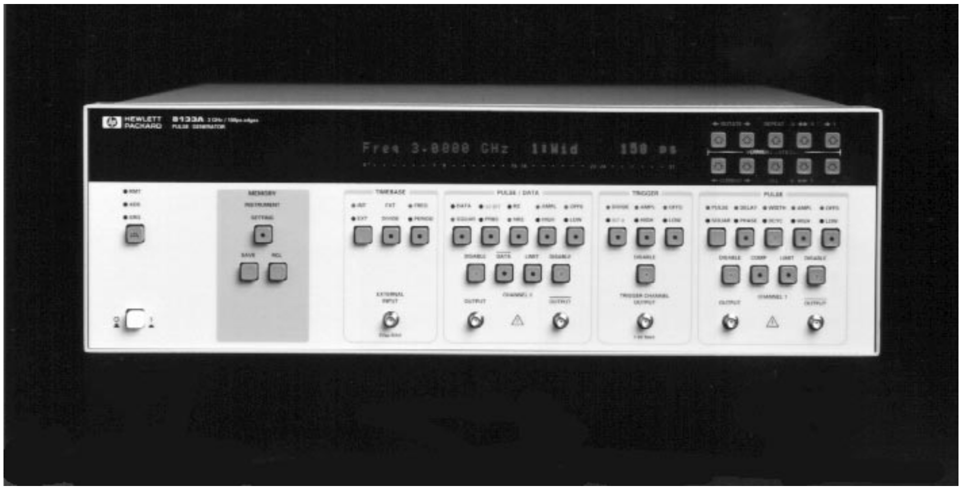


Fig. 10. The HP 8133A 3-GHz pulse generator is an excellent candidate for use as a high-stability, high-resolution signal source for testing Pentium and other high-speed processor designs.

It is unlikely that the gates on the technology board will be exactly the circuitry that appears in the final design, since this sort of activity is most likely performed very early in the design process. A grid of test points in the region where the buffer is likely to be placed offers visibility into the power and ground planes, and these can be evaluated by a high-speed oscilloscope or spectrum analyzer. Once it is known where the quiet locations are, the placement and orientation of the buffer can be specified.

HP 8133A Pulse Generator.[10,11] For many of the measurements described in this section, it is critical to use a high-precision adjustable signal source. The HP 8133A pulse generator (Fig. 10) is an excellent choice for the stimulus instrument in these measurements. It is stable, accurate, and precise. The rms jitter for this instrument is warranted to be less than 5 ps. Both authors have had the opportunity to characterize a number of these instruments. The result of these characterizations is that the distribution is approximately Gaussian. Furthermore, for pulse repetition rates below 500 MHz, the rms jitter of the instruments is typically 1.2 to 1.3 ps. Rms jitter is equal to one standard deviation of the jitter distribution. Worst-case jitter can be taken to be six standard deviations. For a Gaussian distribution, this means that worst-case jitter is approximately 8 ps. Applying this to our 50-ps Pentium tolerance budget, we would have to ensure that the system jitter is less than 42 ps to ensure that the system functions correctly during testing. This also means that most of the jitter of the measurement comes from the system and not from the external source.

When the HP 8133A is configured as a multichannel instrument (Option 003 is recommended for clock characterization and testing activities), the phase delay from one channel to the other can be adjusted in 1-ps increments from the front panel or in 300-fs steps over the HP-IB (IEEE 488, IEC 625).

If a less stable or precise source is used for these measurements, the quality of the results could be compromised. For example, if we assume jitter levels of just a few tens of picoseconds, the system may not even function properly during testing and the measurement of any jitter in the system will be less meaningful since the majority of the jitter will come from the external source.

Summary

In this article, we have reviewed some of the significant challenges that exist in designing a statistically stable timing environment for a 66-MHz Pentium system. Many of the difficulties described easily generalize to most of the other new high-speed processors as well. We have advanced the argument that a new, more informed approach to designing the timing for these more aggressive systems is required. This informed design approach requires the determination of important design information at the front end of the design process so that important subsequent design decisions can be made knowledgeably, with more predictable results.

We also examined a variety of measurements that support this approach. Our tolerance budget for a typical Pentium system revealed much more sensitivity to jitter than has been common for designs at this level. Our discussion centered on the measurement of jitter-related design information. In the course of discussing these measurements, we also examined the role of stimulus equipment. Specifically, we discussed what impact various facets of the performance of a high-stability pulse generator would have on the quality of the measurement data. For example, the simple decision to use a higher-stability pulse generator as an adjustable substitute for the clock means that the design can have higher levels of intrinsic jitter (i.e., a simpler design) and continue to function during testing. In the course of our discussion, we showed how the HP 8133A pulse generator can be employed in designs as aggressively timed as Pentium and others.

References

1. L. Geppert, "The New Contenders," *IEEE Spectrum,* Vol. 30, no. 12, December 1993, pp. 20-25.

2. W.R. Blood, Jr., *MECL System Design Handbook, Fourth Edition,* Motorola Inc., 1988.

3. *Pentium Processor User's Manual—Volume 1: Pentium Processor Data Book,* Intel Corporation, 1993.

4. *Pentium Processor User's Manual—Volume 2: 82496 Cache Controller and 82491 Cache SRAM Data Book,* Intel Corporation, 1993.

5. D. Lin and J. Reilly, *Pentium Processor Clock Design,* Application Note AP-479, Intel Corporation, March 1993.

6. R. Jolly, *Clock Design in 50-MHz Intel486 Systems,* Application Note AP-453, Intel Corporation, 1991.

7. M.K. Williams, *Clock Design in Intel Pentium Systems,* Amherst Systems Associates Application Note ASA 93-3, 1993.

8. M.K. Williams, "Design Trade-offs in High-Speed Clock Distribution and Reception," *Proceedings of the 1993 Hewlett-Packard High-Speed Digital Symposium,* pp. 6-1 to 6-34. Also available as Application Note ASA 93-2 from Amherst Systems Associates.

9. M.K. Williams, "Timing Considerations in Clock Distribution Networks," *Proceedings of the 1992 Hewlett-Packard High-Speed Digital Symposium,* pp. 2-1 to 2-21. Also available as Application Note ASA 92-2 from Amherst Systems Associates.

10. H.J. Wagner, "A Programmable 3-GHz Pulse Generator," *Hewlett-Packard Journal,* Vol. 44, no. 2, April 1993, pp. 52-55.

11. P. Schinzel, A. Pfaff, T. Dippon, T. Fischer, and A.R. Armstrong, "Design of a 3-GHz Pulse Generator," *Hewlett-Packard Journal,* Vol. 44, no. 2, April 1993, pp. 60-72.

Enterprise Modeling and Simulation: Complex Dynamic Behavior of a Simple Model of Manufacturing

Simulating a structurally simple model of a manufacturing enterprise revealed complex dynamic behavior. Enterprise modeling and simulation provided estimates of end-of-life inventory and order delivery performance based on interactions of forecast quality, quoted product availability, material procurement and safety stock policies, vendor lead times, product life cycle, and part commonality. An unexpected result was that end-of-life inventory can exist even under ideal environmental conditions. Prospective applications of these methods include estimating the effects of incremental improvements, verifying impacts of process changes, and generating enterprise behavior information.

by M. Shahid Mujtaba†

Can we understand the potential impacts of process changes? Can we quantify the expected amount of improvements and benefits? Can we anticipate the effects of environmental changes? Can we predict the effects and side-effects of making changes? And can we do all these before taking action and making major resource commitments?

We suggest that the answer is yes to all these questions, and the means is enterprise modeling and simulation.

The purpose of this paper is to show how enterprise modeling and simulation research activities at HP Laboratories can be applied to predict system behavior and gain insights using sound engineering and scientific principles and techniques before implementing the new solution at the level of the business enterprise.

In this paper, we first discuss modeling and simulation technology in broad terms to provide background and context. We then describe one model, the Simple Model, in detail, and present the insights gained from running simulations on that model and analyzing and displaying the results. An unexpected insight was that end-of-life inventory existed at the end of the product life cycle even though the method for computing safety stocks should theoretically have resulted in none when customers ordered exactly according to forecast. Other interesting insights were that high inventory levels can occur when actual orders come in too high or too low with respect to forecasts. In other words, forecast quality has a major impact on some of the metrics under consideration. We then describe the current state of enterprise modeling and simulation, future research directions, and possible application areas, including process reengineering on page 86. In the appendixes we include more detailed explanations and sufficient technical details of the model to permit the results to be duplicated by other researchers. A glossary of terms and a summary of the values for different experiments are provided for quick reference on pages 85 and 95. The evolution of enterprise modeling and simulation activities at HP Laboratories and the place of the Simple Model in those activities provides a historical context and is described on page 90.

Modeling and Simulation

Extensive literature exists on the simulation modeling process, for example Chapter 1 of Law and Kelton,[1] Chapter 1 of Pritsker,[2] Chapter 6 of McHaney,[3] and Law and McComas.[4] The general consensus is that the purposes of the simulation modeling process are to define a problem clearly and to develop a model as a tool to understand and solve that problem.

"Modeling and simulation have become endeavors central to all disciplines of engineering and science. They are used in the analysis of physical systems where they help us gain a better understanding of the functioning of our physical world. They are also important to the design of new engineering systems where they enable us to predict the behavior of a system before it is actually built. Modeling and simulation are the only techniques available that allow us to analyze arbitrarily nonlinear systems accurately and under varying experimental conditions."[5]

"The facility or process of interest is usually called a *system*, and in order to study it scientifically we often have to make a set of assumptions about how it works. These assumptions, which usually take the form of mathematical or logical relationships, constitute a *model* that is used to try to gain some understanding of how the corresponding system behaves."[1]

Thus, a model is a conceptual abstraction of an existing or proposed real system that captures the characteristics of interest of the system. Modeling is the process of building the abstraction (model).

† Author can be reached at email address mujtaba@hpl.hp.com.

"If the relationships that compose the model are simple enough, it may be possible to use mathematical methods (such as algebra, calculus, or probability theory) to obtain exact information on questions of interest; this is called an analytic solution. However, most real-world systems are too complex to allow realistic models to be evaluated analytically, and these models must be studied by means of simulation."[1]

"Simulation is the use of a model to develop conclusions that provide insight on the behavior of any real world elements. Computer simulation uses the same concept but requires that the model be created through programming on a computer."[3]

In general, modeling and simulation are useful when system prototyping is too costly or time-consuming, seriously disruptive, or simply impossible. They are useful for exploring proposed system changes by providing performance estimates of a proposed system or of an existing system under some projected set of operating conditions. A simulation model or set of models can provide an experimental testbed on which to try out new ideas or concepts, since it is cheaper to experiment in the laboratory than on the real system.

Our premise is that these techniques applied to enterprise processes could help predict the behavior of the organization more quantitatively than repeated assertion or the application of mental models.

Enterprise Modeling and Simulation

We define enterprise modeling as the process of building abstractions or models of three primary functional components of an enterprise: manufacturing, marketing, and R&D (research and development) for the purpose of gaining insight into the interactions between these functions and the interaction of the enterprise with other enterprises. The complexity of the enterprise and the large number of people who have ownership of different parts makes it difficult for a single individual to grasp a detailed understanding of all the components. There is a limit to the level of complexity and the means to share and communicate it with others that can be carried in the head of a single individual.

Many process changes and decisions are based on implicit mental models in the heads of decision makers or advocates. Mental models[6] are deeply ingrained assumptions, generalizations, or even pictures or images that influence how we understand the world and how we take action. Very often, we are not consciously aware of our mental models or the effects they have on our behavior.[6] Generally mental models assume that there are a small number of factors in cause and effect relationships. The problem with mental models is the difficulty of communicating them, checking their consistency, and combining the mental models of different people. It is very difficult to estimate the effects of interacting factors and to combine mental models into a larger-scale composite model that incorporates the insights, knowledge, and understanding of many individuals.

One means of merging different viewpoints is the use of Hierarchical Process Modeling,[7,8] which provides an explicit, graphical representation of the process with which individuals can agree or disagree. Experience in applying Hierarchical Process Modeling[9] to the building of enterprise models suggests that the result is a repository for knowledge of the processes we are studying. During its creation, team members develop a common understanding of the dynamics of model behavior through interaction with one another and with the model. The result is an explicit model that reconciles differing points of view and a reusable model that serves as a foundation on which to build future models.

There is an awareness that a model can be used to embody knowledge of a system rather than be used as a tool.[10] For example, Funke[11] states that at the Boeing Company, simulation has provided "a forum for the collection of process operating rules and assumptions in one medium as a basis to develop the model" of a process or system.

Other ongoing works on the application of models to embody knowledge at the enterprise level of manufacturing operations include TOVE[12] and CIM-OSA.[13,14] Pardasani and Chan[15] describe the expansion of an infrastructure for creating simulation models based on the ISO reference model for shop floor production standards to create enterprise models.

In applying the process of enterprise modeling and simulation we need to engage in activities of modeling in the large (with "model as knowledge") where the major issues of interest are communication and documentation, team coordination, modularity and large model development, and multimodel organization, instead of modeling in the small (with "model as tool") where the issues of interest are top-down design, informal and formal program specifications, simplification and elaboration, and validation and verification.[10]

In modeling the manufacturing enterprise, the primary area of focus is the manufacturing function, which includes, in addition to the traditional production and shop floor functions, the production and material planning, material management, and order processing functions. In traditional modeling and simulation applied to the manufacturing domain, computer simulations have been applied to the production floor or machine shop level to study machine utilization and production and material flows and buffers. These methods together with traditional operations research methods have helped reduce inventory on the production floor and cut build times to a level where these are small compared to the other parts of the system. Enterprise modeling and simulation expand the scope so that traditional modeling and simulation are components in the enterprise modeling and simulation system.

Enterprise modeling and simulation indicate the impact of proposed improvement efforts at the enterprise level before the changes are made. The "simulation" in enterprise modeling and simulation is the process of running the model in a computer to understand the behaviors over time under different operating conditions and circumstances. It will help us identify leverage points and indicate where we can expect to get the most impact for a given investment or change.[16]

According to Senge,[6] "The real leverage in most management situations lies in understanding dynamic complexity, not detail complexity." He suggests that most systems analyses focus on detail complexity (that is, a large number of variables), not dynamic complexity ("situations where cause and effect are subtle, and where the effects over time of interventions are not obvious"). We suggest that enterprise modeling and simulation help in understanding dynamic complexity, and in addition provide the framework for slowly expanding the detail complexity.

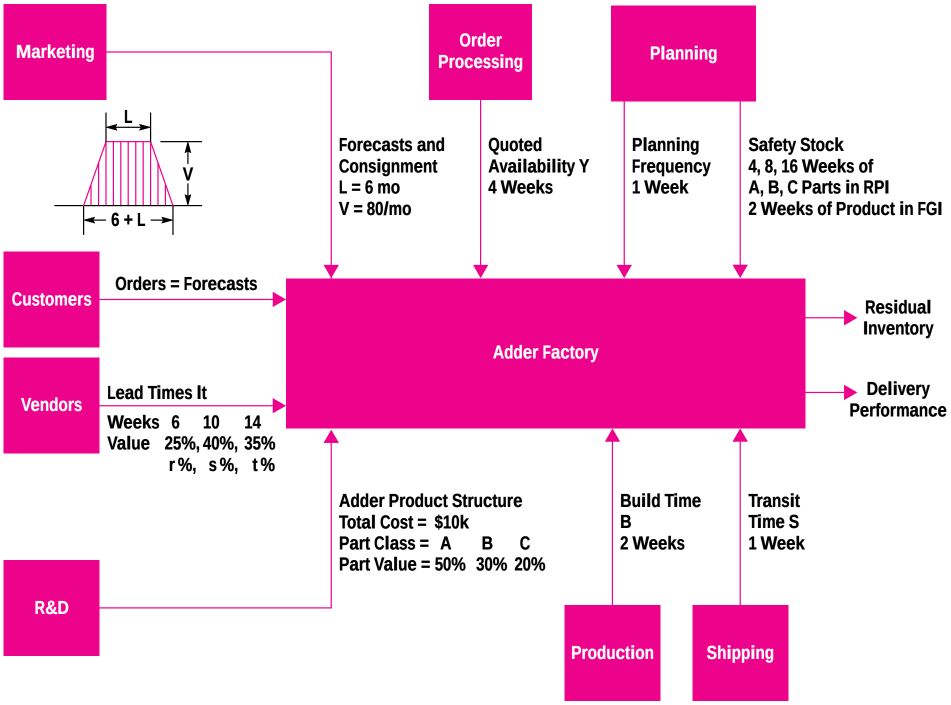


Fig. 1. Diagram of the Simple Model for the nominal case experiment.

Modeling and simulation at the enterprise level are showing increasing levels of activity. For example, a recent article in Fortune magazine[17] discusses business-oriented economics that focuses on what economists call “the firm” and the rest of us call “the company” as the unit of analysis. (Traditional microeconomics, by contrast, is concerned with markets and prices. It looks at the economy or at an industry, but rarely peeks inside the individual enterprise.) Fortune cites the example of Merck’s finance team, which built a completed model and subjected it to Monte Carlo simulation analysis.

The Simple Model

The Simple Model (shown with capital letters because of its importance in this paper), was one in a series of models developed at HP Laboratories (see page 90). The Simple Model was named because of its structural simplicity, but as subsequent descriptions will show, it exhibits dynamic behavior that is complex and not intuitively obvious until it is explained. Expressed in terms used by Senge,[6] the Simple Model is a tool for understanding dynamic complexity using a model with very low detail complexity.

The Simple Model was commissioned to abstract a real manufacturing facility with greatly simplified assumptions, such as a single product with a one-level bill of materials and a trapezoidal order demand pattern. The purpose of the model was to explore the relationship between different factors and metrics used in manufacturing. Although the model can generate data on many different metrics, this paper will focus on two main metrics: (1) inventory levels and write-off at the end of the product life cycle and (2) customer satisfaction metrics. We will first describe the structure and assumptions of the Simple Model and then show the results of running the model under different conditions.

Conceptual Description

Fig. 1 shows conceptually the Simple Model of a factory producing a product called Adder.† Marketing specifies a trapezoidal order forecast profile for customer orders, and the number of consignment units (defined as demonstration units used in the sales offices). R&D specifies the Adder product structure. Order processing quotes a product availability of four weeks. Production determines that the build time is two weeks, and shipping states that transit time for sending the product to the customer is one week. We assume that the production and shipping processes are under sufficient control that they do not vary from these constant numbers.

The problem assumes that the values of class A, B, and C parts in the Adder product make up 50, 30, and 20 percent, respectively, of the product material cost. In valuing the finished product, labor cost is small enough to be factored into the material cost, and the value of the product is the sum of values of its parts. In addition, we assume that the values of 6-week, 10-week, and 14-week lead time parts make up 25, 40, and 35 percent, respectively, of the product cost, that the vendors deliver the parts exactly on time, and that there are no rejects because of defective parts.

These characteristics are reflected in Table I, which shows the value of each part category. There are a large number of unit costs and part quantity combinations that satisfy the above constraints. The actual bill of materials used for the model is shown in Table II.

The length of the longest lead time among the parts is 14 weeks for parts A.3, B.3, and C.3. Allowing a build time of

† There was a little bit of whimsy in naming the product. The author selected the name from a fairy tale in which somebody ordered the biggest adder available, expecting it to be an adding machine. When the box was opened, out popped a snake. Snakes, of course, was an internal HP code name for a class of workstations.

Table I
Simple Model Adder Product Structure

(a) Product Structure by Part Value

Part	Value	Part	Value	Part	Value
A.1	$1250	B.1	$750	C.1	$500
A.2	$2000	B.2	$1200	C.2	$800
A.3	$1750	B.3	$1050	C.3	$700

(b) Part Value by Part Class Safety Stock

Class	Parts in Class	Value	% Value	Safety Stock
A	A.1,A.2,A.3	$5000	50%	4 weeks
B	B.1,B.2,B.3	$3000	30%	8 weeks
C	C.1,C.2,C.3	$2000	20%	16 weeks
	Total	$10,000	100%	

(c) Part Value by Lead Time

Lead Time	Parts	Value	% Value
6 weeks	A.1,B.1,C.1	$2500	25%
10 weeks	A.2,B.2,C.2	$4000	40%
14 weeks	A.3,B.3,C.3	$3500	35%
	Total	$10,000	100%

Table II
Adder Bill of Materials

Part	Quantity	Unit Cost	Value in Product
A.1	1	$1250	$1250
A.2	1	$2000	$2000
A.3	1	$1750	$1750
B.1	1	$750	$750
B.2	1	$1200	$1200
B.3	1	$1050	$1050
C.1	1	$500	$500
C.2	1	$800	$800
C.3	1	$700	$700

two weeks and transit time of one week means that the period from the time parts A.3, B.3, and C.3 are ordered in the manufacturing enterprise to the time that the product using those parts is received by the customer is 17 weeks. This means that the policy of waiting for customer orders before we order parts from our vendors will lead to an order-to-delivery time of at best 17 weeks.

To quote availability of four weeks requires us to order material and plan production before we receive customer orders. The best information we have on current and past customer behavior is actual orders, and the best information we have on future customer orders is the order forecast.

Given that we want quoted availability to be less than the sum of material delivery, production, and product delivery times, we need to plan ahead of time how much to build based on order forecasts. This decision on how much to build in future weeks is the responsibility of production planning, which each week computes the number of units to be started in future weeks.

Forecasts of future customer orders are estimates; customers may order more or less than forecasted. In the event that customers order less, we should have no problem meeting the demand if we build to meet the forecast. However, if customers order more, we might run out of product. To allow for this contingency production planning must specify that we need to build a few more units and carry them in a stock of finished goods inventory (FGI). The amount of extra product to be carried is the safety stock, and depends on many factors including the average expected order level, the expected fluctuations in orders, and how much we want to allow for contingencies. A high safety stock level will protect us from low forecasts, but requires a greater investment in inventory. One way of specifying inventory levels is to use a measure related to number of weeks of forecasted demand. In the case of this model, we assume that production planning specifies two weeks of 13-week leading average forecast as target FGI safety stock.

The discussions for FGI safety stock are also applicable for raw material. There must be enough raw material on hand when the time comes to build the product. To allow for excess demand from the production line because of high customer demand, and for late deliveries by vendors, we need to order some extra material. This extra amount is determined by material planning and is the target raw parts inventory (RPI) safety stock. The amount of RPI safety stock can be determined in different ways. One way is to use part classification.

In practice, part classification indicates the relative importance of a part and hence the attention it receives. Since class A parts are reviewed more frequently, a smaller quantity is carried than for the B or C parts. In our model, part class determines the amount of material safety stock to be carried in weeks, and all parts are reviewed weekly by material planning. For A, B, and C parts the target RPI safety stock is 4, 8, and 16 weeks, respectively, of the 13-week leading average forecast. The 13-week leading average forecast and the FGI and RPI target safety stocks are discussed in greater detail under “Target Safety Stock,” below.

Fig. 2 shows the trapezoidal product order forecast supplied by marketing. The demand during each week of a four-week month is constant. The demand builds up over three months,

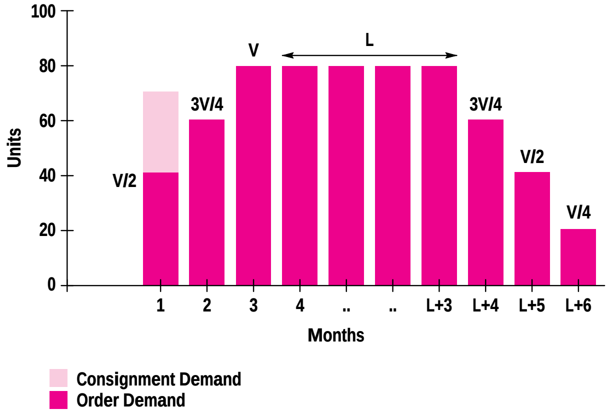


Fig. 2. Adder order forecast and consignment demands in units.

remains constant for L months, and then reduces to zero over three months, so the total product life is L+6 months. In the first month, some units are required for consignment purposes. The mature monthly demand V is 80 units, and the total amount of inventory for consignment is set at 1.5 weeks of projected mature demand, or 30 units. In our experiments we used a baseline value of 6 months for L. This order forecast results in a lifetime total of 780 units, or a total forecasted production cost flowthrough (PCFT, see Glossary, page 85) of $7.8 million, exclusive of the 30 consignment units.

Of the many performance metrics for the system during the product life cycle, the three main ones of interest are the end-of-life inventory, which needs to be disposed of or written off, the shipment and delivery performance, and the inventory during the product life cycle.

Detailed Description

The fundamental description of the Simple Model of the enterprise and the primary flows and dynamic components that interact with it over time are shown in Fig. 3.

Entities External to the Enterprise. Customers send orders to the manufacturing enterprise. In the simulation each order for a single unit is sent individually to the manufacturing enterprise. The orders go into the backlog of the manufacturing enterprise, and at some point a shipment fulfilling each order is delivered to the customer. Customers have the expectation that the time between ordering and receipt of delivery is the quoted availability, but are willing to wait indefinitely for orders.

The manufacturing enterprise sends orders for each part to the respective vendor, shown collectively in Fig. 3 as vendors. The shipment of physical parts arrives at some time in the future determined by the lead time for the part. Ideally the time between the issuance of an order and receipt of the material (parts) should be the lead time quoted by the vendor, and for all the runs in this paper, this will be the case.

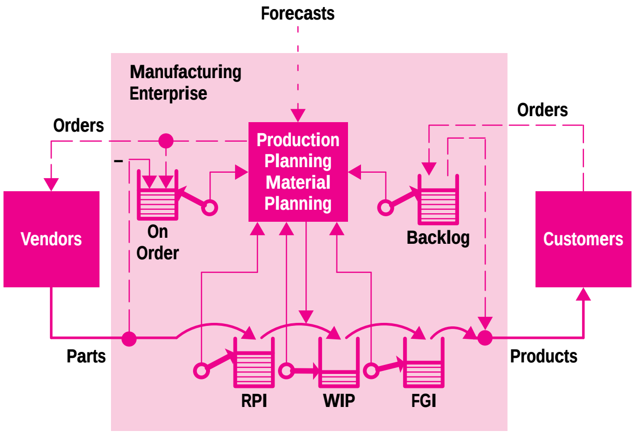


Fig. 3. Material, order, and information flows of the Simple Model simulation. The heavy solid lines represent the flow of physical material, the long-dash lines represent the flow of information related to individual orders, and the short-dash line represents the flow of periodic order forecasts. The containers represent the accumulation of physical material or orders, the pointers represent levels of the quantities in the containers, and the light solid lines from the containers represent this status information being transmitted to the planning function. The light solid line from the planning function represents a control signal flow that regulates the amount of material flowing from RPI to WIP and ultimately to FGI.

Functions Internal to the Enterprise but External to Manufacturing. Periodically, *marketing* provides forecasts of customer orders in future periods. Each forecast is a list of the quantity of products that are estimated to be ordered in subsequent periods. In practice, forecasts are updated periodically and estimates for the same month in the future can vary from month to month. In the model, the forecast is used to compute the shipment plan, and to compute the 13-week leading average forecast for computing FGI and RPI safety stocks. *R&D* (not shown in Fig. 3) provides a bill of materials (BOM) that defines the product structure. Since the BOM does not change during the simulation, we do not show the R&D function.

Processes Internal to the Manufacturing Function. This section should be read in conjunction with Figs. 1, 2, and 3.

Order processing accepts orders and keeps track of all outstanding orders received from customers, and keeps a running total of the quantity of products required in the backlog. In addition, it prioritizes the orders by the ranking criterion, which in this model happens to be first-in, first-out (FIFO), into a ship list. The backlog level is provided to the production planning function. The prioritized list of orders and the quantity that needs to be shipped in the current period are provided to shipping.

Shipping fills and ships the orders on the ship list that order processing provides. From the point of view of the manufacturing enterprise, the duration between receipt of customer order and delivery of the shipment at the customer site should be the time period specified as the quoted availability. Filling an order is attempted no earlier than necessary to satisfy the quoted availability taking transit time into account. An order is filled and shipped only if at the time of the attempt the number of units in FGI is greater than zero. In other words, shipping's objective is to fill outstanding orders that need to be filled and not to try to maintain FGI at some level. This means that the actual order-to-delivery time for a particular order will depend on whether units are available to fill the order at the time the order is due to be shipped. If units are not available, the order will have a higher priority for being filled in the next period because of the FIFO rule used to establish the ship list.

Production planning computes the current shipment plan and build plan. It computes the current shipment plan from the current order forecasts and current order backlog to attempt to satisfy the quoted availability. It then computes the current build plan from the shipment plan, build time, current FGI, current WIP, and FGI safety stock.

To come up with a suitable build plan, production planning must know about the characteristics of the system it is trying to control, that is, it must have a model of the system that it uses for doing its computation. An important aspect of the computation is to take into account the number of units already in process rather than relying only on the number of units of product required. Such a model is generally a mathematical or analytical model, and the formulation is described in Appendix I. The build plan for the current period is used

Glossary of Terms and Abbreviations

Abbreviations

A/F. Actual-to-forecast ratio. This is the ratio of the actual orders received to the forecasted orders. Normally expressed as a percentage. A/F greater than 100% implies that actual orders came in higher than forecasts, that is, forecasts were low or demand was high. A/F less than 100% implies that actual orders came in lower than forecasts, that is, forecasts were high or demand was low.

BOM. Bill of materials. A description of the components that go into an assembly and their respective quantities.

- Single-level BOM. The components are raw materials fabricated or manufactured elsewhere (i.e., purchased parts).
- Multiple-level BOM. The components are other assemblies and purchased parts.

EOL. End of life (end of product life cycle).

FGI. Finished goods inventory.

RPI. Raw parts inventory. Raw material in stores waiting to be processed.

WIP. Work in process. Raw material on the production line being assembled into the final product.

PCFT. Production cost flowthrough. Dollar value of production passing through the manufacturing enterprise. Because of the assumptions underlying the Simple Model, in this paper PCFT is synonymous with shipments from the manufacturing enterprise.

Terms

Backlog. Products ordered by customers but not yet shipped.

Build Time. The time required for completion of the product when all the parts are available.

Committed Inventory. The total inventory to which the manufacturing enterprise is currently committed. It is the sum of the on-order material and the on-hand inventory.

Consignment Inventory. Inventory in the sales offices and for demonstration purposes.

End-of-Life Inventory. The amount of inventory left over at the end of the product life cycle, that is, when no more orders are backlogged or outstanding for the product. EOL inventory includes leftover unused RPI, unshipped units in FGI, and consignment inventory. In general, material and products left over at the end of the product life cycle are not useful for anything else and must be written off.

Forecast Quality. Qualitative description of the amount of deviation of actual customer orders from forecasted orders. The ratio A/F described above is one way to quantify forecast quality. Forecast quality is best for A/F = 100%, and gets worse as A/F moves away from 100%.

Lead Time. The time between placement of an order to the vendors and receipt of the material.

On-Hand Inventory. All physical inventory that is owned by the enterprise. It is the sum of RPI, WIP, and FGI.

On-Order Inventory. Same as on-order material.

On-Order Material. The total amount of material for which orders are currently open and which will eventually be received from vendors. It increases each time a new order is issued and sent to the vendor, and decreases each time a shipment of parts is received from the vendor.

On-Time Delivery. Measures whether the order is delivered to the customer within the quoted availability. When described in units or dollars, it represents the units or dollar value of the deliveries that are delivered within the quoted delivery time. When described as a percentage it represents the percentage of on-time deliveries with respect to the total deliveries.

On-Time Shipments. Products that were shipped to customers within the quoted availability minus the transit time, that is, those shipped to arrive in time to satisfy the quoted availability.

Order Backlog. The total amount of outstanding orders from customers that have not yet been shipped. It increases each time a new order is received from customers, and decreases each time an order is shipped to customers.

Order-to-Delivery Time. The time period from order issue to order delivery at the customer site.

Order-to-Ship Time. Time period from order receipt to order shipment at the manufacturing enterprise.

Orders Delivered. Orders that have been delivered to customers.

Orders Shipped. Orders that have been shipped to customers.

Product Life Cycle. The general shape of the increase, leveling off, and decrease in order volume for the product. We assume here it is trapezoidal.

S and S-Plus. S is a language and interactive programming environment for data analysis and graphics developed at AT&T Bell Laboratories. S-Plus is a product version of S that is sold and supported by Statistical Sciences, Inc.

to trigger the start of the appropriate number of units in the current period.

Material planning uses the BOM to generate a material consumption plan for each part that can support the build plan. It then uses the material consumption plan, on-order material, RPI, RPI safety stock, and part lead times to determine the material ordering plan, that is, how much of each part to order in the current and future weeks. Details of the computation of the consumption and ordering plans are given in Appendix I.

Material ordering sends orders for the appropriate amount of each part in the current week to the vendors. As each order is sent, the on-order material for that part increases.

Raw material stores (not shown in the figures) receives and stores incoming material in RPI and provides material to production when requested. As it receives deliveries from vendors, it sends information about the shipment to on-order material which is reduced by the amount of the shipment received.

Production receives a build plan and requests as much material as required from raw material stores to build the number of units required. Only complete sets of parts are drawn from stores, that is, if one or more parts are not available in sufficient quantities, all parts are drawn partially. For example, if the build plan calls for 10 units to be built, and there are only 5 units of part A.3 and more than 10 units each of the other parts in RPI, only 5 units of of each part will be drawn and sent to WIP, and only 5 units can be started. The objective of raw material stores is to fill requests for material if possible, and not to maintain RPI at any particular level. The mathematical derivation of the number of units actually started subject to the available material is given in Appendix I.

Enterprise Modeling and Simulation Applications in Reengineering

Process reengineering as defined by Hammer and Champy in their book, *Reengineering the Corporation*,[1] is "the fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical, contemporary measures of performance, such as cost, quality, service, and speed." It is being applied at an increasing rate by three kinds of companies: those in deep trouble, those not yet in trouble but whose management has the foresight to see trouble coming, and those in peak condition with no discernible difficulties whose management is ambitious and aggressive. These three categories cover a large number of companies. The impact is on processes with throughputs measured in the billions of dollars.

Reengineering is pervasive, controversial, and disruptive, and has different interpretations. CSC Index, whose chairman is Champy,[1] states that even though they pioneered the practice of reengineering, they are startled by how widespread the phenomenon has become. Their survey results[2] based on 497 large companies in the U.S.A. and another 124 in Europe show that 69% of the U.S. companies and 75% of European companies are already reengineering (average completed or active initiatives in excess of 3). More than half of the rest were planning to launch an initiative over the next 12 months or were discussing one.

Hammer and Champy[3] mention three kinds of techniques that reengineering teams can use to help them get ideas flowing: boldly apply one or more principles of reengineering, search out and destroy assumptions, and go looking for opportunities for the creative application of technology.

A sampling of the literature reveals that redesign is influenced by the past experience of the reengineering team and the recommendations of reengineering consultants. Ultimately, many redesign decisions are made on speculation based on implicit mental models, convincing arguments by vocal proponents for change, sheer optimism, blind faith, or desperation.

A major concern is the uncertainty of predicting outcomes. Radical redesign and new ideas bring the possibility of boundless gain or tremendous loss. While assumptions are being searched out and destroyed ruthlessly, it should not be forgotten that some assumptions are rooted in scientific principles which cannot be ignored with impunity no matter how highly enthusiastic or motivated the reengineering team.

Enterprise Modeling and Simulation

Some areas suggested by Hammer and Champy[4] for reengineering the corporation include product development from concept to prototype, sales from prospect to order, order fulfillment from order to payment, and service from inquiry to resolution.

The Simple Model described in the accompanying article is a start towards addressing order fulfillment. Modeling and simulating the other processes on the list require different kinds of knowledge acquisition. For example, product development requires more knowledge about the R&D function, sales requires more knowledge about the marketing function, and service has not been considered in the current model, where the focus is on manufacturing.

The following paragraphs describe areas where enterprise modeling and simulation and the enterprise modeling and simulation system may provide value in the reengineering effort.

Identifying Processes

Hammer and Champy[5] suggest that once processes are identified and mapped, deciding which ones require reengineering and the order in which they should be addressed is not a trivial part of the reengineering effort. Typically there are three criteria for making the selection: dysfunction, importance, and feasibility.

Enterprise modeling and simulation provide one way of gaining insight in these areas by generating performance metrics with and without the change under different circumstances. For example, the Simple Model showed the importance of different controllable and uncontrollable factors to the different system performance metrics such as EOL inventory and order-to-delivery cycle times.

After selecting a process for reengineering, an understanding of the current process is crucial. It is necessary to know what the existing process does, how well (or poorly) it performs, and the critical issues governing its performance from a high-level view. This understanding is the prerequisite to redesign. The key is understanding the process rather than completely analyzing it in agonizing detail.

Enterprise modeling and simulation offer at least two ways of obtaining this understanding and possibly showing the cause of the dysfunction. First, the very act of building a consensus model that different people can agree with sheds light on what might not be working. Second, simulating the model will confirm or reject the validity of what is suspected. For example, after building the Simple Model, it was possible to test it in a large number of possible operating conditions to provide understanding of the cause and effect relationships. The first major insight from simulating the model was that what appeared to be a reasonable way of computing safety stock that would go to zero as demand went down actually gave rise to end-of-life inventory even though the demand was forecasted accurately. Enterprise modeling and simulation provide a way of gauging the relative impact of different process changes as a step towards selecting the appropriate subprocess to reengineer, and of quantifying the amount of prospective improvement.

Enterprise modeling and simulation can show the prospective impact of infeasible changes. In simulating the proposed reengineering changes, even if they are infeasible, the results will indicate if there is any promise in further consideration of a particular direction. For example, it is clearly not feasible to have zero build

The required material is drawn from RPI and goes into WIP where it remains for the duration of the build period. After that, the completed units go into FGI.

Target Safety Stock. Inventory is the amount of physical material, and ideally the enterprise would like to maintain it at or close to zero in RPI and FGI, and only carry it in WIP when raw material is being converted into final product. In practice, to reduce the effects on production of late vendor deliveries and customer orders coming in higher than forecasts, safety stock needs to kept. In the Simple Model, where vendor delivery time uncertainty is not an issue, to allow for the contingency that customer orders may come in higher than forecast, production planning targets the FGI safety stock to be two weeks of 13-week leading average forecast, and material planning targets RPI safety stock for each part to be the quantity of that part required for the production of the number of weeks specified in Table I(b) of the 13-week leading average forecast.

The 13-week leading average forecast at the end of a particular week in the future is the sum of the order forecasts over the 13 weeks immediately following the particular week divided by 13. This average anticipates trends 13 weeks (one calendar quarter) into the future, increasing as order forecasts increase, and decreasing as order forecasts decrease. In particular, the 13-week average forecast is zero at the end of the product life cycle, which means that any target safety stock expressed in weeks of 13-week leading average will aim for a zero target safety stock level at the end of the life cycle.

Having specified target safety stock in preparation for demands higher than forecasted, what is the impact if customers order exactly according to forecast? The expectation is that actual FGI should be equal to targeted FGI safety stock level, and actual RPI for each part should be equal to targeted RPI safety stock level for that part.

time for products and zero transportation times for shipments in the real world, but setting those values to zero in the model indicates the theoretical maximum benefits of these actions, and the magnitude of the results provides a data point for decisions on how much investment to put on driving these two times to zero instead of on other opportunities.

Furthermore, by showing the time behavior of the changes, enterprise modeling and simulation can show when actions can be expected to take effect. Inertia is a property of most systems, reflected in the time taken to respond to external influences or changes. Most physical systems are predictable in this respect, but the time behavior for organizational systems such as the enterprise is less predictable simply because it is not understood as well. Enterprise modeling and simulation help to increase the predictability of system behavior given that we know something about the system's structure and the behavior of its components. While immediate improvement for reengineering is the desired goal, enterprise modeling and simulation can show the length and causes of delays in obtaining the desired result.

Exposing and Challenging Assumptions

Hammer and Champy suggest that we question assumptions.[6] Enterprise modeling and simulation require assumptions to be stated explicitly during the model building process to reconcile differences in points of views. Challenges and disagreements on the validity are with respect to clearly stated assumptions rather than differences in opinions resulting from differences in mental models of different individuals. For example, the production planning and material procurement processes used in the Simple Model are expressed mathematically in Appendix I. If these are accepted as rational methods of planning, then there is no question or debate on the values of the outputs for a given set of inputs. If processes expressed mathematically are not acceptable as rational methods of planning and an alternative method is proposed, then that alternative method can certainly be tried, and the results compared with the previous method. The debate and challenge for improvement becomes one of improving the logic of planning rather than one revolving around the meaning of words and labels or one on how the model should behave based on past experience or speculation.

The approach advocated by Hammer and Champy suggests that changes be made by understanding the problem and devising the solution. This is central to modeling and simulation in addressing problems in the realm of the enterprise. Enterprise modeling and simulation offer a way of testing and verifying that given the current knowledge, the results of the simulation do not exhibit any obvious flaws before the process is implemented.

Role of Technology

Hammer and Champy devote a whole chapter to discussing the essential enabling role of information technology, and assert that modern state-of-the-art information technology is part of any reengineering effort. They caution that the misuse of technology can block reengineering altogether by reinforcing old ways of thinking and old behavior patterns, and that equating technology with automation does not result in reengineering.

We suggest that the application of enterprise modeling and simulation is a creative application of a well-understood technology to the processes of the enterprise. The technology of modeling and simulation has been applied to fields such as product design and the design of physical systems, but is only now beginning to be applied creatively in analyzing the processes of the enterprise. What enables the creative application of modeling and simulation is the tremendous increase in computational power. In this respect, we would like to suggest another rule along the lines of the rules described in reference 1.

> *Old Rule:* Decisions regarding process changes are based on mental models and analysis of historical data.
>
> *Disruptive Technology:* Enterprise modeling and simulation.
>
> *New Rule:* Decisions regarding process changes are based both on historical data and analysis of computer simulated behavior of explicit models with explicit assumptions that show the prospective consequences of different actions under a large number of operating circumstances.

Conclusion

Reengineering is a philosophy of renewal and rapid, discontinuous, and drastic change in the way corporate enterprises do their work, which brings with it uncertainty and fear of the unknown future. It is disruptive and controversial, and there is as yet no agreement that successes outnumber failures. During the implementation, "People focus on the pain of the present and the joy of the past. They forget about the pain of the past and the joy of the present."[7] However, given that it is occurring on such a wide scale, we suggest that application of enterprise modeling and simulation can increase the chances for success by (1) quantifying the potential benefits of the reengineered process in an explicit, defensible way, (2) illustrating the transition between the pain of the present and the joy of the future, and (3) showing the possible outcomes of current actions, thereby making the future more predictable and less surprising to those most affected by it.

References

1. M. Hammer and J. Champy, *Reengineering the Corporation: A Manifesto for Business Revolution,* Harper-Collins Publishers, Inc., 1993.
2. *State of Re-Engineering Report, Executive Summary,* CSC Index, 1994.
3. M. Hammer and J. Champy, *op cit,* p. 146.
4. M. Hammer and J. Champy, *op cit,* p. 118.
5. M. Hammer and J. Champy, *op cit,* p. 122.
6. M. Hammer and J. Champy, *op cit,* p. 145.
7. J. Kornbluth, "The Prophet of Pain," *Worth,* Vol. 3, no. 6, July/August 1994, pp. 80-81.

Simple Model Simulation

The Simple Model described above represents a simple process design for a manufacturing facility that is subject to simulation. On the surface, the design appears to be reasonable and adequate, and in fact is based on representative data and characteristics of the process. However, simulations will show some unexpected behavior, as well as the envelope of the possible behaviors.

The Simple Model was executed on an evolving system called the EMS system, which consists of two parts: the simulation engine part and the data analysis and display part. The simulation engine has continued to develop with each model that we have studied. It captures and abstracts processes in the enterprise. The simulation engine is an object-oriented, enhanced discrete event simulation software system.

The initial implementation of the simulation engine part of the EMS system was the Manufacturing Enterprise Simulator on the TI Explorer II.[9] The current implementation runs on HP 9000 Series 700 workstations at the Manufacturing Systems Technology Department of HP Laboratories. The implementation language is the Common Lisp Object System (CLOS).[18] The simulation engine has been implemented in CLOS provided by three different vendors: Franz, Inc.,[19] Lucid, Inc.,[20] and Harlequin, Ltd.[21] Models subsequent to the Simple Model (see page 90) were large enough to stress the limits of all three implementations. Graphical output was produced using S-Plus. Further details of the history and development of the EMS system are given in reference 9. The initial version of the Simple Model was implemented within a week based on the full order-to-ship model[22] (see page 90). It then took successive refinement and a tremendous amount of time to analyze the results.

For the reader familiar with discrete event simulation, details of the similarities and differences in concept between this implementation and conventional discrete event simulation are discussed in reference 9. In general, orders and shipments

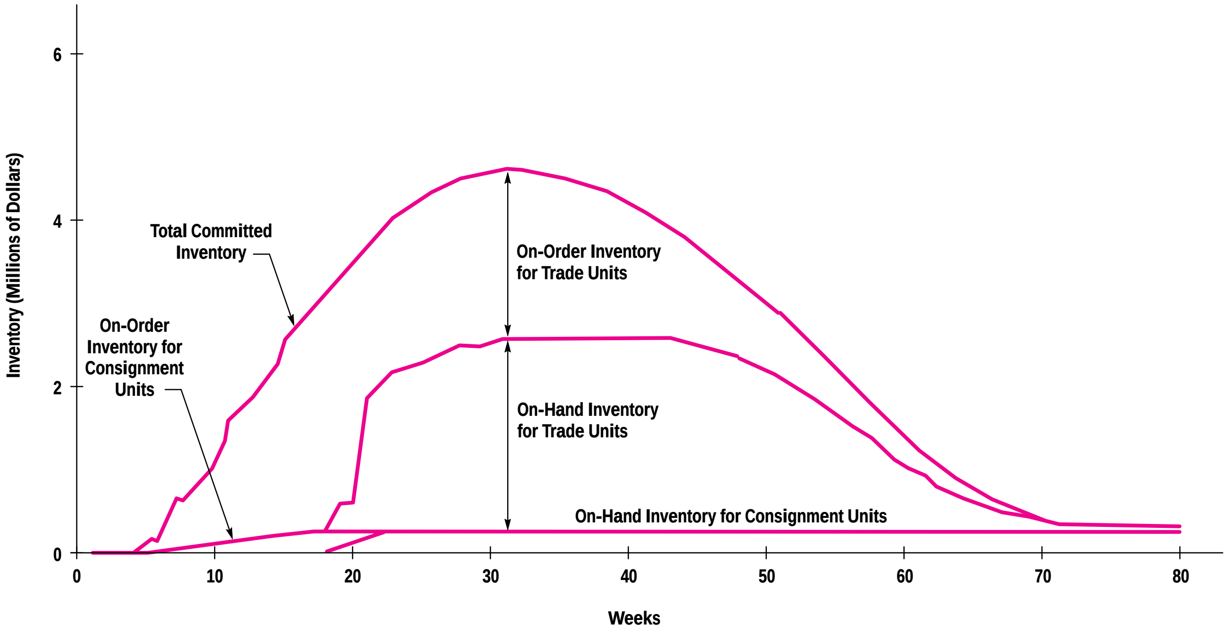


Fig. 4. Nominal case inventory components as functions of time. The experimental conditions are shown in Fig. 1.

are modeled as the entities of discrete event simulation. Backlog, on-order material, RPI, WIP, and FGI are modeled as queues. Customers and vendors are modeled as source-sink combinations of orders and material and vice versa. Production is modeled as an activity.

The production and material planning functions, which are essentially information processing and decision making functions, are implemented as mathematical models embedded in the simulation. The information generated by these planning functions determines when and how many units of product to start building and how many units of material to order. Thus, we can think of the Simple Model as an analytical mathematical model embedded in a discrete event simulation model. The analytical model (formulation given in Appendix I) dictates how the simulation model should behave in the same way as the planning functions dictate how operations should be handled in reality. The simulation model is the reflection of physical reality and reflects the behavior of the physical system that is told what to do.

There are two aspects of uncertainty: bias and variance. Most simulation models focus on variance and assume bias (offset) to be zero. While the EMS system supports the ability to simulate the model under stochastic conditions, in the runs described in this paper, variance is always zero and the emphasis of the analysis is on the situation in which bias can be nonzero.

Each run represents one combination of inputs and parameters of the system, and the traditional statistical analysis of means and confidence levels is not directly applicable for the analysis of these runs. While process variances are important considerations in a system, the motivation of this work was to identify the first-order effects of the various factors, considering the variances as second-order effects.

Details of the timing of the event sequence are shown in Appendix II.

Experimental Results

Experiment 0: The Nominal Case

The nominal case experiment assumes ideal conditions for testing the model. The purpose is to establish model baseline behavior and offer face validation by verifying that results are consistent with intuition and the observed behavior of the real system. Initial conditions for committed inventory and backlog are set to zero. A warmup period of five months (20 weeks) allows material to be ordered and received before customer orders arrive on week 21. The last customer orders arrive on week 68. Order forecasts are consistent with the trapezoidal profile already defined, and while they are generated weekly, they do not change from week to week. Week 21 corresponds to the first week of month 1, and week 68 corresponds to the last week of month L+6 in Fig. 2. Production begins during week 19 to ensure units in FGI at the end of week 21. The computation of FGI and RPI safety stock levels is assumed to apply only for weeks after week 20. Up to and including week 20, the required safety stock level is set to 0.

Time Response of On-Hand Inventory and On-Order Material. Fig. 4 shows inventory levels measured in dollar terms over time. The two bottom regions show the on-order material and on-hand inventory for consignment units. There is a gradual buildup of on-order material, which is rapidly transformed into on-hand inventory over four weeks, followed by a flattening out (since the consignment units are never shipped). The middle region shows on-hand inventory for trade or shippable units, which is the sum of RPI, WIP, and FGI. The upper region shows the on-order material commitment for trade units. The top surface of the graph shows how total material commitment changes over time.

Inventory Investment. Committed inventory at the end of week 20, before the first customer order arrives, is approximately

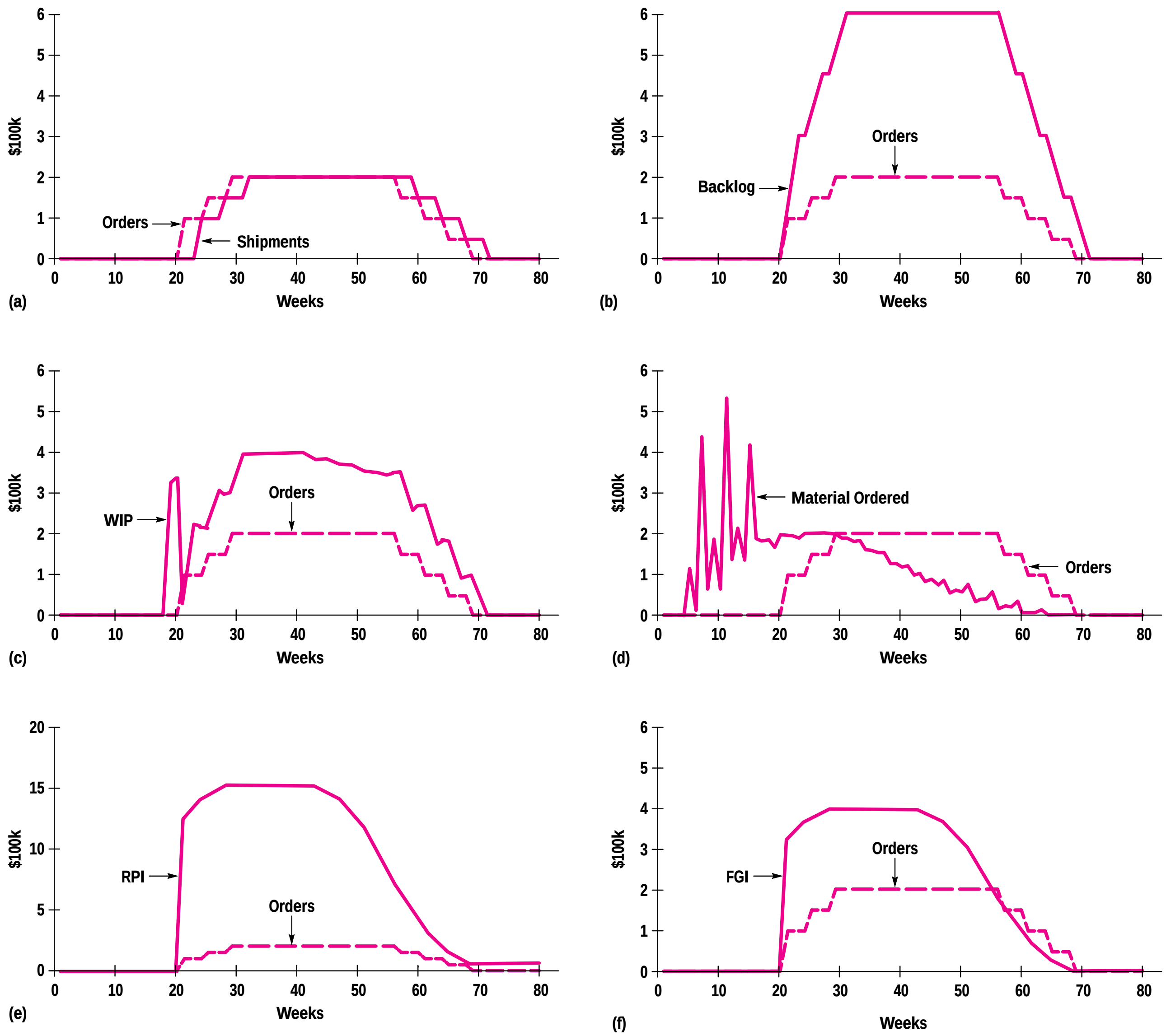


Fig. 5. Metrics as functions of time for the nominal case. (a) Shipments and orders. (b) Backlog and orders. (c) WIP and orders. (d) Material ordered and orders. (e) RPI and orders. (f) FGI and orders.

$3.5 million. If orders to vendors cannot be cancelled, this $3.5 million commitment must be disposed of if we decide to cancel the product before the first customer order arrives.

During the mature part of the life cycle of the product, the on-hand inventory is approximately $2.5 million and the total committed inventory is approximately $4.7 million. To support shipment levels of $200,000 a week requires $4.7 million of committed inventory (23.5 weeks of steady-state PCFT) and $2.32 million of on-hand inventory (11.6 weeks of steady-state PCFT), both of which include $300,000 of consignment units (1.5 weeks of steady-state PCFT). Details of the computations verifying these numbers in the simulation are given in Appendix IV-2. The maximum inventory exposure over the life cycle is $4.7 million.

The EOL consignment inventory of $300,000 reflects the amount of potential write-off because we did not dispose of the consignment units. The EOL nonconsignment inventory for trade units is reflected in the tail of the graph, and its value is approximately $64,000. If the material cannot be consumed any other way, there is an EOL write-off of $64,000 of nonconsignment inventory and $300,000 of consignment inventory for a PCFT of $7.8 million under ideal conditions of perfect forecast quality and on-time vendor delivery.

Time Response of Other Metrics. Fig. 5 shows other time series metrics in comparison to orders received. The shipment profile (Fig. 5a) is identical to the order profile but shifted in time by three weeks. This is because the four-week availability and one-week transit time require three weeks of order-to-ship time for on-time delivery.

Steady-state backlog (Fig. 5b) is $600,000, or three weeks of orders. Again, this is because the four-week availability and one-week transit time result in orders staying in backlog for three weeks, that is, the current backlog is the sum of the last three weeks of orders.

Enterprise Modeling and Simulation Research at HP Laboratories

Our work at HP Laboratories on enterprise modeling and simulation is an outgrowth of the factory modeling project, which began in early 1987. While we were working in the area of robotic automation for manufacturing, we began to appreciate the complexity of the geographically distributed, multientity marketing, manufacturing, and distribution operations necessary for HP to remain competitive. We also realized that there were very few tools available to help understand, design, and operate these complex systems.

Having been involved in product design with the evolving use of CAD and CAE tools, we thought that there was an opportunity of potentially tremendous magnitude for applying similar technologies to the design and operation of the factory and business systems used to market, manufacture, and distribute products. In an effort to capitalize on this opportunity, we began identifying the primary elements of a single factory and building our preliminary order-to-ship model that spanned all major activity from the receipt of an order to its shipment.

Preliminary Order-to-Ship Model

This early model was a vehicle to show the feasibility of applying simulation at a scope larger than a production line, where simulation was beginning to be applied. Developed and proposed for discussion purposes, it was a model to analyze why the order-to-ship time for some products stretched to weeks when the application of modern manufacturing techniques had reduced the build time to a matter of hours. More details on the reasons behind this work are given in references 1 and 2.

Full Order-to-Ship Model

By late 1988 the preliminary model was ready for testing in a real-world context. Data and operational information were provided by a real manufacturing division to help enhance our early model. This process helped to validate the preliminary order-to-ship model and led to the development of the full order-to-ship model.[3] The primary factors considered were order forecast quality, production capacity constraints, supplier lead times, and order filling policies. The primary metrics of interest were order lateness, backlog, and inventory. The model included three distribution centers, one manufacturing entity, and a centralized sales and order entry system. It was configured for one-level bills of materials (BOM), multiline orders, and long life cycle products.

The results of the analysis done with the full order-to-ship model were encouraging; they showed things that were consistent with real-world experiences (e.g., high forecasts led to high inventory and low backlog). The results also provided a view of greater potential by helping to identify areas for future improvement (e.g., the dominant cause of product shortages is long lead time parts coupled with poor forecasts rather than the build time).

While the results of this model were modest, the building and running of this model enabled us to explore some important technologies (i.e., Hierarchical Process Modeling for knowledge acquisition, a discrete event simulation language, SLAM II,[4] and a knowledge-based environment, Knowledge Craft, for system representation and building simulations). Our efforts led to generalized enterprise-level modeling elements and an object-oriented simulator. We also identified some new obstacles (e.g., managing large amounts of simulation data, extracting information) to be overcome in attaining our goals. More details are given in reference 1.

For about a year, no further model development was done, but rather, much effort was put into consolidating what we had learned about the modeling and simulation issues. This effort led to the complete overhaul of our modeling and simulation code while migrating it to the Common Lisp Object System on HP workstations. The power and speed of our system took a quantum leap forward.

Simple Model

With our improved system ready, we were presented with another real-world opportunity to apply our techniques. The Simple Model was proposed as a means of pulling together the main activities, processes and circumstances involved in a manufacturing enterprise. The primary purpose was to understand end-of-life (EOL) inventory and order delivery performance issues. The combined impacts of several environmental factors and operational policies were considered in the

Fig. 5c shows an initial spike in WIP preceding the start of orders by two weeks. This happens because the number of units started during week 19 is not only what is to be shipped two weeks later, but also the quantity that must be in FGI (approximately two weeks of orders) at the end of week 21. The WIP levels taper off downwards starting in week 44 towards the latter part of the life cycle because as the desired FGI safety stock level decreases, less production is required than is shipped because some units shipped from FGI do not have to be replenished.

Fig. 5d shows material orders. The three large spikes in material orders are caused by different lead times for parts to fill the targeted RPI safety stock at the beginning of the cycle. Each of the three small spikes corresponds to the different lead time parts for the initial WIP spike. Once the initial spikes are past, the material ordering volume is approximately the same height as the customer orders, except that it is shifted earlier in time, showing that once the system has reached mature demand, material inflow in the form of material ordered is balanced by the material outflow in the form of shipments. Material ordering starts ramping down beginning in week 28 just as the orders reach the maximum demand for this particular set of circumstances.

Fig. 5e shows RPI as a function of time. Notice that the vertical scale is different from the other graphs. The RPI level is 7.6 weeks of PCFT during the mature demand period and starts ramping down in week 44. Fig. 5f shows FGI as a function of time. The FGI safety stock during the mature demand week is two weeks of PCFT, which is the same as two weeks of steady-state orders. The FGI level starts ramping down in week 44.

Inventory Results. The results establish the baseline behavior of a system designed to take contingencies into account when those contingencies do not occur. Appendix IV provides further details for computing some of these results on a theoretical or common sense basis. Some interesting observations can be made. First, EOL inventory and write-off exist even though customers ordered exactly according to forecast and we expect safety stock to go to zero. Second, the level of inventory required to support this level of business can be quantified. Third, long lead time parts make up a greater percentage of the value of parts on order than their percentage in the product structure.

EOL inventory is important for short life cycle products because the inventory cannot be used for anything else and must be written off. In this case it is a result of the way of computing safety stock. It occurs if in the early part of the life cycle too much material is ordered because of high targeted FGI and RPI. For short life cycle products it can be a significant percentage of PCFT. EOL inventory is less of an issue for long life cycle products because the leftover inventory is generally a smaller percentage of total PCFT and excess inventory in early periods can be used at a later time.

analysis. The model, leveraging our earlier work, dealt with a one-level BOM, one factory, one product, and subsequently a family of successive products with common parts and overlapping life cycles.

Our analysis provided some interesting insights, such as certain material procurement and safety stock policies result in EOL inventory even for perfect order forecasts, and with low forecasts, increasing material lead times and planning frequency result in increased EOL inventory. More important, we began to realize that we were onto something that could really have a positive impact for HP. In fact, the business results led to the development of the planning calendar model with the Simple Model as its foundation. We also continued our technical enhancements by connecting the output to S-Plus[5,6] for data analysis and the creation of a Lotus® interface to display output.

Planning Calendar Model

The purpose of the planning calendar model[7,8,9] was to determine the effects of planning cycle times on inventory levels. It required extension of the Simple Model to include production planning and material planning cycle times. It approximated a two-level BOM and multiple assembly sites using a one-level BOM at one site. It used historical forecasts and orders. The primary factors were forecast quality, the length of the planning cycle, and the maximum lead times for parts. The primary metrics of interest were average inventory, delivery performance, and inventory levels at the start of production. The primary technical development was the application of S-Plus data analysis capabilities to the data.

With this model, material lead times had a dominant effect on inventory levels and committed inventory. Historically, forecasts were generally low, so for the historical data given, the planning cycle time used for the particular product had insignificant impact compared to material lead times. There was greater potential for reducing inventory by reducing lead times than by reducing planning time. Low forecasts increased backlogs.

Current Modeling Activities

We are currently finishing an analysis of a single-site manufacturing system where we were looking at how to improve the supplier response time. The challenges in this application include managing a multilevel bill-of-materials and understanding the consequences of long, variable test cycle times. We are also working with sector-level reengineering teams to help understand the consequences of proposed changes and explore alternatives.

Our enterprise modeling and simulation capabilities have evolved considerably from our preliminary order-to-ship model. However, there are still many more interesting challenges to address before we reach our goal of a computer-aided business process design and operation system.

Robert Ritter
Project Manager
Enterprise Modeling and Simulation Project
HP Laboratories

References

1. M.S. Mujtaba, “Simulation Modelling of a Manufacturing Enterprise with Complex Material, Information, and Control Flows,” *International Journal of Computer Integrated Manufacturing*, Vol. 7, no. 1, 1994, pp. 29-46.

2. M.S. Mujtaba, “Systems with Complex Material and Information Flows,” *Proceedings of the International Conference on Object-Oriented Manufacturing Systems (ICOOMS)*, pp. 188-193.

3. M.S. Mujtaba, *Formulation of the Order-to-Ship Process Simulation Model*, HP Laboratories Technical Report #HPL-92-135, December 1992.

4. A.A.B. Pritsker, *Introduction to Simulation and SLAM II, Third Edition*, Systems Publishing Corp., 1986.

5. *S-Plus Programmer’s, User’s, and Reference Manuals*, Statistical Sciences Inc., 1992.

6. A.A. Becker, J.M. Chambers, and A.R. Wilks, *The New S Language*, Wadsworth & Brooks/Cole Advanced Books & Software, 1988.

7. C.M. Kozierok, *Analysis of Inventory and Customer Service Performance Using a Simple Manufacturing Model*, Master of Science Thesis for Leaders for Manufacturing (LFM) Program, Massachusetts Institute of Technology, May 1993.

8. K. Oliver, *Simple Model Report*, distributed by email on January 12, 1993.

9. M.S. Mujtaba and R. Ritter, *Enterprise Modeling System: Inventory Exposure and Delivery Performance*, HP Laboratories Technical Report #HPL-94-89, October 1994.

This nonzero EOL inventory is significant because our safety stock policy targets zero safety stock levels in FGI and RPI at the end of the life cycle. Having observed this phenomenon in the simulation, we were able to show mathematically why the EOL inventory is not zero. The formal derivation of this result is outside the scope of the current paper, but more detailed analysis of the data showed that it is the Class C parts that are left over. The Class C parts will be zero in the case when orders come in as forecasted for the conditions of experiment 0 only if the target RPI safety stock for Class C parts is less than or equal to the 13-week leading average forecast. Also, for the conditions of experiment 0, any part with target safety stock greater than 13 weeks of 13-week leading average forecast will end up with EOL material. The behavior of the amount of Class C EOL material as the number of weeks of target safety stock goes down is given in Appendix IV-3, and an informal explanation showing the reasoning behind the EOL material is given in Appendix IV-4.

The nonzero EOL is a function of the number of weeks of 13-week leading average forecast. Other techniques of computing safety stock, for example using a cumulative leading forecast rather than the 13-week leading average forecast, might lead to different results.

Smoothing WIP and Production. The initial spike in WIP shows how the policy of starting production in week 19 (and not before) gives rise to a spike in capacity demand at the beginning of the product cycle. It could be eliminated by incorporating production capacity constraints into production and material planning or by allowing FGI to build up before the first order comes in (i.e., before week 21). Both of these require production to start before week 19.

Experiment Set 1a: Single Uncontrollable Factor Variation

In the nominal case, the customer order pattern was accurately forecast. We now consider the situation where the actual orders are different from the forecasts.

We assume that customers order according to a constant order forecast profile multiplied by some constant factor Actual/Forecast or A/F. A/F is the ratio of actual orders to forecast orders; its definition is shown in Fig. 6a. In practice, marketing would change the forecasts periodically. Since we were not modeling the forecasting process, we chose the simplifying assumption that although a new forecast is generated every week, it is identical to the forecast generated the previous week.† Here is an example of bias in the order forecast with no variance. The model interpretation is that although estimates were wrong in the past, we expect that future orders will be equal to the original forecast. This is

† This is not a limitation of the model. A user-specified forecast can be accepted by the model. Later models have incorporated historical forecasts. The reason for this assumption was to get a better understanding of the effect of forecast bias. Fluctuating forecast deviations make interpretation harder.

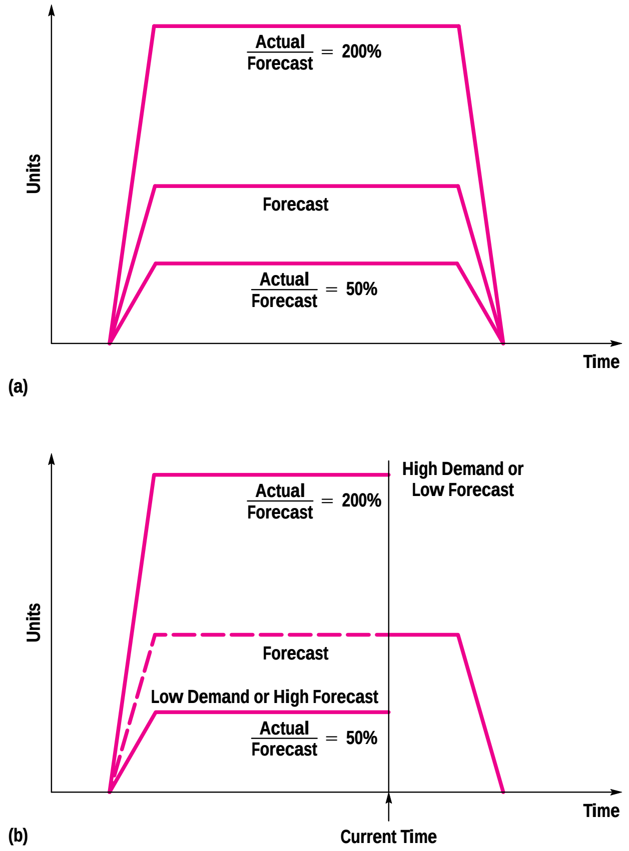


Fig. 6. Definition of A/F. (a) A/F ratio. (b) Actuals and forecasts at the current time.

reflected in Fig. 6b. Actuals came in as shown in the part of the graph to the left of the current time, while the part to the right of the current time line shows the current expectation of future orders.

Clearly, we would expect an effect when A/F is not 100%. If A/F is less than 100%, that is, if forecasts are high, FGI will start to build up, since production planning has directed a larger number of units to be built than are subsequently demanded. Production planning and material planning take this into account and plan to build less and order less material in the future, but the overall material level is higher than when A/F is equal to 100%. On the other hand, if A/F is greater than 100%, that is, forecasts are low, FGI will start to be eaten away because production planning has directed a smaller number of units to be built than are subsequently demanded. Subsequently, production planning and material planning take this into account and raise the production, but since they are always estimating low future demand, we would expect the inventory level in general to be lower than in the case where A/F is 100%. Surprisingly, this intuitive result does not hold, as will be seen later.

We ran the simulations with A/F ranging from 50% to 200% at equal intervals of 25%. In addition, we ran it at smaller intervals in the region of 95% to 125%.

EOL Write-off. A consequence of keeping forecasts identical for all runs is that the consignment profile does not change with respect to A/F. Fig. 7 shows EOL metrics as A/F ranges from 50% to 200%. Note that the changes in value are not constant across the horizontal axis. Fig. 7a shows that total EOL inventory increases as A/F decreases. Fig. 7b shows that the percentage impact is even worse, simply because the write-off is a higher percentage when PCFT, which is directly influenced by A/F, is lower. For low forecasts, that is, A/F greater than 100%, the EOL inventory decreases. For high forecasts, that is, A/F less than 100%, the EOL inventory increases. The lower the A/F, the higher the EOL inventory.

Fig. 7 leads to the obvious conclusion that inventory write-off can be reduced by the strategy of underforecasting orders. However, this is only one side of the story. The complete story is shown in Fig. 8.

Impacts on Time Series of A/F Changes. Fig. 8 shows the impact of A/F changes on different time series measures. *To avoid clutter we will not show inventory for consignment in subsequent time series.* FGI, WIP, RPI, on-order material, and on-hand inventory will refer to the material associated with trade units unless otherwise specified.

All of the graphs in each row of Fig. 8 exhibit identical behavior before week 21. This is to be expected, since before the first orders come in on week 21, the situation is the same for all cases. Only as different amounts of orders come in on or after week 21 is the situation different for different values of A/F.

Fig. 8a shows the order forecasts and actual orders for reference. The ratio of the values of the two lines at any time in the graph is equal to A/F.

Fig. 8b shows the backlog and actual orders time series on the same scale. Notice how the backlog increases spectacularly as A/F goes beyond 125%. Fig. 8c, which displays backlog in terms of mature demand, shows that for an A/F value of

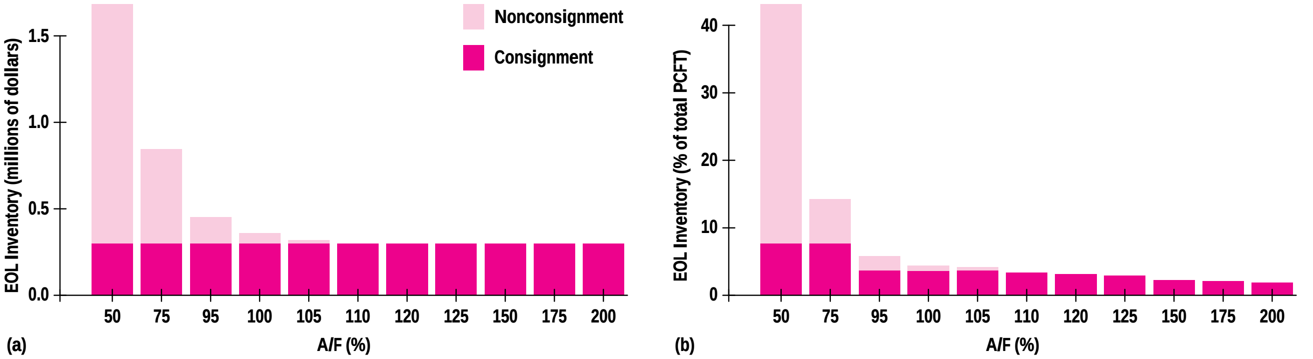


Fig. 7. EOL (end-of-life) inventory for experiment set 1a.

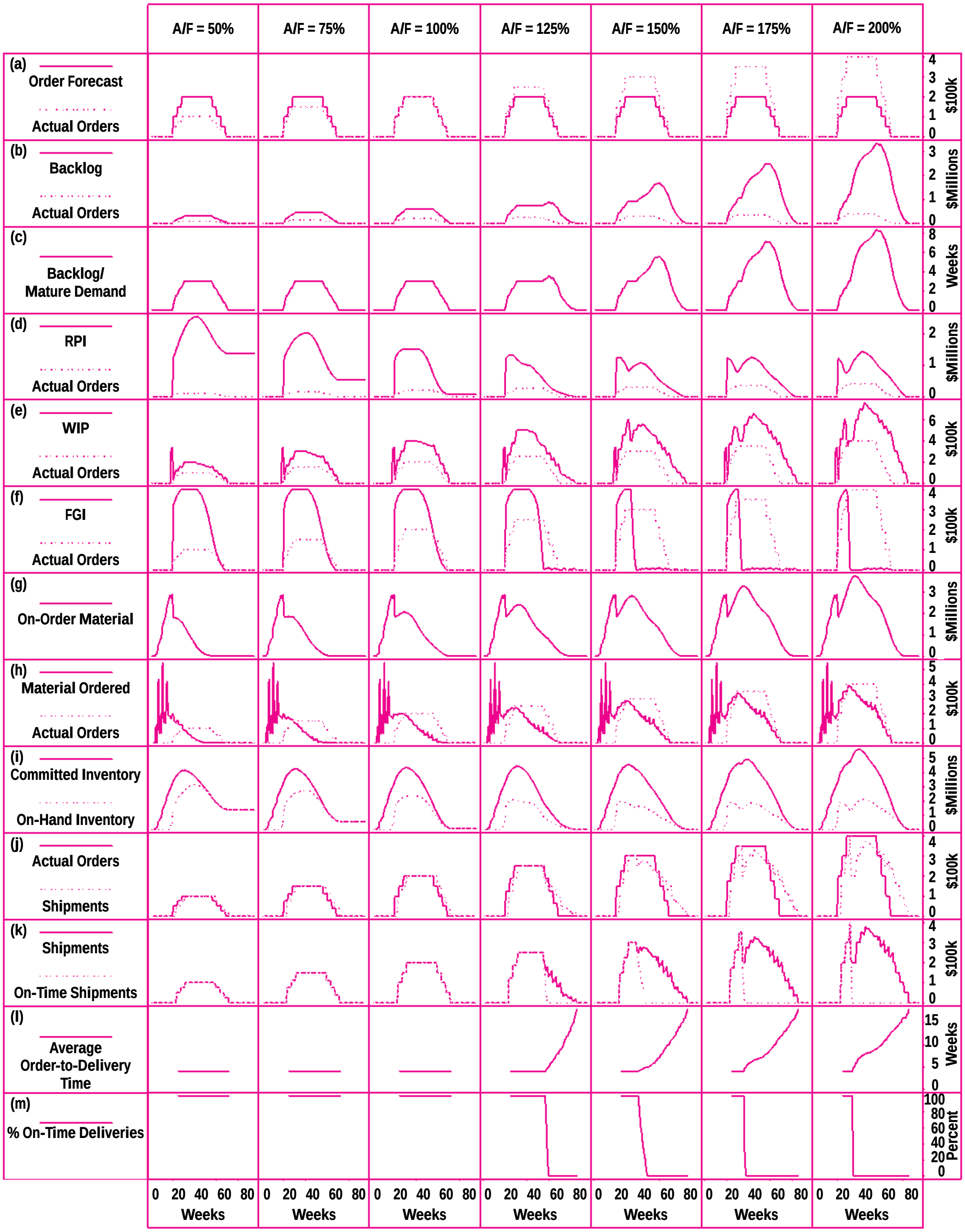


Fig. 8. Time series data for various values of A/F for experiment set 1a.

200% the backlog can be as much as eight weeks of mature demand. Backlog measured in terms of weekly mature demand is constant for low A/F. It increases for high A/F because products cannot be shipped as fast as orders come in.

Fig. 8d shows that the EOL RPI level falls as A/F increases. In addition, the general level of RPI as a function of time falls as A/F increases until A/F is greater than 150%, when the RPI level actually appears to rise as A/F increases. The reason is that because of shortages we order more of all material to build the shortfall in units. The short lead time parts show up first, but cannot be used because of a shortage of the long lead time parts with minimal safety stock. An analysis of the results shows that the critical part is A.3.

Fig. 8e shows that the WIP profile increases as A/F increases. This is expected, since WIP is directly related to the shipments flowing through the system, and the shipments are directly related to orders, which are directly related to A/F.

Remember that this is true only when the production capacity constraint is not reached. If production capacity is only a little greater than forecast, high demands would result in the level of WIP being capped at some limit but spread out over time.

Fig. 8f shows that the FGI level is identical for all values of A/F less than 100%. For A/F greater than 100%, the FGI gets eaten away slowly because the rate of replenishment of new units does not keep up with the shipments because of underforecasting. However, since FGI safety stock levels are based on two weeks of 13-week average forecast and the forecasts used are identical in all the experimental runs, the peak FGI tends to be the same.

Fig. 8g, on-order material, shows initial large spikes for material for RPI and FGI safety stock, followed by a drop after the material for safety stock has been delivered. Subsequently the profile shows an increasing level over time as A/F increases.

Fig. 8h, material ordered, shows the same spikes before week 21 that we have seen before. Again the material ordered versus time increases as A/F increases.

Fig. 8i shows that, in general, committed inventory after week 21 is higher for higher A/F and stretches out farther over time. For lower A/F the committed inventory is lower in the early part of the life cycle, but there is an increase in EOL inventory.

Fig. 8j shows that for A/F less than 100%, shipments follow the order stream nicely. High A/F (high demand) values cause the initial orders to be filled as specified, but subsequently shipments drop off and then catch up. The product shipment over time is smooth when A/F is less than or equal to 100%. When A/F is greater than 100%, during the early part of the life cycle the orders are filled as they come in. As the FGI safety stock is consumed, the shipments fall to the forecasted levels, and then subsequently tend to rise to the actual order levels.

The on-time shipment graphs in Fig. 8k show that initial orders are delivered on time in all cases. For A/F less than 100% (forecasts are high), all orders are delivered on time. For A/F greater than 100% (forecasts are low), initial orders are delivered on time, but subsequent orders are late. As A/F increases beyond 100%, both the percentage and the total dollar value of on-time shipments (and consequently deliveries), go down, and the late orders never catch up. On-time delivery graphs, which are not shown, would be identical to on-time shipment graphs shifted by one week.

As expected, because of the policy of shipping as late as possible, Fig. 8l shows that average order-to-delivery time never goes below four weeks, but increases with time up to 18 weeks as A/F increases to 200%. Fig. 8m, showing the percentage of on-time deliveries, is consistent with Figs. 8k and 8l in terms of on-time deliveries.

How Late Are Late Orders? How late are the late orders and how many orders are delivered on time (namely, within four weeks of being ordered)? These questions are answered in Fig. 9, which shows the dollar volume of deliveries and the order-to-delivery time. For A/F less than or equal to 100% (forecasts high or demand low), all orders are delivered on time. For A/F = 105%, most orders are delivered on time. For A/F = 150% and 200% (forecasts low), some orders are delivered on time, and a large fraction of orders are delivered late. Furthermore, for high A/F values, even though the total volume of shipments is higher, the amount of on-time shipments and deliveries actually goes down. Some orders are delivered as much as 14 weeks late, that is, 18 weeks after receipt of order. This 14 weeks is the upper limit of lateness for this particular model and data configuration. No matter how high A/F gets, orders will never be later than 14 weeks. The explanation for this is given in Appendix IV-5.

Interpretation of Results. In this model, forecasts were not updated on the basis of orders. In reality, when orders are very much under or over forecasts, there will be pressure to change the forecasts. If further information on the forecasting process is available, this can be incorporated into the model. Another study that could be done is to see what happens if we treat the initial orders as early indicators of the whole life cycle, that is, after some period of time, we revise the forecasts so that they more closely represent the volume of actual orders. On the other hand, if the life cycle is very short, it may turn out that revising the forecasts when the first orders come in may not have an impact on system response. We have established a nominal trapezoidal product life cycle, but this could be changed in various ways. It could be stretched out horizontally to increase the life cycle (as is done in subsequent experiments), or vertically, to show a higher level of product demand.

Customers need to receive the products within a reasonably short time, or they might cancel the order. For the model, we assume that customers are willing to wait patiently as long as it takes for the manufacturing facility to produce and ship the products, and that they will not cancel the order.

The purpose of this detailed discussion is to show how changing the one factor, A/F, can have different impacts on different metrics, and how this might affect different parties interested in the outcomes. A/F is partly under the control of customers, and partly under the control of marketing, assuming that greater effort will provide a better estimate of orders. It shows that if A/F is low, order processing and shipping would have excellent performance metrics in getting products out in a timely fashion, whereas material procurement would be in the situation of trying to explain why there is so much material in the plant, and marketing and the plant manager may have to explain why orders are below target. On the other hand, if A/F is high, customers

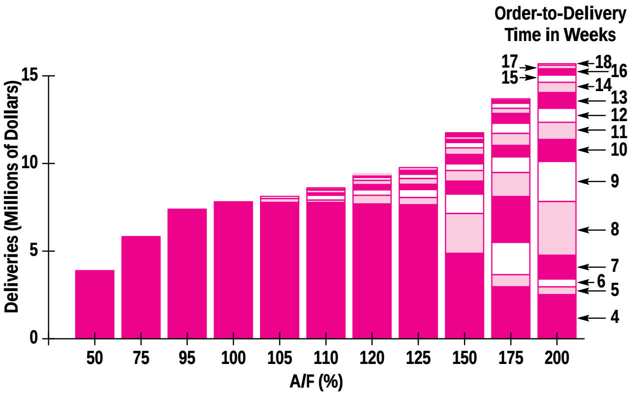


Fig. 9. Deliveries by order-to-delivery time in weeks for experiment 1a.

Table III
Range of Values of Factors for Different Experiment Sets

Factor Description	Parameter Name	Number of Different Values	Values
Actual/Forecast, %	A/F	11	A/F (%) = 50,75,95,*100*,105,110,120,125, 150,175,200
Part Safety Stocks Class A 4K weeks Class B 8K weeks Class C 16K weeks	K	5	K = 0.5,0.75,*1.0*,1.5,2.0
Life Cycle, L+6 months	L	5	L = 0,3,*6*,12,18
Availability, weeks	Y	5	Y = 1,2,*4*,8,12

Percentage value of 6, 10, and 14-week lead time parts in the product (r%,s%,t%)	lt	4	rrr = (100,0,0),rst = *(25,40,35)*,
			sss = (0,100,0), ttt = (0,0,100)

Experiment 0 (nominal case): Values shown in italics.

Experiment Set 1a (uncontrollable factor A/F varied): Values of A/F varied as shown. Values of other factors same as experiment 0.

Experiment Set 1b (A/F = 100%, controllable factors varied): Values of factors other than A/F varied as shown in turn. Values of other factors same as in Experiment 0.

Experiment Set 2 (dual-factor experiments): Values of factor A/F and one other factor varied in turn. Values of other factors same as in Experiment 0.

Experiment Set F (all factors varied): Values of all five factors varied as shown.

will be screaming for products, order processing and shipping will be trying to placate angry customers, production will be under pressure to put out products faster, and material procurement will have to explain the perpetual shortage of raw material A.3 while other material is piling up.

Experiment Set 1b: Controllable Factors Varied with 100% A/F

We next look at the effect of changing the factors over which the manufacturing enterprise has some control. In the single-factor experiments, the variation of each factor is summarized in Table III. Except for the set of runs where A/F varied as in experiment 1a, A/F was set at 100%.

Changes in safety stock levels can be characterized in many ways—for example, for each part individually. We chose to multiply the safety stock levels of experiment 0 by a constant multiplier K whose value ranged from 0.5 to 2.0. Life cycle lengths were changed by using values of L to result in life cycle lengths L+6 between 6 and 24 months.

Availability Y was varied from 1 week to 12 weeks (it cannot be less than 1 week because of the 1-week shipment transit time). Y = 1 requires off-the-shelf delivery and implies a total build-to-forecast strategy. As Y increases, the production strategy shifts from build-to-forecast to build-to-order. From prior considerations, an availability Y of 18 weeks will result in on-time delivery of every order regardless of forecast quality.

While there are different ways to characterize modification of part lead times—for example, changing it for each part—we chose to change part lead times by changing the percentage of parts with lead times of 6, 10, and 14 weeks to be 100% in turn.

EOL Results. The EOL inventory graphs for A/F = 100% are summarized in Fig. 10. EOL inventory increases as safety stock increases; the results are consistent with experiment 0. When K is 0.75, we carry 12 weeks of C parts and there is no EOL RPI. When K is 1, we carry 16 weeks of C parts and end up with EOL inventory of C parts. When K is greater than 1, EOL RPI increases. When K is 2, we carry 16 weeks of B parts and EOL RPI includes both B and C parts.

Fig. 10b shows that product life length has no impact on EOL inventory. This is to be expected in the model because increasing L stretches out the middle portion of the time series graphs, and the behavior towards the end of life tends to be the same in all cases when L increases (illustrated in a future graph, Fig. 11b). For short L, the effect of the rising demand in the beginning of the life cycle affects the behavior at the end of the life cycle. Fig. 10c shows that as availability Y is shortened, EOL inventory increases, that is, quoting shorter lead times to customers exposes us to more risk of EOL inventory. This is intuitively correct; the longer the quoted availability, the longer we can afford to wait before ordering material.

Part lead time has no impact on EOL inventory when A/F = 100% (Fig. 10d).

Other Results. Fig. 11 shows the inventory measures over time as different factors are varied. Delivery performance is not shown because for A/F = 100%, delivery is always 100% on time.

Fig. 11a shows the inventory measures over time as a function of raw material safety stock multiplier K. The heights of the three initial spikes for material orders increase as K increases, directly impact RPI and on-order material, and indirectly impact on-hand and committed inventories. In general, the higher the K, the higher the inventory levels, including EOL inventory, which is the tail of the committed inventory graph. The on-order material level before the start of production increases as K increases. Keeping all the other factors constant, there is no change in backlog or delivery performance, and these are not shown in Fig. 11a.

Fig. 11b shows the inventory measures over time for varying the product life cycle by changing L from 0 to 18 months. This is one of the less interesting graphs, shown here for

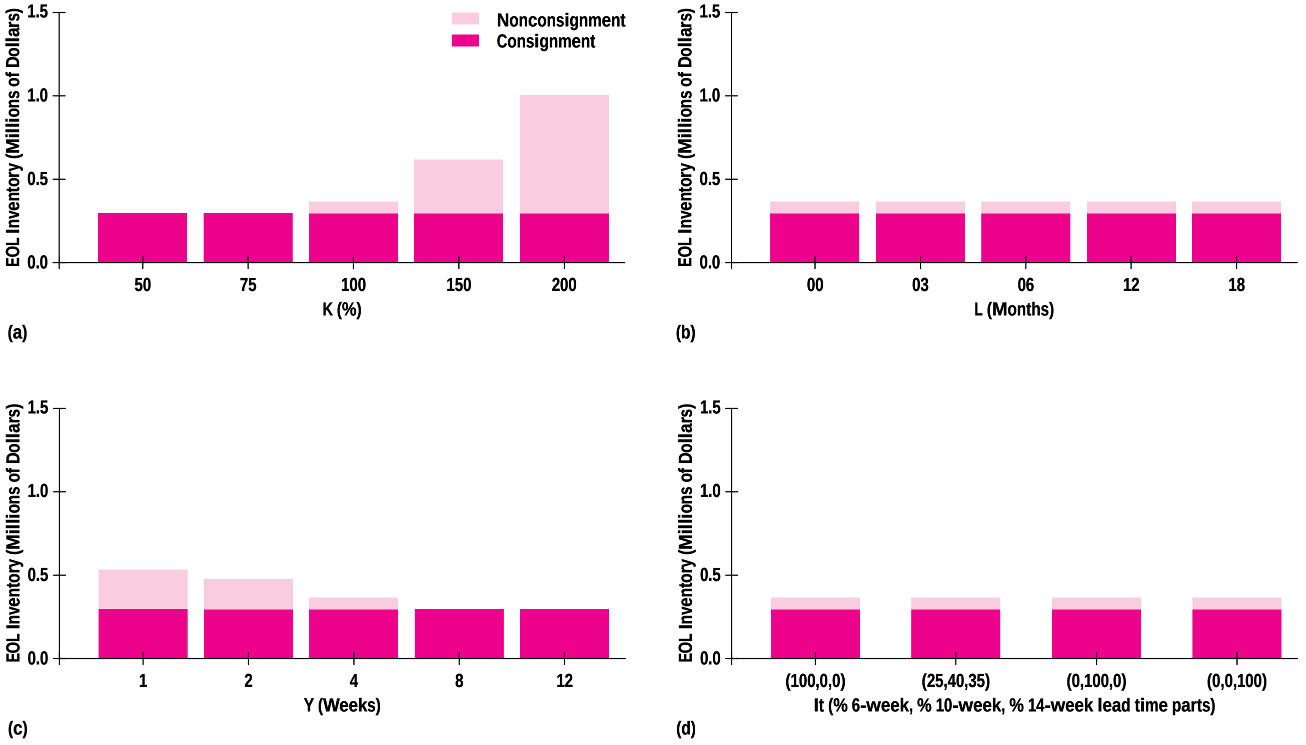


Fig. 10. EOL Inventory by single-factor changes with A/F = 100% for experiment set 1b. (a) Effect of material safety stock (5 runs). (b) Effect of product life (4 runs). (c) Effect of availability (5 runs). (d) Effect of lead time (4 runs).

completeness. EOL inventory is the same in all cases. However, because total PCFT increases, EOL inventory is a lower percentage of PCFT as L increases.

Fig. 11c shows the inventory levels over time for varying quoted availability Y. As Y increases, after the same three initial spikes, the amount of material ordered gets delayed, and the on-order material graphs get stretched to the right. The committed inventory graphs are also stretched into the future. The committed inventory is lower and the EOL inventory (tail of the committed inventory graph) tends to decrease. The delivery profiles are shifted out into the future and the backlog levels are higher.

Fig. 11d shows the time responses of the inventory metrics as part lead times vary. Notice the change in shape of the material ordered graphs. For lt = (25,40,35), there are three large and three small spikes, whereas for the other cases, there is one large spike and one small spike. As lead time increases, the material needs to be ordered earlier. On-order material increases as the lead time increases. On-hand inventory does not change. There is no impact on EOL inventory, order backlog, or on-hand inventory (RPI+WIP+FGI) as long as A/F remains constant at 100%.

Interpretation of Results. This set of results shows how each organization in the manufacturing enterprise can improve its performance metrics assuming that it relies on the forecasts given as being accurate and does not try to second-guess them. For example, if material procurement is under pressure to lower inventory levels, it would naturally try to reduce K. On the other hand, order processing and shipping would prefer to reduce Y to reduce having to deal with impatient customers.

Experiment Set 2: Dual-Factor Experiments

In this experiment set, we varied two factors in combination and attempted to observe the effects. However, instead of looking at all combinations, we looked at the impact of each of the other factors when A/F changed. This enabled us to see the effect of the controlled action in various situations of customer ordering behavior.

Results of Two-Factor Experiments. Fig. 12 summarizes the information on EOL and on-time deliveries as A/F and other factors are varied. Fig. 12a shows that as K increases, there is higher exposure to EOL inventory as A/F decreases. However, increasing K in general gives better delivery performance by shortening the average order-to-delivery time as A/F increases above 100%. Below an A/F value of 100%, K does not have an impact on the already excellent delivery performance shown by 0% late deliveries.

Fig. 12b shows that as L increases, the total shipments for a given A/F increase. For long L, the absolute volume of on-time deliveries initially increases as A/F increases. As A/F keeps on increasing past 100%, the absolute volume of on-time deliveries decreases. The average order-to-delivery time is not affected very much by L, and the EOL inventory is impacted insignificantly. The absolute amount of EOL inventory seems to depend little on L except when L is 0. For L = 0, the long lead time and high safety stock parts may actually cause most of the material for life cycle use to be ordered before the first customer order is received. The percentage of EOL writeoff decreases for a given A/F as L increases, reflecting the fact that the EOL writeoff is a smaller percentage of the total shipments as total shipments increase.

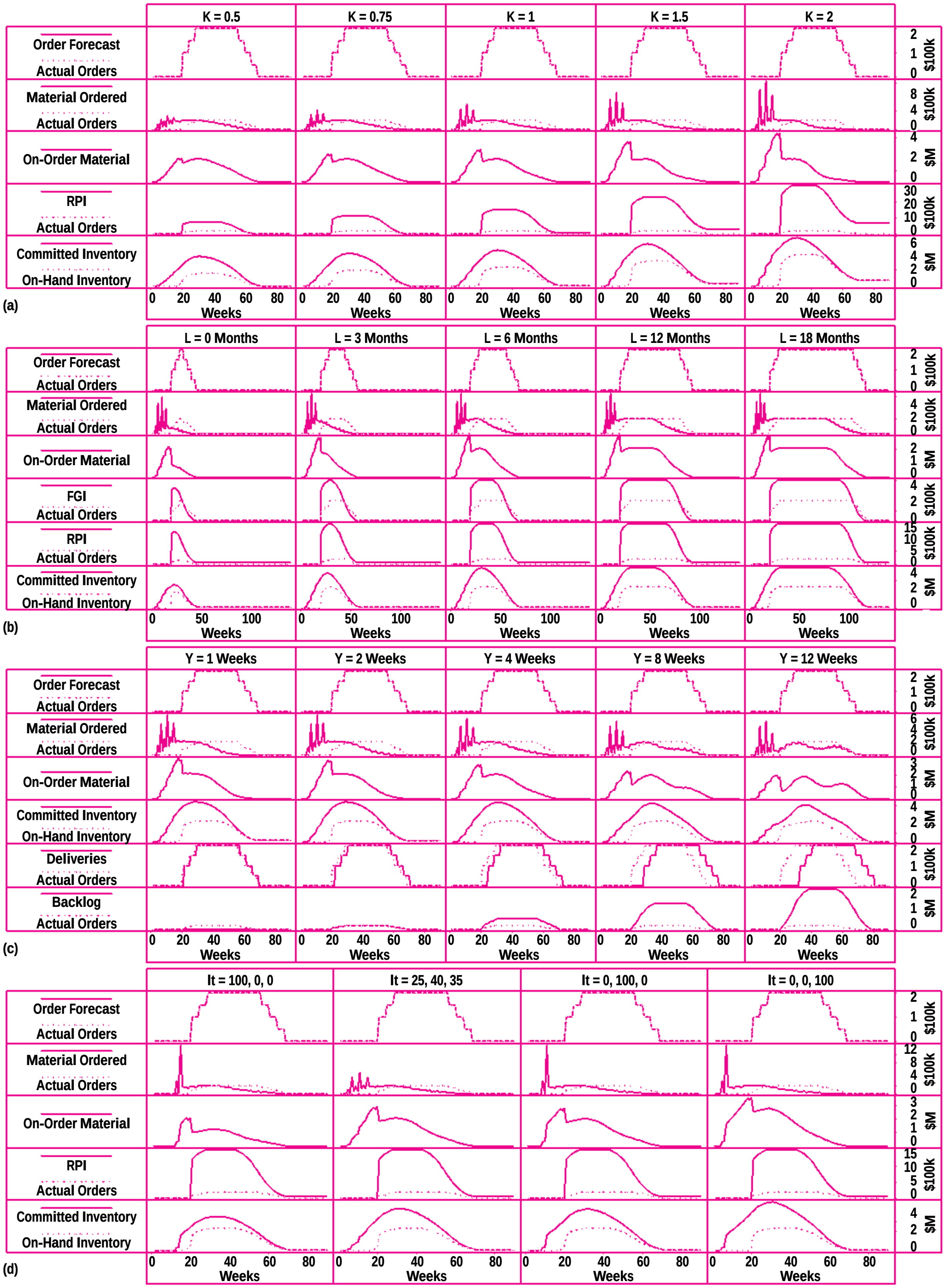


Fig. 11. Inventory measure time series for experiment set 1b: varying different factors with A/F = 100% (order forecast graph coincides with actual orders graph). (a) Varying safety stock levels (5 runs). (b) Varying life cycle (5 runs). (c) Varying availability (5 runs). (d) Varying lead time (4 runs).

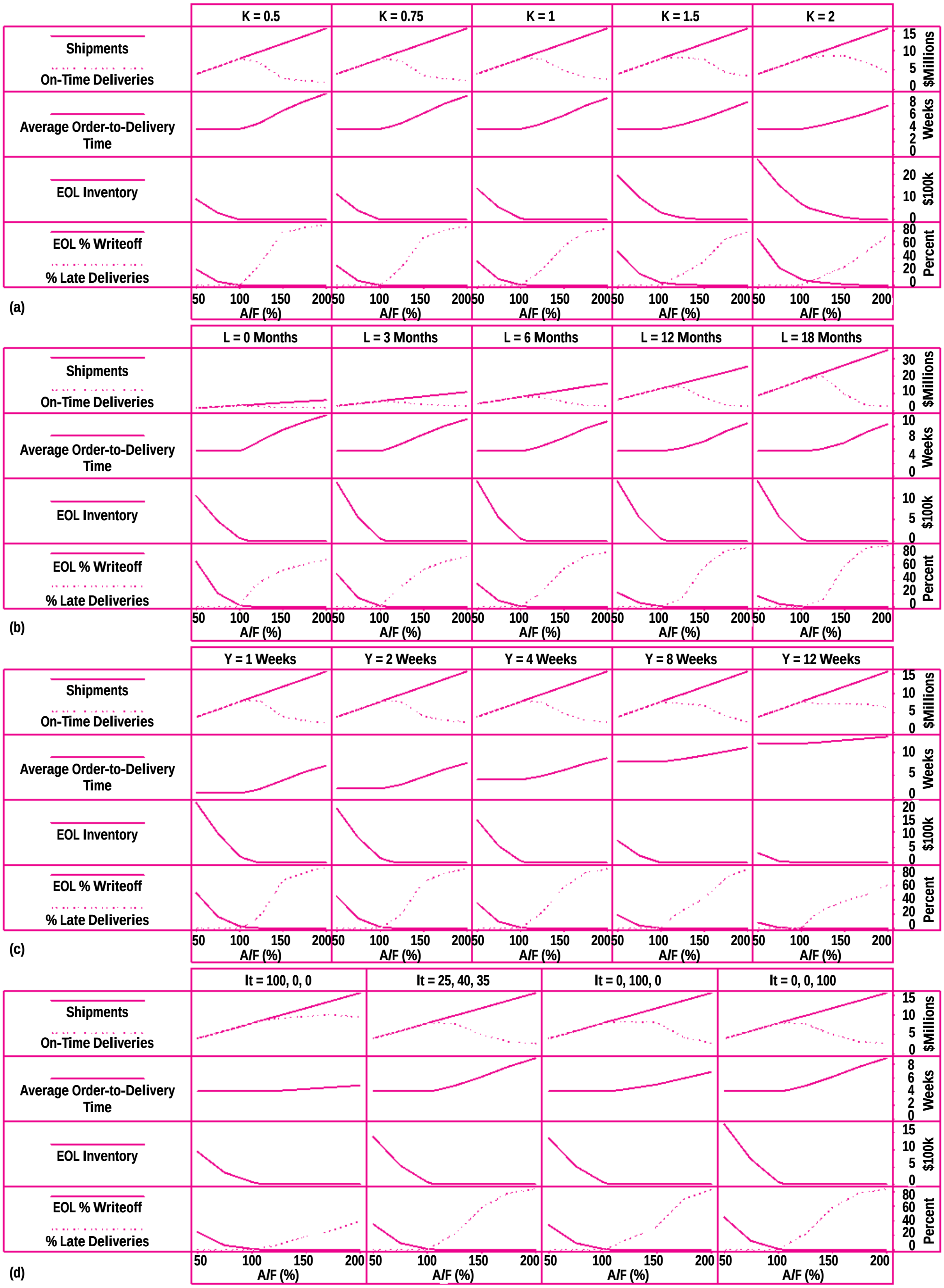


Fig. 12. EOL and shipment metrics as functions of A/F for experiment set 2 as each of the other factors is varied. (a) K varied. (b) L varied. (c) Y varied. (d) lt varied.

Fig. 12c shows that increasing Y is desirable for reducing the percentage of late deliveries and reducing EOL inventory, but that the average order-to-delivery time increases, resulting in customers waiting for long periods of time, which in practice might lead to possible order cancellations. When Y = 1, the worst average order-to-delivery time is lower than the best average order-to-delivery time when Y = 12 weeks. This is an example of a situation in which trying to reduce late deliveries by quoting a longer lead time actually leads to longer average delivery times and possibly lower customer satisfaction.

Fig. 12d shows that if all other things are kept constant, longer vendor lead time leads to poorer performance when A/F is greater than 100% and increased EOL exposure when A/F is less than 100%. For lt = (100,0,0), that is, lead time for all parts is 6 weeks, A/F has little impact on average order-to-delivery time over the given range. Furthermore, the percentage of late deliveries is generally lower than for the other values of lead time. If Y could be set to 12 weeks for the case lt = (100,0,0), no orders will ever be late, regardless of the value of A/F. Applying reasoning similar to that on page 82, the policy of waiting for customer orders to arrive before we order parts could lead to an order-to-delivery time of nine weeks, which is shorter than 12.

Observations. We have looked at the interactions of A/F with the other factors in our experiments and noticed the complexity of the interactions. The results of experiment set 2 show the impact of uncertain customer behavior on various organizations within the enterprise. In an uncertain world where A/F is outside our control, it would appear that increasing K and L, reducing lt, and increasing Y would increase on-time deliveries, which is desirable from the point of view of the manufacturing enterprise. However, increasing Y will tend to increase order-to-delivery times and backlog volumes, which could potentially lead to poorer customer satisfaction and high backlogs for order processing and shipping to deal with.

The other problem of taking these actions is that while delivery performance for the enterprise improves in general, different people and organizations are responsible for influencing and setting the values of K, Y, and lt and obtaining the reward of improved metrics. Increasing K results in better availability but increased write-off, especially if A/F is below 100%. One individual owns K, another individual owns Y, the vendors and R&D together determine lt, marketing owns L, and customers determine A/F. Any one of these can influence the other measures unilaterally, so it is necessary to coordinate the efforts of increasing some parameters and reducing others simultaneously. For example, material procurement could reduce K on the assumption that it will reduce RPI, committed inventory and EOL inventory, and this would be correct if A/F were 100%, but if A/F came in greater than 100%, the overall delivery performance would be poor. On the other hand, if R&D chose longer lead time parts because vendors demanded a premium price for short delivery times, EOL inventory would tend to be higher regardless of what value of K was chosen by material procurement. If quoted availability Y were reduced from 4 weeks to 1 week, inventory levels would tend to go up.

We could also consider the effects of the four other factors on one another, and that would give rise to another six combinations. These discussions are outside the scope of this paper.

Experiment Set F: All Factors Varied

In experiment 0, we looked at the results of one simulation run. In experiment 1, for each factor we looked at four to eleven runs. In experiment 2, we looked at 44 to 55 runs for each combination of A/F and the other factor. As we study the effects of multiple factors, the number of runs increases exponentially. Complexity increases not only in terms of number of simulation runs considered but also the way in which we analyze the data. A full factorial experiment, that is, one in which all the factors are varied in all combinations given here, requires the analysis of 5500 runs. While it is easy to specify different levels of factors, the analysis of the amount of data generated as a result of increasing the number of factors becomes intractable. For example, if all of the time series graphs of a single run were plotted on one sheet of paper each, we would have a pile of printouts eleven reams of paper thick. To do the analysis, we used a graphing technique supported in S-Plus called a design plot.†

Design Plots. Fig. 13 shows the design plots of the means of each of four different metrics at each of the levels of the five factors. The four metrics are EOL inventory, EOL inventory percentage, total on-time deliveries, and percent on-time deliveries. Each plot reflects one metric and summarizes the value of that metric for 5500 runs. The point labeled A is the mean of the EOL values of all experimental runs with A/F = 50% (mean of 500 values). A longer line indicates greater sensitivity of the metric to that factor over the range considered, all other things being equal. For example, A/F appears to have the strongest impact on EOL inventory, EOL percentage, and on-time shipments. On the other hand, the mature demand period L has a strong influence on the total dollar volume of on-time product deliveries.

An interesting point is that mean EOL and EOL percentage decrease steadily as A/F increases. On-time deliveries in dollars increases up to a point as A/F increases to 125%, but subsequently decreases (point B in Fig. 13c). The explanation is that the safety stock policy gives some protection for on-time delivery in dollars when A/F > 100%. On-time deliveries as a percentage remains at 100% for A/F ≤ 100% and subsequently decreases as A/F increases over 100% (point B′ in Fig. 13d).

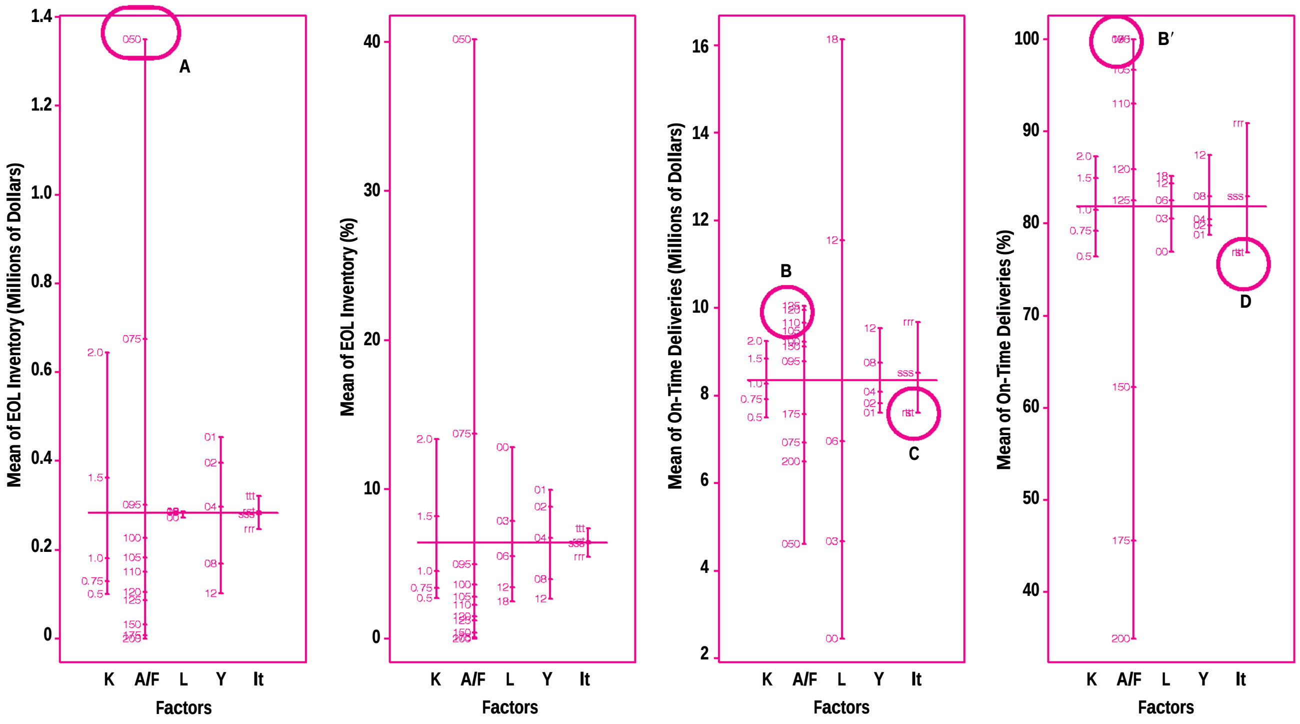
Another interesting behavior is that of the points marked C and D. The fact that the mean values of the metrics appear close together for the (25,40,35) case and the (0,0,100) case suggests that the length of the maximum lead time of parts in the bill of material has a very strong influence on on-time deliveries if all other factors are kept constant.

Further Analysis. We have barely scratched the surface of what is possible in analyzing the simulation data of this one-level bill of material, single-product situation. Further analysis and display of the variables is possible through scatter plots of pairs of variables and responses, and the use of factor plots which show greater detail. For example, further analysis could try fitting a statistical model using least sum of squares of residuals for the responses, separately and jointly. This was not done for this paper.

Experiment Set M: Multiple Product Life Cycles with Part Commonality

This set of experiments showed the impact of part commonality across multiple product life cycles. The product

† We call it a design plot because it is generated by the S-Plus function plot.design. There is no standard name of this plot. In the literature,[23] it is referred to as a "a plot of the mean response for each level of each factor."



Note: lt = (% 6-week, % 10-week, % 14-week lead time parts).
rrr indicates lt = (100,0,0)
sss indicates lt = (0,100,0)
ttt indicates lt = (0,0,100)
rst indicates lt = (25,40,35).

Fig. 13. Design plots for experiment set F: all five factors varied (5500 runs).

cycles overlapped in time, that is, one started before the preceding one finished, and we looked at a series of scenarios that differed in the values of common parts in adjacent products. These were the assumptions:

- There were four products: Adder-1, Adder-2, Adder-3, and Adder-4.
- Part commonality occurred between adjacent products only.
- Demand increased 30% for each new product.
- The unit cost of each product was 85% of the unit cost of the previous product.
- Each product life cycle was 6 months, or L = 0. This means that the complete cycle for each product is 6 months, or 24 weeks.
- There was a one-month overlap between products, that is, the first month of demand of a new product begins in the last month of demand of the previous product. This implies a total lifetime of the product family of 21 months, or 84 weeks.
- Other factors and conditions remained as in the nominal case.

Fig. 14 shows a graphical representational of the part commonality between adjacent products for the different experiments. In particular, since part commonality for experiment M-0 is 0% across adjacent products, there are no shaded areas. A fuller discussion of part commonality is given in Appendix III.

Fig. 15 shows the forecasted and actual order patterns for the four products.

Fig. 16 shows the RPI levels for parts used in the different products in Experiment M-0 (no part commonality). Consignment inventories are not shown to avoid clutter in the graphs. The WIP, FGI, products ordered, PCFT and delivery profiles are identical for all runs in experiment set M. However, each of the runs has a different profile for RPI. Note the EOL inventory of each set of parts.

Fig. 17 shows the consignment and EOL inventory levels for each run. As expected, the consignment level increases by product because the forecasted and actual orders increase by product. The consignment value for a particular product is the same across experiments. The EOL inventory for Adder-4 is the same in all the experiments. There does not appear to be any correlation between part commonality and the EOL inventory. A correlation exists between part obsolescence for a product and the EOL inventory for that product.

Fig. 18 shows the part obsolescence across products for each of the experiments. Notice how the EOL for each product in Fig. 17 is proportional to the obsolete parts for each product in Fig. 18.

Traditionally, in considering part commonality, design principles suggest using as many parts as possible from the old product. However, the results above suggest that from the point of view of EOL inventory, the amount of leftover material at the end of each product is proportional to the percentage of the part value of the obsolete parts in the old product.

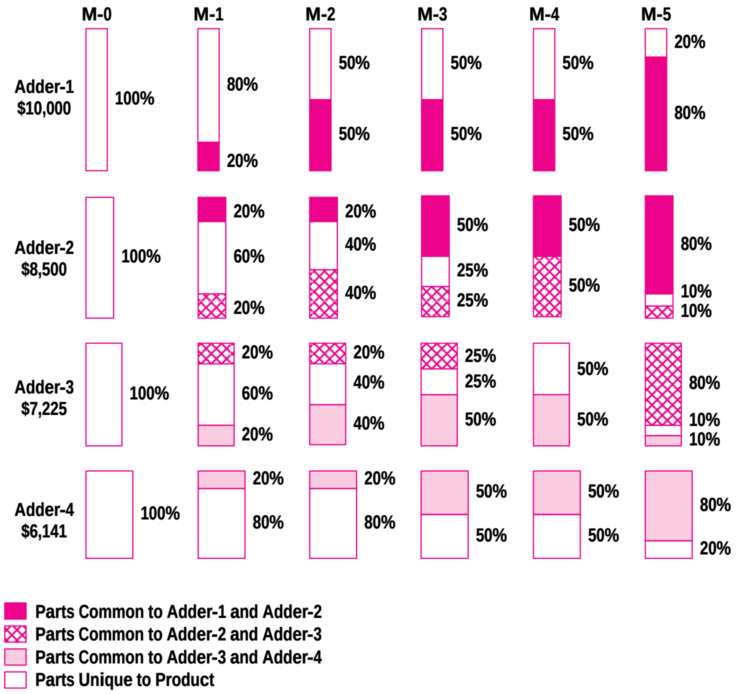


Fig. 14. Part commonality between products across experiment set M. Demand (width of bars) for each product is 30% higher than for the previous product. Unit cost (length of bars) is 85% of previous product cost.

It further suggests that the important consideration from the point of view of EOL inventory is that the percentage value of obsolete parts at the end of each product's life should be minimized.

Discussion

In this section we discuss specific results of the Simple Model, enhancements to the EMS system to do more detailed analysis, the role of the Simple Model in enterprise modeling and simulation, and optional ways of using enterprise modeling and simulation.

The major results can be summarized as follows:

- Rational material ordering and safety stock policies designed to reduce inventory to zero at the end of the product life cycle can give rise to leftover material if customers orders exactly according to forecast.
- System behavior and the impact on different metrics such as write-off, delivery times, and performance deliveries can be quantified with respect to the factors of forecast quality, safety stock levels, material lead times, product life cycles, and quoted availability individually as well as in combination.
- Forecast quality, which is influenced by the external environment, has a major effect on the metrics of interest. For example, high inventory levels may occur when actual orders come in too high or too low.
- The influence of part commonality on write-off can be quantified; this suggests an alternative way of looking at the practice of using common parts in a series of products.

What have we learned from the simulation runs on the Simple Model? We have derived a set of specific insights into system behavior under a variety of operating conditions using a methodology of generating behavior over time. We went through a large number of scenarios and showed how to gauge system behavior from the perspectives of different parties.

Interpreting the Results

The model results are sensitive to the underlying assumptions. Since we assumed the vendors always delivered on time in the simulation, the safety stocks in effect guarded only against demand uncertainties. We examined in detail the situation of order forecast bias with zero variance. This

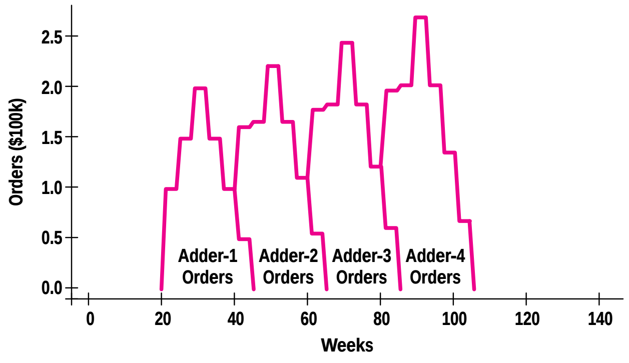


Fig. 15. Orders for different products for experiment set M.

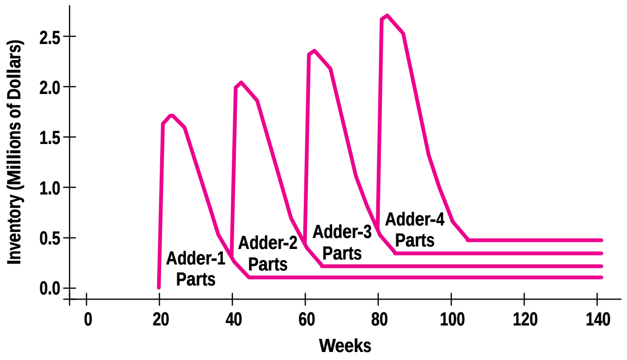


Fig. 16. RPI levels for the different parts as a function of time for experiment M-0 (no part commonality).

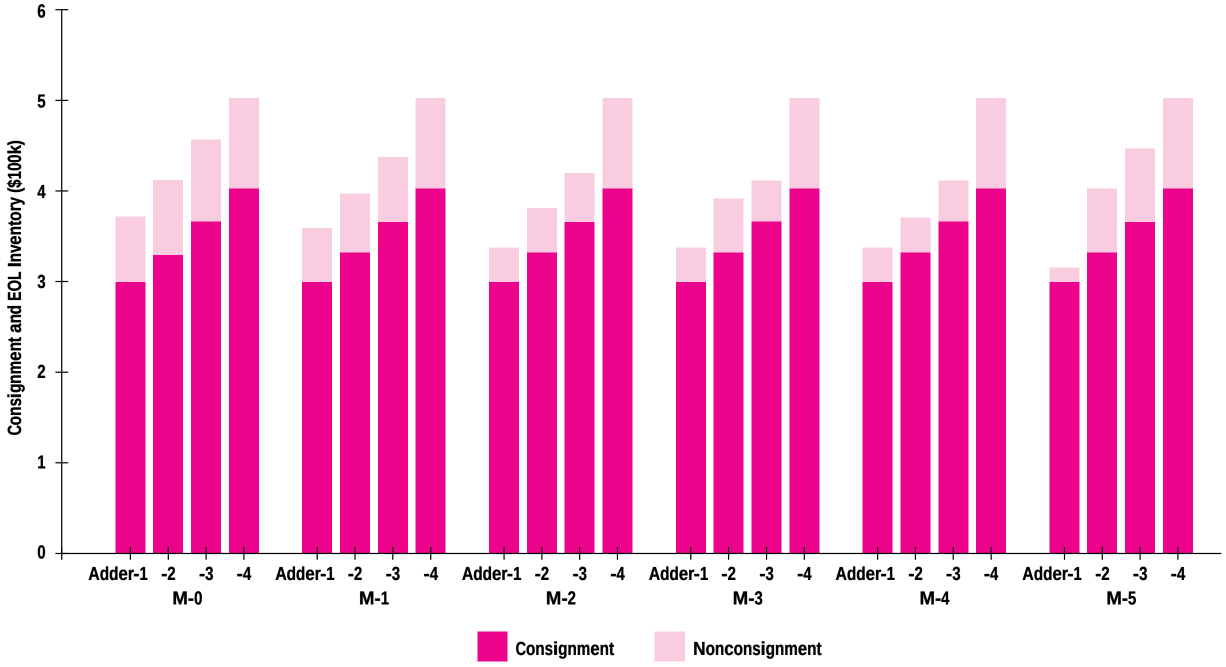


Fig. 17. Consignment and EOL inventory by product for different amounts of part commonality for experiment M-0.

is not an inherent limitation of the model, but reflects only the deterministic circumstances in which we ran the simulations. However, the results indicate that even if production and supplier lead times are completely predictable and suppliers deliver on schedule, interactions and delays within the system lead to long lead times being seen by the customers when there is underforecasting of customer orders. The manufacturing enterprise needs to take this into account and start looking elsewhere—merely making the production faster and more efficient is not sufficient.

The results so far have only scratched the surface of the analysis and interpretation possibilities. Other analysis could be done by varying ship times, FGI safety stock levels, production planning frequency, material ordering frequency,

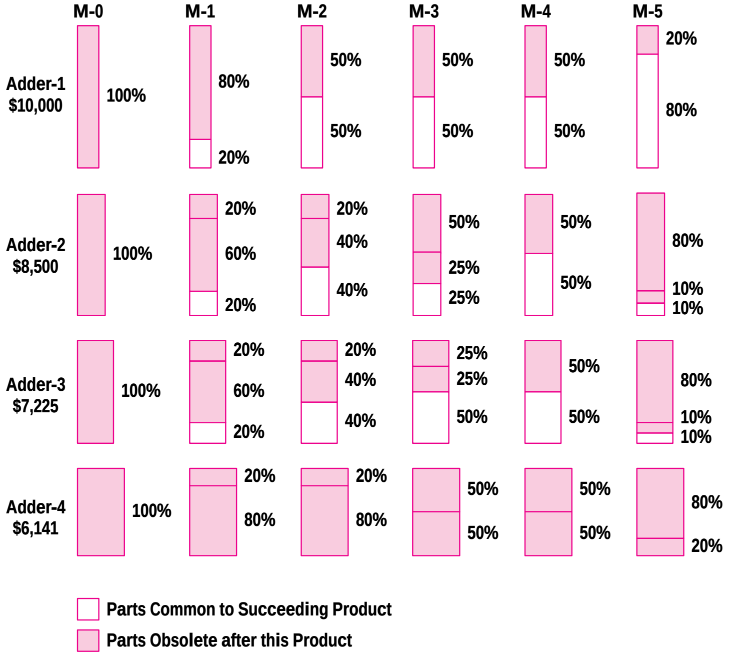


Fig. 18. Parts obsolescence between products across experiment set M.

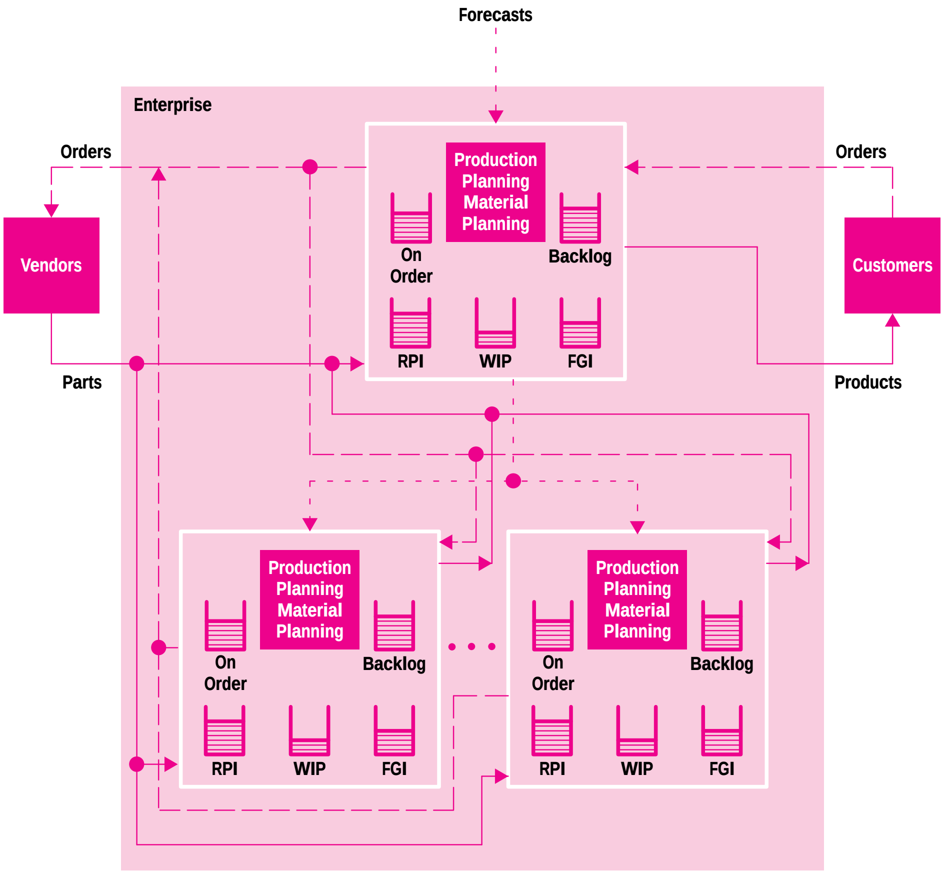


Fig. 19. Material and order flow diagram of a simple multientity distributed enterprise.

order filling policies, and uncertainty and time delays of information flow. This increases the number of runs and the quantity of data collected as well as the complexity of analysis, but would provide a richer set of relationships.

The Simple Model example may have left the reader with the impression that the current EMS system can deal with only simple or trivial cases. One goal of enterprise modeling and simulation research activities is to address successively more complex interactions and to model real-world intricacies more closely. In support of that goal, the following sections discuss subsequent and future enhancements to deal with other issues that have been raised.

Uncertainty and Variability. In the experiments described, the Simple Model was run under deterministic circumstances. Demand values and process times were constant across a particular run for convenience of understanding, and we considered uncertainty in the form of forecast biases where demands were a fixed multiple of forecasts over the period of the forecasts. Other forms of uncertainty could include the actual life cycle being different from the forecasted life cycle. Uncertainly in process times could be handled by using two values for process times: the planned process time for planning purposes and the actual time for execution. This reflects the situation when actual process times are uncertain and different from the estimated times for the process. For example, the build time for planning purposes could be two weeks, but it could turn out that the actual build time was one or three weeks.

We did not deal with variances that might occur when the total demand is forecasted accurately but the week-to-week demand fluctuates widely. Furthermore, variances of process times (e.g., delivery times from vendors and assembly times), yields (e.g., defective units), and build times for individual units were not modeled.

Dealing with variances is fairly straightforward once they are characterized. It requires using random number generators and multiple runs starting with different random number seeds—the current practice of discrete event simulation. There are three primary costs associated with this: the increase in data collection to characterize the variances of different processes, the increase in computational effort, and the increase in analysis effort. Only the data for the model needs to be changed to reflect variances. The model structure itself requires no changes.

Distribution and Multisite/Multiorganizational Interaction. The product distribution function and interaction between multiple sites were not considered in the Simple Model. Multisite and multiorganization interactions have been implemented by enclosing cloned versions of a slightly enhanced manufacturing enterprise model as shown in Fig. 19. The enhancement requires the manufacturing facility to generate and transmit its projected material requirements in addition to material orders.

Capacity and Supply Limitations. In current practice, build plans and material plans are sometimes computed ignoring production capacity and vendor limitations. In some cases, these plans are adjusted to conform to production capacity and vendor supply constraints, such as a minimum order quantity or a maximum that can be ordered in a period. In other cases, these limitations are observed at plan execution, that is, at production, or when deliveries are not received from vendors when expected. There is no unique way of dealing with these limitations.

Implementing capacity limitations in the current Simple Model is straightforward during production. To deal with it during planning requires the inclusion of two classes of capacity constraints in the production planning algorithms: the capacity restrictions for an individual product, assembly, or subassembly as well as total capacity, and the rate at which production capacity can increase.

In reality, when prospective capacity limitations are detected, production and manufacturing line design and engineering considerations determine the rate of capacity expansion. When gross overcapacity is detected, consideration is given to reducing costs by reducing capacity. While currently the EMS system cannot model the strategic decisions of whether to expand capacity or forego extra orders, it can model the consequences of picking either of these actions.

Interaction of Multiple Products. The Simple Model assumed a single product with unconstrained production capacity. Consequently, a single unavailable part stops production of that product. Since this phenomenon also occurs with multiple products with no common parts, multiple products with no common parts can be analyzed by adding up the effects of the individual products separately. The reader familiar with linear systems will recognize this as the principle of superposition.

Adding up the results would also be valid for multiple products with common parts with no part shortages as in experiment set M. It would not be valid for multiple products with common parts, resources, and supply and production capacity limitations under scarcity conditions. When a part or resource is in short supply, decisions must be made on how to allocate the parts and resources based on some simple heuristic or optimal allocation scheme.

Multilevel Bills of Materials. The Simple Model dealt with a single-level BOM. Further expansions allow an arbitrary number of levels of BOM to be passed as data to the model. A seven-level BOM for a real product has been implemented and tested successfully. This capability to pass BOM as data allows us to make different runs with different product structures (as for example in experiment set M) without modifying the model structure.

Connection to a Mathematical Programming or Optimization Package. The Simple Model focused on applying simple algorithms for planning. The production planning and material procurement processes were initially implemented as the explicit closed-form solutions derived in Appendix I. It was realized subsequently that these algorithmic closed-form solutions were the solutions to the linear programming problem formulation. As more sophisticated planning decision techniques are proposed and studied, implementing the algorithmic solution for each new technique becomes impractical. An alternative approach is to formulate the planning process as an optimization problem and separate its solution from the formulation. This leads to concentrating on ways to better formulate the problem, leaving the solution to a separate process such as a mathematical programming package. This could provide a means of rapidly testing alternative strategies for production planning (e.g., global production planning across the entire enterprise versus local production planning at each site).

R&D, Marketing, and Cash Flow. Fig. 20 shows a proposed enterprise model at a broader scope for the next level of complexity. It generalizes Fig. 3 which focused mainly on manufacturing activities. Modeling the marketing function (and associated activities such as the forecasting process, pricing issues, and product obsolescence) could help show the impact of marketing decisions and activities on the overall system response as well as the impact of using current

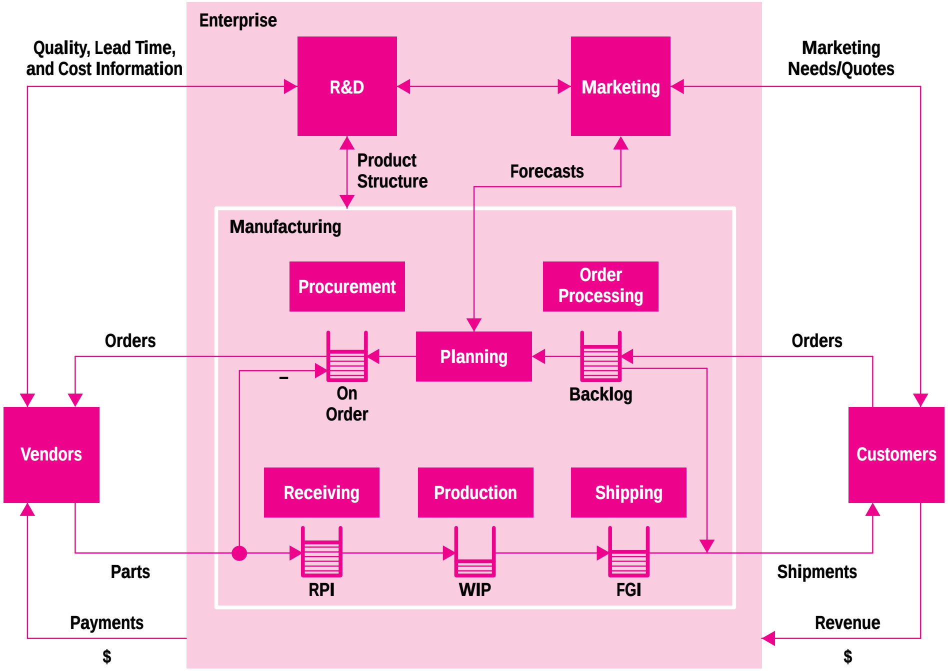


Fig. 20. Proposed enterprise modeling entities for expanded analysis.

orders to project future forecasts. Modeling the R&D function could provide insights on impacts on time to market, with product development time taken into account in addition to build time. Modeling these functions can help us deal with situations that require coordination of marketing, R&D, and manufacturing activities and can help identify the existence of leverage points for process improvement. The blocks shown in the diagram represent functions, and each could describe multiple instances of that function. For example, the block labeled manufacturing could represent multiple manufacturing sites interacting with one another.

The primary flows in the Simple Model concentrated on information (e.g., orders, forecasts, plans, and status information), material, and control (e.g., triggers that cause activities like production to start). Flows and inventory levels were converted to monetary units before being analyzed, but cash flows were not modeled explicitly.

Modeling cash flows for payments of parts, products, and process costs will provide a financial perspective. Showing projected cash flows and investments and the projected financial consequences of investment decisions will provide the stepping stones to doing discounted cash flow and net present value analysis. Modeling cash flows will also help generate pro forma financial statements to estimate revenue, cost, and income owing to different capital budgeting and allocation decisions, and provide a tool that could help address business issues. An example of such an issue is the transition from a high-margin business to a low-margin high-volume business.[24] The model may help by projecting cash requirements for investments and operations and providing estimates for return on assets during the transition.

Whither the Simple Model and the EMS System?

The Simple Model is not an end or final model; it is intermediate in a series of models that have contributed to the evolution of enterprise modeling and simulation (see page 90) and the development of the EMS system. Its simulation demonstrates the kinds of results that can be generated by enterprise modeling and simulation. Its value is in providing greater quantitative analysis where previously qualitative approaches have been adequate (see below). Its immediate subsequent application was the planning calendar model.[25,26,27]

The subsequent and future enhancements discussed make the Simple Model more complete. Some of the changes make the model larger, add detail complexity, and generate more precise results. Other changes broaden the scope of the model, and make it more representative of the other functions of the enterprise besides manufacturing; these changes require the addition of greater levels of abstraction, the ability to consolidate different points of view, and knowledge acquisition across the organization. All the changes are technically feasible and require different kinds of activities: the first set of changes requires greater emphasis on “modeling in the small,” and the second set requires greater emphasis on “modeling in the large” (see discussion on page 81). Discussions based on the experience and views of some managers responsible for operations suggest that expanding the size by increasing the detail complexity, while providing greater predictability of the system, is difficult and requires a tremendous amount of investment to manage the complexity of the models and the generation and interpretation of the resulting data. Monroe[24] and Harmon[28] have individually recommended that there is greater value and potentially a far greater return on investment to be obtained by broadening the scope of future models to address and reflect business issues and concerns.

Regardless of the direction of model enhancement is the challenge of managing simulation data. The simulation runs for the experiments generated large amounts of data, and only aggregate data was collected and summarized. For instance, RPI levels for every part were generated for each week during the simulation, but the data collected was the aggregate dollar value of all the parts. The challenge became one not of collecting all data, but one of deciding ahead of time which data was interesting and not collecting that

The Simple Model: Sponsor's Perspective

As HP's Computer Systems Organization customers increasingly request delivery of complete systems with much shorter lead times, our design, manufacturing and delivery systems are being stretched beyond their performance limits.

Qualitative approaches to improvement have served us well in the past, but more quantitative analysis is needed to understand and improve the total system both from a customer and an HP perspective.

The Simple Model was conceived and developed in teamwork with HP Laboratories. We sponsored it to help learn and communicate the key drivers and characteristics of a manufacturing enterprise. The insight achieved could then be used in our order fulfillment initiative to design product, manufacturing, and delivery systems to match critical business requirements and position us to meet future customer needs effectively in the global marketplace.

Jerry Harmon
General Manager
HP Puerto Rico
Sponsor of Simple Model
for HP Computer Manufacturing

which was not; otherwise the storage requirements for storing all the generated data became significant. The data presented in the form of graphs and charts in this paper is only a small portion of the actual data collected and analyzed. A larger amount of collected data was discarded because it did not look interesting.

The sheer amount of detailed data that needs to be examined and interpreted tends to overwhelm the analyst. The analysis and interpretation of the data was very much a creative team effort requiring much discussion, and is not yet understood well enough to be automated. As we increase the number of factors, the behavior becomes more complex, and the amount of data tends to increase exponentially with the number of factors. When presented with the data in its raw form, decision makers and experts familiar with the problem issues but less familiar with modeling and simulation all have the same general reaction that it is too complex and difficult to understand. While this is a valid reaction, the reality is that the enterprise is a complex system of interacting information, material, resource, and control flows, and whether we like it or not, has complex behavior. Enterprise models as abstractions or idealizations for the real system merely reflect that complex behavior in the simulation. We can choose to ignore the complexity of the real system and use ad hoc qualitative methods to deal with the resulting behavior, or we can choose to face the complexity, understand it by selecting what we think are important factors that influence the behavior of the enterprise, and find opportunities for applying the understanding. Enterprise modeling and simulation represent one means of facing this complexity and providing an understanding of this behavior. As with most endeavors, we have found that the precursor to simplicity of expression is greater depth of understanding.

Increased technology in the hands of the modeling and simulation expert is not sufficient for providing the insight that will help make better decisions and highlight important results. Merely generating large numbers of insights and conclusions is insufficient. It requires the perspective of operations teams and decision makers to guide the direction of exploration and to emphasize the correct metrics to solve the current situation. In fact, Monroe[24] has suggested, and we in the enterprise modeling and simulation project concur, that techniques to digest and present large amounts of data rapidly and in a more easily understood fashion would be a beneficial next step and a fruitful area of research, and that joint work of a modeling expert with an operations team to further understand the issues of data reduction, interpretation, and presentation will help modeling and simulation take its rightful place as a useful tool in analyzing business decisions.

The Simple Model is a descriptive model that illustrates complex dynamic behavior of a manufacturing enterprise with low structural and detail complexity. As we have seen in this paper, its primary output is data and information on the state of the world, and it goes a great distance towards presenting observations. Unlike an optimization model, which is a prescriptive model whose solution recommends the best action under a given set of circumstances, the Simple Model does not suggest actions. It is up to the analyst or decision maker to come up with creative solutions to solve the problems highlighted by observations of the model behavior and then assess the results from a subsequent simulation run incorporating those solutions.

Prospective Applications

Let us now look at application areas for enterprise modeling and simulation. These include but are not limited to improving the performance of the current system (continuous improvement), studying the impact of reducing process times, and generating information for the enterprise, all of which are discussed below. A potentially far more powerful application is looking at new designs where the process itself is being changed (i.e., reengineering). Because of the strong current interest, large impact, and controversy surrounding reengineering, this subject is given its own discussion on page 86.

Incremental Improvements. Actions for continuous improvement can be suggested by running the nominal or baseline model and rerunning it with minor modification and changes in parameters or actions over which we have control. For example, it may not be possible to reduce all the part lead times down to six weeks, but we could certainly see the impact of reducing the value of 14-week parts in the product to determine the impact on the metrics of interest. We could look at the impact of reducing build times or FGI safety stock levels slightly to study the impact on the measures of interest. We could examine the impact of making two small changes at the same time. This application of enterprise modeling and simulation supports the process of continuous improvement by demonstrating the benefits of small changes.

Verifying Impact of Reducing Process Times. Davidow and Malone[29] talk about how short cycle times attenuate "the trumpet of doom," which is a plot of forecasting error versus time that implies that the further a person must forecast into the future, the greater the possibility of error. Rather than speculate on or guess on the impact of this trumpet of doom, enterprise modeling and simulation provide a way to quantify the effect of reducing system cycle times. This can be accomplished by making some estimates of the amount of uncertainties within the model.

Stalk and Hout[30] suggest mapping out explicitly the major causes of problems in processes such as new product development or in operations, and comparing actual versus standard cycle times. These maps provide qualitative relationships. To the extent that processes can be mapped explicitly and quantitatively, enterprise modeling and simulation can show how the system behavior changes for a given change in the process and can verify whether modifying the component processes has the desired overall global effect.

Generating Enterprise Behavior Information. Davidow and Malone[29] identify four categories of information of use to a corporation: content, form, behavior, and action. Content information is historical in nature and reflects the experience. Form information describes shape and composition and is usually more voluminous than content information. Behavior information often begins with form information and usually requires a massive amount of computer power to predict behavior through simulation. They suggest that the final triumph of the information revolution will be the use of action information—information that instantly converts to sophisticated action. Until recently, only the most elementary category, content, has been available to business in any systematic and manageable way, and obtaining or generating the other three categories has become economically feasible only in recent years. They go on to describe how behavior information generated by computer simulation is the new paradigm for product design ranging from molecular design through automotive design to airplane design. With such behavior information design disasters of the past might be averted, and potential and unforeseen future tragedy can be replaced with a successful and predictable conclusion. With the arrival of workstations in the 1980s, it became reasonable for the computer to create realistic models and put them through their paces rather than painstakingly building prototypes and testing them under a variety of operating conditions. High-speed simulators could be built that reproduced the actual electrical characteristics of devices in different configurations.

We suggest that enterprise modeling and simulation represent an assistive and enabling technology for the design and implementation of processes of the enterprise, and that the application of such techniques to the enterprise could potentially have greater impact than product design. Furthermore, these techniques have the characteristic of converting content and form information into behavior information on which action can be taken. While the enterprise modeling and simulation process currently does not suggest actions or alternatives, it describes the behavior of the system designed with alternate processes under different operational scenarios.

Conclusions

In this paper, we outlined activities in enterprise modeling and simulation at HP Laboratories and presented in detail the results of the simulation of a simple model of a manufacturing enterprise. We have also described possible areas where enterprise modeling and simulation might be applicable, and reiterate that enterprise modeling and simulation provide a way of quantifying the impacts of proposed changes before they are implemented.

The Simple Model captures the characteristics and behavior of a manufacturing entity at a fairly high level. It shows that in the best of circumstances (e.g., customers ordering exactly according to forecast), seemingly rational operational policies can lead to end-of-life inventory. The situation only gets more complex as greater uncertainty is introduced.

Experience with using the Simple Model suggests two directions for future research in enterprise modeling and simulation. The first is to expand the scope of the Simple Model to more completely represent the functions and organizations and their interactions in the enterprise. The second is to improve the process by which the data generated by the simulation models can be understood and summarized, and the resulting information presented in a form that permits decision makers to understand more completely and to act more rapidly and with greater assurance that the desired objectives will be achieved.

Acknowledgments

The enterprise modeling and simulation system was developed by the team of Bob Ritter, Bob Joy, and the author, all of whom are affiliated with Hewlett-Packard Laboratories. The Simple Model was proposed by Jerry Harmon of HP Puerto Rico and developed by Shailendra Jain and the author in two parallel efforts that served to verify the results using two different approaches. Jerry Harmon, Bob Ritter, Shailendra Jain, and Paul Williams of Hewlett-Packard Laboratories and the author were involved in interpreting the results of the Simple Model at various times. Others have provided fresh insight during the course of discussions. The author would like to thank his team members, colleagues, and the many reviewers for their helpful comments and assistance in the preparation of this document.

References

1. A.M. Law and W.D. Kelton, *Simulation Modeling and Analysis, Second Edition,* McGraw-Hill, Inc., 1991, p. 1.

2. A.A.B. Pritsker, *Introduction to Simulation and SLAM II, Third Edition,* Systems Publishing Corp., 1986.

3. R. McHaney, *Computer Simulation—A Practical Perspective,* Academic Press, Inc., 1991.

4. A.M. Law and M.G. McComas, “Secrets of Successful Simulation Studies,” *Proceedings of the Winter Simulation Conference 1991,* Society for Computer Simulation, pp. 21-27.

5. F.E. Cellier, *Continuous System Modeling,* Springer-Verlag, Inc., 1991.

6. P.M. Senge, *The Fifth Discipline,* Doubleday/Currency, 1990.

7. L. Marran, M.S. Fadali, and E. Tacker, “A New Modeling Methodology for Large-Scale Systems,” *Proceedings of the International Conference on Systems, Man, and Cybernetics,* IEEE, May 1989, pp. 989-990.

8. *Structured Methods—An Overview for Engineers and Managers,* Hewlett-Packard Corporate Engineering, 1988, pp. 77-90.

9. M.S. Mujtaba, “Simulation Modelling of a Manufacturing Enterprise with Complex Material, Information, and Control Flows,” *International Journal of Computer Integrated Manufacturing,* Vol. 7, no. 1, 1994, pp. 29-46.

10. B.P. Zeigler, *Multifaceted Modeling and Discrete Event Simulation,* Academic Press Inc., 1984.

11. V.B. Norman, et al, "Simulation Practices in Manufacturing," *Proceedings of the Winter Simulation Conference 1992,* pp. 1004-1010.

12. M. Fox, "The TOVE Project—Towards a Common Sense Model of the Enterprise," *Proceedings of the International Conference on Object-Oriented Manufacturing Systems (ICOOMS),* May 1992, pp. 176-181.

13. H.R. Jorysz and F.B. Vernadat, "CIM-OSA Part 1: Total Enterprise Modelling and Function View," *International Journal of Computer Integrated Manufacturing,* Vol. 3, nos. 3 and 4, Fall 1990, pp. 144-156.

14. H.R. Jorysz and F.B. Vernadat, "CIM-OSA Part 2: Information View," *International Journal of Computer Integrated Manufacturing,* Vol. 3, nos. 3 and 4, Fall 1990, pp. 157-167.

15. A. Pardasani and A. Chan, "Enterprise Model: A Decision-Support Tool for Computer Integrated Manufacturing," *Proceedings of the International Conference on Object-Oriented Manufacturing Systems (ICOOMS),* May 1992, pp. 182-187.

16. J. Harmon, *personal communication.*

17. R. Norton, "A New Tool to Help Managers," *Fortune,* May 30, 1994, pp. 135-140.

18. G.L. Steele, *Common Lisp—The Language, Second Edition,* Digital Press, 1990.

19. *Allegro CL Common Lisp User Guide Volumes 1 and 2 Version 4.2,* Franz Inc., January 1994.

20. *Lucid Common Lisp/HP Manuals and User Guide,* Lucid Inc., June 1990.

21. *LispWorks Manuals Edition 3.2,* Harlequin Ltd., March 1994.

22. M.S. Mujtaba, *Formulation of the Order-to-Ship Process Simulation Model,* HP Laboratories Technical Report #HPL-92-135, December 1992.

23. J.M. Chambers and T.J. Hastie, *Statistical Models in S,* Wadsworth & Brooks/Cole Advanced Books & Software, 1992.

24. J. Monroe, sponsor of planning calendar model,[25,26,27] leader of HP Computer Systems Organization planning program 1992-1993, *telephone communication,* September 12, 1994.

25. C.M. Kozierok, *Analysis of Inventory and Customer Service Performance Using a Simple Manufacturing Model,* Master of Science Thesis for Leaders for Manufacturing (LFM) Program, Massachusetts Institute of Technology, May 1993.

26. K. Oliver, *Simple Model Report,* distributed by email on January 12, 1993.

27. M.S. Mujtaba and R. Ritter, *Enterprise Modeling System: Inventory Exposure and Delivery Performance,* HP Laboratories Technical Report #HPL-94-89, October 1994.

28. J. Harmon, sponsor and proponent of the Simple Model, leader of HP Computer Manufacturing forecasting and planning redesign team 1991-1992, currently General Manager, HP Puerto Rico, *telephone communication,* September 1994.

29. W.H. Davidow and M.S. Malone, *The Virtual Corporation,* Harper-Collins Publishers, Inc., 1992.

30. G. Stalk and T.M. Hout, "Competing Against Time," *The Free Press,* A Division of Macmillan, Inc., 1990.

Appendix I: Mathematics of Production and Material Planning for the Simple Model

I-1 The Planning Function

The planning function is actually an analytic model embedded within a discrete event simulation model. The fundamental principle on which the production and material planning algorithms are based is the conservation of mass, that is, consumption cannot be higher than the total supply available. The order in which the build plan computation is done is the reverse of the order in which subassemblies are built and products are shipped (i.e., from shipment to product build to part order). For ease of explanation, the current week is considered to be week 0. This derivation emphasizes clarity of explanation rather than rigorous detail.

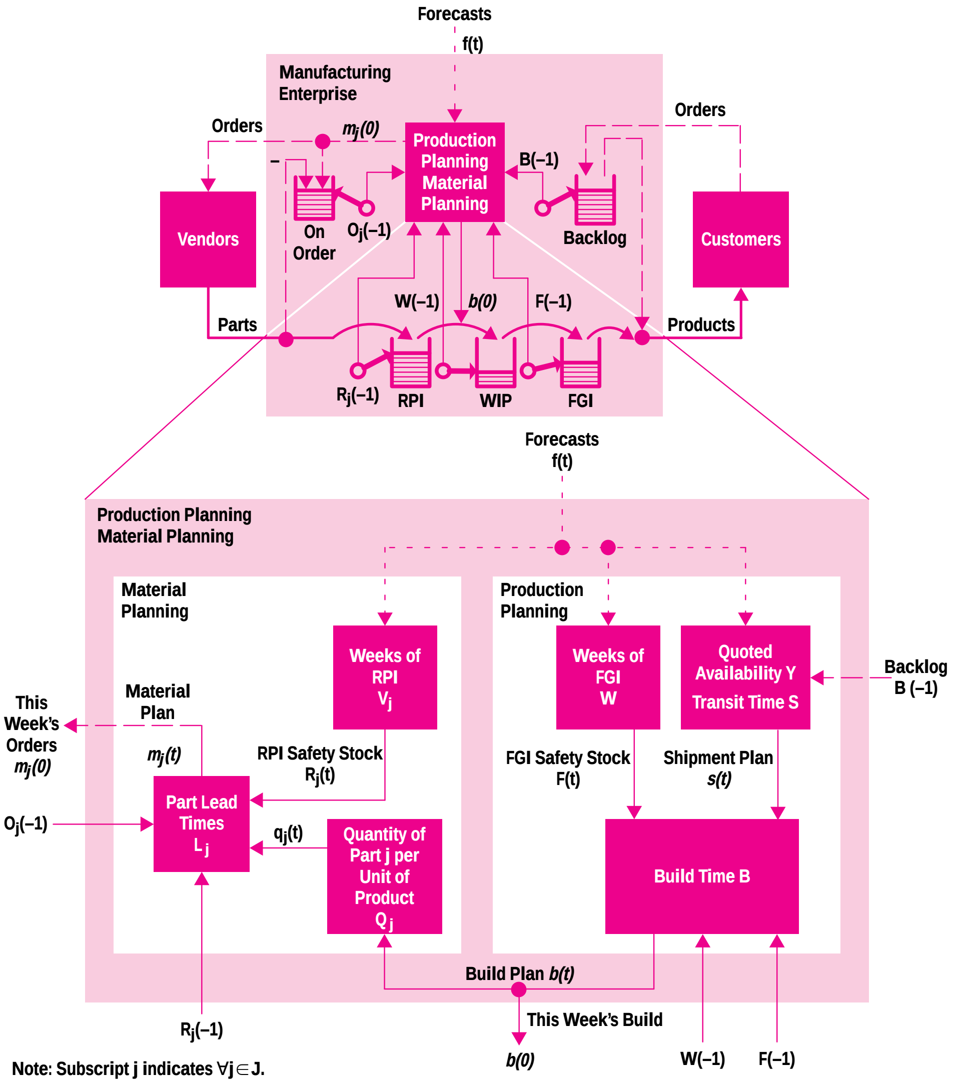
There are three sets of decision variables to be determined for each week: *s(t),* the shipment plan, *b(t),* the build plan, and *$m_j(t)$,* the material ordering plan. These are shown in italics.

Before we get into the mathematical formulation, let us first look at the process of computation. Fig. 1 illustrates how the production and material planning algorithms work in this model. The computational process is described in the following order:

- I-2 describes the notation shown in Fig. 1.
- I-3 describes the safety stock computation.
- I-4 describes the initial conditions for computation.
- I-5 describes the computation of the shipment plan.
- I-6 describes the computation of the build plan.
- I-7 describes computation of the number of units started this week.
- I-8 describes the computation of the material consumption and material ordering plans.
- I-9 describes the actual material ordered this week.
- I-10 describes the computation of the number of weeks for each of the plans.

I-2 Notation

- n, s, t = indexes for week number (current week = 0)
- f(t) = Current forecast of product orders for week t, t = 0, 1, …, N_f
- F(t) = FGI at end of week t
- W(t) = WIP at end of week t
- B(t) = Backlog units at end of week t
- B(t,s) = Backlog units at end of week t having shipment dates in week s
- *s(t)* = Planned shipments during week t
- *b(t)* = Units planned to be started during week t
- B = Build time in number of weeks
- Y = Quoted availability in number of weeks
- S = Shipment or transit time
- j = Index relating to part
- Q_j = Quantity of part j per unit of product
- $q_j(t)$ = Planned consumption of part j during week t



Note: Subscript j indicates ∀j∈J.

Fig. 1. Notation and production/material planning. The shipment plan is computed from the backlog, forecasts, quoted availability, and transit time. The build plan is computed from the shipment plan, the build time, WIP, FGI, and FGI safety stock. The actual build is computed from the build plan and the material availability. The material consumption plan is computed from the build plan and the bill of materials. The material ordering plan is computed from RPI, RPI safety stock, the material consumption plan, on-order material, and lead time.

- $m_j(t)$ = Planned quantity of material j to be ordered during week t, t = 0, 1, …, N_j, $j \in J$
- $R_j(t)$ = RPI of part j at the end of week t
- $r_j(t)$ = Units of part j received during week t
- $O_j(t)$ = Units of part j on order at the end of week t
- L_j = Vendor lead time for part j
- J = Set of parts that go into the product
- w = FGI safety stock in weeks of demand
- V_j = RPI safety stock of part j in weeks of demand
- N_s = Last week for computing shipment plan
- N_b = Last week for computing build plan
- N_f = Last week used for forecasts
- N_j = Last week for computing material order for part j.

Since the current week is 0, the values of these variables represent actual values for weeks before 0, and the values are computed, set, or derived for weeks 0 and later. In particular, the values of variables at the end of week –1 represent the current values of those variables, as described in I-4. All numerical quantities except time indexes are zero or positive.

I-3 Safety Stock Computation

Safety stock is expressed in number of weeks of 13-week leading average forecast. The 13-week leading average forecast at the end of week t is defined as:

$$\overline{f(t)} = \frac{1}{13}\sum_{i=1}^{13} f(t+i) \tag{1}$$

The target FGI safety stock at the end of week t is w weeks and the target RPI safety stock at the end of week t for part j is V_j weeks. The expressions for these quantities are:

$$F(t) = w\overline{f(t)} \tag{2}$$

$$R_j(t) = V_j Q_j \overline{f(t)} \tag{3}$$

I-4 Initial Conditions

- F(–1) = Actual FGI at the end of the previous week, that is, current FGI
- W(–1) = Actual WIP at the end of the previous week, that is, current WIP
- $O_j(–1)$ = Actual part j on order at the end of the previous week, that is, current on-order material
- $R_j(–1)$ = Actual RPI for part j at the end of the previous week, that is, current RPI for part j.
- B(–1) = Order backlog in units at the end of the previous week, that is, current backlog:

$$B(-1) = \sum_{s \in \langle \text{all shipment dates in current backlog} \rangle} B(-1, s) \tag{4}$$

- B(–1,s) = Component of current backlog with shipment date in week s.

I-5 Shipment Plan

The shipment plan indicates prospective shipments during the current and future weeks. It is computed on the assumption that customer orders are not shipped before they are due, but are shipped in time to satisfy the quoted availability requirements. This implies that for any week, the orders planned to be shipped are those that are already late (i.e., should have been shipped in an earlier week) and those that must be shipped to be delivered on time. Notice that in computing the shipping plan, we do not take into account the amount of inventory on hand or in process. This is representative of the way shipment plans are computed and then subsequently checked against reality.

Put another way, this can be expressed as planning to ship the minimum quantity in each week that will satisfy the quoted availability criteria. The problem can be formulated as shown in the set of equations below, which indicate that we are attempting to minimize shipments in the current week, current plus next week, current plus next 2 weeks, and so on such that the total shipments in those weeks is greater than the current existing backlog whose shipment date is already past or in those weeks, plus the forecasted orders whose desired shipment dates lie in those weeks.

Minimize $s(n)$, n = 0,1,…,N_s

$$\text{such that } \sum_{t=0}^{n} s(t) \geq \sum_{t \in \langle i \mid i \leq n \rangle} B(-1, t) + \sum_{t=0}^{n-(Y-S)} f(t)$$

and $s(n) \geq 0$.

These equations define a series of (N_s + 1) linear programming problems. However, this formulation will always return a set of feasible solutions, and the optimal feasible solutions can be expressed in closed form as follows:

$$s(n) = \begin{cases} \displaystyle\sum_{s \in \langle i \mid i \leq 0 \rangle} B(-1, s) & \text{for } n = 0 \\ B(-1, n) & \text{for } 0 < n < Y - S \\ f(n - (Y - S)) & \text{for } n \geq Y - S. \end{cases} \tag{5}$$

The term (Y – S) is the difference between the quoted availability and the transit time (i.e., the order-to-ship time to achieve on-time delivery), and indicates the time in the future after which shipments depend solely on forecasts.

I-6 Build Plan

The build plan, which indicates how many units are to be started in the current week 0 and succeeding weeks, is based on the assumption that the FGI levels at the end of weeks 0,1,…,B–1 have already been determined by the current FGI, WIP, and shipments preceding week 0. It further assumes that we might be able to control FGI at the end of week B or later by deciding how many units we start this week and future weeks, that is, by controlling *b(0),b(1),…,b(n).* We want to keep the *b(n)* as low as possible but greater than or equal to 0, such that the total planned build during weeks 0 through n must be greater than or equal to shipments during weeks 0 through B+n plus FGI at the end of week B+n minus current FGI and WIP. The complete formulation is as follows:

Minimize $b(n)$, n = 0,1,…,N_b

$$\text{such that } \sum_{t=0}^{n} b(t) \geq \sum_{t=0}^{B+n} s(t) + F(B+n) - F(-1) - W(-1)$$

and $b(n) \geq 0$.

Again, the above is a series of (N_b+1) linear programming problems, with optimal feasible solutions that are expressed in closed form as follows:

$$b(n) = \max\left\{0,\ F(B+n) + \sum_{t=0}^{B+n} s(t) - F(-1) - W(-1) - \sum_{t=0}^{n-1} b(t)\right\}, \tag{6}$$

for n = 0, 1, …, N_b.

To summarize the above, the current build plan should look as follows:

Week:	0	1	2	…	n
Planned Build:	*b(0)*	*b(1)*	*b(2)*	…	*b(n).*

I-7 Actual Units Started

The actual units started this week, b_0, will be *b(0)* if there is sufficient material. If there is insufficient material the actual units started is the maximum possible with the available material, or:

Maximize b_0

such that $Q_j b_0 \leq R_j(-1) + r_j(0),\ \forall j \in J$

and $0 \leq b_0 \leq b(0)$,

for which the closed form solution is:

$$b_0 = \min\left\{b(0), \min_{j \in J}\left(\frac{R_j(-1) + r_j(0)}{Q_j}\right)\right\}. \tag{7}$$

I-8 Material Requirement Analysis

If the lead time for a part j is L_j weeks, the RPI level for part j at the end of weeks 0,1, … L_j–1 has been determined by material on hand, material on order, and projected use. We could control RPI for part j at the end of week L_j or later by deciding how much of part j we order in this week and subsequent weeks. The estimated material consumption during a week is the quantity of the material for the build for that week, that is:

$$q_j(t) = Q_j b(t). \tag{8}$$

The material ordered during weeks 0 through n must be greater than or equal to the material consumed during weeks 0 through L_j+n plus the desired safety stock at the end of week L_j+n minus the current on-hand material and the current on-order material. This can be expressed mathematically as follows:

Minimize $m_j(n)$, n = 0,1,…,N_j, j∈J

such that $\sum_{t=0}^{n} m_j(t) \ge \sum_{t=0}^{L_j+n} q_j(t) + R_j\left(L_j + n\right) - R_j(-1) - O_j(-1)$

and $m_j(n) \ge 0$.

After substituting equation 8, this becomes a series of linear programming formulations for which the closed form solution is:

$$m_j(n) = \max\begin{cases} 0 \\ \left(Q_j \sum_{t=0}^{L_j+n} b(t) + R_j\left(L_j + n\right) - R_j(-1) \right. \\ \quad \left. - O_j(-1) - \sum_{t=0}^{n-1} m_j(t) \right) \end{cases} \tag{9}$$

for n = 0, 1, …, N_j, j∈J.

The current material ordering plan is shown by the following table.

	Week				
	0	**1**	**2**	…	**n**
Material 1	$m_1(0)$	$m_1(1)$	$m_1(2)$	…	$m_1(n)$
Material 2	$m_2(0)$	$m_2(1)$	$m_2(2)$	…	$m_2(n)$
…	…	…	…	…	…
Material j	$m_j(0)$	$m_j(1)$	$m_j(2)$	…	$m_j(n)$
…	…	…	…	…	…

I-9 Actual Material Ordered

Given the table above, the actual material ordered in this week must be $m_j(0)$, ∀j∈J.

I-10 Determination of the Required Number of Weeks

Since we want to compute the material procurement plan for material j for periods 0 through N_j, we need to make sure we have values of the forecasts, shipment plan, and build plan far enough in the future to allow us to do so. This section shows how many periods of those plans we need to compute.

In 10 through 16 below, "$m_j(n)$ requires x(n)" should be read as, "Computing $m_j(n)$ requires values of x(0), x(1), …, x(n)." Thus 10 should be read as, "Computing $m_j(N_j)$ requires the values of $R_j(0)$, $R_j(1)$, …, $R_j(L_j{+}N_j)$."

From 9,

$$m_j(N_j) \text{ requires } R_j(L_j + N_j) \tag{10}$$

$$\text{and } m_j(N_j) \text{ requires } b(L_j + N_j). \tag{11}$$

From 10, 3, and 1,

$$m_j(N_j) \text{ requires } f(L_j + N_j + 13). \tag{12}$$

From 11 and 6,

$$m_j(N_j) \text{ requires } F(B + L_j + N_j) \tag{13}$$

$$\text{and } m_j(N_j) \text{ requires } s(B + L_j + N_j). \tag{14}$$

From13, 2, and 1,

$$m_j(N_j) \text{ requires } f(B + L_j + N_j + 13). \tag{15}$$

From 14, 5, and 1,

$$m_j(N_j) \text{ requires } f(B + L_j + N_j - (Y - S)). \tag{16}$$

Computation of N_b. From 11,

$$N_b = \max_{j \in J}\left\{L_j + N_j\right\}. \tag{17}$$

Computation of N_s. From 14,

$$N_s = \max_{j \in J}\left\{B + L_j + N_j\right\}. \tag{18}$$

Computation of N_f. From 12, 15, and 16,

$$N_f = \max_{j \in J}\begin{cases} L_j + N_j + 13 \\ B + L_j + N_j + 13 \\ B + L_j + N_j - (Y - S) \end{cases} \tag{19}$$

Since B ≥ 0, (Y – S) ≥ 0, the middle expression dominates, and 19 reduces to:

$$N_f = \max_{j \in J}\left\{B + L_j + N_j + 13\right\}. \tag{20}$$

Appendix II: Weekly Event Sequence

In the following table, periodically scheduled events are shown in sequence.

Event Time	Event Frequency	Initiators	Event Description
Monday 1:00	Weekly	Customers	Generate and send orders; these orders are received by the Adder factory at 9:30:00 the same day.
Monday 8:00	Weekly	Factory	Completes computing FGI safety stock for future weeks. Completes computing shipment plan and build plans.
Monday 9:00	Weekly	Factory	Completes computing material requirements plan. Completes computing material procurements plan.
Monday 10:00	Weekly	Factory	Generates current week's material orders. Material orders arrive at the vendors instantaneously.
Monday 10:00:01	Weekly	Vendors	Finish filling and shipping orders due this week. Shipments arrive at the factory instantaneously.
Monday 10:30	Weekly	Factory	Begins current week's production. Completes production started two weeks ago.
Friday 16:30	Weekly	Factory	Completes filling and shipping orders for the week.
Friday 23:58	Weekly	Simulation Executive	Records values of all the state variables.

Appendix III: Details of Part Commonality Experiments

The following table shows the definitions used to describe part commonality. MC stands for material cost, with uppercase denoting dollar values and lowercase denoting percentage values. m represents the set of material.

	Set of Material	Value of Material	Percentage Value
Common to products i and i–1	$m_{i,i-1}$	$MC_{i,i-1}$	$mc_{i,i-1} = \frac{MC_{i,i-1}}{MC_i} \times 100$
Unique to product i	$m_{i,i}$	$MC_{i,i}$	$mc_{i,i} = \frac{MC_{i,i}}{MC_i} \times 100$
Common to products i and i+1	$m_{i,i+1}$	$MC_{i,i+1}$	$mc_{i,i+1} = \frac{MC_{i,i+1}}{MC_i} \times 100$

Commonality occurs only between adjacent products. This implies that a part can be used in at most two products.

Each of the $MC_{i,j}$ is further broken up into class A, B, and C parts with relative values 50, 30, and 20 percent. Each of these classes is made up of 6, 10, and 14 week lead times with relative values 25, 40, and 35 percent. (See Table I on page 83.)

At the end of the product i life cycle, obsolete inventory (if any) should come only from parts in sets $m_{i,i}$ and $m_{i,i-1}$. Any leftover parts from $m_{i,i+1}$ can be used in product i+1. This implies that $mc_{i,i-1}$ and $mc_{i,i}$ impact the obsolete inventory at the end of the product life cycle for product i.

The values shown in the following table should be derived from the real bill of materials. For our experiments, we reverse the process, that is, we generate a bill of materials from the table, which was generated heuristically from the experimental scenarios, with the following constraints on the values of mc:

- For each i and j, $mc_{i,j}$ must be greater than or equal to 0 and less than or equal to 100.
- For each i, the sum of $mc_{i,j}$ over all j must be 100.
- In each experiment, if any $mc_{i,i+1}$ is zero, then $mc_{i+1,i}$ must also be zero.

Description of Experimental Scenarios

Run M-0: no part commonality at all between adjacent products.

Run M-1: 20% part commonality between adjacent products. The parts common to products i and i+1 make up 20% of the part values of both products. This may happen by a reduction in either part quantity or part cost, but the reason is not reflected in the dollar value of leftover inventory or material.

Run M-2: 20% part commonality when moving to a new product. The parts common to products i–1 and i make up 20% of the part value of product i; the rest of the value of product i is split equally between the parts unique to product i and those common to products i and i+1. Since product Adder-1 has no prior product, the value is split equally between unique parts and parts common to Adder-1 and Adder-2. 20% of the value of Adder-2 is made up of parts common to Adder-1 and Adder-2; the remaining 80% is split equally between unique parts and parts common to Adder-2 and Adder-3. 20% of the value of product Adder-4 is made up of parts common to Adder-3 and Adder-4; the balance of the value is unique parts since there are no succeeding products.

Run M-3: 50% and 25% part commonality between alternate products. There is 50% part commonality between products Adder-1 and Adder-2 and between Adder-3 and Adder-4; there is 25% part commonality between Adder-2 and Adder-3.

Run M-4: 50% part commonality between adjacent products; no unique parts in Adder-2 and Adder-3; 50% unique parts in Adder-1 and Adder-4.

Run M-5: 80% part commonality between succeeding products.

Part Commonality Data (%) for Multiple Product Crossover

i	Product	Demand (units)	Product Cost ($)	Common Parts (%)	Experiment Run					
					M-0	M-1	M-2	M-3	M-4	M-5
1	Adder-1	V	10,000	$mc_{1,1}$	100	80	50	50	50	20
				$mc_{1,2}$	0	20	50	50	50	80
2	Adder-2	1.3V	0.85 × 10,000	$mc_{2,1}$	0	20	20	50	50	80
				$mc_{2,2}$	100	60	40	25	0	10
				$mc_{2,3}$	0	20	20	25	50	10
3	Adder-3	1.3 × 1.3V	0.85 × 0.85 × 10,000	$mc_{3,2}$	0	20	20	25	50	80
				$mc_{3,3}$	100	60	40	25	0	10
				$mc_{3,4}$	0	20	40	50	50	10
4	Adder-4	1.3 × 1.3 × 1.3V	0.85 × 0.85 × 0.85 × 10,000	$mc_{4,3}$	0	20	20	50	50	80
				$mc_{4,4}$	100	80	80	50	50	20

Appendix IV: Details of Explanations for Experiments 0 and 1a

IV-1 Estimated Financial Impact Based on Theoretical Considerations for Experiment 0

The impact of product Adder on the financial situation of the enterprise, as explained on page 89, is:

- Total PCFT = $7,800,000
- Mature volume = MV = mature PCFT = $800,000/month or $200,000/week
- Consignment inventory = $300,000.

IV-2 Mature Demand Week Considerations for Experiment 0

RPI Material to Support Mature Demand

		Class A	Class B	Class C	All Classes
①	Percentage of Part Value in Product	50%	30%	20%	100%
②	Weekly Use during Mature Demand ① × MV	$100k	$60k	$40k	$200k
③	RPI Safety Stock in Weeks	4	8	16	N/A
④	RPI in $: ③ × MV	$400k	$480k	$640k	$1520k
⑤	RPI in Weeks of MV ④ ÷ MV	2	2.4	3.2	7.6

On-Order Material to Support Mature Demand

①	Lead Time	6 weeks	10 weeks	14 weeks	All Parts
②	Percentage of Part Value in Product	25%	40%	35%	100%
③	Weekly Order during Mature Demand ② × MV	$50k	$80k	$70k	$200k
④	Amount on Order = Weekly Order × Lead Time: ③ × ①	$300k	$800k	$980k	$2080k
⑤	Percent Value of Part on Order: ④ ÷ $2080k	14.4%	38.5%	47.1%	100%
⑥	On-order Material in Weeks of MV ④ ÷ MV	1.5	4.0	4.9	10.4

Total Inventory Metrics during Mature Demand

		Weeks of Mature Demand	Dollars
①	RPI	7.6	$1520k
②	WIP	2.0	$400k
③	FGI	2.0	$400k
④	On-Hand Inventory: ① + ② + ③	11.6	$2320k
⑤	On-Order Material	10.4	$2080k
⑥	Committed Inventory: ④ + ⑤	22.0	$4400k
⑦	Consignment Inventory	1.5	$300k
⑧	Total Committed Inventory: ⑥ + ⑦	23.5	$4700k

IV-3 End-of-Life Considerations for Experiment 0

Total PCFT = $7,800,000. Net profit = $78,000(i/100), where i is the profit as a percent of PCFT.

The following table summarizes the impact on the profitability of various margins i.

Write-Off as a Function of Profit on Shipped Units

①	Profit Margin i	5%	10%	20%	30%
②	Profit from Trade Units $7.8M × ①	$390k	$780k	$1560k	$2340k
③	Leftover Material	$64,615			
④	Leftover Material as % of Net Profit: ③ ÷ ②	16.57%	8.28%	4.14%	2.76%
⑤	Consignment	$300,000			
⑥	Consignment as % of Net Profit ⑤ ÷ ②	76.92%	38.46%	19.23%	12.82%
⑦	Total EOL Material as % of Net Profit: (③ + ⑤) ÷ ②	93.49%	46.75%	23.37%	15.58%

The following table shows the impact on Class C EOL material of reducing safety stock levels. These results were computed using means other than simulation.

Weeks of Class C Safety Stock	Class C EOL Material
16 weeks	$64,615
15 weeks	$35,385
14 weeks	$13,846
13 weeks	$0

IV-4 Why There Is Class C material Left Over for Experiment 0

The last period in which we expect to receive orders is week 68. The end of week 55 is 13 weeks before the end of the product life cycle. From the Adder order forecast in Fig. 2 on page 83 and the target RPI safety stock for class C material being 16 weeks of the 13-week leading average forecast (Table Ib on page 83), at the end of week 55 the amount of class C material in RPI should theoretically be 16/13 of the total demand to the end of life, or (16/13) × (1¾ × V) = (28/13) × V units, where V = 80.

In week 56, we need to start building the units for orders received in week 55. Ignoring the current FGI, the maximum new build from week 56 to the the end of life is equal to the demand from week 55 through the end of life, that is, 2V. Thus, at the end of week 55, there is more class C material on hand—enough to build (28/13) × V units—than needed for the demand to the the end of the product life cycle.

Remember that we did not consider units in FGI. If we want to reduce FGI units down to 0 by the end of the product life cycle, the total new build must be less than that computed above, and hence there will be even more class C material left over.

In summary, one reason for the leftover class C material is that the safety stock computation requires holding more class C raw material in RPI 13 weeks before the end of life than can be consumed by orders received in the last 14 weeks of the product life cycle.

IV-5 Why Orders Cannot Be More than 14 Weeks Late for Experiment 1a

Assume that an order comes in during week x. In the worst case we have not yet ordered any material for the unit that goes with this order. The earliest the material can be ordered is week x+1, and the longest lead time part will be delivered during week (x+1)+14, which is week x+15. Since build time is 2 weeks, the unit is ready in week x+17. Since transit time is 1 week, the unit is delivered to the customer in week x+18. Since the quoted availability is 4 weeks, on-time delivery means the customer should receive it in week x+4. This means that the lateness is 14 weeks.

HEWLETT-PACKARD JOURNAL INDEX

Volume 45 January 1994 through December 1994

Part 1: Chronological Index

February 1994

High-Quality Color Inkjet Office Printers, *Douglas R. Watson and Hatem E. Mostafa*

Laser-Comparable Inkjet Text Printing, *Jaime H. Bohórquez, Brian P. Canfield, Kenneth J. Courian, Frank Drogo, Corrina A.E. Hall, Clayton L. Holstun, Aneesa R. Scandalis, and Michele E. Shepard*

An Inside View of the Drop Generation Process

Modifying Office Papers to Improve Inkjet Print Quality

High-Quality Inkjet Color Graphics Performance on Plain Paper, *Catherine B. Hunt, Ronald A. Askeland, Leonard Slevin, and Keshava A. Prasad*

Polyester Media Development for Inkjet Printers, *Daniel L. Briley*

Inkjet Printer Print Quality Enhancement Techniques, *Corinna A.E. Hall, Aneesa R. Scandalis, Damon W. Broder, Shelley I. Moore, Reza Movaghar, W. Wistar Rhoads, and William H. Schwiebert*

The Third-Generation HP Thermal InkJet Printhead, *J. Stephen Aden, Jaime H. Bohórquez, Douglas M. Collins, M. Douglas Crook, André García, and Ulrich E. Hess*

Development of the HP DeskJet 1200C Print Cartridge Platform, *the Platform Development Team*

Print Cartridges for a Large-Format Color Inkjet Drafting Plotter

Environmentally Friendly Packaging

HP DeskJet 1200C Printer Architecture, *Kevin M. Bockman, Anton Tabar, Erol Erturk, Robert R. Giles, and William H. Schwiebert*

CAD System Organization

Product Design Effect on Environmental Responsibility and Distribution Costs

A New Product Development Model

Print Cartridge Fixturing and Maintenance in the HP DeskJet 1200C Printer, *Michael T. Dangelo, Reza Movaghar, and Arthur K. Wilson*

Media Path for a Small, Low-Cost, Color Thermal Inkjet Printer, *Damon W. Broder, David C. Burney, Shelley I. Moore, and Stephen B. Witte*

Stepper Motor Simulation Model

Automated Assembly and Testing of HP DeskJet 1200C Print Cartridges, *William S. Colburn, Randell A. Agadoni, Michael M. Johnson, Edward Wiesmeier, III, and Glen Oldenburg*

Connectivity of the HP DeskJet 1200C Printer, *Anthony D. Parkhurst, Ramchandran Padmanabhan, Steven D. Mueller, and Kirt A. Winter*

April 1994

Development of a Multimedia Product for HP Workstations, *Gary P. Rose, Jeffery T. Oesterle, Joseph E. Kasper, and Robert J. Hammond*

HP MPower: A Collaborative Multimedia Environment, *William R. Yoder*

X Stations in HP MPower

The HP Instant Ignition Program

Diagnosing and Reporting Problems in the Multimedia Environment

A Graphical User Interface for a Multimedia Environment, *Charles V. Fernandez*

HP SharedX: A Tool for Real-Time Collaboration, *Daniel Garfinkel, Bruce C. Welti, and Thomas W. Yip*

X Window System Client/Server Architecture

Graphics Glossary

Whiteboard: A New Component of HP SharedX

Imaging Services in a Multimedia Environment, *Andrew Munro and Ahmad H. Shekarabi*

HP Image Library Scaling Functions

A Printing Solution for a Multimedia Environment, *John Mandler*

Faxing Documents in HP MPower, *Francis P. Sung and Mark A. Johnson*

Audio Support in HP MPower, *Ellen N. Brandt, Thomas G. Fincher, and Monish S. Shah*

Overview of A-law and μ-law Data Formats

Video Support in a Multimedia Environment, *Craig S. Richard*

Mail Facilities in a Multimedia Environment, *Robert B. Williams, Harry K. Phinney, and Kenneth L. Steege*

MIME Header Fields

A Fast and Intuitive Online Help System, *Michael R. Wilson, Lori A. Cook, and Steven P. Hiebert*

WYSIWYG Printing in an X Application

Developing Online Application Help, *Dex Smith*

June 1994

Corporate Business Servers: An Alternative to Mainframes for Business Computing, *Thomas B Alexander, Kenneth G. Robertson, Dean T. Lindsay, Donald L. Rogers, John R. Obermeyer, John R. Keller, Keith Y. Oka, and Marlin M. Jones, II*

Package Design Using 3D Solid Modeling

PA-RISC Symmetric Multiprocessing in Midrange Servers, *Kirk M. Bresniker*

SoftBench Message Connector: Customizing Software Development Tool Interactions, *Joseph J. Courant*

August 1994

October 1994

December 1994

Part 2: Subject Index

Part 3: Product Index

Part 4: Author Index

Kaplinsky, George	Feb.
Karube, Koji	Oct.
Kasper, Joseph E.	Apr.
Keller, John R.	June
Kelly, Christopher P.J.	Oct.
Kilcrease, Catherine L.	Aug.
Kinsho, Kenji	Oct.
Kishida, Akito	Oct.
Kolts, Bertram S.	Oct.
Kopp, Siegfried	Aug.
Kroboth, Robert H.	Oct.
Krucky, Jan	June
Landis, Adele S.	Aug.
Lee, Daniel T.L.	Dec.
Lindsay, Dean T.	June
Little, Rob	Feb.
Magenis, Sue	Apr.
Mahn, Johannes	Aug.
Mandler, John	Apr.
McClellan, Paul J.	Aug.
Miller, Christopher M.	Aug.
Mooney, Matthew G.	Aug.
Moore, Shelley I.	Feb.
Mostafa, Hatem E.	Feb.
Movaghar, Reza	Feb.
Mueller, Steven D.	Feb.
Mujtaba, M. Shahid	Dec.
Munro, Andrew	Apr.
Nathan, Connie	Aug.
Obermeyer, John R.	June
Oesterle, Jeffery T.	Apr.
Oka, Keith Y.	June
Oldenburg, Glen	Feb.
Owens, Ronnie E.	Oct.
Padmanabhan, Ramchandran	Feb.
Panah, Tony	Feb.
Parkhurst, Anthony D.	Feb.
Patton, Charles M.	Aug.
Peterson, John V.	Apr.
Pfaff, Andreas M.R.	Dec.
Phinney, Harry K.	Apr.
Prasad, Keshava A.	Feb.
Proft, Conrad R.	Oct.
Raak, Gerald I.	Oct.
Reid, Bruce	Feb.
Riedel, Ronald J.	Oct.
Rhoads, W. Wistar	Feb.
Richard, Craig S.	Apr.
Ritter, Robert	Dec.
Robertson, Kenneth G.	June
Rogers, Donald L.	June
Rose, Gary P.	Apr.
Scandalis, Aneesa R.	Feb.
Scheffelin, Joe	Feb.
Schwegler, Tim	Aug.
Schwiebert, William H.	Feb.
Shah, Monish S.	Apr.
Shekarabi, Ahmad H.	Apr.
Shepard, Michele E.	Feb.
Shih, Jonathan C.	Dec.
Simms, Mark J.	Dec.
Slevin, Leonard	Feb.
Smith, Dex	Apr.
Steege, Kenneth L.	Apr.
Sung, Francis P.	Apr.
Tabar, Anton	Feb.
Thoman, Jeff	Feb.
Timm, Dale	Feb.
Tousi, Susan H.	Feb.
Towery, David	Feb.
Trezise, Greg K.	Dec.
Ujvarosy, Damon R.	Dec.
Van Liew, Amy	Feb.
Vasta, John R.	Dec.
Village, Rodney K.	Oct.
Warren, Richard E.	Oct.
Watson, Douglas R.	Feb.
Welti, Bruce C.	Apr.
Wiesmeier, Edward, III	Feb.
Williams, Michael K.	Dec.
Williams, Robert B.	Apr.
Wilson, Arthur K.	Feb.
Wilson, Martha Grewe	Aug.
Wilson, Michael R.	Apr.
Winter, Kirt A.	Feb.
Witte, Stephen B.	Feb.
Yip, Thomas W.	Apr.
Yoder, William R.	Apr.
Yamamoto, Akio	Dec.
Yonekura, Takanori	Oct.