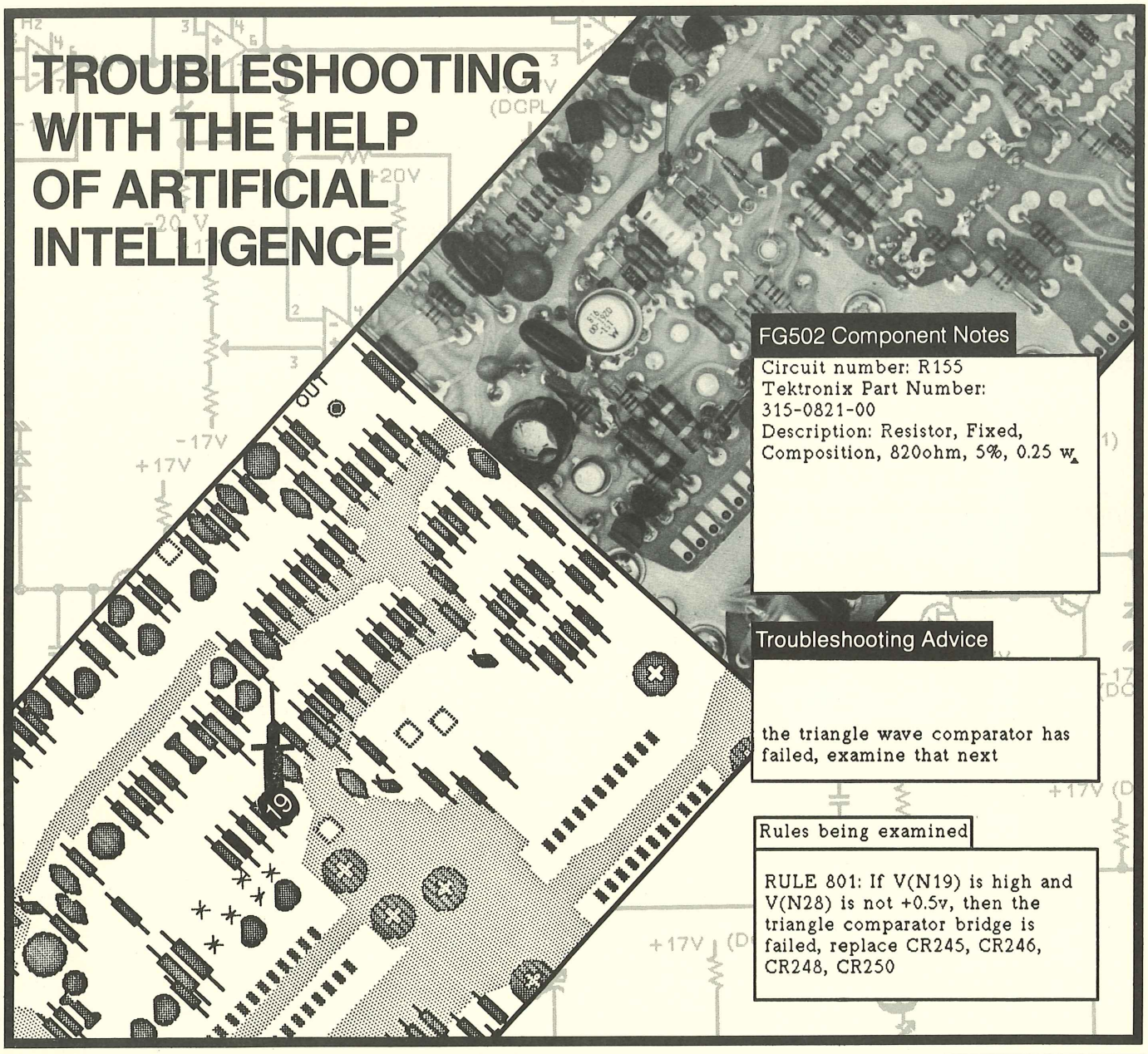
TEK INFORMATION FOR TECHNOLOGISTS

SEPT/OCT 1984

TECHNOLOGY report

TROUBLESHOOTING WITH THE HELP OF ARTIFICIAL INTELLIGENCE



Tektronix®
COMMITTED TO EXCELLENCE

CONTENTS

Volume 6, No. 7, September/October 1984. Managing editor: Art Andersen, ext. MR-8934, d.s. 53-077. Cover: Nancy Pearen; Graphic illustrator: Darla Olmscheid. Composition editor: Sharlet Foster. Published for the benefit of the Tektronix engineering and scientific community.

Why TR?

Technology Report serves two purposes. Long-range, it promotes the flow of technical information among the diverse segments of the Tektronix engineering and scientific community. Short-range, it publicizes current events (new services available and notice of achievements by members of the technical community).

HELP AVAILABLE FOR PAPERS, ARTICLES, AND PRESENTATIONS

If you're preparing a paper for publication or presentation outside Tektronix, the Technology Communications Support (TCS) group of Corporate Marketing Communications can make your job easier. TCS can provide editorial help with outlines, abstracts, and manuscripts; prepare artwork for illustrations; and format material to journal or conference requirements. They can also help you "storyboard" your talk, and then produce professional, attractive slides to go with it. In addition, they interface with Patents and Trademarks to obtain confidentiality reviews and to assure all necessary patent and copyright protection.

For more information, or for whatever assistance you may need, contact Eleanor McElwee, ext. 642-8924. □

WRITING FOR *TECHNOLOGY REPORT*

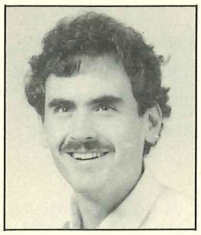
Technology Report can effectively convey ideas, innovations, services, and background information to the Tektronix technological community.

How long does it take to see an article appear in print? That is a function of many things (the completeness of the input, the review cycle, and the timeliness of the content). But the minimum is six weeks for simple announcements and as much as 14 weeks for major technical articles.

The most important step for the contributor is to put the message on paper so we will have something to work with. Don't worry about organization, spelling, and grammar. The editors will take care of that when we put the article into shape for you.

Do you have an article to contribute or an announcement to make? Contact the editor, Art Andersen, 642-8934 (Merlo Road) or write to d.s. 53-077. □

ARTIFICIAL INTELLIGENCE: TROUBLESHOOTING WITH THE HELP OF AN EXPERT SYSTEM



Jim Alexander is a Cognitive Scientist in the Computer Research Laboratory. He joined Tek in 1983. Prior to joining Tek, he was working on his PhD at the University of Colorado in Boulder. His research there focused upon the characteristics of human-computer interaction. Before Boulder, Jim worked at the Bell Telephone Laboratories doing system analysis and requirements. Jim received his BS from Syracuse University, and his MA from the University of Colorado.

Michael Freiling coordinates the Troubleshooting Technology Project in the Artificial Intelligence Department, part of the Computer Research Lab. Mike joined Tek in 1983 from Oregon State University where he was an assistant professor of computer science. He has a PhD from the Massachusetts Institute of Technology (1977). He was a Luce scholar at Kyoto University (Japan) from 1977 to 1978. His BS is from the University of San Francisco.

Good troubleshooters can't be replaced by expert systems, but elements of their expert wisdom can be confided to *expert systems.* With this wisdom, the expert system can "sit in" with the junior technician, advising and suggesting how to diagnose what's wrong as the technician faces a "broken" Tek product. This article describes the *Troubleshooting Assistant,* a Tek-built system, and the effects expert-system technology might have on servicing electronic products.

For Tek, electronic-systems troubleshooting is both a major source of revenue and a major cost of doing business. Consequently, when the Artificial Intelligence Department was formed last year, we decided that we could improve revenues and costs most effectively by applying artifical intelligence to the challenge of diagnosing electronic equipment.

Expert System Technology

For more than a decade, researchers in artificial intelligence (AI) have been developing specialized computer systems known as expert systems. The technology is now emerging as a practical tool for use in real-world applications. For example, MYCIN, one of the first expert systems (Shortliffe, 1976), is now helping doctors diagnose and treat meningitis, a serious and difficult to diagnose disease of the spinal fluid. The MYCIN system has proved to be as accurate a diagnostician as most physicians. Consequentally, many physicians use MYCIN as a source for second opinions.

A system such as MYCIN would be virtually unthinkable if the development followed traditional software methodologies built around Pascal or FORTRAN. The innovation that made MYCIN and other expert systems practical was the development of *knowledge engineering,* a new software development technology. Systems built with knowledge-engineering techniques take the form of a large group of rules. The rules usually take the form IF ⟨observation⟩ THEN ⟨conclusion⟩ (see *Rule Base Examples*). The only program, in the traditional sense, is a small rule interpreter which processes the rules.

Rule Base Examples

IF ⟨all waves bad⟩ THEN ⟨triangle generator bad⟩

IF ⟨only sine wave bad⟩ THEN ⟨sine shaper bad⟩

Knowledge engineers construct an expert system by interviewing experts in a field, attempting to build a body of rules which capture the expertise necessary for a task.

Expert-system technology, then, primarily differs from more traditional programming in two ways:

(One) Facts are represented in a rule base rather than in source code.

Unlike most software systems, where source-code development is a major part of the programming effort, the actual programming for an expert system involves writing only a small rule processor. The lion's share of the work of building an expert system is in developing the rule base, a set of rules describing how to solve the problem. Because a rule base may contain thousands of rules, its behavior can be quite complex.

(Two) Decisions are based on the intuitition and experiences of experts rather than on any formal theory.

Often, expert systems do not rely on the theoretical principles of the problem under attack. Instead, current AI technology requires us to build a system based upon the way experts *think* about the problem. It is important to note that experts seldom solve problems using a rigorous theoretical analysis; rather their understanding has a more *ad hoc* character. Some excellent troubleshooting technicians, for instance, use very little electronic theory. Yet, by using their own simpler *conceptual models,* they can troubleshoot a faulty device quite efficiently.

The Smalltalk Troubleshooting Assistant Project

We have built a prototype expert system engineered to assist in the repair of Tektronix FG502 Function Generators. This system, the *Troubleshooting Advisor,* runs in Smalltalk (Goldberg and Robson, 1983) on the Tektronix 4404 Artificial Intelligence System. We spent about five man months researching the troubleshooting process within Tektronix and three man-months writing code in Smalltalk and developing the rule base. Although this effort focused on the very simple FG502, we were able to develop a general framework for troubleshooting that we will eventually apply to more complex Tektronix equipment.

The first stage of the Advisor project was to visit as many troubleshooting experts within Tek as we could find. With the experts' help we began to characterize electronic troubleshooting within Tek (Freiling and Alexander, 1984). This investigation was critical since knowledge engineering relies heavily upon the expert's knowledge to be successful.

The need for expert knowledge obviously meant that we needed to select a topic for which experts were available – but not too available. We needed an area where expertise was also scarce and therefore valuable. Equally important, we needed an expert whose style of explaining things was effective and understandable.

Generally, there are four levels of troubleshooters at Tektronix; technical responsibilities and abilities are divided accordingly.

When a problem instrument comes into the shop or off the assembly line it is first given to a novice technician who attempts to calibrate the instrument and identify problems. If this technician cannot identify the fault, he or she passes it to a more experienced technician. The pass-along chain continues until either the instrument is fixed or determined to be unfixable by the most experienced technician. (The technicians we spoke to assured us that no Tektronix instrument has ever remained unfixable – every instrument is fixed!) It is the responsibility of each technician who has handled an instrument to learn from the more experienced technician how the instrument was fixed. Thus, training by apprenticeship is melded with the troubleshooting process.

What an expert system can do

Early in our research, we became convinced that the role of an expert troubleshooting system was not to replace, but to help the technician. We saw more effectiveness in using expert systems to increase the productivity of existing technicians – especially in the early stages of their careers.

Troubleshooting is a complex and difficult task. A troubleshooter must know the device under repair intimately. He or she must also have a fair amount of electronic savvy: AI technology is not robust enough to handle all of the factors that are involved in troubleshooting. Current methods permit us to develop systems with knowledge about only the most basis types of failure.

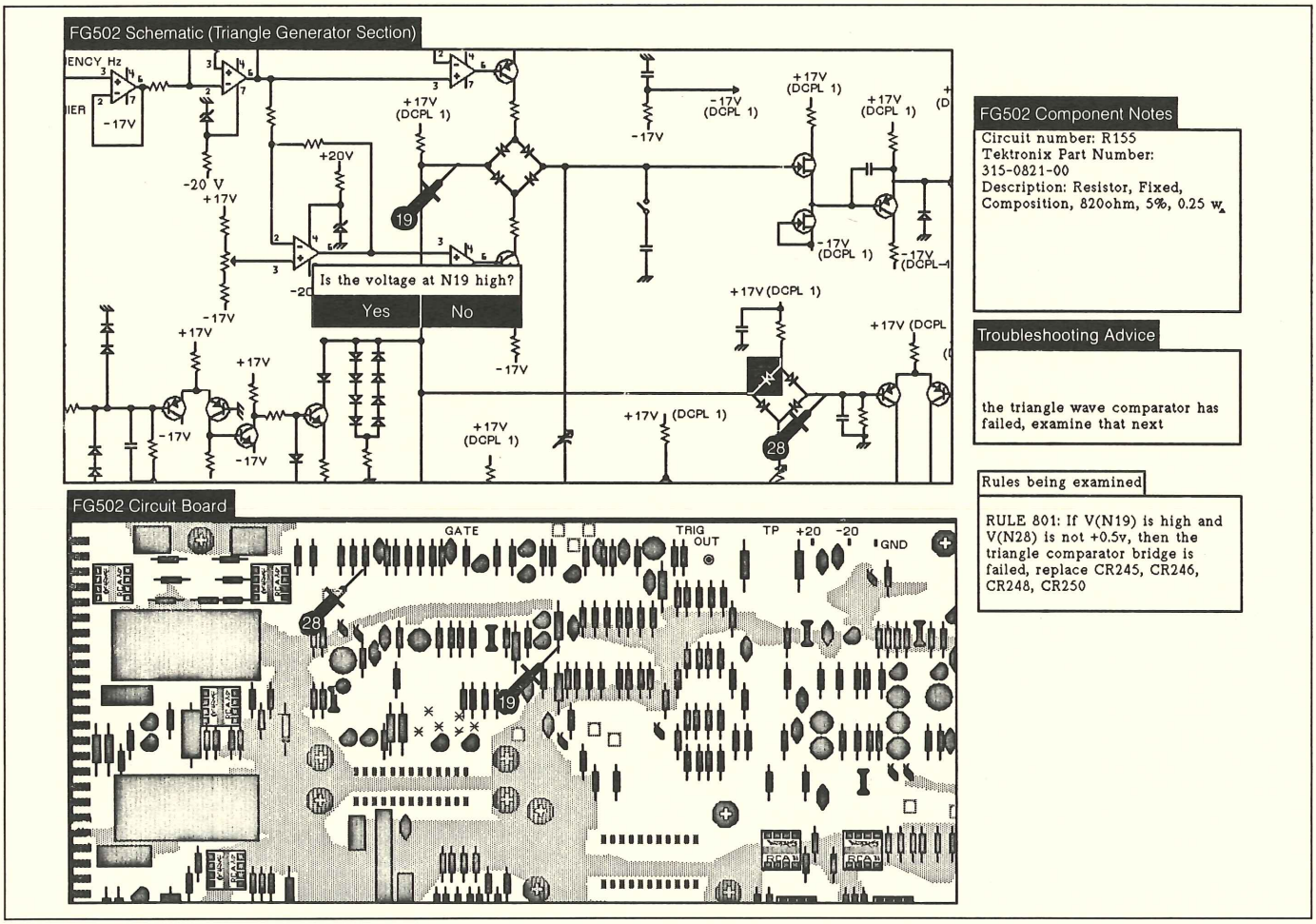


Figure 1. The Troubleshooting Assistant displays the schematic and circuit board for the FG502 during a diagnosic step. Notice the "probes" labeled 19 and 28 and the question about voltage at N19 (node 19). The technician has answered "no." The Advisor suggests, in the advice window, that the comparator has failed. Below this, in the rules window, the advisor is conveying its "reasoning" and giving how-to-fix advice.

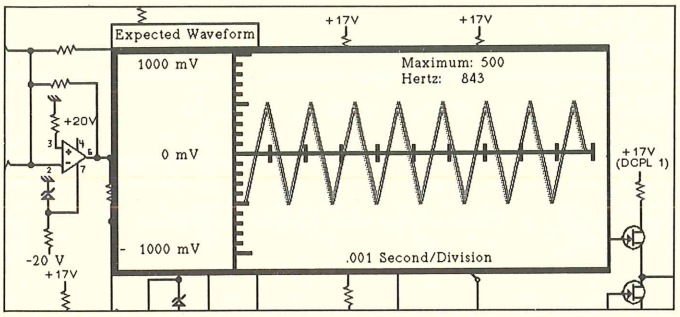


Figure 2. A simulated oscilloscope waveform shows the correct waveform expected at a given node.

We discovered that novice troubleshooters spend most of their time trying to figure out how an instrument works, and then trying to figure out how to isolate problems within the instrument. Usually, the novice is not successful with problems that he or she has not seen before and has to go to a seasoned technician for help. Thus, repairing the instrument commands the time of *two* technicians, even if the problem is simple.

> Early in our research, we became convinced that the role of an expert troubleshooting system was not to replace the technician.

Expert systems can save technicians a lot of time; they can lead inexperienced troubleshooters towards diagnosing a faulty component just as an experienced troubleshooter would. One role of the expert system would be as a guide to the novice technician. Another role would be as a consultant to the junior technician trying to solve unfamiliar problems; this would allow senior technicians to concentrate on the more difficult troubleshooting tasks.

To the experienced troubleshooter, there are other benefits. A complete troubleshooting system – one that would handle any Tek-built equipment – would be invaluable to these technicians because it would archive and make easily available the total troubleshooting knowledge of all instruments.

This archiving would be particularly valuable with older products. Since Tektronix will service any Tek instrument built within the last nine years, Service-Center technicians face a plethora of instruments, more than 1500 different products. They often encounter "unknowns," that is, instruments unfamiliar to anyone in their shop. Technicians spent a lot of time on these unknowns trying to figure out what the instrument is supposed to do. A complete troubleshooting archive would tell them that. They could immediately start fixing the product, without reading manuals and other documentation.

In short, Troubleshooting Assistants can provide a whole set of benefits for technicians ranging from training to memory enhancement.

The Prototype Troubleshooting Assistant

We selected the FG502 Function Generator as our first target because it is one of the simplest instruments that Tek builds, yet it is representative of most of Tek's analog products. Drawing on the expertise of Jim Mauck (Field Service, Rockville, MD) and Dennis Feucht (Electronic Systems Lab), we were able to build a system which would demonstrate the potential of expert system technology.

We built the Trobleshooting Assistant on the Magnolia Workstation, and programmed it in the Smalltalk language. The Magnolia has a bit-mapped display, which combined with the excellent graphics of Smalltalk enabled us to build a very nice user interface. The Advisor has been ported to the Tektronix 4404 AI System.

Figure 1 is a laser copy of a typical display that a technician would see on screen while trying to fix the FG502. Notice that there is a drawing of the circuit board, a circuit schematic, and three other windows; these windows present text.

The first feature of the Assistant is *schematic-to-board cross referencing.* Using a mouse, the user moves a pointer to any component on the schematic and clicks a mouse-button; this selects the component pointed to. The selected component reverses its color on the schematic, and on the drawing of the circuit board. In addition, the component's specifications will appear in the top right-hand corner window. Because no reference to a printed manual is involved, the technician can focus on interpreting the circuit; the computer takes care of finding the part on the board.

The Assistant also allows the user to ask about how the circuit is supposed to operate. Users can point to a node in the circuit and select "waveform" from a menu. The system will then display a picture of the waveform expected at that location. Figure 2 shows how the system will present the correct waveform for a given point. The technician can then put a probe on that spot in the actual instrument and compare the measurement with the one displayed.

> Some excellent troubleshooting technicians use very little electronic theory. Yet, by using their own narrower *conceptual models,* they can troubleshoot a faulty device quite efficiently.

The technician can ask for assistance in troubleshooting by selecting an "assist me" item from a menu. The system will ask a series of questions (left center in figure 1) the answers to which will eventually identify the faulty component and highlight it on the schematic and circuit-board representation. As the consultation progresses, the technician uses the other two windows seen on the screen in figure 1. Troubleshooting advice is given via the middle window; in the bottom window we show the current rule being considered by the rule interpreter. When the system requires a measurement, it displays a probe symbol at the point on the schematic; this tells the user where to place an actual scope probe in the faulty instrument.

Finally, the system generates a report characterizing the failure and the steps taken to fix it. This report can be used to track the failure rate of parts, and the accuracy of the diagnostic system itself. Figure 3 shows the report generated at the end of a consultation.

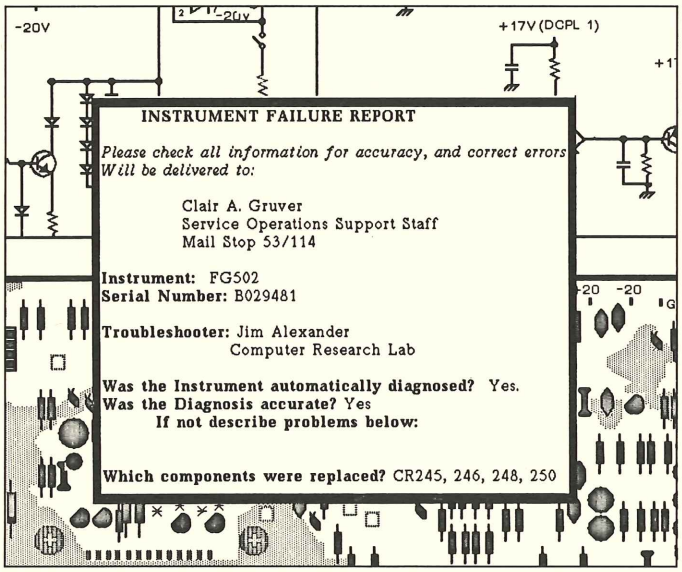


Figure 3. The Advisor automatically generates a report at the end of the "consultation." This report can be used in reliability analysis and in evaluating the performance of the expert system itself.

Is The System Complete?

The prototype system was built to handle about 80% of the most frequent problems encountered by troubleshooters working on FG502s. Although we have not tested it in the field, it appears the techniques used will isolate component failures well enough to assist junior technicians. We are now working on a Troubleshooting Assistant for the 2213 Oscilloscope. This Assistant will be thoroughly field tested.

The FG502 Troubleshooting Assistant was built to handle problems specific to the FG502. Ultimately, expert-system technology will not be effective until we can build tools that quickly adapt an expert system to troubleshoot a variety of instruments.

We are looking at ways to generate a knowledge base with more general rules of troubleshooting, and to allow expert technicians to input troubleshooting rules and facts directly, without the help of a knowledge engineer.

When we have tools for developing troubleshooting systems rapidly, we will be able to build troubleshooting assistants to help technicians repair most, if not all, Tek products. These systems will:

- Significantly increase novice-technician productivity
- Help train both novice and junior technicians
- Store and make readily available knowledge about repairing older Tek products

In short – these systems will further increase the efficiency of Tek's top-notch service organizations.

For More Information

For more information, call Jim Alexander 627-6159 (50-662) or Mike Freiling 627-2522 (50-662).

Acknowledgements

We would like to give special thanks to Dennis Feucht, Todd Paulus, Dick Butler, Tim Kiser, Jim Mauck, and Ed Ohlman for their generous assistance.

References

1. Freiling, M. and Alexander, J., *Survey of Knowledge Engineering Opportunities at Tektronix,* 1983; Applied Research Technical Report, CR-84-06; Tektronix, Inc., Beaverton, OR, 1984.

2. Goldberg, A. and Robson, D., *Smalltalk-80: The language and Its Implementation,* Addison-Wesley, Reading, MA, 1983.

3. Shortliffe, E.H., *Computer-Based Medical Consultations: MYCIN,* American Elevier, New York, NY, 1976.

THERMAL ANALYSIS TOOLS

THERMAL COMPUTATIONS DETAILED IN NEW BOOK

Gordon Ellison, Applications, part of Computer Resources

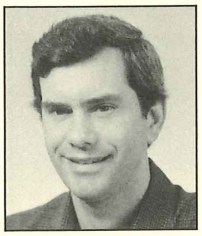
My recent book, *Thermal Computations for Electronic Equipment,* published by Van Nostrand Reinhold, Inc., is now available. You can order this book using the MANUAL program on CYBER or by calling Hazel Ade at 627-1771. The book costs $37.95.

Since the book also contains the program instructions for TAMS (Thermal Analyzer for Multilayer Structures) and TNETFA (Time Dependent Network and Flow Analyzer), individual manuals for these programs will not be stocked. For program-access instructions, see HELP,TAMS and HELP,TNETFA.

I recommend that TAMS users also check into the TAMS interactive preprocessor TIP.

Thermal consulting services are available, as usual, by calling Gordon Ellison at 627-6441 or Nancy Elliott, 627-6593. □

TI's TMS32010:
IMPLEMENTING A FILTER USING A DIGITAL SIGNAL PROCESSOR



Jim Fenton is an evaluation engineer in the Logic Analyzer Division, part of the Design Automation Group. Jim joined Tektronix in 1980 after receiving his BSEE from the University of Michigan. He is currently working on his MSEE from Oregon State University through the Tektronix Education Program.

Many analog functions can now be implemented with readily available digital parts. Digital filtering, spectrum analysis, fast fourier transforms, and speech processing are just a few of the "analog" applications for digital signal-processing (DSP) hardware. This article describes how a low-pass filter was implemented using the Texas Instrument TMS32010 Digital Signal Processor.

There are two types of linear shift-invariant systems: *finite-duration impulse response* (FIR) and *infinite-duration impulse response* (IIR).

Shift-Invariant Systems

Shift-invariant systems are characterized by the property:

if y(n) is the response to x(n),
then y(n – k) is the response to x(n – k),

where k is a positive or negative integer.

When the index n is associated with time, shift-invariance corresponds to time-invariance.

IIR systems are generally difficult to implement using the convolution-sum expression to compute the output. There are, however, systems that can be implemented using a recursive algorithm to compute the output from the input sequence and *n* previously computed output samples. These systems satisfy a linear-constant coefficient difference equation of the form:

$$y(n) = \sum_{k=1}^{N} a_k\, y(n-k) + \sum_{k=0}^{M} b_k\, x(n-k)$$

The flow graph for the computational procedure for implementing this type of difference equation is shown in figure 1.

This article describes how we implemented this type of linear constant-coefficient difference equation using the TMS32010.

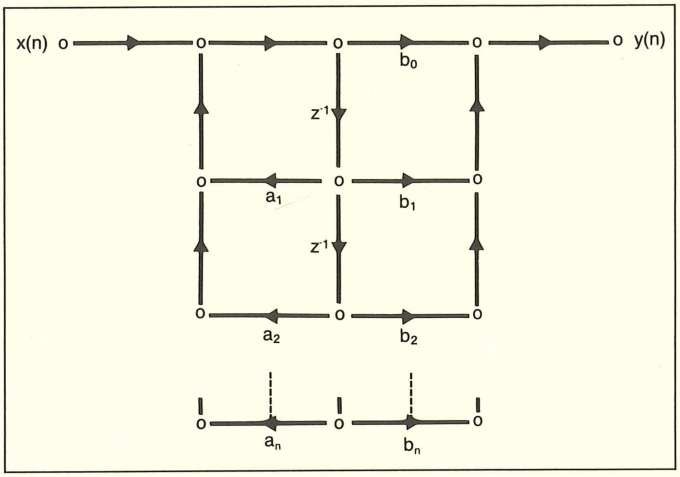


Figure 1. This flow graph shows the computational procedure used to implement the linear constant-coefficient difference equation.

TMS32010 Chip Architecture

The TMS32010 is a 16/32-bit single-chip microcomputer that can execute five-million instructions per second. For speed and flexibility, the TMS32010 uses a modified Harvard architecture. (Harvard architecture separates program memory from the data memory.) The DSP chip's structure allows it to transfer information between program and data memory, giving the designer more flexibility in implementing the multiplication of coefficients. (See figure 2 for a diagram of the TMS32010.) *Note: In this article, "DSP," "chip," and "TMS3210" are used interchangeably.*

Because the chip uses a modified Harvard architecture, it does not need data ROM. Data coefficients can be accessed from program ROM and stored in data RAM. A 12-bit program counter (PC) addresses the 1536 × 16-bit word program ROM and the 4 × 16-bit word stack. Registers AR0 and AR1 (figure 2) are auxiliary registers that can be used as hardware-loop counters; an auxiliary register pointer selects which auxiliary register is active. Data memory is in a 144 × 16-bit on-chip RAM. Instruction operands are fetched from this RAM.

The chip contains a 32-bit ALU and accumulator that support double-precision arithmetic. The ALU can perform a variety of Boolean operations, providing high-speed bit manipulation. Feeding into the ALU via a multiplexor is a 0- to 15-bit barrel shifter, which can shift data to the left *before* it is loaded into the accumulator. Another shifter, located after the accumulator, can

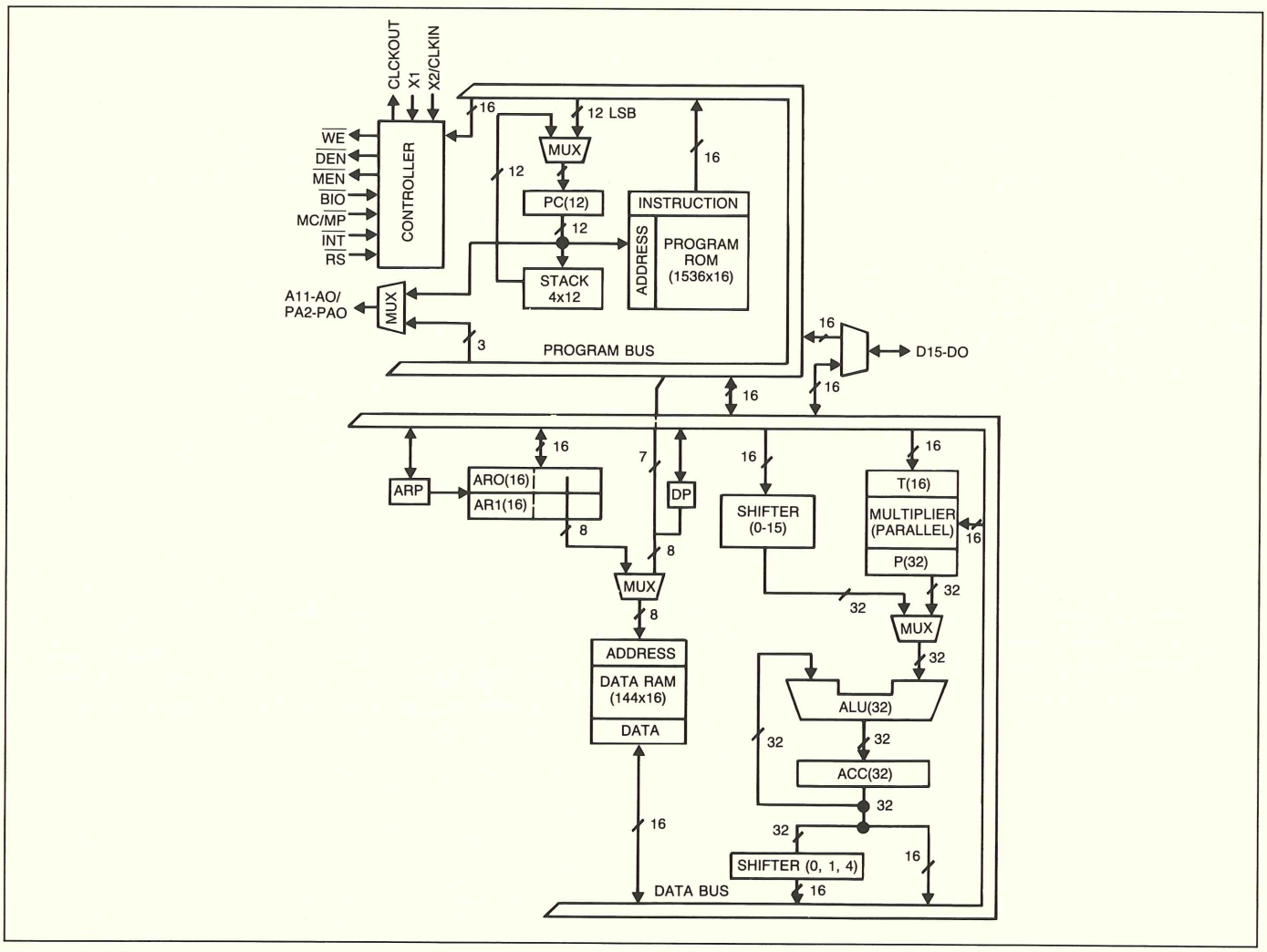


Figure 2. The organization of the Texas Instrument TMS32010 Digital Signal Processor (DSP).

shift the data 0, 1, or 4 places to the left *after* the data is transferred out of the accumulator. A 16 × 16-bit multiplication can be performed by the parallel multiplier in one 200-ns cycle. A 16-bit T register holds the multiplicand and the P register stores the 32-bit result.

One of the nice features of the TMS32010 is its ability to execute instructions from an off-chip ROM at full speed. The MC/MP pin enables the user to tailor his or her system to the memory configuration desired.

Data can be transferred to and from the TMS32010 via the 16-bit parallel data bus. The chip can interface up to eight 16-bit multiplexed input ports and eight 16-bit multiplexed output ports. (See figure 3 for an example of how to interface to an external device.) In addition to the input/output ports, the BIO ~ pin on the TMS32010 allows a polling input for bit test and jump operations. There is also an interrupt pin that allows the DSP to handle multitasking operations.

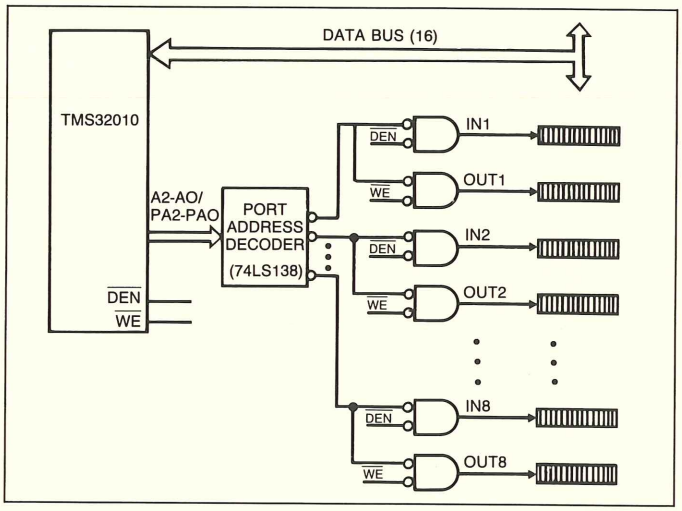


Figure 3. The external-device interface for the TMS32010 Digital Signal Processor.

Instruction Set

The TMS32010's comprehensive instruction set lends itself to both numeric-intensive operations and general-purpose instructions. These instructions are broken down into these sets:

- Accumulator instructions
- Auxiliary registers and data-page pointer instructions
- T register, P register, and multiply instructions
- Branch instructions
- Control instructions
- I/O and data memory operations

Tables 1 and 2 summarize the instruction sets.

The TMS32010 has three main addressing modes: direct, indirect, and immediate. In *direct addressing,* seven bits of the instruction word are concatenated with the data-page pointer to form the data-memory address. *Indirect addressing* forms the data-memory address from the least-significant eight bits of one of the two auxiliary registers, AR0 and AR1. In *immediate addressing,* data are derived from part of the instruction word rather than from the RAM. A few very useful immediate instructions are *multiply immediate* (MPYK), *load accumulator immediate* (LACK), and *load auxiliary register immediate* (LARK).

TABLE 1 — TMS32010 INSTRUCTION SUMMARY

MNEMONIC	DESCRIPTION	NO. OF CYCLES	NO. OF WORDS
ACCUMULATOR INSTRUCTIONS			
ADD	Add to accumulator with shift	1	1
SUB	Subtract from accumulator with shift	1	1
LAC	Load accumulator with shift	1	1
SACL	Store low-order accumulator bits with shift	1	1
SACH	Store high-order accumulator bits with shift	1	1
ADDH	Add to high-order accumulator bits	1	1
ADDS	Add to accumulator with no sign extension	1	1
SUBH	Subtract from high-order accumulator bits	1	1
SUBS	Subtract from accumulator with no sign extension	1	1
SUBC	Conditional subtract (for divide)	1	1
ZALH	Zero accumulator and load high-order bits	1	1
ZALS	Zero accumulator and load low-order bits	1	1
LACK	Load accumulator immediate	1	1
ABS	Absolute value of accumulator	1	1
ZAC	Zero accumulator	1	1
XOR	Exclusive OR with accumulator	1	1
AND	AND with accumulator	1	1
OR	OR with accumulator	1	1
AUXILIARY REGISTERS AND DATA PAGE POINTER INSTRUCTIONS			
SAR	Store auxiliary register	1	1
LAR	Load quxiliary register	1	1
MAR	Modify auxiliary register and pointer	1	1
LDPK	Load data memory page pointer immediate	1	1
LDP	Load data memory page pointer	1	1
LARK	Load auxiliary register immediate	1	1
LARP	Load auxiliary register pointer immediate	1	1
T REGISTER, P REGISTER, AND MULTIPLY INSTRUCTIONS			
LT	Load T Register	1	1
LTA	Load T Register and accumulate product	1	1
LTD	Load T Register, accumulate product, and move date in memory forward one address	1	1
MPY	Multiply with T Register, store product in P Register	1	1
PAC	Load accumulator from P Register	1	1
APAC	Add P Register to accumulator	1	1
SPAC	Subtract P Register from accumulator	1	1
MPYK	Multiply T Register with immediate operand, store product in P Register	1	1

Two special instructions of the TMS32010, TBLR (table read) and TBLW (table write), take advantage of the modified Harvard architecture. The TBLR instruction transfers words stored in program memory to the data RAM. This eliminates the need for a dedicated coefficient ROM. It also allows the user to choose how much program ROM should be dedicated to coefficients and how much to the program. The TBLW command transfers internal data RAM to external RAM. Here is an example of the use of the TBLR command:

TBLR DAT4 – This instruction transfers those items in program memory pointed to by the program counter to the specified data-memory location DAT4. (See "Implementation" for more details of this memory-operation instruction.)

Here are other instruction examples:

LT X(N) – This instruction loads the 16-bit T register with the contents of the specified data-memory location.

SACH TEMP,1 – This instruction stores the upper half of the accumulator's contents into the specified data-memory location after the data has been shifted left one-bit location.

MPY B3 – This instruction multiplies the contents of the T register and B3, and puts the 32-bit result into the P register.

TABLE 2 — TMS32010 INSTRUCTION SUMMARY (Continued)

MNEMONIC	DESCRIPTION	NO. OF CYCLES	NO. OF WORDS
BRANCH INSTRUCTIONS			
BANZ	Branch on auxiliary register not zero	2	2
BV	Branch on overflow	2	2
BIOZ	Branch on $\overline{BIO}$ = O	2	2
B	Branch unconditionally	2	2
BLZ	Branch if accumulator < 0	2	2
BLEZ	Branch if accumulator ≤ 0	2	2
BGZ	Branch if accumulator > 0	2	2
BGEZ	Branch if accumulator ≥ 0	2	2
BNZ	Branch if accumulator ≠ 0	2	2
BZ	Branch if accumulator = 0	2	2
CALL	Call subroutine immediate	2	2
CALA	Call subroutine from accumulator	2	1
RET	Return from subroutine	2	1
CONTROL INSTRUCTIONS			
LST	Load status register	1	1
SST	Store status register	1	1
NOP	No operation	1	1
DINT	Disable interrupt	1	1
EINT	Enable interrupt	1	1
ROVM	Reset overflow mode	1	1
SOVM	Set overflow mode	1	1
POP	Pop stack to accumulator	2	1
PUSH	Push stack from accumulator	2	1
I/O AND DATA MEMORY OPERATIONS			
IN	Input data from port	2	2
OUT	Output data to port	2	2
TBLR	Table read from program memory to data RAM	3	1
TBLW	Table write from data RAM to program memory	3	1
DMOV	Shift contents of data memory foward one address	1	1

Implementation: System Configuration

Figure 4 shows how we configured the digital filter. The 68000 and the DSP communicate by hand-shaking via interrupts. Because the 68000 microprocessor is not dedicated to only the filter, it can do other tasks while the A/D and D/A converters and the TMS32010 are crunching away on their tasks.

The program-flow diagrams (figures 5 and 6) detail how each processor handles data flow. Note that in the system configuration diagram (figure 4), the 20-kHz clock signal starts all filter processes. The sample rate is also 20 kHz, so that on every falling edge of the clock, the analog input is sampled. The data register is the communication register for the processors. The TMS32010 is set to the microcomputer mode by pulling high on the MC/MP ~ pin. We did this to fit the program instructions and coefficients in on-chip ROM.

Nyquist Rate

Since the sampling rate in our filter is 20 kHz, the highest valid-frequency component that can be sampled without aliasing is 10 kHz. As the block diagram in figure 4 indicates, an anti-aliasing filter with a 10 kHz-cutoff frequency is used to prevent aliased frequency samples.

Filter Specification

We implemented a third-order direct-form IIR filter. Traditionally, IIR filters are based on a transformation of an analog-filter approximation to a digital filter. We followed tradition, using a bilinear transformation. Bilinear transformation is primarily useful for designing frequency-selective filters where response consists of flat pass-bands and stop-bands. The pass-band and stop-band cutoff frequencies of the analog filter must be prewarped, so that the resulting digital filter meets its specifications. Because bilinear transformation maps the entire $j\omega$-axis of the s-plane onto the unit circle, the equiripple amplitude response of an elliptic filter is preserved. Therefore, optimal-magnitude response can be obtained for IIR filters by using the bilinear transformation of analog elliptic filters.

Our design required these specifications:

Elliptic Filter
Sample input signal at 20 kHz
$\delta_1 = 10\delta - 0.005$
$20*\mathrm{Log}_{10}(\delta_2) = -15$ dB
$\omega_p = 0.2*\mathrm{PI}$ (2 kHz) (pass-band frequency)
$\omega_s = 0.3*\mathrm{PI}$ (3 kHz) (stop-band frequency)
$\omega_c = 0.25*\mathrm{PI}$ (2.5 kHz) (cut off frequency)

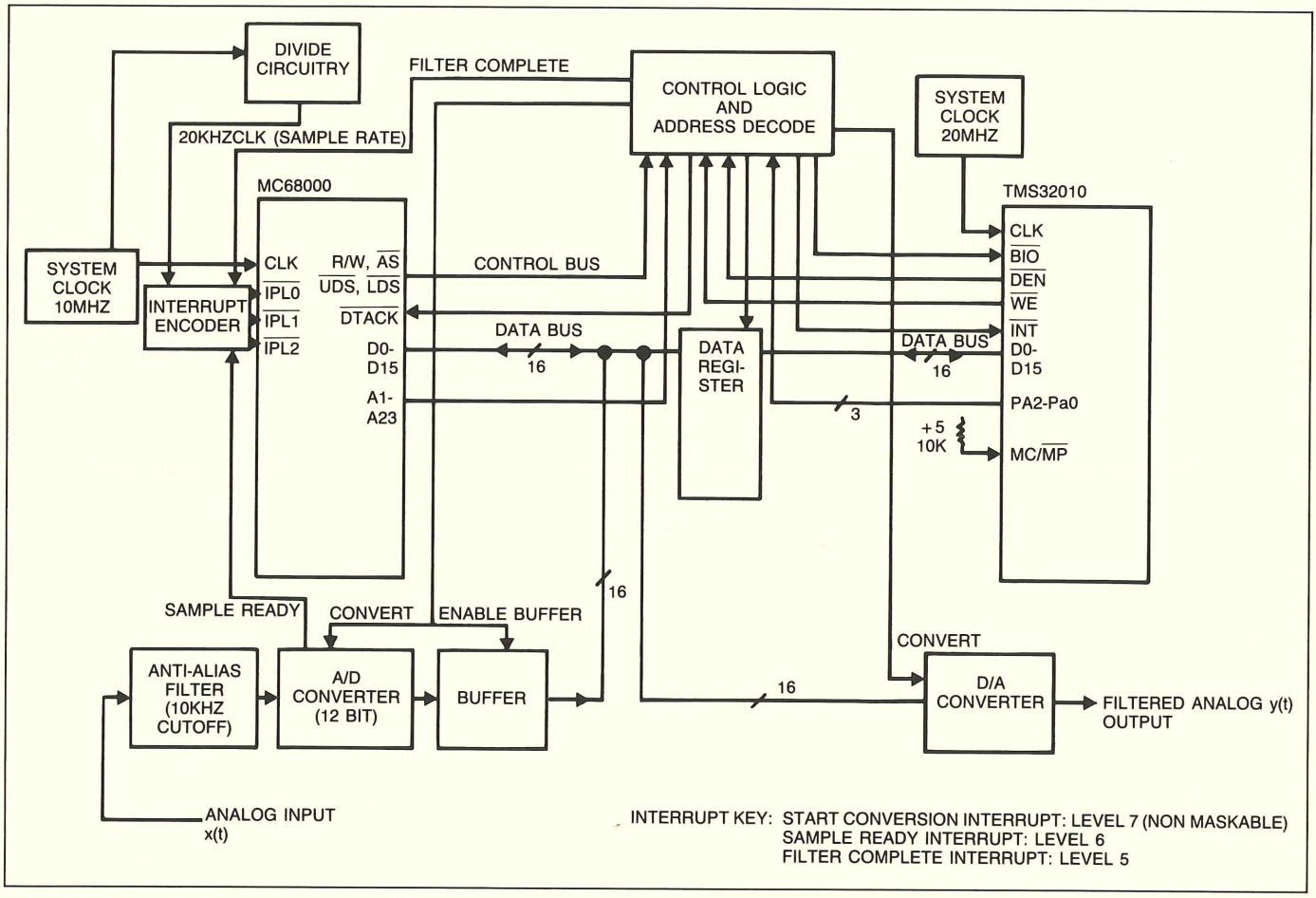


Figure 4. The hardware used to build the digital filter. See figures 5 and 6 for program flows.

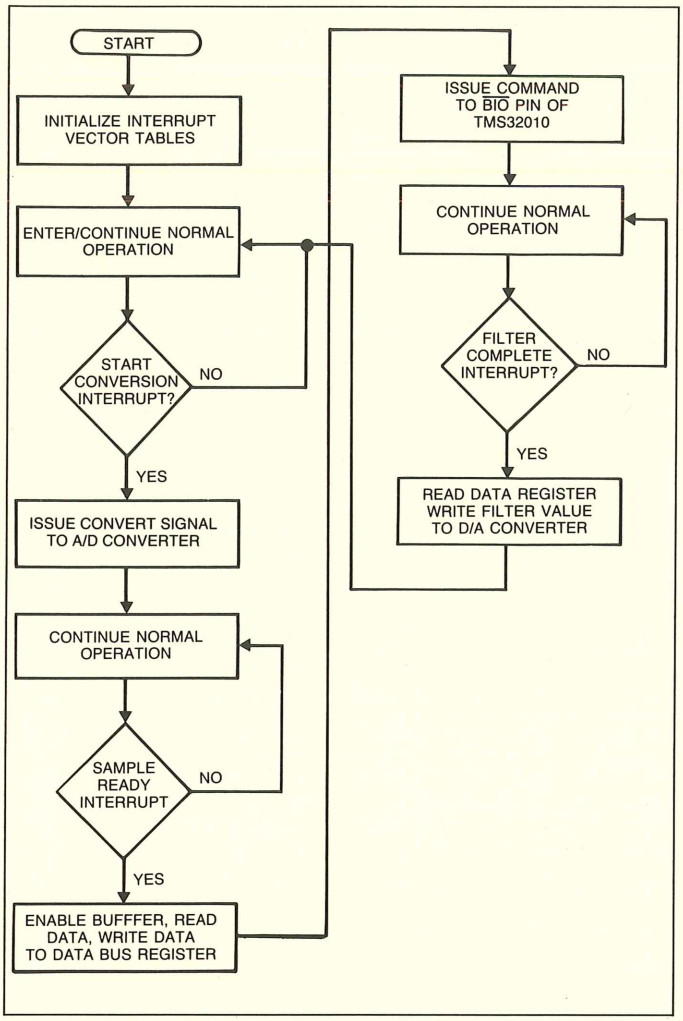


Figure 5. The program flow for the 68000 microprocessor.

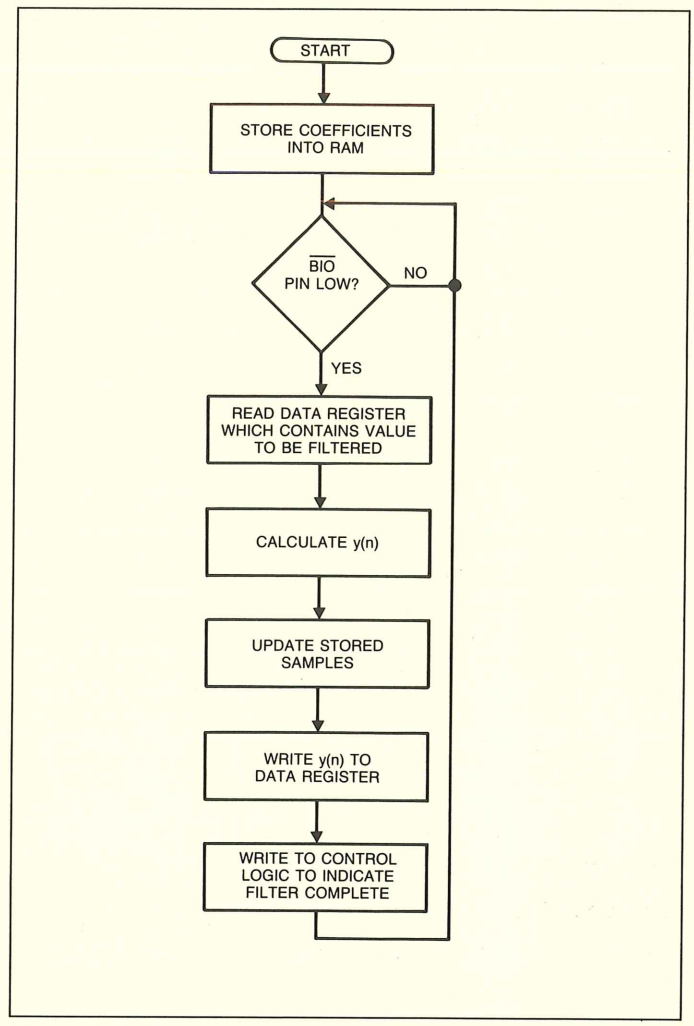


Figure 6. The program flow for the Digital Signal Processor.

We require that the magnitude response within the pass-band be constant to within 1 dB. Within the stop-band, attenuation must be more than 15 dB from 3 kHz to 10 kHz.

If we fix δ_1, ω_p, and ω_s, we end up with N = 3, and $20*\text{Log}_{10}(\delta_2) = -26.7$ dB. This exceeds our specifications.

Prewarping the critical frequencies so that $\Omega_p = 2*\tan(0.2*\text{PI}/2)$ and $\Omega_s = 2*\tan(0.3*\text{PI}/2)$, we obtain the system function:

$$H(s) = \frac{0.12460(s^2 + 1.3040)}{(0.6498s + 0.2448)(s^2 + 0.2521s + 0.4313)}$$

Using the bilinear transformation:

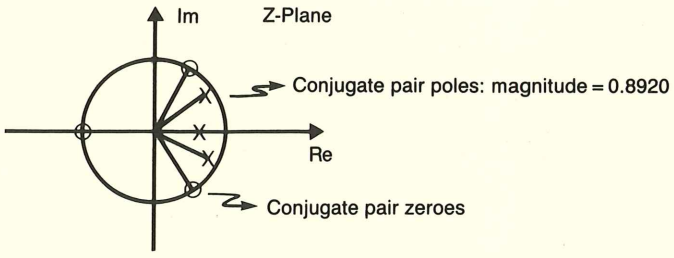
$$s = \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}} \quad \text{(with } T = 1\text{)}$$

We arrive at the system function in the frequency domain:

$$H(z) = \frac{0.05634(1 + z^{-1})(1 - 1.0166z^{-1} + z^{-2})}{(1 - 0.6830z^{-1})(1 - 1.4461z^{-1} + 0.7957z^{-2})}$$

Analysis of pole-zero locations:

zero at $z = -1$
zeroes at $z = 0.5083 \pm 0.8611j$
pole at $z = 0.6830$
poles at $z = 0.72305 \pm 0.5224j$



See figure 7 for the magnitude, gain, and phase plots of this third-order elliptic filter transformed by bilinear transformation.

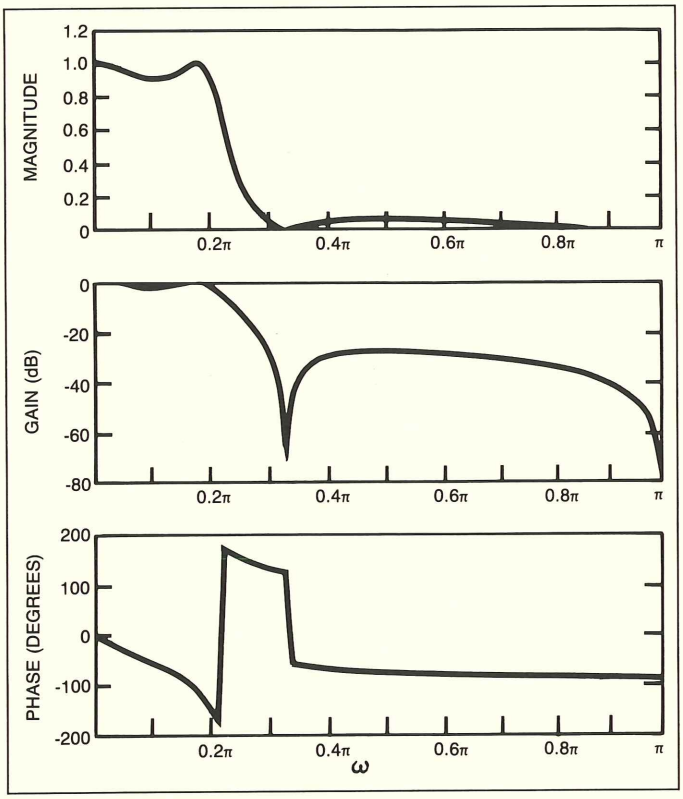


Figure 7. The frequency response of a third-order elliptic filter transformed by bilinear transformation.

Difference Equation Derivation

The following is the difference equation implemented with the TMS32010:

$$\text{System response} = H(z) = \frac{X(z)}{Y(z)}$$

$$H(z) = \frac{0.05634(1+z^{-1})(1-1.0166z^{-1}+z^{-2})}{(1-0.6830z^{-1})(1-1.4461z^{-1}+0.7957z^{-2})}$$

Solving for the difference equation we have:

$$y(n) = 2.129y(n-1) - 1.7834y(n-2) + 0.5435y(n-3) + 0.05634x(n) - 0.00094x(n-1) - 0.00094x(n-2) + 0.05634x(n-3)$$

General form of the above equation:

$$y(n) = a0x(n) + a1x(n-1) + a2x(n-2) + a3x(n-3) + b1y(n-1) + b2y(n-2) + b3y(n-3)$$

Table 3 indicates the normalized coefficients used in the implementation of this filter.

Figure 8 indicates the memory mapping of the coefficients and the data for the filter.

TABLE 3 — FILTER COEFFICIENTS

Coefficient	Normalized Decimal Value	Signed Hex Value
a0	0.5634	49D7
a1	0.9400	7BFF
a2	-0.9400	FBFF
a3	0.5634	49D7
b1	0.2129	1BFF
b2	-0.1783	97FF
b3	0.5435	47FF

Coefficients are in twos' complement binary form.

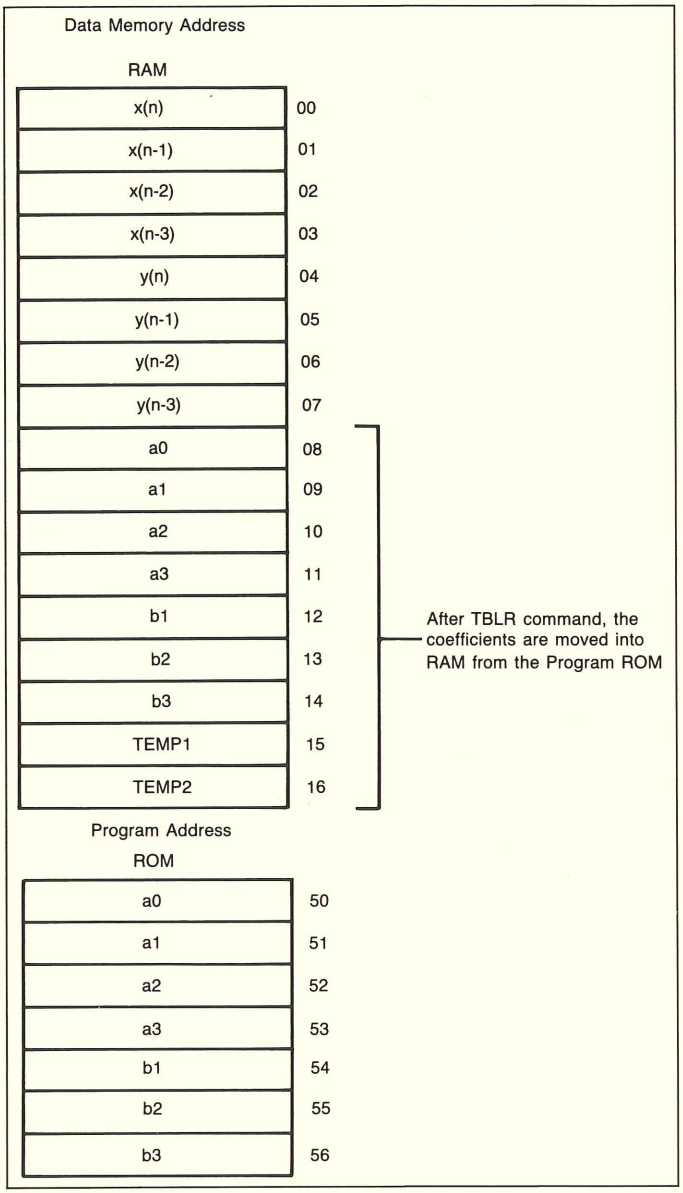
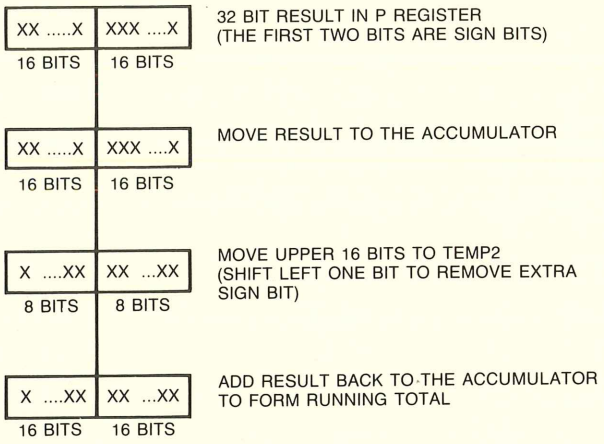


Figure 8. Memory map of the TMS32010 Digital Signal Processor.

Program Listing Note

To multiply correctly with two 15-bit fractional numbers, the result must be shifted to the left by one bit. This shifting is commented on in the program listing (see "Comments" following the listing). This program note shows how we used the 32-bit multiplication result to implement the filter correctly.

This sequence of shifting the multiplication result to the left by one was necessary every time a multiplication was done in the program.



Program listing for the TMS32010

Location		Instruction	Comments
000		DINT	(disable interrupts)
001		LACK 50	(load accumulator with address of first coefficient)
002		LARK AR0,7	(load auxiliary register with loop counter value)
003		LARK AR1,8	(load address of first coefficient into AR1)
004	RCONST	LARP 1	(point AR1)
005		TBLR * +,AR0	(load coefficient into data memory, increment AR1, point to AR0)
008		ADD 1	(increment accumulator to next coefficient address)
009		BANZ RCONST	(decrement AR0 by one, check if zero)
011		ZAC	(zero the accumulator)
012		SACH 00,0	(store 00 into RAM location zero, prepare to clear scratch pad RAM)
013		LARK AR0,8	(load auxiliary register with loop counter value)
014		LARK AR1,0	(load auxiliary register with beginning RAM location)
015	RCLEAR	LARP 1	(point to AR1)
016		DMOV * +, AR0	(move zeroes into next RAM location, point to AR0)
016		BANZ RCLEAR	(decrement AR0 by one, check if zero)
017			
018	START	BIOZ 100	(branch to 100 when BIO∼ pin is low)
020		B START	(wait for BIO∼ = 0)
100		IN 00,PA0	(BIO∼=0, read x(n) value from port address 0)
102		LT x(n)	(load x(n) into the T register)
103		ZAC	(zero the accumulator)
104		MPY a0	(multiply x(n) by a0, store result in P register)
105		PAC	(load accumulator with result)
106		SACH TEMP1,1	(shift result to the left one bit, store in TEMP1)
107		ADDH TEMP1	(move shifted result into higher order byte of accumulator)
108		LT x(n-1)	(load x(n-1) into T register)
109		MPY a1	(multiply x(n-1) by a1, store in P register)
110		SACH TEMP1,0	(move running total to TEMP1)
111		PAC	(load accumulator with second result)
112		SACH TEMP2,1	(shift result left by one, store in TEMP2)
113		ZAC	(clear the accumulator)
114		ADDH TEMP1	(move running total into accumulator)

```
115     ADDH TEMP2          (add second result)
116     LT x(n-2)           (load x(n-2) into T register)
117     MPY a2              (multiply x(n-2) by a2, store in P register)
118     SACH TEMP1,0        (move running total to TEMP1)
119     PAC                 (load accumulator with third result)
120     SACH TEMP2,1        (shift result left by one, store in TEMP2)
121     ZAC                 (clear the accumulator)
122     ADDH TEMP1          (move running total into accumulator)
123     ADDH TEMP2          (add third result)
124     LT x(n-3)           (load x(n-3) into T register)
125     MPY a3              (multiply x(n-3) by a3, store in P register)
126     SACH TEMP1,0        (move running total to TEMP1)
127     PAC                 (load accumulator with fourth result)
128     SACH TEMP2,1        (shift result left by one, store in TEMP2)
129     ZAC                 (clear the accumulator)
130     ADDH TEMP1          (move running total into accumulator)
131     ADDH TEMP2          (add fourth result)
132     LT y(n-1)           (load y(n-1) into T register)
133     MPY b1              (multiply y(n-1) by b1, store in P register)
134     SACH TEMP1,0        (move running total to TEMP1)
135     PAC                 (load accumulator with fifth result)
136     SACH TEMP2,1        (shift result left by one, store in TEMP2)
137     ZAC                 (zero the accumulator)
138     ADDH TEMP1          (move running total into accumulator)
139     ADDH TEMP2          (add fifth result)
140     LT y(n-2)           (load y(n-2) into T register)
141     MPY b2              (Multiply y(n-2) by b2, store in P register)
142     SACH TEMP1,0        (move running total to TEMP1)
143     PAC                 (load accumulator with sixth result)
144     SACH TEMP2,1        (shift result left by one, store in TEMP2)
145     ZAC                 (clear the accumulator)
146     ADDH TEMP1          (move running total into accumulator)
147     ADDH TEMP2          (add sixth result)
148     LT y(n-3)           (load y (n-3) into T register)
149     MPY b3              (multiply y(n-3) by b3, store in P register)
150     SACH TEMP2,1        (move running total to TEMP1)
151     PAC                 (load accumulator with last result)
152     SACH TEMP2,1        (shift result left by one, store in TEMP2)
153     ZAC                 (clear the accumulator)
154     ADDH TEMP1          (move running total into accumulator)
155     ADDH TEMP2          (add last result, we now have y(n)!!)
156     SACH y(n),0         (move the result into memory)
157     DMOV y(n-2)         (move y(n-2) into y(n-3))
158     DMOV y(n-1)         (move y(n-1) into y(n-2)
159     DMOV y(n)           (move y(n) into y(n-1))
160     DMOV x(n-2)         (move x(n-2) into x(n-3))
161     DMOV x(n-1)         (move x(n-1) into x(n-2))
162     DMOV x(n)           (move x(n) into x(n-1))
163     OUT y(n), PA0       (send result to Data Register, causing an interrupt to the
                             68000)
164     EINT                (enable interrupts)
165     B START             (go back and wait for next x(n))
166
```

Comments

Making the TMS32010 function as a filter was not difficult.

The filter was implemented in 87 clock cycles (17.4 microseconds), with the critical loop being 66-steps long. This calculates out to 13.2 μsec to execute the difference equation. The instruction set was complete with few exceptions; completeness made the programming easier.

The LTD command would have been helpful, except in this filter implementation the multiplication result had to be shifted to the left by one.

It should be noted that quantization error is introduced into the sample by the conversion of the sampled analog input into a 16-bit digital word. Another source of error comes from truncating the 32-bit result of multiplication to 16 bits. Truncation introduces a 0.003% error. This figure is arrived at by the following relationship:

$$-2^{-b} < \text{error} \leq 0$$

where b is the number of bits less one for the sign bit.

In this program we did not anticipate a possible overflow. If we had anticipated the need, overflow capacity could have been implemented by the BV command. This branch-on-overflow command could have been placed to jump to a routine for handling overflow conditions after the accumulator computed a running total.

Or we could have used saturation-mode operation as an alternate overflow/underflow technique. The SOVM command sets the accumulator to the highest positive/negative value. The ROVM command removes the saturation mode operation.

Both techniques reduce filter calculation accuracy.

For More Information

For more information, call Jim Fenton 629-1394 (92-789). □

ANSI/X3H4 TECHNICAL COMMITTEE (IRDS) MEETS AT TEK

Mike Meyer, Manager, CAX Data Management

The ANSI/X3H4 Technical Committee met July 23–26 in Beaverton. Tek hosted this bimonthly meeting. The next meeting will be held October 1–4 in Washington, DC, hosted by the Mitre Corporation.

ANSI/X3H4 Meeting Schedule

Meeting Dates	Location	Sponsor
1–4 Oct 84(M-W)	Washington, DC	Mitre
15–17 Nov 84(W-F)	Boston, MA	MSP
21–23 Jan 85(M-W)	Denver, CO	Martin Marietta
18–20 Mar 85(M-W)	Washington, DC	CBEMA
20–22 May 85(M-W)	Dallas, TX	TBA
22–24 Jul 85(M-W)	Boston, MA	TBA

The significant working-group activities at the July meeting follows:

Generalized Database Support – The Optional Module for Generalized Database Support is nearing completion. It will be presented to the full committee at the October meeting. It is available to Tektronix engineers through electronic or interplant mail.

N-ary versus Binary ER Model – Again, the N-ary versus Binary Model for the dictionary schema was a major topic. The results of the questionaire indicated vendors perceived the N-ary as easy to implement, but users felt it would be hard to use.

This working group will now create a prototype of a N-ary Model to show how it can be used in the IRDS.

Programming Language Support – This working group has expanded its scope from just COBOL to include PL/I, Fortran, and Pascal. The group accomplished much at the meeting: their Recommended Support of Programming Languages will be an optional model in the IRDS.

Level 0 Standard – The committee continues to finish up Base Document/dpANS. Although they made several refinements to the Base Document (Core)/dpANS, a letter ballot within the committee before 1985 is unlikely.

The work reported above should not significantly affect Tektronix. Call Mike Meyer, 627-2628 (50-560) for more information. □

A QUICK LOOK AT CAD IN THE SOVIET UNION

Dave Straayer is a senior engineer in IDG Systems Engineering. Dave joined Tek in 1977. He worked earlier at Texas Instruments. He has a BS and an MS in math from Michigan Tech.

The reactions to the power failure in the machine room in Moscow were familiar. The Soviet engineers had that same look of disgust tempered with humor that I've seen more than once in Wilsonville. Dr. Klimow, head of the CAD laboratory at the Moscow Power Institute, didn't comment but, obviously, he couldn't demo their PDP-11/40s. The Institute is a large electrical engineering college with many Soviet and third-world students.

Although my visit wasn't official, I was in Moscow as a direct result of my being Tek's representative on ANSI's Computer Graphics Committee (X3H3) and of ANSI's work with its international standards-setting partner ISO. I've been active in standards-setting activities since 1978. (ANSI is the American National Standards Institute; ISO is the International Standards Organization. ANSI is a member of ISO.)

In March, Dr. V. Klimow of the Moscow Power Institute wrote me asking about the computer graphics standards we were developing. He wrote a similar letter to Janet Chin of ANSI. Could we tell him more about GKS, PMIG, PHIGS, and IGES.

GKS, or Graphical Kernel System, is a standard software interface for computer graphics. ISO is adopting GKS as an international standard. ANSI is making it an American National Standard. In his letter, Dr. Klimow suggested that the Moscow Power Institute might implement GKS.

PMIG is an earlier proposal for a small-scale software interface to graphics. It is now part of the ANSI proposal for GKS.

PHIGS is a sort of "super GKS" that we in X3H3 are developing.

IGES is a standard for formatting design files in computer-aided-design systems. IGES facilitates the movement of design information between CAD systems.

The requests worried me. What about the controls on technology export to the Eastern Bloc? Could I respond freely? I was also intrigued. What sort of computer graphics equipment did the Soviets have? Was there embargoed equipment there?

I considered the requests for a few weeks and then called Martha Prinson, who is part of the secretariat of ANSI X3. She confirmed my understanding that all the documents requested by Dr. Klimow are public, available to anyone who requests them. This openness is essential to the process of developing standards by consensus; openness and consensus shield participants like myself from legal liability for the impact of the standards we develop. (For more on the standards-setting process, see "Electro-Political Engineering" by Maris Graube in *Technology Report,* July 1984.)

I also talked to Andrew Davis, Tek's PLOT-10 marketing manager. PLOT-10 software products include an implementation of GKS. I was curious about whether it would be possible – or desirable – for Tek to try to sell PLOT 10 to Dr. Klimow. Andy Drew said this would not be permitted by U.S. technology-export regulations.

Armed with this information, I wrote to Dr. Klimow, telling him how to get the documents. As a courtesy, I also sent a copy of each document.

Idea for a Visit

Since I was going to France to attend a standards meeting as a part of my job, this was a chance for my wife, Jo Shapland, and I to see Europe. I began to plan an extensive vacation following the business meeting (June 13–20). Could we also visit Dr. Klimow and his CAD laboratory?

In Early May, I wrote Dr. Klimow suggesting a visit to the Institute. I applied for visas to enter the Soviet Union. Since Dr. Klimow didn't respond, I wrote again saying I would be in Moscow July 2 through 4. Would he and his colleagues like a presentation on GKS? No response.

Our visas were not confirmed until June 30. Still no response to my offer. Jo and I left Helsinki by train for Moscow July 1. The next morning, from the Hotel Intourist, the marginally cooperative hotel staff finally got my phone call placed to Dr. Klimow. I got the feeling that he was waiting for my call. It was a school holiday, yet he was in his office. He struggled to converse in English, offering to pick us up and show us Moscow. He proved to be a willing host.

Meeting Dr. Klimow

He drove up at noon and smoothly handled the doorman's request for registration cards, getting through that barrier to meet us in the lobby. Between the monuments and vistas, he and I talked computer graphics and computer-graphics standards. He was delighted that I had 35mm slides for a presentation for a colloquium the next day. That night he took us to a Caucasian restaurant for dinner – excellent, not what I expected. Wine and brandy were plentiful, but our host refrained. "Severe laws about drinking and driving here," he said.

The next morning, Jo surprised Dr. Klimow by taking a bus tour rather than visiting the Institute. She could see computers anytime. We got to his office at 10 am. By this time Dr. Klimow and I would have been on a first-name basis – except I couldn't say Вячеслав.

The building was run-down. The elevator to his top-floor lab was rickety and cramped. Even though it was a holiday, what I took to be graduate students were around in the same proportions as I would expect in a EE school in the US. There was evidence that they had planned to demonstrate their CAD systems for me. Then the power in the lab failed. Fear of file and disk damage seemed to charge the atmosphere much as it had in the states when the dated PDP-11s in the Institute's lab were commonplace in university CAD. Then the whole building lost power and we sat in the natural light pouring in from large windows.

It looked as if the demonstration and my 35-mm GKS show, as well as the computer files, were in hazard. But full power was restored within several hours and they were able to demonstrate their CAD capability. In the interim, partial power allowed me to give my slide show on developing graphics and GKS.

The Machine Room and a Glimpse of Soviet CAD

At this point I should describe the machine room a little. All their graphics display hardware was directed-beam refresh technology. No DVSTs or raster displays to be seen. The machines were DEC PDP-11/40s, running RT and RSX. Klimow mentioned interest in converting to UNIX later this year. I did not mention that UNIX normally requires at least 11/44- or 11/70-sized machines; I'm not sure it would even run on his machines.

There were the usual raised floors of any machine room. Air conditioning was provided by 18 units stuck in high windows! There were two or three digitizers in the room including one gantry-style machine. Except for the air conditioners, all this was typical in an American CAD lab in the early 70s.

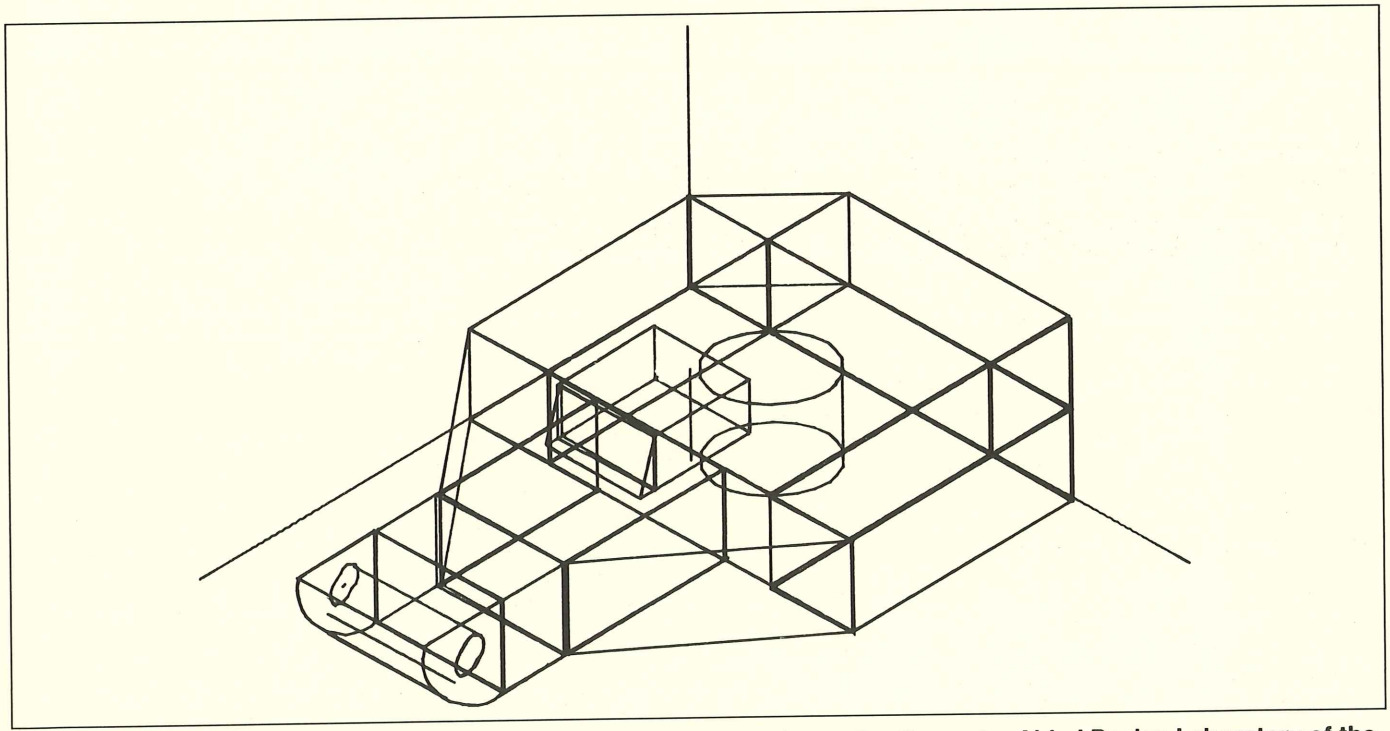
The one touch of the 80s was a recently acquired display with a LSI-based PDP 11/23 in it. Partially disassembled, it appeared to be a basket case donated from somewhere.

No western brands were visible on any of the displays. I didn't recognize the origin of any equipment other than the DEC minicomputers. The two drum plotters in the machine room used late 60s technology.

Several hours later, when the power was restored, I watched his system in action. Its display quality was appalling – fuzzy lines and line mismatches at corners – and line positioning was poor. Displayed text and keyboards were all in the Cyrillic alphabet.

One of the graduate students (I think she was a graduate student) demonstrated mechanical design with three-dimensional wire frame displays of constructive solid geometry. There was no evidence of circuit or IC design anywhere in the lab. As I expected, the software was inhibited by the restricted memory (only 32K words in that class of PDP 11).

Klimow suggested that he would get 11/70-class machines later this year from sources within the Soviet Union. This matched his stated intention to convert to UNIX, as UNIX is a popular operating system on that class of machine. He was always ambiguous about whether he was talking about DEC or IBM machines, or their USSR-built software equivalents, but I saw no evidence of USSR-built clones, only DEC-original machines.



This wire-frame plot was made during a recent informal demonstration at the Computer Aided Design Laboratory of the Moscow Power Institute. A two-color drum plotter was used. The quality and imperfect detail reflect the obsolescence of much of the 60s-vintage equipment available in this engineering school. The system's CRT displays employed directed-beam technology. The host was a PDP 11/40. Other, less public, locations probably use more modern raster-beam displays, and more powerful hosts – even though both are embargoed for export to the USSR.

Klimow grumbled that the promised new machines would not solve the address-memory limitations restricting his current projects. Memory-addressing limitations make full-range graphics difficult on classical 16-bit minicomputers. Only 32-bit machines are appropriate for modern CAD graphics, the VAX, IBM 43xx, and Prime for example. I believe the US embargoes all such machines. I think Dr. Klimow is acutely aware of how his work is being restricted by our embargo.

Dr. Klimow told me he had approval to spend $20,000 for new, raster-based hardware. He expressed frustration that everything he wanted was embargoed. He suggested, indirectly, that it is possible to acquire such equipment, but the presence of international students keep him from taking advantage of any such purchases for his lab.

Dr. Klimow also told me of a possible purchase of a West-German GKS package for his lab. He was in contact with Jose Encarnacao, chairman of the Eurographics association (FRG), about this purchase.

My Demonstration and a Visit to a Well-Stocked Bookstore

While waiting for computer room power, partial power enabled me to make my presentation to Klimow's colleagues and students. It went smoothly. I told them what GKS would mean as a standard. Klimow translated. Questions were knowledgeable. We didn't discuss political topics.

After the presentations, we picked up Jo at the hotel, and went, at our request, to a large, popular bookstore. We saw many texts on microprocessor design, UNIX, ADA, Pascal, and so forth. The embargo on technical information doesn't seem to limit the quality and quantity of technical books, judging from seeing just one book shop.

Dr. Klimow and his wife took us to dinner that night and the next (the Fourth of July). We talked revolution – theirs and ours – with appropriate toasts. Again, he was a good host. He refused our offer to share the check. Although he consistently abstained, he provided wine, brandy and vodka.

We took the famous Moscow Subway on our last night in Moscow. At our destination, Dr. Klimow told us the subway station was the Communist Party Headquarters' bomb shelter during the Second World War. It was very deep and well decorated. I noticed arcs scratched on the floor. Were they made by the movement of blast doors? As I was pointing them out to Jo, Dr. Klimow acknowledged, "they are nuclear-blast doors."

Conclusions

Soviet computer development depends heavily on western machines and literature. They are 10–15 years behind us. I base these conclusions on a few days in Moscow and short look at a laboratory that is too exposed to foreign students to use illicitly acquired goods. By Dr. Klimow's admission, such goods are available and used elsewhere in the USSR. By his inference, those uses are, at least in part, military.

Klimow at one point discussed military spending. "The USSR is very poor," he said. Since the USSR lacks computer technology, this makes what it has to spend on arms particularly ineffective in our modern world. This was confusing to me. I would have expected him to argue strongly against technical embargo and deny or minimize its impact on military uses.

In contrast to American sensitivity to the potential for improving productivity through computer-aided design, Dr. Klimow seemed disinterested. But perhaps I am overly sensitive to this, given my participation in the American industry. □

TEK'S STAKE IN THE STANDARDS NOW DEVELOPING FOR GRAPHICS

Dave Straayer, IDG Systems Engineering

A major part of my job at Tek is invested in helping develop and set new national and international standards. These standards are increasingly important to Tek and our computer graphics product lines. Let's look at these developing standards and what they mean to us.

GKS – Graphical Kernel System

GKS is a standardized subroutine package for graphics. It's a software standard rather than a hardware standard. It specifies the functions needed to do 2-dimensional graphics; it also standardizes the names for these functions in Fortran, C, Pascal, Basic, and Ada. Tek's Plot-10 GKS is an implementation of this standard. GKS is available on DEC-20 computers in Wilsonville, and plans are being laid to make it available on the Cyber systems.

GKS is in the final stages of approval in both ISO, the International Standards Organization, and ANSI, the American National Standards Institute.

Engineers frequently ask me how does a software standard like GKS relate to hardware systems like Tek's 4100 line of terminals? The answer is not simple. 4100 Series terminals can be thought of as a firmware implementation of SIGGRAPH's moribund *Core* proposal – GKS standardizes software. How and when GKS will migrate into firmware is being studied today in Wilsonville, but competitors like Chromatics, Sigmex, and Spectragraphics have already put GKS functionality in hardware.

GKS is a fact, although it will not adopted finally until December or January. No significant technical changes will be made, and the marketplace has already widely accepted GKS.

Core

The Core graphics system was proposed ACM-Siggraph in 1977 and 1979. Although numerous products have been based on Core (including Tek's Plot-10 IGL and 4100 terminal firmware), Core will not become a formal standard. Neither ANSI nor ISO are considering it.

PHIGS

ANSI subgroup X3H31 is working on a proposal called the Programmer's Hierarchical Interactive Graphics System. (Four years ago, this group was working on adopting Core as an American National Standard.) The PHIGS proposal now looks a lot more like GKS than Core. It adds functions, namely segment editing and segment subroutining, that our marketplace clearly needs. Tek, in response to these needs, recently extended the 4100 product line with *Dragon,* an enhancement that adds segment editing and segment subroutines to 4115 software.

The big problem with the PHIGS proposal is that it is not compatible with the soon-to-be-adopted GKS standard – although it could be. Unfortunately, the subgroup working on PHIGS is not willing to change their work to the degree necessary.

The PHIGS schedule plans final adoption in 1987.

GKS Level 3 Proposal

Tek wrote this proposal anticipating some needs in ECS. Its very much like PHIGS; yet it is one-hundred percent compatible with GKS. It provides the same types of functions, namely segment editing and segment subroutines. This proposal has been circulated in ANSI and ISO, but as yet it hasn't formal status. Several Europeans on the ISO committee have suggested that they would like to see the proposal become the basis of an ISO standard.

Virtual Device Interface (VDI)

Because VDI is intended to be a standard for graphics terminals, VDI is more interesting to Tek than GKS. Unfortunately, the schedule for VDI has not met the expectations of those most impatient for such a hardware standard. Also, there is some confusion over exactly what a Virtual Device Interface standard would be: some think of it as a standard graphics terminal, some think of it as a standard graphics chip, others think of it as a standard device driver in software.

The current draft of VDI does not include segmentation, but ANSC X3H3 voted overwhelmingly that the next draft should include segments. VDI is very GKS-like in that it has the same primitives and attributes.

The current VDI schedule plans final adoption in 1986.

Virtual Device Metafile (VDM)

VDM is a proposal for standard picture files. It's very GKS-like – its probably fair to consider VDM as an interpretation of GKS in a file-format context. VDM has had a public comment period in ANSI, and will have another public comment period this fall. VDM is a subject of ISO work too. ANSI will probably adopt this standard in mid to late 1985, ISO will do so somewhat earlier.

Character Coding for GKS

GKS is supposed to be a software standard, but ANSC X3L2, the folks who brought us ASCII, are working on a coding for GKS. Despite the fact that this coding directly conflicts with some VDI goals, it is probably not a bad idea, especially to a company like Tek that is likely to be putting GKS in a terminal. In fact, most of the work on the proposal is being done by Tek's X3L2 rep, Jim Maynard.

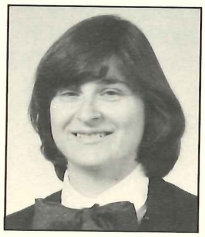
Videotex Graphics: NAPLPS

NAPLPS is a joint ANSI/Canadian standard for videotex; it includes graphics. AT&T supports NAPLPS as a good standard for general purpose graphics terminals. The graphics committee, X3H3, disagrees. NAPLPA does not support segmentation or graphics input. Videotex always seems like an idea whose time is almost here. So far, this is pretty much a watch-and-wait thing for Tek.

CAD/CAM Database Exchange: IGES

IGES is a standard which allows users of one CAD/CAM system, say Computervision, to exchange design data with users of another CAD/CAM system, say CALMA. It affects a few Tek products like TekniCAD, which we offer as a an option to allow the reading and writing of IGES files. The benefit of IGES to Tek would be mostly internal. Its use in CAD systems would allow different systems doing different work to exchange design data. □

MULTILEVEL SIMULATION OF DIGITAL SYSTEMS USING THE TEKTRONIX SIMULATION SYSTEM



Ellen Mickanin is an engineering manager in Logic Design Systems (LDS), part of the Design Automation Group. Ellen joined Tek in 1976. She has a BA in physics from Reed College and an MS in physics from Cornell University.

The Tektronix Simulation System is a state-of-the-art multilevel simulation system. Designed by the LDS Simulation group, the system executes application software on *simulated* digital hardware. This hardware ranges from TTL gates to advanced processors and peripheral chips. The simulation description follows a hierarchy from the functional or register transfer level to the gate level.

The Tektronix Simulation System enables users to easily generate code for virtually any processor device, from advanced microprocessors to microcoded bit-slices. A retargetable assembler and linker allows independent development and evaluation of the software on the simulated hardware. The Tektronix Simulator system runs on DEC VAX computers under the UNIX operating systems (BSD 4.1 or 4.2).

Traditionally, digital systems are evaluated by running appropriate software on prototype hardware. Based upon observed performance, the design and prototype are modified and the software rerun. The cumbersome process is repeated until the design is completed. This approach is shown in figure 1.

We, the LDS Simulation group, developed the Tektronix Simulation System to provide ways to evaluate both the design and the interactions between the hardware and the software before prototyping (see figure 2). Using the simulator, digital systems need be designed to only the detail necessary for their validation.

The simulator enables designers to refine system architecture and hardware efficiently, step by step. The subtasks may be computer modeled first at a high functional level. Then, as the architectural details are evaluated, the subtask models can be refined until the system works with great precision. With the Tektronix Simulation System, the designer works at the level most appropriate to the design phase.

Using the simulator allows software to be designed and integrated with the hardware early. Firmware and microcode development and debugging proceed concurrently with the hardware design. The first hardware prototype is more likely to be correct, as its design is already validated. Fewer prototypes will be needed, speeding product availability and reducing development costs.

The Tektronix Simulation System offers these major features:

- Digital hardware modules can be specified at a high level or at the gate level. Thus, alternative approaches can be evaluated and the design refined to greater and greater levels of complexity in a structured manner.

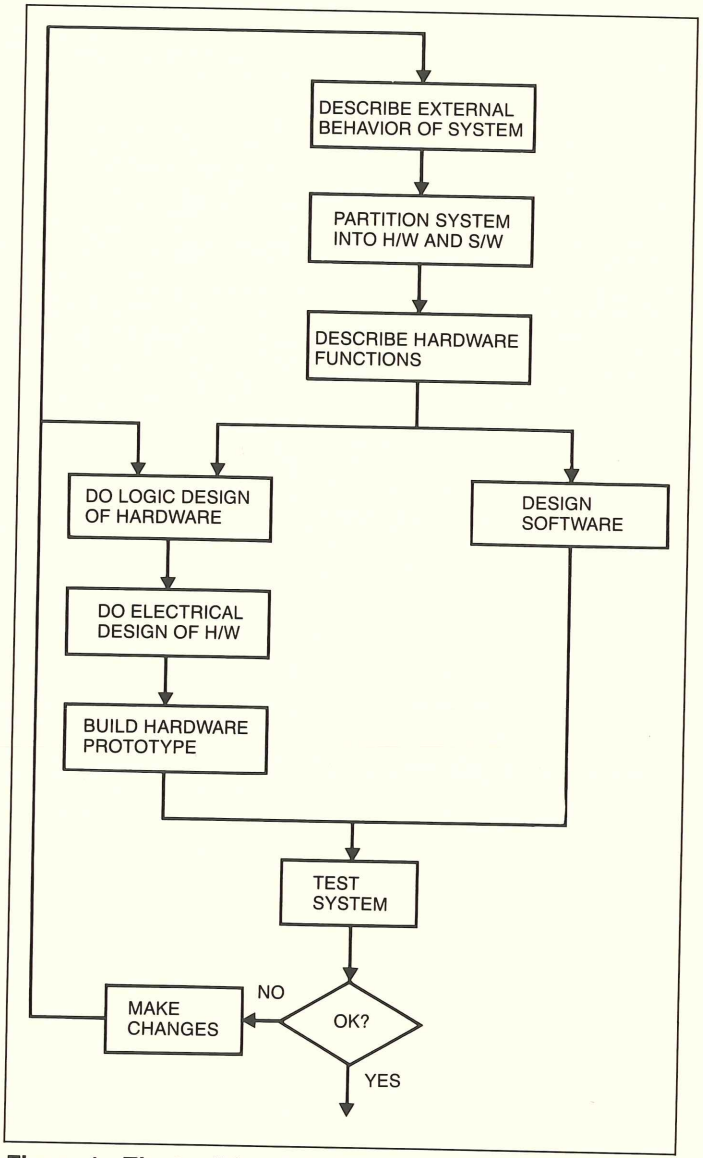


Figure 1. The traditional design process for digital systems is inefficient. The process requires a hardware prototype to be built before hardware and software can be tested as system.

- Interconnection of hardware modules is simple. This simplicity facilitates designing applications ranging from multiprocessor systems to complex VLSI chips.
- Software applications are supported extensively at both the assembler and high-level-language levels.
- The interactive execution environment is similar to the hardware execution environment of conventional emulators and logic analyzers.

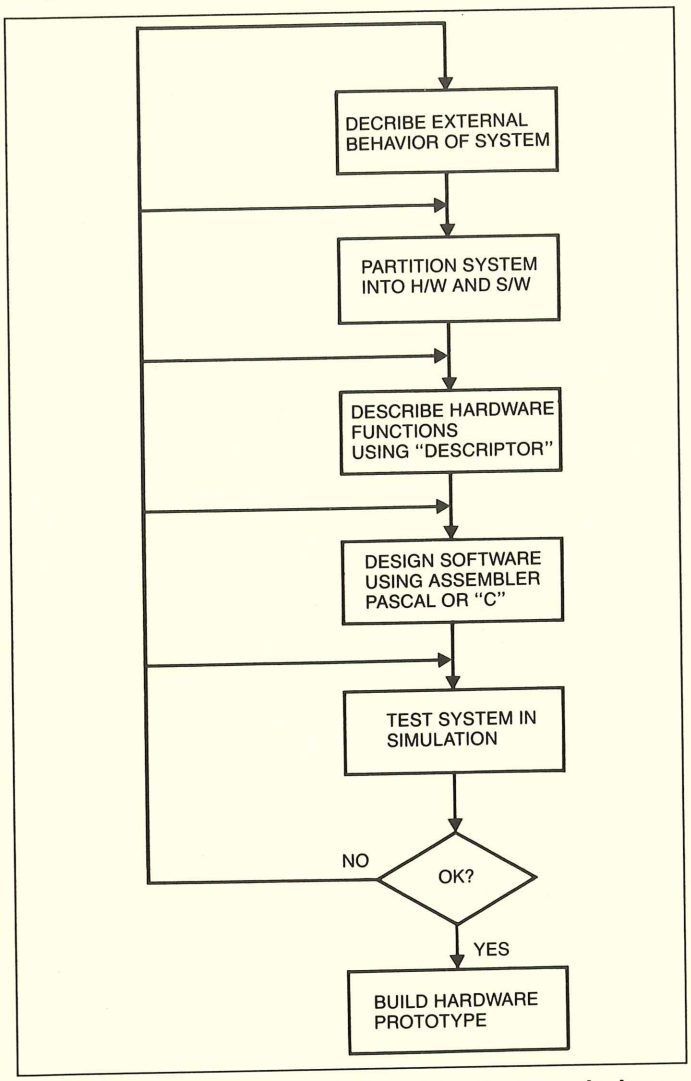


Figure 2. Using the Tektronix Simulator System, designs can be tested before prototyping. This is much more efficient.

The Simulation System

Hardware operation at the functional level is described by predefined library models or by a high-level language (the "Descriptor"). Gate-level descriptions and the interconnections between hardware modules are described by the "Connector" component. Tektronix assemblers and Pascal and C compilers can be used to generate software for the hardware models, thus providing a consistent user interface for all applications. In addition, a retargettable assembler (the "Adaptor") provides software for custom chips. The execution environment "Simulator" executes the software application program and provides debugging facilities like those found in typical software and hardware integration tools. These include breakpoints, program-flow control, internal-state examination and modification, triggering, tracing, etc. Figure 3 shows the simulator's components and their relationships.

There are five major components in the Tektronix Simulator System:

1. The *Descriptor* is a hardware description language for designing functional or register-transfer-level descriptions of digital systems. The Descriptor is a structured, high-level procedural language that can model simultaneously executing tasks, detailed operational timing, and electrical attributes.
2. The *Connector* (CON) for designing gate-level descriptions of digital circuits in a structured, hierarchical manner. Connector also connects individual components into more complex circuitry for simulation.
3. The *Runtime Simulator* (SIM) for interactive control of running simulations. Runtime's debugging capabilities are like most hardware and software debugging environments, such as emulators or logic analyzers:

   Software description languages – Programs written in any Tektronix assembler for microcroprocessors may be directly used in a simulation. Such assemblers exist for most microprocessors.

   Tektronix Pascal and C compilers are available for several 16-bit microprocessors. Programs in either Tek Pascal or C may be used directly in simulations.
4. The *Retargettable Assembler/Linking Loader* (adaptor) permits a custom assembler to be easily generated for virtually any processor, ranging from microprocessors to bit-slice machines with microcode widths up to 256 bits.
5. *Chip libraries* are provided for devices ranging from component gates (NAND, NOR) to complex microprocessors (68000, 8086); these libraries include TTL and bit-slice model descriptions.

Functional-Level Hardware Description

Individual hardware modules are described in the Descriptor language. This lanquage generates descriptions at functional, behavioral, or register-transfer levels. The Descriptor is a modification and extension of ISP' (Straubs [1]). ISP', in turn, is a modification and extension of ISPS (Bell and Newell [2]).

The construct *port* allows communication and coordination among other descriptor-defined hardware modules. Ports usually correspond to pins of chips. Ports may have various attributes, including word width (up to 256 bits), propagation delays, and drive attributes.

Internal registers, memories, or variables are defined as *registers*. Registers may also have attributes, such as word width (up to 256 bits), dimensions, and propagation delays.

Tasks are collections of executable structured C-like statements, calls to functions or procedures or compound statements. Tasks are executed at either specified changes in port or register values, the simulation time, or combinations of these values and times.

Multiple tasks may execute at the same simulated time. Thus, chips which simultaneously execute instructions, generate clock pulses, monitor interrupts, execute pipe-lining of data, maintain counters, timers, or data transmission or reception can be correctly modeled.

Correct operational timing is maintained by several methods. Execution of a particular task, function, or procedure may be suspended for a specified time, or until a specified condition occurs, such as a change in any defined input, output, or internal structure. Furthermore, assignments may be effective immediately or delayed by a propagation time, or may be effective concurrently with other assignments at the same simulated time.

Gate-Level Hardware Description

Hardware modules may be also described in the Connector language, generating gate-level submodels. The Connector can interconnect various predefined and user-defined-gate modules in a structured, hierarchical manner. Gates are connected via *nodes.*

Combinations of gates may be declared a *subcircuit,* and referenced later as an unique entity. In this case, only the input and output nodes of the subcircuit can be accessed by other modules. Figure 4 shows an arithmetic unit constructed in this manner.

Integration Of Hardware

Once hardware modules have been described (at the functional level in the Descriptor or at the gate level in CON), they may be interconnected. A connection file is generated; this file lists the global nodes, the individual hardware modules and their pin connections with the nodes, and – if desired – the initialization of any internal registers, such as memories. This step is another opportunity to modify the timing of any module.

Simulated Software

Any system using a programmable device (such as a microprocessor or a ROM-based state machine) must be able to specify the program for that device. This specifying is done by writing the program either at the assembly level (using standard Tektronix assemblers for common microprocessors) or at a high language level in Pascal or C (for some microprocessors).

In addition, a retargettable assembler is available for systems without an available assembler, or for microcoded systems. In this case, the instruction set for the target machine must be defined and mapping between the instruction mnemonic and the machine bit-format specified. Then the actual application code can be written for the target machine. The retargettable assembler supports horizontal as well as vertical machine architecture.

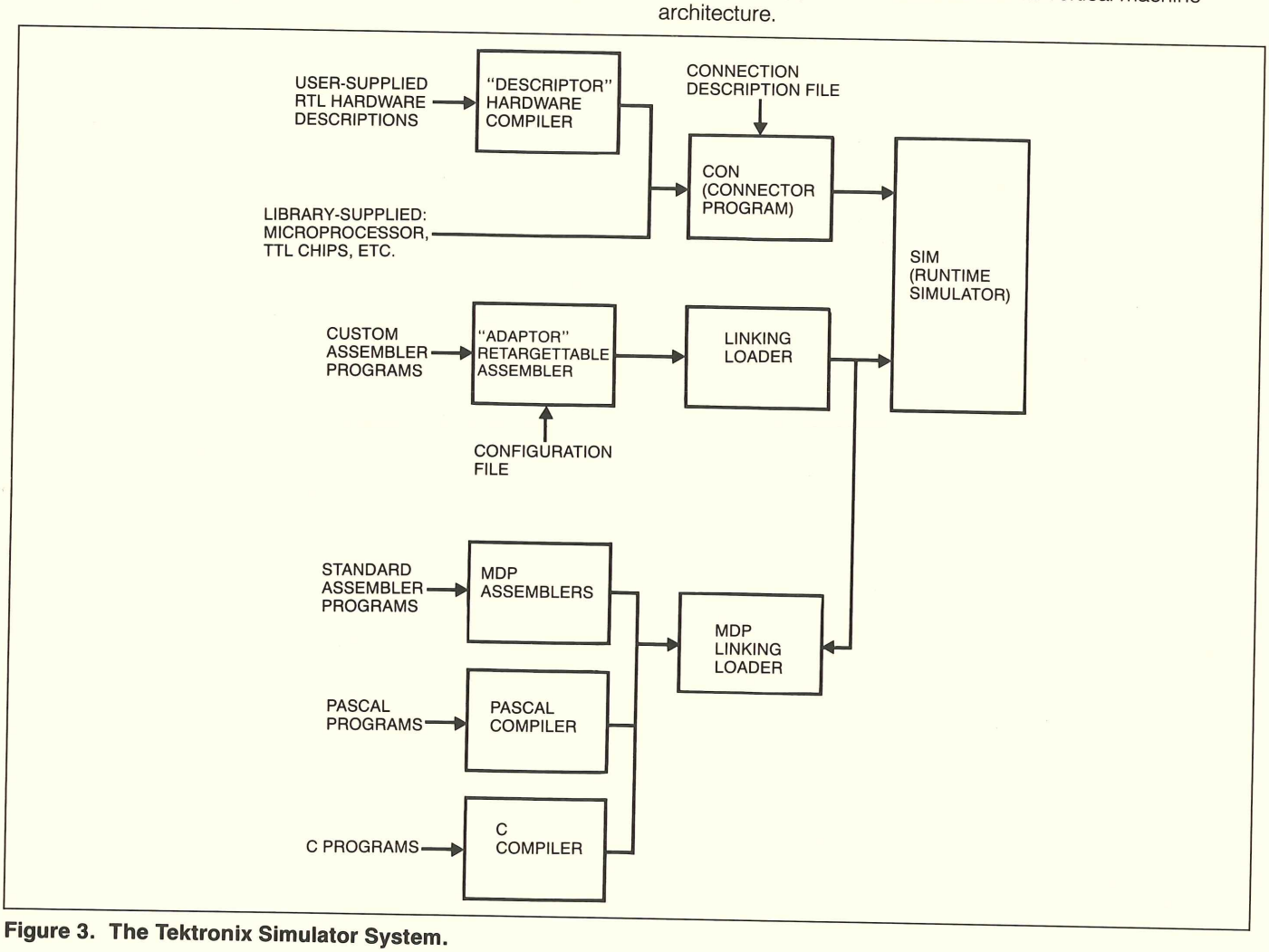


Figure 3. The Tektronix Simulator System.

```
subckt 74181(Cn,M,S(4),A'(4),B'(4),F'(4),AEQB,Cn4,P',G')
    node n1(4),n2(4),B(4)
    var i ;
    connect M' = INV(M)
    // Generate B signals from B'
    for i = 0 to 3 do
        connect B(i) = INV(B'(i))
    endfor
    // Generate internal signals n1 and n2
    for i = 0 to 3 do
        connect n2(i) = NOR( AND(A'(i),S(2),B(i)), AND(S(3),B'(i),A'(i)))
        connect n1(i) = NOR( AND(A'(i)), AND(B'(i),S(0)), AND(S(1),B(i)))
    endfor
    // Generate F outputs
    connect F'(1) = XOR( AND(INV(n1(1)),n2(1)), NAND(Cn,M'))
    connect F'(2) = XOR( AND(INV(n1(2)),n2(2)),
        NOR(AND(Cn,M',n2(0)),AND(M',n1(0))))
    connect F'(3) = XOR( AND(INV(n1(3)),n2(3)),
        NOR(AND(Cn,M',n2(0),n2(1)),AND(Cn,n1(0),n2(1)),AND(Cn,n1(1))))
    connect F'(4) = XOR( AND(INV(n1(4)),n2(4)),
        NOR(AND(Cn,M',n2(0),n2(1),n2(2)),AND(Cn,n1(0),n2(1),n2(2)),
        AND(Cn,n1(1),n2(2)),AND(Cn,n1(3))))
    // Generate AEQB output
    connect AEQB = AND(F'(0),F'(1),F'(2),F'(3))
    // Generate P', G' outputs
    connect P' = NAND(n2(0),n2(1),n2(2),n2(3))
    connect G' = NOR(AND(n1(3)),AND(n1(2),n2(3)),AND(n1(1),n2(2),n2(3)),
        AND(n1(0),n2(1),n2(2),n2(3)))
    // Generate Cn4 output
    connect Cn4 = NAND(G',NAND(Cn, n2(0), n2(1), n2(2), n2(3) )
endsub 74181;
```

Figure 4. An arithmetic unit constructed, by the Connector, as a subcircuit implemented at the gate level.

Linking loaders are provided for all methods of code generation. This allows the user to link one or more files for relocation into the address space specified. Most data segmentation schemes are supported.

Simulation Execution

Once the hardware modules have been modeled, their interconections described and the software developed, the simulation is ready to run. The simulation runtime environment is interactive. The user controls the initial state of the Tektronix Simulator System – timing parameters, flow of control, execution, and dynamic monitoring of the results. The types of commands provide the user with most of the operations available in typical hardware debugging environments, such as emulators and logic analyzers.

The user may specify combinations of triggers, breakpoints, and timers on values of any defined registers, nodes, memories, or system time. Diassembled traces are under complete user control; this includes specifying which internal structures in which chips are to be viewed. The contents of any register, signal, or memory may be modified.

A graphic representation of the changing values of the structures is also available, producing timing diagrams that present much, easily comprehended data.

The simulation can be stopped and restarted at any time, and results sent to an output file for later processing and statistical examination of data via a postprocessor.

Conclusions

The multilevel digital Tektronix Simulation System has proved invaluable in system design.

It facilitates top-down development of structured systems – using hardware appropriate to the design phase – starting with functional descriptions. As the system architecture becomes more completely defined, the functional descriptions can be replaced with more accurate gate-level descriptions.

The variety of means for describing software allows application code to be developed quickly. And when the hardware prototype is built, this code will be completely compatible.

Because the execution environment is similar to what hardware engineers are accustomed to using on prototypes, they will find the simulator straightforward and easy to learn and use.

For More Information

For more information, call Ellen Mickanin, 629-1487 (92-826). ☐

References:

1. Straubs, R.V., "A Compiler for a Register Transfer Based Simulation Language," CWRU, 1978.

2. Bell O., and A. Newell, "The PMS and ISP Descriptive System for Computer Systems," Proc. AFIPS SJCC, 1970.

TEK LIBRARIES HAVE EXTENSIVE PERIODICALS LIST

The Tektronix Libraries have recently revised their *Periodical Holdings List;* copies are available upon request. The library system has an extensive collection of sci/tech journals plus holdings in business and management. The list indicates holdings for the Corporate, Walker Road, and Wilsonville Libraries, and is especially useful when doing research.

Articles from the Libraries' holdings may be requested by Tek mail using the Photocopy Order Request Form (000-6937-00). This form is also on some UNIX systems under the name 'libreq'.

Single copies of the *Periodical Holdings List* may be requested from the Libraries by Tek mail or via UNIX.

Julianne Williams
Corporate Library 627-5388 (50-210)
UNIX: teklabs!library

Yan Soucie
Walker Road Library 629-1062 (94-501)
UNIX: tekmdp!yans

Linda Appel
Wilsonville Library 685-3986 (63-531)
UNIX: iddic!lindaa

□