Conference Abstracts

The following abstracts are from presentations made at the 1983 Rochester Forth Applications Conference, June 7-11, 1983. Many of these presentations appear as papers in the Proceedings of that conference, published by the Institute for Applied Forth Research, Inc.

C-Forth: A Portable Forth System

Peter Blaser
Nuclear Structure Research Laboratory
University of Rochester
Rochester, New York 14627
and
Lawrence P. Forsley
Laboratory for Laser Energetics
250 East River Road
Rochester, New York 14623

C-Forth is a reasonably fast, compact, transportable, and standard Forth system. Because it is written in the C programming language, it will work on any computer supporting C. This means that Forth can now be distributed quickly and easily to most computers both large and small. The system conforms to the 1983 Forth Standard and is approximately twelve pages of source code consisting of forty kernel words. These kernel words are used later by bootstrap Forth code to build the entire system. Despite the compactness and flexibility of the system, it manages to run only about three times slower than Forth written in assembler for the VAX.

Real-Time Interactive Spectroscopy

Robert Boni
and
Lawrence P. Forsley
Laboratory for Laser Energetics
University of Rochester
250 East River Road
Rochester, New York 14623

An optical multichannel analyzer (OMA) using a silicon vidicon and a DEC LSI 11 computer (available from Princeton Applied Research) have been interfaced to a ¼ meter spectrometer used in Omega coronal physics experiments. Written in Forth, its interactive commands allow comparisons of data from different shots, selection of specific features (line broadening, splitting), and removal of the OMA spectral and intensity responses. Acquisition, reduction and manipulation of 500 data channels takes at most a few seconds as compared to over an hour for the equivalent developing and digitizing of film. The systems responsiveness must be weighed against the vidicon's limited spatial resolution and dynamic range as compared with film.

The Multiple Mirror Telescope Observatory Mount Servo Control System

Marc Chamberlin
Multiple Mirror Telescope Observatory
University of Arizona
Tucson, Arizona 85721

One of the largest robots in the world today does not even remotely resemble the modern conventional idea of a robot. Rather, it is the world's third largest telescope, and sits on top of Mount Hopkins in southern Arizona. The Multiple Mirror Telescope is controlled by a computer system running a variant of Forth known as SuperForth, which, to a high degree of precision, solves such classical problems in robotics as servo control and repetitious pointing

at and tracking of various objects (in this case: stars, galaxies, etc.). This paper will examine the methodology used to make this telescope one of the most accurate instruments used by astronomers today.

Tucson Amateur Packet Radio Dynamic Addressing Protocol

Marc Chamberlin
Tucson, Arizona

Recently a group of amateur radio operators have formed an organization whose purpose was to develop a means of computer to computer communications using the radio for transmitting data. A board was designed and developed based on the 6809 microprocessor which included 24K of rom and 8K of ram, the 1933B HDLC chip, the 6551 UART, a simple modem and power supply. The author has developed an experimental network protocol based on the ideas of packet data transmission, digital repeaters, and a highly robust centralized network controller. This paper will discuss the reasons why Forth was chosen to implement this system, the internal structures of the Dynamic Addressing Protocol, and why this protocol is highly suited for working in a radio environment.

The Industrial Robot

Donald A. Davenport
Standard Oil Company of Ohio
Cleveland, OH

This paper discusses the development of robotics in both a historical and mythological sense. The industrial robot is observed to be less a revolutionary device and more of an evolutionary advancement in automation. This robot is defined in terms of its environment, and the importance of adaptability and programmability is covered. However, fully realized programmability requires a robotic language with Forth-like capabilities. eds.

Using the AIM-65 FORTH As A Macro Assembler

F. N. DiMeo
Department of Electrical Engineering
Villanova University
Villanova, Pennsylvania 19085

When developing software for industrial use, the requirement that it be memory efficient sometimes becomes paramount. This situation, when coupled with the further requirements that run time speed is important and that programming ease be maintained, leads one to carefully choose his programming tools and techniques.

Easy development implies the use of a high level language such as FORTH. Fast execution implies the judicial use of the FORTH assembler and the memory efficiency requirement indicates that the FORTH run time package may have to be eliminated.

One approach is to make use of FORTH first in its high level mode to develop the application program and then to take advantage of FORTH's ability to be a macro assembler. Using the macro assembler concepts gives one the ability to develop application programs which are easily developed and yet are totally independent of the FORTH environment when executing.

This paper examines such programming techniques using the AIM-65 FORTH system. One problem area does exist, namely that of insufficient branching range. This problem and some possible solutions are considered.

The QUAN Concept Expanded

Thomas Dowling
Miller Microcomputer Services
61 Lake Shore Road
Natick, Massachusetts 01760

The QUAN (multiple CFA words) was presented by Evan Rosen (1) at the fourth FORML Conference. This concept is very close to the "TO concept" but moves the selection of an EXECUTE routine from the execution to the compilation phase. The QUAN concept has been implemented in MMSFORTH V2.2 and extensions have been studied. The use of this type word for data hiding and implementation-independent applications will be investigated. Examples of the use of these concepts in practical situations will be presented. Using QUAN instead of VARIABLE leads to both time and memory savings. The resulting code is also easier to read with the removal of redundant @ and !. Experience with this concept in several projects proves the advisability of including it in the next FORTH standard. The history of TO concept words will also be presented.

A QUAN defining word has been developed and implementations for the Z80 and 8088 will be presented. E.g.:

QUAN-DEF QUAN @ ! NOP BUILD-QUAN 0 , ; This defines the defining word QUAN. Words defined with QUAN will return their value when used alone, set their value when preceded by IS, and return their parameter address (BODY) when preceded by AT.

QUAN X QUAN Y QUAN Z 6 IS X 7 IS Y
X Y + IS Z Z is now equal to 13.

: @EXECUTE @ 2- EXECUTE ;

QUAN-DEF VECT @EXECUTE ! NOP BUILD-QUAN ' NOP , ; This defines the defining word VECT. Words defined with VECT will execute the word whose PFA is in their parameter field when used alone, set their value when preceded by IS, and return their parameter address (BODY) when preceded by AT.

VECT PLUS ' + IS PLUS When PLUS is used it will execute +.

QUAN-DEF would not normally be used for these definitions, but they are presented to clarify the idea rather than as recommended applications.

FORTH-Based Products for Robotic Applications

Randy M. Dumse
New Micros, Inc.
2100 North Highway 360
Suite 1607
Grand Prairie, Texas 75050

The power of FORTH as a control language suitable to robotic applications is well known. Until the introduction of low cost FORTH board level products, however, users typically had to dedicate their several thousand dollar development systems to the control of each task. The application of the R65F11 FORTH-based microcomputer to a board level development/application system (for only a few hundreds of dollars) and the addition of slave processor boards that can run tasks passed from the host (for less than a hundred dollars) adds a new dimension to robotic controls. The small size of these systems makes them ideal for mobile robots. The addition of a newly developed computer readable magnetic compass working in concert with the FORTH board products can put an end to the toughest mobile robot problem - navigation.

A Robotic Application for Contamination Free Assembly

Randy M. Dumse
New Micros, Inc.
2100 North Highway 360
Suite 1607
Grand Prairie, Texas 75050

The heart of the Electro Static Gyro Navigation (ESGN) System is a one gram beryllium ball that functions as the rotor of the gyro. It is round within four millionths of an inch - about the size of a part of a human hair that was split into one thousand pieces. It is suspended in a cavity with only two thousandths of an inch clearance between the ball and the walls. The electrostatic forces that center the ball approach a million volts per inch. The presence of an detectable impurity during assembly could cause an electrostatic discharge that would destroy the gyro. Cleaning by conventional methods typically took 25 hours per good assembly, due to the high rejection rate that returned most parts to cleaning once again. This paper discusses the mechanical construction and FORTH programming of a robot arm that reduced the process to a reliable 17-minute cycle.

The paper describing this project appears in the *Journal of Forth Application and Research*, Vol. 1, No. 1, September 1983.

A Non-Conventional Implementation of Forth for the Z80 with CP/M

David C. Farden
Department of Electrical Engineering
University of Rochester
Rochester, New York 14627

An implementation of FORTH for the Z80 microprocessor using the CP/M operating system is described. The implementation, MFORTH, is a family of subroutines written in Z80 assembler. A subset of these subroutines implement the FORTH-79 Standard word set minus the disk handling words. User commands (those recognized by the outer interpreter) are stored in one contiguous table of ASCII characters with the MSB of the last character set. Pointers to sub-tables are maintained to implement up to 16 vocabularies. An address table contains a list of the

addresses of the code to be executed by a FORTH word. The object code (which is executed by the inner interpreter) consists of the table number (one byte — corresponding to the vocabulary) and the relative position in the table (one byte). The core (or table zero) is implicitly assumed. Hence, the object code is relocatable — with the most frequently used words occupying a single byte of memory.

MFORTH loads as a CP/M transient program at 100H and relocates to 5000H with 4-1K buffers reserved for file blocks in high memory. Floating point routines load at 4000H. The memory from 100H to 5000H is used for CP/M transient programs, text editing, overlays, and/or data. A full-screen text editor (hardware-dependent) is built in to MFORTH. The file system currently implements only sequential files. MFORTH source "programs" can be written using any "standard" text editor and may include fixed tabs, carriage-returns, etc. A source "program" may contain nested files of up to three levels. String arrays and floating point routines are implemented by passing addresses on the data stack. Variable type information is stored with the variable. Separate stacks are maintained for loop parameters, file nesting parameters, data, and return addresses.

A Review of Recursive Structures in Forth

Lawrence P. Forsley
Laboratory for Laser Energetics
University of Rochester
250 East River Road
Rochester, New York 14623

The earliest documented uses of recursion in Forth were by Bartholdi for calculating N factorial and performing Hoare's quicksort. Since then several researchers have developed structures based upon recursion and defining words. This presentation will review recursion and recursive structures in their simplest forms, and demonstrate their application to data structures (Forsley), programming PAL's (Stolowitz), and system error recovery (Joosten).

A Forth-Based Array Processor Driver

Dr. Robert Frankel
Laboratory for Laser Energetics
University of Rochester
250 East River Road
Rochester, New York 14623

We have developed a Forth-based driver for a SKYMNK-02 array processor interfaced to an LSI-11/23 microcomputer. This system controls a two-dimensional T.V.-based X-ray detector used in pulsed X-ray diffraction experiments at the Laboratory for Laser Energetics of the University of Rochester. The array processor's computational speed greatly augments Forth's basic strength in generating compact real-time instrument control software.

The driver includes software that compiles array processor command blocks (1-10 words in length) into lists that are sequentially shipped to the SKYMNK at execution time. The software continually checks that the SKYMNK command queue (which has a maximum depth of 13 command blocks) has not overflowed. The SKYMNK handles DMA from the mega-byte of on-board memory, circumventing the need to handle memory mapping while moving data.

The array processor driver is used in image processing packages that include intensity and geometric correction of images, circular averaging of diffraction patterns, and Fourier analysis. The SKYMNK is also used in high speed intra-memory moves.

In summary, an array processor-based Forth system provides enhanced real-time control in areas where significant computation is required.

Use of State Tables and Sensory Information for Control of a Cartesian Robot

Judy Franklin, Terri Noyes, and Gerry Pocock
Perceptual Robotics Laboratory
Computer and Information Science Department
University of Massachusetts
Amherst, Massachusetts 01060

The work in control applications at the Perceptual Robotics Laboratory has been directed toward the General Electric prototype Cartesian Assembler. State tables are used in multiple axis coordinated-movement routines. These routines simulate the action of single axis controllers, but are implemented on a single PDP-11/23 system. Encoder/positional information is the basis of the low-level control structure which will later be tailored for use with processors devoted to each axis. High level control issues such as sensory-based (visual and tactile) control and adaptive learning techniques are addressed.

Towards a More Writable Forth Syntax

Harvey Glass
University of South Florida
College of Engineering
Department of Computer Science and Engineering
Tampa, Florida 33620

While Forth is a powerful and efficient language, it may also be justly criticized for creating unnecessary programmer obstacles [Loel] [Frei]. Particularly objectionable are the stack operators which tend to obscure the original solution, increase programmer error, and result in code which is often virtually unreadable.

We will describe a set of Forth extensions designed to reduce programming effort. These extensions provide the following features:

1) A programming style that requires no explicit parameter stack operations,
2) A straightforward notation for defining recursive functions,
3) The handling of forward references, and
4) A new construct for defining higher order functions that is similar to CREATE DOES> but easier to use.

The principal goal in designing a programming language should be to reduce the effort required to translate a problem solution into a machine language. Our specific goal with the extensions to Forth has been to preserve the power and flexibility of the language while providing a simpler and more readable notation. The resulting code is not unlike a postfix dialect of Pascal or Algol.

We will discuss the syntax of the extensions, and their implementations.

Forth as the Kernel of a Functional Programming System

Harvey Glass
University of South Florida
College of Engineering
Department of Computer Science and Engineering
Tampa, Florida 33620

Recent interest is surfacing in programming languages and programming systems which support more productive and creative programming environments [Turn]. This interest stems from research in a variety of areas including new machine architectures [Acke], artificial intelligence [Suss], expert systems, and proof of the correctness and equivalence of programs [Back].

The often unconventional languages required to support these environments, are associated with a programming style called functional or applicative [Hend]. Languages mentioned frequently in connection with this programming style include Lisp [Frei], Smalltalk [Mark], and FP [Back].

We will discuss the characteristics of functional programming languages, and describe work now in progress which uses Forth as the kernel of a language with these characteristics.

Telescope Pointing — A Forth Application

James V. Harwood
Institute for Astronomy
Honolulu, Hawaii

Most major telescopes are computer controlled and require software to accurately maintain pointing and tracking of celestial objects. Mechanical sources of pointing errors in the NASA 120 inch infrared telescope on Mauna Kea are examined. A sky map of 100 stars in all parts of the viewable sky was made and is maintained along with pointing correction coefficients. The map is then used to compensate for telescope axis mis-alignment, encoder errors, telescope flexures and other errors. The result is a substantial improvement in pointing and tracking accuracy. eds.

Blocking Out Bad Memories

Peter H. Helmers
Adaptable Laboratory Software, Inc.
P.O. Box 18448

Blocks have traditionally been used within the FORTH community to provide a standard mass storage system interface. While useful for some "virtual data" storage purposes, the application of blocks to source program textual data storage has many disadvantages. To replace blocks for text storage, a simple system was developed to allow access

to standard operating system text files. Although written for Laboratory Microsystem's PC/FORTH running on an IBM personal computer with the PC-DOS operation system, the concept can be applied to most FORTHs. The advantages - and disadvantages - of the resulting system are discussed.

Is This Still FORTH?

Peter H. Helmers
Adaptable Laboratory Software, Inc.
P.O. Box 18448
Rochester, New York 14618

An Apple II computer running A.L.S.'s ALS.GRAPHICS/MATH will be shown to demonstrate several advanced FORTH capabilities:

- An interactive HELP system
- Complex and Real arithmetic using "typed" numbers,
- Algebraic notation
- Engineering graphics

Generic Operators for Use with Infix, Postfix, and Prefix Notations

David Hofert
and
Lawrence P. Forsley
Laboratory for Laser Energetics
University of Rochester
250 East River Road
Rochester, New York 14623

Specific number types, like single precision and double precision integers, floating point, and complex numbers, require specialized mathematical operators. In Forth, this results in an excessive number of operators. As an example, with plus there would be I+, D+, F+ and C+ for the above set of number types. We have developed generic dyadic operators which are executed via a sparse, three-dimensional matrix which is triangularized to save space. A plane passing through the diagonal of the matrix describes dyadic functions of like elements, whereas those elements off of the diagonal contain conversion rules and an address on the plane for the converted operation. Monadic functions have also been implemented, such as negation, but they only require a sparse two-dimensional matrix. In addition to standard mathematical functions, generic, relational, boolean, and storage operators can be defined. This system can be coupled with a parsing front end, such as described by Helmers or Stolowitz, to work with prefix or infix notations as well.

DELTA: A Forth-Based Expert System

Harold E. Johnson, Jr.
General Electric Company
Corporate Research and Development
1 River Road - Building 37-545
Schenectady, New York 12345

In recent years, expert systems have become the most visible and fastest growing branch of Artificial Intelligence. Their objective is to capture the knowledge of an expert in a particular problem domain, represent it in a modular, expandable structure, and transfer it to other users in the same problem domain.

Expert systems consist of a knowledge base (facts about the problem and heuristics, or rules, that control the use of knowledge to solve problems in a particular domain) and an inference engine for interpreting this knowledge. The inference engine can operate in both forward mode (event-driven) and backward mode (goal-driven).

General Electric Company Corporate Research and Development has applied this technology to demonstrate its feasibility in the area of troubleshooting diesel electric locomotives. The selected problem is to provide "expert" assistance to railroad maintenance personnel in "running repair shops", enabling them to detect and repair a large variety of faults, which have partially disabled a diesel electric locomotive, within a two-hour window. The solution is DELTA (Diesel Electric Locomotive Troubleshooting Aid), a rule based expert system which guides the troubleshooter in performing his task, enforcing procedures which will minimize the cost and time of the corrective maintenance.

A prototype DELTA system has been implemented in FORTH and runs on a Digital Equipment PDP 11/70 under RSX-11M+. This system contains approximately 400 rules, partially representing the knowledge of a Senior

Field Service Engineer. The system also provides an on-line help system with graphic/video capabilities to aid the user in locating and identifying locomotive components, as well as illustrating repair procedures. Additionally, the user can ask the system why a particular question is being asked of the user, giving insight into the expert system's current line of reasoning. The result of a DELTA session is a final diagnosis which indicates the successful hypothesis (faults) and their corresponding actions (repairs).

The inference mechanism, originally developed in FRANZLISP for a feasibility study, is now fully implemented in FORTH. The FORTH version provides a powerful representational language consisting, in part, of seven types of predicate functions, eight types of verbs, and five help functions. It also runs efficiently and can be easily transported to small microprocessor-based systems. A rugged version of this system will be tested in the field in July 1983. The number of rules will soon be raised to approximately 1000 to cover, with increased depth, a larger portion of the problem space.

A FORTH-System for a Waste Water Treatment: Plant Control and Data Logging

Dr. C. P. Kuznia and A. Kroneberg
Contraga GmbH
Herbststr. 9a
8038 Grobenzell
West Germany

As requirements for waste water treatment are becoming more strict, computers are more widely used in the control of waste water treatment plants. Our company has installed a number of microcomputer-based systems, most of them for data logging.

The hardware is based on the STD bus with the 8085 CPU. Digital I/O is handled over optocouplers. The slow, varying analog signals are multiplexed by a relay mux and isolated before the ADC. Terminals and graphic plotters are standard. The software is essentially the fig-model, modified and expanded for our purposes. Our multiprocessor system consists of a master and several satellites out in the site. Each satellite serves a subsystem. Communication, master-satellite, is by RS232. All processors have stand-alone Forth systems. When timeout occurs, they continue working in a default mode.

For process control, a flexible main task handler starts the tasks from a real-time clock. The system does not use hardware interrupts, as no fast response is needed. Typical control loops are: control of dissolved oxygen in a number of aeration tanks, control of blowers, pumps (e.g. reflux sludge), and energy consumption. We use a number of generating words to set up data acquisition modules, PID control loops, and other repeating code.

Conclusion: We use Forth exclusively in our systems. Main advantage: The different software levels are interactively available to the engineer. This means great flexibility in setting up and debugging the system in the plant. We know of no other software tool which makes work so effective in real-time control applications.

A FORTH-Based Bioengineering Class

Steven M. Lewis
Department of Bioengineering
University of Southern California
Los Angeles, California 90089

The bioengineering department at U.S.C. was faced with the problem that the computer experience of most students was limited to large time-sharing systems. Most of the biomedical uses of computers involve small systems which are intimately involved in data acquisition and in the control of laboratory apparatus. This meant that the most valuable skills for our students were the control of peripheral devices and real-time programming, skills which were unavailable on a larger machine. We decided to develop our own computer course to emphasize these skills.

VERIFY - A Useful FORTH Programming Tool

Steven M. Lewis
Department of Bioengineering
University of Southern California
Los Angeles, California 90089

In preparing to teach a class in FORTH, I realized the need for a FORTH word which would be rapidly able to list the consequences of the execution of a FORTH word. It is clumsy to print the stack before and after execution. Sometimes, particularly when the stack is full, it is difficult to tell from this printout exactly what a word is doing. Furthermore, words have effects beyond modification of the stack, changing the contents of variables and other

memory locations. I found in my own use of the language that it is rarely necessary to single step a word to find a bug, usually words may be treated as black box routines and only their consequences need be examined. I sought to develop a debugging tool which would rapidly and easily tell me about those consequences.

Forth-Based Software for Real-Time Control of a Mechanically-Scanned Ultrasonic Imaging System

E. T. Lynk and *H. E. Johnson*
General Electric Company
Corporate Research and Development Center
Schenectady, New York 12345

FORTH is currently being used as the control software for an ultrasonic imaging system which is under development at G.E./CRD. Our original intent was to use FORTH in this application only for debug of the hardware interfaces. However, the ease with which enhancements were made, to allow high level interrupt handlers and access to extended memory, encouraged us to write the entire application in FORTH. The system is an "Ultrasonic Microscope", and is to be used in industrial "Nondestructive Evaluation" applications. It forms an image by mechanically moving an ultrasound beam over the surface of the material under test. The beam is tightly focused at a constant depth below the surface. Strong echoes are received from flaws, voids and inclusions which lie within the focal region. By electronically gating the portion of the signal generated by the echoes, then plotting the maximum value of the gated data, a map can be made of flaw locations within the sub-surface slice.

The Evolution of FORTH through its Application

David B. McClain
FORTHright Engineering, Inc.
7901 East Boojum Street
Tucson, Arizona 85730

FORTH programming languages are becoming more popular all the time. Engineers find it more direct and effective than other languages in the development cycle of new products — especially in the robotics fields. Newcomers to the language experience a robust love affair with it and the use of computers to control equipment. However, this effectiveness is largely due to the ignorance of programmers in the engineering disciplines and the engineers' ignorance of the programming discipline. Distinguished professionals in the computer science field look with disdain upon FORTH. Why is this and what can be done to enhance their view of the language?

This paper will address the question of the true usefulness of the FORTH language to the newer technologies beginning with a historical review of one man's encounter with the language. It will be argued that FORTH, although very good for certain applications, is not sufficient for the technologies of today and tomorrow. Nor are any other conventional programming systems — although a combination and evolution can be developed, and indeed is within view — which can allow for the error free and speedy development of control systems, as well as other more conventional programming tasks. Some constructive suggestions for future work in language design will be presented.

Identification of Functionally-Related Word Groups in Forth Programs by Correlating Word Usage

Herbert C. McClees
ELDEC Corporation
16700 Thirteenth Avenue West
Lynnwood, Washington 98036

The extensibility of Forth makes it practical for a special purpose language to be developed as an integral part of an application program. However, the 'open-ended' nature of Forth presents a special challenge in controlling and documenting an evolving application shaped by a creative individual. A new generation of software tools can be imagined which support the requirements of extensible languages. The tool described uses a correlation technique to identify functionally related words based on their actual usage in a program. This tool can be consulted by the programmer during program development to rationalize word naming, usage and factoring as well as to gain insight into the functional structure of programs lacking documentation. It could also become the basis for objective comparison of similar programs by different authors.

They, Robots

Charles H. Moore
Moore Associates
Manhattan Beach, CA

What is a robot and how should it be programmed? Robots, whether they are a gas chromatograph, a 300 foot radio telescope, or something more popularly recognizeable as one, can interact with their environment. They are inherently real-time devices and require Forth's data structures, control structures and multitasking. Forth's strengths and simplicity increase the probability that reliable robotic control software can be written. eds.

Forth Products for Business Packages

Pierre Moreton
40 Rue du Mont Valerien
92210 St. Cloud
Paris, France

The presentation covers two products:

1. MFORTH : Multi-machine Forth System

   This is especially designed to meet the specific requirements of the business packages that have to be marketed on a large scale.

   - A unique source code (Metacompiler) is used to generate interactively a multitasking Forth System for any classical micro- or minicomputer, Z80, 8086, or LSI-11 based.
   - A powerful DBMS is included with:
     - built-in security system for file integrity and resources management.
     - complete dynamic disk space allocation for:
       editing source code
       creating binary overlays
       data file management
   - A very high level programming instruction set (close to natural language) to reduce development costs and give full readability of the source code decreasing the maintenance costs.
     We call it HFORTH (Human Forth).

2. OMNISOFT

   This package enables anybody (even non-programmers) to develop, in some hours, any business software at his own specifications. It can be favorably compared to DBASE II or MDBS for instance. It has an internal, relational-like, database (it runs on MFORTH), a report generator, and a word-processor connected to the base. High-level, menu-driven, functions enable the user to create, modify, delete, multi-index, update related files, control, select, extract, calculate, report, . . .

   OMNISOFT has been deeply studied for an ergonomic and very easy to use approach. For professionals it can be a powerful tool to create a business software catalogue.

Computer Music Synthesis
A Real-Time Application in FORTH

Jeff C. Morriss
4711 Beau Bien Boulevard East
Lisle, Illinois 60532
and
Kim Harris
1055 Oregon Avenue
Palo Alto, California 94303

This paper describes a computer music synthesis system consisting of a high-speed digital signal generator unit (DSGU), a microcomputer, and a music compile/playback program implemented in FORTH. Attention has been concentrated upon achieving polyphonic, real-time capabilities in software that are commensurate with those of the DSGU hardware.

Consisting of 64 time multiplexed digital oscillators, the DSGU supports additive, frequency modulation, and stored waveform synthesis techniques. Each oscillator has independent low-level parameters for amplitude, frequency, phase, envelope shape, and mode control.

The music compile/playback process consists of two steps and involves the filling and subsequent emptying of 64 queues (one per oscillator). The music compiler operates upon input parameters of tempo, timbre, and score to

generate a time-ordered set of low-level parameters which are stored in the queues. Each queue element specifies DSGU parameters, queue number, and a time value of when to send them. Playback involves reading the queues' time signature, and at a specified time passing the low-level parameters from the microcomputer to the DSGU.

Recursive Dragon Designs

Victor T. Norton, Jr.
Department of Mathematics and Statistics
Bowling Green State University
Bowling Green, Ohio 43430

We use FORTH to recursively generate variants of the Heighway dragon curve. This mysterious curve first appeared in Martin Gardner's mathematical games section of *Scientific American* (March, April 1967). It was discovered by NASA physicist John E. Heighway after an exercise in paper folding.

Our constructions are based on the dragon recursion formulas.

$S_1 = L$, $S_{n+1} = S_n L S_n$ (binary dragon),

$S_1 = LR$, $S_{n+1} = S_n L S_n R S_n$ (ternary dragon).

Here L and R represent left and right turns respectively. Moves before and after turns are not symbolized.

The binary dragon formula can be realized by the FORTH word

```
: BDRAGON (level--)
   ?DUP IF DUP ABS 1- SWAP OVER
           MYSELF BDRAGON-TURN NEGATE MYSELF
        ELSE BDRAGON-MOVE THEN ;
```

with

```
: BDRAGON-TURN (level--)
   90 SWAP 0 < IF NEGATE THEN TURN ;
```

The ternary dragon formula is implemented similarly, with 120° replacing 90°.

When one interchanges L and R at arbitrary, pre-chosen levels in either recursion formula, the generalized dragon curves of C. Davis and D. E. Knuth result. Striking patterns arise from using the empty word for BDRAGON-MOVE and incorporating moves in BDRAGON-TURN. The latter variations are easily accomplished in FORTH by vectoring the execution of the words BDRAGON-TURN and BDRAGON-MOVE.

Under appropriate circumstances, twin-dragons and triple-dragons,

$T_n = S_n L S_n L$ (twin binary dragon),

$T_n = S_n L S_n L S_n L$ (triple ternary dragon),

sweep out simple closed curves. In the generalized case, L and R are interchanged by the prescription for the underlying dragons. Low level twin and triple dragon blocks can be fitted to create an endless variety of interesting tilings and wallpaper designs.

Design of a FORTH-Based Robot Control Language

Terri Noyes, Gerry Pocock, and *Judy Franklin*
Perceptual Robotics Laboratory
Computer and Information Science Department
University of Massachusetts
Amherst, Massachusetts 01060

The Perceptual Robotics Laboratory is using the FORTH language to build manipulation and low-level control routines. A high-level language (the Perceptual Robotics Language) for specifying tasks which use dynamic sensing within a distributed architecture is being constructed to ride atop FORTH. A major focus of this paper is the philosophy adhered to when designing the Perceptual Robotics Language. The design has evolved from (and thus the paper begins with) a discussion of what properties are desirable in a language for robotics, and which features are pertinent to the needs, resources and goals of the Perceptual Robotics Laboratory. The paper discusses the usefulness of FORTH as a base language from which to build a planning language with the features desired. Included are considerations on concurrency, interrupt capabilities, force and sensor monitoring, data manipulation and language type and structure.

System Architecture of the Perceptual Robotics Laboratory

Gerry Pocock, Judy Franklin and *Terri Noyes*
Perceptual Robotics Laboratory
Computer and Information Science Department
University of Massachusetts
Amherst, Massachusetts 01060

This paper serves to introduce to the FORTH community the Perceptual Robotics Laboratory in the Department of Computer and Information Science at the University of Massachusetts at Amherst. It describes the specific hardware layout of the laboratory, including the computer equipment and the robotic machinery. The Laboratory includes a General Electric prototype cartesian assembler, donated to this group by General Electric of Schenectady, New York, as well as several Digital Equipment Corporation computer systems. Also included in this paper is a discussion of future extensions to the lab in terms of computing power, manipulator hardware, and software.

A Software Clock-Divider

Carol Pruitt
Laboratory for Laser Energetics
250 East River Road
Rochester, New York 14623

The problem: An A/D converter is to be sampled twice per second. A line-clock interrupt is available sixty times per second, but the sampling routine takes longer than a sixtieth of a second to run.

The solution: A clock-interrupt handler updates time-of-day, then if half a second has elapsed, allows interrupts (including itself) to be serviced while the sampling routine runs. Except for a small amount of set-up and exit code, everything is written in Forth.

Single-Key Macros: Vectored I/O in Action

John Rible
Miller Microcomputer Services
61 Lake Shore Road
Natick, Massachusetts 01760

I/O vectoring has been presented in many versions, usually to allow multiple devices to be easily interchanged. A new approach to vectoring was presented at the FORML conference last year. This talk describes an application made very easy by this vectoring scheme: single-key macros, which expand a single key-press into an arbitrary string.

The word ?KEY , defined as

```
: ?KEY ( -> c )  0 NOP NOP ?KEYBOARD NOP NOP ;
```

allows the device-driver words for the various sources to be arranged in a particular order. By designing each driver to accept input only when the top-of-stack is zero, a simple priority scheme is implemented.

Single-key macros are implemented by replacing two NOP's in the ?KEY vector with ?MACRO and INTERCEPT . ?MACRO , placed before keyboard input, retrieves keys from the expansion string instead of from the keyboard. INTERCEPT is placed after keyboard input and, when activated, looks for the keycodes to be expanded.

Support words also are described: those required to enter, edit, and display the macro strings, as well as the words used to activate and deactivate the facility.

FORTH and Automation Research at the National Bureau of Standards

William G. Rippey
National Bureau of Standards
Building 220, Room A-353
Washington, D.C. 20234

The National Bureau of Standards (NBS) is conducting a large research project in industrial automation. Automated control of manufacturing processes will be achieved through extensive use of computers. Some of the individual projects implemented in FORTH will be described. Some FORTH techniques used on each project will be highlighted.

Dysan IEEE P-754 Binary Floating Point Architecture

Jonathan R. Sand
Dysan Corporation
15495 Los Gatos Boulevard, Suite 1
Los Gatos, California 95030
and
John O. Bumgarner
Information Appliance, Inc.
15010B Monte Bello Road
Cupertino, California 95014

This paper describes a software implementation of Floating Point Arithmetic operations conforming to the proposed IEEE P-754 standard. It is written in FORTH and integrates floating point operations into the FORTH environment without the need for special hardware processors. The source program is entirely in high level FORTH to enhance transportability and modifiability. A test suite is described which can verify both software and hardware implementations of the IEEE standard.

Describing Activity Scheduling Problems with Forth

Joel Shprentz, Jim Whitehead, and *Art Kubo*
The BDM Corporation
7915 Jones Branch Drive
McLean, Virginia 22012

Forth was used to develop and test several resource constrained activity scheduling algorithms. The algorithms require data describing the duration, resource allocation rules, and precedence constraints of activities; and the quantity and performance of resources. An extension to Forth for describing activity scheduling problems is presented.

A "scheduled language" was developed by extending Forth to include words that describe the resources and activities of scheduling problems. The user can describe a scheduling problem in familiar terms without understanding the details of the underlying Forth program. The "input" is not input in the traditional sense, but is Forth source code which is compiled by the Forth outer interpreter.

For example, resources are defined by the Forth word RESOURCE. The availability of five graders is defined by the Forth code:

```
5 RESOURCE GRADER
```

This is usually followed by a description of the graders' performance parameters. GRADER is a Forth word and can be used in a definition. For example, it is used in the calculation of the quantity of pushing equipment in use by an activity:

```
: PUSHING-EQUIPMENT
    GRADER USED   DOZER USED   LOADER USED   + + ;
```

When the scheduler was rewritten in Fortran 77, an LR parser was used to preserve the flexibility and ease of use provided by a scheduling language.

A File System in Forth: One Approach

Theodore Sizer II and Lawrence Forsley
Laboratory for Laser Energetics
250 East River Road
Rochester, New York 14623
and
Peter H. Helmers
Adaptable Laboratory Software, Inc.
P.O. Box 18448
Rochester, New York 14618

A file system has been developed for use with the FORTH language combining many features that we deem desirable for clear, well documented FORTH programming. Some of these features are: 1.) Compressed text to allow for blank spaces inserted for program clarity without sacrificing program storage space; 2.) Unlimited number of lines per file and the ability to insert a line in the middle of a file with ease; 3.) Unlimited number of characters per line (usually terminal limited to 80) to more completely document each line of code; and 4.) Maintain the ability to load these files with the standard block-oriented compiler after only minor modifications made to handle compaction

characters. This modification once made, however, one can then load both the file oriented programming and the block oriented programming. In addition to the file system, a non-memory mapped screen editor and a formatter used in conjunction with this file system have been developed.

A more complete description of the file system will be presented along with more of the motivation behind such a system, the benefits it brings the programmer, and the modifications planned for the future.

A Robot Language for Two Unique Applications

Dan Slater
Elicon
273 Viking Avenue
Brea, California 92621

Robotic technology is rapidly expanding into new areas of application. Two new applications are the filming of motion picture special effects and the measurement of spacecraft antenna performance. Two robotic systems performing these functions are described. Both systems are programmed in the FORTH language and use state space methods to produce a highly modularized and reliable control program.

An Instruction Set Architecture for Abstract Forth Machines

Nicholas Solntseff
Unit for Computer Science
McMaster University
Hamilton, Ontario, Canada L8S 4K1

This paper describes current work at McMaster on the design of an instruction set architecture for the implementation of an Abstract Forth Machine [SOL82]. The machine language is called F-code and is designed to simplify the comparison of Forth implementations on real machines with widely different architectures, as well as to provide an aid to the standardization of Forth.

Reserved Prefixes: A Route to Identifying Standardized FORTH Words and Segregated Application Packages

Dr. Leonard Spialter
INSTRUMENTORS
P.O. Box 338
Dayton, Ohio 45406

To counter the growing proliferation and resulting confusion between standard and non-standard FORTH words, a pair of suggestions is offered. The first is to attach "reserved prefixes", possibly of the form "Snnn.", where 'nnn' is an identifier of the relevant standardization authority, to each authorized standard FORTH word. The second is to redefine the ":" (colon) word with filters which will not allow redefinition of a standard word, such as the current single characters ":", ".", ";", etc., or the proposed prefixed ones. These suggestions will make feasible the establishment of useful transportable FORTH application packages and software catalogs, the setting up of accepted word genealogies, and the immediate visual identification of standard words. Current non-standard practices can be continued but resulting words and definitions will be considered local constructs with an implied user caveat.

A Hardware/Software Finite State Machine Implementation

Michael K. Starling
Union Carbide Corporation
South Charleston, West Virginia 25303

Finite state machines have numerous uses in the real world. If implemented in a flexible way they can be quickly adapted to new problems. This paper discusses an approach which combines both hardware and software to yield a flexible engine for the service of contact closures. It provides a means of controlling program flow based on external conditions in a direct and highly structured way.

Improvement of a Human/Robot Interface through the Use of FORTH

Doug Thompson
International Robomation/Intelligence
2281 Las Palmas Drive
Carlsbad, California 92008

International Robomation/Intelligence (IRI) is a forward looking company located in Carlsbad, California. It manufactures a computer-controlled 5 axis robot arm for industrial uses such as machine loading/unloading and parts transfer.

IRI recently decided to update its computer operating system from the present assembly code version to FORTH. This paper describes the usage of FORTH for man/machine interfaces. It deals with such concerns as maintainability, expandability, processing speed, memory usage, and operator interface. Language tradeoffs are discussed as well as non-typical FORTH usage such as PROM resident code and limited disk usage.

IRI has combined many unique features to provide a truly "affordable" robot which includes an onboard computer, pneumatic drive, and aerospace-derived construction techniques. The IRI M50 is controlled by a heirarchy of microcomputers with the Motorola 68000 16/32 bit processor as the manager of six 6803 axis control computers.

User programming is accomplished via IRI's simple and straightforward on-line Robot Command Language (RCL). Special emphasis will be placed on how FORTH has been adapted to the production environment where the end user may have little or no programming experience. Topics of concern include system security, ease of use (flexibility vs. understandability), and debugging features of RCL.

The approach taken emphasizes at all times the relationship between human and robot rather than the physical control of the robot itself. Industry is constantly striving to simplify this relationship. The role FORTH can play is central to this paper.

Optimization of Three-Dimensional Forth Graphics Using an Array Processor

Carole A. Winkler
Standard Oil Company of Ohio
Laboratory for Laser Energetics
250 East River Road
Rochester, New York 14623

This paper is basically a case study in optimizing three-dimensional graphics in Forth. A PDP-11/23 was used, together with a Chromatics color graphics computer and an RL01 5-megabyte disk.

The original graphics package could rotate a wire frame cube at a rate of 17 seconds per rotation. By installing a Sky Computers array processor and rewriting much of the high level Forth code in assembler this time has been cut to 3.3 seconds per rotation.

A FORTH Simulator for the TMS 320 IC

Richard Wyckoff
University of Rochester
Rochester, New York 14627

The Texas Instruments TMS 320 chip is a special purpose, low cost, high speed signal processing chip. The chip has a limited amount of parallelism which makes its programming somewhat similar to horizontal microcoding. This paper describes a cross-assembler and hardware simulator developed in Forth. The cross-assembler allows for labels, macros, and high level language constructs. The simulator makes use of knowledge obtained at cross-assembly time, and hence, executes Forth code emulating the chip directly. Forth naturally provides many of the micro-tasks required to fetch and execute TMS 320 instructions, and possibly other computer architecture instructions as well.