SOFT

Volume 1
No.6
95p/$2.50

THE MICROCOMPUTER USER'S JOURNAL NOVEMBER/DECEMBER 1983

FINANCIAL PLANNER
Ashton-Tate's modeller

NANO 6502
Learning machine code on a simulator

SUCK IT AND "C"
Three C compilers tested

REAL TIME ON ZX81!
About that power station...

OPTIMISER
Linear programming without tears

MISTER KNOW-IT-ALL
The Knowledgeman database

WAR IN THE AIR
Games for Vic & Spectrum

SWOT SQUAD
Chemistry & Optics on the Beeb

STIX PIX ON DISKS
Pictorial databases are here!

INSIDE THE GUIDE TO BUSINESS SOFTWARE

ABOUT THAT POWER STATION...

An incredulous Dick Pountain found a way to do real-time multi-tasking on a Sinclair ZX81 using Forth.

It seems like an age ago that we were all guffawing over Uncle Clive's original ad. for the ZX81. Control a power station indeed!. With that curious little sliver of black plastic?

It's rather easy to forget when contemplating the ZX81 that its diminutive case conceals a Z80 running at 3.25 MHz, which gives it rather more potential processing power than some CP/M machines costing thirty times more than its £50 price tag. What prevents much of this potential being realised is lack of memory and a ROM-based Basic interpreter which is far from quick even in its Fast mode (and the way that video output is handled doesn't help at all since 80 percent of the processor's time is consumed in updating the screen if a continuous display is required).

Shortage of memory plus lack of quickness? That sounds like a job for . . . Forth! (pause for snatch of William Tell overture, galloping hooves). The ZX81 contains at least as much RAM and processor as scores of Forth-driven controller boards, some of which contain as little as 256 bytes of workspace.

ZX81-Forth, the subject of this review, is remarkable not because it is the first Forth implementation for the ZX81 (that honour goes to Artic) but for the features which it has contrived to include. The most prominent of these is multi-tasking and real-time programming; it is possible to have ten tasks running simultaneously on a machine with the 16K expansion RAM pack fitted. It also has a screen-oriented editor and many interesting and innovative (hence non-standard) extensions, especially in the area of character I/O.

As well as running very much faster than Sinclair Basic, this Forth turns the ZX into a potential real-time controller, though I suspect it is better employed as a cheap way of learning about real-time programming than to run Sizewell B.

Multi-tasking is necessary for real world control jobs because, in the words of the sage, "time waits for no man, let alone some dumb computer". In other words, things happen in the real world when they feel like happening, not when it is convenient for the computer. It avails naught to send a message to the reactor saying "please don't go critical just now, 'cause I'm in the middle of updating the screen". A real world controller must be able to give its attention at all times to all the processes it is controlling, which means that it must be able to do more than one thing at once, and it must know what the time is too.

ZX81-Forth is written and sold by David Husband of Poole, Dorset, who has considerable experience of using Forth in instrument control applications via his activities in amateur radio, and has previously implemented a more orthodox fig-Forth system for the Nascom. Some of the novel aspects of *ZX81-Forth* are based on Tree-Forth, a US developed version which is used to drive a well known bulletin board network.

INSTALLATION

ZX81-Forth is ROM based and is sold in an 8K EPROM which replaces the Sinclair Basic ROM. Depending on your machine's vintage, this could involve you in soldering work. The Basic ROM is replaced by a 28-pin IC socket which accepts the Forth ROM. If you don't have a ZX already, Husband has an arrangement with a Poole dealer who can supply ready converted machines (see Fact Sheet) for around the normal ZX81 price.

A 16K RAM pack is required as minimum, but if you have one of the larger ones *ZX81-Forth* will automatically sense its presence and use the available space.

Once the ROM is correctly installed the machine comes up into *ZX81-Forth* on power up. The functioning of the machine is altered in more ways than merely a new language; the Sinclair character set is replaced by the standard ASCII one (no graphics characters) and the keyboard now has only one shift mode which gives the ASCII symbols. The internal key codes remain Sinclair's though. The single stroke keyword entry, which would hardly be appropriate for an extendible language, is not retained.

One perhaps obvious point is that the extra typing involved points up the inadequacy of the ZX81 keyboard (which could have lost St Francis his sainthood had it been invented) rather cruelly, so one of those keyboard kits would be a useful extra purchase.

IMPLEMENTATION

ZX81-Forth is a subroutine-threaded version of the language. Without going into too many esoteric details of how Forths (or more generally Threaded Interpretive Languages) work, it is hard to explain what this means. Very roughly it means that it compiles source programs into a list of subroutine calls, which are direct calls to executable machine code. Most Forths compile into a list of addresses which are then interpreted by an inner interpreter. The distinction is almost that between a native code compiler and one which compiles into p-code. What it means is a speed improvement because *ZX81-Forth* doesn't need the inner interpreter. As it is ROM-based and entirely written in machine code (most Forths are largely written in Forth) the core of the language cannot be modified at all, though the user can of course add new definitions to the part of the dictionary which lives in RAM.

As 8K is quite small even for a Forth implementation, chunks of the fig standard have had to be left out to fit in the extra code to do multi-tasking. Nevertheless it looks very like fig-Forth in its essentials, namely stack manipulation, arithmetic, memory manipulation and control structures. It actually goes beyond fig in having a simple but ingenious CASE statement which is a defining word used in place of a colon. I was able to run my Forth Benchmarks with only one major modification due to the odd syntax of BEGIN. . .WHILE. . .AGAIN which acts like a switch between the two code segments rather than as a "one-and-a-half loop" as it should do. These timings incidentally show *ZX81-Forth* to be averagely fast for a Z80 system, though they do not test some features, such as screen handling, which are very fast indeed. In practice it is likely to be more than 50 times faster than Sinclair Basic for many tasks.

The <BUILDS. . .DOES> construct is included so that the full flexibility of user-defined data structures is available.

The major omissions are the lack of vocabularies, which will not be much missed in a system this size,

DATASHEET

NAME OF PROGRAM
ZX81-Forth

MANUFACTURER/COUNTRY OF ORIGIN
David Husband, UK

PURPOSE/TYPE OF PROGRAM
Multi-tasking programming environment

COMPUTERS SUPPORTED
Sinclair ZX81

MEDIUM ON WHICH SUPPLIED
EPROM

MINIMUM HARDWARE TO RUN
16K RAM Pack

PRICE OF PROGRAM/MANUALS (INC VAT)
£28.75

REVIEW COPY OBTAINED FROM
David Husband, 0202 764724

and a reduced set of mass storage instructions to go with the simple editor and cassette storage.

No Editor vocabulary is required as the editor is always present, and no Assembler vocabulary is provided; instead machine code may be freely introduced in line with Forth code using the words CODE and ;C.

Arithmetic is very generous, with a number of double and quadruple precision (64 bit) operations provided as befits a system which is intended for real-time applications. Floating point will be offered later as part of an extension ROM.

The handling of string and character I/O is very unorthodox and poses interesting questions for the future direction of Forth. None of the words which will be familiar to Forth programmers such as EXPECT, QUERY, WORD or TYPE are given; instead a separate character stack is used to hold string data and a whole new set of words to manipulate it. A most unexpected feature of these is that multiple screens may be set up on the (32x24) video display, each with independent scrolling (though overlapping Lisa style is not permitted). Screens, defined by the coordinates of their diagonal corner points, are given names in the dictionary and then the word .W can be used to write strings from the character stack to a named screen. To display numeric information in such a screen it must first be converted to string data using one of several conversion words. Screens may be switched between reversed field and "normal" ie. black on white. Together with the multi-tasking and the extremely rapid screen writing, this allows for some very un-ZX-like displays!

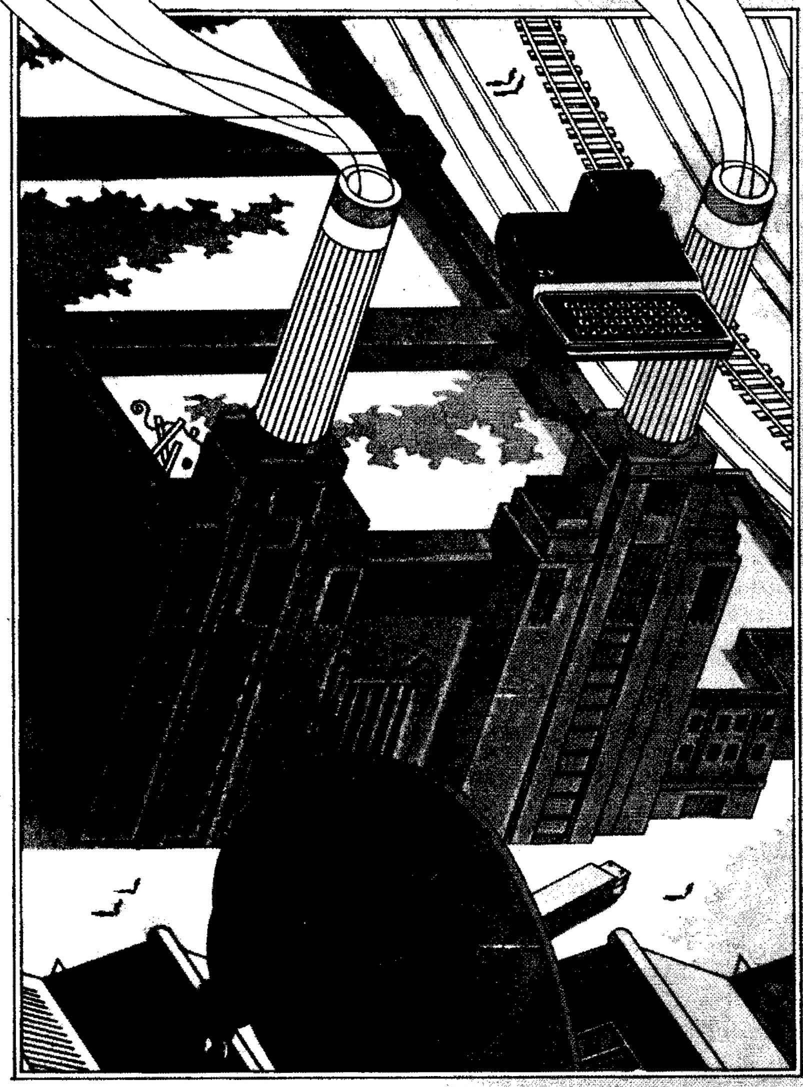
This way of doing character I/O is very different from any of the standard ways, which are more machine oriented and regard strings as arrays of bytes to be manipulated. The difference runs very deep; although *ZX81-Forth* has many standard words (such as '.' to print the stack top) they are all defined using the character stack. '.' works by converting the stack top value into a string and printing it with .W which is the more primitive operation!

FAST and SLOW modes are available as in Sinclair Basic (in FAST the screen blanks while computation is in progress) but an AUTO mode exists which blanks the screen only if a process takes longer than a quarter of a second to execute. This is the mode in which I ran the benchmarks.

ZX81-Forth certainly feels rather different from any other Forth I have used. Its method of compilation allows the use of DO. . .LOOP, IF and BEGIN structures directly from the keyboard outside of colon definitions; in addition error checking is performed when you press SPACE so that you can get a wrist-slap even before you hit return! Several words actually execute on SPACE as well as RETURN which is disconcerting at first.

THE EDITOR

ZX81-Forth also departs from orthodoxy in its editing and storage methods. Instead of a separate text editor which works on 1024 character blocks, as is usual, Husband uses a permanently present edit mode. The screen is split horizontally in two, the upper part being used for editing and the lower for execution. Pressing SHIFT EDIT toggles the cursor between these two display areas. Code written in the editing part (which does not scroll) stays permanently on the screen until erased or a warm start is performed. Editing within the edit screen is by full four direction cursor movement in an inserting mode, with RUBOUT for deletion. The edit screen holds 15 lines of 32 characters, and is separated from the 8 line execution screen by a reversed strip in which the contents of the PAD are displayed; this is used a temporary "parking space" for lines of text that need to be moved.


ROY INGRAM

Once a definition is finished it can be compiled immediately by pressing SHIFT Q or the whole edit screen can be compiled together by executing CPL in the execution screen. If your program fills more than one edit screen you will have to compile one screen, erase it and then write the next. The contents of the edit screen can be saved to tape or loaded from tape as a numbered block by issuing SAVE or LOAD in the execution screen; any text, not only Forth code, can be so stored. Provision is made for chained loading of blocks using the --> word, which means "now compile the next higher numbered block". It came as no surprise to me to find that the ZX81 would not SAVE or LOAD on my cassette recorder (I've never successfully loaded a ZX81 program in my life, though a million people somewhere do it). It did cause some cursing in the early hours when I was trying to run the benchmarks.

Though this is a very simple and limited editor, it is highly effective and far less terrifying to the novice than the dreaded Ragsdale line editor, which must have put more people off Forth than I care to think about; its immediacy resembles that of a good Basic editor such as Commodore's or Sharp's. It is rather unwieldy for large programs but this is hardly the machine for such project anyway.

MULTIPLE TASKS

And so the meat of the matter. *ZX81-Forth*'s claim to fame (and indeed to an article in this magazine) lies in its multi-tasking ability. As multi-tasking is more usually associated with minicomputers, or at the least 16-bit micros, it is surprising to say the least to find it on a machine at this price. The answer to this riddle lies in the ZX81's architecture which makes extensive use of interrupts; it would actually be harder to achieve multi-tasking on the average CP/M machine.

ZX81-Forth is multi-tasked even before you set about defining your own tasks. Three tasks run continuously: a 1/50 of a second clock task which is the Master Task (and so uninterruptable); a System Task which does the interpreting and compiling; and the keyboard scan which is treated as a task. As a consequence of the latter, it is always possible to stop a *ZX81-Forth* program with the break key. Even a definition such as:

: CONTEMPLATE-NAVEL BEGIN AGAIN ;

which spells suicide in ordinary Forths, can be stopped since the keyboard task continues to run alongside the infinite loop.

Defining your own tasks is easy to do, though inevitably also easy to do wrong. Any Forth definition can be declared to be a task by giving it an arbitrary task-name, so:

TASK FIRST-TASK MYDEF

where MYDEF is a colon or code defined word. The task-name is recorded in the dictionary but nothing happens until you schedule the task to be executed. A system clock (accessed by the TIME word) is used to do this in real-time. The syntax chosen for scheduling is delightfully natural and helps comprehension of what can be a potentially confusing subject. For example:

EVERY 5 TS FIRST-TASK

would cause MYDEF to be executed every 5 seconds (the available time units are TT (for Task Ticks of 1/50 sec), TS, TM (Task Minutes), TH, TD, TW and TY for hours, days weeks and years. Alternatively one could say:

IN 3 TD FIRST-TASK

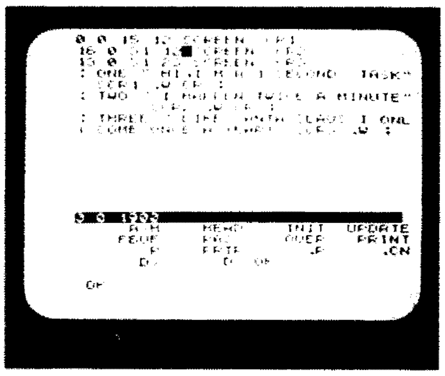
which would run MYDEF in three days time (assuming you leave the computer switched on that is!). AT can be used in a similar way to execute a task AT a certain time; it calculates the time interval from current for you and then passes it to IN. These commands can be combined:

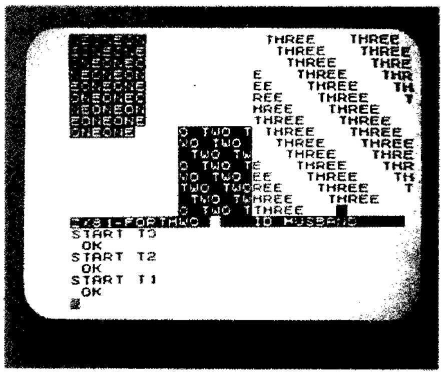
IN 12 TH FIRST-TASK EVERY 30 TS FIRST-TASK

causes MYDEF to start executing in 12 hours and then to do so every 30 seconds. Second, third and so on tasks can be defined and scheduled in a similar way, the limit being memory space, processor oomph and the practicalities of task queueing. Tasks are controlled by a queueing mechanism; if you schedule a task to execute every second which takes two seconds to execute, it will blow the system up in fine style by overflowing the task queue. Tasks can be controlled independently of their scheduling by the words STOP, START and RUN (which runs a named task immediately). A short program can also be made to run continuously in the background, though such a program will bring the system to its knees if it tries to access the keyboard or the console; in practice background programs are best confined to ones which merely schedule higher priority tasks to run. For instance:

```
: DO-IT RUN FIRST-TASK ;
  BACK DO-IT
```

will put the program DO-IT into background so that whenever the system is doing nothing else it will run, causing FIRST-TASK to be executed. If FIRST-TASK takes much more than 1/10th of a second to execute though, the whole system's performance will degrade noticeably.


The editor runs in the top half of the screen while the lower half is reserved for compiling and executing words.



```
START T3
 OK
START T2
 OK
START T1
 OK
```

Three tasks are executing, printing the words "one", "two" and "three" at 1, 2 and 3 second intervals. Meanwhile, the keyboard is still available. . .

There is no mention in the manual of any process for communication between tasks; each task gets its own stack so that route is excluded. David Husband informs me that there are some other words in the tasking set which will only be documented in a forthcoming Technical Manual for advanced users, which will render this possible.

The standard ZX81 has no I/O apart from the printer/RAM expansion bus but it is possible to do some control work via this route, as support for both the Sinclair printer and a standard ASCII serial device are built in. By purchasing one of the several available third party I/O expansions it should be possible to use this machine as a cheap real-time controller; this will probably involve you in some machine code definitions for the driver routines though, as the port addressing words P! and P@ are not provided.

Husband's next project, as reported in SOFT two issues ago, is a similar implementation for the Spectrum which comes on a card with parallel and serial I/O hardware, thus forming the basis of quite a heavyweight real-time machine.

Combining the split screen capability with multiple tasks offers some unusual opportunities for games software, though creating robust programs with user interaction will not be easy. Forth can be a highly strung environment at the best of times and the addition of real-time tasking opens up multiple possibilities for crashing the system, especially as few of us are familiar yet with the concepts involved.

DOCUMENTATION

The manual for *ZX81-Forth* is a 70-page duplicated document which provides all the information that an experienced Forth user would need to fire it up, though it makes no pretence at being a beginner's tutorial. Nevertheless the tone is friendly and unpatronising with no deliberate obfuscation and if used, as it suggests, with a book such as Allan Winfield's *Complete Forth*, it could provide a somewhat unusual introduction to the language. My only complaints about it are the lack of an index, and the non-standard (and occasionally haphazard) stack notation used. Some useful example programs are included, one of which is a concurrent stack display routine. A memory map and a chart of the new key allocations are presented in appendices; a keyboard overlay or stickers would have been even more useful, though at the price I suppose you ought to do it yourself.

CONCLUSIONS

Though all the hounds of hell will never make me like the ZX81, I'm impressed with Husband's achievement in *ZX81-Forth*. It provides a very cheap way to learn what real-time programming is about without the agonies of assembly language. The Forth implementation is surprisingly complete and even rather luxurious in its editing facilities. It isn't hard to imagine this machine controlling a model railway, or a maze-running Mouse or even laboratory instruments, though some extra I/O hardware would be needed. I suspect that the main customers for it will be electronics hobbyists, or programmers who are curious about Forth if it doesn't cost too much. It's a nice illustration of what powerful software can do for a machine that was, let's face it, designed down to a price rather than for high performance. SOFT

Ready converted ZX81 available from Densham Computers Ltd, 329 Ashley Road, Parkstone, Poole, Dorset. Tel: 0202 737493.

SUMMARY

TEST	TIME
Magnifier	**1 sec**
Do-loop	**12 secs**
Literal	**18 secs**
Literal-store	**39 secs**
Variable	**20 secs**
Variable-fetch	**33 secs**
Constant	**18 secs**
Dup	**36 secs**
Increment	**43 secs**
Test>	**48 secs**
Test<	**49 secs**
While-loop*	**84 secs**
Until-loop	**84 secs**
Link	**2 secs**
Arithmetic	**36 secs**

**** Non-standard syntax. For a full listing and explanation of these Forth Benchmarks see Personal Computer World, December 1983.***

CENTRAL ELECTRICITY GENERATING BOARD

TECHNOLOGY, PLANNING & RESEARCH DIVISION
BERKELEY NUCLEAR LABORATORIES,
BERKELEY,
GLOUCESTERSHIRE GL13 9PB.

Telephone: DURSLEY 810451 Telex: 43227

PURCHASE ORDER
No. BN 67625/ NJP
Please quote this number on all invoices and correspondence

DATE 21st June 1984

To:

Skywave Software
73 Curzon Road
BOURNEMOUTH
BS1 4PW

Please consign to:
THE STORES,
CENTRAL ELECTRICITY GENERATING BOARD,
TECHNOLOGY, PLANNING
& RESEARCH DIVISION,
BERKELEY NUCLEAR LABORATORIES,
BERKELEY,
GLOS. Carriage Paid

Please supply the following goods or services in accordance with the conditions detailed overleaf.

Item	Quantity	Description	Commodity Code	Price £
A	1	2 x 81 - forth ROM & Manual		£27.00

1201
26/6/84

SE 13' DELE

Post
29/6/84

NOTE: THE ABOVE PRICE(S) AND/OR RATES ARE EXCLUSIVE OF VALUE ADDED TAX. ANY VALUE ADDED TAX WHICH IS PROPERLY CHARGEABLE WILL BE PAID BY THE BOARD.

SETTLEMENT TERMS	Nett M/A
DELIVERY REQUIREMENTS	19.7.84 or before
YOUR QUOTATION	Current Advertisement

Signed [illegible]
B.E. REES
PURCHASING AND CONTRACTS OFFICER,
for CENTRAL ELECTRICITY GENERATING BOARD

IMPORTANT: INVOICE TO - FINANCE SECTION, C.E.G.B. BERKELEY NUCLEAR LABORATORIES, BERKELEY, GLOS.

IN THE EVENT OF QUERY PLEASE CONTACT:— TECHNICAL :— K P Stook reqnno. B18278
COMMERCIAL:— N J Pitcher