Forthwrite

July
2004
Issue 126
ISSN 0265-5195

Forth Utilities

FIGUK magazine:

Debugging Tools

Certifying your Code

A VNM in Forth: update

Book Review

Forthwrite 126 — July 2004

(inside front cover – blank)

Forthwrite
July
2004
Issue 126
ISSN 0265-5195

Graeme Dunbar [g.r.a.dunbar@rgu.ac.uk]

Editorial

Once again I am very pleased to say we have a good crop of contributed articles from members in this issue. They illustrate what FIG UK and Forthwrite is all about – sharing our ideas and experiences of programming in Forth with fellow enthusiasts. Keep them coming!

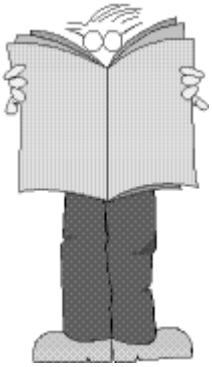
I am sorry to say that after much consideration, Chris Jakeman has decided not to take up the position of Editor again. He reports that he has settled into his new job as a lecturer and although working evenings and weekends on preparing teaching material he is thinking ahead to take on research activities. I'm sure you will all join me in thanking him for his excellent work as Editor and wish him all the best in his new career.

When I stepped in as Editor it was to cover for Chris on a temporary basis. The FIG UK Library is being neglected and needs my attention, so now it is time to look for a new Editor. The Committee has a number of thoughts on the matter and these are reported in this issue (see Forth News), but it is my intention to step down at the next AGM.

Don't forget the monthly IRC session. Our next ones are Saturday 7th August and Saturday 4th September on the IRC server called "IRCNet", channel #FIGUK from 9:00pm BST (that's 2000 UTC for international participants, see: http://www.timeanddate.com/worldclock/).

Until next time, sally Forth,

Graeme Dunbar



Graeme Dunbar [g.r.a.dunbar@rgu.ac.uk]

Forth News

Graeme Dunbar

A roundup of news and events from around the Forth world.

Forth Events

euroFORTH 2004

The dates for this year's euroFORTH conference have been confirmed as the 19th to the 22nd of November 2004, to take place at Castle Dagstuhl in Saarland, Germany. Details can be found on the web site:
http://www.dagstuhl.de/Events/04/

Anton Ertl reports the following:

August 28th
Deadline for draft papers (academic stream)
September 11th
Deadline for abstracts (business/industrial stream)
September 29th
Notification of acceptance of papers
October 26th
Deadline for final camera ready copy
November 19th
Conference opens.

Submission address for the academic stream:

Anton Ertl
[anton@mips.complang.tuwien.ac.at]
Institut für Computersprachen E185/1
Technische Universität Wien
Argentinierstraße 8/4
A-1040 Wien
Austria

Submission instructions can be found at
http://www.complang.tuwien.ac.at/anton/euroforth2003/

FIG UK 25th Anniversary Reunion

FIG UK will be holding its Silver Jubilee this November. Details to follow.

Forth Resources

FIG UK Forth CD

The FIG UK CD has been "Beta tested" by a number of members and as a result some minor changes are being made that will make it easier for new Forth enthusiasts to use.

Forthwrite Editor

Chris Jakeman stepped down as Editor last year and the post has been filled on a temporary basis by the Librarian. I feel I cannot devote as much time as it needs to ensure the high standards of content and production that he set and that my skills and the facilities at my disposal are more suited to running the FIG UK Library than continuing as Editor. Therefore I plan to step down at the next AGM and consequently the Committee is looking for a new Editor or Editorial Team.

As we do not have regular group meetings or exhibitions where members can get together Forthwrite is central to FIG UK's activities. It needs a team of members to produce it.

At present Forthwrite is compiled by the Editor from the excellent contributions by a small group of dedicated members. An electronic copy is emailed to Douglas Neale, the membership secretary, who prints it, oversees the reproduction and then mails it to members. This division of labour works very well and makes best use of local resources.

(continued on page 11)

mpe [mpe@mpeltd.demon.co.uk]

Debugging Tools

Steven Pelc

This article from Stephen Pelc of MPE Ltd has been published almost exactly as it was generated by DocGen in VFX Forth from the original Forth code. An explanation of how this was achieved is given at the end of the article.

Debugging tools

The file *Common\DebugTools.fth* provides debugging tools for MPE embedded systems created by Forth 6 Cross Compilers. The emphasis is on 32 bit systems and interactive testing. The tools can easily be ported to other systems. Copyright is retained by MPE. The code may be freely used on non-MPE systems for non-commercial use.

Porting the code to other systems is up to you. This code may require some carnal knowledge of how your system works. Most Forths contain the required words, but they may not have the same names that MPE use.

Implementation dependencies

In MPE embedded systems, the **USER** variables **IPVEC** and **OPVEC** contain the address of the device structure used for input and output by **KEY**, **EMIT** and friends. In VFX Forth for Windows/Linux, the variables are **IP-HANDLE** and **OP-HANDLE**.

```
: consoleIO     \ --
```

Select debug console for output. By default this is the **CONSOLE** device.

```
  console opvec !  console ipvec !

  Echoing on  Xon/Xoff off

;
```

: name? \ addr -- flag MPE.0000

Check to see if the supplied address is a valid NFA, returning true if the address appears to be a valid NFA. This word is implementation dependent. For MPE cross compilers, a valid NFA for MPE embedded systems satisfies the following:

- All characters within string are printable ASCII within range 33..126
- String Length is non-zero in range 1..31 and bit 7 is set, ignore bits 6, 5

```
  count                                  \ c-addr u --
  dup  $9F and  $81 $9F within? 0=       \ NFA first byte = 1SIxxxxx, count = xxxxx
                                         \ mask           = 10011111
  if  2drop  0  exit  then
  $01F and bounds ?do
    i c@ #33 #126 within? 0=             \ check all ascii chars
    if  unloop  FALSE exit  then
  loop
  TRUE
;
```

: ip>nfa \ addr -- nfa

Attempt to move backwards from an address within a definition to the relevant NFA.

```
  2-                            \ NFA must be at least 'n' bytes backwards
  begin
    dup name? 0=
   while
    1-
  repeat
;
```

: >name \ xt -- nfa

Move from a word's xt to its name field. If **>NAME** does not exist **IP>NFA** will be used.

```
  ip>nfa
;
```

: .name \ nfa –

Given a word's NFA display its name.

```
  count $1F and type
;
```

: .DWORD \ dw –

Display the 32 bit long word 'dw' as an 8 digit hex number.

```
  base @ hex  swap
  0 <# # # # # ascii : hold # # # # #> type
  base !
;
```

Miscellaneous

MPE systems use **`TICKS ( -- ms)`** to return a running time count in milliseconds. Windows systems can use the **GetTickCount** API call.

```
: times          \ n -- ; n TIMES <word>
```

Execute **<word>** n times, and display the execution time. The ticker interrupt must be running.

```
  ticks  '  rot 0                        \ -- ticks xt n 0
  ?do  dup execute  loop
  drop
  ticks swap - . ." ms"
;
```

```
: .ColdChain     \ --
```

Display all words added to the cold chain. Note that the first word added is displayed first. In VFX Forth this word is called ShowColdChain.

```
  cr  ColdChainFirst
  begin
    dup
   while
    dup cell + @ >name .name             \ execute XT
    @                                    \ get next entry
  repeat
  drop
;
```

```
: .decimal       \ n -
```

Display a value as a decimal number.

```
  base @ >r  decimal .  r> base !
;
```

```
: .hex           \ n -
```

Display a value as a hexadecimal number.

```
  base @ >r  hex u.  r> base !
```

```
;
```

: [con \ -- ; R: -- consys

Saves **BASE** and the current i/o vectors on the return stack, and then switches to the console and decimal.

```
   r>
   base @ >r  opvec @ >r  ipvec @ >r
   ConsoleIO decimal
   >r
;
```

: con] \ -- ; R: consys –

Restores **BASE** and the current i/o vectors from the return stack.

```
  r>
  r> ipvec !  r> opvec !  r> base !
  >r
;
```

: CheckFailed \ ip caddr len –

Given the address at the fault occurred and a string, ouput the string and some diagnostic information.

```
  [con
    cr type  ."  failed at "
    dup .dword ."  in " ip>nfa .name
  con]
;
```

Stack checking

Especially in multi-tasked systems, stack errors can be fatal. Detecting them as early as possible reduces debugging time. These words rely on Forth return stack cells containing return addresses. This is true on the vast majority of Forth systems except for some 8051 and real-mode 80x86 systems. If you find others, please let us know.

: ?StackDepth \ +n –

If the stack depth before +n is not n, issue a console warning message and clear the stack. Note that this word is implementation dependent.

```
  dup 2+ depth =
  if  drop  exit  endif                  \ no failure
  [con
  cr ." *** Stack fault: depth = " depth 1- 0 .r ." (d) "
  ." in task " self .task                \ indicate current task
  >r  s0 @ sp!  r> 0 ?do  0  loop        \ set required depth
```

```
  cr ."   Stack updated."
  con]
;
```

: ?StackEmpty   \ --

If the stack depth is non-zero, issue a console warning message and clear the stack.

```
  0 ?StackDepth
;
```

: TaskChecks   \ --

Use in task to check for creeping stacks and so on. This word can be extended to provide additional internal consistency checks.

```
  ?StackEmpty
;
```

: SF{   \ n -- ; R: -- depth

n SF{ .... }SF will check for stack faults. n describes the stack change between SF{ and }SF. If the stack change is different, an error message is generated. This word will work on most systems in which the return address is held on the return stack.

```
  r>  swap depth 2- + >r  >r
;
```

: }SF   \ -- ; R: depth -- ; perform stack check

The end of an SF{ ... }SF structure. This word is not strictly portable as it assumes that the Forth return stack holds a valid return address. In the vast majority of cases the assumption is true, but beware of some 8051 implementations. See SF{

```
  r>
  r> depth 2- <> if
    dup s" Stack check" CheckFailed
  endif
  >r
;
```

Assertions

Assertions are a useful way to check that the system is behaving correctly. When the phrase:

```
  [ASSERT <test> ASSERT]
```

is compiled into a piece of code, the test is performed and generates an error report if the result is false. If you do not want the performance overhead of the test, set the value **ASSERTS?** to zero. To remove even the small overhead of of testing **ASSERTS?**, comment out the line.

-1 value assert?   \ -- n

Returns non-zero if asserts will be tested.

```
: (assert)      \ flag -
```

If flag is zero, report an ASSERT error.

```
  if  exit  endif                        \ faster on some CPUs
  r@ s" ASSERT" CheckFailed
;
```

```
: [assert       \ --
```

Compile the code to start an assert.

```
  ?comp
  postpone assert?  postpone if
; immediate
```

```
: assert]       \ --
```

Compile the code to end an assert.

```
  ?comp
  postpone (assert)  postpone then
; immediate
```

Here is a simple assert that will fail if **BASE** is not **DECIMAL**.

```
: foo     \ --
  [assert  base @ #10 =  assert]
;
```

Editing the Article

The text above has been presented almost exactly as it appeared in the html file supplied by the author. It has simply been cut and pasted with the inter-paragraph spacing then being adjusted to make it fit the A5 format. Shown over the page are extracts from the original DEBUGTOOLS.FTH source code file to illustrate the various formatting tags used.

Stephen Pelc added the following explanation of how the process is carried out in the following email exchange. – *Editor.*

```
> Many thanks for the article. I can load the html into Word and
> preserve most of the formatting. Is the formatting done automatically
> by DocGen?

Yes. The file DEBUGTOOLS.FTH contains both code and the
documentation. Lines of the form
\ *X .......
are processed by DocGen, which automatically generates the HTML.
```

```
If you have the free evaluation version of VFX Forth for Windows
from the web site, you can try it yourself:

+docgen docgen_html
doconly debugtools
-docgen

I'll be putting up the v3.7 version that generated the HTML in a
few days.

> How does the code appear on the screen as it is being edited?
It's in DEBUGTOOLS.FTH so it depends how you set up your editor. The
whole point is that that the documentation is generated
*directly* from the source code.
```

The free evaluation version of VFX Forth for Windows can be downloaded from the MPE web site at: http://www.mpeltd.demon.co.uk/arena.htm

Original Source Code

```
\ DEBUGTOOLS.FTH - debug tools for XC6.2

\ =============
\ *! debugtools
\ *T Debugging tools
\ =============

\ *E Copyright (c) 1996-2004
\ ** MicroProcessor Engineering
\ ** 133 Hill Lane
\ ** Southampton SO15 5AF
\ ** England
\ **
\ ** tel: +44 (0)23 8063 1441
\ ** fax: +44 (0)23 8033 9691
\ ** net: mpe@mpeltd.demon.co.uk
     tech-support@mpeltd.demon.co.uk
\ ** web: www.mpeltd.demon.co.uk

((
To do
=====

Change history
==============
20040614 MPE001 Overhaul and used to test DocGen from VFX Forth 3.7
))

decimal

\ *P The file *\i{Common\DebugTools.fth} provides debugging tools
\ ** for MPE embedded systems created by Forth 6 Cross Compilers.
\ ** The emphasis is on 32 bit systems and interactive testing.
\ ** The tools can easily be ported to other systems. Copyright is
\ ** retained by MPE. The code may be freely used on non-MPE systems
\ ** for non-commercial use.

\ *P Porting the code to other systems is up to you. This code
\ ** may require some carnal knowledge of how your system works.
```

```
\ ** Most Forths contain the required words, but they may not
\ ** have the same names that MPE use.


\ ****************************
\ *S Implementation dependencies
\ ****************************
\ *P In MPE embedded systems, the *\fo{USER} variables *\fo{IPVEC}
\ ** and *\fo{OPVEC} contain the address of the device structure
\ ** used for input and output by *\fo{KEY}, *\fo{EMIT} and friends.
\ ** In VFX Forth for Windows/Linux, the variables are *\fo{IP-HANDLE}
\ ** and *\fo{OP-HANDLE}.

[undefined] consoleIO [if]
: consoleIO \ --
\ *G Select debug console for output. By default this
\ ** is the *\fo{CONSOLE} device.
\ *[
  console opvec !  console ipvec !
  Echoing on  Xon/Xoff off
;
```

(middle section deleted)

```
compiler
: [assert   assert? if  ;
: assert]   (assert)  then  ;
target

\ *P Here is a simple assert that will fail if *\fo{BASE} is not
\ ** *\fo{DECIMAL}.
\ *E : foo     \ --
\ **   [assert  base @ #10 =  assert]
\ ** ;


\ ======
\ *> ###
\ ======
```

(Forth News - continued from page 3)

You will recall that Chris Jakeman not only edited the newsletter also made a considerable contribution as author and reporter. I do not have the time or the Forth skills to emulate him and neither, I suspect, do many other members.

The committee have mulled over the problem and have come up with a number of ideas.

Firstly, the Editorship might be taken on by someone who has retired from full time employment. I am assured that an active retirement does not bring with it abundant spare time, but it does perhaps give the individual a little more control over how that time is allocated. Other suggestions involve sharing the load between several individuals in various ways. Part of this might involve a “Forth News” Reporter, and a “From the Net” Correspondent as advertised in the last issue. Further splits between information gathering and compositing may be appropriate.

What is clear is that it is not the typing and formatting that consumes the time but seeking out articles and authors to publish in these pages and following them up.

If you are interested in taking on the post, or contributing to production please contact the present editor.

Paul E. Bennett [peb@amleth.demon.co.uk]

Certifying your Code

Paul E. Bennett [HIDECS Consultancy]

Thorough testing and verification of code is often neglected. Paul Bennett introduces us to concepts and procedures that we can all use and concludes that Forth can be used in most safety critical situations.

It has been a long held belief, within the software industry generally, that the only programmes that are capable of being 100% tested are "Toy" programmes. I am presuming that this is meant to indicated programmes of under a couple of hundred LOC. Programmes of such a small size would be quite within the realms of any programmer (in whatever language) to fully test in reasonable time. However, let us look at what Forth gives us that is different to many programming environments.

High Integrity Systems, we are assured, have been thoroughly tested. This leaves us all with the unanswered question of "How Thorough is thorough?" Considering that much of the High Integrity software that keep our planes in the sky, run our traffic lights and keep us alive on the operating theatre table are usually many thousands of lines of code can we trust that the testing has been done well enough to give us confidence. What the hardware sector has that the software sector would probably like to have is the ability to attach a "Certificate of Conformity" that can present tangible proof that the system has been developed and tested in accordance with some auditable standard method.

The software tools industry always seem to be promising that the new whizzo software tool is the "Silver Bullet" we have been seeking. Development suites like **Rhapsody** (from Ilogix) or **SPARK-Ada** have been touted as the most logical choice for Safety Critical Systems Development because they are proven correct mathematically. Not only are these tools very expensive they also seem to give an unwelcome boost the resources requirements on the embedded system.

In a recent thread on *comp.lang.forth Pete* < *forthsafe@yahoo.com* > asked whether or not Forth could be used in Safety Critical Systems and pass the stringent requirements of FDA or DO178B. As one who has been involved in the development and certification of systems that

have had to pass such scrutiny I am happy to say that Forth, produced in the right way, can indeed be used for such projects.

In Forth, each word can be considered as a complete programme in its own right. It may be just a subroutine of a much larger programme but in itself it is quite complete and, at the interpreted level, is interactively available for quite thorough testing. Any Forth word can be considered as a fully single minded simple programme in itself. The ultimate "Toy" programme.

Because Forth words, written properly (preferably with reference to a coding standard), are usually quite small and simple we are merely coupling a number of individual, complete, simple programmes.

Each word constructed using other previously tested words would inherit the properties of the testing already carried out. The words in a specific lexicological grouping can be seen as existing between programming surfaces that provide the API between functional entities. In this way, the most complex Forth programmes can carry through the testing at a level that is more or less trivially accomplished.

The idea of programming surfaces establishes a useful reference for situations where application code is developed away from the real target. So long as the certification can fully prove equivalence there should remain no issues (bar hardware timing) that would invalidate the certification of the software.

It was realising this simple facet of Forth that gave me confidence enough to develop the review and testing process to a method for fully certifying Forth code. The method also works with assembly, so long as the assembly subroutines are small, single function, highly coherent and utilise minimal coupling of the systems the code is being developed for.

It should be obvious that the development process by which High Integrity Software is developed for these Safety Critical Systems should be rigorous, and rigorously applied. Development Management processes that fall below CMMR level 3 are less likely to be trusted well enough for software certification to be believable. Ideally, the organisation should be at level 4 or 5. Therefore, establishing a decent development process is a necessary pre-requisite to being able to produce certifiable software. It is an absolute necessity that the exact source code that is certified is always uniquely identifiable (name, version, production date, test dates etc). A version control and configuration management system, therefore, becomes essential.

Within this article I have included an arbitrary bit of code which should illustrate how the certification process not only improves the presentation of the code but also ensures its logically correct implementation. The form format is made part of the source code file along with the words glossary description and any additional notes. The example code is simple enough to follow without the addition of the glossary comments. However, the incorporation of the glossary text within the source files aids fuller understanding without having to reference other sources.

```
\ ************************************************************
: .TOS ( S: n1 -- n1 )
( G: Non-destructively print the top of parameter stack item )
( n1 to the current terminal. )
DUP . ;
\ ************************************************************
\ *   Inspection     *  Function Test    *    Limits Test      *
\ *                  *                   *                     *
\ *                  *                   *                     *
\ *                  *                   *                     *
\ ************************************************************


\ ************************************************************
: FIB-ALG ( S: n1\n2 -- n2\n3 )
( G: Given two numbers, n1 and n2, of a Fibonacci series )
( calculate the next number, n3, of the series. )
SWAP OVER + ;

\ ************************************************************
\ *   Inspection     *  Function Test    *    Limits Test      *
\ *                  *                   *                     *
\ *                  *                   *                     *
\ *                  *                   *                     *
\ ************************************************************


\ ************************************************************
: FIBONACCI ( S: n1\n2\limit -- )
( G: Given the two numbers, n1 and n2, of a Fibonacci series )
( calculate and print the succeeding numbers in the series )
( up to limit to the current terminal. )
>R
BEGIN DUP R@ > NOT
WHILE .TOS FIB-ALG
REPEAT 2DROP R> DROP
\ ************************************************************
\ *   Inspection     *  Function Test    *    Limits Test      *
\ *                  *                   *                     *
\ *                  *                   *                     *
\ *                  *                   *                     *
\ ************************************************************
```

The three boxes underneath the word definition are intended as an appropriate space in which to collect signatures from the code inspection and test personnel. The three boxes deal with a specific aspect of the inspection and tests to be applied. You can choose whether or not the three boxes are a permanent feature within the source file or generated during documentation print-out. You may even discover a way, within your own configuration management system, to incorporate electronic signatures within these boxes.

Inspection

This is the traditional static inspection run along Fagan Inspection lines. The code inspector should ensure that the code appears to fully implement the requirements stated within the glossary text that is part of the source. This requires that the inspector will need to ensure that the called words used within this definition are also previously certified. Previous reviews of the glossary text should have, of course, ensured that the intent for the word under examination as defined in the glossary text is right for the application. This is one of the benefits of writing the glossary text first.

Function Test

The function test is the first time that the code is formally run. The code might have been exercised by the programmer immediately after coding but this formal run records the performance of the word in comparison to the intent expressed in the glossary text. Several occurrences of the function test would be usual, especially with different representative values, to provide a level of confidence that the function always does as is expected. This test only covers the normal operating range.

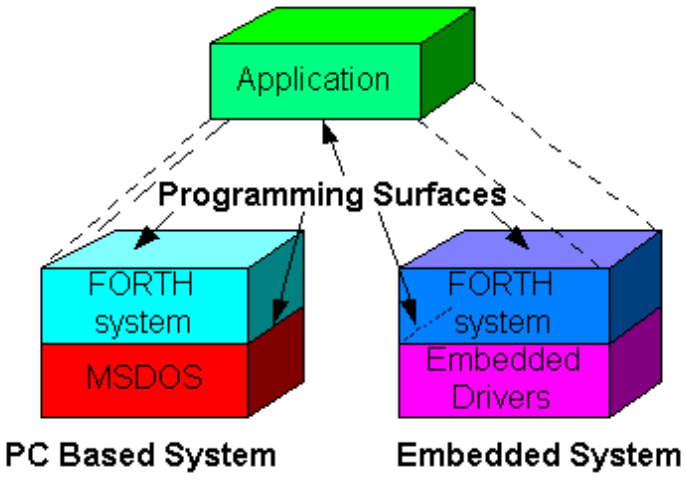
Limits Test

The Limits test is an opportunity to test the code to beyond the expected normal range of operation. If the code is able to deal with wildly out of bounds input stimuli, high rate call demand (if the code is an interrupt) and still behave in a rational manner then this test can be deemed passed. In order to perform this test on your code often requires a very inventive frame of mind to ensure that the operation is, as far as is practicable, out of the normal operational range.

With the code having satisfied the test and inspection personnel, their signatures can be attached to the source print-out. This signed listing then needs storing in a long term archive until the product in which the software exists is no longer in service and an additional five years following its decommissioning.

Programming Surfaces

I have used the term **programming surfaces** in this article and perhaps now is a good time to explain what they are.



The best description of programming surfaces is perhaps as an interface between underlying machine level code and the application language layer or between the application language layer and the application itself. This idea of surfaces is very useful for demarcation of the various elements of a system which, if the surfaces can be proven equivalent, allow easy movement of segments of

code from one platform to another. This is most useful when the underlying processor platform goes out of production and you need to replace it with another similar item of hardware. So long as the machine-level to application language level interface surfaces are proven equivalent there should be very little problem in making the substitution. This also works on a finer grained level throughout Forth.

Programming surfaces are part of a component oriented approach to system development. Components are a firmer, harder edged form of objects that can be comprehended, tested and proven of sufficiently good quality.

Summary

This technique to certification is very simple to apply and, coupled with a suitably robust version and change management process that tracks modifications of the source code, can lead to the production of fully certifiable code that satisfies the most stringent requirements (DO178B, FDA, CE marking etc.).

Just creating source code that incorporates the inspection boxes is not enough. The code has got to qualify by proper inspection and testing methods being applied. Coverage of testing can easily approach 100% of all statements.

Application of a certification process against coding standards and language specification standards will improve the quality factors for the software. This improvement is mainly won through an orderly increase in the level of inspection that the code undergoes. It is also due to an improvement in monitored testing to beyond mere functional parameters.

With this simple certification process in mind I was able to confirm that Forth could indeed be used in the most safety critical projects where a software based solution was a viable choice.

Paul E. Bennett

Paul E. Bennett is an Independent Consultant currently working in the Nuclear Power Industry. He has had experience in many Safety Critical sectors including Petrochemical, Marine, Railway, Medical and Transportation Industries. He has been an advocate of the use Forth in High Integrity Distributed Embedded Control Systems. He has written articles on many topics in Forthwrite including real-world interfacing (see http://www.amleth.demon.co.uk).

Letters

Letters to the Editor are always welcome. Please send them the old-fashioned way or by email – which ever is most convenient.

James Boyd has been delving into Win32Forth…

Dear Graeme,

I have noticed that Win32Forth uses compiler security to prevent mismatched control structures. There is no compiler security to prevent defining locals within the scope of a control structure (which the standard forbids). To add compiler security to locals in Win32Forth requires only two simple redefinitions:

```
: locals|   ?csp  postpone locals| ; immediate
: {         ?csp  postpone { ; immediate
```

I have included a Forth source file for even more locals security for those who want it. The code prevents defining locals with data stored on the return stack with >r and\or 2>r. It also prevents interpreting >r r> 2>r 2r> from the console and the resulting exception. Maybe the attachment is overly cautious but, a newbie would sure appreciate it.

Regards,
Jim

```
\ Safer way to implement locals on Win32Forth
\ James Boyd    April 24th, 2004 - 6:29

variable balance

: :    ( -<name>- )   balance off : ;
: >r   ?comp 1 balance +!  postpone >r ; immediate

: 2>r   ?comp 2 balance +!  postpone 2>r ; immediate

: r>   ?comp -1 balance +!  postpone r> ; immediate

: 2r>   ?comp -2 balance +! postpone 2r> ; immediate

: ?balance    ( -- )   balance @ abort" items on return stack" ;

: locals|   ?csp  ?balance  postpone locals| ; immediate
: {         ?csp  ?balance  postpone { ; immediate

comment:

: BadTest1   >R  locals| a b |  r> ;

: BadTest2   if   locals| a b |   then ;

: GoodTest   locals| a b |   cr a . b . 3 >r . r> ;

see GoodTest

comment;
```

James Boyd [JimBoyd@techemail.com]

A VNM in Forth: update

James A. Boyd

Following his two part article on Virtual Nondeterministic Machines in Forth, James Boyd presents a modification to deal with locals.

Here is an update on the Virtual Nondeterministic Machine (VNM). As defined, locals and nondeterministic operations (choice and failure anyway) can not be in the same word. Also, nondeterministic operations can not be in a word called by a word with locals. Here are some modifications to fix that minor problem.

In Win32Forth the locals are saved on the return stack and referenced by a local pointer therefore, saving and restoring the local pointer by redefining:

```
: SaveData      ( -- )   SaveDataStack  SaveOther ;

: RestoreData   ( -- )   RestoreOther  RestoreDataStack ;
```

as:

```
: SaveData      ( -- )   SaveDataStack   LP @ >history   SaveOther ;

: RestoreData   ( -- )   RestoreOther   history> LP !   RestoreDataStack ;
```

will allow words with locals to contain nondeterministic operations.

In Gforth it is a little trickier (okay, a lot trickier). I can not guarantee the following code even though I tested it briefly. I wish I could do better, but I am not that familiar with Gforth. Maybe someone familiar with Gforth can improve upon this.

In Gforth the locals are maintained on a separate locals stack therefore the following definitions can be added:

```
: SaveLocal   ( -- )

        lp@  [ lp@ ] literal  over - tuck  ( # adr # )

        m>history  >history ;

: RestoreLocal   ( -- )

        history> [ lp@ ] literal  over -  lp!  ( # )

        lp@ swap mhistory>  ( -- ) ;
```

and SaveData and RestoreData can be redefined as:

```
: SaveData   ( -- )   SaveDataStack  SaveLocal  SaveOther ;

: RestoreData   ( -- )   RestoreOther  RestoreLocal  RestoreDataStack ;
```

to allow nondeterministic operations. I tested this with some nondeterministic code with locals and it seems to work.

I realize that compiling the address of the local stack as a literal is a kludge, but I could not find the base address of the local stack in Gforth. The VNM code to save and restore the locals does not actually use locals (nor does any code for the VNM). Using a fixed address instead of an address from a user variable will make it even harder to integrate multi-tasking within the Forth system with the VNM (The VNM is of course unaffected by the host OS multitasking as long as sufficient resources are available). Currently, the only multi-tasking possible with the VNM is if the background tasks do not contain any nondeterministic operations. I do not, at present, see any use for background tasks with nondeterministic operations, so I do not consider this much of a limitation.

I wrote some test code to make sure that if the Gforth system used locals that the local stack depth would be the same whether compiling or interpreting (when not using locals in the new definition) by executing test3 from the keyboard:

```
: test1   ( -- )   [ lp@ ] literal ;    \ compile value of lp@ as literal

: test2   ( -- )   lp@ ;                \ lp@ returns present value at
runtime

: test3   ( -- )   cr  test1 test2 =    \ is local stack depth same for
both?

        if     ." Same"

        else   ." different"

        then ;
```

and the result was that they are the same. I also verified that the local stack in Gforth grows toward lower memory.

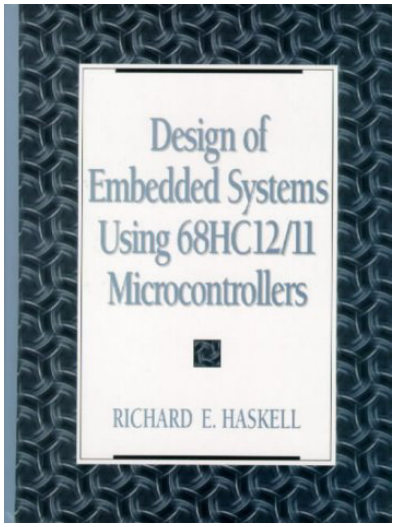
Errata

Regions: into unknown Win32Forth, by David R. Pochin

In Forthwrite Issue 125 (May 2004) the author's name was misspelled. The Editor expresses his apologies to Dave for this.

The electronic copy of Forthwrite which will be posted on the web in due course will show the correct spelling.

Paul E. Bennett [peb@amleth.demon.co.uk]



Book Review

"Design of embedded systems using 68HC12/11 microcontrollers"

by Richard E. Haskell.

Reviewer: Boris Fennema.

This book does what it says in the title. It comes with a diskette but there are updates and further information available on Richard Haskell's web site, http://www.cse.secs.oakland.edu/haskell/ . There are errata in the diskette as shipped with the book so it is important to download the new images from the author's web site if you consider using it on a target board.

The book can be used as a reference for the 68HC11, but its main focus is on the 'HC12. However, it does cover both in detail and in the appendices.

It starts with a brief outline in which it introduces the Motorola 68HC11 and 68HC12, including a description of the hardware in block diagram form with peripherals and schematics, followed by the register model, flags, addressing modes and the opcode model. All this information is also available as an Appendix in datasheet format.

It then shows how you would write code for the microcontroller using Motorola tools such as AS12 and D-BUG12. The first program adds and subtracts two literals producing the sum and difference - this simple example is then used to explore the weaknesses in this approach (hardcoded values, hardcoded result locations and a one-shot program). These flaws are shown to be remedied by introducing a data stack for argument passing and a means of calling standard sub-routines ('words') that act on that data. This is then used to introduce WHYP (Words that Help You Program) which is an umbilical Forth language.

The design of WHYP is discussed from the ground up. For example it is explained that the host program which is written in C++ communicates with the HC11/HC12 board via the serial port and that there is a small kernel running on the board that looks like (simplified no handshaking):

```
LOOP   BSR INWDY
       JSR 0,Y
       BRA LOOP
```

This means that the development board sits in an infinite loop. It starts by branching to a routine 'INWDY' - this reads 2 bytes from the serial link and stores it in the Y register. The next line then jumps to this address and upon return branches back to the start of the loop.

The address to execute can reside in EEPROM or in RAM. If desired WHYP can be expanded by adding the words to the image and loaded into EEPROM.

The PC side of WHYP (PC-WHYP) is responsible for maintaining the correlation between addresses on the target board and their names (dictionary). It also carries out screen display and keyboard I/O on behalf of the target board. For example, if the target board executes the word "." (dot) it will execute the location in RAM or EEPROM where "." resides as directed by the dictionary and will send the command "display_integer" followed by the value at the top of the stack back to the PC. PC-WHYP reads the opcode of the serial line and will know how to interpret the next 2 bytes and display the integer on the screen. In the next chapters additional words are added to WHYP. It may be of interest to know that WHYP can be ported to various development boards from Motorola and Axiom as described in the book (F12UK would be interesting project).

On other microcontrollers the concepts would be "portable" such as a possible means of implementing an umbilical Forth. Along the way the examples vary from purely software tasks to interfacing to various sensors, LED's and LCD screens to key pads etc. These explorations are quite detailed for instance when keypads are discussed key debouncing in software is explained in detail.

Other topics of interest are: interrupts, parallel interfacing, the serial-peripheral interface (SPI), A/D converters, timers, the serial communications interface (SCI), fuzzy logic (the HC12 has fuzzy logic hardware) and some topics that are limited to some HC12 parts that have these features available : pulse-width modulation, key wakeup and programming of flash EEPROM.

In other chapters WHYP is extended with among others flow control and branching words, string/number conversions, and WHYP interrupt routines. (You can also write in assembler routines and have them called from WHYP by informing the dictionary where they reside in the kernel image).

Throughout the book, all is explained in great detail and in plain language with important points emphasised. The mixture of assembly language and WHYP/Forth works really well in that it gives the reader good exposure to both without requiring previous knowledge other than the basics.

Title: Design of embedded systems using 68HC12/11 microcontrollers
Author: Richard E. Haskell
Publisher: Prentice Hall, 2000
ISBN: 0-13-083208-01
Hardback: 569 pages
Price: £68.99 (www.amazon.co.uk current price).

Henry Vinerts [Volvovid@aol.com]

Across the Big Teich

Henry Vinerts

This material was prepared for Vierte Dimension by Henry Vinerts, and printed by kind permission of Forth Gesellschaft (German FIG).

Hello, Friederich, Fred, and Graeme!

Silicon Valley FIG Meeting – April / May / June 2004

I have little to report for the past three months. Instead of a formal meeting in April, seven of us traveled to Bruce Damer's Ancient Oak Farm in the Santa Cruz mountains and enjoyed the host's personally guided tour of his "Digibarn" (http://www.digibarn.com) collection of vintage computers. I cannot write fast enough to recount all the fascinating impressions from this visit, but you could get an idea from the website, which, Bruce says, gets millions of visitors per year. The May meeting conflicted with some sight-seeing trips that my wife and I took with our visitors from the Netherlands, and in June, instead of a SVFIG meeting, those of us who are members of the Computer History Museum (and I am not) are invited to attend an event titled "Then and Now: Computer Graphics in Games," featuring Jordan Mechner, Rand Miller, and Will Wright with Vince Broady. (More at http://www.computerhistory.org/nvidia_06102004/)

Such are the "signs of life" among the Silicon Valley forthers. I may not have anything to report until after the end of July, when regular meetings are expected to start again.

I have been thinking that if I / we cannot find enough time or new material to write about to enlighten our readers, perhaps it is time to dig into the archives and collect some pearls of wisdom from the many oysters that the Forth community has produced. Someone has said that the best compliment to an author is a quotation from his works. What do you say if we send our readers on a search for Forth quotes, print the best in the magazines under a ." (dot-quote) rubric, and when enough have been collected, use them as an introduction to a book, which I would hope would be titled something like "Understanding Forth" and not "Ending Forth"? Can anyone tell me how long it has been since any of the big bookstores has had a Forth book on their shelves?

Let me start with one of the most famous quotes from Chuck Moore (in the foreword to "Starting Forth"):

> *"Simplicity provides confidence, reliability, compactness, and speed."*

Wolfgang Allinger has already aptly translated it into German:

> *"Einfachheit erzeugt Vertrauen, Zuverlaessigkeit, Ueberschaubarkeit und Schnelligkeit."*

For those who may have read something interesting, but no longer have the book, I can at least provide the attached list of the books that I own. Who knows, there may still be a good number of lost treasures buried under the toils of all those authors?

Wishing you all a good summer,

Henry

Attachment:

List of Henry's Forth books (\mydocuments\my eBooks\4thbooks.doc, June 2004)

1. "Starting FORTH" by Leo Brodie, 1981, Forth Inc.
2. "Starting Forth" 2nd ed., by Leo Brodie, 1987, Prentice-Hall.
3. "Dr. Dobb's Toolbook of Forth" Vol. I, 1987, M&T Books.
4. "Dr. Dobb's Toolbook of Forth" Vol.II, 1987, M&T Books.
5. "All About Forth" MVP-FORTH Series Vol.1, by Glen B. Haydon, 2nd ed., 1984, Mountain View Press.
6. "All About Forth--An Annotated Glossary," by Glen B. Haydon, 3rd ed., 1990, MVP-FORTH SERIES.
7. "Real Time Forth" by Tim Hendtlass, 1993, Swinburne Univ. of Technology, Australia.
8. "Thinking Forth" by Leo Brodie, 1984, Prentice-Hall (proofreading copy for 1994 edition).
9. "Using FORTH" by Rather, Brodie, Rosenberg, 2nd ed., 1980, Forth, Inc.
10. "Mastering FORTH" by Tracy & Anderson, Advanced MicroMotion, Inc., 1989, Brady Books.
11. "FORTH: A Text and Reference" by Kelly & Spies, 1986, Prentice-Hall.
12. "Introduction to FORTH" by Ken Knecht, 1982, Howard W. Sams.
13. "The Complete FORTH" by Alan Winfield, 1983, Sigma/Wiley.

14. "FORTH-79" ver. 2, Z-80 CP/M edition, by Anderson & Tracy, 1982, MicroMotion.
15. "Discover FORTH" by Thom Hogan, 1982, Osborne/McGraw-Hill.
16. "Learning FORTH--A Self-teaching Guide" by Margaret Armstrong, 1985, John Wiley.
17. "Threaded Interpretive Languages" by R.G. Loeliger, 1981, Byte Books.

As always, Henry shows his unique insight into the language and philosophy of Forth. What books do other members have by their computer / on the bedside table for reference, instruction or inspiration? Send in your "dot-quotes" to the editor via the usual channels.

In a recent email Henry expressed his delight on receiving his FIG UK Achievement Award, "I feel as if I had just been knighted by the Queen. Thank you all who contributed to such a delightful surprise for me". Don't let it go to your head and forget us commoners, Henry! We look forward to reading your next report from Silicon Valley.

– *Editor*

What Languages Fix

Back in Issue 122 an observation on programming languages was taken from the web page of Paul Graham, author and designer of the Arc language (http://www.paulgraham.com/fix.html) and the question "what problem does Forth fix?" was posed.

Here are some of the contributions from readers:

From James Boyd:

- Forth doesn't "tie my hands".

Chris Jakeman found the following on the 'net:

- Forth: assembly language should be interactive and extensible. – John Passaniti
- Forth is postfix – Al Todd
- Forth: assembly syntax is scary. – Michael Gassanenko
- Forth: FORTRAN is not interactive, and Chuck's pre-Forth interpreter was not extensible. – Bernd Paysan
- Forth: OS built on files are too big and slow for a decent database. – Jeff Fox
- Forth: all that scary stuff is not needed. – Michael Gassanenko
- Forth: fixes the megasoftware problem. – Charles Moore.

Dutch Forth Users Group

Reading Dutch is easier than you might think. And as Forth is an international language, reading Dutch code is easier still for a Forth enthusiast. Are you interested? Why not subscribe to

HCC-Forth-gebruikersgroep

For only 10 euros a year (about £6.70), we will send you 5 to 6 copies of our "fig-leaf" broadsheet 'Het Vijgeblaadje' . This includes all our activities, progress reports on software and hardware projects and news of our in-house products.

To join, contact our Chairman:

Willem Ouwerkerk
Boulevard Heuvelink 126
6828 KW Arnhem, The Netherlands
E-Mail: w.ouwerkerk@kader.hobby.nl

The easiest way to pay is to post a 10 euro note direct to Willem.

Joe Anderson [jia@jia.abel.co.uk]
phone: 0131 662 4007

Vierte Dimension 1/2004

Joe Anderson

Joe provides a look at the latest issue of the German FIG magazine.

Editorial. 4

Friederich Prinz [vd@forth-ev.de]

Friederich draws attention to the current Jubilee celebration (20 years of the Forth-Gesellschaft). He recalls the beginnings from his own life: NAXOS (a Forth compiler programmed in Turbo-Pascal), TCOM (under ZF and F-PC) from Tom Zimmer and HOLON from Wolf Wejgaard. He admits that even today he likes working in the venerable and trusted DOS environment, which also allows him easy access to Lego-RCX.

Readers' Letters. 5

Readers' letters from: Michael Kalus (Celebration committee, Conference on Fehmarn), Rafael Deliano (working with graphics), Carsten Strotmann (KNOPPIX-Linux-Forth CD), Ulrich Paul (Operator overloading in Forth too), Ulrich Paul (the future of Forth), Wolf Wejgaard (HOLON86).

Book Review. 7

Friederich Prinz

Friederich Prinz reports on the book "Extreme Mindstorms" by Baum, Gasperi, Hempel, Villa: pbForth, NQC (Not Quite C), LegOS and many tips for DIY Lego-RCX.

Ushi-Tag in Maarssen. 8

Friederich Prinz

Ushi is a DIY robot from the Dutch Forth fans, programmable in AVR-ByteForth. Martin Bitter and Friederich Prinz were guests at the robot gathering in the neighbourhood of Utrecht.

A Solution to the "Fours" Problem, programmed in Forth. 9

Ewald Reiger [Ewald.Rieger@t-online.de]

The numbers from 1 to 50 are to be produced using nothing but the numeral 4 and a few basic mathematical functions. A few further questions were posed. The problem was sent via e-mail by Henry Vinerts to Fred Behringer. Ewald solves the problem in elegant Forth style through complete enumeration in a tree structure.

Advertisements 13

Notices regarding membership adverts for FIG UK and the Dutch Forth-gebruikersgroep.

FIG UK Contacts and Information

Chairman	**Jeremy Fowell,**	11 Hitches Lane, EDGEBASTON B15 2LS 0121 440 1809 jeremy.fowell@btinternet.com
Secretary	**Douglas Neale,**	58 Woodland Way, MORDEN SM4 4DS 020 8542 2747 dneale@w58wmorden.demon.co.uk
Editor **(temporary)**	**Graeme Dunbar,**	School of Engineering, The Robert Gordon University, Schoolhill, ABERDEEN AB10 1FR 01224 262415 g.r.a.dunbar@rgu.ac.uk
Treasurer	**Neville Joseph,**	Marlowe House, Hale Road, WENDOVER HP22 6NE 01296 62 3167 naj@najoseph.demon.co.uk
Webmaster	**Jenny Brien,**	Windy Hill, Drumkeen, BALLINAMALLARD, Co. Fermanagh BT94 2HJ 02866 388 253 webmaster@figuk.plus.com
Librarian	**Graeme Dunbar,**	School of Engineering, The Robert Gordon University, Schoolhill, ABERDEEN AB10 1FR 01224 262415 g.r.a.dunbar@rgu.ac.uk

Membership enquiries, renewals and changes of address to Douglas.
Technical enquiries and anything for publication to Graeme.
Borrowing requests for books, magazines and proceedings to Graeme.

FIG UK Web Site

For indexes to Forthwrite, the FIG UK Library and much more, see http://www.fig-uk.org

FIG UK Membership

Payment entitles you to 6 issues of Forthwrite magazine and our membership services for that period (about a year). Fees are:

National and international	£12
International served by airmail	£22
Corporate	£36 (3 copies of each issue)

Forthwrite Deliveries

Your membership number appears on your envelope label. Please quote it in correspondence to us. Look out for the message "SUBS NOW DUE" on your sixth and last issue and please complete the renewal form enclosed. Overseas members can opt to pay the higher price for airmail delivery.

Copyright

FIG UK Services to Members

Magazine Forthwrite is our regular magazine, which has been in publication for over 100 issues. Most of the contributions come from our own members and Graeme Dunbar, the Editor, is always ready to assist new authors wishing to share their experiences of the Forth world.

Library Our library provides a service unmatched by any other FIG chapter. Not only are all the major books available, but also conference proceedings, back-issues of Forthwrite and also of the magazine of International FIG, Forth Dimensions. The price of a loan is simply the cost of postage out and back.

Web Site Jenny Brien maintains our web site at http://www.fig-uk.org. She publishes details of FIG UK projects, a regularly-updated Forth News report, indexes to the Forthwrite magazine and the library as well as specialist contributions such as “Build Your Own Forth” and links to other sites. Don’t forget to check out the “FIG UK Hall of Fame”.

IRC Software for accessing Internet Relay Chat is free and easy to use. FIG UK members (and a few others too) get together on the #FIG UK channel every month. Check Forthwrite for details.

Members The members are our greatest asset. If you have a problem, don’t struggle in silence - someone will always be able to help. Do consider joining one of our joint projects. Undertaken by informal groups of members, these are very successful and an excellent way to gain both experience and good friends.

Beyond the UK FIG UK has links with International FIG, the German Forth-Gesellschaft and the Dutch Forth Users Group. Some of our members have multiple memberships and we report progress and special events. FIG UK has attracted a core of overseas members; please ask if you want an accelerated postal delivery for your Forthwrite.

Copy deadlines:

Issue 127: 18 August 2004

Issue 128: 27 October 2004

All material for publication to the Editor by email or post by that date please. Plain text, MS Word or Rich Text format preferred.

Back cover (Advert)