MICROTAN 65

FORTH-R

USER MANUAL

Contents

Introduction

FORTH-R is a modern port of fig-FORTH for the Microtan 65. It is an alternative to the three versions of FORTH which were available for the Microtan in the 1980s, Tansoft's TAN-FORTH, Tangerine User Group (TUG) FORTH (hereafter referred to as TUG-FORTH) and the EPROM based Microtanic FORTH. Whilst some versions of these packages survive, not all versions are available and the source code for them is not available making maintenance impossible. FORTH-R is primarily based on the fig-FORTH source code but borrows heavily from TAN-FORTH for the extensions such as the Editor, Assembler and Graphics support.

The FORTH Language

FORTH was created in 1969 by Charles H. Moore whilst working at the United States National Radio Astronomy Observatory. The Forth Interest Group (FIG) was a world-wide, non-profit organization for education in and the promotion of the Forth computer language. It published versions of FORTH for a range of computers and associated documentation.

FORTH is in many ways an obvious choice of language for the Microtan, it is faster and more powerful than BASIC, and also supports more formal structuring. It is small enough to work within about 16K of memory and being interpreted requires no additional development tools. There is a significant amount of FORTH related content in the later Microtan magazines.

Further information about the Forth Interest Group is available from http://www.forth.org/ and information about modern FORTH can be found at https://www.forth.com/forth/

FORTH-R Design Goals

FORTH-R aims to meet the following goals:

- Targeted at Disk based systems with fully expanded memory. TUG-FORTH and TAN-FORTH both had versions to cope with cassette-based systems and less than full memory, it is not intended to recreate this flexibility.
- Based on the Forth Interest Group (FIG) 1.1 Standard. This appears to be the standard used by all contemporary versions of FORTH.
- Incorporate fixes for all public bugs in fig-FORTH 1.1 There are a number of well publicised issues such as U* and U/ bugs with public domain fixes available.
- Inclusion of a fully featured Editor and Assembler. The Editor and Assembler in TUG-FORTH have some limitations so the versions from TAN-FORTH are used.
- Support of up to 2 drives with 80 track disks with 256byte sectors, 10 sectors per track. Disk drive numbering is as TANDOS so DR0 and DR1 are sides A and B of drive 0.

- Support for Microtan Graphics, both Chunky and HRG (HRG not implemented at time of writing this version of the manual)

Copyright and Warranty

The fig-FORTH source code is public domain and therefore no copyright issues should arise. A number of the extensions, the editor and assembler are copyright Tangerine Computer Systems. It is assumed that as these are freely circulating in the public domain that permission is granted for their reproduction here.

If anybody believes their rights have been infringed please contact the author immediately.

The FORTH-R software and this manual are published as a hobby project for personal study purposes. No warranty of any kind should be implied.

Getting started

Installation

FORTH-R is distributed as 3 files. FORTH-R itself as a .WAV and a .BIN file; Editor.txt and Drive build script.txt. FORTH-R assumes the system is disk based and the following steps create an initial system disk. First decide on which disk will be used to store the Forth Screens. Notice that FORTH uses raw disk access, not TANDOS directories and files so once FORTH has written to a disk it is no longer readable by TANDOS.

A convenient setup with a double-sided drive is to use drive 0 as a TANDOS drive to store FORTH-R and the other side referred to by both TANDOS and FORTH-R as drive 1. In this instance, both disk sides may be prepared using either CREATE or FORMAT & INIT. However, Drive 1 only actually requires FORMAT. If it has been INITialised, the directory and sector indexing will be overwritten when the side is used for saving FORTH Screens.

Loading from .WAV

FORTH-R is saved in XBUG Fast mode so can be loaded with

F

F,FORTHR

It resides from $400 to $1CCA so can be saved to disk with

S FORTHR 400 1CCA T400

Loading from .BIN

The .BIN file is formatted for use with X MODEM so it is simply a Binary dump with 00,04 at the start as a load address. It can be loaded using XMODF0 on the Microtan and TeraTerm on a Windows PC. Or any other terminal emulator that supports X Modem.

FORTH-R resides from $400 to $1CCA so can be saved to disk with

S FORTHR 400 1CCA T400

Loading the Editor

Editor.txt is a text file intended to be sent from a terminal emulator to the Microtan. The emulator should be set to put a delay of 20mSec between characters and 2000mSec between lines

Load FORTH-R

Configure Microtan to use serial terminal then on serial terminal

G400

EMPTY-BUFFERS

0 WARNING !

(The command 0 WARNING ! turns off verbose error messages. It is not normally required at start-up but when loading the editor always throws a warning and accessing the disks will usually cause the system to miss the next line from the serial port.)

Then send the file editor.txt.

Creating the disk Screens

With FORTH-R running and the editor loaded as above, select the drive to be used for FORTH Screens, with the suggested layout this would be:

DR1

Now send the Drive build script.txt file to the Microtan using the terminal emulator. This uses the editor to create several screens and save them to disk. Blanks lines are inserted to give plenty of time for the disks.

Screen 0 is a list of all the other screens.

0 LIST

Displays the contents of screen 0 which should be

```
SCR# 0

 0 ( Screen 0          This directory )
 1 ( Screen 1          Graphics extensions and PICK )
 2 ( Screen 2          Random Number extensions )
 3 ( Screen 3          Copyright Message )
 4 ( Screen 4 to 5     Error Messages )
 5 ( Screen 6          Block and Screen copy utility )
 6 ( Screen 7          Tanforth Array Extension )
 7 ( Screen 8          Tanforth Case Extension )
 8 ( Screen 10 to 14   Breakout game (needs extensions) )
 9 ( Screen 20 to 26   Editor )
10 ( Screen 30 to 34   Assembler )
```

See the Extra Words section for an explanation of these additional screens

Using Forth-R

Starting

FORTH-R has two entry points, G400 executes a cold start and G404 executes a warm start.

It will respond with:

FORTH-R based on fig-FORTH 1.1

Neither cold or warm start initialise the disk buffers so it is essential issue the command:

EMPTY-BUFFERS

Error messages are stored in screens 4 and 5 of the currently selected disk. Unless and appropriate disk is inserted type:

0 WARNING !

To disable reading from the disk.

Loading from Disk

If any of the extensions are required they can be loaded from the appropriate screens before the application that needs them is loaded for example breakout requires the graphics and random number words on screens 1 and 2 so to load breakout:

1 LOAD

2 LOAD

10 LOAD

Typing VLIST will reveal that a number of words have been added to the dictionary. The last of these is BREAKOUT and executing this word runs the game.

Implementation Specifics

As FORTH-R is based on fig-FORTH most of the information in the excellent TAN-FORTH manual applies. The following sections detail the differences between FORTH-R, TAN-FORTH and TUG-FORTH

Disk access

FORTH-R is hard coded to use drives with the following characteristics:

Bytes per Sector	256
Sectors per track	10
Tracks per side	80
Sides per drive	2

A FORTH Screen is always 1024 bytes so in FORTH-R it is stored in 4 blocks, and screen n starts at block 4n. fig-FORTH treats all drives as a continuous list of blocks. This means that in FORTH-R blocks and screens are laid out as follows:

Block Number	**Screen Number**	**TANDOS Drive**	**Physical drive**
0 - 799	0 - 199	DRV0	Drive 0 side 0
800 - 1599	200 - 399	DRV1	Drive 0 Side 1
1600 - 2399	400 - 599	DRV2	Drive 1 Side 0
2400 - 3199	600 - 800	DRV3	Drive 1 Side 1

FORTH keeps track of a variable OFFSET which is set by the DRx words. DR0 sets OFFSET to 0 DR1 sets it to the size in blocks of one side (800) DR2 to two sides etc. OFFSET is used by BLOCK so if DR0 has been issued then OFFSET is 0 and the screens and blocks will be referenced by the numbers in the table 4 LIST will list screen 4 on DRV0 (drive 0 side 0), once DR1 has been issued OFFSET will be 800 and 4 LIST will list the contents of screen 4 on DRV1 (drive 0 side 1) which in absolute terms is screen 204 which starts at block 816.

TAN-FORTH supports different drive sizes and has slightly different definitions for DR1 etc. In TANFORTH DR1 sets OFFSET to point to Drive 1 side 0. FORTH-R should be able to read a disk written by TAN-FORTH if absolute block numbers are used.

TUG-FORTH (the DFORTH version) is only able to access drive 0 and uses 256 Bytes per Sector, 9 Sectors per track, 40 Tracks per side. FORTH-R can read DFORTH blocks correctly although those not on track 0 will be numbered incorrectly.

Cassette Storage

FORTH-R does not support cassette storage. The simplest way to transfer screens stored on cassette is to load them into the version of FORTH that created them and LIST them capturing the output in a terminal emulator and the recreating them in FORTH-R using the editor by issuing the commands

N CLEAR (Where N is the desired Screen number)

EDITOR (Switches to the Editor vocab if not already active)

0 NEW (Instructs the Editor to start inserting text at line 0)

Then stream the text for the screen ensuring no blank lines. If the text is not 16 lines long add an extra blank line at the end to indicate the end of the screen to the editor.

FLUSH (Writes the screen to disk)

Error Messages

FORTH-R looks for the error message files in Screens 4 and 5 of the current drive as selected by DR0, DR1 etc. TAN-FORTH and TUG-FORTH follow the fig-FORTH standard and use screens 4 and 5 of drive 0.

This change is deliberate to allow DR0 to be used for a TANDOS formatted drive.

End of Routine JMP Addresses

Routines written in Assembler or machine code end by with a JMP to different locations depending on how they wish the stack to be processed. The absolute locations vary between implementations as follows:

Vector	**FORTH-R**	**TAN-FORTH**	**TUG-FORTH**
NEXT	$0442	$0444	$0447
POP	$0554	$05EE	$0559
PUSH	$043B	$044D	$0440
PUT	$043D	$043f	$0442
PUSH0A*	$0762	$07DC	$0747
POPTWO	$0552	$05EC	$0557

* The TUG-FORTH assembler refers to this label as PUSHOA (with the letter O) in TAN-FORTH and FORTH-R it is PUSH0A (with the number 0)

Where these are referenced by their label in assembler the assembler should set them to the correct value (provided the correct assembler for that version of FORTH is being used. In some FORTH programs direct machine code is used (an example can be seen in BREAKOUT) and these need to by adjusted to the correct addresses. Look for the 4C xx xx at the end of the code fragment.

Other Addresses

Other Addresses that may be referenced in Assembler are

Address	**FORTH-R**	**TAN-FORTH**	**TUG-FORTH**
SETUP	$046A	$0504	$46F

Glossary

The following glossary list the most common FORTH words grouped logically. For less common words not listed here consult the full fig-FORTH glossary document.

Operand Keys:

n 16-bit integer
u 16-bit unsigned integer
d 32-bit signed double integer
addr 16-bit address
b 8-bit byte
c 7-bit ASCII character
f boolean flag.

FIG Stack Instructions

DUP	(n - n n)	Duplicate top of stack.
DROP	(n -)	Discard top of stack.
SWAP	(n1 n2 - n2 n1)	Reverse top two stack items.
OVER	(n1 n2 - n1 n2 n1)	Copy second item to top.
ROT	(n1 n2 n3 - n2 n3 n1)	Rotate third item to top.
-DUP	(n - n ?)	Duplicate only if non-zero.
>R	(n -)	Move top item to return stack.
R>	(- n)	Retrieve item from return stack.
R	(- n)	Copy top of return stack onto stack.
+	(n1 n2 - sum)	Add.
D+	(d1 d2 - sum)	Add double-precision numbers.
-	(n1 n2 - diff)	Subtract (n1-n2).
*	(n1 n2 - prod)	Multiply.
/	(n1 n2 - quot)	Divide (n1/n2).
MOD	(n1 n2 - rem)	Modulo (remainder from division).
/MOD	(n1 n2 - rem quot)	Divide giving remainder and quotient.
*/MOD	(n1 n2 - rem quot)	Multiply then divide (n1*n2/n3) with double-precision intermediate.
*/	(n1 n2 - quot)	Like */MOD but give quotient only.
MAX	(n1 n2 - max)	Maximum.
MIN	(n1 n2 - min)	Minimum.
ABS	(n - absolute)	Absolute value.
DABS	(d - absolute)	Absolute value of double-precision number.
MINUS	(n - -n)	Change sign.
DMINUS	(d - -d)	Change sign of double-precision number.
AND	(n1 n2 - and)	Logical bitwise AND.
OR	(n1 n2 - or)	Logical bitwise OR.
XOR	(n1 n2 - xor)	Logical bitwise exclusive OR.
<	(n1 n2 - f)	True if n1 less than n2.
>	(n1 n2 - f)	True if n1 greater than n2.
=	(n1 n2 - f)	True if n1 equal to n2.
0<	(n - f)	True if top number negative.
0=	(n - f)	True if top number zero.

FIG Input Output Instructions

.	(n -)	Print number.
.R	(n u -)	Print number right-justified in u column.
D.	(d -)	Print double-precision number.
D.R	(d u -)	Print double-precision number in u column.
CR	(-)	Do a carriage-return.
SPACE	(-)	Type one space.
SPACES	(u -)	Type u spaces.
."	(-)	Print message (terminated by ").
DUMP	(addr u -)	Dump u numbers starting at address.
TYPE	(addr u -)	Type u characters starting at address.
COUNT	(addr - addr+1 u)	Change length byte string to TYPE form.
?TERMINAL	(- f)	True if terminal break request present.
KEY	(- c)	Read key put ASCII value on stack.
EMIT	(c -)	Type ASCII character from stack.
EXPECT	(addr u -)	Read u characters (or until carriage-return) from input device to address.
WORD	(c -)	Read one word from input stream delimited by c.
NUMBER	(addr - d)	Convert string at address to double number.
<#	(-)	Start output string.
#	(d1 - d2)	Convert one digit of double number and add character to output string.
#S	(d - 0 0)	Convert all significant digits of double number to output string.
SIGN	(n d - d)	Insert sign of n to output string.
#>	(d - addr u)	Terminate output string for TYPE.
HOLD	(c -)	Insert ASCII character into output string.
DECIMAL	(-)	Set decimal base.
HEX	(-)	Set hexadecimal base.
OCTAL	(-)	Set octal base.

FIG Memory and Dictionary instructions

@	(addr - n)	Replace word address by contents.
!	(n addr -)	Store second word at address on top.
C@	(addr - b)	Fetch one byte only.
C!	(b addr -)	Store one byte only.
?	(addr -)	Print contents of address.
+!	(n addr -)	Add second number to contents of address.
CMOVE	(from to u -)	Move u bytes in memory.
FILL	(addr u b -)	Fill u bytes in memory with b beginning at address.
ERASE	(addr u -)	Fill u bytes in memory with zeros.
BLANKS	(addr u -)	Fill u bytes in memory with blanks.
HERE	(- addr)	Return address above dictionary.
PAD	(- addr)	Return address of scratch area.
ALLOT	(u -)	Leave a gap of n bytes in the dictionary.
,	(n -)	Compile number n into the dictionary.
'	(- addr)	Find address of next string in dictionary.
FORGET	(-)	Delete all definitions above and including the following definition.
DEFINITIONS	(-)	Set current vocabulary to context vocabulary.
VOCABULARY	(-)	Create new vocabulary.
FORTH	(-)	Set context vocabulary to Forth vocabulary.
EDITOR	(-)	Set context vocabulary to Editor vocabulary.
ASSEMBLER	(-)	Set context vocabulary to Assembler.
VLIST	(-)	Print names in context vocabulary.

FIG Defining and Control Structure Instructions

:	(-)	Begin a colon definition.
;	(-)	End of a colon definition.
VARIABLE	(n -)	Create a variable with initial value n.
	(- addr)	Return address when executed.
CONSTANT	(n -)	Create a constant with value n.
	(- n)	Return the value n when executed.
CODE	(-)	Create assembly-language definition.
;CODE	(-)	Create a new defining word with runtime code routine in high-level Forth.
DO	(end+1 start -)	Set up loop given index range.
LOOP	(-)	Increment index terminate loop if equal to limit.
+LOOP	(n -)	Increment index by n. Terminate loop if outside limit.
I	(- index)	Place loop index on stack.
LEAVE	(-)	Terminate loop at next LOOP or +LOOP.
IF	(f -)	If top of stack is true execute true clause.
ELSE	(-)	Beginning of the false clause.
ENDIF	(-)	End of the IF-ELSE structure.
BEGIN	(-)	Start an indefinite loop.
UNTIL	(f -)	Loop back to BEGIN until f is true.
REPEAT	(-)	Loop back to BEGIN unconditionally.
WHILE	(f -)	Exit loop immediately if f is false.

FIG Miscellaneous Instructions

(	(-)	Begin comment terminated by)
ABORT	(-)	Error termination of execution.
SP@	(- addr)	Return address of top stack item.
LIST	(screen -)	List a disk screen.
LOAD	(screen -)	Load a disk screen (compile or execute).
BLOCK	(block - addr)	Read disk block to memory address.
UPDATE	(-)	Mark last buffer accessed as updated.
FLUSH	(-)	Write all updated buffers to disk.
EMPTY-BUFFERS	(-)	Erase all buffers.

FORTH-R Specific Instructions

U.	(u -)	Print unsigned number.
CHUNKS	(x y – addr bit byte)	Takes x,y chunky graphics coordinates and returns display address, bit mask, and current contents of address.
CLS	(-)	Clears the screen
DR0	(-)	Sets offset to select drive 0
DR1	(-)	Sets offset to select drive 1
DR2	(-)	Sets offset to select drive 2
DR3	(-)	Sets offset to select drive 3

FIG System Constants

FIRST	Address of the first byte of the disk buffers.
LIMIT	Address of the last byte of disk buffers plus one pointing to the free memory not used by the Forth system.,
B/SCR	Blocks per screen. A screen is 1024 bytes used in editor.
B/BUF	Bytes per buffer or Block
C/L	Characters per line of input text.
BL	ASCII blank.

FIG User Variables

S0	Initial value of the data stack pointer.
R0	Initial value of the return stack pointer.
TIB	Address of the terminal input buffer.
WARNING	Error message control number. If 1, disk is present, and screen 4 of current drive is the base location of error messages. If 0, no disk is present and error messages will be presented by number. If -1, execute (ABORT) on error.
FENCE	Address below which FORGETting is trapped. To forget below this point the user must alter the contents of FENCE .
DP	The dictionary pointer which contains the next free memory above the dictionary. The value may be read by HERE and altered by ALLOT .
VOC-LINK	Address of a field in the definition of the most recently created vocabulary. All vocabulary names are linked by these fields to allow control for FORGETing through multiple vocabularies.
BLK	Current block number under interpretation. If 0, input is being taken from the terminal input buffer.
IN	Byte offset within the current input text buffer (terminal or disk) from which the next text will be accepted. WORD uses and moves the value of IN .
OUT	Offset in the text output buffer. Its value is incremented by EMIT. The user may alter and examine OUT to control output display formatting.
SCR	Screen number most recently referenced by LIST .
OFFSET	Block offset to disk drives. Contents of OFFSET is added to the stack number by BLOCK .
CONTEXT	Pointer to the vocabulary within which dictionary search will first begin.
CURRENT	Pointer to the vocabulary in which new definitions are to be added.
STATE	If 0, the system is in interpretive or executing state. If non-zero, the system is in compiling state. The value itself is implementation dependent.
BASE	Current number base used for input and output numeric conversions.
DPL	Number of digits to the right of the decimal point on double integer input. It may also be used to hold output column location of a decimal point in user generated formatting. The default value on single number input is -1.
FLD	Field width for formatted number output.
CSP	Temporarily stored data stack pointer for compilation error checking.
R#	Location of editor cursor in a text screen.
HLD	Address of the latest character of text during numeric output conversion.

Additional words

The distribution includes a number of additional screens as follow

Screen 1 Graphics extensions and PICK

These were supplied with TANFORTH and are documented in the TANFORTH manual.

Screen 2 Random Number extensions

These were supplied with TANFORTH and are documented in the TANFORTH manual.

Screen 3 Copyright Message

A Standard part of the FigFORTH distribution

Screen 4 to 5 Error Messages

A Standard part of the FigFORTH distribution, although usually in uppercase these are in sentence case.

Screen 6 Block and Screen copy utility

Created for FORTH_R to copy either a single block or a single screen.

source dest BCOPY --- copy block

source dest SCOPY --- copy screen

Notice that the commands respect OFFSET as set by a DRx word

Screen 7 Tanforth Array Extension

These were supplied with TANFORTH and are documented in the TANFORTH manual.

Screen 8 Tanforth Case Extension

These were supplied with TANFORTH and are documented in the TANFORTH manual.

Screen 9 Cursor on and off

Created for FORTH-R these are two very simple calls to TANBUG routines

CON calls TANBUG CURSON via $F826

COFF calls TANBUG CURSOFF via $F829

COFF can be called before CLINE or PTC to prevent a stray cursor being left on the screen.

Screen 10 to 14 Breakout game

As supplied with TANFORTH needs Graphics and Random number extensions in screen 1 and 2

Screen 20 to 26 Editor

This was supplied with TANFORTH and is documented in the TANFORTH manual.

Screen 30 to 34 Assembler

This was supplied with TANFORTH and is documented in the TANFORTH manual.

FORTH-R Development

The following section contains some notes on how FORTH-R was developed. It is intended to help anyone looking to further develop the project but may also be of general interest.

Tools

FORTH-R is based on the public domain fig-FORTH 1.1 ported to the 6502 by W.F. Ragsdale. The specific version uused was published by Douglas Beattie Jr. for his MAS65 cross assembler. There are several ways to assemble and transfer code to the Microtan. The toolchain used for this project was:

Editing with NotePad++

Assembly by MAS65 running under DOSBox to a .BIN file

BIN files edited with free version of Hex Editor NEO to add start address

Edited BIN file transferred over serial port to Microtan using XMODEM protocol and XMODF0 on the Microtan and TeraTerm on a Windows PC.

Once the core FORTH is running on the Microtan, FORTH based source can be edited in Notepad++ and sent to the Microtan over serial using the TeraTerm 'Send File' function. Line delay needs to be set to about 2000mSec to allow the interpreter time to process longer lines.

Resources

The key document needed to understand FORTH is the System Guide by DR C. H. Ting, Ph. D A very recent version of this is published at http://www.forth.org/OffeteStore/1010_SystemsGuideToFigForth.pdf this includes description of how each FORTH word works.

The original fig-FORTH installation guide, which include the model source is also a useful reference the version published here https://github.com/larsbrinkhoff/forth-documents/blob/master/FIGINST.TXT is useful as it is a text file therefore can be searched.

Understanding the source code

FORTHRxxxx.ASM contains 3 types of routine. The majority of the code is in FORTH dictionary entries which are strung together with linking pointers. There is a good description of the dictionary structure in the TANFORTH manual or the systems guide.

Internal assembler routines

These are simply assembler code, there are very few of them, and they are easy to understand to anyone familiar with 6502 assembler.

Dictionary items in assembler

There are a number dictionary items that are written in assembly, the lowest level of language primitives and maths routines for example. They have a dictionary header, followed by the assembly code itself. For example, the word MINUS is:

```
L670      .BYTE $85,'MINU',$D3              (This is the name)
          .WORD L649     ; link to D+       (Link to previous entry)
MINUS     .WORD *+2                         (Address of the code)
          SEC                               (Code itself)
          TYA
          SBC 0,X
          STA 0,X
          TYA
          SBC 1,X
          STA 1,X
          JMP NEXT                          (Go to the next word)
```

The JMP instruction at the end varies depending on how the stack is to be left. For details see the assembler section of the TANFORTH Manual, and the implementation specifics section of this manual.

Dictionary items in FORTH

The majority of FORTH is written in FORTH. You can see the source code in forth in the installation guide. The FORTHRxxx.asm file contains the compiled version of this source. The code for these has a dictionary header, then lists the FORTH words that make up the word, with some of the words including parameters. For example here is the code for -DUP (which duplicates the item on the top pf the stack if non zero:

```
;                                -DUP
;                                SCREEN 38 LINE 13 (refers to model)
;
L1296     .BYTE $84,'-DU',$D0
          .WORD L1286    ; link to SPACE    (Dictionary header)
DDUP      .WORD DOCOL                       (Calls interpreter)
          .WORD DUP                         (Forth word DUP)
          .WORD ZBRAN                       (Forth word Branch is 0)
L1301     .WORD $4       ; L1303-L1301      (Paramater for ZBRAN)
          .WORD DUP                         (Forth word DUP)
L1303     .WORD SEMIS                       (Forth word ; to end)
```

It can be seen that these cabn be easily modified although care is need when working between ZBRAN or BRAN words are there parameters are signed relative jumps.

The xxFF Problem

Because of the way the dictionary is searched word headers must not be in memory at page boundaries. If making changes then the xxxx.LST file must be checked to see that no headers have been moved to reside at an xxFF address. .WORD 0 can be added between words to pad the code out to solve any found.

Changes from fig-FORTH 1.1

The following changes were made from the original fig-FORTH source

1. XEMIT masks MSB in line with FORTH specification and calls TANBUG OUTALL
2. XKEY calls TANBUG JPLKB
3. XQTER returns 0
4. XCR calls TANBUG OUTRET
5. RSLW calculates drive track and sector and calls XRSLW
6. XRSLW takes disk values passed from RSLW and calls TANDOS routines to read or write a sector.
7. B/SCR changed for 4 x 256 byte buffers per screen
8. EXPECT normally sets Backspace character to $08 changed to use $7F
9. EMPTY-BUFFERS sets PREV and USE to FIRST. This is described in the systems guide for FORTH but not in the original code.
10. Added DR2 and DR3
11. U< replaced with fixed version
12. U/ replaced with fixed version
13. U* replaced with fixed version
14. Added U.
15. Added CHUNKS (copied from TANFORTH)
16. Added CLS
17. MESSAGE now does not override OFFSET. figFORTH assumes that the error messages are stored in screen 4 and 5 of drive 0. This is unhelpful on a Mirrotan so FORTH-R assumes they are stored in screen 4 and 5 of the current drive.
18. Removed the form feed from INDEX
19. Fixed a bug in FORGET