FORTH Workstation for the 1802

DRAFT RELEASE 5 - code rev r.2.4 2025/09/19

Contents

Introduction

Why another Forth implementation for the RCA1802?

There are several very good published versions of Forth for the RCA1802. However, they mostly implement core Forth words from the figFORTH model. Which then essentially turns an 1802 system into a kind of RPN programmable calculator. Lacking disk storage or any means of writing, compiling, running and saving code in a reusable manner they tend to get implemented (a fun project) and then ignored.

Meanwhile. sometime back in the early 1980's I decided I wanted more. So, I pulled together various publicly available bits of code, wrote some more of my own, and built a true multi-tasking Forth based 1802 workstation. And rather than relying on compiling basic functionality from source every time you want to use it, I integrated an editor, assembler, and multitasker into the base system.

From 1982 to 2001 my 1802 workstation functioned as my development system and home automation system. Roll the clock forward to the 2020's and that old system is still running (albeit not doing HA anymore) and I updated the code to support Q/EF type serial I/O as well as the original CDP1854 UART. Which means I'm not using that system on the two Lee Hart Membership Card's I own – both the early version and later version with 7 segment LEDS.

So, in conclusion, I've released this code to capture my 40-year project and share. Hopefully somebody else might enjoy it, or at least pull ideas and code for their project.

Scope: What's Included?

Off the top of my head, I've added the following to the original fig-FORTH release:

1. Console code supporting **19200 baud Q&EF serial I/O**. With ?TERMINAL support integrated into the inner interpreter to allow "BREAK" functionality when words like VLIST or DUMP are outputting long listing.
2. Console code supporting an interrupt driven **CDP1854 UART**
3. **RAM disk** I/O - as many 1K FORTH screens as you have free memory - possibly battery backed / EPROM / EEPROM as required.
4. **Line Editor** - pre-assembled as words in a FORTH dictionary for editing RAM disk FORTH "screens" (see above)
5. **1802 Assembler** - pre-assembled as words in a FORTH dictionary to allow adding inline 1802 code to other Forth words
6. **ROM-able** – generic 1802 figFORTH will run from ROM but would not work with multiple vocabularies. That has been fixed.
7. Support for **loading and storing Forth screens** or random memory blocks over the serial port as either Forth formatted text files or Intel Hex format raw data files.
8. **Multitasking** - cooperative multitasking support for concurrent tasks - task 1 runs the console (works best in a fully interrupt driver system but work well on a simple ELF)
9. **Configurable build modes** for different hardware and software combinations.
10. **1802 specific words** for controlling Q, testing reading the EF lines, and doing N-line port I/O
11. Built in **ASCII text error messages** (from the original screen 4 in figFORTH).
12. **Pre-configured versions** can be built for a simple Q/EF serial I/O Elf, an 1802 system with a 1854 UART (interrupts supported), or both early and recent versions of Lee Hart's Membership Card (with or without interrupt support).
13. **RTC support** - interrupt driven Motorola MC146818
14. Support for a few other **hardware specific things** like an ADC0808 A/D converter, intel 8255 I/O port, eight digit 7-segment display, and a single line alpha-numeric LCD display.
15. **Example screens** showing words written in Forth assembler, and words using the multitasking
16. **Autoload** screens on boot option via dip switch selection.
17. .... and probably some **other goodies**.

History

In case you are interested, I started this project back in 1982. Step one was hand typing into PDP11 system a photocopy of a photocopy of the figFORTH reference implementation for the 1802. Then I created hex files to burn to EPROM using an 1802 assembler for the PDP11 written by Wayne Bowdisk (hi Wayne wherever you are). I also wrote an 1802 simulator in PDP-11 FORTRAN to facilitate debugging of the code. And then I wrote about all of this in Ipso Facto issue #29 (May 1982).

link > IPSO FACTO Issue #29
http://cosmacelf.com/publications/newsletters/ipso-facto/ipso-facto-29.pdf

There were several follow-on articles (email me for copies if these links are dead):

link > IPSO FACTO Issue #31
http://cosmacelf.com/publications/newsletters/ipso-facto/ipso-facto-31.pdf
link > IPSO FACTO Issue #32
http://cosmacelf.com/publications/newsletters/ipso-facto/ipso-facto-32.pdf
link > IPSO FACTO Issue #34
http://cosmacelf.com/publications/newsletters/ipso-facto/ipso-facto-34.pdf
link > IPSO FACTO Issue #35
http://cosmacelf.com/publications/newsletters/ipso-facto/ipso-facto-35.pdf
link > IPSO FACTO Issue #37
http://cosmacelf.com/publications/newsletters/ipso-facto/ipso-facto-37.pdf
link > IPSO FACTO Issue #38
http://cosmacelf.com/publications/newsletters/ipso-facto/ipso-facto-38.pdf

At one time my system worked with the A.C.E. WD1771 controller board running dual 8” floppy disk drives. I converted to battery backed RAM in the mid-80’s and sadly those drives, and the supporting code, are long gone.

In the meantime, while I've added a few things to the A.C.E. release of fig-FORTH over the years, the original version of the source code file is long lost. So back during the Covid19 lockdown, I recreated the current version recreated by reverse assembly of the ACE EPROM. Of course I used tools written in 1802 figFORTH to disassemble itself. And having played with two versions of Lee Hart’s Membership Card (link> https://www.sunrise-ev.com/1802.htm) I naturally ported the code to those.

Which brings us to 2025 where I’ve collected all the bits and pieces and created a single source file with configurable build options for console and timer functionality, as well as several memory models and a few special features.

Implementation

Pretty much every version of Forth for the 1802 requires a console, which in turn requires the definition of three words

- KEY : a word to get a single character from the console,
- EMIT : a word to output a single character to the console,
- ?TERMINAL ; a word to indicate a console key has been pressed

In ACE1802FORTH the labels for these words are getKEY, CSEND and qTERM. Sample implementations have been provided for using software serial I/O via Q & EF3, as well as hardware UART I/O via an interrupt driven CDP1854 UART. Modify you implementation using those routines as a guide.

In addition to the console words, figFORTH has example words meant to be used for interface to two floppy disk drives. ACE1802FORTH originally supported two 8" drives via a WD1771 chip but has been modified to use RAM disk only. To make that useful, please see the Work Flow section of this document for information about how to load and store programs to and from the RAMDISK via the console (and a suitable terminal emulator program on your PC).

Finally, to support cooperative multitasking ACE1802FORTH needs a "tic timer" – a signal used to indicate when tasks should run and when they should sleep. A tic time is typically created from a periodic hardwire interrupt but ACE1802FORTH also simulates a tic timer in software if no hardware source is available. Naturally timing accuracy suffers when a software timer is used but it still allows cooperative multitasking. In any case, example 1802 interrrupt service routines are provided for a periodic hardware interrupt (like the later versions of Lee Hart's Membership Card) and a real time clock chip (the Motorola Mc 46818).

Source Code

Complete assembler source code and documentation can be found at :

https://github.com/anthonylhill/ACE1802FORTH

Build Options

ACE1802FORTH source code is written to allow various build option so that it supports multiple hardware platforms and different memory configurations. Listed below are the build options that can be found at the start of the source code file.

Build Option	Choices	Description
memory_model	• ram_elf • rom_ram_elf • ram_rom_elf • ace_cpu_card • custom	• simple Elf with 32K RAM at $0000 • Elf or Membership Card with ROM at $0000 and RAM at $8000 • ELF or Membership Card with ROM at $8000 and RAM at $0000 • ACE CPU card with 16K ROM at $0000 and 48K RAM • custom memory model - edit for user system needs
uart_type	• hardware • software	UART implemented in software or hardware
timer_type	• hardware software	tic timer implemented in software or hardware ?
extra_hardware	• yes • no	include code for extra hardware support for ACE CPU systems
example_screens	• yes • no	include example Forth source screens at a screen # XSCREEN in the code section
autoload_screen	• yes • no • selectable	enable autoload of example screen on startup (selectable assumes an I/O switch)
clock_mhz	• 1 • 4	1802 CPU clock speed for software UART timing (1.8 Mhz or 4 Mhz)
uart_config	• $3E	CDP1854 control register (hardware UART only) : Interrupts enabled, 8 data bits , 2 stop bits , even parity , parity enabled
editor	• yes • no	include line editor code
assembler	• yes • no	include assembler code
stackptr_show	• yes • no	show top of stack address as part of the OK prompt
zero_ram	• yes • no	include code to set all RAM to $00 at start-up
lbr_at_zero	• yes • no	insert a LBR START at $0000 (optional for memory maps where code does not start at $0000)
prescaler_value	• $80	a value used to roughly calibrate the software tic timer (if used) by dividing down how ofter it updates

Memory Map (Example)

ORG	$0000	code start
EXAMPLE_SCREEN	(end of code space)	storage address of example screen
START_OF_RAM	$4000	start of RAM area - must be on a page bountry
END_OF_RAM	$8000	end of RAM block - first byte after
S0_START	END_OF_RAM -$0200	data stack for console task -grows up) variable name = S0
R0_START	END_OF_RAM-$0101	return stack for console task (grows down) variable name = R0
USER_START	END_OF_RAM-$0100	USER area - 64 variables max variable name = UP
TIB_START	END_OF_RAM-$0080	terminal input buffer variable name = TIB
tx_buffer	END_OF_RAM-$0400	256 bytes of RAM used as a hardware UART tx buffers NOTE : buffer must be on page boundaries
rx_buffer	END_OF_RAM-$0300	Reserve 256 bytes of RAM for hardware UART rx buffers NOTE : buffer must be on page boundaries
FIRSTB	END_OF_RAM-$0400	used for RAM disk - address of first disk screen (note #0 is unusable) variable name = FIRST
LIMITB	$FFFF	end of RAM disk area variable name = LIMIT
task1stacks	END_OF_RAM-$0780	task 1 stacks
task2stacks	END_OF_RAM-$0700	task 2 stacks
task3stacks	END_OF_RAM-$0680	task 3 stacks
task4stacks	END_OF_RAM-$0600	task 4 stacks
task5stacks	END_OF_RAM-$0580	task 5 stacks
task6stacks	END_OF_RAM-$0500	task 6 stacks
task7stacks	END_OF_RAM-$0480	task 7 stacks

Multitasking

One fun aspect of the ACE1802FORTH workstation is the ability to run multiple tasks simultaneously with the console. Multitasking on the workstation is cooperative – tasks run until they release control to the next task in the queue. A simple "tic timer" mechanism exists to allow tasks to start and stop themselves for defined intervals. The timer runs from either a hardware interrupt or a software simulated interrupt (which is necessarily less accurate).

As implemented, the workstation supports up to eight tasks, but more can be added with a simple edit to the source code. Task 0 is by default the console task, leaving seven tasks for user functions. Each task has its own return stack and computation stack, along with a tic timer used to let it "sleep" periodically. The tic timer is an unsigned 8 bit value so valid setting are 1 to 255.

There is a configuration values to roughly calibrate the value of a tic (prescaler_value) if the software emulated tic timer option is selected. Adjust that value between 1 and 255 to prescale the rate at which the system counts emulated tics.

Multitasking requires only six new Forth words :

1. **START -** used to create a dormant task that will run a single Forth word when started. Usage is : **n START text** where n is the task number (1 – 7) and text is the Forth word for the task,
2. **RUN** – used to start a dormant task. Usage : **n RUN** where n is the task number.
3. **HALT** – used to stop a running task. Usage : **n HALT** where n is the task number
4. **PAUSE** – used within a task to release CPU control until the next pass through the task list.
5. n **TIC** – used within a task to release CPU control for "n" tics. Range 1 – 255. If n=0 then task will halt.
6. **TASK#** - used within a task to retrieve its task number. Useful for letting a task HALT itself.

An example task to blink the 1802 Q LED might look like :

```
: QTASK   BEGIN  QON 20 TIC QOFF 40 TIC AGAIN ;
2 START QTASK
2 RUN
```

A Brief Note about Software UARTs

Implementing a UART in software via discrete I/O pins (Q & EF3) creates some unique problems for a multitasking system. Proper timing requires masking interrupts during character transmission and reception. And detecting an incoming character can fail if other task in the system have control of the processor. To that end, for software UART the ACE1802FORTH uses a hybrid console that locks out multitasking while an input line is being received. Once the line has been fully received and is ready to process (i.e. the user pressed carriage return CR) multitasking is reenabled while the input line is processed and/or executed. In addition, the user can enable or disable multitasking by using the words **EI** (enable interrupts) and **DI** (disable interrupts). That can be especially useful when using the Editor when multi-tasking can be

distracting. Note: none of this applies to systems using hardware interrupt drive UARTs, which will happily multitask while processing user input from the console.

RAM Disk

Storage of Forth source code “screens” in ACE1802FORTH workstation was originally implemented for two 8” floppy disk drives. Those drives were noisy, bulky, and slow so the system converted to using a “RAM disk”. A RAM disk is simply an area of the microprocessors 64K memory used as storage to simulate a real disk drive. In Forth, disk drives are divided up into “screens”, which were traditionally 1K blocks of storage that could be displayed as sixteen 64-byte lines of ASCII text. So, the ACE1802FORTH RAM drive was simply a mapping of the screen concept onto the processor’s memory. This allowed up to 64 screens, aligned to the start of free memory as screen 1, with some screens obviously pointing to unused memory regions or the code of ACE1802FORTH itself.

Of course, the problem with a RAM drive is that data stored in it is not persistent. When the power to the system is turned off, the data goes away. It is of course possible to use battery backed up memory, NVRAM, or even EPROM / EEPROM for all or part of a RAM drive. In fact, the Forth demo example screens included in ACE1802FORTH are just that – screen data stored with the ROMable Forth code. But for other development purposes, a better way to save and restore Forth source code from the RAM drive is needed.

With ACE1802FORTH, it is assumed that the “console” is typically a PC of some type with its own non-volatile storage that can be used to save & restore Forth screens and so words are available that support that storage. The user has the choice of storing screens as slightly compressed ASCII text files, or embedded in error checking Intel hex formatter file. To save or restore the appropriate word is typed on the console and ACE1802FORTH switches into a send or receive mode. The use then uses the text file transfer capability of the terminal emulator software on their PC to send or receive the selected type of file. (Note : recommended terminal emulator for Windows = TeraTerm and Linux = minicom)

File transfer words :

Word	Stack Diagram	Description
SCR>	n screen# ---	Transmits n screens, starting at screen#, to the console. Lines are truncated at the last non-space character and a CR inserted.
>SCR	screen# ---	Receives incoming characters from the console to screen#. Input text files should be delimited by a ~ (tilde). Lines are padded to 64 characters with spaces if necessary.
IH>	addr n ---	Transmits to the console n bytes of memory data in Intel Hex format from memory starting at addr.
>IH	---	Receives data from the console formatted as Intel Hex and stores in memory address from the intel hex file.

Vocabularies

The ACE1802FORTH has three Forth style vocabularies built-in. It has been modified to allow the additional of more vocabularies even if the code is burned into an EPROM or other non-volatile memory.

The built-in vocabularies are : FORTH ASSEMBLR EDITOR.

Switching between vocabularies is done by simply typing in the name of the vocabulary. To extend any vocabulary, type the word DEFINITIONS after the vocabulary name.

eg :

```
FORTH DEFINITIONS
EDITOR DEFINITIONS
ASSEMBLER DEFINITIONS
```

A complete word list to each vocabulary can be found on the github repo :

https://github.com/anthonylhill/ACE1802FORTH

Editor

To use the editor, type : **EDITOR**.

To select the screen to edit, type : **nn EDIT** (where nn is the screen number).

To initialize a screen to all blanks , type : **nn CLEAR** (where nn is the screen number).

SCREENS

n	**LIST**	list screen *n*
n	**CLEAR**	clear screen *n*
n1 n2	**COPY**	copy screen *n1* to *n2*
-	**L**	list current screen

CURSOR

-	**TOP**	position cursor at start of screen
n	**M**	move cursor by signed amount *n*
-	**F** ***text***	move ***text*** to PAD and search forward
-	**N**	find next occurrence of ***text*** from by F
-	**B**	back up over ***text*** found by F

LINES

n	**P** ***text***	overwrite line *n* with ***text***
n	**E**	erase line n with blanks
n	**S**	spread at line n. Move lines down

PAD

n	**H**	hold line *n* at PAD
n	**D**	delete line *n* to PAD.
n	**T**	type line *n* and save in PAD
n	**R**	replace line with PAD
n	**I**	insert text from PAD at line *n*.

TEXT EDIT

-	**C** ***text***	spread and copy in ***text*** at cursor
-	**TILL** ***text***	delete from cursor to ***text***
-	**X** ***text***	find and delete ***text***
n	**DELETE**	delete *n* characters before cursor

Assembler

The ACE1802FORTH workstation includes a Forth vocabulary for an 1802 assembler that allows the creation of assembly code words. Direct input of all 1802 opcodes is support but it's one-pass assembler, meaning forward branches have to be implemented using pseudo operator structures. Supported structures are :

BEGIN, …code… **AGAIN,**
BEGIN, …code… test **UNTIL,**
BEGIN, …code… test **WHILE,** …code… **REPEAT,**
test **IF,** …code… **ELSE,** …code… **ENDIF,**

Forth words writing in assembler look like these examples:

ASSEMBLER

CODE QON SEQ, NEXT

CODE LEDS 9 INC, 9 LDN, STXD, IRX, 4 OUT, 9 DEC, 9 DEC, 9 DEC, 9 DEC, NEXT

CODE TEST1 Z IF, SEQ, ELSE, REQ, ENDIF, NEXT

CODE TEST2 BEGIN, SEQ, REQ, 0 LDN, Z UNTIL, NEXT

CODE TEST3 BEGIN, SEQ, AGAIN, NEXT

CODE TEST4 BEGIN, 4 LDN, Z WHILE, REQ, REPEAT, NEXT