[feb 2017]

What is v4th?

v4th is a Forth-like programming platform/framework for embedded microcontrollers. Much like Forth and LISP derivatives, it provides another "secret weapon" in the savvy programmer's arsenal/toolkit/bag-of-tricks.

Since its first version on the RCA 1802, v4th has been used to build many successful products, and ported to a wide variety of CPUs over the years, with continuous evolution and improvements. Today, versions exist for MSP430, MIPS, RX, and various flavors of ARM targets.

v4th is not Forth -- not quite. It only runs an "inner interpreter" threading engine, and thus has no interactive REPL console, native compiler, nor its own IDE.

Then why use it?

High Performance. v4th's speed is in the same league as the very best fully-compiled Forths or C, along with a very compact code size footprint. Note that v4th was developed not for the mere ego-gratification of offering Yet Another Forth, but to fulfill a real need that classic Forths cannot.

Just like Forth, v4th uses a dual-stack Virtual Machine and nested definitions, and allows you to use the same design methodology and programming techniques as Forth. The author finds it much more productive and less annoying than C.

v4th is intended for embedded turnkey systems that are purpose-built machines, where user-programmability is undesirable and inappropriate. Nevertheless v4th is completely "open"; an experienced MCU jockey can easily modify the software at will.

In the hands of a savvy programmer, v4th can easily produce high-quality results similar to Forth Inc.'s SwiftX or MPEforth's VFX. Forth programmers (especially those with assembly-language experience) have no trouble grokking v4th's idioms. It is in fact an assembly language package that rides on top of the assembler, and thus provides all the convenient and productive features that modern developers expect.

This actually has some advantages:

(caveat: we are now into nerdy technicalities that will make the most sense to people who are familiar with Forth internals.)

- extremely compact footprints are possible; apps can be built within (for instance) 256 bytes of ROM and 16 bytes of RAM. There is no dictionary structure, so all v4th words are headerless.

- transparent access to the datasheet's standardized symbols; there is no need for a lot of 'constant' declarations; the assembler already knows the vendor's standardized labels (e.g. for the UART's baud-rate-divisor register), because the vendor has already pre-supplied the appropriate header files that contain the necessary #define and/or EQU directives. It's a nice thing, when your symbols agree with the manufacturer's documentation.

- inherits the assembler's IDE, with all those nifty features, and all the JTAG debug amenities that the assembler provides.

- WYWIWYG debugging; there is no doubt about what machine code will execute, because What You Wrote Is What You Get. There is no compiler that makes any (perhaps unexpected) decisions on your behalf.

- very high performance; v4th is single-indirect (aka "direct-threaded"), and (depending on the target CPU) can often use a one-instruction NEXT. The v4th nucleus is practically guaranteed to remain resident in caches.

- highly optimized primitives; v4th provides state-of-the-art efficiency; needless stack-pumping is eliminated.
  - non-destructive variants that obviate 'dup' and 'over'.
  - reversed variants that obviate 'swap'.
  - fully conjugated conditionals, e.g. 'nif' instead of "not if".
  - multi-way and table-based branches for state-machine designs.
  - branch target addresses are absolute, not relative offsets; thus no calculation cycles are required.
  - ToS (and typically also NoS) are cached in CPU registers, instead of being held in RAM.
  - high-level inline literals that use scratchpad registers, instead of the parameter stack. e.g. "addk, 5" instead of "5 +"; "strva, VALUE, ADDR" instead of "VALUE ADDR !"; wasteful push/pop thrashing is eliminated; for example, Forth's

```
Daddr @ PIXSIZE CHARWIDTH * - Daddr !
```

is

```
  DW  pstrkk,-(PIXSIZE * CHARWIDTH),Daddr
```

in v4th's idiom, and (of course) runs much faster.

- low-level inline macros that eliminate nesting; many v4th words have both inline and "worded" cognates, e.g. for ARM (note the upper-/lower-case difference):

<pre>
<code>

```
NEXT    MACRO
        ldr     PC, [i], #4
        ENDM

DUP     MACRO
        str     n, [p, #-4]!
        mov     n, t
        ENDM

dup     DUP
        NEXT
```

Here is v4th's NEXT for some other targets:

```
MSP430:
        mov     @i+, PC

RX:
        RTS  ; the UserStackPointer is hijacked, and used as the Interpreter Pointer.
```

RX again, with an alternate implementation for the meek and timid:

```
        mov.l   [i+], w
        jmp     w
```

MIPS:

```
        lw      w, 0(i)
        jr      w
        addi    i, i, #4 ; executed in branch delay-slot.
```

</code>
</pre>

- utterly flexible; because v4th words are all in assembly language, you can quite seamlessly switch to/from writing in machine code and high-level, on the fly. This point deserves a bit of amplification: Forth is not a perfect language, and some jobs (e.g. DSP or graphics) can become awkward and klunky. v4th allows you to completely bypass the Forth Virtual Machine, and take full advantage of the CPU's general-purpose register set and complete instruction repertoire.

Lastly, v4th is interoperable with C; you can take advantage of existing middleware such as USB drivers or TCP/IP protocols (like it or not, they're probably implemented in C), without re-inventing the wheel.

As a comparative example, here are three ways of flashing an LED:
(the LED blinks quickly at first, then slows down)

<pre>
<code>

in Forth:

```
literal1 constant LEDPORT
literal2 constant LEDBIT


: toggleLED     \ read/modify/write, toggle LEDBIT only
LEDPORT @       \ get LED status
dup not         \ toggle status
LEDBIT and      \ isolate bit
swap LEDBIT not and    \ clear LED bit
or LEDPORT !    \ merge new status, and update port
;
```

\ : toggleLED LEDPORT @ dup not LEDBIT and swap LEDBIT not and or LEDPORT ! ;
\ this one-line version is bad style, but it's fun to say the words out loud... :-)

```
: LEDflash
0
begin
  1 + dup       \ increment delay
  begin
    1 - dup 0=
  until
  drop toggleLED
again
;
```

in v4th:

```
LEDPORT EQU     AsmSymbolForPortReg
LEDBIT  EQU     AsmSymbolForPortPin


 NEST   toggleLED       ; 'NEST' is v4th's equivalent of 'DOCOL'.
        DW      atk, LEDPORT, nott, rmwam, LEDPORT, LEDBIT    ; read/modify/write
                                                 using inline address and mask.
        DW      nexit   ; 'nexit' is v4th's equivalent of 'SEMIS'.


 NEST   LEDflash
        DW      zero
        DW      begin
        DW        inc, dup      ;  increment delay
        DW        begin
        DW          dec
```

```
        DW        zuntiln      ; v4th's equivalent of "dup 0= until"
        DW        drop, toggleLED
        DW      again
; infinite loop, so 'nexit' is not required
```

using v4th inline machine-code macros:

```
LEDflash
        ZERO
_flashloop
        INC     ;  increment delay
        DUP
_delayloop
        DEC
        bnz     _delayloop

        DROP
        ATK     LEDPORT
        NOTT
        RMWAM   LEDPORT, LEDBIT        ; read/modify/write, toggle LEDBIT only
        b       _flashloop

</code>
</pre>
```

And of course, for those instances where normal v4th may not be the most appropriate means of solving the problem, you can still write a v4th word as fully-handcrafted assembly code, and that word is added to the vocabulary like any other.

Finally, I must give my sincere thanks and acknowledgement to all the fine Forth folks who have helped me with v4th's evolution and improvement over the years. There are too many people to list here, but I hope that they all know who they are...

I'm somewhat clever, but really it has been other people's feedback and input to v4th that makes me look more brilliant. :-)

Two people deserve special mention:

1 - My brother Myron Plichota; he is brilliant.

2 - I feel truly honored that Cyde W. Philips Jr. (another brilliant fellow) has seen fit to incorporate some v4th constructs into his recent FISH forth **.**

cheers, - vic

vic plichota
embedded systems architect

vic.plichota@gmail.com
vic@MachineQuiltingRobot.com
voicemail: 1 254 247 2265
mobile/text 1 254 913 9149 (please pre-arrange)
email to mobile/text 2549139149 (@ "at") mms ("dot") att ("dot") net
Skype **vapats1802**
Kik **vapats**
https://www.linkedin.com/in/vicplichota
https://www.dropbox.com/sh/grnt4bb34och0jd/AABuuZ71ZkiIbfrv_QwpyGVwa?dl=0
http://MachineQuiltingRobot.com/
