CamelForth vs 4e4th05a

Why did we make a 4e4th out of CamelForth?

We needed a Forth that was tailored to our experiments with the MSP430 Launchpad. It should be very error-tolerant and practical in conjunction with the 4e4th-IDE programming environment. At that time CamelForth was a good base for that. But the constant changes to IAR's "Kickstarter" IDE was not, and unfortunately CF was written in it. This proprietary software became more and more of a hindrance.

But now there is Mike Kohn's assembler. And Brad Rodriguez has ported and improved his CamelForth there. CF now offers tools like MARKER and SAVE and RESTORE, as well as good ways to restore the intact Forth kernel even if garbage has been programmed. From the terminal, this is achieved interactively with SRCUB or via the hardware of the launchpad, the two buttons S1 and S2. So, barely any changes were needed to get our 4e4th look & feel back.

Developing Applications.

CamelForth/MSP430 has the ability to save and autostart user-defined application programs. 4e4th05a works the same way. That was not changed.

Read the appropriate chapter in **msp430development.pdf** to understand how it works.
This also applies to the following chapters.

Annotation:
In the older 4e4th, version 0.34, it worked a little differently. You had to APP ! to start the auto-launch of an application. That is now eliminated. SAVE is enough to start the last defined Forth word automatically after the reset.

SCRUB and RESTORE

In 4e4th there was WIPE instead of SCRUB. Both work identically. In 4e4th05a **both** are included.

In 4e4th there used to be WARM as a starting option besides COLD. This notation was still from the old AIM65 Forth times. CamelForth can perform a RESTORE after COLD, which equates to a warm start when SAVE is done. For all these commands this is true:
They are provided for development and debugging purposes. There should never be a need for your application program to call RESTORE or COLD or SCRUB.

Autostart Bypass

4e4th had this, and CamelForth implemented it too. So it works in both of them alike.

Summary of behaviors

See: msp430development.pdf, page 4.

4e4th and CF have identical behaviors.

Creating an “autostartable” Application

Same as CF, see the PDF:
"Application programs must reside entirely in ROM! ... the word which starts your application must be a colon definition, and must be the last word you compile."

MARKER

Same as CF, see the PDF:
"If you use MARKER, normally you’ll just use one, so this isn’t a major problem. After you’ve finished debugging, and are ready to compile your completed application, you should remove the MARKER from your source file."

So what is different in 4e4th?

Case sensitivity while typing

CF is case sensitive, 4e4th is not. In the older 4e4th this was switchable, now it is an integral part of 4e4th05a.

The OK-Prompt

After interpreting a line, forth prompts OK if no error occurred. CF does a simple OK. 4e4th displays its number base as well. The format is: $hh
Prefix $ to indicate it's value is in hexadecimal. And then two digits of a hex number.
Usually you will see one of these tree codes:

`$0Aok`	ok with decimal number base.
`$10ok`	ok with hexadecimal number base.
`$02ok`	ok with binary number base.

But you can choose either BASE you like by typing `<value> BASE !`

The prompt pictures the stack by printing a dot for each value on the stack. An empty stack has none.

`$0Aok`	Empty stack, decimal number base.
`$0Aok..`	Two values on stack, decimal number base.
`$10ok...`	Three values on stack, hexadecimal number base.

Stack underflow (SUF) detection

In 4e4th ?STACK aborts compilation if a stack underflow occurs, and prints SUF, otherwise does nothing. We made it part of INTERPRET so it won't hurt that much to fool around on the stack.

Additional features and words

See the forth word glossary to learn the syntax and how to use them. The expression in brackets following a word, is a stack comment. In Forth, as you may know already, words get or put values from a data stack.

(– –)	No stack effect.

(n – –)	Word will consume number n from stack.
(– – n)	Word will leave number n on stack.
(n – – m)	Word will consume number n from stack, leaving number m.

… and so on.

WORDS

It has a stop&go feature. Press the space-bar to toggle listing, other key to abort it.
WORDS (– –)

\ “Backslash Character”

It is provided to suspend compilation till end of line. Write comments there after.
\ (– –)

WIPE

Same as SCRUB for compatibility reasons: Make a clean user flash.
WIPE (– –)

2CONSTANT GREEN RED S2 S2?

CF and 4e4th use the (mask adr --) notation to address a single bit in a memory word or byte, or in a port. So 2CONSTANT is provided to save mask and adr as one word. GREEN is such a Word. It puts mask and address of the green LED's corresponding port pin onto the stack—the pin on the Launchpad of course. RED does it for the red LED. And S2 does it for the S2-button, so you can pull it's state. S2? already does the job, providing a flag on the stack indicating the current status of S2.
These are provided to enable quick first steps in forth on the Launchpad.

2CONSTANT	(n – –) <name>
GREEN	(– – mask adr)
RED	(– – mask adr)
S2	(– – mask adr)
S2?	(– – flag)

CLRB SETB TSTB

Clear, set or test a bit in a data **word**. In old 4e4th they were called CLR ans SET. There was no TST. Now we use this CF notation as well. So this is *not different* any more.

CLRB	(mask adr – –)
SETB	(mask adr – –)
TSTB	(mask adr – –)

CCLRB CSETB CTSTB

Clear, set or test a bit in a data **byte**. Stack effect as before. In old 4e4th we named them CCLR CSET CGET. Now we use the CF notation as well, *no difference* any more.

So `GREEN CSETB` will turn the green LED on.
And `GREEN CCLRB` will turn it off.
Same with RED.

CTOGB

Toggle a bit in a byte. 4e4th has it. Let your LED's blink.
CTOGB (mask adr – –)

P1OUT P1IN

Output and input register address of port1.

P1OUT	(– – adr)
P1IN	(– – adr)

1MS

Delay of about 1 millisecond. This is done by nested loops, not a calibratable counter. So do not expect to much precision.
1MS (– –)

MS

Delay of several milliseconds. `1000 MS` is a second.
MS (n – –)

So, this is it. Have fun!
Michael

erratum: mik.kalus@gmail.com