CamelForth

CamelForth/MSP430

Version 0.2 - 9 January 2009

This is an ALPHA TEST version of CamelForth/430, an ANSI Standard* Forth for the Texas Instruments MSP430 family of microprocessors. This means that I have tested the bulk of this code for correct functioning, but you may still discover bugs. I'd appreciate hearing of any such via the contact form on this web site.

Download CamelForth/MSP430

System Requirements

As distributed, CamelForth/430 will assemble to run on the New Micros Tini430 board, which uses the MSP430F1611 processor. It assumes an 8 MHz crystal for XT2, and USART0 at 9600 baud (8,N,1) for terminal I/O.

CamelForth should be usable with any MSP430 device having at least 512 bytes of RAM, 8K of ROM, and one USART.

CamelForth/430 is written to be assembled with the IAR Systems MSP430 Workbench "Kickstart", which can be downloaded from the TI web page.
Download MSP430 Workbench

Note that the files deps430.s43 and hilvl430.s43 are INCLUDEd from the core430.s43 source file; they are not linked separately. The Workbench project should contain only the files vecs430f1611.s43, init430f1611.s43, and core430.s43. You will also need to use the provided linker control file, and not the default provided by IAR.

Building CamelForth using the IAR Workbench

I assume that you have already installed the IAR MSP430 Workbench.

1. Create a new working directory. For this example, "cf430".
2. Extract the files from camel430-0.2.zip into the new directory.
3. Launch the MSP430 Workbench. In the Embedded Workbench Startup window, click "Create new project in current workspace."
4. In the Create New Project window, select the "asm" template (double-click "asm" and then click the "asm" that appears, then click OK).
5. In the Save As window, you will need to specify a directory and a Project file name. Navigate to the directory you created in step 1. For "File name:" you can type whatever you want -- for this example, "430forth". Click Save.
6. In the Workspace window, click on the Project name (e.g. "430forth") to select it. Then on the menu bar click Project, Add Files. In the Add Files window select core430.s43, init430f1611.s43, and vecs430f1611.s43, then click Open. Do NOT add deps430.s43, hilvl430.s43 -- these are included from core430.
7. On the menu bar, click Project, Options. Select "General Options", then the "Target" tab. Under "Device" select the MSP430F1611. (Click the selection button to the right of the current device, then select MSP430x1xx Family, then MSP430F1611.) Click "OK".
8. In the Options window, select "Linker", then the "Config" tab. Under

Quick Downloads
8051
8086
Z80
6809
MSP430
8052/C8051F

Contributed:
1802
8051
MSP430 (eZ430-RF2500T) **NEW!**

Offsite downloads:
Z180
Z80, ZNEO, Z180, Z380/Z382, and eZ80
Acclaim!

Links
navigator
- view links frontpage

Recent Additions
- Forth 200x Standard
  A new Forth standardization effort, now underway. Meets annually.
- EuroForth Conference
  The European Forth Conference -- apparently the only annual Forth conference still happening. Held each autumn in various European countries. Some proceedings are

"Linker command file," select the "Override default" box. Click the selection button to the right of the current file, browse to the project directory you created in step 1, select lnk430F1611.xcl, and click Open. Click OK.

9. In the Workspace window, right-click on "asm.s43", then click Remove. Confirm the removal.
10. On the menu bar, click Project, Make. You will need to specify a Workspace file name. This can be whatever you want...you can use "430forth" again. Click Save. Then the project will be built. You should see number of errors and number of warnings both "0".
11. If you have a Tini430 board connected with a parallel JTAG cable, you should be able to now click Project, Debug to download CamelForth to the target board.

Memory map

1000-10FFh: "information" Flash ROM (not currently used)
1100-12FFh: CamelForth RAM, 512 bytes (stacks, buffers, etc.)
1300-13FFh: RAMDICT - Data RAM for new variables, arrays, etc.
1400-38FFh: ROMDICT - Program RAM for new Forth definitions

4000-DFFFh: Program Flash ROM for new CamelForth definitions
E000-FFFFh: CamelForth kernel

The memory map is controlled by equates in the init430f1611.s43 assembler source file, and statements in the lnk430F1611.xcl linker control file. There are also #defines in the forth.h file that pertain specifically to Flash memory usage.

CamelForth/430 uses a split Program/Data model. New definitions are compiled into the Program (Instruction) space, as indicated by the dictionary pointer IHERE. New data structures (e.g., VARIABLEs) are allocated in the Data space, as indicated by HERE.

CamelForth/430 also features direct-to-Flash compilation. You can set the Instruction Dictionary Pointer (IDP) to an address within Flash ROM,

```
    HEX 4000 IDP !
```

and new definitions will be compiled into the ROM. There are some restrctions to this, and at least one ANSI Forth violation (see below). If you reset IDP to use Flash ROM, all the RAM after 1300h is available for data structures. (Note: future releases will default to having the program dictionary in Flash ROM. It is located in RAM now for testing.)

Development

There are TWO WAYS to write programs in CamelForth:

1. If you have CamelForth running on your MSP430 board, you can download Forth code directly to CamelForth. This lets you type new words from the keyboard, test them as they are defined, and re-define them to make changes. Or you can edit an ASCII text file, and use a program such as Hyperterminal to send this file over the serial port to your MSP430. It can take a few seconds to compile each line, so be sure to leave plenty of delay after the line. Also be sure that no line exceeds 80 characters.

2. You can add your code to the assembler source files. This requires you to convert your Forth code to assembler code. To show how this is done, every high-level Forth word in the file is shown with its equivalent Forth code in a comment. Be especially careful with control structures (IF..ELSE..THEN, BEGIN..UNTIL, DO..LOOP, and the like), and with the Forth word headers. For this option it is recommended that you create a new .s43 assembler file, and INCLUDE it at the end of core430.s43. This is necessary to preserve the dictionary linking between your new definitions and the kernel definitions.

available on-line.

Latest Forum Posts

Posted by Jack
John wrote ...I'm running IAR v4.11A and see no co[more ...]
05 Dec : 13:24

Posted by John
Glad to see autostart for CamelForth/430 being dis[more ...]
05 Dec : 12:23

Posted by John
Jack wrote ...The main reason I prefer C::B is tha[more ...]
05 Dec : 12:20

Posted by Brad R
I think what I'd prefer to do is what the C8051F C[more ...]
05 Dec : 08:12

Posted by Jack
Been looking on how things are done in other house[more ...]
04 Dec : 11:40

Posted by Brad R
As a security measure I've disabled the "public up[more ...]
04 Dec : 06:14

Posted by Eric Smith
Since I prefer to use an assembler that is Free So[more ...]
04 Dec : 02:54

Posted by Jack
The main reason I prefer C::B is that I'm used to [more ...]
02 Dec : 18:38

Posted by John
Jack wrote ...I tend to think of Eclipse as an alt[more ...]
02 Dec : 16:17

Posted by Jack
I tend to think of Eclipse as an alternative to C:[more ...]
01 Dec : 20:24

Reassemble core430.s43, and download to the MSP430 board, then test. This is a much slower process, and is best saved for the final stage when you have a tested & debugged program that you want to put in PROM.

Future releases will have an "autostart" feature to let you download and compile Forth code directly to nonvolatile Flash ROM, and then execute that code on a CPU reset.

Direct-to-Flash Compilation

CamelForth/430 can compile source code directly into the MSP430's Flash memory. No special action is required; merely set the Instruction Dictionary Pointer to a location within Flash ROM. For example, to begin compiling code at the start of Flash ROM on the MSP430F1611, type

```
    HEX 4000 IDP !
```

To store data into Flash ROM, you can use the Forth words

I! (u adr --) store a 16-bit cell in Flash
IC! (c adr --) store an 8-bit byte in Flash
I, (u --) append a cell to the I dictionary
IC, (c --) append a byte to the I dictionary
D->I (src dst n --) copy from RAM to Flash

NOTE: When using IC, remember that the MSP430 requires that cell fetches and stores occur at even addresses. Appending single bytes can cause the dictionary pointer to become unaligned (odd address) and can cause subsequent data to be read incorrectly.

These operators are "smart" in that they will work correctly with addresses in either RAM or Flash ROM, and they will refuse to overwrite the CamelForth kernel (locations E000-FFFF). Remember, though, that a Flash location can be written only one time. NOTE ESPECIALLY that writing a Flash location more than once can violate the MSP430 specs and damage the chip.

D->I is also "smart" in that it will attempt to use word writes, rather than byte writes, to minimize the total number of Flash write cycles performed. It will correctly handle even or odd source addresses, even or odd destination addresses, and even or odd lengths.

To erase Flash ROM, use the Forth word

FLERASE (adr n --) erase a range of Flash memory

The stack effect of FLERASE is the same as that of ERASE, but the effects are somewhat different:

1. Erased Flash will contain FFh bytes, not 00.
2. Main Flash (4000-DFFF) is always erased in 512-byte segments.
3. Information Flash (1000-10FF) is erased in 128-byte segments.
4. FLERASE will not operate below 1000h or above DFFFh.
5. Do not use FLERASE with RAM; it is NOT "smart" about RAM. If you attempt to FLERASE an area of RAM, it will write the cell 0FFFFh to every 512th location.

FLERASE will bascially loop, 512 bytes at a time, until 'n' is exceeded. This may have unexpected results. For example:

HEX 4133 10 FLERASE will erase 4000-41FF.
HEX 4133 1FF FLERASE will also erase 4000-41FF.
HEX 4133 200 FLERASE will erase 4000-43FF.

To preserve your sanity, it is best to always use an 'adr' which is aligned to a Flash segment, and an 'n' which is a multiple of 512 bytes (or 128 bytes if erasing Information memory).

Welcome
Username:
Password:
Login
☑ Remember me

[Signup]
[Forgot password?]
[Resend Activation Email]

CamelForth/430 supports the ANS Forth word MARKER, which automatically erases Flash memory that you have used. If you put the phrase

```
MARKER name
```
where any "name" can be used

at the beginning of your code, CamelForth will align IDP to the next available Flash segment, and will create a word "name" that will erase memory back to the marked point. To be specific, "name" will
restore IDP, DP, and LATEST (the dictionary head) to the values they had before MARKER was executed, thus "unlinking" all definitions following "name", and will then erase all Flash used by the definitions
following "name". Note that the MARKER word "name" will also erase itself.

* Non-ANSI CREATE and DOES>

CREATE..DOES> will not work correctly in a direct-to-Flash environment, because they require the Code Field of a defined word to be written twice. (The first time when CREATE gives it the default action "return the parameter address", and the second time when DOES> gives it a user-defined action.) To address this problem, CamelForth provides <BUILDS to be used with DOES>.

<BUILDS is the same as CREATE, except that the newly defined word has no action (its Code Field cell remains in the erased state). The word DOES> can then write the Code Field cell with the desired action. NOTE that you should not attempt to execute a word created with <BUILDS until you have performed a DOES> for that word. (An erased Code Field will normally cause a processor reset.)

Also, DOES> can only be used ONCE for a newly-defined word. This is normally not a restriction, since Forth applications that "re-DOES>" a defined word are extremely rare (typically, clever academic exercises). However, this limitation also violates the ANSI specification.

You can use CREATE..DOES>, and use DOES> multiple times, when compiling to RAM. This means that CamelForth/430 is only ANS compliant when compiling to RAM!

For "flashable" applications, simply use <BUILDS..DOES> instead of CREATE..DOES, and your programs should work. (Though they won't be ANSI Standard.)

Licensing

Commercial inquiries should be directed to the author at 115 First St., #105, Collingwood, Ontario L9Y 4W3 Canada or via the contact link on this web site.