# Compiling to Flash

Stephen Pelc
MicroProcessor Engineering
133 Hill Lane
Southampton SO15 5AF
England
t: +44 (0)23 8631 441
f: +44 (0)23 8033 9691
e: sfp@mpeforth.com
w: www.mpeforth.com

## Abstract

*In the last two or three years, a number of embedded Forth systems have emerged that are self-hosted and compile directly to Flash. This paper explores some of the issues found at MPE when we implemented such a kernel for the MPE Lite edition cross-compilers. Issues for the TI MSP430 family and several ARM implementations are discussed.*

## Introduction

With the vast quantity of extremely low-cost hardware provided by semiconductor manufacturers comes an attitude that there should be software at an equivalent price. For a third-party toolmaker such as MPE, there's no money in this. The race to the bottom is asymptotic to zero.

Why on earth should any compiler vendor give tools away? The only answer is to expose students, hobbyists and evaluators to good-quality tools. There's a huge number of free (of cost) Forth systems available, but to professional eyes the vast majority of them are poorly implemented and woefully documented. Overall such Forth systems damage rather than enhance the reputation of Forth.

Schools in the UK have ridiculously small budgets for electronics and technology projects, so the Lite compilers are also part of our contribution to school science and technology education.

In designing the free-of-charge Lite Forth for our cross compilers, we had to be careful not to damage our own market. This achieved in several ways:

1. The Lite Forth kernel is not compatible with the standard MPE PowerForth kernel. The Lite Forth kernel compiles directly to Flash and is subject to change.
2. The Lite Forth kernel is not ANS or Forth200x compliant. It has changes to meet the requirements of compilation to Flash.
3. The Lite Forth target code supports a restricted range of CPUs and target hardware.
4. The Lite Forth cross compiler is limited to producing no more than a certain amount of code, 16 kb for an MSP430 and 64 Kb for a Cortex M0.
5. The Lite Forth compiler may not be used for commercial purposes.

The Lite compilers are available for the MSP430 and the ARM Cortex-Mx CPUs. The MSP430 is ideal for school and low-power use, but the low-power advantage over a Cortex-Mx is much less than may be anticipated. The Cortex-Mx CPUs are the current CPU of choice – they cost no more than 8 bit CPUs, and the ease of programming 32 bit CPUs reduces time to market.

## The nature of Flash

There is wide variation among Flash devices in terms of how they are programmed. What is common is that they erase to all bits set to '1', and you can only program a '1' bit to a '0'. In order to set a '0' bit back to a '1', you have to erase a range of memory known as a page or sector, which range in size from 128 bytes to 64 kbytes. In some chips, there are sectors of varying size.

In simple chips such as the MSP430 CPUs, you perform a simple operation to permit writing and then write byte by byte to an erased sector. At a later time, you can rewrite a byte if you just change '1' bits to '0' bits. In others, such as many ST devices, Flash programming is performed through a Flash controller peripheral which has additional requirements, e.g. that bytes to be programmed are set to 0xFF, all bits set.

The important thing about all of this is that you can only program a Flash location once without encountering restrictions. These restrictions have an impact on the Forth dictionary structure and Forth notation.

## Direct compilation to Flash

When we did Forth Lite for the MSP430, we were using a simple Flash system that could rewrite '1' bits to '0' bits. The major issues in writing a Forth that compiles directly to Flash are the dictionary header layout and compilation of forward branches.

### *Dictionary header layout*

In a traditional Forth, there are two flag bits that are changed during compilation or even after compilation. These are the dictionary visibility bit ("smudge" bit) and the **IMMEDIATE** bit. The dictionary visibility bit turns out to be problematic, as some code, e.g. the common words **HIDE** and **REVEAL**, imply that this bit can be changed twice, which contradicts our rule that we can only change a '1' bit to a '0' bit. The solution to this was to remove the visibility bit completely, and only to link the word into the dictionary thread when we were satisfied that the word was correct. Hence the link field is initialised to all ones.

We can deal with the **IMMEDIATE** bit by inverting it so that a '1' bit indicates non-immediate and a '0' bit indicates that the word is immediate. Another change makes the implementation even simpler. The traditional phrase

```
: foo  …  ; immediate
```

is replaced by

```
imm : foo  … ;
```

which means that the **IMMEDIATE** bit is known when the header is constructed.

### *Forward branches*

We can deal with forward branches by setting the branch target offsets to all '1' bits so that a typical branch looks like $opc111, $11111111.

### *Ah, but …*

This all worked fine on an MSP430 which has what might be called a classical Flash controller. However, the whole plan fell apart on a couple of ARM Cortex parts. For one family, you can only program the flash in minimum units of 16 bytes on a 16 byte boundary. For another, you can only modify bytes that have all bits set.

It was time for a rethink and to understand what we were really trying to achieve with this Forth kernel. The main requirement is to be able to cross-compile the kernel itself, and then to be able to compile as much code as is wanted using the target hardware. In our world, the cross compiler generates highly optimised code, but we are not really concerned with the

quality of the code created by the target. After all, you can always upgrade the Lite compiler. A secondary initial aim was to use a common code base for the MSP430 and Cortex devices. We also asked ourselves if we actually wanted to increase the range of Lite compilers beyond MSP430 and ARM Cortex, and we decided that we do not.

By not using a common code base, we could simplify the 16 bit target for the MSP430. We also freed up what we could do with the 32 bit targets. A typical low cost 10 Euro board for an ARM Cortex-M0 has 128 kb Flash and 16 kb RAM. Thus we continued with the "classical" Flash assumption for the MSP430, and permitted ourselves to use a more complex approach for the 32 bit targets.

### *Compilation to a RAM buffer*

After examining a variety of approaches such as generating a linked list of partially compiled bytes and rewriting them, we concluded that this would lead to having to special-case each ARM part with a different peculiarity in its Flash controller. Instead, we would take the simple approach of compiling the code to a temporary RAM buffer and then copying it to Flash when complete. We use special versions ( **`C!C W!C`** and **`!C`**) of the store operations which take the Flash address but actually store the data in a buffer in RAM. Words that finish compilation such as **`;`** flush the buffer to Flash. This approach has the advantage of requiring almost no change to on-chip compilation and requires the least sacrifice of Flash.

## Conclusions

Direct compilation to Flash is tedious because it can be affected by the minutiae of specific chips. However, there is sufficient commonality that it can be done using a small number of techniques – a single technique is not enough.

We are often reminded in Forth that we should only solve the problem at hand; we should not over-generalise. Compilation to Flash is one such class of problem. In particular, our desire to use a common code base for 16 and 32 bit systems conflicted with reality. Now we use separate code bases.

## Acknowledgements

Dirk Bruehl persuaded me that an educational Forth is important. Michael Kalus provided editorial comments for the Lite compilers. Juergen Pintaske stubbornly prodded me to do more and reminded me what can be achieved with a solderless breadboard.