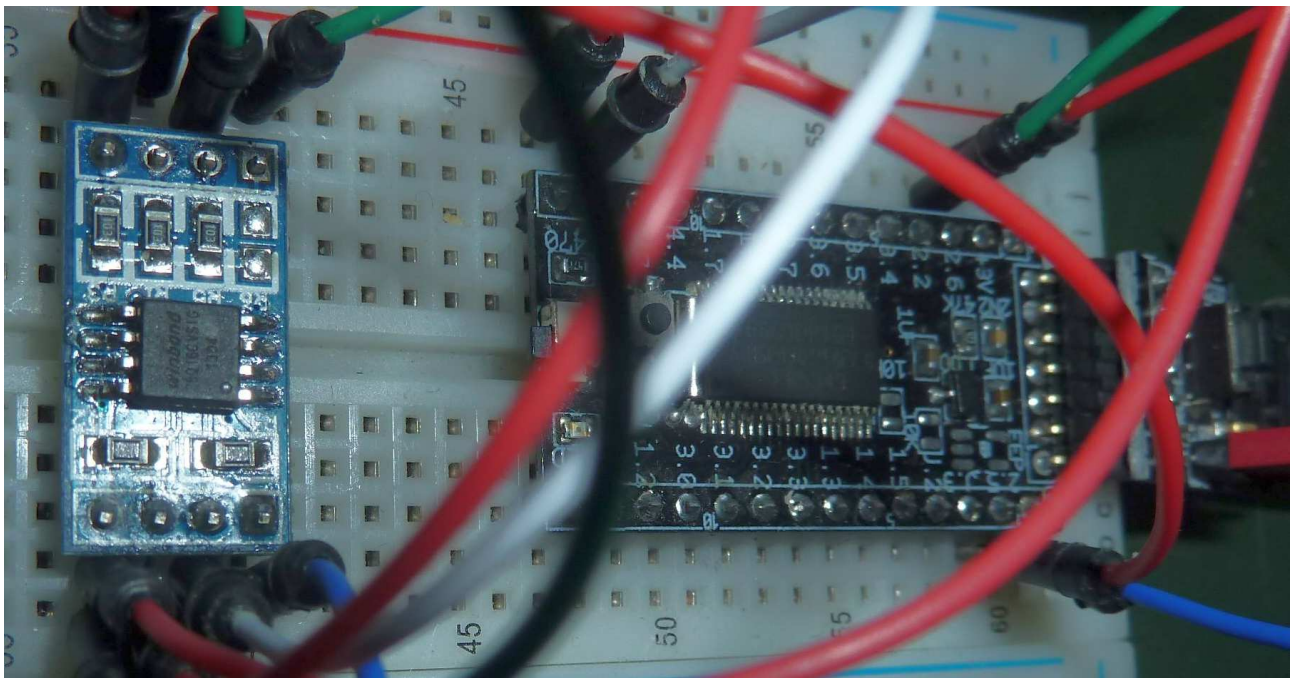
Een file systeem voor serieël Flash geheugen

Voor de BSL programmer is een massa opslag nodig. Dit voor de opslag van meerdere (noForth) binaries. De eerste keus was I2C FRAM, maar dat is relatief traag en prijzig. Er lag nog een W25Q16 in de doos "te doen". Dat is een 16 Mbit Flash geheugen met SPI-interface.
Daar passen meer dan voldoende (ongeveer 200) noForth binaries in. Het is snel (gebruikt een 16 Mhz SPI klok) en goedkoop (€0,40) en zit in een kleine 8-pins SO behuizing.

Impressie van de W25Q16

- 2 Mega Bytes Flash
- Lezen als bytes, woorden, etc.
- Schrijven gebruikt 256 Byte sectoren
- Wissen gaat met 4 kByte sectoren of groter
- Snel (max 130 Mhz klok) en een laag stroomverbruik



Micro Launchpad FR5 met W25Q16

De SPI en Flash commando set:

De SPI routines zijn over genomen van het Egel project, alleen de I/O-adressen hoefden te worden aangepast.

Voor het gebruik van Flash zijn deze commando's nodig	
Commando	Funktie
02	Schrijf een 256 byte sector naar Flash
03	Lees data uit Flash (een of meer bytes)
05	Lees status register 1
06	Sta schrijven en wissen toe
20	Wis een 4 kByte Byte sector
60	Wis de hele Flash chip
90	Lees chip ID en fabrikant code

Let op er zijn veel meer commando's maar met deze set kunnen we goed uit de voeten. Een paar code voorbeelden:

```
: WRITE-ON     ( -- )      6 {spi spi} ;

: STATUS1      ( -- b )    5 {spi  spi-in  spi} ;
: BUSY         ( -- )      begin  status1 1 and 0= until ;

: CHIP-ERASE)  ( -- )      write-on  60 {spi spi}  busy ;
: CHIP-ERASE   ( -- )      0 #sector x!  chip-erase) ;

: ADDR-FLASH   ( da -- )   spi-out  split spi-out spi-out ;
: FC@          ( da -- b ) 3 {spi  addr-flash  spi-in  spi} ;

\ Fill buffer +b from sa, the address of a 256 byte sector
: READ-SECTOR  ( sa +b -- )
    >r  03 {spi  addr-sector  r> buffer
    100 0 ?do  spi-in  over i + c!  loop  spi}  drop ;

\ Write buffer +b to sa address of a 256 byte sector
: WRITE-SECTOR  ( sa +b -- )
    >r  write-on  02 {spi  addr-sector  r> buffer
    100 0 ?do  count spi-out  loop  drop  spi}  busy ;
```

Het file systeem:

De files worden opgeslagen in Flash geheugen, voor de administratie wordt de FRAM vanaf 10000 gebruikt.
De MLP FR5 heeft er daar 16 kByte van.

De structuur van de file administratie:

Directory				
File naam	Start sector	Lengte	Token	ID string
RAMBSL	0	10	DROP	First ....
V2x55	10	20	PROGRAM-FR	NoForth ....
Readme	30	10	SHOW	FAT Readme
Etc.	..	..	..	..

De Filenaam heeft ook een aparte structuur:

File naam												
Count		String										
0	6	R	A	M	B	S	L	-	-	-	-	-
1	5	V	2	x	5	5	-	-	-	-	-	-
3	6	R	e	a	d	m	e	-	-	-	-	-

Het hoge nibble van de lengte byte bevat ook een van 16 file types.

```
\ Search directory for "name"
: FIND>     ( "name" -- entry-addr true | false )
    bl-word
    xdir x@ 0 ?do
        dup  0 'entries  i #entry *  m+  s=
        if  drop  i 'entries true  unloop exit  then
    loop  drop  ."  File not found "  false ;
```

Het lezen van files:

Het schrijven van Flash geheugen is alleen per 256 bytes sector mogelijk. Een RAM buffer wordt gebruikt als tijdelijke opslag.
De MLP FR5 heeft 2 Kbyte RAM en heeft daarmee voldoende ruimte voor een paar buffertjes.

ASCII files worden opgeslagen zoals ze zijn met op het eind van de regel een CR en LF. Het einde van de file wordt gemarkeerd met een 0.

Intel HEX files of TI TXT files worden opgeslagen als een of meerdere binary strings. Een stuk geheugen wordt als één binary string opgeslagen als het geheugen doorloopt. Zodra een HEX file een adres sprong heeft wordt een nieuwe string begonnen. Ook hier wordt het einde van de file met een nul gemarkeerd.

```
@)dir
 0 T: RAMBSL        0 10  First working F149 BSL
 1 H: V2x55        10 20  noForth V2x55 100519
 2 A: Readme       30 10  FRAM FAT system with Flash disk
Free 2032 kBytes  OK.0
@)  OK.0
@)  OK.0
@)use> V2x55
 1000:  0 18 28  0 C0 9D 1A 9E 2E 9E 44 9A 24 9E B6 9D |  (     . D $   |
 1010: 5C 9E 54 9B B2 94 28 84 6E 84 48 84 3E 80 6C 9E |\ T   ( n H > l |
 1020: 3E 20 E0 2E 30 2F 80 2F  7  0 10  0 32 18  E  0 |>  .0/ /    2   |  OK.0
@)
```

```
: B,    ( b -- )
   #n 100 = if
     buf 1 !data  -1 0 !data  buf 1 xor to buf  0 to #n
   then
   buf buffer #n + c!  incr #n ;  \ Store byte

: <PATCH>   ( -- )      \ Patch length in saved sector
    2 @data if                  \ Buffer used?
        ch * emit
        3 @data 2 read-sector  \ Get sector in spare buffer
        2 @data  4 @data 2 buffer +  ! \ Patch length
        3 @data 2 write-sector  0 2 !data \ Save sector again
    then ;
```

Hoe ziet een file er uit:

Omdat we naar noForth slechts een simpele karakter interface hebben, via een terminal programma, moeten files aangepast worden.

Dit is een verknipte HEX file:

```
c" noForth C2x55 190513"    ' program-fr    INTELHEX: C2X55

:20180000C09D1A9E2E9E449A249EB69D5C9E549BB29428846E8448843E806C9E3E20E02E27
:08182000302F802F070010009B

:0E183200CF070006F40006080102040051551D

:20FF7E0000137EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFFFD
:20FF9E007EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF73
:20FFBE007EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF53
:20FFDE007EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF33
:02FFFE00529916
:00000001FF
```

En zo ziet een aangepaste ASCII of Forth file er uit:

```
c" Sector buffer pattern demonstration"    ' show    FORTH: DEMO1

( Fill sector buffer with a binary pattern )

inside

\ Set patterns in sector buffer
: >BUF       ( buf -- )      to buf  0 to #n ;    \ Set dest. buffer
: PATTERN1   ( buf -- )      >buf  0 FF do  i b,  -1 +loop ;
: PATTERN2   ( buf -- )      >buf  100 0 do  i b,  loop ;

: DEMO       ( u -- )        0 ?do  i . loop ;

forth

\ End

;;;
```