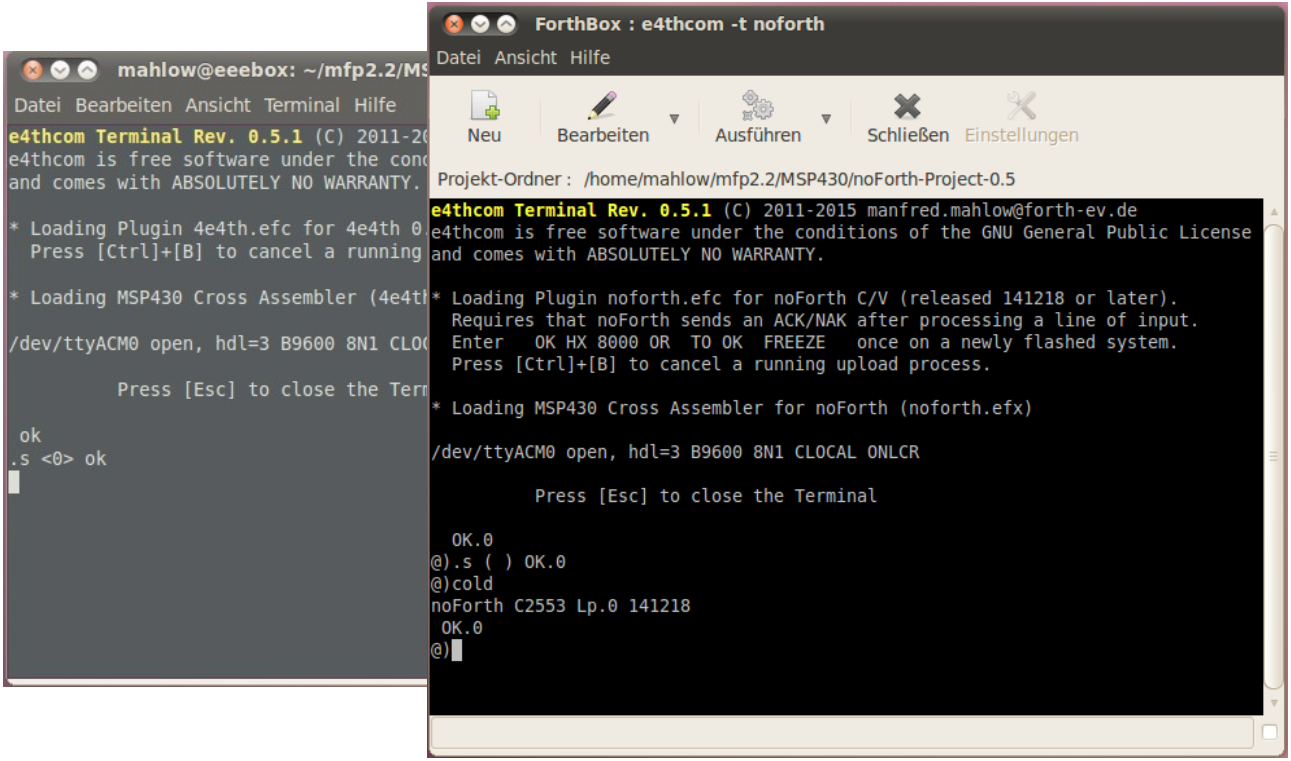
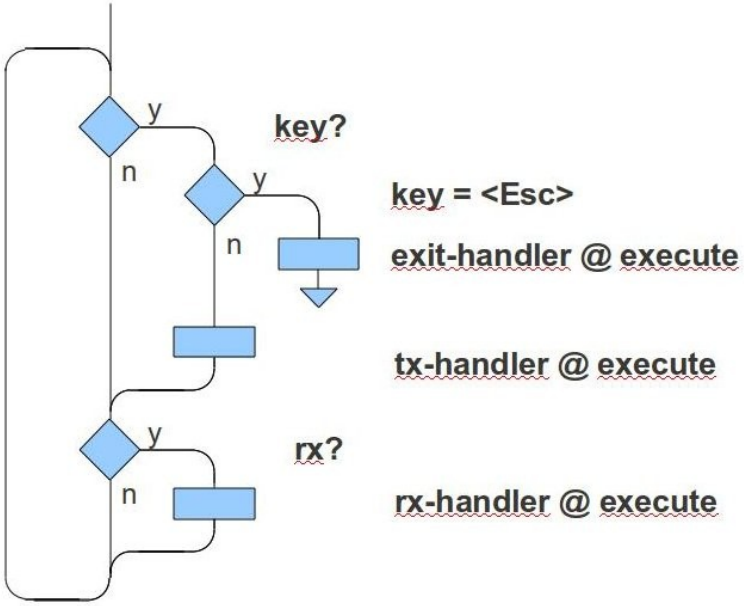
e4thcom Terminal (in a ForthBox)

- Target-specific Plugins (7)
- Cross-Assembler Interface (12)

manfred.mahlow@forth-ev.de


mahlow@eeebox: ~/mfp2.2/MS
Datei Bearbeiten Ansicht Terminal Hilfe
e4thcom Terminal Rev. 0.5.1 (C) 2011-2(
e4thcom is free software under the conc
and comes with ABSOLUTELY NO WARRANTY.
* Loading Plugin 4e4th.efc for 4e4th 0
Press [Ctrl]+[B] to cancel a running
* Loading MSP430 Cross Assembler (4e4t
/dev/ttyACM0 open, hdl=3 B9600 8N1 CLO
Press [Esc] to close the Ter
ok
.s <0> ok
ForthBox : e4thcom -t noforth
Datei Ansicht Hilfe
Neu
Bearbeiten
Ausführen
Schließen
Einstellungen
Projekt-Ordner : /home/mahlow/mfp2.2/MSP430/noForth-Project-0.5
e4thcom Terminal Rev. 0.5.1 (C) 2011-2015 manfred.mahlow@forth-ev.de
e4thcom is free software under the conditions of the GNU General Public License
and comes with ABSOLUTELY NO WARRANTY.
* Loading Plugin noforth.efc for noForth C/V (released 141218 or later).
Requires that noForth sends an ACK/NAK after processing a line of input.
Enter OK HX 8000 OR TO OK FREEZE once on a newly flashed system.
Press [Ctrl]+[B] to cancel a running upload process.
* Loading MSP430 Cross Assembler for noForth (noforth.efx)
/dev/ttyACM0 open, hdl=3 B9600 8N1 CLOCAL ONLCR
Press [Esc] to close the Terminal
OK.0
@).s ( ) OK.0
@)cold
noForth C2553 Lp.0 141218
OK.0
@)

e4thcom Terminal: Top Level Flow Chart



manfred.mahlow@forth-ev.de

tx-handler (c oid –)

| → tx.1 (c oid –)

| directs input to the target if a new line does not start with #

| → tx.2 (c oid –) sends c to the target until c = ^CR ([Enter])

| directs input to the cmd-buffer if a new line starts with #

| → tx.3 (c oid –) appends c to the cmd-buffer until c = ^CR

| → cmd-handler (oid –)

| Input = #include name

| → include-handler (a1 u1 a2 u2 oid – f)

| Input = #require name

| → require-handler (a1 u1 a2 u2 oid – f)

a1,u1 = name a2,u2 = path to cwd

```
Input = #include name [Enter]
| → cmd-handler ( oid – )
    | → include-handler ( a1 u1 a2 u2 oid – f )
        | → upload ( a1 u1 a2 u2 oid – f )
            | → upload-file ( fid oid – f )
                | → upload-lines ( oid – f )
                    | → refill ( – f )
                    | → source ( – a3 u3 )
                    | → preproc-line ( a3 u3 oid – a4 u4 f )
                    | → upload-line ( a4 u4 oid – f )
                    |   | → upload-string ( a4 u4 oid – f )
                    |   | → done? ( oid – f )
                    |       | → plugin @ execute ( oid – f )
                    |           | → noname defined in <target>.efc
```

```
preproc-line ( a3 u3 oid – a4 u4 f )
```

Supported terminal directives:

- `#include`
- `#require`
- `\ comment`
- `{ multi-line
  comment }`
- `\\  ignore rest of file`

e4thcom Terminal (in a ForthBox)

- Target-specific Plugins
- Cross-Assembler Interface

manfred.mahlow@forth-ev.de

```
Input = #include name [Enter]
| → cmd-handler ( oid – )
    | → include-handler ( a1 u1 a2 u2 oid – f )
        | → upload ( a1 u1 a2 u2 oid – f )
            | → upload-file ( fid oid – f )
                | → upload-lines ( oid – f )
                    | → refill ( – f )
                    | → source ( – a3 u3 )
                    | → preproc-line ( a3 u3 oid – a4 u4 f )
                    | → upload-line ( a4 u4 oid – f )
                    |   | → upload-string ( a4 u4 oid – f )
                    |   | → done? ( oid – f )
                    |       | → plugin @ execute ( oid – f )
                    |           | → noname defined in <target>.efc
```

Plugin example: Target sends ^ACK/^NAK on OK/Error and no prompt.

```
:noname ( oid -- flag )
 \ Wait while the target evalutes a line of uploaded source code. Return true
 \ on error or if [Ctrl] [B] was pressed. Otherwise return a false flag.
    >self  white bright letters  ( self rx-buf erase ¹)
    begin
      \ receive a char from the target, exit if the user pressed [Ctrl][B]
      self rx?break ( c f ) if drop true exit then  ( dup self rx-buf append ¹)
      ( c ) dup bl <
      if \ control char received
        ( c ) dup ^ACK = if drop false exit then
        ( c ) dup ^NAK = if drop true  exit then
      then
      ( c ) self ?emit
    again
; terminal plugin !
```

¹) only required for the bidirectional cross-assembler interface

manfred.mahlow@forth-ev.de

Very often OK/Error detection is not that easy. Then a) analysing received messages or b) timeout monitoring or both might be required:

a) words for message analysis

```
self rx-buf erase ( -- )
self rx-buf append ( c -- )
self rx-buf compare$ ( a u -n|0 -- f )
```

b) timeout monitoring

```
self timeout ! ( u -- )     \ timeout after ~ u ms
self rx?timeout ( c f )     \ f = true on timeout
self timeout error ( -- )   \ prints error message
```

Examples: See <target>.efc files enclosed in the e4thcom distribution.

Writing a plugin requires knowledge about the targets response. An easy way to get this information is to use the test.efc plugin that comes with the e4thcom distribution:

a) connect the target to the computer
b) make the e4thcom directory the currend working directory (cwd)
c) start the terminal with ./e4thcom –t test –d <device> –b <baudrate> [Enter]
d) include the file test from cwd with #i test [Enter]

For noForth the program output will look like this:

```
\ Target response in interpret mode if no error occures:
base  OK.1^0D^0A@)^06
drop  OK.0^0D^0A@)^06
...
\ Target response in interpret mode if an error occures:
X ^0D^0A Msg from INTERPRET \ Error # D7CD ^0D^0A@)^15
…
```

Control chars are printed as hex numbers with prefix ^.

e4thcom Terminal (in a ForthBox)

- Target-specific Plugins
- Cross-Assembler Interface

manfred.mahlow@forth-ev.de

New since **Rev 0.4.4 : Unidirectional Cross-Assembler Interface**

New terminal directives:

- **code** („name“ --)
  ```
  \ Upload the string „code name“ to the target and activate the cross
  \ assembler mode (code must be defined in the target dictionary).
  ```

- **end-code** (--)
  ```
  \ Upload the string „end-code“ to the target and deactivate the cross-
  \ assembler mode (end-code must be defined in the target dictionary).
  ```

- **\xas** (ccc<eol> --)
  ```
  \ Evaluate ccc, delimited by <eol>, in the cross-assembler context.
  \ (Default base is hex.)
  ```

New resource-oriented cross-assembler words:

- **equ** („name“ x –)
  \ Create a constant (in the cross assembler dictionary) if it doesn't
  \ exist.
  \ Example: \xas 01 **equ** BIT0 021 **equ** P1OUT

- **MCU:** („name“ –)
  \ Load a file name with MCU resource identifiers if it's not yet loaded.
  \
  \ Example: \xas **MCU:** MSP430G2553

An appropriate target specific cross-assembler, written in Forth, may be included by the Plugin <target>.efc (see noforth.efc for an example).

New since **Rev 0.5.1 : Bidirectional Cross-Assembler Interface**

New cross-assembler words for the application programmer:

- **x[** („ccc]“ -- x)
  ```
  \ Upload ccc, delimited by ], to the target. Wait for the target to
  \ evaluate ccc and then return the value from the targets TOS on the
  \ cross-assembler stack.
  \
  \ Example:   code hex ( -- )
  \                     10 # x[ base ] & mov next end-code
  ```
- **label** LB[01,05]
  ```
  \ Assign the targets next free code address to one of the global labels
  \ LB01...LB05, predefined in the cross-assembler dictionary.
  ```
- \xas **export** name1 name2 ... nameN
  ```
  \ Copy bit or register identifier(s) (constants) from the cross-assembler
  \ dictionary to the targets dictionary.
  ```

The code directive is less restrictive now:

Any line that starts with <spaces>code is now send to the target and the cross-assembler is activated. So more words, starting a code definition, can be defined and used on the target, e.g.:

- **code-label** („name" -- code-sys)
  \ Create a label name (-- a) for a code sequence and activate the cross-
  \ assembler.

- **code-begin** (-- code-sys)
  \ Activate the cross-assembler (without creating a header or label).

New interface words for the system programmer:

- **xeval** (h: a u --) (t: i*x -- j*x)
  \ Upload the string a,u to the target and wait for the target to process
  \ it. Do not display any chars received from the target.

- **xeval&pop** (h: a u -- x) (t: i*x -- j*x)
  \ Upload the string a,u to the target, wait for the target to process it,
  \ receive a hex number string and place the number on the hosts stack.
  \ Do not display any chars received from the target.

Based on this two words, all other words required to integrate an assembler into the terminal, can be defined.

Integrating a given assembler as a cross-assembler

Words for memory access need to be changed, e.g.

assembler	cross-assembler
`,` (x --)	**x,** (h: x --) (t: --)
! (x a -- x)	**x!** (h: x a --) (t: --)
@ (a -- x)	**x@** (h: a -- x) (t: --)
here (-- a)	**xhere** (h: -- a) (t: --)

and different cell size needs attention

? Bits/Cell	**terminal : 32 Bits/Cell**
	target : ? Bits/Cell

For further information please see the .efx files in the e4thcom distribution.

Note: The terminals FORTH-System is case sensitive.

manfred.mahlow@forth-ev.de