Anbindung eines Forth-Targets an einen Forth-Host

Rechner z.B. PC steuert ein Target System

z.B. Evaluation Board

Grundsätze

- Für den Host erscheint der Target als integraler Teil seines eigenen Systems.
- Der Target enthält nur die unbedingt notwendigen Programm-Teile, die für seinen eigenen Ablauf notwendig sind
- Alle Programmteile, die nur zur Konfiguration, Verwaltung, Compilierung, Programmierung usw. des Targets benötigt werden, liegen im Host.

Folgerungen

- Die komplette Bedienung erfolgt vom Host aus. Der Target ist über eine Schnittstelle mit dem Host verbunden (z.B. RS232).
- Damit ist der Host Master, der Target ist Slave
- Der Host enthält Compiler, Interpreter, Assembler, Editor, Disk-System.
- Er enthält auch die Header („Name“ und „Link“)

System

- Die Unterscheidung, ob ein Befehl vom Host selbst oder vom Target ausgeführt wird liegt im Vokabular
- Das Vokabular (Vocabulary) erhält neben den in Forth üblichen Daten (CURRENT, VOC-LINK) ein Target Flag (0 = Target, >0 = Host)
- Damit kann der Interpreter erkennen, ob der Befehl im Host oder Target liegt

Arbeitsablauf bei der Ausführung eines Target-Befehls durch den Host

Theorie

3. Der Host sendet die Parameter zum Target.
4. Der Host sendet den Target-Befehl zum Target.
5. Der Target führt den Befehl aus.
6. Der Target sendet die bearbeiteten Parameter zurück zum Host.

Arbeitsablauf bei der Ausführung eines Target-Befehls durch den Host

Praxis

- Da der Host keine Information darüber hat, wie viele Parameter der Befehl braucht sendet er den gesamten Parameter-Stack!
- Zusätzlich legt er als obersten Parameter die Code-Adresse (cfa) des auszuführenden Befehls auf den Stack (als TOS).
- Der Target benutzt den obersten Parameter als Befehl und führt ihn aus.
- Er benutzt dabei so viele Parameter, wie er für den Befehl braucht, die weiteren lässt er unbearbeitet, legt aber gegebenenfalls Ergebnisparameter zusätzlich auf den Stack.
- Dann schickt er alle Parameter d.h. den gesamten Parameter-Stack zurück zum Host.

Arbeitsablauf bei der Ausführung eines Target-Befehls durch den Host

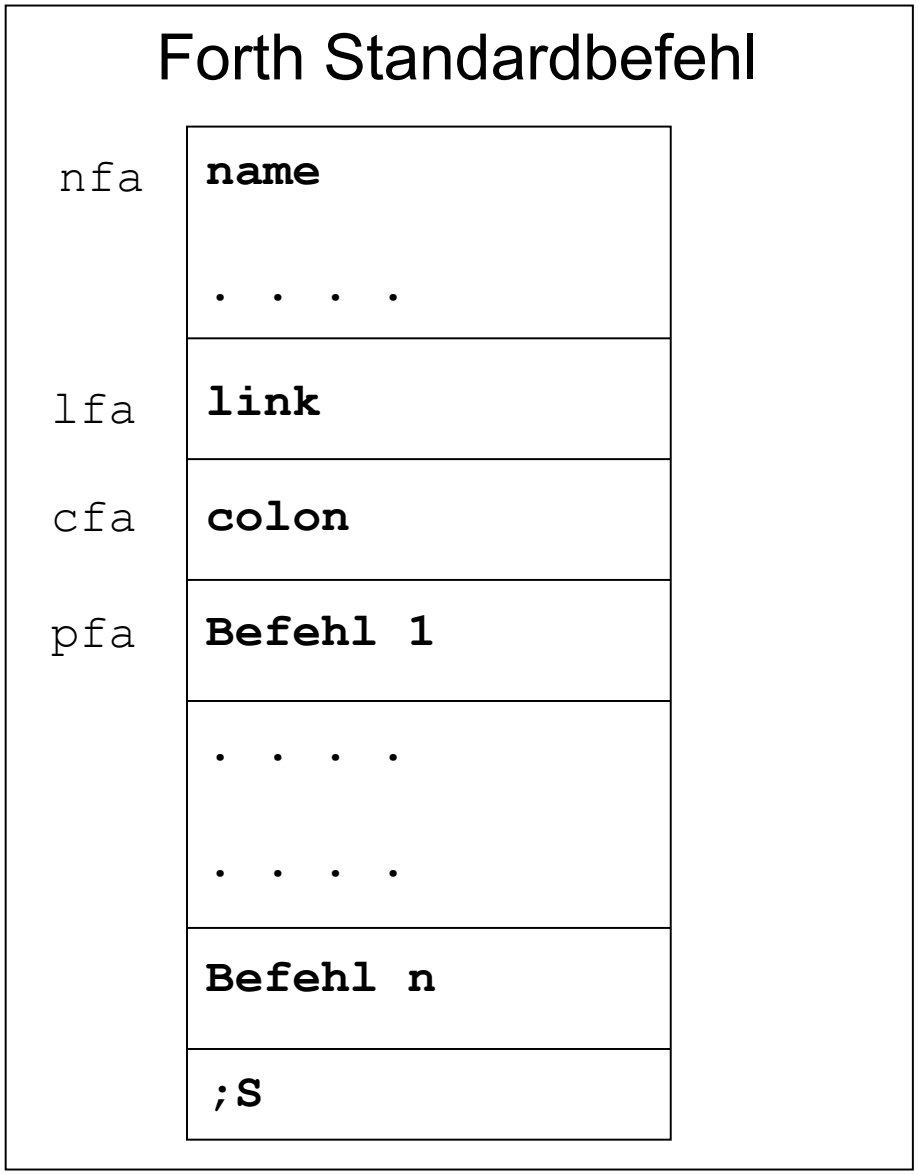
Anmerkung:

Um den Datenverkehr zwischen Host und Target nicht zu aufwendig zu machen wird in der Praxis die Anzahl der transferierten Parameter begrenzt (z.B. 8, 10 oder 16)

Arbeitsablauf bei der Ausführung eines Target-Befehls durch den Host

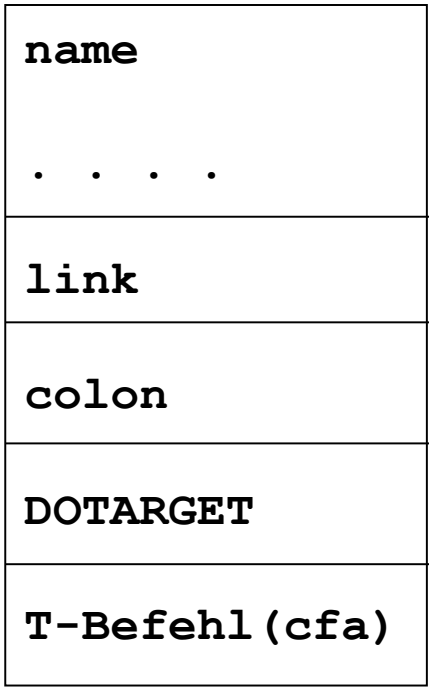
Host		Target
• arbeitet		• wartet
• sendet Parameter	=>	• empfängt Parameter
• sendet Befehl (cfa)	=>	• empfängt Befehl (cfa)
• wartet		• führt Befehl aus
• empfängt Parameter	<=	• sendet Parameter
• arbeitet weiter		• Wartet

Befehlsaufbau in Forth

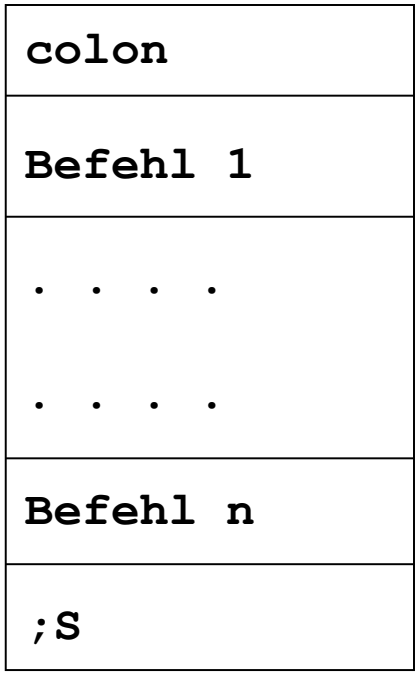


Befehlsaufbau in Forth
Target-Befehl

Target-Befehl im Host



Target-Befehl im Target



Host Kommunikations-Funktion

Host-Befehl “Dotarget”

```
: DOTARGET
  R>        Return Adresse
  @         get cfa vom Target Befehl
  PUTPAR    Parameter senden zum Target -
             Befehlsausführung
  GETPAR    Parameter empfangen vom Target -
             Ergebnis
;
```

Ablauf der Kommunikation

Übertragungsprotokoll:

Bytefolge: erst das Low-Byte dann das High-Byte (High Endian Format)

1. Byte	= 0 = Start-Byte
2. Byte	= Parameter Count (Words = 2 Bytes)
3. Byte	= 1. Parameter, Low Byte
4. Byte	= 1. Parameter, High Byte
.	.
.	.
(2n+1). Byte	= n. Parameter, low Byte
(2n+2). Byte	= n. Parameter, high Byte

Ablauf der Kommunikation

Übertragungs-Reihenfolge:

Der Stack wird immer von “unten”(Bottom of Stack BOS) beginnend gesendet!

Folge:

- Beim Senden muss der Stack von „unten“ aus „aufgerollt“ werden.
- Beim Empfangen werden die Parameter einfach nacheinander auf den Stack gelegt

Sende-Programm

Vorhanden: **`PUTCHAR ( byte -> )`**

sendet ein Zeichen (Byte)

```
: PUTPAR      ( n Parameter -> )
   0 PUTCHAR            Senden Start-Byte
   SP@ DUP              Parameter Count
   PUTCHAR              Senden Count
   BEGIN                Sendeschleife für die n Parameter
        DUP             Count
   WHILE
        DUP
        PICK            pick nächsten Parameter
        DUP PUTCHAR     Senden Low Byte
        CSWAP  PUTCHAR  Senden high Byte
        1-              decrement count
   REPEAT
   DROP
   0 SP!                Stack initialisierung
;
```

Empfangs-Programm

Vorhanden: **`GETCHAR ( -> byte)`**

empfängt ein Zeichen (Byte)

```
: GETPAR ( -> n Parameter)
   BEGIN
        GETCHAR 0=     Empfang Start-Byte + Test
   UNTIL
   GETCHAR             Empfang Parameter Count
   BEGIN               Empfangsschleife für die n Parameter
        DUP            Count
   WHILE
        GETCHAR        Empfang Low Byte
        GETCHAR        Empfang High Byte
        CSWAP OR SWAP  zusammenfassen
        1-             decrement Count
   REPEAT
   DROP
;
```

Handshake

Ein Handshake ist bei der Kommunikation zwingend notwendig, weil die Zeitdauer der Befehls-Ausführung durch den Target undefiniert ist.

Anmerkung: Ein Hardware-Handshake ist nicht sinnvoll, da viele Schnittstellen keinen Hardware-Handshake vorsehen (zB. Evaluation Kits).

Grundsatz: Der Host ist Master, der Target ist Slave.

Folge: Der Target reagiert nur, wenn er dazu vom Host aufgefordert wird.

Ablauf:

- Der Host (Master) sendet ein Byte und wartet danach auf eines vom Target.
- Der Target (Slave) wartet auf ein Byte vom Host und sendet erst dann eins.

Dieser Ablauf gilt für jedes übertragene Byte!

Handshake-Befehle

Grundbefehle für die Kommunikation mit einer Schnittstelle

`GETBYTE ( byte -> )`

empfängt ein Byte von der Schnittstelle

`PUTBYTE ( -> byte )`

sendet ein Byte an die Schnittstelles

Code ist abhängig von der Hardware, kann in Assembler oder in Forth realisiert werden

Handshake-Befehle

Host sendet ein Byte mit Handshake

```
: PUTCHAR ( char -> )
        PUTBYTE
        GETBYTE
        DROP
;
```

Target empfängt ein Byte mit Handshake

```
: GETCHAR ( -> char )
        GETBYTE
        DUP
        PUTBYTE
;
```

Host empfängt ein Byte mit Handshake

```
: GETCHAR ( -> char)
        0
        PUTBYTE
        GETBYTE
;
```

Target sendet ein Byte mit Handshake

```
: PUTCHAR ( char -> )
   GETBYTE
        DROP
        PUTBYTE
;
```

Target-Kommunikationsprogramm

Für die Programmierung des Targets bestehen 2 Möglichkeiten:

3. Der Target enthält bereits einen minimalen Forth-Kernel, der mindestens diejenigen Forth-Befehle bereitstellt, die für die Kommunikation benötigt werden.
4. Der Target enthält nur ein Basisprogramm, das ausschließlich die für die Kommunikation mit dem Host benötigten Programmteilte enthält (Minimalprogramm).

Target-Forth-Kommunikation

Die Kommunikation ist eine unendliche Schleife, sie reagiert nur auf Anweisung durch den Host.

```
: COMMUNIC ( -> )
   PORTINIT          Port-Initialisierung
   BEGIN             unendliche Schleife
       GETPAR        Empfang der Parameter + Befehl(cfa)
       EXECUTE       Ausführung des Befehls
       PUTPAR        Rücksenden des Ergebnisses
   AGAIN             Rücksprung
;
```

Die Befehle GETPAR und PUTPAR sind identisch für Host und Target, nur die Handshake-Befehle GETCHAR und PUTCHAR sind unterschiedlich

Target-Assembler-Kommunikation

```
INIT_HARD:
    ;Hardware
    ;Initialisierung-;
    Befehle
INIT_SOFT:
    LOAD RP ; RP laden
    LOAD SP ; SP laden
    LOAD IP ; IP laden
JUMP COMMUNIC
```

```
COMMUNIC:
   CALL PORTINIT
START:
   CALL GETPAR
   [Execute_TOS]
   CALL PUTPAR
   JUMP START
```

unendliche Schleife!

Processor R8C => ca. 300 Byte

Target-Assembler-Kommunikation

Theorie

Um mit einem Target zu korrespondieren (lesen, Schreiben, programmieren) werden nur 2 (3) Funktionen (Befehle) benötigt:

Byte laden	**=> C@**
Byte speichern	**=> C!**
Prozessor-Daten	**=> ORIGIN**

Mit Hilfe dieser 2 (3) Befehle lässt sich ein vollständiges (Forth-) Programm schreiben (programmieren) lesen bedienen und bearbeiten(debuggen)!

Target-Assembler-Kommunikation

Teilprogramm Execute_TOS:

```
MOV   WP,[SP+]        POP WP
CMP   WP,0            if WP=0=>ORIGIN
JMP   EQ,ORIGIN
CMP   WP,1            if WP=1=>C@
JMP   EQ,CFETCH
CMP   WP,2            if WP=2=>C!
JMP   EQ,CSTORE
JMP   EXECUTE         else EXECUTE
```

Anmerkung: Zusätzlicher Code nötig!