Vektorisierung in Forth

Claus Kühnel*

Die Programmierung in Forth besteht in der Definition neuer, auf dem aktuellen Wortschatz aufbauender Worte, was im Forth-Slang gleichbedeutend mit einer permanenten Erweiterung des Wörterbuches (Dictionary) ist. Dabei ist es nicht zwingend, dass die definierten Worte festgeschriebene Eigenschaften aufweisen müssen. Forth bietet die Möglichkeit, die Eigenschaften solcher Worte im nachhinein zu verändern.
Sinnvoll ist ein solches Vorgehen beispielsweise bei der Umschaltung von Ausgabeoperationen zwischen Bildschirm und Drucker. Im Programm wird dann nur ein Ausgabewort benötigt. Ein anderer Einsatzfall sind Routinen, die zum Zeitpunkt der Programmerstellung noch nicht festliegen und erst später definiert werden können. Vorab lassen sich diese Routinen mit einer Hilfsfunktion belegen.
Diese als Vektorisierung bezeichnete Möglichkeit soll im folgenden Beitrag anhand einer standardgerechten Forth-Version (Standard Forth-83) näher betrachtet werden.

1. Vektorisierung «von Hand»

Den Ausgangspunkt für die folgenden Betrachtungen soll der interne Aufbau einer Forth-Variablen bilden. Im Standard Forth-83 ist die Headerstruktur nicht festgeschrieben, weshalb an dieser Stelle die in vielen Forth-83-Implementierungen weiter genutzte fig-Forth-Headerstruktur (zum Beispiel LMI-Forth) betrachtet wird.

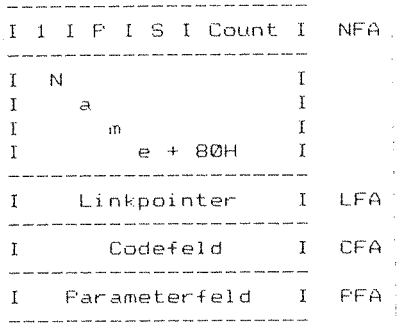

Bild 1 Fig-Forth-Headerstruktur.

Die Eigenschaften einer Variablen sind mit ihrer Definition festgelegt. Eine mögliche High-Level-Definition der Variablen lautet:

```
: VARIABLE
    CREATE 2 ALLOT
    DOES> ;
```

Während der Compilierungsphase wird mit Hilfe von **CREATE** ein neuer Wörterbucheintrag erzeugt. Im Gegensatz zu fig-Forth ist die Initialisierung der Variablen getrennt von der Definition vorzunehmen und liegt damit in der Verantwortung des Programmierers. Während der Laufzeit legt **DOES>** die Parameterfeldadresse der Variablen auf den Stack (Top-Of-Stack). Der Inhalt dieser Parameterfeldadresse ist damit für die Eigenschaften der vorstehend definierten Variablen kennzeichnend.
Bei einer «normalen» Variablen wird dieser Inhalt den numerischen Wert der Variablen darstellen. Dieser Fall stellt nichts Besonderes dar und bedarf keiner weiteren Betrachtung. Steht aber in diesem Parameterfeld die Codefeldadresse (CFA) eines Forth-Wortes, dann ist der Variablen die Eigenschaft dieses Wortes von «aussen» aufgeprägt. Der Aufruf dieses Wortes geschieht nach der Bereitstellung der Parameterfeldadresse in der Form

(pfa —) @ EXECUTE

Eine Variable, in deren Parameterfeld die Codefeldadresse eines anderen Forth-Wortes abgelegt wurde, bezeichnet man als Vektorvariable.
Ein praktisches Beispiel aus der Messdatenverarbeitung soll das Prinzip und Anwendung der Vektorisierung verdeutlichen. Über eine parallele Schnittstelle (Interface) sollen ein Analog/Digital-umsetzer angesprochen und das Umsetzergebnis vom Rechner erfasst werden. Wenn zum Zeitpunkt der Softwareerstellung noch Fragen der Hardwaregestaltung und deren Ansteuerung unklar sind, muss man dennoch die Arbeiten am geplanten Projekt nicht bis zur endgültigen Klärung aufschieben, sondern bedient sich vorerst einer Hilfsfunktion. Später kann dann eine entsprechende Aktualisierung erfolgen oder die Anpassung an weitere Baugruppen vorgenommen werden.
Im Forth-Quelltext soll nun unabhängig von der zum Nutzungszeitpunkt gewünschten Ansteuervariante die genannte Hilfsfunktion verankert werden. Mit der Definition einer Variablen, hinter der sich die Routine zum Datenaustausch mit dem Analog/Digital-Umsetzer verbirgt, hat man dazu alle Vorleistungen erbracht.

VARIABLE ^ADU

Die Variable **^ADU** ist eine ganz normale Variable, die aber durch das vorangestellte Zeichen«^» (willkürlich) als Vektorvariable gekennzeichnet wurde. Nach erfolgter Definition ist die Variable noch nicht initialisiert, das heisst, ein Aufruf in der Form

^ADU @ EXECUTE

hat im allgemeinen einen Systemabsturz (Crash) zur Folge. Erst wenn die Vektorvariable mit der noch zu schaffenden Hilfsfunktion initialisiert ist, kann der gewünschte Aufruf erfolgen.
Damit eventuell benötigte Auswerte- und Darstellungsroutinen des Programmes zur Messdatenverarbeitung bereits programmiert werden können, soll die Hilfsfunktion Daten bereitstellen, die auch vom Analog/Digital-Umsetzer herrühren könnten. Ausgehend von einem Analog/Digital-Umsetzer mit einer Auflösung von 10 Bit sind auf dem Stack Zahlenwerte im Bereich von 0 bis 1023 bereitzustellen. Hierzu kann ein Pseudo-Zufallsge-

* Dr. Claus Kühnel, Zschertnitzerstr. 52, DDR-8020 Dresden

nerator herangezogen werden, der mitunter bereits Bestandteil des Forth-Systems ist. Fehlt dieser, dann ist für den betrachteten Einsatzfall der einfache Pseudo-Zufallsgenerator nach [1] vollkommen ausreichend:

```
VARIABLE RND

HERE RND ! ( Initialisierung der Variablen RND )

: RANDOM
    RND @ 31421 * 6927 + DUP RND ! ;

: ZUFALLSZAHL ( u1 — u2 )      ( 0 <= u2 < u1 )
    RANDOM UM* SWAP DROP ;
```

Die Hilfsfunktion kann damit wie folgt definiert werden:

```
: (ADU)        ( — 0 . . . 1023 )
    1024 ZUFALLSZAHL ;
```

Zeigt der separate Test des Wortes **(ADU)** dessen Funktionstüchtigkeit, kann die Initialisierung der Vektorvariablen ^ADU derart erfolgen, dass die Codefeldadresse des Wortes **(ADU)** in die Parameterfeldadresse der Vektorvariablen ^ADU geschrieben wird.

```
' (ADU) ^ADU !
```

Nun kann die Aktivität der Vektorvariablen ^ADU durch den Aufruf

```
^ADU @ EXECUTE
```

ausgelöst werden. Im vorliegenden Fall wird auf dem Stack erwartungsgemäss eine Zahl zwischen 0 und 1023 bereitgestellt.
Nachteilig an der vorgestellten Art der Vektorisierung ist, dass in der Definition die Variable noch nicht initialisiert ist. Auf den Systemabsturz bei einem versehentlichen Aufruf wurde bereits hingewiesen. Des weiteren ist der sicher häufiger auftretende Aufruf umständlich. Mit Hilfe eines eigens für diesen Fall geschaffenen Definitionswortes lässt sich das in gewissem Umfang ändern:

```
: VEKTOR
    CREATE ['] NOOP ,
    DOES> @ EXECUTE
;
```

Die Definition einer Vektorvariablen geschieht nun in der Form:

```
VEKTOR ^ADU
```

In die Parameterfeldadresse wird bei der Compilierung die Codefeldadresse des Wortes **NOOP** eingetragen. Sollte **NOOP** im Forth-System nicht enthalten sein, dann definiert man es schnell selbst:

```
: NOOP ; ( — )
```

Das Laufzeitverhalten der Vektorvariablen ist so definiert, dass die bei Aufruf auf dem Stack zu liegen kommende Parameterfeldadresse durch ihren Inhalt (eine CFA) ersetzt und das betreffende Wort ausgeführt wird. Nach der Initialisierung mit **NOOP** nimmt das Wort ^ADU bei einem vorzeitigen Aufruf also keinen schädigenden Einfluss mehr. Da das Wort nun aber eine eigene Laufzeitaktivität besitzt, ist die Vektorisierung wiederum etwas umständlicher geworden und in folgender Form vorzunehmen:

```
' (ADU) ' ^ADU >BODY !
```

Das Programm zur Messdatenverarbeitung kann nun komplett unter der Verwendung von Pseudo-Zufallszahlen getestet werden. Problemlos kann nun sogar der Anwender seine spezielle, im allgemeinen als Primitive Word formulierte Routine zur Kommunikation mit dem Analog/Digital-Umsetzer an die Vektorvariable ^ADU übergeben.
Beide vorgestellten Arten der Vektorisierung bleiben dennoch umständlich, und erwartungsgemäss bietet Forth komfortablere Varianten, für die jedoch dieses beschriebene Prinzip die Grundlage darstellt.

2. DEFER-IS-Konstrukt (F83)

Mit der von Laxen & Perry veröffentlichten, als F83 bezeichneten Implementierung des Standards Forth-83 steht im «public-domain» eine leistungsfähige Forth-Version zur Verfügung, die in vielen Fällen weit über den Standard hinausgeht [2]. Der im folgenden betrachtete Konstrukt ist eine F83-spezifische Forth-83-Erweiterung, die aber nicht nur aus Kompatibilitätsgründen als Systemerweiterung aufgenommen werden sollte [3].
In Bild 1 ist in einem Screen der insgesamt erforderliche Quelltext dargestellt. Bevor auf die Einzelheiten eingegangen werden soll, wird die Handhabung betrachtet. Als neues Definitionswort steht nun **DEFER** zur Verfügung:

```
DEFER ^ADU
```

Durch die vorstehende Angabe ist das Wort ^ADU als sogenanntes «defered»-Wort (Vektor-Wort) definiert. Bevor diesem Wort eine eigene Eigenschaft zugeordnet wird, reagiert es bei Aufruf in der folgenden Weise:

```
^ADU
ERROR: "^ADU" undefined execution vektor
```

Diese Reaktion gibt nun den Hinweis, dass dem neuen Wort noch seine Eigenschaften mitzuteilen sind. Das Wort **IS** steht für diesen Fall bereit:

```
' (ADU) IS ^ADU
```

Das Wort **IS** kann in der vorgestellten Weise interpretativ angewendet werden, aber auch in Colon-Definitionen eingebaut sein.

```
: TEST ['] (ADU) IS ADU ;
```

Die Reaktion ist wunschgemäss, so dass die Frage nach der Realisierung des Verhaltens geklärt werden kann.
Wie in Bild 2 zu sehen ist, wurde mit **DEFER** ein Definitionswort geschaffen, das mit dem im vorangegangenen Absatz erläuterten

```
Screen #    1
( DEFER-IS-Konstrukt                                  CK  6/ 9/88)
( nach R. Zech: FORTH 83. Franzis Verlag Muenchen 1987        )

: CRASH
    TRUE ABORT" undefined execution vector" ;

: DEFER
    CREATE ['] CRASH ,
    DOES>  @ EXECUTE ;

: (IS)       ( cfa -- )
    R> DUP 2+ >R @ >BODY ! ;

: IS         ( cfa -- )
    STATE @
    IF COMPILE (IS) ELSE ' >BODY ! THEN ; IMMEDIATE
```

Bild 2 Definitionswort DEFER.

Wort **VEKTOR** deutliche Verwandtschaft zeigt. Bei der Compilierung wird hier aber nicht die Codefeldadresse des Platzhalters **NOOP**, sondern die Codefeldadresse des Wortes **CRASH** eingetragen. Solange keine andere Vektorisierung vorgenommen wurde, liegt damit in der Parameterfeldadresse des mit Hilfe von **DEFER** definierten Wortes die Codefeldadresse von **CRASH**, und es erfolgt beim Aufruf des Wortes die entsprechende Fehlermeldung.
Im Normalfall wird aber zu irgendeinem Zeitpunkt vor dem ersten Aufruf das Wort mit seiner aktuellen Eigenschaft versehen werden. Das Wort **IS** hat die Aufgabe, die auf dem Stack bereitstehende Codefeldadresse an die Parameterfeldadresse zu transportieren. Durch die Worte **STATE @** wird der aktuelle Systemzustand geprüft. Wird ein Wert gleich Null übergeben, dann befindet sich das System im interpretierenden Modus und der ELSE-Zweig der bedingten Verzweigung wird durchlaufen. Die hier eingetragenen Bestandteile entsprechen der Vektorisierung im vorangegangenen Absatz. Befindet sich das System im compilierenden Modus, dann wird die Codefeldadresse des Wortes **(IS)** compiliert. Von **(IS)** wird der Eintrag auf dem Returnstack so manipuliert, dass das auf **IS** folgende Wort nicht ausgeführt wird [3]. Anschliessend erfolgt das Eintragen der Codefeldadresse in die Parameterfeldadresse der Vektorvariablen.
Die vorgestellten Worte **DEFER** und **IS** sind eine geeignete Möglichkeit für den Nutzer, Vektor-Worte in einem Forth-Programm zu nutzen. Da vom System nicht überprüft wird, ob es sich bei der für **IS** bereitgestellten Adresse um eine Codefeldadresse handelt, ist vom Nutzer entsprechende Sorgfalt vonnöten. Neben der vorgestellten Variante bietet [4] auch eine Variante für den Nutzer von fig-Forth. Der Verweis darauf soll genügen.

3. DOER-MAKE-Konstrukt

Eine andere Möglichkeit zur Ausführung von Vektoroperationen ist mit dem DOER-MAKE-Konstrukt gegeben [5].
In Bild 3 ist in zwei Screens der insgesamt erforderliche Quelltext dargestellt. Bevor auf die Einzelheiten eingegangen werden soll, wird wiederum die Handhabung betrachtet. Als neues Definitionswort steht nun **DOER** zur Verfügung:

DOER ^ADU

```
Screen #    1
( DOER-MAKE-Konstrukt                                 CK  4/11/88)
( nach L. Brodie: In FORTH denken. Hanser Verlag Muenchen Wien )
(                                 Prentice-Hall London 1986    )
: NOTHING ;

: DOER
    CREATE ['] NOTHING >BODY ,
    DOES>  @ >R ;

VARIABLE MARKER

: (MAKE)
    R> DUP 2+ DUP 2+ SWAP @ >BODY !
    @ ?DUP IF >R THEN ;

-->

Screen #    2
( DOER-MAKE-Konstrukt                                 CK  4/11/88)

: MAKE
    STATE @ IF
    COMPILE (MAKE) HERE MARKER ! 0 ,
    ELSE HERE [COMPILE] ' >BODY ! [COMPILE] ] THEN ; IMMEDIATE

: ;AND
    COMPILE EXIT HERE MARKER @ ! ; IMMEDIATE

: UNDO
    ['] NOTHING >BODY [COMPILE] ' >BODY ! ;

( Dieser Code ist in "Public Domain".)
```

Bild 3 Quelltext mit dem DOER-MAKE-Konstrukt.

Die Vektorvariable **^ADU** ist auf diese Weise definiert. Ein Aufruf der noch nicht initialisierten Vektorvariablen wird mit einem «ok» quittiert, ansonsten passiert nichts. Um nun das Vektorwort **^ADU** mit einer Eigenschaft zu versehen, verwendet man das Wort **MAKE** in der folgenden Weise:

MAKE ^ADU (ADU) ;

Dieses **MAKE** kann in der vorgestellten Weise interpretativ angewendet werden, aber auch in Colon-Definitionen eingebaut sein.

: TEST MAKE ^ADU (ADU) ;

Durch Aufruf des Wortes **TEST** wird die Vektorvariable **^ADU** mit der Codefeldadresse des Wortes **(ADU)** initialisiert. Innerhalb einer solchen Colon-Definition lassen sich auch weitere Vektorisierungen vornehmen. Hierzu steht das Wort **;AND** zur Verfügung. Dieses Wort steht dann anstelle des Semikolons. Hier ein kleines Beispiel:

```
DOER ^STATUS
: TEST
    MAKE ^ADU (ADU) ;AND
    MAKE ^STATUS ." ADU wird simuliert" ;
```

Durch Aufruf des Wortes **TEST** wird neben der bereits beschriebenen Vektorisierung des Wortes **^ADU** nun auch noch das Wort **^STATUS** vektorisiert. Auf diese Weise kann man das System selbst über die aktuelle Betriebsart befragen. Will man beispielsweise seine Tests am konkreten System der Messdatenverarbeitung in der Weise modifizieren, dass nur das Verhalten um eine Triggerschwelle untersucht wird, kann das Ausgangssignal des Analog/Digital-Umsetzers durch eine sägezahnförmige

Spannung im interessierenden Bereich simuliert werden. Das Wort **NEUER_TEST** lässt sich nun in der folgenden Weise definieren:

```
: NEUER_TEST
    MAKE ^ADU 121
    MAKE ^ADU 123
    MAKE ^ADU 125
    MAKE ^ADU 127
    MAKE ^ADU 129 ;
```

Nach der Ausführung des Wortes **NEUER_TEST** werden nun bei Aufruf der Vektorvariablen **^ADU** nacheinander die Werte von 121 bis 129 auf den Stack gelegt. Dieses Verhalten wird erreicht, weil die einzelnen Zuweisungen nicht durch **;AND** voneinander getrennt worden sind. Hat man schliesslich genug von aller Vektorisierung, dann kann die Vektorvariable mit Hilfe des Wortes **UNDO** auch wieder neutralisiert werden, indem die bei der ursprünglichen Definition eingetragene Parameterfeldadresse eingeschrieben wird und damit das Wort wieder zum «Nichtstun» verurteilt ist.

```
UNDO ^ADU
```

Die Arbeitsweise der hier verwendeten Worte kann dem Listing in Bild 1 entnommen werden. In [5] ist die Implementierung des **DOER-MAKE**-Konstrukts für weitere Forth-Versionen, so auch fig-Forth, enthalten.
Das Definitionswort **DOER** erzeugt mittels **CREATE** einen Header für die neu zu definierende Vektorvariable. In das Parameterfeld wird die Parameterfeldadresse des Wortes **NOTHING** (das mit **NOOP** aus dem vorangegangenen Abschnitt vollkommen identisch ist, compiliert. Bei Ausführung des auf diese Weise definierten Wortes wird die eingetragene Parameterfeldadresse (jetzt noch die von **NOTHING**) zur Abarbeitung auf den Parameterstack gelegt.
Die Initialisierung des Vektors, das heisst die Zuweisung einer gewünschten Eigenschaft konnte mit Hilfe des Wortes **MAKE** sowohl im interpretativen als auch im compilierenden Modus vorgenommen werden. Im interpretativen Modus, **STATE @** hat ein False-flag hinterlassen, werden die Adresse von **HERE** in die Parameterfeldadresse der Vektorvariablen geschrieben und anschliessend die im Eingabestrom liegenden Worte ins Wörterbuch compiliert. **HERE** zeigt dadurch auf die Adresse der Worte aus dem Eingabestrom.
Wird **MAKE** im compilierenden Modus, also innerhalb einer Colon-Definition, angewendet, dann wird die Laufzeitroutine **(MAKE)** compiliert, **HERE** in die Variable **MARKER** gerettet und an diese Stelle eine **0** compiliert. Diesen Wörterbucheintragungen folgen nunmehr die Codefeldadressen der weiteren Bestandteile der Colon-Definition. Bei Aufruf eines so definierten Wortes nimmt **(MAKE)** eine Schlüsselstellung ein. Das Wort **(MAKE)** holt die Adresse der (derzeit) mit **0** beschriebenen Zelle vom Returnstack. Im Anschluss an das zweimalige Duplizieren **(DUP 2+ DUP 2+)** stehen auf dem Stack die Adressen der «Null»-Zelle, sowie die, die die Adressen des Vektorwortes (hier **^ADU**) und des ersten Forth-Wortes der Colon-Definition (hier **(ADU)**) enthalten. **SWAP @ >BODY !** schreibt dann die Adresse des neuen Code (hier **(ADU)** in die Vektoradresse. Nun steht nur noch die Adresse der mit **0** beschriebenen Zelle auf dem Stack und der Rest von **(MAKE)** gelangt wegen dieser Null nicht zur Ausführung.
Die Berechtigung dieser Zelle liegt in der Verwendung des Wortes **;AND**. Wie aus der Anwendung schon ersichtlich war, können zusammengesetzte Definitionen erzeugt werden. **;AND** separiert die einzelnen Bestandteile so, dass der erste Teil durch das Compilieren von **EXIT** abgeschlossen wird und anschliessend die Variable **MARKER** mit der Adresse des folgenden Codes aktualisiert wird. Das Wort **UNDO** zeigt keine grossen Neuigkeiten und soll deshalb nicht näher betrachtet werden.

4. Zusammenfassung

Im vorliegenden Beitrag sollten verschiedene Möglichkeiten der Vektorisierung in Forth vergleichend betrachtet werden. Zum Verständnis der in den verschiedenen Forth-Implementierungen enthaltenen Konstrukte wurde eingangs die Vektorisierung «von Hand» erläutert.
Wer in seinem Forth-System bislang auf die komfortable Möglichkeit der Vektorisierung mit Hilfe von **DEFER-IS** oder **DOER-MAKE** verzichten musste, sollte nicht zögern, eine der beiden Varianten in sein System aufzunehmen. Der geringe Aufwand wird reichlich belohnt. Die Listings wurden so wiedergegeben, wie sie in den angegebenen Literaturstellen vorgefunden wurden.
Nach Geschmack des Autors sollte im an sich mächtigeren **DOER-MAKE**-Konstrukt das **CRASH** aus **DEFER-IS** Berücksichtigung finden. Über einen Hinweis bei Aufruf einer nicht initialisierten Vektorvariablen ist man u. U. recht dankbar.

```
: DOER
    CREATE ['] CRASH >BODY ,
    DOES> @ >R ;
```

Literatur

[1] Brodie, L.: Programmieren in Forth. München, Wien: Carl Hanser Verlag; London: Prentice-Hall International 1984
[2] Laxen, H.; Perry, M.: F83. Implementierungsmodell für Forth-83. Version 2.0 1984
[3] Wejgaard, W.: Menus in Forth. Elektroniker, Aarau 27 (1988) 9, S. 109–113
[4] Zech, R.: Forth 83. München: Franzis Verlag 1987
[5] Brodie, L.: In Forth denken. München, Wien: Carl Hanser Verlag; London: Prentice-Hall International 1986