Modulare Cross-Entwicklungsumgebung unter Windows.

Dr.-Ing. Egmont Woitzel
Stephan Lange
FORTecH Software GmbH
Joachim-Jungius-Straße 9
D-18059 Rostock
Tel. (03 81) 4 05 94 72
Fax (03 81) 4 05 94 71

Seit ihrer Entwicklung werden Forth-Systeme erfolgreich für die Lösung von Automationsaufgaben eingesetzt. Durch die kontinuierlich fallenden Preise für Rechenleistung und die Durchsetzung dezentralisierter Konzepte in der Steuerungstechnik stellen eingebettete Rechnersysteme ein überproportional wachsendes Einsatzgebiet von Rechnersystemen dar. Der folgende Beitrag stellt das System fieldFORTH als eine neuartige Umgebung für die Programmierung derartiger Systeme unter Windows vor.

Forth im Feldeinsatz

Forth eignet sich aufgrund seiner interaktiven Arbeitsweise hervorragend für die Programmierung eingebetteter Systeme. Vor allem in der Inbetriebnahmephase können durch die Möglichkeiten zur Erforschung des Rechnersystems und des angeschlossenen Prozesses ganz im Sinne des Rapid Prototyping erhebliche Einsparungen an Entwicklungszeit erreicht werden. Eingebettete Systeme stellen jedoch auch andere Forderungen an die Entwicklungsumgebung als typische Desktop-Systeme. Dabei haben die folgenden zwei Randbedingungen besonders wesentliche Auswirkungen auf die Struktur der Entwicklungsumgebung.

Aus ökonomischen Gründen werden die Ressourcen eines eingebetteten Rechnersystems gemessen an den Forderungen der Applikation minimal gehalten. Die Implementation eines vollständigen Forth-Systems auf dem Zielsystem ist daher nicht möglich oder mit sehr großen Kompromissen verbunden. Daher sollte eine Entwicklungsumgebung für eingebettete Systeme auf einer *Crosscompiler*-Technologie basieren, wobei der interpretative Charakter gewahrt bleiben muß.

Da eingebettete Systeme nur selten über Standardperipherie wie Tastatur, Videoadapter und Massenspeicher verfügen, sollte wenigstens während der Entwicklungsphase die Möglichkeit bestehen, vom Zielsystem aus auf die I/O-Ressourcen des Entwicklungssystems zuzugreifen. Das bedeutet, daß durch die Entwicklungsumgebung nicht nur ein Interpreterinterface bereitgestellt werden muß, sondern auch *Betriebssystemeigenschaften* in das Zielsystem übertragen werden müssen.

Mit dem System fieldFORTH wird eine Entwicklungsumgebung bereitgestellt, die diesen Forderungen nachkommt und gleichzeitig durch einen neuen Ansatz zur Lösung von Problemen der Crosscompilation deren Handhabung wesentlich vereinfacht.

Bevor jedoch näher auf Struktur und Komponenten von fieldFORTH eingegangen wird, ist eine genauere Betrachtung des Compilationsprozesses unter Forth erforderlich.

Forth-Compilation

Unter Compilation wird allgmein die Übersetzung eines als Quelltext vorliegenden Programmes in eine ablauffähige Form verstanden. In der Forth-Literatur wird der Begriff der Compilation dagegen häufig auf die Erzeugung von Forth-Code innerhalb von :-Definitionen eingeschränkt. Da dieser jedoch nur einen Teil des zu einem ablauffähigen Programm gehörigen Codes darstellt, verwenden die folgenden Ausführungen den Begriff der Compilation im allgemeineren Sinn.

Integrierte Compiler

In einem konventionellen Forth-System wird die Kommunikation mit dem Benutzer über einen Interpreter abgewickelt. Dieser übernimmt in seiner compilierenden Betriebsart Teilaufgaben der Compilation, speziell der Erzeugung von Forth-Code. Die meisten Aufgaben der Codegenerierung werden jedoch durch interpretativ aufgerufene Worte erfüllt, die selbst integraler Bestandteil des Systems sind.

Die Übersetzung von Quelltext und die Erzeugung ablauffähiger Programme wird in Forth als inkrementeller Prozeß behandelt. Anders als bei den meisten inkrementell arbeitenden Compilern für konventionelle Programmiersprachen dürfen die bereits generierten Codeteile in den folgenden Übersetzungsschritten bereits benutzt werden. Für Forth-Programmierer ist das eine fast selbstverständliche Eigenschaft des Programmiersystems.

Crosscompiler

Enthält ein Forth-System keinen integrierten Compiler, muß zu seiner Herstellung und Erweiterung ein separater Compiler benutzt werden. Für solche Compiler ist in der Literatur eine verwirrende Vielzahl von Bezeichnungen zu finden. Im folgenden werden Forth-Compiler, die Code für ein anderes Forth-System erzeugen, als Crosscompiler bezeichnet. Dasjenige System, welches den Compiler trägt, wird im weiteren als Compilersystem, dasjenige, für das der Code bestimmt ist, als Zielsystem bezeichnet.

Crosscompiler können je nach Ankopplung des Zielsystems in zwei recht unterschiedliche Kategorien eingeteilt werden. Existiert das Zielsystem während der Compilation selbst nur in Form eines Speicherabbildes innerhalb des Compilersystems, kann es an der Codeerzeugung nicht aktiv mitwirken. Ein solcher off-line-Compiler wird benötigt, wenn ein Forth-System völlig neu generiert werden soll.

Falls das Zielsystem bereits arbeitsfähig ist und eine Kommunikationsverbindung zum Compilersystem besteht, kann es an der Codegenerierung beteiligt werden. Da hier das Downloaden des generierten Codes mit der eigentlichen Übersetzung verschachtelt wird, kann man diese Betriebsweise auch als Downcompilation bezeichnen. Derartige on-line-Compiler werden vor allem zur Inbetriebnahme von Applikationen benötigt, die selbst keinen integrierten Compiler enthalten.

Betrachtet man die eingangs aufgestellten Forderungen an eine Entwicklungsumgebung für eingebettete Systeme, so stellt man fest, daß diese Komponenten für beide Kategorien enthalten muß.

Syntaktische Probleme

Das komplizierteste Problem das bei der Implementation bzw. Benutzung eines Crosscompilers anstelle eines integrierten Compilers entsteht, ist die Behandlung von Compilererweiterungen. Bei genauerer Betrachtung muß man in diese Kategorie nicht nur die offensichtlichen Compilererweiterungen in Form von Definitionsworten oder `IMMEDIATE`-Worten einordnen, sondern jedes anwenderdefinierte Wort, das während der Übersetzung ausgeführt wird.

Da in diesem Problem der Schlüssel für das Verständnis der Crosscompilation liegt, soll die feine Vermischung zwischen Compilercode und Zielcode anhand einer Implementation der von F83 bekannten `DEFER`-Konstruktion und einer einfachen Anwendung, die diese benutzt, demonstriert werden. Im Quelltext wurden alle definierten Worte, die nur im Zielsystem ausgeführt werden, einfach unterstrichen. Die während der Programmausführung und Compilation ausgeführten Worte wurden gepunktet unterstrichen. Alle nicht markierten Worte werden nur während der Compilation ausgeführt.

Diese Unterteilung macht sichtbar, daß zum Zeitpunkt der Definition eines Wortes keinesfalls automatisch entschieden werden kann, ob ein Wort im Compilersystem oder Zielsystem angelegt werden muß. Das Wort `(IS)` wird beispielsweise durch `[IS]` in das Wort `SET-FLOORED` compiliert und durch dieses ausgeführt. Das Beispiel zeigt auch, daß die Behandlung des Problems ausschießlich auf Basis der definierten Worte nicht zur Lösung führt. Der Quellcode für das Definitionswort `DEFER` enthält genau genommen zwei völlig

```
\ ----------------------------------------------
\ Beispiel für Compilererweiterungen
\ Laxen/Perry's DEFERs
\ Forth '95 Berlin
\ ----------------------------------------------

\ === DEFER-Implementation
\ --- Laufzeitfunktionen

: DUMMY ( ps: ===> )( Nichtsnutz ) ;

: (IS) ( ps: xt ===> )( ip: 'defer )( setzt xt als Vektor ein )
        R> DUP CELL+ >R @ ! ;


\ --- Compilerfunktionen

: IS ( ps: xt ===> )( ib: defer )( setzt xt als Vektor ein )
        ' >BODY ! ;

: [IS] ( ps: ===> )( ib: defer )( sorgt für Umsetzen des Vektors )
        COMPILE (IS) ' >BODY , ; IMMEDIATE


\ --- Definitionswort

: DEFER ( ps: ===> )( ib: name )( definiert Vektor )
        ['] DUMMY CONSTANT DOES>
        ( ps: ... ===> ... )( führt aktuellen Vektor aus )
        @ EXECUTE ;


\ === Anwendungsbeispiel: Runden

: 10** ( ps: u1 ===> u2 )( berechnet 10**u1 )
        1 SWAP 0  ?DO  10 *  LOOP ;

2     ( zu rundende Dezimalstellen )
10**  ( Rundungsbereich )
CONSTANT ROUND-RANGE ( ps: ===> u )

: FLOORED ( ps: n1 ===> n2 )( rundet gegen minus unendlich )
        ROUND-RANGE / ROUND-RANGE * ;

: NEAREST ( ps: n1 ===> n2 )( rundet gegen Null )
        ROUND-RANGE 2/ +      FLOORED ;

DEFER ROUND ( ps: n1 ===> n2 )( rundet FLOORED oder NEAREST )

: SET-FLOORED ( ps: ===> )( aktiviert FLOORED )
        ['] FLOORED [IS] ROUND ;

: SET-NEAREST ( ps: ===> )( aktiviert NEAREST )
        ['] NEAREST [IS] ROUND ;

SET-FLOORED ( ROUND initialisieren )
```

separate Definitionen. Einerseits wird das Definitionsverhalten definiert, das nur während der Compilation benötigt wird. Durch den `DOES>` folgenden Code wird das Ausführverhalten der durch `DEFER` erzeugten Worte definiert. Dieser Code wird nur im Zielsystem ausgeführt.

Bisher bekannte Crosscompiler konnten dieses Problem bisher kaum zufriedenstellend lösen. Typisch kann durch Compilersystem und Zielsystem kein Code gemeinsam benutzt werden, an dieser Stelle ist der Quelltext zu duplizieren. Ein zweites nur unzulänglich bewältigtes Problem stellt der Übergang zwischen off-line- und on-line-Crosscompilation dar. Aufgrund der unterschiedlichen Möglichkeiten des Zielsystems müssen Compilererweiterungen in der Regel syntaktisch anders formuliert werden. Die Übergabe Compiler-interner Informationen z. B. über das innerhalb von Definitionsworten definierte Laufzeitverhalten war ebenfalls problematisch.

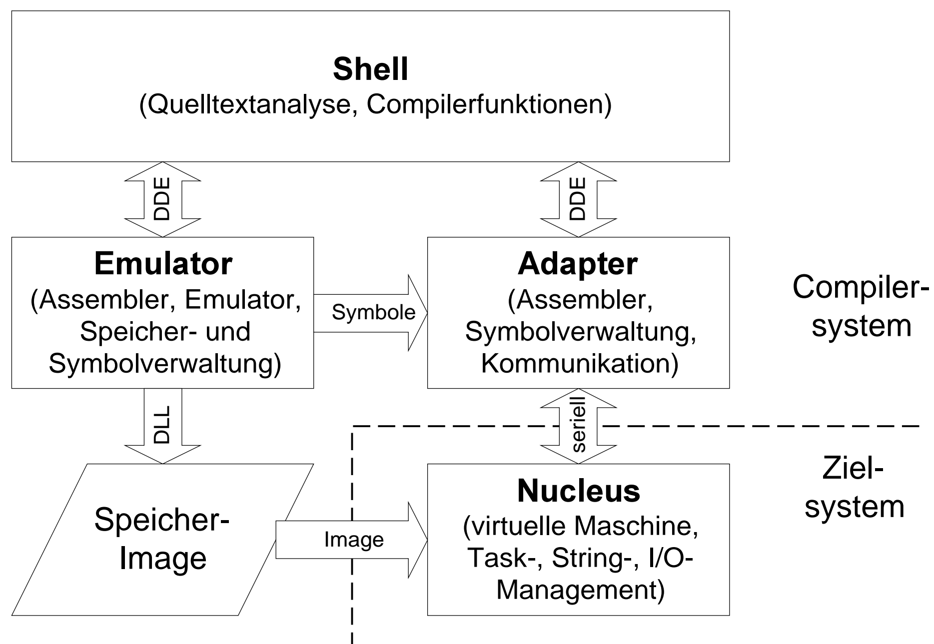
Das fieldFORTH-Konzept

Der Ansatz von fieldFORTH zur Lösung der aufgeführten Probleme ist in seiner Grundidee überraschend einfach. Das Problem der Ausführung von Definitionen des Zielsystems während der Compilation tritt bei Benutzung eines on-line-Crosscompilers bzw. Downcompilers nicht auf, da diese Funktionalität für den Interpreterbetrieb ohnehin erforderlich ist. Wichtig ist nur, daß Definitionen, die als echte Compilererweiterungen nur im Compilersystem residieren, virtuell im Zielsystem ablaufen, d. h. das Adreßmodell und den Stapel des Zielsystems verwenden sowie Worte des Zielsystems aufrufen können.

Um nun off-line und on-line-Umgebung syntaktisch aneinander anzugleichen, muß die Funktionalität der off-line-Umgebung auf die eines on-line verfügbaren Zielsystems angehoben werden. Dies wird erreicht, indem der off-line generierte Code mit Hilfe eines Software-Emulators ausgeführt werden kann. Soll während der Compilation Code des Zielsystems ausgeführt werden, wird anstelle der Ziel-CPU der Software-Emulator aktiv.

Nachdem beide Compilerkategorien einheitlich behandelt werden können, liegt es nahe, den eigentlichen Compiler zu faktorisieren und eine möglichst schmale systeminterne Schnittstelle zu definieren, die alle speziellen Eigenschaften des Zielsystems kapselt. Dies erlaubt einerseits aus Entwicklersicht die relativ einfache Einbindung neuer Zielsysteme und andererseits aus Anwendersicht auch die gemeinsame Benutzung von Compilererweiterungen durch verschiedene Zielsysteme.

Der Compiler stellt natürlich nur eine Seite des Systems dar. Das Zielsystem stattet fieldFORTH mit einem skalierbaren Forth-Kern aus, der auch ein I/O-Interface zum Compilersystem enthält, über das auf Betriebssystemfunktionen zugegriffen werden kann. Die folgende Abbildung illustriert das Zusammenwirken der einzelnen Komponenten.



Im off-line-Betrieb kann über Shell und Emulator ein fieldFORTH-Nucleus für eine bestimmte Systemkonfiguration generiert werden. Das Speicherimage kann anschließend in das Zielsystem geladen werden. Über den Adapter kann dann die Shell mit dem Nucleus für den on-line-Betrieb verbunden werden. Die erforderlichen Symbolinformationen kann der Adapter aus dem Emulator übernehmen.

Im folgenden sollen einige der wesentlichen Merkmale der einzelnen Komponenten in Bezug auf ihre Auswirkungen auf den Entwicklungsprozeß diskutiert werden.

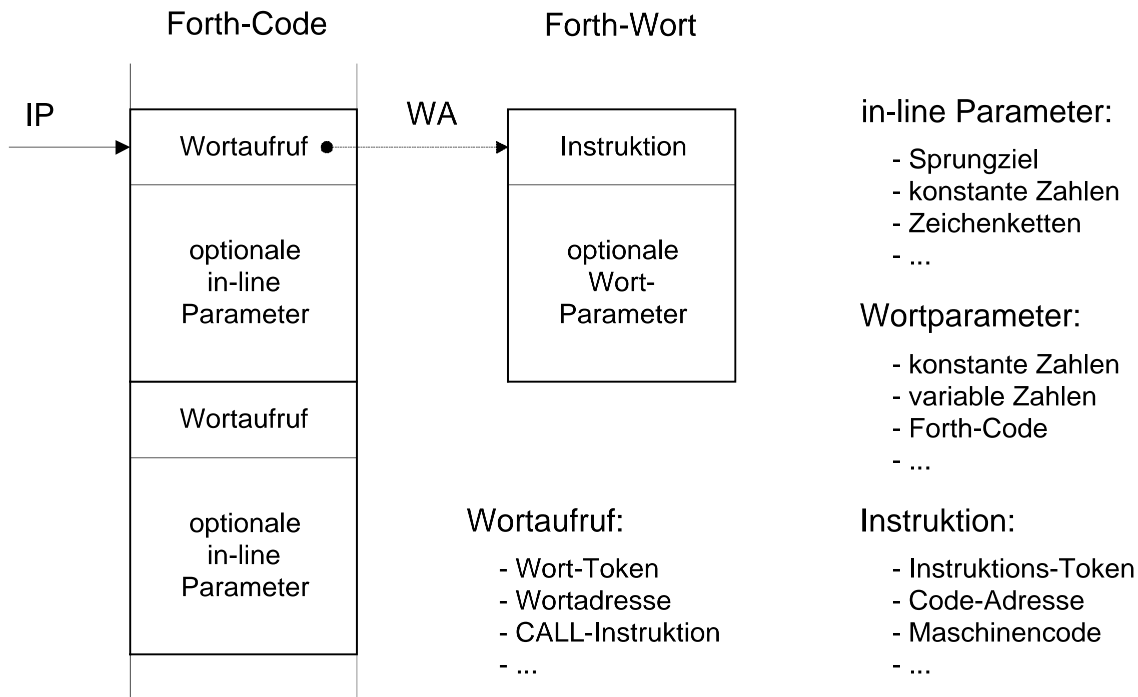
Shell

Die Shell enthält alle Zielsystem-unabhängigen Anteile der Cross-Entwicklungsumgebung und stellt gleichzeitig die wichtigste Komponente der Bedienoberfläche von fieldFORTH dar. Da die Shell als Aufsatz auf comFORTH für Windows implementiert wurde, können alle Bedienmerkmale dieses Systems ohne Einschränkung auch während der Arbeit mit dem Zielsystem genutzt werden.

Die Shell nimmt alle diejenigen Definitionen auf, die nur während der Compilation benutzt werden und daher nicht im Nucleus enthalten sind. Durch die Verwendung einer relativ abstrakten Schnittstelle zu den elementaren Compilerfunktionen ist der Code weitestgehend portabel, so daß er für verschiedene Zielsysteme gemeinsam verwendet werden kann. Diese Definitionen werden daher als generisch bezeichnet. Die Shell enthält keine fest eingebauten generischen Definitionen. Dadurch besteht die Möglichkeit, den für die Programmierung des Zielsystems benutzten Compiler nahezu beliebigen Forderungen anzupassen. Man kann beispielsweise anstelle des vom fieldFORTH-Nucleus verwendeten Forth-83-Compilers auch einen ANS-Forth oder f.i.g.-kompatiblen Compiler definieren und benutzen.

Target-Emulator

Der Emulator verwaltet alle Zielsystem-abhängigen Informationen für den off-line-Betrieb. Dazu gehören alle Informationen über Wortbreite, Adressierungsmodell, den Assembler etc. Der Emulator wird nicht vom Bediener direkt, sondern ausschließlich über die Shell angesprochen. Da der Emulator zur Erzeugung kompletter Forth-Systeme verwendet werden soll, unterstützt er im Gegensatz zu gewöhnlichen Forth-Compilern auch Vorwärtsreferenzen, die automatisch angelegt und aufgelöst werden. Das interessanteste Detail des Emulators stellt jedoch das zur Emulation des Targetsystems verwendete Verfahren dar.



Betrachtet man den Aufbau von Forth-Code wird man de facto jede Implementation auf das oben abgebildete Schema zurückführen können. Der Forth-Prozessor arbeitet Code ab, der im Gegensatz zu konventionellen Prozessoren nicht aus einer Liste von Instruktionen, sondern aus einer Liste von Wortaufrufen besteht. Worte können aus Sicht des Forth-Prozessors als eine Kopplung von einem Operationscode für genau eine seiner Instruktionen mit einem optionalem Parametersatz aufgefaßt werden. In dieser Strukturierungsmöglichkeit wurzelt die große Kompaktheit von Forth-Code. Die Fähigkeit zur Adressierung von Wortparametern unterscheidet einen Forth-Prozessor von einem Stapelprozessor.

Benutzt man dieses Modell eines Forth-Prozessors, erkennt man leicht, daß alle gewöhnlichen Code-Worte Instruktionen definieren, die keine Wortparameter adressieren. Instruktionen, die Wortparameter adressieren, werden demgegenüber zusammen mit Definitionsworten definiert, d. h. alle Worte, die mit demselben Definitionswort erzeugt werden, benutzen dieselbe Instruktion aber jeweils mit spezifischen Wortparametern.

Um das Verhalten von Forth-Code zu emulieren, genügt es daher, für alle Instruktionen eine Emulation anzugeben. Ist dies einmalig für den Kern ausgeführt, können z. B. beliebige :-Definitionen automatisch emuliert werden.

Analog zur Shell enthält auch der Emulator keinen fest eingebauten Emulationscode. Dieser wird erst mit Hilfe einer speziellen Emulations-Assemblersprache definiert. Da diese vom spezifischen Adreßmodell und der Datenbreite des Zielsystems abstrahiert, besitzt der Emulationscode selbst gute Portierungseigenschaften. Für alle im fieldFORTH-Nucleus definierten Instruktionen wird Emulationscode zur Verfügung gestellt.

Wie in der Abbildung vermerkt, lassen sich alle verwendeten Implementierungstechniken für Forth auf dieses Schema abbilden. Selbst optimierende native-Code-Compiler können nur in begrenztem Umfang Instruktionen ohne Wortparameter in Form einer Makroersetzung in den Wortaufruf expandieren und dort eine zusätzliche Optimierung auf Register-Niveau durchführen. Dadurch wird zwar die Rücktransformation und damit das Auffinden der erforderlichen Emulation erschwert, die Arbeitsweise wird jedoch nicht prinzipiell verändert.

Target-Adapter

Der Adapter verbindet einen aktiven Nucleus mit der Shell. Ebenso wie der Emulator kapselt auch der Adapter alle spezifischen Eigenschaften des Zielsystems wie Adreßmodell, Assembler etc. Da jedoch das Zielsystem selbst ablauffähig ist, unterstützt der Adapter keine Vorwärtsreferenzen. Worte des Zielsystems werden nicht emuliert, sondern direkt auf dem Zielsystem ausgeführt. Dazu findet ein Kommunikationsprotokoll Verwendung, das in seinem Kern auf der Übertragung von execution token beruht und ein Forth-spezifisches remote procedure call (im weiteren RPC) realisiert. Um das System nicht auf ein spezielles Kommunikationsmedium festzulegen, wurde die Ansteuerung der Kommunikationsschnittstelle in eine frei konfigurierbare DLL verlegt.

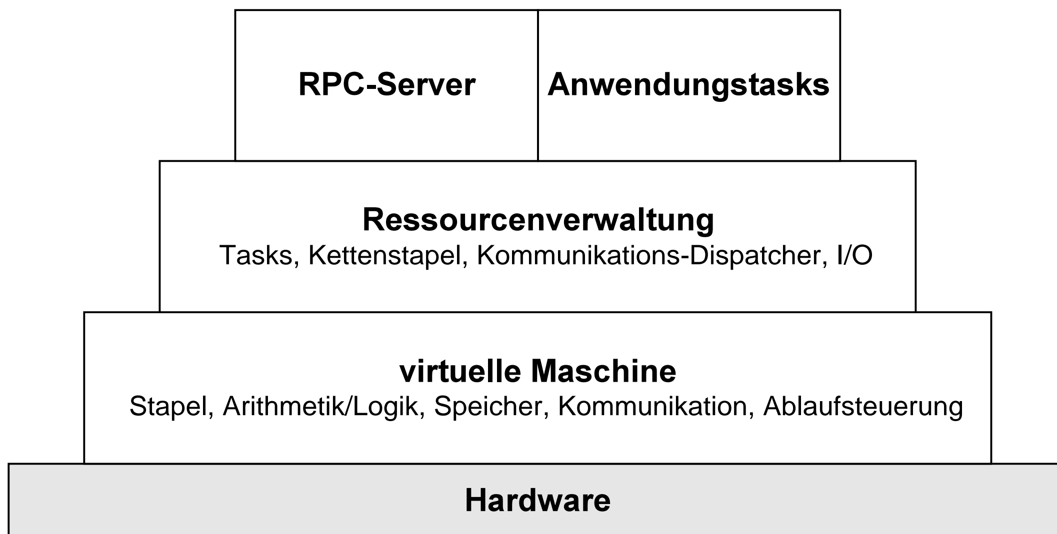
Eine zweite Aufgabe des Adapters besteht in der Bereitstellung von I/O-Diensten für den Nucleus. Die I/O-Forderungen werden zeitmultiplex über die auch vom RPC-Protokoll benutzte Schnittstelle transportiert. Zur zeichenorientierten Ein- bzw. Ausgabe wird der Workspace des Adapters benutzt.

Nucleus

Der fieldFORTH-Nucleus ist ein unvollständiges Forth-System, da es weder Interpreter noch Compiler enthält. Im Sinne des Forth-83-Standarddokuments enthält er im wesentlichen die Funktionalität von Kernschicht und Geräteschicht. Die im Nucleus enthaltenen Funktionen sind zu denen des comFORTH-Kern kompatibel, so daß Algorithmen in einem frühen Projektstadium unter comFORTH getestet werden können.

Der Nucleus enthält zur Behandlung von Zeichenketten und zur Zahlenkonvertierung einen Kettenstapel. Die Lösung typischer Probleme eingebetteter Systeme wird durch einen optionalen kooperativen Multitasker sowie die integrierte Interrupt-Behandlung auf Forth-Niveau unterstützt.

Schichtenmodell des fieldFORTH Nucleus



Gegenwärtig ist fieldFORTH als 32-Bit-Implementation für die Prozessoren Motorola M68332 und Texas Instruments TMS320C40 verfügbar.

Softwareentwicklung mit fieldFORTH

Abschließend soll der Softwareentwicklungsprozeß mit fieldFORTH anhand eines Quelltextbeispiels sowie einigen Bemerkungen zur Projektorganisation illustriert werden.

Crosscompilation

Die Leistungsfähigkeit des implementierten Crosscompilerverfahrens wird am besten anhand der oben zitierten Implementation der `DEFER`-Konstruktion sichtbar.

Der originale Quelltext kann nahezu unverändert übernommen werden. Nur die expliziten Compilererweiterungen werden nicht in das Zielsystem übersetzt, sondern als generische Definitionen implementiert. Auffälligste Besonderheit ist die explizite Deklaration des Ausführungsverhalten von DEFER-definierten Worten mit Hilfe des Instruktions-:. Dies ist deshalb notwendig, weil der DOES> folgende Part der generischen Definition von DEFER als Emulationscode behandelt wird.

Aufgrund des Emulationsverfahrens kann der Beispielquelltext völlig unverändert weiterverwendet werden. Bereits mit dem Emulator kann ein Test des Verhaltens von ROUND erfolgen.

Projektablauf

Erste Erfahrungen mit fieldFORTH zeigen, daß eine dreistufige Projektorganisation den realen Anforderungen angemessen ist. Bei Start eines neuen Projekts muß zur Anpassung an das Speicherlayout sowie das Kommunikationsinterface des Zielsystems einmalig eine Neugenerierung des Kerns erfolgen.

Der wesentliche Teil der Projektarbeit wird on-line auf dem Zielsystem ausgeführt. Hat sich genügend viel getesteter Quelltext angesammelt, so daß das Nachladen zu Sitzungsbeginn

```
\ ----------------------------------------------
\ Beispiel für Compilererweiterungen
\ Laxen/Perry's DEFERs
\ Forth '95 Berlin
\ ----------------------------------------------

\ === DEFER-Implementation
\ --- Laufzeitfunktionen

: DUMMY ( ps: ===> )( Nichtsnutz ) ;

: (IS) ( ps: xt ===> )( ip: 'defer )( setzt xt als Vektor ein )
        R> DUP CELL+ >R @ ! ;


\ --- Compilerfunktionen

GENERIC
: IS ( ps: xt ===> )( ib: defer )( setzt xt als Vektor ein )
        ' >BODY ! ;

: [IS] ( ps: ===> )( ib: defer )( sorgt für Umsetzen des Vektors )
        COMPILE (IS) ' >BODY , ; IMMEDIATE


\ --- Definitionswort

: DEFER ( ps: ===> )( ib: name )( definiert Vektor )
        ['] DUMMY CONSTANT DOES>
        ( ps: ... ===> ... )( führt aktuellen Vektor aus )
        @ EXECUTE ;

TARGET
I: DEFER ( ps: ... ===> ... )( führt aktuellen Vektor aus )
        @ EXECUTE ;
```

störend viel Zeit in Anspruch nimmt, kann dieser Teil im off-line-Verfahren auf den Kern compilert werden. Das dabei entstehende Image kann dann wiederum als Ausgangsbasis für die weitere on-line-Arbeit dienen.