Wörter definieren

Damit ein Mikrocontroller eine bestimmte Aufgabe erfüllt, muss man ihm entsprechende Befehle geben. Bislang hatten wir dazu die Programmiersprache BASCOM oder den Assembler von AVR benutzt. Jetzt soll gezeigt werden, wie man hierzu unser MikroForth-System einsetzen kann.

FORTH-Befehle werden **Wörter** genannt. Derartige Wörter kann man zu Befehlsgruppen zusammenfassen; so entstehen neue Wörter. Die Gesamtheit aller Wörter, welche FORTH zur Verfügung stehen, bezeichnet man als **Vokabular**. Wenn man ein neues Wort herstellt, bedeutet dies letztlich eine Erweiterung des Vokabulars.

Am Beispiel eines Ampelprogramms wollen wir dies verdeutlichen. Zur Vereinfachung lassen wir dabei im Folgenden die in Deutschland übliche Rot-Gelb-Phase weg.

```
: ampelzyklus rotphase grünphase gelbphase ;
```

Unser FORTH-Compiler arbeitet grundsätzlich in zwei Schritten. Im ersten Schritt wird der eingegeben Quelltext interpretiert. Der Teil des FORTH-Programms, welcher dafür zuständig ist, wird **Interpreter** genannt. In unserem Fall stößt der Interpreter zunächst auf den Doppelpunkt; dieser zeigt ihm, dass ein neues Wort mit dem Namen `ampelzyklus` erzeugt werden soll. Dieses Wort setzt sich aus den folgenden Befehlen `rotphase`, `grünphase` und `gelbphase` zusammen. Das Semikolon zeigt dem Interpreter das Ende der Befehlsfolge an.

Da unser Interpreter grundsätzlich nur eine Wortdefinition pro Zeile zulässt, könnte man eigentlich auf das Semikolon verzichten. Andere FORTH-Compiler lassen aber auch mehrere Wortdefinitionen pro Zeile zu; da wird das **Semikolon** als **Begrenzer** unverzichtbar.

Ein neues Wort wird demnach allgemein so definiert:

Doppelpunktdefinition

`:` <Name des neuen Wortes> <Befehlsfolge mit bereits definierten Wörtern> `;`

Sämtliche Wörter - auch der Doppelpunkt und das Semikolon - müssen dabei durch (mindestens) ein Leerzeichen getrennt werden.

Testen wir nun unsere erste Wort-Schöpfung: Wir starten das Programm Forth2 und geben den FORTH-Quelltext ein. Groß-Klein-Schreibung spielt für MikroForth übrigens keine Rolle. Anschließend betätigen wir die Interpretieren-Schaltfläche. Im Statusfeld am unteren Rand des

Forth2-Formulars erscheinen sogleich die folgenden Meldungen:

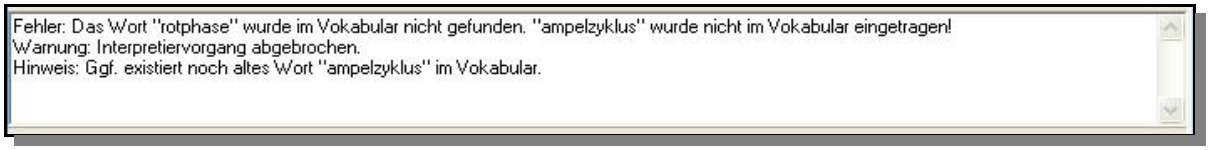

Abbildung 1

Was haben sie zu bedeuten? Die erste Meldung ist eine Fehlermeldung. Fehler führen in der Regel zum Abbruch eines Vorgangs. In diesem Fall weist die nächste Warnung darauf hin, dass der Interpretiervorgang abgebrochen wurde.

Um unseren Fehler beseitigen zu können, müssen wir seine Ursache finden. Offensichtlich kennt unser FORTH-System das Wort `rotphase` nicht. Das ist nicht schlimm, denn wir können diesen "Fehler" beseitigen , indem wir die Definition von `rotphase` nachholen. Dazu fügen wir *vor* der Definition von `ampelzyklus` die folgende Zeile ein:

```
: rotphase rotesLicht an warte rotesLicht aus ;
```

Jetzt beschwert sich unser FORTH-System nicht mehr über das nicht gefundene Wort `rotphase`, dafür aber meldet es, dass es das Wort `rotesLicht` nicht finden kann. Also müssen wir auch dieses Wort noch definieren. Ähnliches gilt für die Wörter `grünphase`, `gelbphase`, `an`, `warte`, `aus` sowie die Wörter `gelbesLicht` und `grünesLicht`.

All diese Definitionen liegen schon fix und fertig in der Datei ampel.frth vor. Öffnen Sie diese Datei mit "Datei - öffnen". Der Quelltext sieht dann so aus:

```
: initialisierePortB 7 DDRB ;
: warte 3 wait ;
: rotesLicht 2 ;
: gelbesLicht 1 ;
: grünesLicht 0 ;
: an 1 outPortB ;
: aus 0 outPortB ;
: rotphase rotesLicht an warte rotesLicht aus ;
: grünphase grünesLicht an warte grünesLicht aus ;
: gelbphase gelbesLicht an warte gelbesLicht aus ;
: ampelzyklus rotphase grünphase gelbphase ;
: main initialisierePortB ampelzyklus ampelzyklus ;
```

Damit der Interpreter keine Fehler mehr meldet, müssen unsere neuen Wörter - über Zwischenstufen - auf solche Wörter zurückgeführt werden, die sich bereits im Vokabular befinden. In diesem Fall sind das die Zahlen 0, 1, 2, 3 und 7 (Auch diese können als Wörter angesehen werden!) sowie die Wörter `wait`, `DDRB` und `outPortB`.

Das Wort `wait` veranlasst den Attiny zu warten, `outPortB` gibt Werte am Port B aus und `DDRB` stellt das Datenrichtungsregister von PortB ein. Wie diese drei Wörter funktionieren, werden wir in den nächsten Kapiteln noch eingehend betrachten. Hier sollte nur eines deutlich werden:

Komplexe Wörter wie unser Wort `ampelzyklus` können wir Schritt für Schritt auf elementare Wörter zurückführen; diese Vorgehensweise nennt man auch **Top-Down-Programmierung.**

Wir hätten natürlich auch genau umgekehrt vorgehen können: Ausgehend von den elementaren Wörtern hätten wir immer komplexere Wörter definieren können, bis wir schließlich bei unserem Wort `ampelzyklus` ausgekommen wären. Diese Vorgehensweise bezeichnet man als **Bottom-Up-Programmierung**. In der Praxis arbeitet man häufig mit beiden Methoden gleichzeitig.

Wichtig ist allerdings für uns: Wörter, die zum Definieren eines neuen Wortes benutzt werden, müssen vorher bereits definiert worden sein. Das bedeutet: Sie müssen schon zum Grundvokabular von FORTH gehören oder in den vorangehenden Zeilen definiert und somit beim Interpretieren bereits zum Vokabular hinzugefügt worden sein. Die grundlegenden Worte müssen im FORTH-Quellcode also immer oben stehen, die daraus abgeleiteten weiter unten.

Unabhängig davon, ob wir die Top-Down-Methode oder die Bottom-Up-Methode benutzen - im Ergebnis ist das zu lösende Problem, eine Ampelanlage zu programmieren, schrittweise in viele kleine Teilprobleme zerlegt worden. Solche Teilprobleme nennt man auch **Module** und die Zerlegung selbst wird als **Modularisierung** bezeichnet. Modularisierung ist ein wesentliches Merkmal der Programmiersprache FORTH. Gute FORTH-Programme zeichnen sich dadurch aus, dass die einzelnen Wort-Definitionen sinnvolle Einheiten bilden und nicht zu lang sind. Natürlich sollten auch die benutzten Wortnamen aussagekräftig sein.

Schauen wir daraufhin noch einmal den Quelltext an. Erfüllt er die Kriterien eines guten FORTH-Codes? Sicherlich sind die ersten Zeilen - so kurz sie auch sein mögen - nicht unmittelbar einleuchtend; das hängt aber damit zusammen, dass wir die Wörter `wait`, `DDRB` und `outPortB` noch nicht genügend kennen. Wort-Folgen wie

```
grünesLicht an warte grünesLicht aus
```

lassen sich dagegen auch ohne Programmierkenntnisse leicht verstehen.

Das wichtigste Wort im ganzen Quelltext haben wir noch nicht besprochen; es ist das Wort `main`. Wenn der Attiny eingeschaltet wird, startet er immer mit der Ausführung genau dieses Wortes. Das Wort **`main`** hat demnach die Bedeutung eines Hauptprogramms; daher stammt auch die Wahl des Wortnamens (“main” = “haupt”). Alle Aktionen, welche der Mikrocontroller ausführen soll, müssen letztlich von diesem Wort ausgehen.

Somit muss der Quelltext immer mit der Definition von `main` enden, und beim Interpretieren

muss man das Überschreiben eines bereits bestehenden `main`-Wortes stets zulassen; ansonsten arbeitet FORTH mit einem solchen alten "Hauptprogramm". Und das hat womöglich gar nichts mit unserer Ampelsteuerung zu tun.

In unserem Fall sehen wir als letzte Zeile:

```
: main initPortB ampelzyklus ampelzyklus ;
```

Das bedeutet: Der Attiny soll zunächst das Port B initialisieren und danach zwei volle Ampelzyklen durchlaufen.

Bestimmt haben Sie inzwischen der Versuchung nicht mehr widerstehen können und den Quelltext unseres vollständigen Ampelprogramms interpretieren lassen. Wenn Sie ihn nicht abgeändert haben, müsste im Statusfeld jetzt angezeigt werden, dass das (im Vokabular schon) bestehende Wort `main` (wie gewünscht) überschrieben wurde.

Der für uns aufwändige Teil des Programmierens ist damit getan. Der Quelltext wurde erstellt und die neu definierten Wörter ins Vokabular übernommen. Jetzt muss unser FORTH-System ans Arbeiten: Die Wörter müssen in Attiny-Maschinencode umgesetzt werden. Diesen Schritt bezeichnet man als **Kompilieren**. Wie dieses Kompilieren im Einzelnen funktioniert, lässt sich bei FORTH recht gut nachvollziehen. In einem späteren Kapitel werden wir darauf ausführlich eingehen.

Jetzt aber machen wir es uns einfach: Wir drücken die Kompilieren-Schaltfläche und im Anschluss daran die Intel-HEX-Code-Schaltfläche. Den HEX-Code übertragen wir schließlich wie üblich mit dem Uploader-Programm auf den Attiny, auf dessen Platine wir in weiser Voraussicht eine rote LED bei PortB.2, eine gelbe bei PortB.1 und eine grüne bei PortB.0 eingesteckt haben.

Probieren Sie es selbst aus. Bestimmt werden auch Sie bei Ihrer Attiny-Platine die zwei Ampelzyklen beobachten können.

Aufgabe 1:

Ergänzen Sie die Datei ampel.frth so, dass eine "deutsche Ampel" mit einer zusätzlichen Gelb-Grün-Phase entsteht.