Ingenieurbüro für Echtzeitprogrammierung

RT-Link

Dok-Rev. 1.1 vom 09.08.2001

Software-Rev. 5.1 vom 09.08.2001

Inhaltsverzeichnis

Revisionsliste:

Rev.	Datum	Na.	Änderung
1.0	22.08.2000	Ko	Übernahme der Tex-Doku
1.1	09.08.2001	Kr	-P option des RTLink

1 Urheberrecht und Haftung

Alle Rechte an diesen Unterlagen liegen bei der IEP GmbH, Langenhagen.

Die Vervielfältigung, auch auszugsweise, ist nur mit unserer ausdrücklichen schriftlichen Genehmigung zulässig.

In Verbindung mit dem Kauf von Software erwirbt der Käufer einfaches, nicht übertragbares Nutzungsrecht. Dieses Recht zur Nutzung bezieht sich ausschließlich darauf, daß dieses Produkt auf oder in Zusammenhang mit jeweils **einem** Computer zu benutzen ist. Das Erstellen einer Kopie ist ausschließlich zu Archivierungszwecken unter Aufsicht des Käufers oder seines Beauftragten zulässig. Der Käufer haftet für Schäden, die sich aus der Verletzung seiner Sorgfaltspflicht ergeben, z.B. bei unautorisiertem Kopieren, unberechtigter Weitergabe der Software usw.. Der Käufer gibt mit dem Erwerb der Software seine Zustimmung zu den genannten Bedingungen. Bei unlizensiertem Kopieren muß vorbehaltlich einer endgültigen juristischen Klärung von Diebstahl ausgegangen werden. Dies gilt ebenso für Dokumentation und Software, die durch Modifikation aus Unterlagen und Programmen von IEP hervorgegangen ist, gleichgültig, ob die Änderungen als geringfügig oder erheblich anzusehen sind.

Eine Haftung seitens IEP für Schäden, die auf den Gebrauch von Software, Hardware oder Benutzung dieses Manuskriptes zurückzuführen sind, wird ausdrücklich ausgeschlossen, auch für den Fall fehlerhafter Software oder irrtümlicher Angaben.

Das Einverständnis des Käufers oder Nutzers für den Haftungsausschluß gilt mit dem Kauf und der Nutzung der Software und dieser Unterlagen als erteilt.

1.1 Handhabung

Lesen Sie bitte zuerst sorgfältig diese Dokumentation bevor Sie anfangen zu programmieren. Sie sparen Zeit und vermeiden Probleme.

1.2 Erklärung

Wir behalten uns das Recht vor, Änderungen, die einer Verbesserung der Schaltung oder des Produktes dienen, ohne besondere Hinweise vorzunehmen. Trotz sorgfältiger Kontrolle kann für die Richtigkeit der hier gegebenen Daten, Schaltpläne, Programme und Beschreibungen keine Haftung übernommen werden. Die Eignung des Produktes für einen bestimmten Einsatzzweck wird nicht zugesichert.

2 Die Anwendung von RT-LINK

2.1 Vorwort

Bei diesen Seiten handelt es sich um eine komplette Überarbeitung des bisherigen Handbuches, das im Laufe der vergangenen Jahre etwas den Bezug zur Realität verloren hat.

2.2 Intention des Handbuches

Das Manual ist nach Art eines Schnellkurses aufgebaut. Sie sollten dadurch in die Lage versetzt werden, PEARL-Projekte problemlos in ein EPROM brennen zu können.

Im Kapitel 2.7 geht es um eine Erläuterung, weshalb der schnelle Versuch, ihr eigenes Projekt mal eben schnell zu linken, wahrscheinlich scheitern wird und welche Maßnahmen bei der Verwendung von externen Referenzen ergriffen werden müssen.

Darauf folgt eine Einführung in das Konzept der Aufteilung von RAM- und ROM-Bereichen mittels symbolischer Namen im Kapitel 2.8. Die Möglichkeiten von RT-Link, unterschiedliche Ausgabedateien zu erzeugen, werden im Kapitel 2.9 behandelt.

2.3 Lieferumfang

Die Auslieferungsdiskette enthält jeweils die aktuellste Version von RT-Link in Form eines S-Records bzw. als EXE-File für Windows-NT/Windows-95. Die MSDOS-Varianten sind dauerhaft abgekündigt! Die Versionsnummer des Linkers auf der Diskette kann und wird sich von der im Handbuch unterscheiden - Doku wächst eben stets langsamer. Der Name lautet `RTLINK` bzw. `RTL.EXE`.

Weiterhin befindet sich auf der Linkerdiskette der S-Record `CROSS`, dessen Verwendung im Kapitel 3 erläutert wird.

Im Ordner `PROBE` sind einige Beispieldatei abgespeichert, die den Einstieg im Kapitel 2.6 erleichtern sollen.

Die Datei `README.RTL` enthält ein ausführlich kommentiertes Linkfile.

2.4 Installation

Die RTOS-UH-Version kann in den Arbeitsspeicher Ihres RTOS-UH-Rechners geladen, oder auf ein Execution-Directory gelegt werden.

Die MS95/98/NT-Version sollten Sie in ein Verzeichniss auf der Platte Ihres Rechners spielen.

2.4.1 Erster Aufruf

Beim Aufruf ohne Argumente meldet sich RT-Link mit einer Usage-Meldung, die so oder ähnlich aussieht:

```
IEP-Linker RT-Link                    Version for WINDOWS NT i80x86
Release 5.1                           Copyright by IEP Hannover
Copyright (C) 1992-2001               Aug  9 2001  14:01:44

Usage:RTLINK linkfile [-options]

  -C     Generate '.crs'-File
  -I     Show DATION's
  -M     Dump loader map
  -S     Show global symbols
  -T     Force short SR-lines (80 Bytes)
  -V     Verbose messages
  -W     Suppress warnings
  -P     Disable LW-Alignment

  -D     Generate internal debugfile

Expected linkfile
```

Als einzige Eingabe erwartet RT-Link ein Linkfile mit der Extension `.RTL`, in dem die Anweisungen zum Linken des Projektes zusammengefaßt sein müssen. Die Option `-M` zwingt den Linker zur Ausgabe eines MAP-Files, in dem die Ergebnisse des Linkerlaufes aufgelistet sind. Der Name der MAP-Datei ergibt sich aus dem Namen der Linkdatei durch Ersetzung der Extension `.RTL` mit der Endung `.MAP`.

Mittels `-V` werden weitere – gerade in der Entwicklungsphase – recht aufschlußreiche Informationen ausgegeben.

Mit der Option `-C` läßt sich für das gelinkte Projekt ein Cross-File erstellen, das in Form eines S-Records die Definitionen sämtlicher globalen Symbole des Projektes enthält. Der Name der Cross-Datei ergibt sich aus dem Namen der Linkdatei durch Ersetzung der Extension `.RTL` mit der Endung `.CRS`.

Die Option `-T` wurde eingeführt, um EPROM-Brenner zu unterstützen, die im S-Record-Modus nur eine maximale Zeilenlänge von 80 Zeichen unterstützen.

Die `-W`-Option unterdrückt die Ausgabe von Warnings. Unter Umständen kann es dann aber passieren, daß die lautlos erstellten EPROM's später nicht lauffähig sind!

Die -P-Option schaltet das normalerweise wirkende Langwortalignment zwischen einzelnen Code- bzw. Datasegmenten aus und macht zwischen diesen Segmenten nur noch ein Wortalignment.

2.5 Das Konzept des RT-Link

Ein Linker hat prinzipiell nur die triviale Aufgabe, aus mehreren getrennt übersetzten Modulen eine funktionelle Einheit (das Programm) zu erstellen, das entweder ins RAM oder getrennt in RAM und EPROM geladen werden soll. Dabei stellen sich ihm folgende Probleme in den Weg:

- Die Zieladressen im EPROM und die benötigten RAM-Bereiche ergeben sich als Summe der einzelnen Module und müssen vom Linker erst ermittelt werden.
- Die Module enthalten Definitionen und offene Referenzen, die komplett aufgelöst werden müssen.

Für den Programmierer, der von Hand ein EPROM zusammenstellen möchte, ergeben sich identische Probleme. Der RT-Link automatisiert die vormals notwendigen handwerklichen Eingriffe und minimiert somit die potentiellen Fehlerquellen. Dazu sind **einmalige** Eingriffe in die Quelltexte notwendig. Alle UH-PEARL-Module des Projektes **müssen** mit:

`SC=`*`modulsize`*`,CODE=$0,VAR=$0`

übersetzt werden. Sind diese Vorarbeiten abgeschlossen, so müssen bei späteren Änderungen in einzelnen Modulen nur noch diese neu übersetzt werden. Um hier Fehler auszuschließen, bietet sich bei größeren Projekten stets ein `MAKE`-Utility an, das für die Konsistenz des Endproduktes Sorge trägt.

Damit wäre schon ein Großteil des störenden Wasserkopfes bei der Erstellung von EPROM's mit einem Schlag entfallen. Die Zeit zur Herstellung eines EPROM's sinkt nach bisherigen Erfahrungen um den Faktor 10 bis 20 und die Gefahr von fehlerhaften Bedienereingriffen ist auf ein unumgängliches Minimum reduziert.

Bei weiteren Erklärungen des RT-Link setze ich nun einfach ihre Einsicht voraus, daß nur korrekt mit `CODE=$0` und `VAR=$0` übersetzte S-Records als Eingabefiles akzeptiert werden. Selbstredend werden auch S-Records des UH-Assemblers und des CREST-C-Compilers sinnvoll verarbeitet. In diesen Fällen entfällt sogar eine besondere Aufbereitung der Quelltexte.

Endprodukt von RT-Link sind – nach Auswahl der entsprechenden Optionen – binäre oder S-Recorddateien, die direkt an das EPROM-Programmiergerät überspielt werden können. Und mit dem Zwischenschritt – der Bedienung des Linkers – beschäftigt sich der Rest dieses Manuals.

2.6 Schnellkurs

Gesetzt den Fall, daß Sie lediglich einen Sack voll UH-PEARL- und UH-Assembler-Module in ein einzelnes EPROM brennen wollen, ist die Bedienung von RT-Link für Sie nach dem Durchlesen dieses kurzen Kapitels erschöpfend behandelt.

Auf der Diskette finden Sie den Ordner `PROBE`, mit dessen Hilfe ich nun erläutern möchte, wie man ein komplettes Projekt ins EPROM brennen kann. Zunächst mal zu den UH-PEARL-Quelltexten, die Ihnen in leicht abgewandelter Form vielleicht noch aus dem Schnellkurs der RTOS-UH-Manuals vertraut sind. Ursprünglich war das Programm dazu gedacht, den RTOS-UH-Einsteigern das Multitasking begreiflich zu machen. Um RT-Link zu erklären, habe ich daraus (unsinnigerweise) vier Module gemacht. Ansonsten sollten Sie sich erst gar nicht darum bemühen, nach Tiefsinn zu forschen. Für Anhänger bedruckten Papiers sind die hier verwendeten Dateien im Anhang Seite 24 nachzuschlagen.

Da sind also zunächst die vier Module PROBE1.P bis PROBE4.P. Sie enthalten ein paar Definitionen, Referenzen und Tasks. Die jeweils zweite Zeile enthält die Anweisung, CODE- und VAR-Bereiche auf die Adresse 0 zu legen. Die Angabe bei S=... bzw. SC=... **muß** groß genug sein, um das Ergebnis der Compilierung – den fertigen S-Record – aufnehmen zu können (nie mit SIZE LIMIT weiterwursteln), sollte jedoch nicht unnötig groß gewählt werden, da RT-Link diese Angabe bei der Anforderung des Speichers während des Linkerlaufes heranzieht und man schließlich nicht unnötigerweise einen Abbruch von RT-Link wegen Speicherplatzmangels zu forcieren braucht. Zu Ihrer Beruhigung sei noch angemerkt, daß RT-Link selbstverständlich während des Linkvorgangs für jedes Modul den real benötigten Speicherplatz ermittelt und auch nur den im EPROM belegt.

Damit können wir nun zur Compilierung schreiten. Kopieren Sie dazu die vier PEARL-Quelltexte von der RT-Link-Diskette in ihr Arbeitsverzeichnis. Der Tip, nicht mit Originaldisketten zu spielen, scheint mir bisweilen notwendig. Jetzt erzeugen wir S-Records und ein paar Listen:

```
P PROBE1.P > PROBE1.SR LO PROBE1.LST
P PROBE2.P > PROBE2.SR LO PROBE2.LST
P PROBE3.P > PROBE3.SR LO PROBE3.LST
P PROBE4.P > PROBE4.SR LO PROBE4.LST
```

In den Listen können Sie sich notfalls noch davon überzeugen, daß der Compiler tatsächlich Code für die gewünschten Adressen erzeugt hat. Wichtig ist nur die Angabe in der MODULE-SUMMARY, daß die CODE- und VAR-Segmente tatsächlich auf Adresse 0 liegen:

```
MODULE-SUMMARY:

TASKS:
(INT)Start   (EXT)Rest    (EXT)Machs   (EXT)Steuer

VAR(RAM):0000-0025  CODE(ROM):0000-0096 SHIFTABLE

$00BC BYTES  (FOR 68020+68881)       0 ERRORS.
```

Der Hinweis des PEARL-Compilers, daß das Modul PROBE1.P frei verschieblich ist (SHIFTABLE), spielt für RT-Link keine Rolle, da natürlich auch nicht verschiebbare Module von RT-Link auf jede beliebige Adresse gelegt werden können.

Nun kommt der eigentliche Linkvorgang. Dazu können Sie die Datei PROBE.RTL verwenden oder auch schnell neu eingeben. Die Extension .RTL für Linkdateien ist **zwingend** vorgeschrieben. Es wird Ihnen nicht gelingen, RT-Link davon zu überzeugen, Dateien mit anderslautenden Endungen zu akzeptieren.

In der Linkdatei sind sämtliche Informationen aufgeführt, die RT-Link zum Binden des Projektes benötigt. In der ersten Zeile befindet sich die Angabe, daß der Bereich von $10000 bis $1FFFF als freier RAM-Speicher des Zielsystems zur Verfügung steht (kann mit dem S-Kommando ermittelt werden, Werte des MARK eintragen). In der nächsten Zeile finden Sie eine entsprechende Definition für den verfügbaren EPROM-Speicherplatz, der sich von $20000 bis $2FFFF erstrecken soll. Bei den Adreßangaben wird zwingend die erste freie und die letzte freie Adresse verlangt. Die Angabe der Adressen erfolgt in hexadezimaler Schreibweise und das Dollarzeichen ist syntaktisch

vorgeschrieben. Die Angabe von RAM- und ROM-Bereichen hat so – und nicht anders – zu erfolgen.

```
RAM $10000 $1FFFF
ROM $20000 $2FFFF

USE MAIN
   /H1/C/PROBE1.SR
   /H1/C/PROBE2.SR
   /H1/C/PROBE3.SR
   /H1/C/PROBE4.SR
```

Die Zeile `USE MAIN` betrachten Sie erstmal als gottgewollte Notwendigkeit. Solange Sie nicht vorhaben, kompliziertere Dinge anzustellen, lohnt es sich nicht, Sie bereits im Schnellkurs mit Dingen zu verwirren, die Sie selten oder nie nutzen werden. Im Kapitel 2.8 können Sie sich bei Bedarf später schlaulesen. Wenn Sie die Zeile bei eigenen Linkdateien vergessen, werden Sie durch RT-Link daran erinnert, indem er Sie unnachgiebig darauf aufmerksam macht, daß er eigentlich nicht recht weiß, woher er den Speicher für die Module nehmen soll.

Gut, und nun kommt nur noch eine Liste der Module ihres Projektes. Die Syntax ist auch hier sehr restriktiv. Eine Datei pro Zeile und stets unter Angabe des kompletten Pathes in dieser – und keiner anderen – Schreibweise. Sollten Sie unter RTOS-UH zu den hartnäckigen Verfechtern von Punkt und Doppelpunkt als Devicetrenner zählen, dann können Sie mich gerne beschimpfen, aber ändern wird sich dadurch bestimmt nichts. Unter WINDOWS muß zuerst der Laufwerksbuchstabe, gefolgt von Doppelpunkt, Backslash und restlichem Pfad angegeben werden:

```
C:\C\PROBE1.SR
```

Die Angabe vollständiger Pathnamen mag Ihnen ebenfalls als Schikane erscheinen. Wenn Sie Besitzer eines totalen Gedächtnisses sind, dann kann ich Ihnen beipflichten: es ist Schikane! Für die wenigen Programmierer mit dem Makel von Gedächtnislücken halte ich es jedoch für angebracht, eine gewisse Dokumentierung der Projekte zu erzwingen. Es ist durchaus hilfreich, nach Monaten oder Jahren noch ein Stück Papier in Händen zu halten, auf dem man wenigstens noch nachlesen kann, welche Arbeitsumgebung damals zum Erfolg geführt hat. In meinem Fall lagen die Dateien zum Zeitpunkt der Erstellung des Handbuches eben auf `/H1/C/`. Für Ihren ersten Test müssen Sie nun also erstmal die Pfade entsprechend der aktuellen Arbeitsumgebung einrichten.

Wenn das vollbracht ist und Sie überprüft haben, daß keine Datei doppelt in der Liste steht, kann der Linker geladen und angeworfen werden.

```
RTL /H1/C/PROBE.RTL
```

Das wars dann auch schon. RT-Link schreibt Reklame und Versionsnummer auf den Schirm, klappert kurz und heftig mit der Platte und meldet sich mit dem Namen der Ausgabedatei, der benötigten Laufzeit und dem während der Linkphase dynamisch benötigten Speicher wieder bei Ihnen zurück:

```
IEP-Linker RTL                  Version for MC68020
Copyright (C) 1993              All Rights Reserved
Release 1.613                   Copyright by IEP Hannover
Serial Number -Master-          Jan 16 1993  13:11:34

Output file   :  MAIN.EPR

Link time     :  00:00:02     Allocated bytes  : 111510
```

Sie haben nun in der Datei `MAIN.EPR` einen S-Record, der die vier Module enthält und zum EPROM-Programmiergerät geschickt werden kann.

Für Ihre Projekte müssen Sie lediglich den RAM- und ROM-Bereich auf die Gegebenheiten des Zielsystems einstellen, meine Beispieldateien im Linkfile durch ihre Projektdateien ersetzen und den Linker starten. So einfach geht das. Wenn es Probleme gibt, sollten Sie das Handbuch schnell wieder rausholen und weiterlesen.

2.7 Crossentwicklung und Referenzen

Sehr schnell werden Sie feststellen, daß Beispiele zwar nett und schön sind, aber leider nicht alle Fragen beantworten. Ein Vorteil des RTOS-UH liegt darin, daß eine Reihe von Funktionen und Systemdiensten, deren sich der UH-PEARL-Compiler automatisch bedient, im Betriebssystem eingebettet sind und gemeinsam von allen Tasks genutzt werden können. Die Einsparung von EPROM-Speicherplatz ist recht erheblich. Aber bei der Erstellung von EPROM's gibts jetzt eben ein paar kleinere Probleme.

Erstmal zum Begriff der Crossentwicklung. Wenn Sie z.B. für einen kleinen Einplatinencomputer ein EPROM braten wollen, dann bietet sich dieser kleine Computer in den meisten Fällen nicht als Entwicklungsrechner an. RT-Link will (viel) Speicher sehen und wenn Sie nur popelige 128kB RAM auf der Platine haben, dann erübrigt sich ein Versuch, RT-Link zu starten. Der Speicherbedarf von RT-Link liegt aktuell bei etwa 80kB und dann fängt er auch noch an, Listen im Speicher aufzubauen, Dateien in Puffer zu lesen und ...

Nun, ohne ausreichend Speicher läuft jedenfalls nichts. Ich werde mich zwar in näherer Zukunft nochmal an die Optimierung des Speicherverbrauches innerhalb von RT-Link machen, aber Komfort und Geschwindigkeit gibt es leider niemals gratis.

Also ist eben Crossentwicklung angesagt. Die Programme werden auf einem *erwachsenen* Rechner geschrieben, übersetzt und erst zum Testen auf das Zielsystem überspielt. Ob die Crossentwicklung unter Linux, WINDOWS oder direkt auf RTOS-UH-System erfolgt, ist dabei völlig belanglos. Auf dem Zielrechner läuft dann schließlich ein RTOS-UH, mit dem man die S-Records in Empfang nehmen kann. Durch `LOAD` wird das Modul (oder die Module) in den Speicher geladen. Dabei werden automatisch Referenzen auf Symbole des Betriebssystems befriedigt. Soll heißen, daß Sie in einem PEARL-Programm zwar die Funktion `SIN` verwenden können, aber außer dem Aufruf keine Spuren einer Sinusberechnung innerhalb ihres S-Records vorfinden werden – sofern Sie nicht für FPU-Betrieb übersetzt haben. Erst der Lader stellt fest, daß die Funktion `SIN` nicht im S-Record enthalten ist und sucht im Betriebssystem nach dem entsprechenden Symbol. Ist er dabei fündig geworden, so trägt er die korrekte Adresse dieser Systemroutine an der Stelle ein, wo `SIN` aufgerufen wird – und damit sind dann alle glücklich und zufrieden!

Alle? Nein, denn RT-Link hat auf einem Crossentwicklungssystem diese Chance nicht! Was immer im Speicher eines Crossentwicklungsrechners steht, hat bestimmt keinerlei Bezug zu dem Micro-Controller, für den das EPROM gelinkt werden soll. Also wird sich RT-Link kurz und final zu dem Thema äußern und abbrechen, solange ihm nicht alle Symbole bekanntgemacht werden, die er in den S-Records als Referenz antrifft.

Um das zu verdeutlichen, sollten Sie im Modul `PROBE2.P` die Zeile

```
PUT x, x*x, 1.0/x TO Disp
```

durch

```
PUT x, SIN(x), 1.0/x TO Disp
```

ersetzen, das Modul neu übersetzen und nochmals linken. Die entsprechende Datei liegt auch als `PROBE2_1.P` auf der RT-Link-Diskette vor. Da in den Quelltexten der Beispieldateien explizit `P=68000` steht und der PEARL-Compiler nun keine Chance hat, Inline-Code für eine eventuelle FPU zu erzeugen, werden nun in dem S-Record Referenzen auf die Systemfunktion `#SSIN` auftauchen – so heißt der interne Anschluß der Sinusfunktion. Der Linker beschwert sich entsprechend mit

```
Unknown T-symbol #SSIN
```

über die fehlende Definition dieses T-Symbols innerhalb der ihm bekannten S-Records und bricht den Linkvorgang ab. Ihre Aufgabe besteht also darin, RT-Link mit den fehlenden Referenzen zu versorgen. Die Bezeichnung `T-symbol` bezieht sich auf die Syntax der S-Records, bei denen die Buchstaben `G` bis `Z` quasi als Escape-Sequencen wirken. Das `T` bedeutet lediglich eine Referenz auf ein Symbol, dessen Adresse dem PEARL-Compiler zum Zeitpunkt der Compilerung noch unbekannt war. Es können Ihnen bei der Verwendung von RT-Link noch andere Symbole-Bezeichner als Fehlermeldungen begegnen. Wenn Sie sich mit S-Records auskennen, haben Sie jetzt eine Chance, in den Eingabefiles zu wühlen. Als unbedarfter Anwender sollten Sie die zusätzliche Information schlicht ignorieren.

Zur Ermittlung des Definitionen ihrer System-EPROM's finden Sie ein kleines Tool namens RT-Cross auf der RT-Link-Diskette. Die Funktionsweise des RT-Cross sollten Sie nun im Kapitel 3 nachlesen.

Das Ergebnis dieses Ausfluges in die Welt der Referenzen dürfte nun sein, daß Ihnen jetzt bekannt ist, wie man auf einem Zielrechner an die dort definierten Symbole herankommt. Für unser Beispiel `PROBE` habe ich nun zum Demonstration ein Crossreferenzfile `PROBE.CRS` auf meinem heimatlichen Rechner erzeugt, das ebenfalls auf der Linkerdiskette und im Anhang auf Seite 1 zu finden ist.

Nun müssen wir RT-Link nur noch klarmachen, daß es sich bei der Datei `PROBE.CRS` nicht um einen regulären S-Record handelt, der auf eine bestimmte Zieladresse verschoben werden muß. Zu diesem Zweck erweitern wir das alte Linkfile `PROBE.RTL` aus dem Schnellkurs um zwei Zeilen:

```
RAM $10000 $1FFFF
ROM $20000 $2FFFF

USE AS REFERENCE
   /H1/C/PROBE.CRS

USE MAIN
   /H1/C/PROBE1.SR
   /H1/C/PROBE2_1.SR
   /H1/C/PROBE3.SR
   /H1/C/PROBE4.SR
```

Mit dem Ausdruck `USE AS REFERENCE` teilt man RT-Link mit, daß alle folgenden Dateien bis zum nächsten `USE ...` Crossdateien sind und die darin gespeicherten Adressen als Absolutwerte betrachtet werden müssen. An welcher Stelle der Linkdatei die Referenzen bekanntgemacht werden, ist dabei völlig gleichgültig. RT-Link sammelt zunächst alle Information innerhalb der S-Records einer Linkdatei auf und versucht erst dann, die Referenzen aufzulösen. Da **alle Symbole innerhalb sämtlicher S-Records global bekannt** sind, sind Sie dafür verantwortlich, daß es nicht zu Doppeldefinition kommt.

Wenn Sie das Beispielprogramm mit dem Sinus mit meinem Cross-File erneut linken, so kennt der Linker jetzt die Adresse in meinem Zielsystem und erzeugt den passenden S-Record. Beim Umgang mit Cross-Files sollten Sie geradezu sklavisch darauf achten, immer mit aktuellen Versionen zu arbeiten. Fehler, die auf die Verwendung falscher – vorzugsweise alter – Cross-Files basieren, sind nicht gerade leicht zu identifizieren. Die Task, die einen solchen falschen Anschluß benutzt, landet im Wald und es ist pures Glück, wenn man wenigstens noch eine Fehlermeldung des RTOS-UH an den Kopf geworfen bekommt bevor der Rechner in die ewigen Jagdgründe eingeht.

2.8 Symbolische Namen für Bereiche

Nun kommen wir zur Verwaltung mehrerer RAM- und ROM-Bereiche innerhalb eines Projektes. Dazu wollen wir uns ein neues Linkfile schreiben, in dem ein zusätzlicher ROM-Bereich definiert wird, der von `$30000` bis `$3FFFF` reicht. Dabei benutzen wir ein paar neue syntaktische Elemente von RT-Link. Betrachten wir zunächst die erste Zeile des neuen Linkfiles `PROBE_2.RTL`. Dort sehen Sie die Vereinbarung eines symbolischen Namens. Ein Symbol beginnt stets mit einem Buchstaben und kann mit einer beliebigen Folge aus Buchstaben, Zahlen und dem Underline fortgesetzt werden. Abgeschlossen wird ein solcher Symbolname durch einen Doppelpunkt. Jedes Symbol muß vor seiner ersten Anwendung in anderen Kommandos bereits definiert worden sein.

```
MAIN: RAM $10000 $1FFFF
      ROM $20000 $2FFFF
      ROM $30000 $3FFFF

USE AS REFERENCE
   /H1/C/PROBE.CRS

USE MAIN
   /H1/C/PROBE1.SR
   /H1/C/PROBE2_1.SR
   /H1/C/PROBE3.SR
   /H1/C/PROBE4.SR

WRITE MAIN TO /H1/C/PROBE2.EPR
```

Was haben wir damit nun angerichtet? Wenn Sie ein Symbol definiert haben, so beginnt RT-Link damit, alle folgenden Definitionen von RAM- und ROM-Bereichen aufzusammeln. Abgebrochen wird dieser Vorgang erst, wenn ein anderes Symbol entdeckt wird. Das Symbol `MAIN` steht deshalb jetzt für den RAM-Bereich von `$10000` bis `$1FFFF`, die ROM-Bereiche von `$20000` bis `$2FFFF` und `$30000` bis `$3FFFF`. Die Adressen der ROM-Bereiche, die innerhalb eines Symbols zusammengefaßt werden, müssen in aufsteigender Reihenfolge angegeben werden. Demnach ist das folgende Beispiel unzulässig:

```
MAIN:
      RAM $10000 $1FFFF
      ROM $30000 $3FFFF
      ROM $20000 $2FFFF
```

Das Symbol `MAIN` spielt unter den Symbolnamen eine besondere Rolle. Solange Sie noch kein Symbol vereinbart haben, benutzt RT-Link automatisch das Symbol `MAIN`, um RAM- und ROM-Bereiche aufzusammeln. Diese Besonderheit wurde ja bereits im Kapitel 2.6 in der Linkdatei `PROBE.RTL` ausgenutzt. Die explizite Vereinbarung von `MAIN` ist demnach eigentlich redundant.

Wenn Sie das File `PROBE_2.RTL` durch den Linker jagen, so werden Sie feststellen, daß sich der Name der Ausgabedatei verändert hat. Ansonsten entspricht der S-Records Byte für Byte dem Versuch mit dem Linkfile `PROBE_1.RTL`. Der Linker hat die vier Module immer noch im ersten EPROM unterbringen können und deshalb den zweiten ROM-Bereich schlicht ignoriert.

Weitere Informationen zum `WRITE`-Kommando sind im Kapitel 2.9 nachzulesen. Einstweilen sollte das Wissen ausreichen, daß Sie mit `WRITE` *`symbol`* `TO` *`filename`* die ROM-Bereiche, die zu *`symbol`* vermerkt sind, als S-Record-Datei *`filename`* abspeichern können.

Nun kommen wir zum eigentlichen Anwendungszweck des Symbole. In `PROBE_2.RTL` sind wir davon ausgegangen, daß der Linker die freie Auswahl hat, in welchem ROM-Bereich er die S-Records unterbringen kann. Dabei verfolgt RT-Link stets folgende Strategie:

1. Zunächst werden sämtliche Dateinamen in der Reihenfolge ihrer Definition im Linkfile aufgesammelt.

2. Zu jedem Dateinamen bestimmt RT-Link aus dem zugehörigen `USE`-Kommando, welche RAM- und ROM-Bereiche für eben diese Datei erlaubt sind. In unserem Beispiel wird das Symbol `MAIN` verwendet und es werden die drei dort gespeicherten Bereiche gefunden. Hinter `USE` kann jedoch auch eine mit Kommas getrennte Liste von Symbolen auftauchen, die dann von links nach rechts ausgewertet wird.
3. Jetzt wird die Dateiliste der Reihe nach abgearbeitet. Dabei werden die RAM- und ROM-Bereiche in der Reihenfolge ihrer Definition aufgefüllt. Die ROM-Bereiche werden dabei von niedigen zu hohen Adressen vergeben. Die RAM-Bereiche füllt RT-Link von hohen zu niedrigen Adressen auf.
4. Erst wenn ein Modul Speicheranforderungen stellt, die nicht mehr in den aktuell in Bearbeitung befindlichen RAM- oder ROM-Bereich gelegt werden können, fährt er mit dem nächsten zulässigen Bereich fort. Ist kein weiterer Bereich mehr definiert, so bricht er mit einer Fehlermeldung ab:

   `RAM-area(s) too small to allocate $`*`xx`*` bytes`

   oder

   `ROM-area(s) too small to allocate $`*`xx`*` bytes`

   Für Sie bedeutet das schlicht, daß das EPROM zu klein für das Projekt ist oder ein paar RAM's nachgestöpselt werden sollten.

Bislang haben Sie mir glauben müssen, daß RT-Link sich so verhält. Durch den Aufruf von:

```
RTL /H1/C/PROBE_2.RTL -M
```

veranlassen wir den Linker nun dazu, uns eine MAP-Datei zu erzeugen, in der sein Tun dokumentiert ist. RT-Link erzeugt aus dem Namen der Linkdatei automatisch den Namen der MAP-Datei. Dabei wird einfach die Extension `.RTL` durch die Dateiendung `.MAP` ersetzt. In unserem Fall erhalten wir demnach die MAP-Datei `/H1/C/PROBE_2.MAP`, die Sie im Anhang Kapitel 4.3 finden.

Hinter der unvermeidlichen Reklame und der Versionsnummer finden Sie dort zunächst den Namen der verwendeten Link- und Listdatei abgespeichert. Es folgen – zur besseren Dokumentation – auch noch Datum und Uhrzeit des Linkerlaufes.

Nun kommt der interessante Teil der MAP-Datei. Hinter `ROM`-Ranges finden Sie eine Auflistung sämtlicher von Ihnen definierter ROM-Bereiche des Linkfiles.

```
00020000 -> 0002FFFF  MAIN
|           |         |
|           |         +--  Symbol fuer den ROM-Bereich
|           +--  Endadresse des ROM-Bereiches
+--  Startadresse des ROM-Bereiches
```

Hinter dem durchgezogenen Strich ist die Belegung des jeweiligen ROM-Bereichs aufgelistet. Dabei sind zwei Möglichkeiten zu beachten. Betrachten Sie zunächst die Information zum Modul `/H1/C/PROBE1.SR` mit jeweils zwei Einträgen zu jedem Modul innerhalb der ROM-Liste.

```
00020000 -> 00020095  /H1/C/PROBE1.SR       CODE
|           |         |
|           |         +--  Filename des S-Records
|           +--  Endadresse des CODE-Bereichs
+--  Startadresse des CODE-Bereichs
```

In der obigen Abbildung können Sie die Lage des Code-Bereiches des Moduls `PROBE1.SR` ablesen.

```
0002027C -> 000202B7  /H1/C/PROBE1.SR       DATA
|           |         |
|           |         +--  Filename des S-Records
|           +--  Endadresse des DATA-Bereichs
+--  Startadresse des DATA-Bereichs
```

Entsprechend ist hier die Position der Initialdaten des Moduls zu entnehmen. Die dort gespeicherten Informationen veranlassen RTOS-UH bei Kaltstart dazu, entsprechende RAM-Speicherbereiche für das jeweilige Modul einzurichten und zu initialisieren.

Zum Abschluß jedes nicht komplett gefüllten ROM-Bereiches finden Sie zusätzlich die Angabe des nicht benutzten Speicherplatzes:

```
00020372 -> 0002FFFF                    FC8E Bytes free
```

Aus dem Listfile können Sie weiterhin erkennen, daß RT-Link den ROM-Bereich von `$30000` bis `$3FFFF` nicht angerührt hat.

Wenn alle ROM-Bereiche ausgedruckt wurden, wird zudem die geplante Belegung des RAM-Speichers durch das Projekt angezeigt. Der Aufbau ist identisch zu dem Aufbau der ROM-Listen. Da allerdings der RAM-Speicher von hohen zu niedrigen Adressen verteilt wird, finden Sie den Freispeicher zu Beginn der Liste und dann erst die belegten Segmente.

Stellen Sie unbedingt sicher, daß die angegebenen RAM-Bereiche auf dem Zielsystem tatsächlich `FREE` sind und die Adressen sinnvolle Werte aufweisen. Sinnvoll heißt z.B., daß die Startadressen **gerade** sind und tatsächlich RAM's dort zu finden sind. Wenn dies nicht der Fall ist, brennen Sie sehr schweigsame EPROM's, denn wenn RTOS-UH bei der Einrichtung der RAM-Bereiche scheitert, bekommen Sie beim Einschalten keinerlei Reaktion mehr von ihrem Zielrechner! Am einfachsten ermitteln Sie die RAM-Bereiche durch Eingabe des `S`-Kommandos. Die Adressen, die bei `MARK` stehen, tragen Sie in Ihre RTL-Datei ein. Besitzt Ihr Rechner mehrere RAM-Bereiche, die durch `NORAM`-Module getrennt sind, so müssen Sie auch mehrere RAM-Bereiche in die RTL-Datei eintragen.

Kommen wir nun zu etwas komplexeren Anwendungsfällen. Sinn des Spieles soll diesmal sein, die Verteilung der Module auf die ROM-Bereiche zu steuern. Bislang ist das Modul `PROBE1.SR` immer im Bereich von `$20000` bis `$2FFFF` untergebracht worden. Diesmal soll RT-Link angewiesen werden, den Code des Moduls ab `$30000` abzulegen. `PROBE2_1.SR` soll dagegen definitiv im Bereich von `$20000` bis `$2FFFF` verstaut werden. Bei `PROBE3.SR` ist uns die Vergabe dann egal, aber wenn möglich, soll RT-Link den ersten Bereich nutzen. Umgekehrt soll `PROBE4.SR` möglichst im zweiten Bereich landen – wenn das nicht mehr passen sollte, wäre allerdings auch der erste ROM-Bereich noch verfügbar.

```
RAM_BEREICH: RAM $10000 $1FFFF
ERSTES_ROM:  ROM $20000 $2FFFF
ZWEITES_ROM: ROM $30000 $3FFFF

USE AS REFERENCE
   /H1/C/PROBE.CRS

USE ZWEITES_ROM, RAM_BEREICH
   /H1/C/PROBE1.SR

USE ERSTES_ROM, RAM_BEREICH
   /H1/C/PROBE2_1.SR

USE ERSTES_ROM, ZWEITES_ROM, RAM_BEREICH
   /H1/C/PROBE3.SR

USE ZWEITES_ROM, ERSTES_ROM, RAM_BEREICH
   /H1/C/PROBE4.SR

WRITE ERSTES_ROM  TO /H1/C/PROBE3_1.EPR
WRITE ZWEITES_ROM TO /H1/C/PROBE3_2.EPR
```

Diesmal wurden drei Symbole verwendet, um auf jeden einzelnen Bereich namentlich zugreifen zu können. Im Anhang Kapitel 4.3 finden Sie die zugehörige MAP-Datei. Eine Erklärung erübrigt sich, denn RT-Link hat die Dateien exakt so aufgeteilt, wie von uns gewünscht.

Im Unterschied zum letzten Beispiel sind hier die ROM-Bereiche von `$10000` bis `$1FFFF` und `$20000` bis `$2FFFF` **getrennt** in zwei Dateien herausgeschrieben worden. Alternativ – oder zusätzlich – hätte hier auch die Möglichkeit bestanden, eine einzelne Datei zu generieren.

```
WRITE ERSTES_ROM, ZWEITES_ROM TO /H1/C/PROBE3_X.EPR
```

Durch diesen Befehl werden sämtliche ROM-Bereiche von `ERSTES_ROM` und `ZWEITES_ROM` gemeinsam in der Datei `/H1/C/PROBE3_X.EPR` abgespeichert.

An diesem Punkt sollten Sie erstmal mit dem Linker spielen, um einen Eindruck vom Verhalten des RT-Link zu bekommen. Die Gruppierung von RAM- und ROM-Bereichen unter symbolischen Namen ist der einzige Punkt des Linkers, der etwas Übung verlangt. Die folgenden Kapitel behandeln nur noch Variationen und Möglichkeiten bereits vorgestellter RT-Link-Kommandos.

2.9 Ausgabe von EPROM-Dateien

Kommen wir nun zu einer detaillierten Beschreibung der Ausgabemöglichkeiten. Sie kennen aus den bisherigen Beispielen bereits das `WRITE`-Kommando, mittels dessen Sie die Ergebnisse des Linkerlaufes abspeichern können. Die vollständige Syntax lautet:

`WRITE` *`writelist`* `TO` *`filename`* `[BY` *`name`* `]`

Ein paar Anwendungsbeispiele kennen Sie ja bereits aus den bisherigen Versuchen:

```
WRITE MAIN                    TO /H1/C/PROBE2.EPR
WRITE ERSTES_ROM              TO /H1/C/PROBE3_1.EPR
WRITE ERSTES_ROM, ZWEITES_ROM TO /H1/C/PROBE3_X.EPR
```

Mit [BY *name*] haben Sie nun die Möglichkeit, weiteren Einfluß auf die Form der Ausgabedatei(en) zu nehmen. Fehlt der Hinweis auf ein Format, so wird – wie bereits mehrfach demonstriert – automatisch ein UH-S-Record erzeugt. Das Kürzel *name* steht dabei wiederum für einen symbolischen Namen, den Formatnamen. Betrachten wir dazu das folgende Beispiel:

```
FORMAT beispiel
   BINARY
ENDFORMAT
```

Ein Formatblock wird mit dem Schlüsselwort FORMAT und einem eindeutigen Namen – hier eben beispiel – eingeleitet. Beendet wird ein solcher Block durch das Schlüsselwort ENDFORMAT. Dazwischen können zeilenweise die Angaben erfolgen, in welcher Form die Ausgabedatei beeinflußt werden soll. In unserem Fall merkt sich RT-Link, daß es gilt, eine Binärdatei zu schreiben.

```
WRITE MAIN       TO /H1/C/PROBE2.EPR   BY beispiel
WRITE ERSTES_ROM TO /H1/C/PROBE3_1.EPR BY beispiel
```

Sie erhalten bei Ausgaben gemäß obiger Abbildung Binärfiles, die exakt die Länge der von RT-Link ermittelten Nutzdaten haben. Es fällt auf, daß eines der Beispiele fehlt.

```
WRITE ERSTES_ROM, ZWEITES_ROM TO PROBE3_X.EPR BY beispiel
```

Der Grund dafür ist leicht einsehbar, er wird im folgenden erklärt. Bei der Ausgabe von S-Records fällt es leicht, Bereiche, die keine Nutzdaten enthalten, zu ignorieren. Jede S-Recordzeile beginnt schließlich mit einer Adresse, an der die folgenden Daten untergebracht werden können. Treten Lücken (GAPS) in den Nutzdatenbereichen auf, so ist es nicht unbedingt notwendig, diese Lücken mit Dummy-Zeilen in den S-Records aufzufüllen. Bei S-Records ist das zwar möglich, aber keineswegs nötig.

Anders sieht es bei Binärdateien aus. Dort steht nur eine Adresse fest: die Startadresse der Nutzdaten der Datei! Beim obigen Beispiel können Sie jedoch sehr schnell sehen, daß unsere beiden ROM-Bereiche große Löcher enthalten, in denen keine Nutzdaten untergebracht sind. Die Nutzdaten in unserer Binärdateien nun schlicht hintereinander abzulegen, wäre wohl etwas zu roh. Deshalb füllt RT-Link die Freibereiche mit dem Byte-Muster $FF auf – was Ihnen im Gegensatz zu anderen Mustern ermöglicht, die freien Bereiche des EPROM's später nachzubrennen. Im nächsten Beispiel können Sie sehen, daß dieses Default-Muster sich auch übersteuern läßt.

```
FORMAT beispiel2
   BINARY
   FILL GAPS WITH $AA
ENDFORMAT
```

Hier würden die leeren Bytes innerhalb der ROM-Bereiche durch $AA ersetzt und in die Datei geschrieben.

Derartige Aktionen sind selbstverständlich auch bei S-Records machbar – aber nicht immer besonders sinnvoll. Im nächsten Beispiel sehen Sie die Definition eines Formatblockes zu Erzeugung von S-Records, bei denen auch die Lücken mit Füllmustern beschrieben werden. *Per default* generiert

RT-Link in solchen Situationen allerdings nur Adreßsprünge in den Records und spart damit viel Platz auf dem Ausgabemedium.

```
FORMAT beispiel3
   S_RECORD
   FILL GAPS WITH $AA
ENDFORMAT
```

Die Adressen in S-Records werden unter RTOS-UH von Adresse `$0` ab vergeben. Der `LOAD`-Befehl und die meisten EPROM-Programmiergeräte sind mit dieser Syntax durchaus glücklich und zufrieden. In der nächsten Abbildung sehen Sie die ersten paar Spalten eines normalen UH-S-Records, die zum Zwecke der Lesbarkeit hinter dem `S`*x*-Symbol, der Zeilenlänge, der Adreßangabe und den Nutzdaten durch Leerzeichen aufbereitet wurden.

```
S0 06 000190 1000 58
S2 3C 000000 AEB1BF95...
S2 3C 000038 FFC60042...
S2 3C 000070 00544E4E...
S2 1A 0000A8 0001FFF8...
S2 3C 0000BE AEB1BF95...
S2 26 0000F6 FFFCF239...
S2 18 000118 AEB1BF95...
S2 32 00012C 0001FFB4...
S2 08 00015A 00000000 9C
S2 18 00015E AEB1BF95...
S2 1E 000172 0001FF98...
S2 08 00018C 00000000 6A
S9 08 000190 00000000 66
|  |  |       |
|  |  |       +--  Daten + Checksumme
|  |  +--  Adress- oder Größenfeld
|  +--  Restdatenlänge
+--  S-Kennung
```

Wenn Sie allerdings im Besitz eines EPROM-Brenners sind, der absolute Adreßangaben verlangt, wird Sie die Nummerierung `$00`, `$38`, `$70`, `$A8` in den S2-Zeilen nicht heiter stimmen.

```
FORMAT beispiel4
   ABSOLUT
ENDFORMAT
```

In obiger Abbildung sehen Sie, daß RT-Link durch Angabe des Schlüsselwortes `ABSOLUT` innerhalb eines Formatblockes dazu überredet werden kann, seine S-Recordausgaben mit absoluten Adressen zu tätigen. Entsprechend sähe das Ergebnis dann entsprechend der gewählten ROM-Bereiche so oder ähnlich aus:

```
S0 06 000190 1000 58
S2 3C 020000 AEB1BF95...
S2 3C 020038 FFC60042...
S2 3C 020070 00544E4E...
S2 1A 0200A8 0001FFF8...
S2 3C 0200BE AEB1BF95...
S2 26 0200F6 FFFCF239...
S2 18 020118 AEB1BF95...
S2 32 02012C 0001FFB4...
S2 08 02015A 00000000 7C
S2 18 02015E AEB1BF95...
S2 1E 020172 0001FF98...
S2 08 02018C 00000000 4A
S9 08 000190 00000000 46
|  |  |       |
|  |  |       +--  Daten + Checksumme
|  |  +--  Adress- oder Größenfeld
|  +--  Restdatenlänge
+--  S-Kennung
```

2.10 Autostartfähige Tasks

In den meisten Fällen ist es sinnvoll, EPROM's so zu erstellen, daß nach dem Einschalten des Rechners sofort Aktionen anlaufen. Dies erreicht man bekanntlich durch autostartfähige Tasks. Leider gibt es im UH-PEARL keine Möglichkeit, bereits bei der Compilierung festzulegen, welche Tasks autostartfähig sein sollen. Deshalb übernimmt RT-Link die Aufgabe, Tasks diese Eigenschaft zuzusprechen. Ihre Aufgabe besteht lediglich darin, dem Linker eine Liste der Tasknamen in geeigneter Form ins Linkfile zu schreiben. In der folgenden Abbildung werden Verwendungen des Schlüsselwortes `AUTOSTART` vorgestellt.

```
AUTOSTART task1
AUTOSTART task2, task3
AUTOSTART task4, task5, task6
```

Es folgt der Name einer einzelnen Task – oder eine durch Kommata getrennte Liste von Tasknamen. An welcher Stelle Sie diese Anweisungen im Linkfile unterbringen, ist dabei völlig gleichgültig. RT-Link durchsucht die zu linkenden Dateien nach den Tasks und meldet bei eingeschalteter `-V`-Option auch den Erfolg seiner Aktionen. Wird eine oder mehrere Task(s) dabei nicht aufgefunden, so bricht er seinen Linkvorgang mit einer entsprechenden Fehlermeldung ab.

Für das Beispiel `PROBE` könnte z.B. die Task `Start` mittels `AUTOSTART Start` autostartfähig gemacht werden. Das Demo würde dann selbstständig anlaufen.

2.11 Kommentare in Linkfiles

Die bisherigen Beispiel-Linkdateien enthielten nur die unbedingt notwendigen syntaktischen Mittel zur Lösung der anstehenden Beispielaufgaben. In der Praxis sind Linkdateien wie das allererste

Beispiel eher für die Tonne. Mal abgesehen von der Tatsache, daß PROBE*x*.SR als Dateiname wohl eher nicht selbstdokumentierend ist, stehen keinerlei Information innerhalb der Linkdatei, die eine spätere Wartung und Pflege des Projektes erleichtern könnten.

```
REMARK ##############################################
REMARK #    Beispiel-Linkdatei zur Demonstration    #
REMARK #    des RT-LINK bei einem RAM- und einem    #
REMARK #    ROM-Bereich.                            #
REMARK ##############################################

; Unbenamte RAM- und ROM-Bereiche zur Demonstration
; der default-Vorgaben des RT-Link
RAM $10000 $1FFFF
ROM $20000 $2FFFF

USE MAIN
   /H1/C/PROBE1.SR  ; Enthaelt die Haupttask
   /H1/C/PROBE2.SR  ; weitere Tasks des Projekts PROBE
   /H1/C/PROBE3.SR  ; "                           "
   /H1/C/PROBE4.SR  ; "                           "

; Das Ergebnis wird als automatisch als S-Record-Datei
; MAIN.EPR abgespeichert.
```

Hier sehen Sie zwei Arten, Kommentare in Linkfiles unterzubringen. Zeilenkommentare beginnen hinter einem Semikolon. Alle Zeichen hinter dem ; werden beim Parsen des Linkfiles ignoriert. Die zweite Möglichkeit besteht in der Verwendung des Schlüsselwortes REMARK oder REM. Die Texte, die hinter REMARK stehen, werden von RT-Link aufgesammelt und im Kopf des MAP-Files ausgegeben – sofern Sie eine MAP-Datei mittels der Linkeroption -M angefordert haben. Somit steht Ihnen eine einfache Möglichkeit zur Verfügung, Informationen auch in der Protokoll-Datei zu sammeln.

2.12 Symboltabellen im MAP-File

Während des Linkerlaufes sammelt RT-Link alle in den S-Records auftretenden lokalen und globalen Symbole auf. Die globalen Symbole liegen namentlich in den zu linkenden S-Records vor und sind RT-Link folglich bekannt. Zu Debug-Zwecken kann es hilfreich sein, die Lage von Variablen im System zu kennen. Deshalb ermöglicht RT-Link die Ausgabe der gespeicherten Symbole und deren Adressen im MAP-File. Die Syntax lautet dazu:

```
SYMBOLTABLE ADDRESSES
SYMBOLTABLE ALPHABETICAL
```

In ersten Fall werden die Symbole nach den Adressen sortiert in die Datei geschrieben. Im zweiten Fall sortiert RT-Link nach dem Alphabet. In den Tabelle sind alle globalen Symbole aller Linkdateien enthalten. Da RT-Link zusätzlich auch nach 17er-Scheiben innerhalb der Linkdateien sucht, befinden sich auch diese Symbole in den Tabellen.

2.13 Splitten von Binärfiles

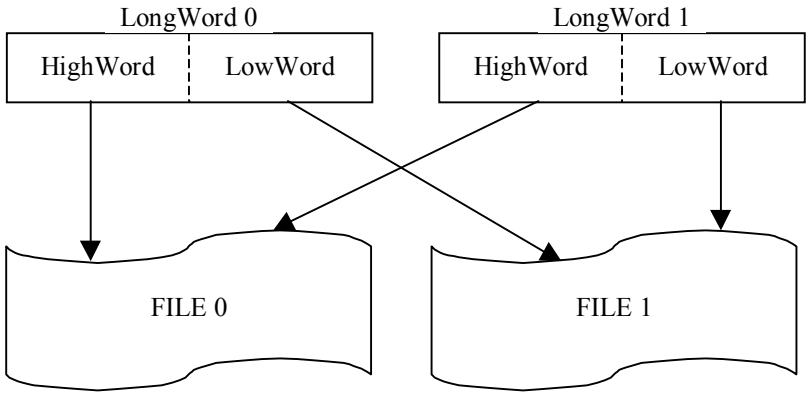
Üblicherweise erzeugt RT-Link Ausgabedateien, in denen die Daten linear abgespeichert sind. Wenn die zu erstellenden EPROMs byte- oder wortweise orientiert sind, übernimmt in der Regel der Eprommer die Aufgabe, den linearen Datenstrom in geeigneter Form auf die einzelnen Eproms zu verteilen. Durch zusätzliche Angaben innerhalb der `FORMAT`-Blöcke läßt sich jedoch auch RT-Link zum Aufteilen der Daten in verschiedene Dateien überreden.

Mittels des `FORMAT`-Blockes `BINAER_LONG_WORD` und des dort enthaltenen `WRITE`-Kommandos werden nun zwei Dateien (`TEST_0.EPR` und `TEST_1.EPR`) generiert.

```
FORMAT BINAER_LONG_WORD
   BINARY
   SPLIT LONGS TO WORDS
ENDFORMAT
```

Der anzugebende Name der Ausgabedatei **muß** das Escapezeichen `$` enthalten. Dieses Zeichen wird von RT-Link durch die Ziffern '`0`' und '`1`' ersetzt, wobei die Datei mit der Kennung '`0`' die High-Words enthält.

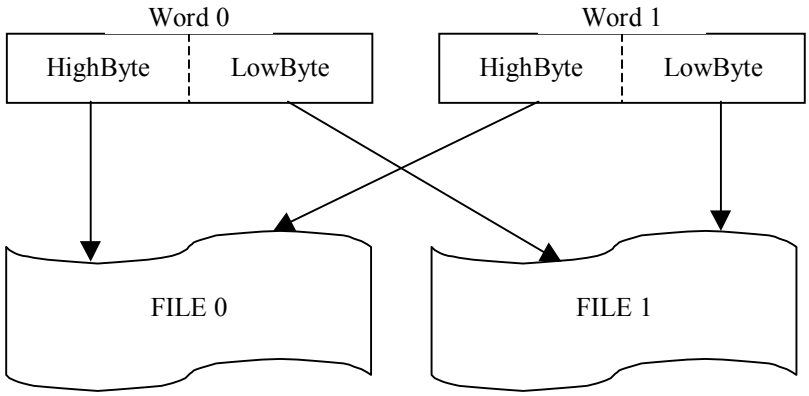
```
WRITE TEST TO TEST_$.EPR BY BINAER_LONG_WORD
```



Da es immer noch Eprommer gibt, die das Aufteilen auf verschiedene EPROMs nicht beherschen, wird auch das Aufteilen von Worten in Bytes und von Langworten in Bytes unterstützt.

```
FORMAT BINAER
   BINARY
   SPLIT WORDS TO BYTES
ENDFORMAT
```

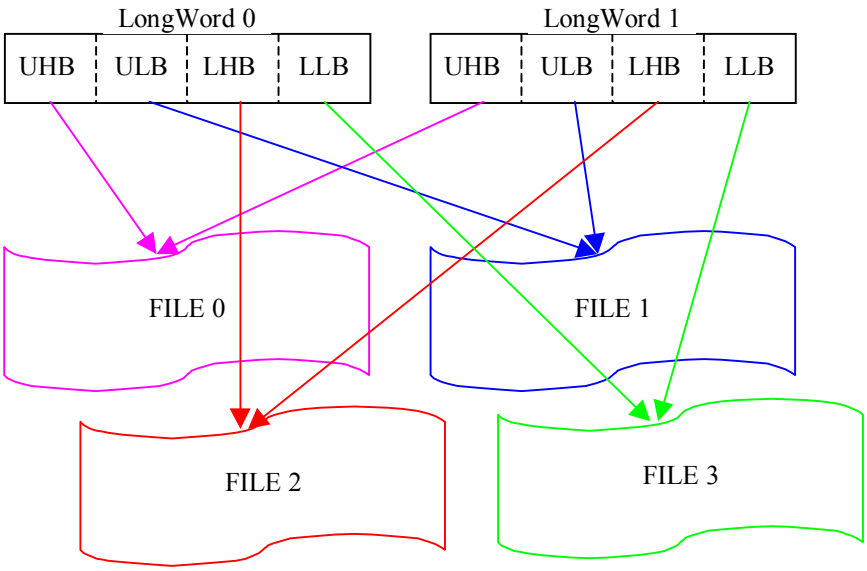
Auch hier muß wieder das Escapezeichen `$` angegeben werden, um RT-Link anzuzeigen, wo er die Nummerierung der Dateien unterbringen darf.



Es werden zwei Dateien erzeugt, die Datei mit der Nummer '`0`' enthält die High-Bytes.

Mit dem folgenden Format werden die Daten auf 4 Dateien verteilt:

```
FORMAT BINAER
   BINARY
   SPLIT LONGS TO BYTES
ENDFORMAT
```



Die Aufteilung der Bytes auf die 4 Dateien ist obigem Bild zu entnehmen.

3 CROSS für Referenzen

Das Tool RT-Cross ermöglicht es, die globalen Symbole eines RTOS-UH-Rechners zu ermitteln. Es befindet sich als `CROSS` auf der RT-Link-Diskette und muß vor seiner Anwendung auf dem RTOS-UH-Rechner geladen werden, dessen Definitionen erfaßt werden sollen. Der Aufruf von RT-Cross liefert eine Usage-Meldung der Form:

```
IEP-Cross-Referenz CROSS      Version for MC68000
Copyright (C) 1993            All Rights Reserved
Release 1.602                 Copyright by IEP Hannover
Serial Number -Master-        Jan 04 1993  15:20:49

Usage:CROSS [outputfile] [-options]

  -C                          Generate REF-File
  -N name [ name ]            Scan named module
  -R address range [offset]   Scan memory range
  -S                          Scan scanner ranges
  -V                          Verbose messages
```

Für Sie als Anwender von RT-Link ist dabei nur die Option `-S` von Bedeutung. Der RT-Cross durchsucht dann die Scan-Ranges des Rechners nach Symboldefinitionen. Diese werden, sofern Sie kein Ausgabefile angegeben haben, als `CROSS.CRS` im aktuellen Working-Directory abgespeichert. Sie sollten darauf achten, daß das aktuelle Working-Directory **nicht** auf `/ED` steht, sondern auf `/EDX`, sonst erhalten Sie nur Müll in der Cross-Datei!

Eine solche Datei besteht aus S3-Records, in denen die Definitionen der gefundenen Symbole abgelegt sind. In der nächsten Abbildung sehen Sie die Definition des Symbols `GET_TASKNAME` auf Adresse `$DAB2`, mittels derer sich der Aufbau einer Zeile innerhalb des Crossfiles erläutern läßt.

```
S3 0D 0000DAB2 U0 0C GET_TASKNAME 72
|  |  |         |  |  |             |
|  |  |         |  |  |             +-- Checksumme
|  |  |         |  |  +-- Name des Symbols
|  |  |         |  +-- Länge des Symbols
|  |  |         +-- Escape-Sequence: Symboldefinition
|  |  +-- Zieladresse der S-Recordzeile
|  +-- Restlänge der S-Recordzeile
+-- Startsymbol der S-Recordzeile
```

Die so ermittelte `.CRS`-Datei können Sie nun wieder auf den Entwicklungsrechner transferieren und die bislang offenen Systemeinsprünge ergänzen.

RT-Cross ermöglicht es auch, die globalen Symbole geladener Module zu ermitteln. Dazu müssen Sie den Cross mit der Option `-N` und einem oder mehreren Modulnamen aufrufen.

Entsprechend lassen sich auch Symbole bestimmen, die innerhalb eines bestimmten Speicherbereiches liegen. Hier greift die Option `-R` gefolgt von Anfangsadresse und Bereichslänge. Optional lassen sich durch Angabe eines dritten Parameters die ermittelten Adressen der Symbole auf die neue Startadresse *offset* umrechnen.

Die Option `-C` ermöglicht die Generierung von `REF`-Files, die der Linker des CREST-C-Compilers verwendet, um offene Referenzen einzubinden. Für PEARL-Programmierer sind diese binären Dateien ohne weitere Bedeutung.

Für kleinere Rechner, die den Einsatz des RT-Cross nicht erlauben, ist dem Paket das Programm `XREF` in der Datei `IEPXREF` in Form unter RTOS-UH ladbarer S-Records beigefügt. `XREF` besteht aus dem Modul `IEPXREF` und der Task `IEPXREF`. Diese Task erzeugt nach ihrer Aktivierung eine Datei mit dem Namen `/ED/CROSS`, die als Cross-Datei für RT-Link verwendet werden kann und die globalen Adressen des Rechners, auf dem sie erzeugt wurde, enthält.

4 ANHANG

4.1 Beispieldateien des Schnellkurses

```
P=68000;
SC=$1000,CODE=$0,VAR=$0;

MODULE PROBE1 ;

SYSTEM ;

PROBLEM ;

   SPC (Steuer,Machs,Rest) TASK  GLOBAL ;
   DCL Ausgabenummer       FIXED GLOBAL ;
   DCL (x,Leer)            FLOAT GLOBAL ;

Start: TASK PRIO 48 ;
   x             = 1.0 ;
   Leer          = 0.0 ;
   Ausgabenummer =  1  ;

   ALL 1 SEC ACTIVATE Steuer ;

   ACTIVATE Machs ;
   ACTIVATE Rest  ;
END;

MODEND;
```

Das Beispielmodul PROBE1.P

```
P=68000;
SC=$1000,CODE=$0,VAR=$0;

MODULE PROBE2 ;

SYSTEM ;
   Disp : A1 <-> ;

PROBLEM ;

   SPC Disp  DATION INOUT ALPHIC ;
   SPC x     FLOAT GLOBAL        ;

Machs: TASK PRIO 50 ;
   REPEAT
      PUT x, x*x, 1.0/x TO Disp
          BY SKIP, (2)F(20), E(20,7), SKIP ;
      x = x + 1.0 ;
   END;
END;

MODEND;
```

Das Beispielmodul `PROBE2.P`

```
P=68000;
SC=$1000,CODE=$0,VAR=$0;

MODULE PROBE3 ;

SYSTEM ;

PROBLEM ;

   SPC Leer  FLOAT GLOBAL ;

Rest: TASK PRIO 100 ;
   REPEAT
      Leer = Leer + 1.0 ;
   END;
END;

MODEND;
```

Das Beispielmodul PROBE3.P

```
P=68000;
SC=$1000,CODE=$0,VAR=$0;

MODULE PROBE4 ;

SYSTEM ;

   File : ED.Daten  -> ;

PROBLEM ;

   SPC File          DATION   OUT ALPHIC ;
   SPC Ausgabenummer FIXED GLOBAL ;
   SPC (x,Leer)      FLOAT GLOBAL ;
   SPC (Machs,Rest)  TASK GLOBAL  ;

Steuer: TASK PRIO 49 ;
   PUT Ausgabenummer, x, Leer TO File
       BY F(8), (2)F(20), SKIP ;

   IF Ausgabenummer EQ 10 THEN
      TERMINATE Machs ;
      TERMINATE Rest  ;
      PREVENT Steuer ;
   FIN;

   Ausgabenummer = Ausgabenummer + 1 ;
END;

MODEND;
```

Das Beispielmodul PROBE4.P

```
P=68000;
SC=$1000,CODE=$0,VAR=$0;

MODULE PROBE2 ;

SYSTEM ;
   Disp : A1 <-> ;

PROBLEM ;

   SPC Disp  DATION INOUT ALPHIC ;
   SPC x     FLOAT GLOBAL        ;

Machs: TASK PRIO 50 ;
   REPEAT
      PUT x, SIN(x), 1.0/x TO Disp
          BY SKIP, (2)F(20), E(20,7), SKIP ;
      x = x + 1.0 ;
   END;
END;

MODEND;
```

Das Beispielmodul PROBE2_1.P

4.2 Cross-Dateien

```
S00600000000000F9
S30A0000E50EU006#SSIN 27
S3080000DF32U002PIF6
S3090000E214U004#SPI98
S3090000E20EU004#DPIAD
S30A0000EB76U006#DSIN C8
S30A0000E520U006#SCOS 1A
S30A0000EB88U006#DCOS BB
S30A0000E6B8U006#STAN 83
S30A0000ED26U006#DTAN 1D
S30A0000E600U006#SSQRTF4
S30A0000EC44U006#DSQRTB9
S30A0000E3AEU006#SEXP 86
S30A0000E952U006#DEXP EB
S30A0000E2FCU006#SATAN22
S30A0000E86AU006#DATANBD
S30A0000E23AU006#SASINDD
S30A0000E790U006#DASIN91
S30A0000E21CU006#SACOS00
S30A0000E76AU006#DACOSBC
S30A0000F2B6U006TORTOS07
S30A0000F2CCU006#SRTOS1E
S30A0000F378U006TORTOD53
S30A0000F392U006#DRTOS66
S30A0000F412U006ASSIGNCF
S30A0000F7FCU006WRITE FC
S3090000F7EEU004READ9C
S9080000000000000F7
```

Beispieldatei `PROBE.CRS`

4.3 MAP-Dateien

```
IEP-Linker RTL 1.613

used Commandfile  :  /H1/C/PROBE_2.RTL
used Listfile     :  /H1/C/PROBE_2.MAP

Date :  17.01.1993
Time :  13:40:45

ROM-Ranges
==========

00020000 -> 0002FFFF  MAIN
----------------------------
00020000 -> 00020095  /H1/C/PROBE1.SR             CODE
00020096 -> 00020153  /H1/C/PROBE2_1.SR           CODE
00020154 -> 000201AD  /H1/C/PROBE3.SR             CODE
000201AE -> 0002027B  /H1/C/PROBE4.SR             CODE
0002027C -> 000202B7  /H1/C/PROBE1.SR             DATA
000202B8 -> 000202FD  /H1/C/PROBE2_1.SR           DATA
000202FE -> 0002032F  /H1/C/PROBE3.SR             DATA
00020330 -> 00020371  /H1/C/PROBE4.SR             DATA
00020372 -> 0002FFFF                   FC8E Bytes free

00030000 -> 0003FFFF  MAIN
----------------------------
00030000 -> 0003FFFF                  10000 Bytes free
RAM-Ranges
==========

00010000 -> 0001FFFF  MAIN
----------------------------
00010000 -> 0001FF61                   FF62 Bytes free
0001FF62 -> 0001FF8D  /H1/C/PROBE4.SR             DATA
0001FF8E -> 0001FFA9  /H1/C/PROBE3.SR             DATA
0001FFAA -> 0001FFD9  /H1/C/PROBE2_1.SR           DATA
0001FFDA -> 0001FFFF  /H1/C/PROBE1.SR             DATA
```

MAP-Datei PROBE_2.MAP

```
IEP-Linker RTL 1.613

used Commandfile  :  PROBE_3.RTL
used Listfile     :  PROBE_3.MAP

Date :  17.01.1993
Time :  15:42:08

ROM-Ranges
==========

00020000 -> 0002FFFF  ERSTES_ROM
----------------------------
00020000 -> 000200BD  /H1/C/PROBE2_1.SR           CODE
000200BE -> 00020117  /H1/C/PROBE3.SR             CODE
00020118 -> 0002015D  /H1/C/PROBE2_1.SR           DATA
0002015E -> 0002018F  /H1/C/PROBE3.SR             DATA
00020190 -> 0002FFFF                   FE70 Bytes free

00030000 -> 0003FFFF  ZWEITES_ROM
----------------------------
00030000 -> 00030095  /H1/C/PROBE1.SR             CODE
00030096 -> 00030163  /H1/C/PROBE4.SR             CODE
00030164 -> 0003019F  /H1/C/PROBE1.SR             DATA
000301A0 -> 000301E1  /H1/C/PROBE4.SR             DATA
000301E2 -> 0003FFFF                   FE1E Bytes free

RAM-Ranges
==========

00010000 -> 0001FFFF  RAM_BEREICH
----------------------------
00010000 -> 0001FF61                   FF62 Bytes free
0001FF62 -> 0001FF8D  /H1/C/PROBE4.SR             DATA
0001FF8E -> 0001FFA9  /H1/C/PROBE3.SR             DATA
0001FFAA -> 0001FFD9  /H1/C/PROBE2_1.SR           DATA
0001FFDA -> 0001FFFF  /H1/C/PROBE1.SR             DATA
```

MAP-Datei PROBE_3.MAP

4.4 Befehlsüberblick

Es folgt in einem kurzen Überblick die Verwendung der Schlüsselworte des RT-Link und deren Kontext.

```
REMARK text
REM text

RAM $startram $endram
ROM $startrom $endrom

USE label [,label]
USE AS REFERENCE

FORMAT label
  S_RECORD | BINARY
  ABSOLUT | RELATIV
  FILL GAPS WITH $xx
  SPLIT LONGS TO WORDS
  SPLIT LONGS TO BYTES
  SPLIT WORDS TO BYTES
  NO SPLITTING
ENDFORMAT

AUTOSTART taskname [,taskname]

SCANRANGE_68K | SCANRANGE_PPC scanrange

WRITE label TO filename [BY label]

SYMBOLTABLE  ADDRESSES
SYMBOLTABLE  ALPHABETICAL
```