Remote-Debugging am 80x51

Franz Fiala, N, TGM DSK-407: MULTI.EXE

1. Erläuterungen

Debugger

Menschliche Produkte sind in der Regel fehlerbehaftet, wenn sie nur ein bißchen mehr als nur trivial sind. Programme sind komplex. Ein „Bug" ist ein Synonym für Fehler im Programm. Debugger sind in der Lage, einem fertigen Programm „auf die Finger zu schauen". Sie geben dem Programmierer einen Blick ins Innere des Programms frei. Im MS-DOS-Betriebssystem wird der einfache Debugger `DEBUG.EXE` mitgeliefert. Jeder Compiler bietet symbolische Debugger an. Diese Debugger laufen auf derselben Maschine wie auch das endgültige Produkt, teilen sich daher mit diesem den verfügbaren Platz.

Remote-Debugger

Ist nicht ausreichend Platz vorhanden, könnten Debugger und Programm nicht mehr gleichzeitig ablaufen. Compilerhersteller bieten daher sogenannte Remote-Versionen an, die mit einem nur sehr kleinen Debug-Modul auskommen und die Hauptarbeit einem zweiten PC übertragen. Die Kommunikation übernimmt eine Verbindung über die serielle Schnittstelle.

Debuggen in Mikrocontrollern

Im Mikrocontroller befindet sich im allgemeinen nur ein Anwenderprogramm im EPROM, das man debuggen will. Dieses Programm hat aber kein Betriebssystem wie im PC, das es erlauben würde, gleichzeitig zwei Programme (den Debugger und das auszuführende Programm) auszufühen.

Emulatoren

Ein aufwendiger Ausweg sind sogenannte Emulatoren, die anstelle der CPU in die Schaltung eingesetzt werden. Ein Emulator ersetzt die CPU in einer beliebigen Hardwareumgebung und erlaubt das Debuggen in der fertigen Schaltung. Er ist eine aufwendige Hardware, bestehend aus einem sogenannten Probe und einer Steckkarte im PC. Der Probe ersetzt die CPU in der Schaltung, die Steckkarte bildet das Interface zum PC und enthält darüberhinaus auch alle erforderlichen Speicher. Für jede CPU ist im allgemeinen ein eigener Probe erforderlich, natürlich auch für jede Bauform.

Natürlich, das Non-plus-ultra der Programmentwicklung für Mikrocontroller ist ein Emulator aber ebenso selbstverständlich ist auch, daß diese Geräte nicht für den Alltagsgebrauch bestimmt sind. Emulatoren sind teuer und sind daher auch in großen Firmen nicht auf jedem Arbeitsplatz verfügbar. Sie werden in jenen Fällen eingesetzt, in denen herkömmliche Meßtechnik, Simulation und das hier beschriebene Remote-Debugging nicht zum Ziel führen.

Für alle kleinen Probleme bei der Inbetriebnahme von Mikrocontrollerschaltungen genügen Remote-Debugger, Monitore, die in die fertige Baugruppe eingesetzt werden und die, mit einem entsprechenden Partnerprogramm im PC kommunizierend, alle Maßnahmen des Debuggens auch in der fertigen Platine erlauben.

Remote-Debugger im Mikrocontroller

Ein Remote-Debugger am Mikrocontroller ist ein kleines Monitor-Programm, das vor dem endgültigen Betrieb in die fertige Mikrocontrollerschaltung (gemeinsam mit dem zu testenden Programm) eingesetzt wird, um alle Hardwarefunktionen testen zu können. Ebenso wie bei reinen Software-Debuggern können Programme gestartet oder im Einzelschritt-Modus ausgeführt werden.

Nachteile des Remote-Debugging

- man benötigt eine serielle Schnittstelle zur Kommunikation
- wenn die Hardware nicht läuft, nützt dieser Debugger natürlich nichts
- man muß dafür sorgen, daß der Remote-Debugger seinen Platz in der fertigen Hardware findet.
- Es muß eine von Neuman-Adressierung vorliegen (siehe Kasten an anderer Stelle)

Vorteile des Remote-Debugging

- man testet in der endgültigen Hardware
- man benutzt dieselbe Softwareumgebung wie bei der Simulation
- er ist billig

Welche Remote-Debugger verwenden wir?

In unserem Laboralltag werden zwei Debugger eingesetzt

- FSD51 für den µProfi-51
- DS51/TS51 für beliebige Hardware-Umgebung

Der FSD51 ist ein in vielen Jahren getestetes und verbessertes Produkt von Kollegen Scharl, das für das Debuggen von Assemblerprogrammen gut geeignet ist.

Der DS51 (Bildschirmorientierter Debugger und Simulator) und der TS51 (Bildschirmorientierter Remote Debugger) von KEIL bestechen durch die Möglichkeit, auch im C-Source-Kode Programme untersuchen zu können.

Der FSD51 hat Schwierigkeiten, mit den Symboldateien zurechtzukommen, die C-Compiler produzieren und der TS51 war bisher nicht für den µProfi-51 verfügbar.

Die folgenden Seiten zeigen einerseits, wie man den MON51/KEIL in verschiedenen Hardware-Gegebenheiten einsetzt, der Leser kann also in Anlehnung an diese Vorgangsweise den MON51/KEIL an seine Hardware anpassen.

Anderseits wird ein universeller Monitor vorgestellt, der den gleichzeitigen Betrieb von MON51/TGM und MON51/KEIL am µProfi-51 erlaubt.

Mikrocontroller-Entwicklung am TGM

Am TGM werden den Schülern bei der Konstruktion von Mikrocontrollerschaltungen zwei Möglichkeiten angeboten:

- **Konstruktion von Erweiterungsplatinen zum µProfi-51**. Dabei wird die bewährte CPU-Platine von den Schülern mit Peripherie erweitert. Die in Assembler (ASM51/INTEL) geschriebenen Programme werden mit dem Remote-Debugger FSD51 getestet. Die Software besteht aus dem PC-Programm FSD51.EXE und dem ROM-residenten Teil MON51. (**Achtung**: Es besteht Namensgleichheit mit dem KEIL-Produkt, daher wird in der Folge immer von TGM-MON51 und von KEIL-MON51 gesprochen, un Verwechslungen zu vermeiden.)
  Problem: Programme, die mit den KEIL-Produkten gebildet werden, konnten bisher nicht symbolisch am µProfi-51 getestet werden.
- **Konstruktion selbständiger CPU-Platinen mit Peripherie**. Dabei werden je nach Klasse und Lehrer entweder Assembler (INTEL) oder der C-Compiler C51 von KEIL verwendet. Zum Testen der Schaltungen wurden bisher Emulatoren verwendet. Die Emulatoren sind aber nur in geringer Stückzahl vorhanden und können daher auch nicht im Informatik-Unterricht eingesetzt werden.
  Problem: Bisher gab es keine Möglichkeit, die Programme ohne Emulator zu testen.

µProfi-51

Der ist ein Einplatinenmikrokontroller mit Erweiterungssteckplatz, der an anderer Stelle in diesem Heft vorgestellt wird. Die Platine kann über den PCC-TGM als Bausatz gekauft werden. Im Bausatzpreis (siehe Preisliste auf der Impressumseite) sind alle Bauteile, die Bauanleitung, der FSD51 mit Beschreibung sowie das fertige Boot-Prom MON51 enthalten.

2. MMON51, V2.1 für den µProfi-51

Der Monitor FSD51 (und das zugehörige EPROM MON51) haben derzeit die Versionsnummer 2.1. Es ist ein Nachteil, daß Symboldateien (M51), die vom Linker L51 der KEIL-Programmfamilie generiert werden, nicht nachgeladen werden können.

Das hier beschriebene Monitor-Eprom MMON51 für den µProfi-51 (das erste „M" steht für Multi) erlaubt das Ausführen und Testen von Programmen, die mit dem KEIL-C-Compiler C51 geschrieben wurden. Gleichzeitig bleibt die volle Funktionalität des FSD51 erhalten.

Eigenschaften

Das Bootprom MMON51 für den µProfi-51 ermöglicht folgende zusätzlichen Betriebsarten:

- Arbeiten mit dem TS51/MON51 von KEIL
- Veränderung des seriellen Interruptvektors
- Alternatives Booten vom Monitor oder vom Anwenderprogramm
- Löschen des internen Datenspeichers beim Booten

Bei der Programmgestaltung wurde Wert darauf gelegt, daß sich der MMON51 ohne Zusatzmaßnahmen genauso verhält wie das bisherige BOOT-PROM MON51.

Voraussetzungen

Die zusätzlichen Eigenschaften des neuen EPROMs werden durch den Pin P1.0 gesteuert, der für die Steuerung des Watchdog-Bausteins IC9 (MAX 690) vorgesehen ist. Bei Verwendung des Watchdogs durch das Anwenderprogramm könnte ebenso eine der Leitungen P1.4, P1.5, P1.6 oder P1.7 verwendet werden. Wenn man im µProfi-51 den Watch-Dog-Baustein IC9(MAX690) entfernt, ist der Boot-Vorgang wie bisher, mit folgenden Softwareänderungen:

- Der Interruptvektor für die serielle Schnittstelle ist nicht fest wie beim MON51, sondern wird auf A023 umgelenkt. Das ist einerseits für den 2-Monitor-Betrieb erforderlich, anderseits ermöglicht es das Testen von Programmen, die die serielle Schnittstelle benötigen.
- Durch diese Umlenkung ist es notwendig, daß Anwenderprogramme bei der Adresse A100 beginnen, da der Adreßraum A000..A0FF für die verschobenen Interruptvektoren und einige Hilfsvariablen benötigt wird.

Die weitere Arbeit mit dem FSD51 ist so wie bisher.

P1.0

Die Portleitung P1.0 hat eine Mehrfachfunktion übertragen bekommen. Wird P1.0 nicht beschaltet, verhält sich MMON2.1 wie MON2.1.

Kaltstart

Beim Kaltstart übernimmt P1.0 die Auswahl zwischen den beiden Monitoren (FSD51/TGM und TS51/KEIL). Da eine offene Eingangsleitung als 1 erkannt wird, wurde dieser Zustand dem TGM-Monitor zugewiesen.

Der gewünschte Monitor wird auf die Adresse A0FE „vermerkt" (F5=KEIL, alle anderen Zustände=TGM). Weiters wird auf A0FB mit BB „vermerkt", daß jeder weitere Reset ein Warmstart ist. Schaltet man den µProfi-51 nicht von der Spannung ab, bleibt der gewählte Monitor in der Adresse A0FE gespeichert, der Portpin P1.0 kann daher eine weitere Aufgabe übernehmen.

Warmstart

P1.0 entscheidet bei einem Warmstart, ob ein Reset den Monitor (P1.0=1) oder das Benutzerprogramm (P1.0=0) startet.

Damit ist es möglich, Programme zu debuggen, die den seriellen Interruptvektor verändern. Ein Reset kann daher den Monitor umgehen und direkt zum Anwender-EPROM gelangen, wenn P1.0=0. Stürzt dieses ab, ist man mit P1.0=1 und RESET wieder im Monitor (vorausgesetzt, man hat im Adreßbereich A0Fx nichts verändert).

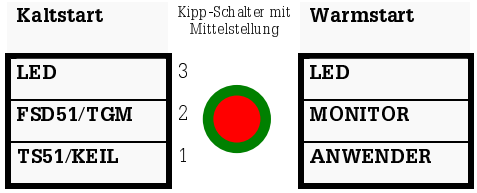
Wenn P1.0=1 (offener Eingang), startet man wie gewohnt mit jedem Reset den Monitor. Wenn P1.0=0, startet man mit jedem Reset das Anwenderprogramm neu. Selbstverständlich ist das Anwenderprogramm dafür verantwortlich, daß der Interrupt-Vektor des seriellen Interrupts an die richtige Adresse 0023 kommt. Es ist möglich, die Interruptadresse wie in einem endgültigen EPROM auf die Adresse 0023 (statt auf die Adresse A023) zu laden. Das Programm HEXPAT kann die Aufgabe übernehmen, die Interruptadressen auf A0xx zu verschieben. Das besorgt die Zeile `HEXPAT name A0 00` mit der Datei `<name>.HEX`. Sie erzeugt eine Datei `name.HEY`, die statt `name.HEX` geladen wird.

P1.0 als Treiber für das rote LED

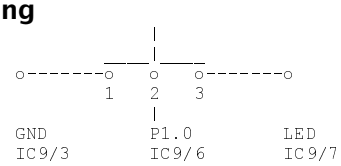
Da der Watchdog bei Auswahl der Bootmöglichkeiten durch P1.0 nicht in seiner Fassung ist, kann Pin P1.0 zusätzlich zum Testen kleiner Programme verwendet werden, indem man ihn mit IC8, dem Treiber für das rote LED verwendet.

Umschalter für P1.0

Ein Umschalter mit Mittelstellung ist für die Bedienung von P1 geeignet. Die folgende Skizze kann für eine kleine Blende für den Schalter verwendet werden.



Beschaltung



Beim Kaltstart erkennt P1.0, welchen Monitor man benutzen will (TGM oder KEIL). Beim Warmstart entscheidet er, ob vom Monitor oder vom Benutzerprogramm gestartet wird. Wenn das Benutzerprogramm das rote LED benutzt, dann schaltet man auf Stellung 3.

Bedienung des Monitors MMON2.1

An Hand des folgenden Programms `MINI.C` soll die Benutzung des Monitors MMON51 gezeigt werden. Das Programm läßt das rote Led über die Portleitung P1 blinken.

```
#include <reg51.h>

void main(void)
{
  for (;;)
  {
      unsigned int i;
      for (i=0; i<65000; i++);
      P1 ^= 1;
  }
}
```

Die Interrupt-Routine (hier nicht abgebildet) wird nicht benutzt, es wird lediglich geprüft, ob der Interruptvektor richtig auf die Adresse A00B gelegt wird.

Kompilierung der Datei mini.c

```
C51 mini.c
l51 mini.obj to mini.abs
ohs51 mini.abs
hexpat mini A0 00
```

Programmtest mit dem FSD51/TGM

Wo	Was	Bemerkung
HW	Sicherung lösen (> 2 s)	Speicher löschen
HW	P0.1 = offen	TGM-MON51 anwählen
HW	Sicherung einsetzen	**Kaltstart** erzwingen
DOS	FSD51	Monitor starten
FSD51	F2 File Download	MINI.HEX
FSD51	Pos1	Startadresse PC A000 einstellen
FSD51	Ende	zurück in den Code-Bereich
HW	P01 mit LED verbinden	
FSD51	F6	Programm starten, LED blink
HW	P0.1 = offen	LED getrennt
HW	RESET	**Warmstart Monitor** (da P0.1) offen, FSD-51 synchronisiert wieder
HW	P0.1 = 0 (GND)	User-Boot einstellen
HW	RESET	**Warmstart User** A000 „Empfänger nicht bereit“ (kuzzeitiger Kurzschluß)
HW	P0.1 mit LED verbinden	Lämpchen blinkt
HW	P0.1 = offen	LED getrennt
HW	RESET	**Warmstart Monitor** (da P0.1) offen FSD-51 synchronisiert wieder

Programmtest mit dem TS51/KEIL

Wo	Was	Bemerkung
HW	Sicherung lösen	Speicher löschen
HW	P0.1 = 0 (GND)	KEIL-MON51 anwählen
HW	Sicherung einsetzen	**Kaltstart** erzwingen
DOS	TS51 MINI.ABS INIT (MINI.INI)	Monitor starten, Programm und Initialisierungsdatei laden, letzte Zeile: >$=0a000
HW	P01 mit LED verbinden	
TS51	g 0a000	Programm starten, LED blink
HW	P0.1 = offen	LED getrennt
HW	RESET	**Warmstart Monitor** (da P0.1 offen), Meldezeichen: >
HW	P0.1 = 0 (GND)	User-Boot einstellen
HW	RESET	**Warmstart User** Programm ab Adresse A000 starten
HW	P0.1 mit LED verbinden	Lämpchen blinkt (kurzzeitiger Kurzschluß)
HW	P0.1 = offen	LED getrennt
HW	RESET	**Warmstart Monitor** (da P0.1) offen, TS51 synchronisiert wieder

Programmtest mit dem TS51/KEIL

Wo	Was	Bemerkung
HW	Sicherung lösen	Speicher löschen
HW	P0.1 = 0 (GND)	KEIL-MON51 anwählen
HW	Sicherung einsetzen	**Kaltstart** erzwingen
DOS	MON51	Monitor starten, Meldezeichen #
MON51	load mini.HEY	gepatchtes Programm laden
HW	P01 mit LED verbinden	
MON51	g 0a000	Programm starten, LED blink
HW	P0.1 = offen	LED getrennt
HW	RESET	**Warmstart Monitor** (da P0.1 offen),Meldezeichen: # „PROCESSING TERMINATED AT 0000“
HW	P0.1 = 0 (GND)	User-Boot einstellen
HW	RESET	**Warmstart User,** Programm ab Adresse A000 starten
HW	P0.1 mit LED verbinden	Lämpchen blinkt (kurzzeitiger Kurzschluß)
HW	P0.1 = offen	LED getrennt
HW	RESET	**Warmstart Monitor** (da P0.1) offen, MON51 synchronisiert wieder

Startadresse von C-Programmen

Die Ladeadresse eines C-Programms wird beim Linken festgelegt und ist in der Grundeinstellung der BAT-Dateien `A100`. Die Startadresse ist `0H`. Der Linker generiert selbständig auf die Adresse `0` einen `LJMP` auf die Startadresse des C-Programms, die im allgemeinen größer ist als A100. Die beiden Monitore MON51/TGM und MON51/KEIL verhalten sich etwas verschieden, was das Laden nicht vorhandener XCODE-Adressen anlangt (die Adresse 0 ist in ein nicht vorhandenes RAM gerichtet). Der TGM-Monitor ignoriert sie, der KEIL-Monitor gibt eine Fehlermeldung aus, daß er die Daten nicht laden konnte.

Da auf 0H das Monitor-Rom liegt, wird mit einem HEX-Patch, die Ladeadresse auf `0A000H` verschoben

```
hexpat mini A0 00
```

und auch auf diese Adresse verzweigt.

Der Monitor startet jedenfalls das Anwenderprogramm auf der Adresse `A000`. Er prüft aber zuerst, ob sich auf dieser Adresse ein `LJMP` befindet. Ist das nicht der Fall, schreibt er selbständig einen `LJMP A100` auf die Adresse `A000`. Das wurde gemacht, um einfachen Assemblerprogrammen, die für die Adresse `A100` gelinkt werden, einen problemlosen Start via User-Reset zu ermöglichen.

Es folgt jetzt eine Darstellung der Entwicklung dieses Monitors, die auch zeigt, wie man den KEIL-Remote-Debugger für verschiedene Adreßbereiche konfigurieren kann. Die Beschreibung soll für Besitzer des KEIL-Compilers nachvollziehbar sein, die Generierung der EPROMs sollte mit dieser Beschreibung unter Benutzung der zusätzlichen Programmteile möglich sein (Theorie). Bei Schwierigkeiten können Sie bis Ende Juni den Autor kontaktieren, wie wär's via EMAIL?

3. MON51/KEIL für viele Fälle

TS51 für den µProfi-51

Das „Professional Developer" Paket von KEIL für die 8051-Familie enthält unter anderem auch einen Remote-Debugger, der für andere Systeme gut anpaßbar ist. Ein eigenes Installationsprogramm generiert eine HEX-Datei, die man unmittelbar in ein EPROM laden kann.

Das Programm MON51/KEIL besteht aus zwei Teilen (wie auch der FSD51). Einem Teil im Hardware-Aufbau und einem Teil im PC.

Der Programmteil im PC kann entweder ein einfacher, zeilenorientierter Terminal-Debugger sein (heißt ebenfalls MON51) oder ein Remote-fullscreen-Debugger TS51, der durch Laden des Kommunikationsmoduls `MON51.IOT` mit dem ROM-residenten Programm in der Baugruppe verbunden wird.

Der Programmteil in der Hardware ist etwa 4k groß und kann unter verschiedensten Betriebsbedingungen generiert werden.

Bei der Installation des Remote-Debuggers auf ein Fremdsystem müssen folgende Parameter für das Programm `INSTALL.BAT` (in dieser Reihenfolge) eingegeben werden:

- Serielle Schnittstelle, mit der der Remote-Debugger betrieben werden soll
- Page-Adresse eines 256-Byte großen Speicherbereichs im externen RAM (XDATA)
- Page-Adresse, an der der externe Monitor geladen werden soll
- EPROMCHECK gibt an, ob eine Detektion eines EPROMs ab Adresse 0 gewünscht wird

Die Batchdatei `INSTALL.BAT` generiert eine HEX-Datei `MON51.HEX`.

Beispiel 1: KEIL liefert auch eine Baugruppe **MCB-517**, für die dieser Remote-Debugger ursprünglich gedacht war. Für diese Baugruppe generiert man den Debugger-Code so:

```
INSTALL 1 7f 80 promcheck
                ^ testet, ob auf Adresse 0 ein ROM ist und
                  startet das dort befindliche Programm
              ^----Monitor wird ab Adresse 8000h generiert
            ^-------256 Bytes ab XRAM-Adresse 7f00h reserviert
          ^---------Ser 0, Baudratengenerator (nur 80517/515)
```

Beispiel 2: Die Generierung erfolgt für den **µProfi-51** mit:

```
INSTALL 0 7F 0
                ^ testet Adresse 0 nicht
              ^----Monitor wird ab Adresse 0000h generiert
            ^-------256 Bytes ab XRAM-Adresse 7f00h reserviert
          ^---------Ser 0, Timer 1
```

wobei die erste 0, die erste serielle Schnittstelle mit 11 Mhz Taktfreqünz bedeutet, 7f die Pageadresse des benutzten RAM und die zweite 0 die Ladeadresse für den Monitor.

Im Prinzip kann man mit diesem Monitor bereits arbeiten, die Verbindung der Programmteile gelingt sofort. Leider hat die Sache noch einen Haken, wie das Laden von Programmen zeigt:

Ladeadresse

Anders als bei Systemen, die ab Adresse 0000 starten, muß der Kode für den µProfi-51 ab Adresse A100h beginnen. Dazu muß beim Linken eine neü Kode-Adresse angegeben werden. Weiters muß für eine geeignete Aufteilung zwischen Code-Bereich und XDATA-Bereich gesorgt werden.

Die Kommandozeile

```
L51 %1.OBJ,%2.OBJ TO %1.ABS CODE (A100) XDATA (A800) DATA (20)
SYMBOLS DEBUGSYMBOLS
```

verbindet zwei OBJ-Dateien, linkt auf die richtigen Adressen und bereitet mit den zusätzlichen Parametern `SYMBOLS DEBUGSYMBOLS` auf das symbolische Debuggen vor. Die Dateien `%1.OBJ` und `%2.OBJ` können sowohl vom Assembler A51 als auch vom Compiler C51 stammen.

Programme, die solcherart für den Ablauf ab Adresse A100 gelinkt werden, werden auch einwandfrei auf diese Adresse geladen. Im zugehörigen HEX-File sind aber Schönheitsfehler:

Reset-Vektor

Der Linker generiert jedenfalls auf die Adresse 0000 einen Sprung zur Startup-Routine, die bei A0xx beginnt.

Dieses Problem kann man beheben, indem man den ebenfalls mitgelieferten Startup-Kode geringfügig ändert und den Sprungbefehl von der absoluten Ladeadresse 0000 auf A000 verschiebt. [Wenn man die störende Zeile im HEX-File nur löscht, verliert man die Adresse des RESET_vektors.]

Interruptvektoren

Außerdem generiert der Linker (nicht abschaltbar) für jede Interruptservice-Routine einen Sprungbefehl auf die korrespondierenden Adressen im tiefen Speicherbereich folgend auf den Reset-Vektor. Im µProfi sollten alle um den Offset A000 verschoben sein.

Startet man den TS51 mit der Ladedatei des Linkers, werden die auf ROM-Adressen liegenden Interruptvektoren ohne Fehlermeldung geladen, werden aber nicht wirksam, da das ROM ja bezüglich seines Inhalts nicht mit sich reden läßt.

Lädt man dagegen die HEX-Datei (die das Programm OHS51 generiert), beschwert sich der TS51 wiederholt, daß es ihm nicht möglich ist, den Code richtig zu plazieren. Was zunächst sehr störend ist, bietet aber einen Schlüssel zur Problemlösung, denn während die binäre Ladedatei nicht beeinflußbar ist, kann man das HEX-File verändern.

Es gibt zwei Lösungen zu diesem Problem:

1. Die neuere Version des Compilers benutzen! Ab Version 3.4 des Compilers kann man mit einer Pseudoanweisung erreichen, daß die Interruptvektoren um einen beliebigen Versatz verschoben werden.
2. Das entstehende HEX-File patchen! Das erledigt das Programm `HEXPAT` mit den Argumenten `A0 00`. Alle Ladeadressen im HEX-File, die mit `00` beginnen, werden mit `A0` überschrieben und die Prüfsumme korrekt nachgezogen. (Dieser Vorgang ist im Detail an anderer Stelle beschrieben.)

Erfreulicherweise wird mit dem HEX-File-Patch auch das erste Problem des RESET-Vektors gelöst (der Reset-Vektor verhält sich genauso wie ein Interruptvektor, nur wird er in jedem Fall generiert), sodaß man von der Veränderung des Startup-Code absehen kann.

Ein Programm, das mit dem TS51 im µProfi-51 getestet werden soll, erfordert daher eine etwas erweiterte Abarbeitung. Die Schritte sind für eine Datei `NAME`.

```
C51 NAME.C
L51 NAME.C TO NAME.ABS
OHS51 NAME.ABS
HEXPAT NAME A0 00
TS51 NAME.ABS INIT (NAME.INI)
```

Dieser automatisierbare Vorgang setzt voraus, daß neben der Quelldatei `NAME.C` auch eine Datei `NAME.INI` angelegt wurde, die im wesentlichen darin besteht, den Treiber `MON51.IOT` und danach die gepatchte Datei `NAME.HEY` nachgeladen wird:

```
LOAD MON51.IOT CPUTYPE (8051)
LOAD NAME.HEY
```

Außerdem werden in diese INI-Datei alle Einstellungen für den Remote-Debugger TS51 vorgenommen, die für das zu testende Programm wesentlich sind.

Die vorige Lösung, den µProfi-51 mit dem KEIL-Monitor zu betreiben, erscheint schon sehr vorteilhaft. Sie hat aber den Nachteil, daß man bei häufiger Verwendung beider Monitore das EPROM-rein-raus-Spiel eher hinderlich, denn erfreulich empfindet.

Es bieten sich zwei Lösungen an:

Nachladen des MON51 mit dem FSD51

Dazu wird eine MON51-Version für den Adreßbereich B000h generiert. Die zusätzlich benötigte XRAM-Page wird an das Ende des freien RAM-Bereiches geladen. Man generiert den MON51 so:

```
INSTALL 0 AF B0
              ^ testet Adresse 0 nicht
            ^----Monitor wird ab Adresse B000h generiert
          ^-------256 Bytes ab XRAM-Adresse Af00h reserviert
        ^---------Ser 0, Timer 1
```

Zuerst startet man den FSD51 und lädt danach mit F2 , File Download die Datei MON51.HEX nach und startet mit F5 . Das Programm läuft und man kann aus dem FSD51 mit F10 aussteigen. Danach kann man je nach Bedarf den Terminal-Remote-Debugger MON51 oder den Full-Screen-Remote-Debugger TS51 aufrufen.

Der Nachteil ist klar: bei jedem Absturz muß man diese Prozedur wiederholen. Das Verfahren wird also nur dann vorteilhaft sein, wenn man hauptsächlich mit dem FSD51 und den MON51 nur nebenbei betreibt.

FSD51 Master, MON51 Slave

Da der Kode des FSD51 feststeht, der des MON51 aber anpaßbar ist, kann man ein EPROM generieren, das beide Monitore enthält, Platz ist ja genug.

Man generiert den MON51 wie folgt:

```
INSTALL 0 BF 10
              ^ testet Adresse 0 nicht
            ^----Monitor wird ab Adresse 1000h generiert
          ^-------256 Bytes ab XRAM-Adresse Bf00h reserviert
        ^---------Ser 0, Timer 1
HEX2BIN MON51
```

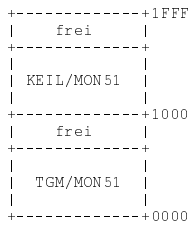
Jetzt muß man das FSD51-EPROM und die durch INSTALL entstandene Datei kombinieren. Ich verwendete dazu den EPROM-Programmer.

1. FSD51-EPROM auslesen und zur Sicherheit speichern- (FSD51.BIN)
2. Datei MON51.BIN ab Adresse 1000H dazuladen
3. Neues EPROM generieren

Gestartet wird immer mit dem FSD51, bei Bedarf kann man aus dessn Oberfläche den MON51 aufrufen:

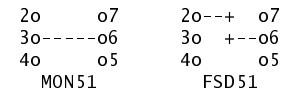
```
Pos1  PC auf 1000 einstellen
End   Zurück in den CODE-Bereich
F5    startet das Programm, FSD51 meldet,
      daß das Programm ausgeführt wird
F10   entweder temporär oder ständig ins System aussteigen
MON51 oder TS51 aufrufen
Programm debuggen
F1/exit aussteigen
exit  wieder im FSD51, wenn man temporär ausgestiegen ist.
```

Diese Prozedur hat zwar noch immer den Nachteil, daß man bei häufigen Abstürzen immer wieder via FSD51 hochfahren muß, man erspart aber immerhin das Nachladen der Datei MON51BO.HEX.



FSD51 - MON51 Dualboot

Opfert man dagegen eine der IO-Leitungen von Port 1 (P1.0, die für den Watch-Dog vorgesehen ist oder P1.6 oder P1.7), kann man an einer dieser Leitungen einen Jumper anbringen und mit einer Verzweigung den einen oder den anderen der beiden Monitore aktivieren. Gewählt wurde die Leitung für den Watchdog (P1.0), da man sie einfach am Sockel des Watchdogs mit Drahtbrücken jumpern kann. P1.0 endet an Pin 6 des Sockels von IC9.



Verbindet man Pin 6 mit Pin 3 (Ground), soll MON51 aktiv sein, verbindet man Pin 6 mit Pin 2 (VCC), soll sich FSD51 aktivieren.

Bei dieser Konfiguration muß man an den Resetvektor ein kleines Programm anbinden, das die Umschaltung vornimmt.

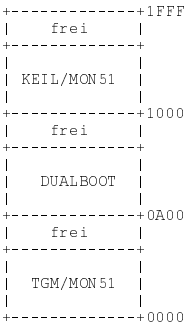
Da danach der Monitor aktiv ist, kann man die kleine Brücke entfernen (für den Fall, daß man P1.0 für den Watch-Dog oder für andere Zwecke verwendet).

Das Umschalteprogramm wird durch den RESET-Vektor aufgerufen. Ist das Bit 90H(P1.0) gesetzt, verzweigt das Programm an die Adresse 00A5H (Einsprungpunkt des FSD51) sonst verzweigt es auf die Adresse 1000H (Einsprungpunkt des MON51). Diese Vorgangsweise setzt voraus, daß der Einsprungpunkt des FSD51 mit 00A5H immer konstant ist. Das könnte sich durchaus bei späteren Versionen ändern. Sollte das der Fall sein, müßte man auch das Boot-Programm entsprechend ändern.

```
            dualboot:
20 90 03              JB 90H,loadFSD
            loadMON:
02 10 00              LJMP 1000H
            loadFSD:
02 00 A5              LJMP 00A5H
```

Dieses Programm ist sehr kurz und durch den Aufbau auch relokatibel; wir verzichten auf den Assembler und bestimmen im Monitor DS51 durch direktes Assemblieren auf irgendeine Adresse, welcher Kode zu laden ist. Das Ergebnis wurde in den obigen Kode eingetragen.

Jetzt muß man nur mehr klären, wie man die einzelnen Bestandteile zusammenfügen kann. Da der MON51 bedeutend größer ist als der FSD51, ist im Adreßbereich 0000..0FFF viel mehr Platz als in 1000..1FFF. Die letzte belegte Seite durch den FSD51 ist 0800. Danach ab 0A00 ist Platz.



Arbeitsschritte am EPROMmer:

1. FSD51-EPROM auslesen und zur Sicherheit speichern FSD51.BIN
2. Datei MON51.BIN ab Adresse 1000H dazuladen
3. Die 9 Bytes des Dualboot-Programms auf Adresse 0A00H eintragen
4. Resetvektor von 02 00 A5 auf 02 0A 00 ändern
5. Neues EPROM generieren

Man könnte theoretisch das Dual-Bootprogramm auch im Startup-Kode INSTALL.A51 unterbringen. In der obigen Version erspart man aber jede Änderung.

4. MMON51 Multiboot, Entwicklung

Erste Versuche zeigten, daß der Dualboot zwar funktioniert aber es wurde verabsäumt, die verschiedenartigen Interrupts für die serielle Schnittstelle zu berücksichtigen. Beim FSD51 ist der Interruptvektor für die serielle Schnittstelle fest auf eine EPROM-Adresse ausgerichtet und daher nicht änderbar. Daher lief der MON51 korrekt mit der ihm eigenen Interruptroutine, dagegen der MON51 mit einer falschen. Wie das trotzdem gehen konnte, ist unklar.

Folgende Eigenschaften erschienen wünschenswert:

Man sollte aus einem laufenden Monitor einen Neustart einleiten können, dessen Verhalten beeinflußbar ist. Dazu schreibt man in eine definierte Speicherstelle einen variablen Wert. In Abhängigkeit davon wird nach Auslösen des Hardware-Reset entweder der FSD51, der MON51 oder das Anwenderprogramm angewählt.

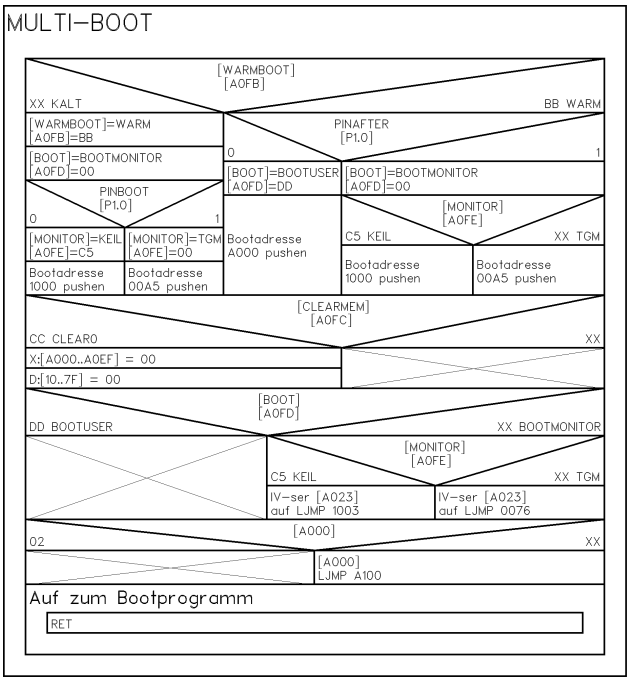
Ebenso sollte es möglich sein, daß der Hardware-Reset, so, wie im Dual-Boot-Verfahren, in Abhängigkeit des Umschalters an P1.0 wirkt.

Der RESET-Vektor hat die Aufgabe, das System am Beginn zu initialisieren. Ein durch das Programm ausgelöster RESET wirkt im Prinzip genau so, nur sind die Speicherinhalte im allgemeinen nicht auf einem zufälligen Anfangszustand, sondern haben möglicherweise noch „Restinhalte“ vom vorigen Programmlauf. Nicht ganz einwandfreie Programme können sich daher bei einem Hard-Reset (Spannung aus) anders verhalten als bei einem Soft-Reset (RESET-Taste drücken)

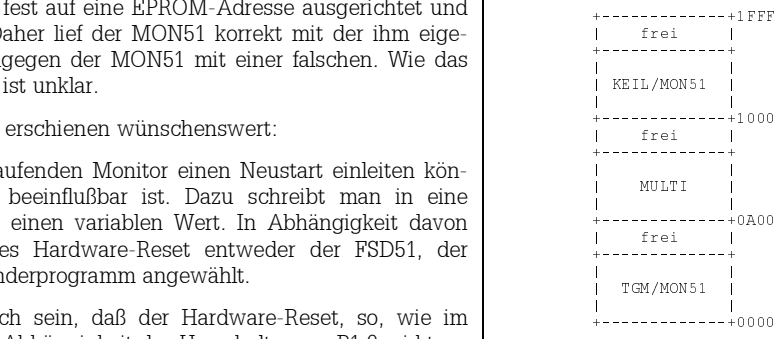
Daher wurde die Möglichkeit vorgesehen, daß nach einem Hardware-RESET die internen Speicherinhalte gelöscht werden, um Fehler dieser Art untersuchen zu können.

Eine eigene Speicherzelle verrät dem Programm, ob ein Hardware-Reset vorliegt, oder ob die Spannung eingeschaltet wurde.

Das folgende Struktogramm zeigt, wie der Ablauf des Multiboot vor sich geht. Schreibt man selbst nichts in die Speicherstellen `A0FF`, `A0FE` und `A0FD`, dann verhält sich das Programm wie das Dual-Boot-PROM. Durch entsprechendes Beschreiben dieser Adressen, kann man ein abweichendes Boot-Verhalten einstellen.



Wie generiert man ein BOOT-EPROM, das aus verschiedenen Komponenten zusammengesetzt ist? Die zusammenzufügenden Teile sind der nachfolgenden Skizze zu entnehmen.



Modifikationen am FSD51

Der residente Teil des FSD51 liegt als EPROM (Binär-Datei) vor. (`TGM51V21.BIN`). Die Datei ist kürzer als das EPROM, der Teil über der Adresse `9FFh` wurde abgeschnitten.

In diesem Binärfile sind folgende Änderungen auszuführen:

- Umlenkung des RESET-Vektors auf die Adresse `0A00H`
- Umlenkung des SERIELL-Vektors auf die Adresse `0A023H`

Damit der gesamte Vorgang reproduzierbar ist und nicht immer händisch ausgeführt werden muß, wird die Prozedur im `DEBUG` in eine eigene Datei geschrieben und mit der DOS-Umlenkung ausgeführt. Die BAT-Datei `TGMPAT.BAT` enthält diesen Vorgang. Außerdem ergibt sich nach dem Patch eine Binärdatei, die mit `BIN2HEX` in eine HEX-Datei für die Adresse 0 umgewandelt wird.

```
TGMPAT.BAT
debug tgm51v21.bin < tgmpat.deb
bin2hex tgm51v21 0000
```

Die Datei `TGMPAT.DEB` enthält alle Kommandos, die man in `DEBUG` einzugeben hat, um den Patch auszuführen:

```
TGMPAT.DEB
e0123               Patchen des SERIELL-Vektors
 A0 23
e0100               Patchen des RESET-Vektors
 0A 00
w                   Zurückschreiben in die Datei
q                   Verlassen des Debuggers
```

Generierung des MON51

Der residente Teil des MON51 wird durch die bereits mehrfach erwähnte BAT-Datei INSTALL generiert. Für dieses Problem wird aber der Kode von `INSTALL.A51` geringfügig verändert. Die Interruptvektoren müssen in dieser Konfiguration nicht generiert werden, da sie durch den feststehenden FSD51 schon vorgegeben sind.

Damit der Einsprungpunkt des seriellen Interrupt konstant ist, wird der Einsprungpunkt für den MON51 auf die Beginnadresse der Datei gelegt und der Einsprungpunkt für den seriellen Interrupt unmittelbar danach (also auf die relative Adresse 3).

```
INITSEG  SEGMENT CODE          beim Installieren auf A0
         RSEG    INITSEG       festgelegt
         LJMP    InitSerial    auf Adresse A000
         LJMP    SER_ISR       auf Adresse A003
```

Die Generierung erfolgt mit der BAT-Datei `MAKKEIL.BAT`:

```
MAKKEIL.BAT
INSTALL 0 BF 10
```

Es entsteht eine HEX-Datei `MON51.HEX`.

Jetzt fehlt nur noch der größte Brocken, das Multi-Boot-Programm selbst. Durch exakte Nachbildung des Struktogramms konnte das Programm in kürzester Zeit ohne wesentliche Fehler erzeugt werden.

Aus Platzgründen wird auf den Abdruck des Kode verzichtet, er befindet sich auf der Begleitdiskette zu diesem Heft.

Das Assemblieren dieses Codes geht mit MAKMULTI.BAT:

```
MAKMULTI.BAT
REM GENERIERUNG DES BOOT-PROGRAMMS
REM ============================
A51 MULTI.A51
L51 MULTI.OBJ TO MULTI.ABS
OHS51 MULTI.ABS
```

Wir haben jetzt drei HEX-Dateien: FSD51.HEX, MON51.HEX und MULTI.HEX, die zu kombinieren sind, und die ein gemeinsames EPROM ergeben sollen. Die Verbindung der drei Dateien kann etwa mit COPY erfolgen.

Was stört ist, daß jede Datei eine Endezeile (:00000001FF) enthält und beim Verbinden mit COPY diese Zeile natürlich bestehen bleibt und den späteren Programmiervorgang vorzeitig abbrechen würde.

Man kann durchaus mit Mitteln des Betriebssystems eine Lösung herbeiführen: Man vereinigt die Dateien mit COPY und sortiert die Zeilen mit SORT nach den Werten im Adreßfeld (Zeichenpositionen 4, 5, 6 und 7 wenn die Zählung bei 1 beginnt). Man verwendet bei SORT die Option /+n, die den Sortiervorgang ab der n-ten Spalte vornimmt. Dieser Vorgang steht in der BAT-Datei KOMBI.BAT:

```
KOMBI.BAT
COPY MON51.HEX+FSD51V21.HEX+MULTI.HEX UPR51.UNS
SORT /+4 < UPR51.UNS > UPR51.HEX
```

Eine Kleinigkeit, die man noch händisch ausführen muß: Die drei Schlußzeilen kommen jetzt aufgrund des Sortiervorgangs hintereinander zu liegen. Da ihr Adreßfeld 0000 enthält, stehen sie in der kombinierten HEX-Datei am Anfang und nicht am Ende. Man verschiebt also mit einem Texteditor die Zeilen ans Ende der Datei und löscht zwei davon.

Jetzt muß man mit dem Kommando

```
HEX2BIN UPR51
```

eine Binärdatei zum Erstellen eines EPROM generieren.

Fertig!

Selbstverständlich kann man jetzt die einzelnen Schritte in einer einzigen BAT-Datei UPR51.BAT zusammenfassen.

```
UPR51.BAT
CALL TGMPAT
CALL MAKKEIL
CALL MAKMULTI
CALL KOMBI
```

Testen des MULTI-BOOT-Eproms

Trotz der Kürze des Multiboot-Programms sind doch einige Möglichkeiten auf ihre richtige Funktion zu prüfen. Beschränkt man sich auf das Brennen eines EPROM, steht man bei Fehlern oft ziemlich ratlos da, sodaß man diese Versuche alsbald unterläßt.

Der Simulator/Debugger DS51 ist dabei eine vorzügliche Hilfe. Aber auch mit dem Debugger hat man bald genug, denn es sind gar nicht so wenige Eingaben zu machen, bis eine Einstellung korrekt vorgenommen ist. Man behilft sich daher mit einem eigenen INI-File, mit dem der Testlauf gut dokumentiert und gut reproduzierbar wird.

MULTI.INI

Einstellungen des DS51
Die Schalterstellung an P1.0 muß simuliert werden, es wurde nur der Multiboot-Programmteil geladen, daher fehlt der Reset-Vektor des späteren EPROM. Dieser wird mit der asm-Zeile nachgetragen. Die drei Breakpoints bei 1000, A5 und A000 entsprechen den drei Sprungzielen des Boot-Vorgangs. Das gesamte Programm wird jeweils durch g gestartet und nach Auflaufen auf einen der drei Breakpoints wird geprüft, ob die Sprungziele eingehalten werden und ob die Speicherinhalte korrekt sind. Die KEYWAIT-Aufrufe unterbrechen den Ablauf, damit man die Zwischenergebnisse verfolgen kann. In der Anmerkung nach KEYWAIT steht jeweils was getestet wird und was an welchen Speicheradressen stehen soll.

```
/* MULTI.INI */
\r                          /* Register off          */
\u\u\u                      /* Increase Window       */
\o\m                        /* Options Medium (Lines) */
\v\s                        /* View Serial (Off)     */
load ..\..\ds51\8051.iof    /* IOF driver for 8051   */
map 0,0xffff                /* XDATA memory 64KByte  */
P1.0=0 /* KEIL-MONITOR */
asm 0
ljmp 0a00H
.
asm 0x1000
nop
.
bs 0x1000
bs 0x00A5
bs 0xA000
KEYWAIT("")
/* Kaltstart ************************************/
/* $=1000H [A023]=LJMP 1003 [A0FB]=BB, [A0FE]=C5 */
KEYWAIT("")
g 0
t
d x:0xA000,0xA00F
d x:0xA020,0xA02F
d x:0xa0f0,0xa0ff
KEYWAIT("")
/* Warmstart vom Monitor ************************/
/* $=1000H [A023]=LJMP 1003 [A0FB]=BB [A0FE]=C5 */
KEYWAIT("")
P1.0=1
g 0
t
t
d x:0xA000,0xA00F
d x:0xA020,0xA02F
d x:0xa0f0,0xa0ff
KEYWAIT("")
/* Warmstart vom Userprogramm *******************************************/
/* $=A000, [A000]=LJMP A100 [A023]=LJMP 1003 [A0FB]=BB [A0FD]=DD [A0FE]=C5") */
KEYWAIT("")
P1.0=0
g 0
d x:0xA000,0xA00F
d x:0xA020,0xA02F
d x:0xa0f0,0xa0ff
KEYWAIT("")
/* Warmstart vom Userprogramm */
/* Speicher initialisieren **********************************************/
/* $=A000, [A000]=LJMP A100 [A023]=00 [A0FB]=BB [A0FD]=DD [A0FE]=C5")    */
KEYWAIT("")
EC d:0x10
0x12
0x23
0x34
.
EC x:0xA010
0x12
0x23
0x34
.
P1.0=0
d d:0x10,0x1F
d x:0xA000,0xA00F
d x:0xA010,0xA01F
KEYWAIT("")
EC X:0xA0FC
0xCC
.
q 0
d d:0x10,0x1F
d x:0xA000,0xA00F
d x:0xA010,0xA01F
d x:0xA020,0xA02F
d x:0xa0f0,0xa0ff
EC d:0x10
0x12
0x23
0x34
.
EC x:0xA010
0x12
0x23
0x34
.
KEYWAIT("")
/* Warmstart vom Monitor */
/* Speicher initialisieren **********************************************/
/* $=A000, [A000]=LJMP A100 [A023]=LJMP 1003 [A0FB]=BB [A0FD]=DD [A0FE]=C5") */
d d:0x10,0x1F
d x:0xA000,0xA00F
d x:0xA010,0xA01F
d x:0xA020,0xA02F
KEYWAIT("")
EC X:0xA0FC
0xCC
.
P1.0=1
g 0
d d:0x10,0x1F
d x:0xA000,0xA00F
d x:0xA010,0xA01F
d x:0xA020,0xA02F
d x:0xa0f0,0xa0ff
```

□

```
        (__)                (__)                (__)
        (\/)                ($$)                (**)
  /-------\/          /-------\/          /-------\/
 / | 666 ||          / |=====||          / |     ||
*  ||----||         *  ||----||         *  ||----||
   ^^    ^^            ^^    ^^            ^^    ^^

Satanic cow      This cow is a Yuppie      Cow in love
```