

Michael Baldischweiler

Dokumentation zur
Arbeiten mit der Start900.A51 (LPC900-Familie)
V 1.0

MEBA

Weitere Informationen zu den LPC900-Beschreibungen erhalten Sie unter www.c51.de

Inhaltsverzeichnis

<i>Inhaltsverzeichnis</i>	2
<i>1. Zu dieser Beschreibung</i>	4
1.1. Beschreibungskonventionen	4
1.2. Glossar	4
<i>2. Arbeiten mit der START900.A51</i>	8
2.1. Allgemeines	8
2.2. Verwenden der eigenen Startup-Datei	9
2.3. Konfiguration der Stackgröße	13
2.4. Wann wird die START900.A51 noch benötigt ?	14
<i>3. Literaturverzeichnis</i>	15

Kapitel 1

Zu dieser Beschreibung

1. Zu dieser Beschreibung

1.1. Beschreibungskonventionen

Um die Übersichtlichkeit für den Leser zu gewährleisten, wurden verschiedene Gestaltungsformen verwendet. Somit haben sie einen leichteren Überblick und können schneller an einzelne Informationen gelangen.

Ausdruck Bezeichner 1 Bezeichner n

Ausdruck: Beschreibung des Ausdrucks

Bezeichner 1: Beschreibung des Bezeichners 1

... : ...

Bezeichner n: Beschreibung des Bezeichners n

Abbildung 1 Syntaxaufbau

Die Syntax eines Befehls oder eines Steuerparameters wird mit einer Beschreibungsregel dargestellt (siehe Abbildung 1 dunkelgrau hinterlegt). Der Ausdruck, um den es sich handelt, wird fett dargestellt. Ausdruck und Bezeichner werden im darunterliegenden Abschnitt erklärt (hellgrau hinterlegt).

Im Buch sind die einzelnen Kapitel bzw. Abschnitte mit graphischen Symbolen versehen. Diese zeigen den Informationsgehalt an. Hier finden sie eine Erklärung zu allen im Buch verwendeten Symbolen.



	Wenn sie dieses Symbol im Kapitel finden, dann sollten sie diese Informationen unbedingt beachten, um Fehler zu vermeiden.
	Dieses Symbol wird als Tip-Marker verwendet. Hier wird ein Verfahren oder ein Trick angegeben.
NEW	Alle mit NEW gekennzeichneten Beschreibungen sind ab Version 6.0 enthalten.

Tabelle 1 Erklärung der verwendeten Symbole

1.2. Glossar

Erklärung technischer Begriffe:

Adresse: Zahl oder symbolischer Name zur Identifizierung eines Registers, Speicherworts, Speicherbereichs, Eingabe- / Ausgabe-Kanals.

ANSI-C: Eine Definitionsnorm für die Programmiersprache C. ANSI-C wurde 1989 vom American National Standards Institute definiert.

Anweisung: Eine Anweisung kann eine Operation oder ein Funktionsaufruf sein. Jede Anweisung wird mit einem Semikolon abgeschlossen.

Argument: Werden bei einem Funktionsaufruf Werte übergeben, so spricht man von einer Argumentübergabe.

A51-Assembler: Ein Programm zur Übersetzung eines in Assemblersprache geschriebenen Programms in ein auf dem Zielprozessor ablauffähiges Maschinenprogramm.

CISC-Architektur (Complex Instruction Set Computer). Typisch für diese Architektur ist die große Anzahl von unterschiedlichen Befehlen und Befehlsgruppen.

Weitere Informationen zu den LPC900-Beschreibungen erhalten Sie unter www.c51.de

-
- Compiler:** Ein Programm zum Übersetzen von Sourcemodulen in einen Objektcode.
- Cross-Compiler:** Ein Compiler, der aus Sourcemodulen einen Objektcode für andere Prozessoren erzeugt.
- Definition:** Bei der Definition einer Variablen, Funktion usw. wird vom C-Compiler Speicherplatz, z. B. im RAM oder ROM, reserviert.
- Deklaration:** Einem Sourcemodul wird eine Variable oder Funktion bekanntgegeben. Bei einer Deklaration wird kein Speicherplatz reserviert.
- Funktion:** Eine Funktion ist eine in sich abgeschlossene Operation. Die Funktion wird über ein CALL aufgerufen und mit einem RET abgeschlossen.
- Harvard-Architektur:** Die Speicherbereiche von ROM und RAM werden mit eigenen Befehlsgruppen verarbeitet. ROM und RAM können denselben Adressraum verwenden.
- von-Neumann-Architektur (VNM):** Die Speicherbereiche ROM und RAM werden mit derselben Befehlsgruppe abgearbeitet.
- H-File (Header-File):** In diesem File stehen alle Definitionen und Deklarationen, die ein Programm benötigt.
- IDE (Integrated Development Enterprise):** Unter einer IDE wird eine integrierte Entwicklungsumgebung verstanden, die alle Tools für die Entwicklung einer bestimmten Software beinhaltet.
- Ini-File:** Mit diesem File können Kommandos im Simulator über das Command-Fenster eingelesen werden.
- I-File (Preprint-File):** Zusätzliches Output-File des C51-Compilers, wenn der Steuerparameter PREPRINT verwendet wurde.
- BL51-Linker:** Ein Programm zum Binden von OBJ-Files zu einem ablauffähigen Programm.
- LNK-File:** Eingabefile für den Linker. In diesem sind die Informationen enthalten, welche OBJ-Files und welche Libraries mit hinzugebunden werden sollen. Zudem können in ihm Angaben enthalten sein, wie die einzelnen Programmsegmente im Speicher abzulegen sind.
- LST-File:** Compiler und Assembler liefern als Output ein File mit der Erweiterung LST. Dieses File enthält in Abhängigkeit von Steuerparametern Informationen über den Verlauf einer Compilierung bzw. Assemblierung.
- main():** Jedes C-Programm wird mit dieser Funktion gestartet. Dies gilt auch für die C51-Umgebung.
- Makro:** Ein im Sourcecode erzeugter Ausdruck, der während des Compilerlaufes in einen Sourcecode expandiert wird.
- M51-File:** Der BL51 Linker erzeugt beim Linken ein File mit der Erweiterung M51. In ihm sind alle Informationen über die Aufteilung des Adressraumes sowie den benötigten Speicherbereich vorhanden.
- OBJ-File:** Der Output von Compiler und Assembler. Dieser Output wird dem Linker als Input übergeben.
- Operation:** arithmetischer oder logischer Ausdruck
- OPT-File:** Das OPT-File ist Bestandteil eines µVision2 Projekts. Es enthält alle Einstellungen für den Simulator, ROM-Monitor bzw. Emulator.

#pragma Steuerparameter: Sie beeinflussen die Tools, z. B. C51-Compiler bei der OBJ-Code Erstellung oder den BL51-Linker bei der Adressvergabe. Diese Steuerparameter sind Schlüsselwörter.

PAP (Programm Ablauf Plan): Graphische Darstellung für ein Programm bzw. für eine Programmsequenz.

PC (Program Counter): Er enthält die Speicheradresse des Programmspeichers, die im Mikrocontroller ausgeführt wird.

RAM: Random Access Memory

RISC (Reduced Instruction Set Computer): Diese Architektur verwendet nur einfache Befehlsgruppen. Jeder Befehl wird in einem Maschinenzklus ausgeführt.

ROM: Read Only Memory

Schlüsselwort: Der C-Compiler hat einige englische Wörter reserviert. Diese dürfen nicht anderweitig verwendet werden.

SFR: Special Function Register

Simulator: Der Simulator ist ein Programm, das einen Zielprozessor mittels Software nachbildet. Es können damit Programme ohne die Zielhardware getestet werden.

Sourcemodul: Ein File, in dem der Sourcecode enthalten ist. Ein Programm wird meist in mehrere Module aufgeteilt. Ein Modul sollte nicht mehr als fünf DIN A4 Seiten Sourcecode enthalten, da alles darüber Hinausgehende unübersichtlich wird.

SRC-File: Über den Steuerparameter SRC erzeugt der C51-Compiler als Output ein Assemblerfile. Dieses Modul hat die Extension SRC und kann direkt als Input für den A51 Assembler genommen werden.

Target: Das Target enthält den Zielprozessor (8051-Derivat) und die Tool spezifischen Einstellungen.

Tool-Chain: Sie enthält die Programmierwerkzeuge (C-Compiler, Assembler, Linker) für das ausgewählte Mikrocontroller Derivat.

uv2-File: Dieses File enthält alle Projektinformationen.

XRAM: eXternal Random Access Memory

Kapitel 2

Arbeiten mit der START900.A51

2. Arbeiten mit der START900.A51

2.1. Allgemeines

Wird ein Projekt mit C erstellt, wird automatisch die Funktion `main()` angesprungen. Zudem wird der SP (Stack Pointer) auf die erste Adresse oberhalb der verwendeten Variablen gesetzt und die globalen und lokal statischen Variablen werden initialisiert. Alle diese Aufgaben übernimmt die Startup-Sequenz, die automatisch beim Linken eines Projekts mit hinzugebunden wird. Die Startup-Sequenz besteht aus den Modulen `?C_STARTUP` und `?C_INIT`. Die Module sind in der Library `C51S.LIB`, `C51C.LIB` bzw. `C51L.LIB` enthalten. Welche Library verwendet wird, hängt von der Einstellung des Memory Models im Window „Options for Target“, Reiter Target ab.

Das Modul `?C_INIT` ist für die Initialisierung der globalen und lokal statischen Variablen zuständig. Das Modul `?C_STARTUP` ist für die Initialisierung des SP, das Löschen des Speichers usw. zuständig.

Um einen besseren Einblick in den Zusammenhang von diesen Modulen zu bekommen, erstellen sie folgendes Projekt:

Projektname	Verzeichnis	Verwendete Sourcemodule
StartupSequenz	Startup_1	TestStartup.c

Tabelle 2 Projekt StartupSequenz

①	<code>unsigned char ucGlobVal;</code>
	<code>main()</code>
	<code>{</code>
②	<code>unsigned int uiLoop;</code>
	<code>for (uiLoop = 0; uiLoop < 30; uiLoop++);</code>
	<code>while(1);</code>
	<code>}</code>

Listing 1 TestStartup.c

Öffnen sie nach dem Übersetzen des Projekts das File `StartupSequenz.M51`. Dieses File wird vom Linker erstellt und enthält alle Informationen, welche Sourcemodule zum Projekt hinzugebunden wurden und welche Module aus der Library verwendet wurden. In diesem Beispiel ist es nur das Modul `?C_STARTUP` (siehe Listing 2, ①) da die globale Variable nicht mit einem bestimmten Wert vorbelegt wurde. Der Startupcode hat eine Größe von `0x000C` (siehe ②).

	INPUT MODULES INCLUDED:			
	TestStartup.obj (TESTSTARTUP)			
①	C:\KEIL\C51\LIB\C51S.LIB (?C_STARTUP)			
	* * * * * C O D E M E M O R Y * * * * *			
	CODE	0000H	0003H	ABSOLUTE
	CODE	0003H	0010H	UNIT ?PR?MAIN?TESTSTARTUP
②	CODE	0013H	000CH	UNIT ?C_C51STARTUP

Listing 2 Auszug aus StartupSequenz.M51

Weitere Informationen zu den LPC900-Beschreibungen erhalten Sie unter www.c51.de

Fügen sie nun folgende Änderung in das Sourcemodul TestStartup.c ein und übersetzen sie das Projekt erneut.

```
① unsigned char ucGlobVal = 35;
```

Listing 3 Auszug aus TestStartup.c

Öffnen sie das File StartupSequenz.M51 erneut, falls sie es schon geschlossen haben. Andernfalls betätigen sie den Button „Ja“, wenn die Anforderung zum Reload des M51-Files kommt.

```
① INPUT MODULES INCLUDED:
    TestStartup.obj (TESTSTARTUP)
    C:\KEIL\C51\LIB\C51S.LIB (?C_STARTUP)
    C:\KEIL\C51\LIB\C51S.LIB (?C_INIT)
```

Listing 4 Auszug aus StartupSequenz.M51 (globale Variable vorbelegt)

Der Linker hat nun zusätzlich das Modul ?C_INIT mit zum Projekt hinzugebunden (siehe Listing 5, ①). Wenn sie nun einige Zeilen im M51-File nach unten gehen, finden sie die Speicherplatzzuordnung (siehe Listing 5). Der SP beginnt ab der Adresse I:0x09 (siehe ①). Der Startupcode hat sich um 128 Byte auf 0x008C vergrößert (siehe ②). Dafür ist Modul ?C_INIT verantwortlich. Es enthält die Laderoutinen für die Vorbelegung der globalen und lokal statischen Variablen. Zudem ist ein 4 Byte großes ?C_INITSEG (siehe ③) hinzugekommen. In diesen vier Bytes ist der Wert 0x35 und die Lage im Speicher hinterlegt.

	TYPE	BASE	LENGTH	RELOCATION	SEGMENT NAME
	*****		D A T A	M E M O R Y	*****
	REG	0000H	0008H	ABSOLUTE	"REG BANK 0"
	DATA	0008H	0001H	UNIT	?DT?TESTSTARTUP
①	IDATA	0009H	0001H	UNIT	?STACK
	*****		C O D E	M E M O R Y	*****
	CODE	0000H	0003H	ABSOLUTE	
②	CODE	0003H	008CH	UNIT	?C_C51STARTUP
	CODE	008FH	0010H	UNIT	?PR?MAIN?TESTSTARTUP
③	CODE	009FH	0004H	UNIT	?C_INITSEG

Listing 5 Auszug aus StartupSequenz.M51 (Speicherplatzzuordnung)

2.2. Verwenden der eigenen Startup-Datei

Sie können eine eigene Startup-Datei nach ihren Bedürfnissen zu ihrem Projekt hinzufügen. Bei der LPC900-Familie ist es die Datei START900.A51. Wenn sie ein neues Projekt erstellen, öffnet sich nach der Auswahl des LPC900-Derivats automatisch folgendes Window:

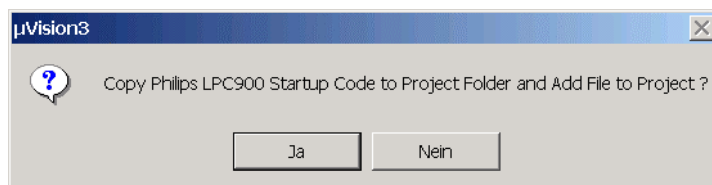


Abbildung 2 Window Copy-Anfrage

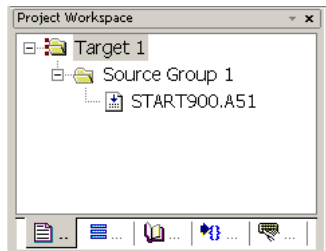
Weitere Informationen zu den LPC900-Beschreibungen erhalten sie unter www.c51.de

10 Kapitel 2

Erstellen sie das Projekt aus Tabelle 3 und betätigen sie den Button „Ja“, bei der Abfrage.

Projektname	Verzeichnis	Verwendete Sourcmodule
OwnStartup	Startup_2	TestOwnStartup.c

Tabelle 3 Projekt OwnStartup



Die Datei START900.A51 wird nun automatisch in ihr Projektverzeichnis kopiert und zu ihrem Projekt hinzugefügt (siehe Abbildung 3). Öffnen sie die START900.A51 im Editor.

Abbildung 3 Window Project Workspace

Die START900.A51 gliedert sich in folgende Bereiche:

- Angabe der Speichergrößen für das Löschen der RAM-Bereiche

Mit der Definition IDATALEN (siehe Listing 6, ①) wird die Länge des idata-Bereichs angegeben (siehe Abbildung 4). Diese muss für die kleineren LPC900-Derivate auf den Wert 0x80 gesetzt werden.

☞ !! Die C_STARTUP aus der Library löscht nur die ersten 128 Bytes im idata-Bereich. !!

Mit der Definition XDATASTART (siehe Listing 6, ②) wird die Startadresse des zu löschenden Bereichs angegeben. Mit XDATALEN wird die Anzahl der zu löschenden Bytes definiert. Beim P89LPC932 entspricht es dem Wert 0x200.

	;	the absolute start-address of IDATA memory is always 0
①	IDATALEN	EQU 100H ; the length of IDATA memory in bytes.
	;	
②	XDATASTART	EQU 0H ; the absolute start-address of XDATA memory
③	XDATALEN	EQU 0H ; the length of XDATA memory in bytes.

Listing 6 Auszug aus START900.A51

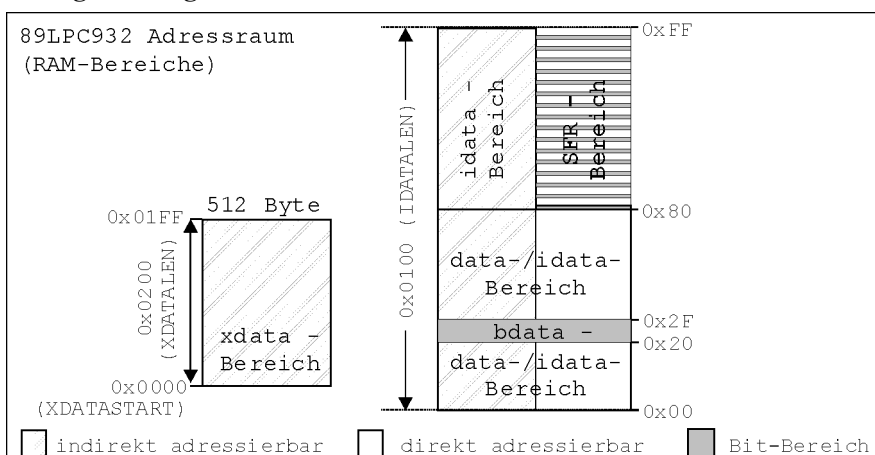


Abbildung 4 RAM-Adressbereich beim P89LPC932

Weitere Informationen zu den LPC900-Beschreibungen erhalten Sie unter www.c51.de

- Initialisierung der Bereiche für den reentrant-Stack

Werden reentrant-Funktionen verwendet, also Funktionen die z. B. rekursiv aufgerufen werden bzw. Funktionen, die von einer ISR aufgerufen werden, muss der reentrant Stack definiert werden. Je nach verwendetem Speichermodell muss die Definition IBPSTACK (siehe ①), XBPSTACK (siehe ②) bzw. PBPSTACK (siehe ③) auf „1“ gesetzt werden.

☞ !! Mehr Informationen zu den reentrant-Funktionen und die Funktionsweise des reentrant-Stacks finden Sie im Buch Teil 1, Kapitel 8.14. !!

```

; Stack Space for reentrant functions in the SMALL model.
① IBPSTACK EQU 0 ; set to 1 if small reentrant is used.
   IBPSTACKTOP EQU 0FFH+1 ; set top of stack to highest location+1.
;
; Stack Space for reentrant functions in the LARGE model.
② XBPSTACK EQU 0 ; set to 1 if large reentrant is used.
   XBPSTACKTOP EQU 01FFH+1 ; set top of stack to highest location+1.
;
; Stack Space for reentrant functions in the COMPACT model.
③ PBPSTACK EQU 0 ; set to 1 if compact reentrant is used.
   PBPSTACKTOP EQU 0FFH+1 ; set top of stack to highest location+1.

```

Listing 7 Auszug aus START900.A51

- Initialisierung der Konfigbytes (UCFG1, UCFG2, BOOTVEC,)

In diesem Abschnitt des Assemblermoduls können sie die Konfigurationsregister der LPC900-Familie definieren. Diese liegen ab der Adresse c:0xFFFF0 und umfassen insgesamt 12 Bytes. Eine ausführliche Beschreibung zu den einzelnen Konfigurationsregistern und deren Bits finden sie im Buch „Keil C51/Philips LPC900“, Kapitel 10. Über die einzelnen Definitionen (siehe Listing 8, ①, ②) können die einzelnen Bits für die jeweiligen Konfigbytes definiert werden.

```

; Setup LPC9xx Configuration Register (UCFG1, BOOTVEC, BOOTSTAT,..)
; Oscillator Configuration (UCFG1.0 .. UCFG1.2)
; FOSC      Val  Description
; ----      -
① FOSC EQU 3 ; 0 = high freq. crystal or resonator (4 .. 20MHz)
               ; 1 = medium freq. crystal or resonator (100kHz .. 4MHz)
               ; 2 = low freq. crystal (20kHz .. 100kHz)
               ; 3 = internal RC oscillator (7.373MHz +/- 2.5%)
               ;      (default on unprogrammed part)
               ; 4 = internal Watchdog oscillator (400kHz +20/-30%)
               ; 7 = external clock input on X1
; Watchdog Safety Enable (UCFG1.4)
; WDSE      Val  Description
; ----      -
② WDSE EQU 0 ; 0 = user can set WDCLK to select clock source
               (default on unprogrammed part)
               ; 1 = Always use WDCLK, WDCON and WDL can be written
               ;      once, WDT is always running
...

```

Listing 8 Auszug aus START900.A51

Weitere Informationen zu den LPC900-Beschreibungen erhalten sie unter www.c51.de

12 Kapitel 2

Die Bitdefinitionen aus Listing 8 werden mit OR-Anweisungen zur Definition _UCFG1 zusammengefasst (siehe Listing 9, ①). Mit Hilfe der Anweisung CSEG AT (siehe ②) werden die Bytes ab der Speicheradresse 0xFFFF0 im code-Bereich abgelegt. Die Funktionsweise von CSEG und DB können sie dem Buch „Keil C51/Philips LPC900“, Kapitel 11, Abschnitt 11.9 entnehmen.

①	<code>_UCFG1 EQU FOSC OR (WDSE SHL 4) OR (BOE SHL 5) OR (RPE SHL 6) OR (WDTE SHL 7)</code>		
	<code>_UCFG2 EQU 0</code>		
②	<code>CSEG</code>	<code>AT</code>	<code>0FFFF0H</code>
	<code>DB</code>	<code>_UCFG1</code>	<code>; place UCFG Bytes in hex file</code>
	<code>DB</code>	<code>_UCFG2</code>	<code>; reserved for future use</code>
	<code>DB</code>	<code>BOOTVEC</code>	
③	<code>DB</code>	<code>BOOTSTAT</code>	
	<code>DB</code>	<code>SEC0</code>	
	<code>DB</code>	<code>SEC1</code>	
	<code>DB</code>	<code>SEC2</code>	
	<code>DB</code>	<code>SEC3</code>	
	<code>DB</code>	<code>SEC4</code>	
	<code>DB</code>	<code>SEC5</code>	
	<code>DB</code>	<code>SEC6</code>	
	<code>DB</code>	<code>SEC7</code>	

Listing 9 Auszug aus START900.A51

- Löschroutinen für die einzelnen RAM-Bereiche

In Abhängigkeit der Definitionen aus Listing 6 und Listing 7 werden Assemblerbereiche zur Assemblierung freigeschaltet (siehe Listing 10, ①) und Register bzw. Speicherzellen (siehe ②) vordefiniert.

①	<code>IF IDATALEN <> 0</code>		
②	<code>MOV</code>	<code>R0, #IDATALEN - 1</code>	
	<code>CLR</code>	<code>A</code>	
	<code>IDATALOOP:</code>	<code>MOV</code>	<code>@R0, A</code>
		<code>DJNZ</code>	<code>R0, IDATALOOP</code>
	<code>ENDIF</code>		
①	<code>IF XDATALEN <> 0</code>		
②	<code>MOV</code>	<code>DPTR, #XDATASTART</code>	
②	<code>MOV</code>	<code>R7, #LOW (XDATALEN)</code>	
①	<code>IF (LOW (XDATALEN)) <> 0</code>		
②	<code>MOV</code>	<code>R6, #(HIGH (XDATALEN)) + 1</code>	
	<code>ELSE</code>		
②	<code>MOV</code>	<code>R6, #HIGH (XDATALEN)</code>	
	<code>ENDIF</code>		
	<code>CLR</code>	<code>A</code>	
	<code>XDATALOOP:</code>	<code>MOVB</code>	<code>@DPTR, A</code>
		<code>INC</code>	<code>DPTR</code>
		<code>DJNZ</code>	<code>R7, XDATALOOP</code>
		<code>DJNZ</code>	<code>R6, XDATALOOP</code>
	<code>ENDIF</code>		

Listing 10 Auszug aus START900.A51

Weitere Informationen zu den LPC900-Beschreibungen erhalten Sie unter www.c51.de

Erstellen sie noch das C-Sourcemodul „TestOwnStartup.c“ und fügen den Inhalt aus Listing 11 ein.

```
unsigned char ucGlobVal;

main()
{
    unsigned int uiLoop;
    for (uiLoop = 0; uiLoop < 30; uiLoop++);
    while(1);
}
```

Listing 11 TestOwnStartup.c

Übersetzen sie das Projekt mit dem Button “Build All“ und öffnen sie das M51-File. Das Projekt besteht aus den OBJ-Files START900.obj (siehe Listing 12, ① und dem TestOwnStartup.OBJ (siehe ②). Da das START900.obj das Label ?C_STARTUP enthält, wird kein Libraryfile zum Projekt hinzugebunden.

```
INPUT MODULES INCLUDED:
①  START900.obj  (?C_STARTUP)
②  TestOwnStartup.obj  (TESTOWNSTARTUP)
```

Listing 12 Auszug aus OwnStartup.M51

Der Linker erzeugt für dieses Beispiel folgende Speicheraufteilung:

	TYPE	BASE	LENGTH	RELOCATION	SEGMENT NAME

	* * * * *		D A T A	M E M O R Y	* * * * *
	REG	0000H	0008H	ABSOLUTE	"REG BANK 0"
	DATA	0008H	0001H	UNIT	?DT?TESTOWNSTARTUP
	IDATA	0009H	0001H	UNIT	?STACK
	* * * * *		C O D E	M E M O R Y	* * * * *
	CODE	0000H	0003H	ABSOLUTE	
	CODE	0003H	0010H	UNIT	?PR?MAIN?TESTOWNSTARTUP
	CODE	0013H	000CH	UNIT	?C_C51STARTUP
		001FH	FFD1H		*** GAP ***
①	CODE	FFF0H	000CH	ABSOLUTE	

Listing 13 Auszug aus OwnStartup.M51

Im Gegensatz zur C_STARTUP aus der Library wird ein 12 Byte großer Speicherbereich ab der Adresse c:0xFFF0 angelegt (siehe Listing 13, ①). Dieser enthält die Werte für die Konfigbytes des LPC900. Das FM (Flash Magic) programmiert automatisch die Werte anhand der Adresslage korrekt in die Konfigbytes.

2.3. Konfiguration der Stackgröße

Wenn sie eine eigene START900.a51 verwenden, können sie zudem die Größe des Stacks beeinflussen und die Startadresse für den SP vorgeben. Die Angaben erfolgen im Stacksegment ?Stack (siehe Listing 14, ①). Die Größe des Stacks wird über die Speicherdirektive DS vorgenommen (siehe ②). Per default wird nur ein Byte für den Stack definiert, da der Stack oberhalb der letzten Variablen liegt und nach oben wächst. Wenn

Weitere Informationen zu den LPC900-Beschreibungen erhalten sie unter www.c51.de

14 Kapitel 2

sie die maximale Tiefe des Stackbereichs wissen, sollten sie diese auch angeben. Somit kann schon beim Linkerlauf überprüft werden, ob es einen Stacküberlauf gibt.

①	?STACK	SEGMENT	IDATA
		RSEG	?STACK
②		DS	1

Listing 14 Auszug aus START900.A51

2.4. Wann wird die START900.A51 noch benötigt ?

Sie können eigenen Assemblercode nach dem Label STARTUP1: schreiben. Dieser ist dann erforderlich, wenn der WDT aktiv ist und das Löschen des Speichers zu lange dauert. In diesem Fall wird der Refresh-Code zwischen das Löschen der Speicherbereiche eingefügt (siehe Listing 15, ①).

	IF IDATALEN <> 0	
	MOV R0, #IDATALEN - 1	
	CLR A	
	IDATALOOP: MOV @R0, A	
	DJNZ R0, IDATALOOP	
	ENDIF	
	; Reload Sequenz fuer WDT	
①	MOV WFEED1, #0xA5	
①	MOV WFEED2, #0x5A	
	IF XDATALEN <> 0	
	MOV DPTR, #XDATASTART	
	MOV R7, #LOW (XDATALEN)	
	IF (LOW (XDATALEN)) <> 0	
	MOV R6, # (HIGH (XDATALEN)) + 1	
	ELSE	
	MOV R6, #HIGH (XDATALEN)	
	ENDIF	
	CLR A	
	XDATALOOP: MOVX @DPTR, A	
	INC DPTR	
	DJNZ R7, XDATALOOP	
	DJNZ R6, XDATALOOP	
	ENDIF	

Listing 15 Beispiel für den Refresh des WDT START900.A51

3. Literaturverzeichnis

Michael Baldischweiler	Der Keil C51-Compiler Teil 1 ELECTRONIC MEDIA Verlag ISBN 3-9804331-6-1
Michael Baldischweiler	Praxis mit dem C51-Compiler Teil 2 ELECTRONIC MEDIA Verlag ISBN 3-9804331-7-x
Michael Baldischweiler	Keil C51/Philips LPC900 Hardware-Software-Toolchain ELECTRONIC MEDIA Verlag ISBN 3-9804331-9-6
Rolf Klaus	Der Mikrocontroller 8051, 8052 und 80C517 Vdf Hochschule AG an der ETH Zürich ISBN 7-281-2478-8
Jürgen Walter	Mikrocomputertechnik mit der 8051-Controller Familie Springer Verlag ISBN 3-540-60540-1
Michael J. Pont	Patterns for Time-Triggered Embedded Systems ACM PRESS BOOKS ISBN 0-201-33138-1
Kernighan/Ritchie	Programmieren in C Hanser Verlag ISBN 3-446-15497-3