

Zenon Schymiczek

**Einführung in den Umgang
mit Meßsignalprozessor
MSP430**

Studienarbeit

Fachhochschule Mannheim-Hochschule für Technik und Gestaltung
Fachbereich Automatisierungstechnik/Elektrische Energietechnik
Institut für Elektronische Steuerungstechnik

Mannheim 2000

Danksagung

Besonderer Dank gebührt meiner Frau für ihre engelhafte Geduld, Unterstützung und Verständnis.

Herrn Professor Krüger und seinem Assistenten, Herrn Fath, die als meine Betreuer enorm viel Verständnis und Geduld für einen „mit Akzent sprechenden Studenten“ gezeigt haben und auch für deren Erklärungen, Anregungen und Verbesserungen möchte ich mich bedanken.

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig angefertigt, alle benutzten Quellen und Hilfsmittel vollständig und genau angegeben habe. Ich habe alles kenntlich gemacht, was ich aus Arbeiten anderer unverändert oder mit Änderungen übernommen habe.

Worms, 30.01.2000

Vorwort

Was bekamen Sie für ihr Geld?

6 Bücher, 2 Disketten, 1 Platine + Verbindungskabel! Schön! Da fragt sich mancher: „Wo bleibt der Joystick oder mindestens ein paar Leuchtdioden?“

Diesen Irrtum wollen wir gleich am Anfang klären. Bei dem Starter Kit handelt es sich nicht um ein Spielset, sondern um ein Werkzeug, mit Hilfe dessen man den Prozessor MSP 430 näher kennenlernt, aber auch die Programmiersprache ASSEMBLER lernen kann.

Der MSP 430 gehört zu der Familie der digitalen Prozessoren (DSP - Digital Signal Prozessor), die meistens im Verborgenen ihre Arbeit tun und auf ihre spezielle Aufgabe zugeschnitten sind. Sicherlich können die DSP keinen PC steuern, sie sind jedoch in vielen Teilen eines PCs vorhanden, z.B. in Grafikkarten oder Netzwerkkarten. Meistens werden sie jedoch in technischen Anwendungen, wie Unterhaltungselektronik, Steuerungen, Meßgeräten und Sensoren aller Art benutzt.

Die Assemblersprachen der unterschiedlichen Prozessorfamilien sind oft recht ähnlich, die Abweichungen sind meistens durch den technischen Aufbau bedingt.

Was brauchen Sie noch?

Einen PC, etwas Zeit (die ist aber relativ) und viel Geduld. Denken Sie immer daran: Die „Maschine“ tut nur das, was WIR wollen, die Frage lautet aber: „Wissen wir, was wir wollen, und haben wir das unserer Maschine auch korrekt vermittelt?“

Inhaltsverzeichnis	Seite
1. Einleitung	1
2. Beschreibung des STK	2
2.1 Hardware	2
2.2 Software	3
2.3 Bücher und Unterlagen	5
3. Installation der Software	6
3.1 Vor der Installation	6
3.2 Installation der Software	7
4. Der erste Test	9
5. Beschreibung der Software	10
5.1 Simulations Environment	10
5.1.1 Menü-Leiste	11
5.1.2 Kurzes Beispiel zum Simulator	13
5.2 ASM430 Assembler	15
5.3 STK430-Terminal	18
5.3.1 HyperTerminal-Fenster	18
5.3.2 Übersicht über die Einstellungen	19
5.3.3 Senden der .txt-Datei an die STK	21
5.3.4 Der EPROM-Monitor	22
5.3.5 Kurzes Beispiel zum Assembler, Terminal und EPROM-Monitor	23
6. Details des Prozessors	24
6.1 Das Wesen des Interrupts	24
6.2 Die wesentlichen Bestandteile des Prozessors	25
6.2.1 CPU	25
6.2.2 Register	25
6.2.3 Speicher	26
6.2.4 Peripherie	27
7. Details eines Assemblerprogramms	30
7.1 Die wesentlichen Bestandteile eines Programms	30
7.1.1 Kommentare	30
7.1.2 Definitionen	30
7.1.3 Initialisierungen	31
7.1.4 Hauptprogramm	31
7.1.5 Routinen/Unterprogramme	32
7.1.6 Interruptroutinen/Unterbrechungsrountinen	32
7.1.7 Tabellen	32
7.1.8 Vektorentabelle	33
7.2 Abarbeiten eines Programms durch den Prozessor	34
7.2.1 Hauptprogramm	34
7.2.2 Routinen	34
7.2.3 Interrupt/Unterbrechung	34
8. Die Beispielprogramme	35
8.1 Die .lst-Dateien	35
8.2 Beschreibung der Beispielprogramme	35
WD_BSP	
PO_0_BSP	
AD_BSP	
8.4 Erläuterungen zu den Programmen	38
Literaturverzeichnis	40
Stichwortverzeichnis	41
Anhang	

1. Einleitung

Diese Studienarbeit ist eine Hilfe für alle Studenten, die das MSP430 Starter Kit bei Erlernen der Assemblerprogrammierung einsetzen. Diese Einführung soll als Ergänzung und nicht als Ersatz für die mitgelieferten Bücher und den von Prof. Krüger geschriebenen Skript (Mikrocontroller MSP430, Eine Einführung in die Assemblerprogrammierung energiesparender Mikrocontroller) dienen.

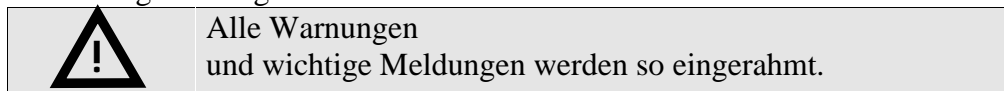
Diese Arbeit hilft Ihnen bei der Installation der Software, erklärt anschließend, wie sie funktioniert, und zeigt die ersten Schritte im Umgang mit dem gesamten Starter Kit. Sie werden auch einige Details des Prozessors und der Assemblerprogramme kennenlernen. Es werden drei Beispielprogramme vorgestellt, die für Sie den ersten Kontakt mit Programmieren im Assembler darstellen werden.

Was erwartet Sie in den einzelnen Kapiteln:

- **Kap. 2** wird Ihnen kurz den Inhalt des MSP430 Starter Kit's beschreiben, damit Sie den Überblick über die vielen Bücher, die Hard- und Software bekommen.
- **Kap. 3** hilft Ihnen, Schritt für Schritt, die mitgelieferte Software zu installieren.
- **Kap. 4** wird Ihnen, nach der erfolgreichen Installation, einen ersten Erfolg bei der Arbeit mit dem MSP430 Starter Kit beschern. Sie werden den ersten Test des STK vornehmen und sehen wie das Demoprogramm funktioniert.
- **Kap. 5** beschreibt die Programme, die Sie nach der Installation in der ADT430-Gruppe finden. Sie finden in diesem Kapitel auch zwei Beispiele, in denen Sie spielerisch, Schritt für Schritt, den Umgang mit der Software lernen.
- **Kap. 6** ist als Nachschlagewerk gedacht, es beschreibt kurz die Teile, aus denen der Prozessor besteht, und deren Funktionen. Am Anfang des Kapitels wird zunächst der Interrupt beschrieben, da diese Funktion des Prozessors sehr eng mit seiner Struktur zusammenhängt.
- **Kap. 7** gibt Ihnen einen Überblick über die Bestandteile eines Assemblerprogramms und wie diese vom Prozessor umgesetzt werden.
- **Kap. 8** beschreibt drei Programme, die die Fähigkeiten des MSP430 Starter Kit's zeigen und für Sie als Grundlage für eigene Programme dienen können. Die Beispielprogramme sollen Ihnen vor allem den Umgang mit der LCD-Anzeige, dem Watchdog, dem P0.0-Eingang und dem AD-Wandler näher bringen.

In den Programmen finden Sie aber auch viele Routinen, die Sie in eigenen Programmen nutzen können und sollen. *Vorteil: sie sind kommentiert ☺*


Jetzt noch einige wichtige Details zur Textstruktur:




- Alle Verweise in die (gelben) Bücher finden Sie in den eckigen Klammern [Buch, Seite]. Um den Text nicht unnötig zu verlängern, werden für die Titel der Bücher nur Abkürzungen benutzt z.B. statt [MSP430 Family Architecture User's Guide and Module Library, Data Book User's Guide, 14-2] einfach [DB, 14-2]. Alle Abkürzungen finden Sie im Kap. 2.3
- Texte, die das Bedienen des PC und der Software betreffen, werden mit diese Schriftart hervorgehoben, z.B. c:\adt430\examples\stk.
- Texte die aus den Beispielprogrammen stammen, haben diese Schriftart, z.B. MOV #1h,R5 .

2. Beschreibung des STK

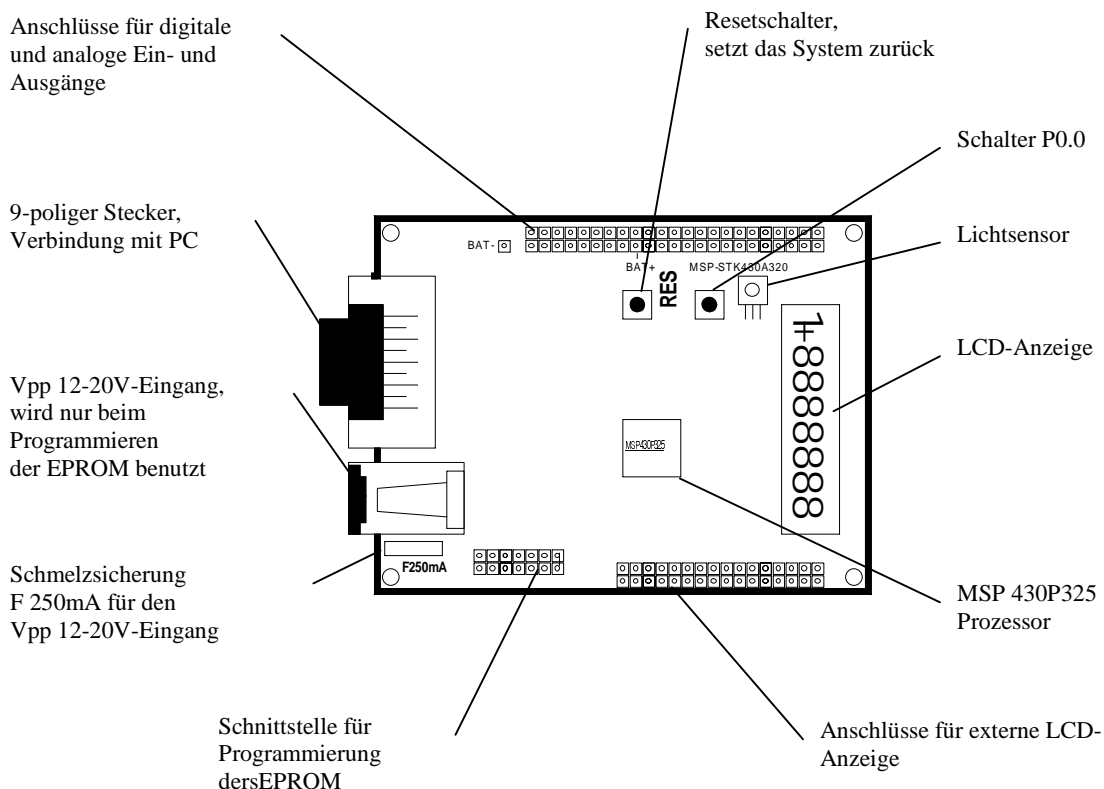
2.1 Hardware [SKM, 1-4]

 Den Vpp (12-20V)-Eingang nicht benutzen. Die Platine entnimmt die Versorgungsenergie der seriellen Schnittstelle. Der Vpp (12-20V)-Eingang wird lediglich beim Programmieren des EPROM-Speichers benutzt und ist für unsere Übungszwecke irrelevant.

 Vor dem Anschließen an den Computer bitte zuerst die Software installieren.

Zu der Hardware des Starter Kit's gehören eine Platine (weiter im Text als STK bezeichnet) und ein Verbindungskabel, um den STK an den PC anzuschließen. Auf der Platine finden Sie den Prozessor, eine LCD-Anzeige, einen Lichtsensor, einen Schalter, der direkt mit dem I/O-Eingang P0.0 verbunden ist und einen, der eine Initialisierung des STK verursacht (Reset).

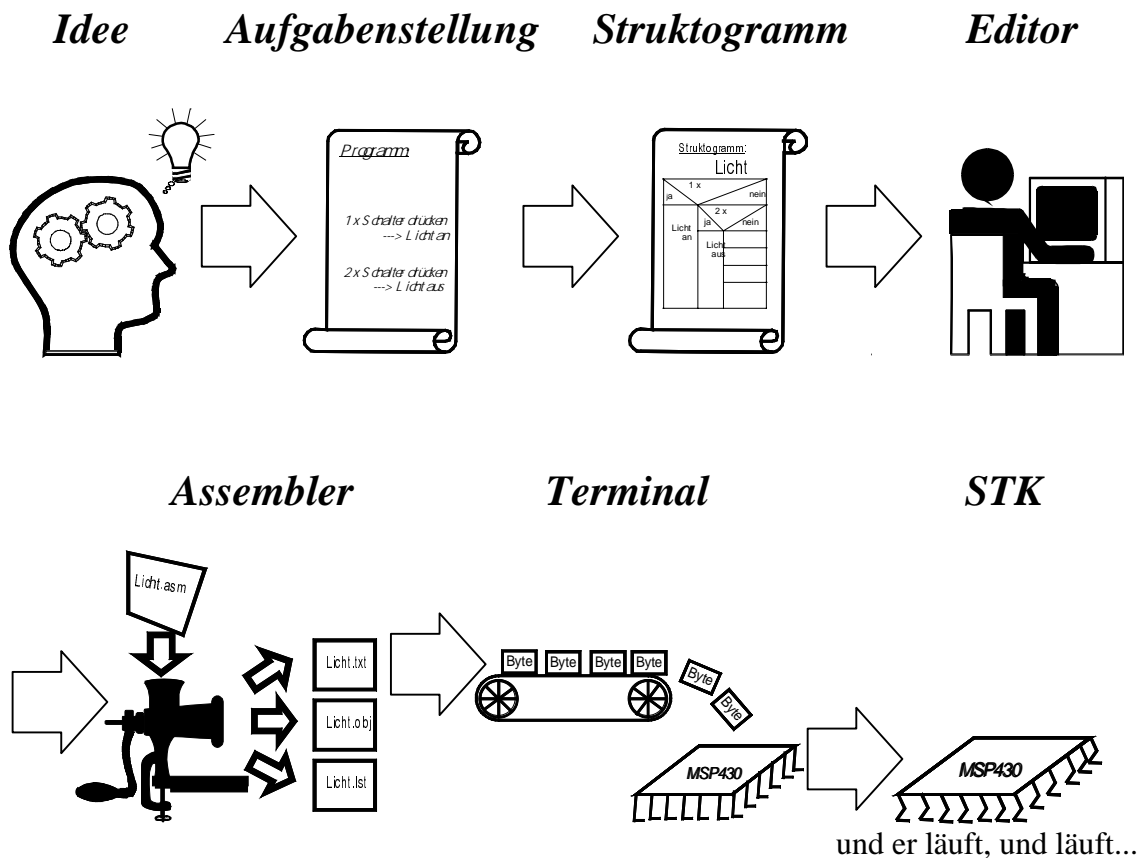
Hier deren Positionierung auf der Platine [SKM, 1-6].



2.2 Software

Auf den mitgelieferten Disketten befinden sich Programme, die für die Programmierung und Bedienung des STK notwendig sind (Alternativ ist die Software im FH-Netz zu finden).

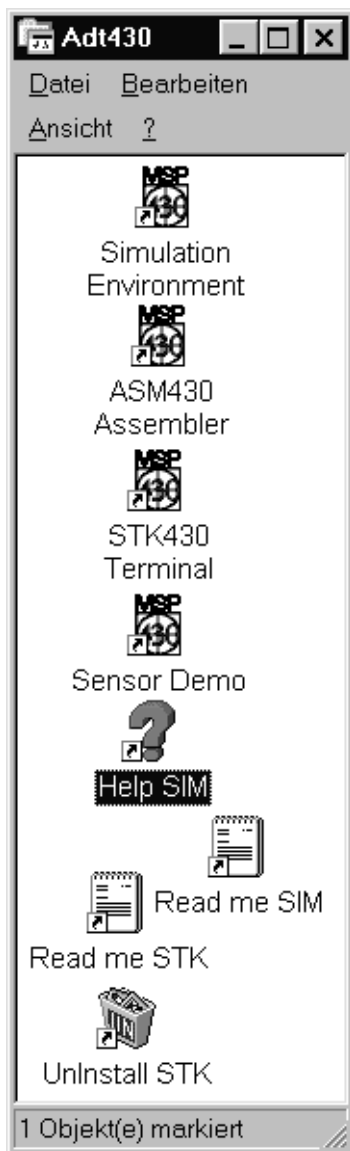
Vor der Präsentation der einzelnen Programme erfolgt eine kurze Beschreibung des Weges von der Idee bis zum fertigen Assemblerprogramm. Auf diesem Weg werden die verschiedenen Programme der ADT340-Gruppe benutzt. Die ADT430-Programmgruppe finden Sie nach der Installation der mitgelieferten Software auf ihrem PC.



- Am Anfang steht die Idee, ähnlich wie es beim Programmieren mit anderen Computersprachen geschieht, wird auch hier die Idee zuerst in ein Struktogramm umgesetzt.
- Danach wird aus dem Struktogramm, mit Hilfe eines Texteditors (z.B. der, der im Simulations Environment-Programm integriert ist), eine Quelltextdatei, die einen Suffix `.asm` erhält (im C wäre es z.B. `.c` bzw. `.cpp`).
- Ähnlich wie in der Programmiersprache C muß diese noch in die Maschiensprache übersetzt werden. Dies geschieht im Assembler (ASM430Assembler), der als Ergebnis der Übersetzung unter anderen die Datei mit dem Suffix `.txt` liefert.
- In der `.txt`-Datei befindet sich unser Programm, jetzt aber in einer Form, die von dem Prozessor „verstanden“ wird. Genau diese `.txt`-Datei und nur diese wird über das Terminal (STK430Terminal) in den Speicher des Prozessors gesendet.
- Dann wird mit Hilfe des STK-Monitors das Programm vom PC aus gestartet.

Bei der von Texas Instruments vorgeschlagenen Installation finden Sie die Software in dem Verzeichnis ADT430 mit den Unterverzeichnissen ASM, DT430 und STK. Für die Bedienoberfläche (Desktop) wird die Gruppe ADT430 angelegt und in das Verzeichnis „Programme“ der Start-Liste eingebunden.

An dieser Stelle wird die ADT430-Gruppe mit den einzelnen Programmen nur kurz angesprochen, genauere Beschreibung der einzelnen Programme finden Sie im Kapitel 5.



- Simulations Environment – Programmumgebung, die die Hardware simuliert, beinhaltet Programmeditor und Linker
- ASM430 Assembler – Übersetzungsprogramm, kreiert u.a. aus der .asm-Datei die .txt-Datei, die für den Prozessor verständlich ist
- STK430 Terminal – Programmumgebung, die die Kommunikation und Dateien-Übertragung zwischen dem STK und dem PC ermöglicht
- Sensor Demo – Demonstrationsprogramm, Messung der Lichtintensität
- Help SIM – Hilfe zu den Programmen
- Read me SIM und Read me STK – Textdateien
- Uninstall STK – Deinstallationsprogramm, entfernt die STK-Software

2.3 Bücher und Unterlagen

Zu dem Inhalt des MSP-STK430X320-Pakets gehören:

- **drei Disketten** mit der Entwicklungsoftware für die MSP430-Prozessorfamilie
- **MSP-STK430B320** Entwicklungsplatine (weiter im Text als STK bezeichnet)
- **Verbindungskabel** zwischen STK und der seriellen Schnittstelle des PC's
- **Bücher:**

- [GS] **Getting Started With The MSP430 Microcontroller** – sehr kurze, einführende Beschreibung des Microcontrollers
- [SKM] **MSP430 Family Starter Kit/Evaluation Kit Manual** – Einführung in den Umgang mit dem STK
- [SE] **MSP430 Family Simulation Environment and LCD-Editor Manual** – Beschreibung des Simulatorprogrammes und des LCD-Editors
- [LUG] **MSP430 Family Assembly Language Tools User's Guide** – ausführliche Beschreibung der Befehle der Programmiersprache Assembler und des Linkers
- [DB] **MSP430 Family Architecture User's Guide and Module Library, Data Book User's Guide** – ausführliche Beschreibung des Prozessors und seiner Bestandteile
- [SUG] **MSP430 Family Software User's Guide** – Beschreibung der Befehle und der Interdependenz zwischen Hard und Software mit Beispielen
- [AR] **MSP430 Family Application Report** – Beispiele für Anwendungen des MSP-Prozessors mit Programmen und ausführlichen Beschreibungen
- [DS] **Datasheet MSP430x... Mixed Signal Microcontrollers** – Datenblätter zu den Prozessoren der MSP430-Familie.

Weitere interessante Publikationen zum Starter Kit bzw. MSP430 wie z.B.

Experiments for the MSP430 Starter Kit (SLAA079)

Understanding the MSP430x325 14-Bit ADC (SLAA039)

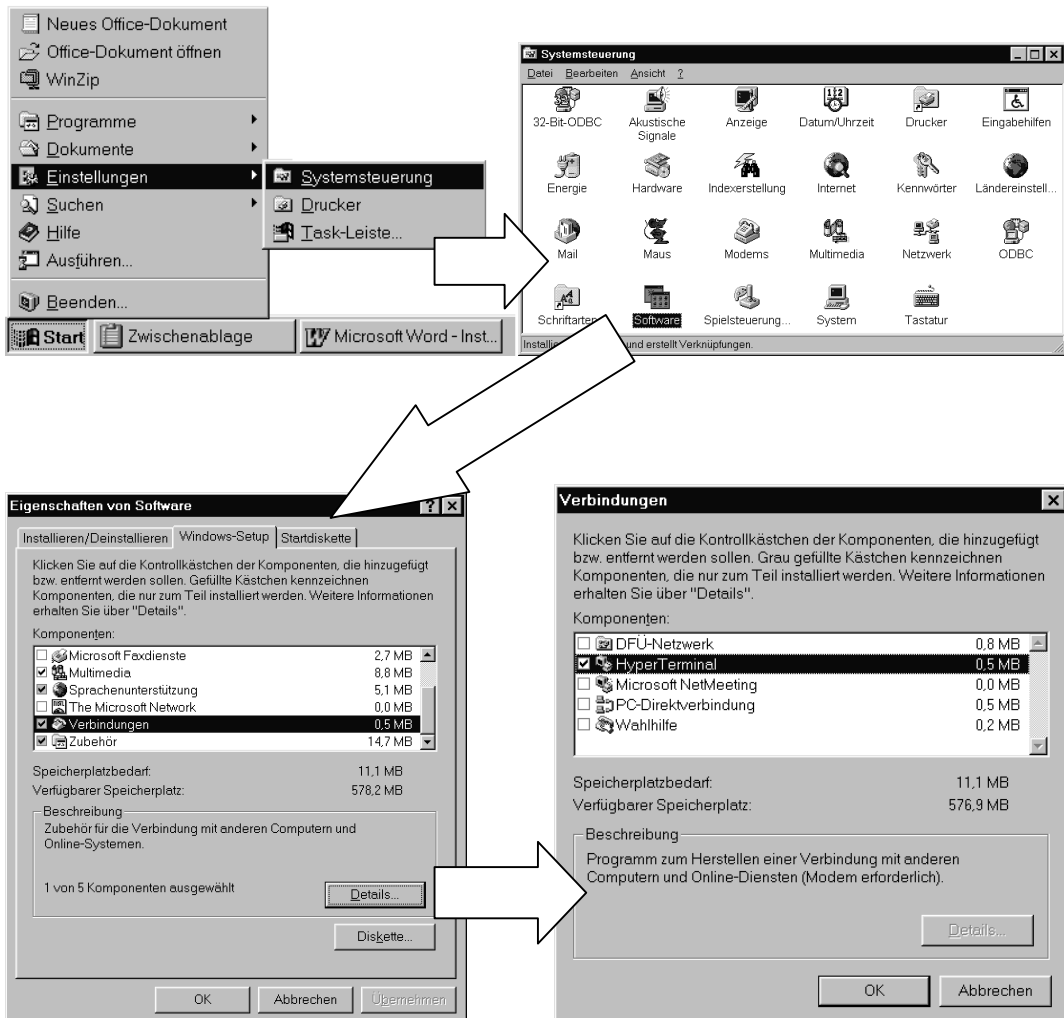
finden Sie unter der Internetadresse:

http://www.ti.com/sc/docs/apps/process/msp430_ultra_low_power_microcontroller.html#App_Notes

3. Installation der Software

3.1 Vor der Installation

- **Unter dem Betriebssystem WIN 95 bzw. WIN 98:**
Überprüfen Sie, ob der Hyperterminal verwendbar ist, dazu der folgender Weg:
Start/Einstellungen//Systemsteuerung/Software/Windows-Setup/Verbindungen/Details/Hyper Terminal

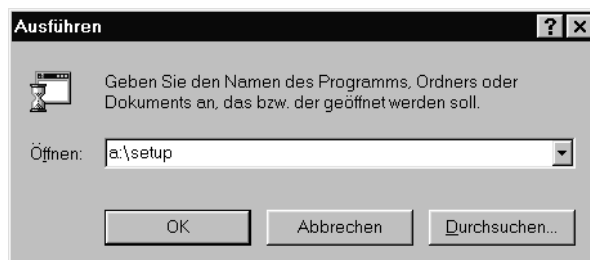


- Überprüfen Sie am PC (Rückwand), ob und welche der seriellen Schnittstellen (9-polige Stecker) frei sind?
- Schließen Sie alle unwesentlichen Programme ab.

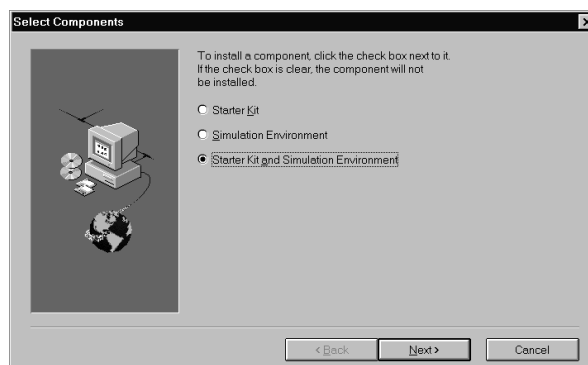
3.2 Installation der Software [SKM,1-1]

Die mitgelieferte Software installieren Sie folgendermaßen:

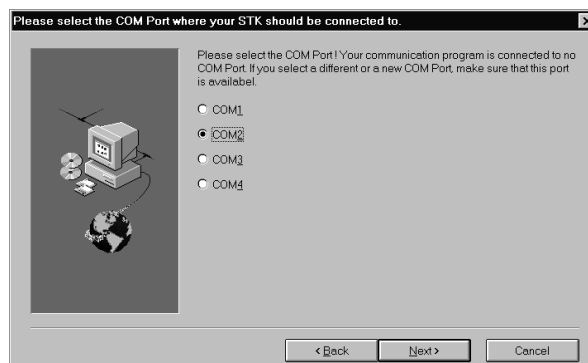
1. Schalten Sie Ihren Computer ein und legen Sie die erste Diskette in das Laufwerk a:\ ein.
2. Starten Sie das SETUP-Programm mit Hilfe der Funktion Ausführen (Start/Ausführen). Geben Sie a:\setup ein und bestätigen Sie mit OK.



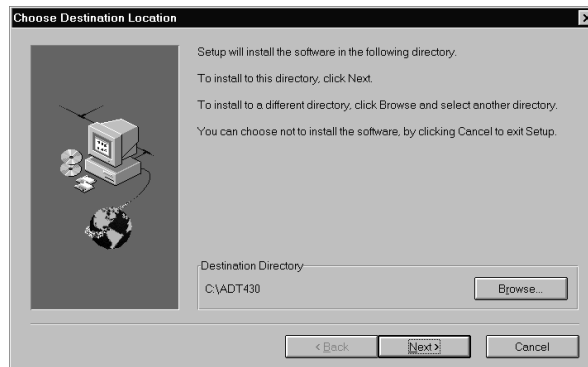
3. Befolgen Sie die im SETUP angegebene Schritte. Wählen Sie bei Bedarf die Next- Schaltfläche, um in der Installation fortzufahren, Back-Schaltfläche, um das vorhergehende Fenster zu erreichen, oder die Cancel-Schaltfläche, um abzubrechen.
4. Wählen Sie bei der ersten Abfrage Starter Kit and Simulation Environment.



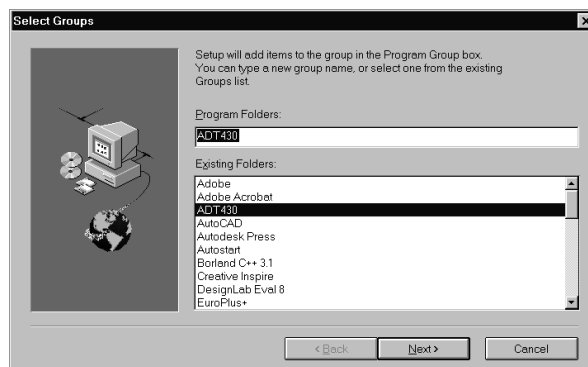
5. Als nächstes wählen Sie die Schnittstelle, an der STK angeschlossen wird.



6. Lesen Sie die Lizenzbedingungen (License agreement) und bestätigen Sie mit OK.
7. Das SETUP-Programm schlägt als Standardverzeichnis für die STK-Software das Verzeichnis c:\adt430 vor. Falls Sie es ändern wollen, klicken Sie den Browse-Schalter.



8. Nun wechseln Sie die Disketten, wie es von dem SETUP-Programm verlangt wird.
9. Das SETUP-Programm schlägt die Programmgruppe ADT430 für die STK Programme vor. Falls Sie es ändern wollen, schreiben den neuen Namen in das Program Folders Fenster.



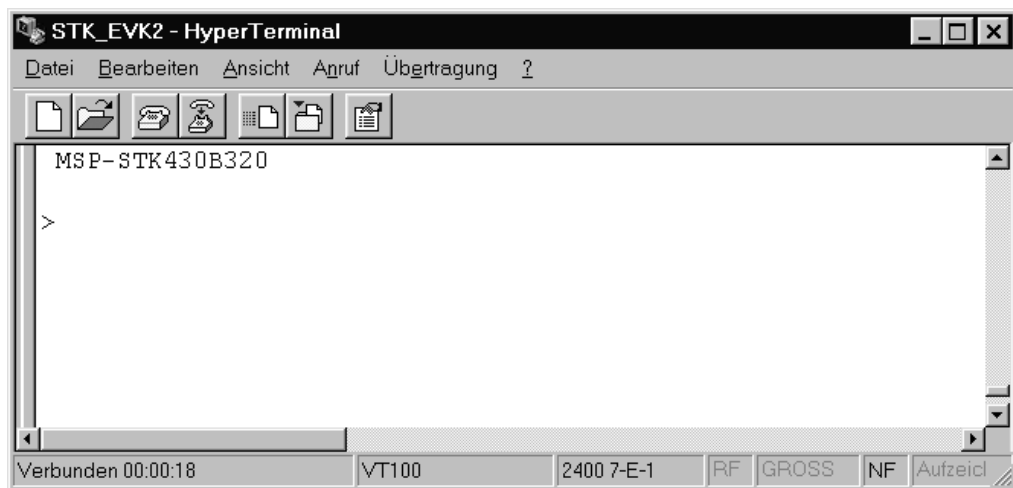
10. Damit ist die Installation beendet, lesen Sie noch kurz die Readme-Datei und beenden das SETUP-Programm mit YES.

4. Der erste Test [SKM,1-12]

Nun, nach der erfolgreichen Installation der Software, wird die Verbindung zwischen dem PC und STK überprüft. Die Überprüfung führen wir mit Hilfe des Demonstrationsprogramms, das über den Lichtsensor die Lichtintensität ermittelt und diese als Dezimalwert auf der LCD-Anzeige anzeigt.

Hier die Vorgehensweise:

1. Schließen Sie den STK, über das mitgelieferte Verbindungskabel, an die bei der Installation gewählte serielle Schnittstelle des PC's (COM1, COM2, COM3 bzw. COM4) an.
2. Öffnen Sie STK430 Terminal-Programm in der entsprechenden Programmgruppe (Start/Programme/Adt430/STK430Terminal). Falls die Installation erfolgreich durchgeführt wurde und die Verbindung über das Hyperterminal ordnungsgemäß funktioniert, sieht man folgendes Fenster auf dem Monitor:



und auf der LCD-Anzeige des STK: **MSP430**



Falls das Terminal-Programm beim ersten Mal die Angabe von Vorwahlnummern o. ä. verlangt, diesen Vorgang abbrechen.

3. Drücken Sie den Reset-Knopf des STK (der Prozessor wird neu initialisiert).
4. Geben Sie ein „d“ (für Demo) über die Tastatur, um das Demoprogramm zu starten. Auf dem LCD-Display erscheint: **A 888** – die Zahl steht für die Lichtintensität, die mit dem Lichtsensor gemessen wird.
5. Wenn Sie die Funktion genügend lange beobachtet haben: drücken Sie „x“ um abzubrechen.

Falls Sie Mißerfolg erlitten, überprüfen Sie:

- Die Verbindung zwischen STK und PC (ist die COM richtig?).
- Einstellungen des Hyperterminals, dazu siehe Kapitel 5.3.

5. Beschreibung der Software

5.1 Simulation Environment (Simulations-Umgebung)

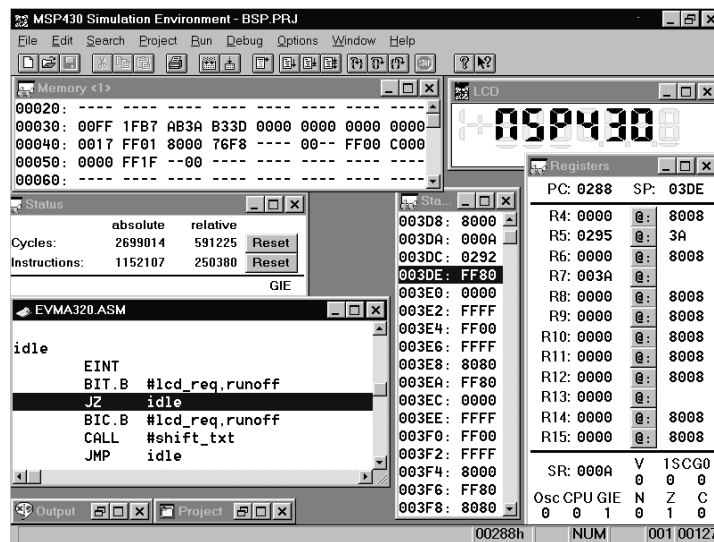
In vielen Bereichen der Technik werden verschiedene Prozesse und Abläufe simuliert. Oft ist dies direkt mit Kostenersparnis verbunden, noch öfter aber liegen die Gründe im besserem Verständnis der Vorgänge und der Möglichkeit, unterschiedliche Alternativen, die zu dem gleichem Ergebnis führen, auszutesten und dabei die Abläufe detailliert zu beobachten.

Die Simulations Environment Software ermöglicht es uns, den Ablauf des Programms zu simulieren und dabei die Vorgänge im Prozessor genau zu beobachten. Wir können die Inhalte der Register und des Speichers nach jeder Befehlszeile sehen und auch verändern.

Die Funktionen des Debuggers, der in dem Simulator integriert und bei der Fehlersuche im Programm von besonderen Bedeutung ist, erlauben die Ausführung einzelner Schritte und die Beobachtung der einzelnen Variablen bzw. Adressen. Eine dieser Funktionen, die bei der Programmierarbeit sehr nützlich und oft unverzichtbar ist, sind die Haltepunkte im Programm (Breakpoints). Dabei wird die Ausführung des Programms an einem Haltepunkt gestoppt und der Programmierer kann die Inhalte der Register und des Speichers in Ruhe analysieren. Voraussetzung: man weiß, was zu erwarten ist.

Wie Sie an der nächsten Abbildung sehen können, ist ein typisches Simulator-Fenster [SE,7-4] übersät mit unterschiedlichen Fenstern. Die Anzahl der offenen Fenster und deren Verteilung innerhalb des Simulator-Fensters ist vollkommen dem Benutzer überlassen.

Die Größe mancher Fenstern kann, durch Ziehen an den Ecken bzw. Seiten, verändert werden, es gibt aber auch starre Fenster, z.B. Status- und Register-Fenster, deren Größe man nicht beeinflussen kann.



Um ein bestimmtes Fenster zu öffnen, klicken Sie dessen Name in der Liste der Fenster an, die Sie in dem Menü-Element Windows finden.

An dem unterem Rand des Simulator-Fensters befindet sich die sog. Statuszeile [SE,7-6]. Dort finden Sie u.a. Informationen zu dem Befehl bzw. Button, an dem gegenwärtig der Mauszeiger steht, die Adresse des aktuellen Befehls im Programm oder Spalte und Zeile, in der sich der Cursor im Programmtext befindet.

5.1.1 Menü-Leiste [SE,7-16]

Die Menü-Liste des Simulators funktioniert ähnlich wie die Menü-Listen der meisten Windows Programme. Durch Anklicken des Menü-Elementes öffnet sich eine Liste von Funktionen, die diesem Oberbegriff (Menü-Element) zugeordnet sind. Durch Anklicken der Funktionen in dieser Liste werden Aktionen durchgeführt oder Dialogfenster geöffnet.

Alle Funktionen, hinter denen drei Punkte stehen, z.B. File/Open..., öffnen ein Dialogfenster.

Die Funktions-Elemente erscheinen manchmal grau, dies ist ein Hinweis darauf, daß diese im Moment inaktiv sind und nicht auf die momentan durchgeführten Aktionen angewendet werden können. Beispiel: bei einer laufenden Simulation sind alle Editorfunktionen (Edit) blockiert (grau), da das Programm während einer Simulation nicht verändert werden darf.

Direkt unter der Menü-Liste befindet sich die sog. Werkzeugkiste (Toolbar) [SE,7-47] mit den Ikonen (Buttons), die den oft benutzten Funktionen des Simulators zugeordnet sind.

Nachfolgend finden Sie eine kurze Aufzählung aller Funktionen der verschiedenen Menü-Elemente:

<u>F</u>ile	Funktionen mit Bezug auf Dateien [SE,7-17]
<u>N</u>ew	Öffnen eines neuen Editorfensters für eine neue .asm-Datei oder eine andere Datei
<u>O</u>pen...	Öffnen einer existierenden .asm-Datei, einer .prj-Datei oder einer anderen Textdatei
<u>C</u>lose	Schließen der aktuellen Datei (Editorfensters)
<u>S</u>ave	Speichern der aktuellen Datei
<u>S</u>ave <u>A</u>s...	Speichern der aktuellen Datei unter anderem Namen
<u>S</u>ave <u>A</u>ll...	Speichern alle geöffneten Dateien
<u>L</u>oad...	Öffnen einer existierenden, ausführbaren Objekt-Datei und Einfügen in das aktuelle Projekt
<u>P</u>rint...	Drucken der editierten Datei
<u>P</u>rint <u>P</u>review	Vorschau (Seitenansicht)
<u>P</u>age <u>S</u>etup...	Einrichten der Kopf- und Fußzeile
<u>P</u>rint <u>S</u>etup...	Einrichten des Druckers
<u>E</u>xit	Beenden des Simulationsprogramms
<u>E</u>dit	Funktionen zur Bearbeitung von Text [SE,7-22]
<u>U</u>ndo	Rückgängig
<u>R</u>edo	Wiederherstellen
<u>C</u>ut	Ausschneiden des markierten Textabschnittes
<u>C</u>opy	Kopieren des markierten Textabschnittes
<u>P</u>aste	Einfügen des ausgeschnittenen oder kopierten Textabschnittes
<u>D</u>elete	Löschen des markierten Textabschnittes
<u>S</u>elect <u>A</u>ll	Alles markieren
<u>D</u>uplicate <u>L</u>ine	Duplizieren der Zeile in der sich der Cursor befindet

<u>S</u>earch	Suchfunktionen [SE,7-24]
<u>F</u>ind...	Suchen nach bestimmtem Textabschnitt
<u>R</u>eplace...	Ersetzen eines Textabschnittes durch einen anderen
<u>S</u>earch Again	Wiederholen der letzten Suche
<u>G</u>o To Line...	Springe in die Zeile...
<u>P</u>rojekt	Funktionen zur Verwaltung des Projekts [SE,7-26]
<u>N</u>ew/Open...	Öffnen eines Projekts
<u>C</u>lose	Schließen eines Projekts
<u>C</u>lose <u>W</u>ithout Save	Schließen eines Projekts, ohne zu speichern
<u>A</u>dd File...	Einfügen einer Datei in das aktuelle Projekt
<u>R</u>emove File	Entfernen einer Datei aus dem aktuellen Projekt
<u>A</u>ssemble File	Assemblieren (übersetzen) der aktuellen Datei
<u>B</u>uild	Übersetzen aller <u>geänderten</u> .asm-Dateien und Binden des Programms
<u>R</u>ebuild All	Übersetzen <u>aller</u> .asm-Dateien und Binden des Programms
<u>R</u>un	Simulationsausführung [SE,7-29]
<u>R</u>un	Starten des Programms
<u>S</u>top	Anhalten des Programms
<u>H</u>old On Breakpoint	Ausführen des Programms bis zum Haltepunkt
<u>S</u>napshot On Breakpoint	Ausführen des Programms bis zum Haltepunkt, nach Ablauf der Haltezeit läuft das Programm weiter
<u>S</u>napshot <u>T</u>ime...	Einstellen der Haltezeit
<u>S</u>tep <u>I</u>nto	Ausführen der Programmsimulation schrittweise
<u>S</u>tep <u>O</u>ver	Ausführen der Programmsimulation schrittweise, Unterprogramme und Funktionen werden in einem Schritt komplett ausgeführt
<u>S</u>tep <u>O</u>ut	Ausführen des aktuellen (Unter-) Programms bzw. der aktuellen Interrupt-Service-Routine bis zum Rücksprung
<u>R</u>eset <u>P</u>rogram	Zurücksetzen des Programms
<u>D</u>ebug	Debuggerfunktionen [SE,7-32]
<u>B</u>reakpoints...	Setzen/löschen, aktivieren/deaktivieren der Haltepunkte
<u>T</u>oggle Breakpoint	Umschalten (aktiv/passiv) eines Haltepunktes
<u>A</u>dd Watch...	Einfügen einer zu überwachenden Variablen in die Liste
<u>R</u>emove Watch	Löschen einer Variable aus dem Überwachungsfenster
<u>M</u>emory Dump...	Speicherabzug
<u>M</u>odify Memory...	Modifikation von Speicherzellen
<u>O</u>ptions	Optionen des Simulators [SE,7-35]
<u>P</u>rojekt	Projektoptionen
<u>A</u>ssembler...	Einstellungen des Assemblers
<u>L</u>inker...	Einstellungen des Linkers
<u>D</u>irectories...	Verzeichnisse der .obj .lib .inc Dateien

Processor...	Auswahl des Prozessors
Supply Voltage...	Versorgungsspannung
I/O File Names...	Auswahl der Dateien für die Ein- und Ausgangs- simulation
Simulation Priority...	Priorität der Simulation
LCD Setup...	LCD-Display-Fenster in die Simulation einbinden
Editor...	Einstellungen des Editors
Font....	Schriftart, Schriftgröße usw.
Window	Funktionen zur Verwaltung der Fenster [SE,7-40]
Cascade	Ordnen der geöffneten Fenster überlappend
Tile Horizontally	Ordnen der geöffneten Fenster nebeneinander
Tile Vertically	Ordnen der geöffneten Fenster untereinander
Arrange Icons	Ordnen der geöffneten Fenster
Close All	Schließen aller Fenster
Toolbar	Ein/ausblenden der Werkzeugliste
Statusbar	Ein/ausblenden des Statusbalken
Register	Öffnen des Register-Fensters
Status	Öffnen des Status-Fensters
Stack	Öffnen des Stack-Fensters
Memory	Öffnen des Memory-Fensters
LCD	Öffnen des LCD-Display-Fensters (benutzerdefi- niertes Fenster, siehe Options/LCD Setup...)
Watch	Öffnen des Watch-Fensters
Output	Öffnen des Output-Fensters
Project	Öffnen des Project-Fensters
Help	Hilfe zum Simulator [SE,7-44]
Table Of Contents	Inhaltsverzeichnis
Search For Help On...	Suchen nach Thema
Using Help	Benutzen der Hilfe
About...	Über MSP430

5.1.2 Kurzes Beispiel zum Simulator

Eine kurze Einführung in die Arbeit mit dem Simulator wird hier am Beispiel des Programms EVMA320.ASM, das Sie im Verzeichnis c:\adt430\dt430\examples finden, dargestellt:

1. Öffnen Sie das Simulation Environment Programm.
2. Überprüfen Sie die Verzeichnisse, in die ihre .inc, .lib und .obj-Dateien [SE,7-14] gespeichert sind: Options/Project/Directories. Falls nötig, legen Sie ein neues Verzeichnis für ihre Projektdateien an und geben Sie den Weg in Directories ein.
3. Öffnen Sie ein neues Projekt: Project/New [SE,3-5] [SE,6-4] [SE,4-4].
4. Wählen Sie den Prozessor MSP430P325 im Dialogfenster Configure New Project aus [SE,3-5] [SE,6-5]. Das Fenster öffnet sich automatisch, nachdem Sie neues Projekt geöffnet haben. Falls Sie den simulierten Prozessor zu einem späterem Zeitpunkt ändern wollen, öffnen Sie das gleiche Auswahlfenster über: Options/Project/Processor [SE,7-37] [SE,6-5].
5. Es erscheint das Project-Fenster [SE,7-14].

6. Addieren Sie die Datei EVMA320.ASM zum Projekt: mit Project/Add File... [SE,3-7] [SE,6-7] öffnen Sie das Dialogfenster Add File to Project. Wählen Sie die zuvor genannte Datei aus und addieren sie, durch Betätigen des Add-Buttons, zum Projekt. Mit Close verlassen Sie das Add File to Project -Dialogfenster.
7. Öffnen Sie ein Editorfenster mit dem Text der EVMA320.ASM-Datei : mit File/Open... öffnen Sie das Dialogfenster File Open.... Wählen Sie die zuvor genannte Datei aus. Mit OK bestätigen Sie die Wahl und verlassen das File Open...-Dialogfenster.
8. Überprüfen Sie, ob in der Datei EVMA320.ASM in der ersten Zeile des Programms: `SIM .set 1` steht. Falls nicht (`SIM .set 0`), ändern Sie dies und speichern das Programm: File/Save.
9. Wählen Sie die LCD-Anzeige aus: mit Options/LCD Setup öffnen Sie das Dialogfenster LCD Setup. Die LCD-Datei finden Sie unter `c:\adt430\dt430\examples\demo lcd`, markieren Sie diese. Mit OK bestätigen Sie die Wahl und verlassen das LCD Setup-Dialogfenster.
10. Öffnen Sie die folgenden Fenster im Simulator-Fenster: LCD, Status, Registers. Sie erreichen dies durch Anklicken der entsprechenden Fenster-Namen im Menü-Element Windows.
11. Assemblieren /Linken Sie das Projekt: mit Project/Build [SE,3-9] [SE,3-16] [SE,6-9].
12. Setzen Sie ein Haltepunkt im Programm: mit Debug/Breakpoint... öffnen Sie das Dialogfenster Breakpoints. Wählen Sie die folgenden Einstellungen aus:
 - Source Type: File/Line
 - File: EVMA320.ASM
 - Line: 187Mit Add setzen Sie diesen Haltepunkt und mit Close verlassen Sie das Dialogfenster Breakpoint.
13. Starten Sie die Simulation mit Run/Hold on Breakpoint [SE,3-16]. Das Programm wird an der Zeile 187 angehalten.
14. Alternativ zu dem Punkt 13 können Sie die Funktion Snapshot On Breakpoint [SE,3-16] benutzen. Die Simulation hält dann für kurze Zeit an dem Haltepunkt, danach wird sie fortgeführt. Die Haltezeit können Sie im Snapshot Time-Dialogfenster einstellen: Run/Snapshot Time....



Während des Simulationsablaufs werden die Fensterinhalte nicht aktualisiert. Deshalb genügt auch eine beliebig kurze Haltezeit zum Auffrischen der Fenster.

Die Abbildung des gesamten Simulator-Fensters sahen Sie am Anfang des Kapitels 5.1. Betrachten Sie die Anordnung der Fenster als einen Gestaltungsvorschlag.

5.2 ASM430 Assembler [SKM,3-3] [LUG,1-15]

Programme, die mit Hilfe eines Editors geschrieben wurden, sind noch nicht auf dem Prozessor lauffähig. Damit der Prozessor die Intentionen des Programmierers versteht, ist es notwendig, das Programm auf mögliche Fehler (Syntax-Fehler) zu untersuchen und in eine dem Prozessor verständliche Form zu übersetzen. Alles dies geschieht im Assembler beim Assemblieren (lt. Lexikon: Assembler – Computerprogramm, das ein in Assembler geschriebenes Programm in die rechnerereigene Maschinensprache übersetzt). Der für unseren Prozessor geeignete Assembler heißt ASM430.

Als Ergebnis des Assemblierens werden drei neue Dateien erzeugt mit den Suffixen .obj, .lst und .txt. Dabei verbirgt sich unter .obj eine Datei in Maschinensprache, unter .lst die für den Programmierer am besten geeignete Darstellung mit Quelltext und Maschinencode und unter .txt der Maschinencode als gut übertragbare Textdatei.

Bei evtl. Fehler im Programm (.asm-Datei) wird das Übersetzungsprozeß unterbrochen und der Assembler gibt eine Fehlermeldung aus. Beschreibung der Fehlermeldungen finden Sie im *MSP430 Family Assembly Language Tools User's Guide* im Anhang C.



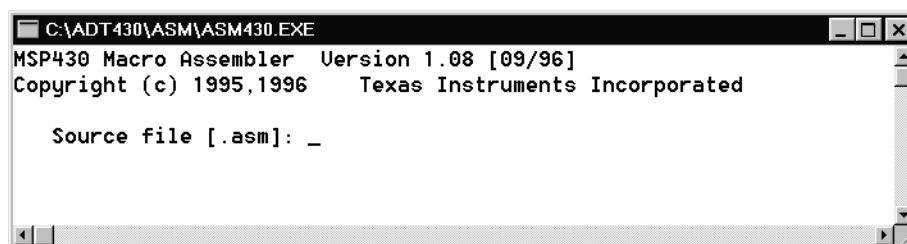
Die Dateien .obj, .lst und .txt werden in gleichem Verzeichnis wie die .asm-Datei gespeichert.



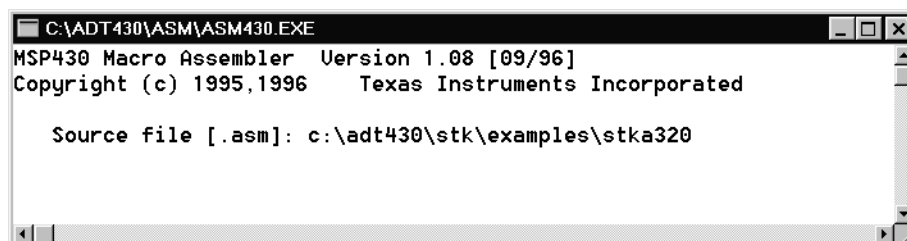
Schließen Sie das Assembler-Programm (jedes Mal) direkt nach seiner Benutzung, sonst kann es zu Konflikten mit dem Editor-Programm kommen.

Die Vorgehensweise beim Assemblieren:

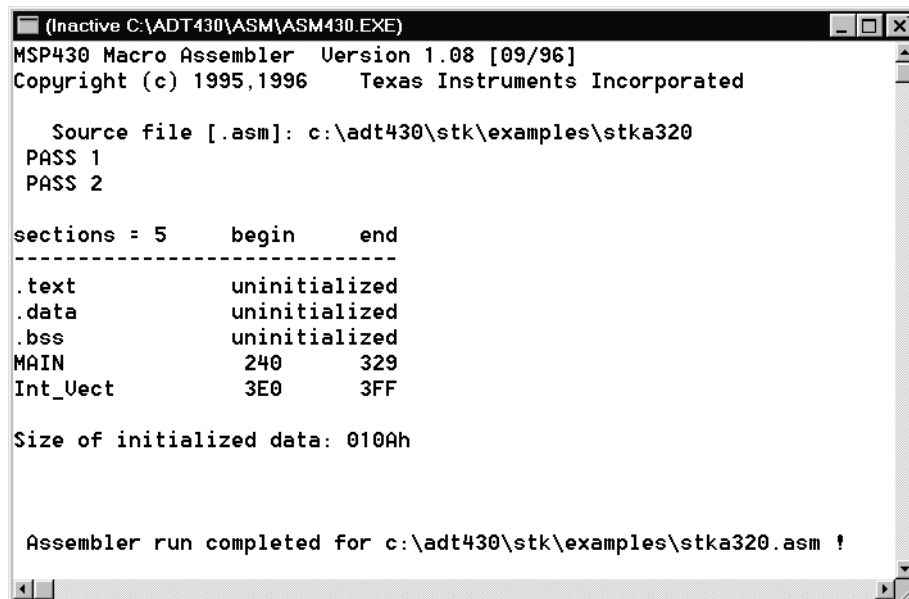
1. Starten Sie das ASM430-Programm in der ADT430-Gruppe. Es erscheint folgendes Fenster:



2. Geben Sie den Namen der .asm-Datei (!!ohne Suffix .asm) mit dem Verzeichnis an und bestätigen Sie mit Enter. Beispiel:



3. Als Ergebnis bei einem fehlerfreien Programm erscheint am Ende der Meldung folgender Satz: Assembler run completed for [SKM,7-16]. Beispiel:



```
(Inactive C:\ADT430\ASM\ASM430.EXE)
MSP430 Macro Assembler Version 1.08 [09/96]
Copyright (c) 1995,1996 Texas Instruments Incorporated

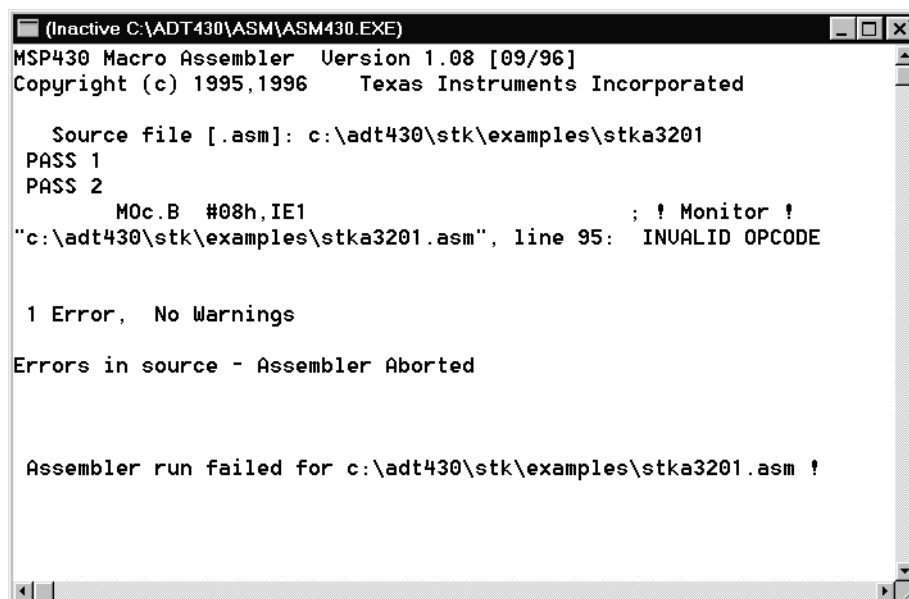
Source file [.asm]: c:\adt430\stk\examples\stka320
PASS 1
PASS 2

sections = 5      begin      end
-----
.text             uninitialized
.data             uninitialized
.bss              uninitialized
MAIN              240      329
Int_Uect         3E0      3FF

Size of initialized data: 010Ah

Assembler run completed for c:\adt430\stk\examples\stka320.asm !
```

4. Bei einem fehlerhaften Programm [LUG,A-3] wird die Zeile angegeben, die den Fehler enthält, die Zeilennummer, die Anzahl der Fehler und Warnungen, und am Ende der Meldung folgende Satz: Assembler run failed for Beispiel:



```
(Inactive C:\ADT430\ASM\ASM430.EXE)
MSP430 Macro Assembler Version 1.08 [09/96]
Copyright (c) 1995,1996 Texas Instruments Incorporated

Source file [.asm]: c:\adt430\stk\examples\stka3201
PASS 1
PASS 2
      M0c.B #08h,IE1                                ; ! Monitor !
"c:\adt430\stk\examples\stka3201.asm", line 95:  INVALID OP CODE

1 Error, No Warnings

Errors in source - Assembler Aborted

Assembler run failed for c:\adt430\stk\examples\stka3201.asm !
```

Das Ergebnis der Assemblierung (die Dateien .obj, .lst und .txt) finden Sie in dem Verzeichnis, aus dem die .asm-Datei stammte.

Als Beispiel der Ergebnisse der Assemblierung der evma320.asm-Datei, zwei übereinstimmende Ausschnitte, die die Vektorentabelle zeigen:

- aus der evma320.txt-Datei

```
@03E0
02 03 F0 02 40 02 40 02 02 03 02 03 40 02 40 02
40 02 40 02 02 03 40 02 02 03 02 03 40 02 40 02
q
```

▪ und aus der evma320.lst-Datei

```
265             ; * * * * *
266             ; Interrupt vectors
267             ; * * * * *
268
269 03e0         .sect      "Int_Vect",USER_END-31
270
271 03e0 +0302   .word      Int_P_27      ; Port0, bit 2 to bit 7
272 03e2 +02f0   .word      Int_BT       ; Basic Timer
273 03e4 +0240   .word      RESET       ; no source
274 03e6 +0240   .word      RESET       ; no source
275 03e8 +0302   .word      Int_TP       ; Timer/Port
276 03ea +0302   .word      Int_ADC     ; EOC from ADC
277 03ec +0240   .word      RESET       ; no source
278 03ee +0240   .word      RESET       ; no source
279 03f0 +0240   .word      RESET       ; no source
280 03f2 +0240   .word      RESET       ; no source
281 03f4 +0302   .word      Int_WDT_T    ; Watchdog/Timer, Timer mode
282 03f6 +0240   .word      RESET       ; no source
283 03f8 +0302   .word      Int_P0_1     ; Address of UART handler
284 03fa +0302   .word      Int_P0_0     ; P0.0
285 03fc +0240   .word      RESET       ; NMI, Osc. fault
286 03fe +0240   .word      RESET       ; POR, ext. Reset, Watchdog
287
288             .end
```

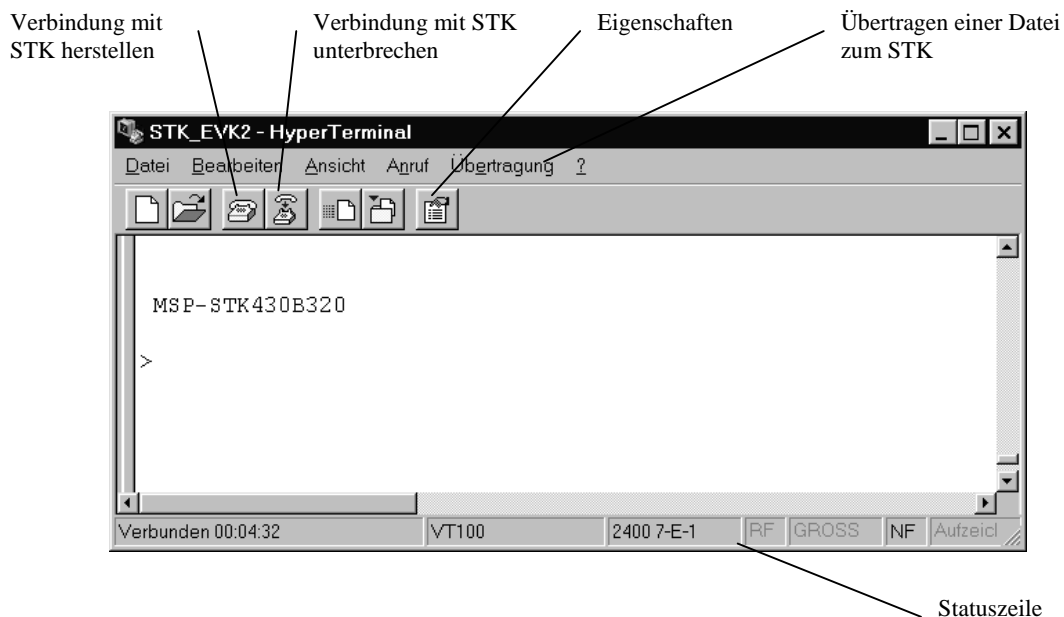
5.3 STK430-Terminal [SKM,1-18]

Das STK430-Terminal dient zur Kommunikation zwischen STK und PC. Dem Terminal-Symbol wird bei dem Betriebssystem Windows 3.1x das Programm Terminal.exe und bei Windows 95/98/NT das Programm Hypertrm.exe zugeordnet. Das Prinzip ist jedoch in beiden Fällen gleich, der Unterschied liegt nur in Bedieneroberflächen.

Standardmäßig können Sie mit Hilfe des HyperTerminals - in Verbindung mit einem Modem – Kontakt zur Außenwelt aufnehmen und Daten mit anderen Computern austauschen. In erster Linie wird der HyperTerminal eingesetzt, um mit Mailboxen und Datennetzen zu kommunizieren. Bei der STK handelt es sich auch um einen Computer und wir verschicken Textdateien.

Erfahrungsgemäß (gemessen an den Erfahrungen des Verfassers) bringt die korrekte Einstellung und Nutzung des Terminals viele unerwartete Probleme mit sich, besonders für diese Benutzer, die nur wenig Wissen über HyperTerminal oder Schnittstellen besitzen. Es kostet viel Zeit und ist sehr frustrierend, bis die Verbindung zwischen STK und PC problemlos funktioniert, falls sie direkt nach der Installation nicht korrekt funktionierte. Meist liegt es aber am schlampigen Umgang mit der COM-Nummer. Bei einer korrekten Installation geht alles „wie von selbst“.

5.3.1 HyperTerminal-Fenster

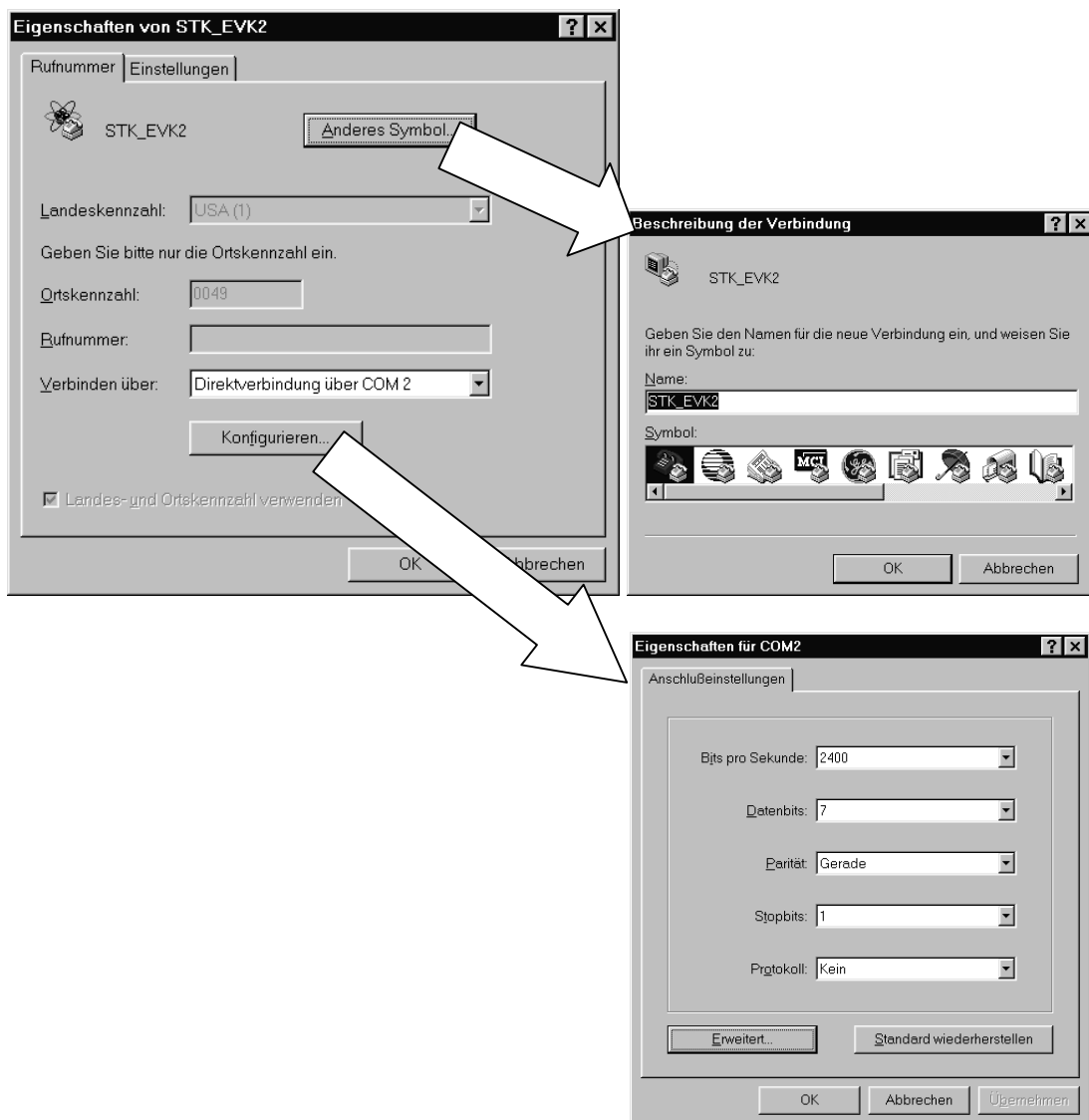


Die für unsere Zwecke wichtigsten Funktionen wurden oben bezeichnet. In der Statuszeile findet der Benutzer u.a. Informationen über die Dauer der Verbindung, der Emulation und der Einstellungen der Schnittstelle.

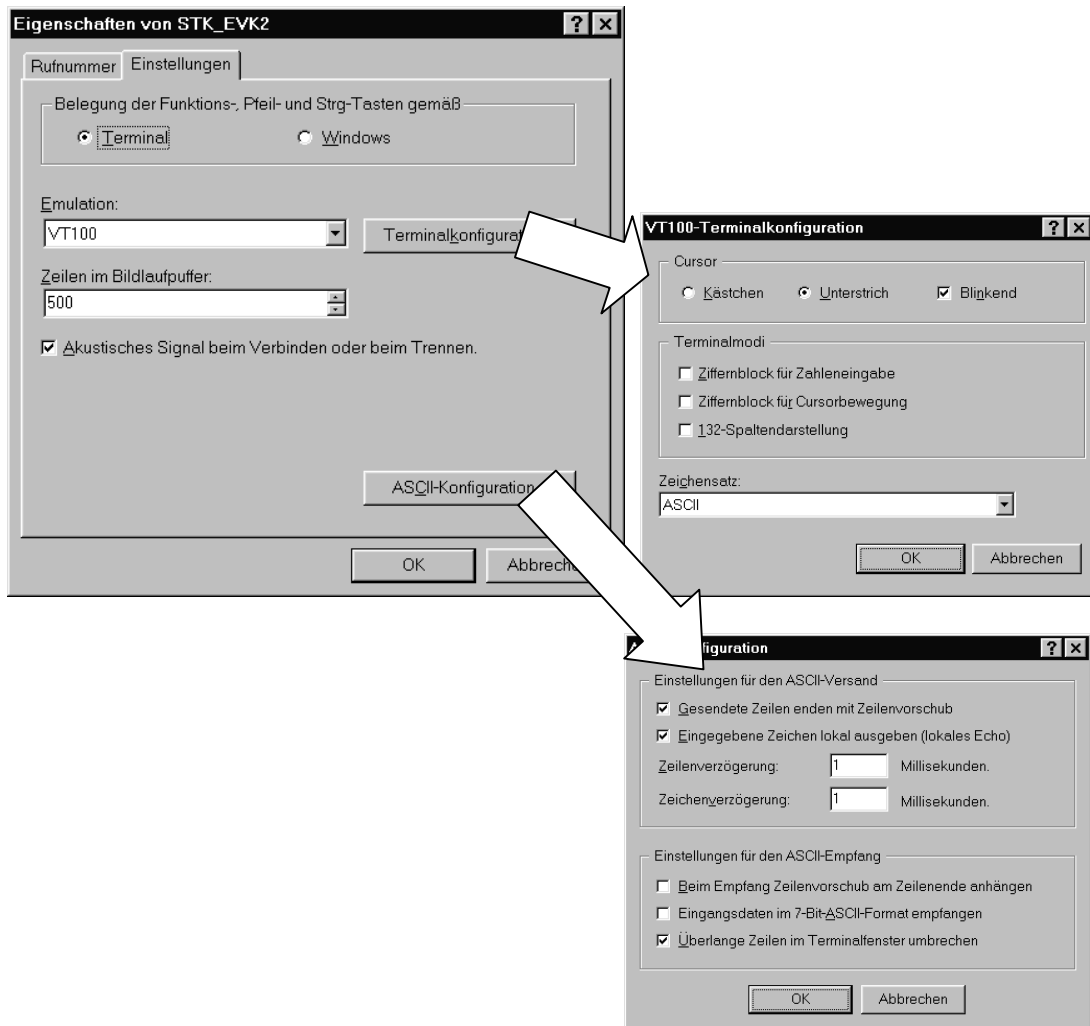
5.3.2 Übersicht über die Einstellungen

Zu den für ordnungsgemäßes Funktionieren der Verbindung wichtigen Einstellungen gehören die Konfiguration der Schnittstelle und die Emulation (It. Lexikon: Emulator – Computerprogramm zur Anpassung eines Computers an anders genormte Peripheriegeräte oder an Programme, die z.B. mit einem anderen Prozessor erarbeitet wurden). Diese sorgen nämlich für einen geordneten Austausch der Bits und Bytes, ohne das es zu Staus und Missverständnissen beim Senden und Empfangen kommt. Die meisten anderen Einstellungen sind nur für das äußere Erscheinungsbild des Terminals wichtig und werden hier nicht beschrieben. Hier die Fenster mit den Einstellungen:

- Schnittstelle



- Terminal



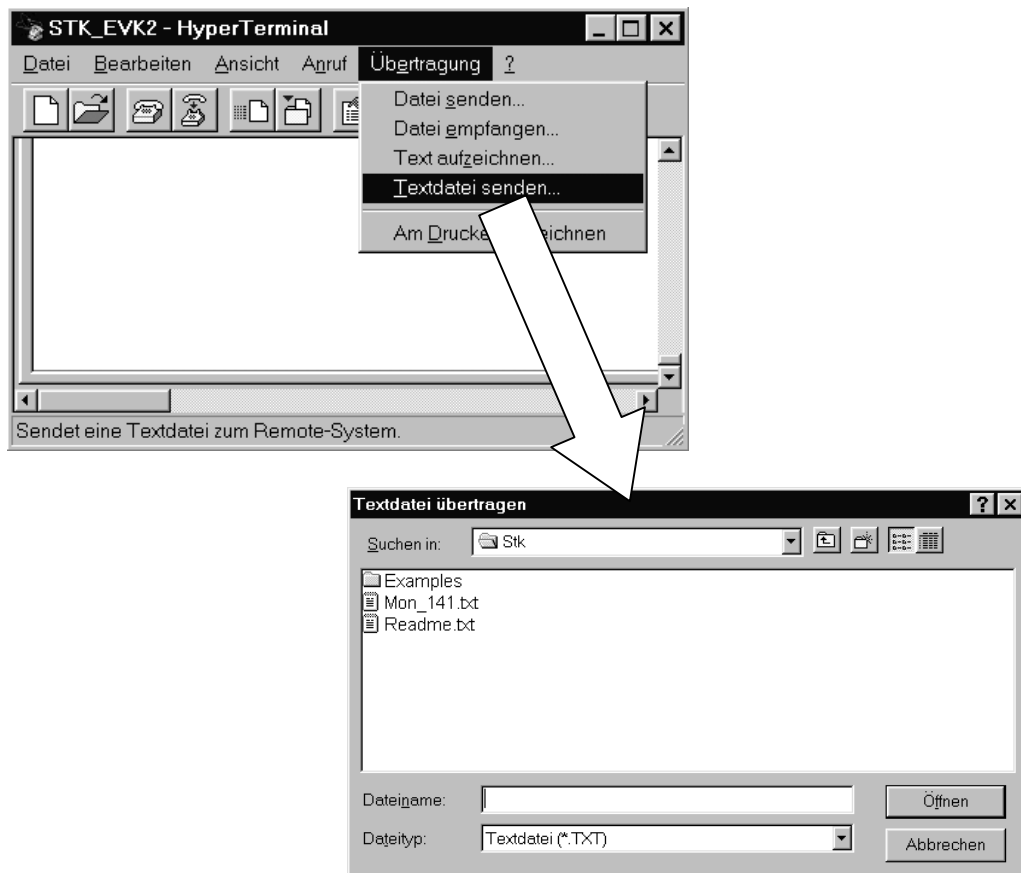
Bei Problemen mit der Verbindung überprüfen Sie:

- die Verbindung zwischen STK und PC.
- stimmen der im Eigenschaften/Rufnummer/Verbinden über: angegebene Port (COM1, COM2, COM3 bzw. COM4) und die Einsteckstelle des STK überein.
- die Konfiguration der Schnittstelle Eigenschaften/Rufnummer/Konfigurieren/Anschlußeinstellungen
 - **Bits pro Sekunde: 2400,**
 - **Datenbits: 7,**
 - **Parität: gerade,**
 - **Stopbits: 1,**
 - **Protokoll: kein**
- die Emulation Eigenschaften/Einstellungen/Emulation: **VT100**
- ggf.: unterbrechen Sie die Verbindung und drücken den Resetschalter des STK. Danach versuchen Sie erneut, die Verbindung herzustellen.

5.3.3 Senden der .txt-Datei an die STK [SKM,1-8]

Jedes Programm, das von uns entwickelt und editiert wurde, muß mit Hilfe des ASM430-Programms, übersetzt werden. Die nach der Übersetzung entstandene Datei mit dem Suffix .txt wird über das Terminal zu dem STK gesendet. Hier die Vorgehensweise:

1. Terminalprogramm starten.
2. Verbindung mit dem STK herstellen (bei einer korrekten Einstellung des Terminals geschieht es automatisch).
3. In der Menüzeile Übertragung/Textdatei senden... auswählen, es öffnet sich folgendes Fenster:



4. Den Dateinamen evtl. auch Verzeichnis anklicken oder eintragen und mit Öffnen bestätigen.
5. Im Terminalfenster werden die übertragene Bits und Bytes in Form von HEX-Zahlen angezeigt [SKM,1-20].
6. Nach dem Ende der Übertragung, drücken Sie g (für go, siehe Kap. 5.3.4), um das Programm zu starten [SKM,1-21].

5.3.4 Der EPROM-Monitor [SKM,2-1]

Der EPROM-Monitor ermöglicht dem Benutzer das Steuern und Überwachen der Programmausführung im STK mit Hilfe des PCs. Die Funktionen sind sehr ähnlich denen im Simulator, z.B. g (für go) entspricht Run.

Hier die Aufstellung der Steuertasten und deren Bedeutung.

- b,w** Anzeigen des Inhalts von Speicher oder Registern in Byte- bzw. Wordmodus [SKM,2-3]
- i** Zurücksetzen, initialisieren des STK zum Monitorprogramm [SKM,2-4]
- u** Zurücksetzen, initialisieren des STK zum Benutzerprogramm (User) [SKM,2-5]
- r** Anzeigen der Registerinhalte aller 16 Register [SKM,2-6]
- r[n]** Anzeigen des Registerinhalts des n-ten Registers und ggf. Überschreiben mit neuen Inhalt [SKM,2-6]
- m [n]** Anzeigen des Speicherinhalts ab der absoluten Adresse x und überschreiben mit neuen Inhalt. Optional [n] – Anzahl der angezeigten Stellen ab x. Um den Speicherinhalt zu ändern, einfach den neuen Inhalt schreiben und mit ENTER bestätigen [SKM,2-11]
- @x** Laden des Programms in den RAM/EPROM-Speicher Byte für Byte, ab der Adresse x bis q [SKM,2-14]
- ex n** Löschen des RAM-Speichers ab der Adresse x. Der Speicher wird mit FF überschrieben [SKM,2-15]
- s** Anzeigen/setzen der Haltepunkte [SKM,2-16,2-22]
- c** Löschen der Haltepunkte [SKM,2-18]
- g** Starten des Programms [SKM,2-19]
- q** Ende des zu ladenden Programms (siehe @x)
- d** Starten des Demoprogramms [SKM,1-12]
- h** Anzeigen der Monitor Steuerbefehle (Hilfe) [SKM,2-2]
- <space>** Ausführen eines Schrittes
- <esc>** Beenden eines Befehls

5.3.5 Kurzes Beispiel zum Assembler, Terminal und EPROM-Monitor

Am Beispiel des Programms EVMA320, das Sie in dem Verzeichnis `c:\adt430\dt430\examples` finden, werden wir den Weg verfolgen von einem editiertem (mit Editor geschriebenen) Programm (`evma320.asm`), über den Assembler, wo das Programm in eine `.txt`-Datei übersetzt wird, bis hin zum STK, wohin wir das Programm mit Hilfe des Terminals übersenden und seine Ausführung überwachen werden.

Hier die Vorgehensweise:

1. Überprüfen Sie ob in der Datei `evma320.asm` in der ersten Zeile des Programms: `SIM .set 0` steht. Falls nicht (`SIM .set 1`), ändern Sie dies und speichern das Programm: File/Save.
Sie können dies mit Hilfe des Editors, der im Simulation Environment Programm integriert ist, tun. Es ist nicht erforderlich ein neues Projekt zu öffnen; öffnen Sie lediglich die `evma320.asm`-Datei.
2. Starten Sie das ASM430-Programm in der ADT430-Gruppe [SKM,1-15].
3. Geben Sie `c:\adt430\dt430\examples\emva320` ein und bestätigen Sie mit Enter.
4. Schließen Sie das ASM430-Programm.
5. Starten Sie das STK430-Terminal in der ADT430-Gruppe [SKM,1-19].
6. Wählen Sie in der Menüzeile Übertragung/Textdatei senden... und anschließend in dem Dialogfenster Textdatei übertragen die `evma320.txt`-Datei aus dem Verzeichnis `c:\adt430\dt430\examples` aus. Bestätigen Sie mit Öffnen.
7. Nachdem die Datei in den Speicher des Prozessors übersendet wurde (q am Ende der Hexzahlen-Kolonen), starten Sie das Programm durch Drücken von g.
8. Auf dem LCD-Display des STK sehen Sie, wie der Text MSP430 nach links wandert.
9. Setzen Sie einen Haltepunkt (Breakpoint, BP) [SKM,1-22] im Programm auf der Adresse 300h. Um dies zu erreichen, geben Sie über die Tastatur ein x (das Programm wird gestoppt) und nachfolgend ein s (Haltepunkt setzen) ein. Nachdem auf dem Monitor folgende Meldung kommt: `set Bkpt 0000 0000`, tippen Sie 300 (Adresse des Haltepunkts im Programmspeicher - Zeile 187 im Programm) ein und bestätigen mit Enter.
10. Starten Sie das Programm wieder durch Drücken von g.
11. Das Programm hält jetzt jedes Mal an, wenn die Adresse 300h (auch im PC-Register sichtbar) im Programmspeicher erreicht wird. Auf dem Monitor erscheint gleichzeitig folgende Meldung: `BP halted` und der Inhalt aller Register wird angezeigt. Durch Drücken von g wird das Programm wieder gestartet, dann an der Adresse 300h angehalten usw., usw..
12. Als Übung: drücken Sie mehrmals die `<space>`-Taste (ausführen eines Schrittes) und beobachten das PC-Register und die LCD-Anzeige.

Wie Sie sehen können funktioniert unser Programm (EVMA320) im STK wie auch im Simulator gleich. Auch die Haltepunkte im Programm (Breakpoints) sind identisch.

Jetzt fragen Sie sich sicherlich: „Woher soll ich wissen, daß die Zeile 187 im Programmtext der Adresse 300h im Speicher entspricht?“

Die Antwort: „Schauen Sie sich die `evma320.lst`-Datei an, dort ist die Abhängigkeit zwischen der Adresse im Speicher und dem Programmtext sehr gut sichtbar“.

Genauer dazu im Kap. 8.1

6. Details des Prozessors

Um die Beispielprogramme zu verstehen und künftig auch eigene Programme zu schreiben, ist es notwendig, den Aufbau und die Funktionsweise des Prozessors zu begreifen. Zu den Aufgaben dieser (wohlgerne) „Einführung“ gehört nicht ein detailliertes Beschreiben der einzelnen Teile des Prozessors und deren Zusammenspiels. Es wird an dieser Stelle lediglich oberflächlich beschrieben, welche Teile es gibt, wie sie gesteuert und angesprochen werden.

Als erstes wird jedoch eine Funktion beschrieben, die die meisten Prozessoren besitzen, die aber nicht jedem geläufig ist.

6.1 Das Wesen des Interrupts [SKM,1-24,4-1] [DB,3-3]

Es handelt sich hier nicht um ein Dinosaurier oder eine andere uns unbekanntes Spezies. Die deutsche Bezeichnung „Unterbrechung“ beschreibt diese Funktion des Prozessors etwas deutlicher, es ist eine Unterbrechung des aktuellen Programmablaufs. Der Prozessor unterbricht das aktuelle Programm, um eine Programmroutine auszuführen, die eindeutig mit der Quelle der Unterbrechung verknüpft ist. Zu den Quellen von Unterbrechungen gehören viele Peripheriegeräte. Unser Prozessor ist mit vielen Peripheriegeräten ausgestattet, die Frage lautet also: „Wie soll er es merken, wenn eine seiner Peripherien sich meldet?“. Soll er, im Laufe des Programms, permanent die Peripheriegeräte fragen: „Ist was?, willst du was?“. Es wäre sicherlich machbar (programmierbar), es würde aber viel von der Kapazität des Prozessors verbrauchen und das eigentliche Programm würde zu eine Nebensache verkümmern.

Wie funktioniert es also in unserem Prozessor?

Jeder der möglichen Quellen für eine Unterbrechung wurden jeweils ein Unterbrechungsfreigabe-Bit (Interrupt-Freigabe-Bit oder IE-Bit (Interrupt Enable)) und ein Unterbrechungs-Anforderung-Bit (Interrupt-Merker-Bit oder IFG-Bit (Interrupt Flag)) zugeordnet. In dem SR-Register des MSP430 wurde das vierte Bit (GIE) zu einer zentralen Freigabestelle für alle Programmunterbrechungen erklärt, die von den Peripheriegeräten kommen. Gibt es also einen Grund, das Programm anzuhalten, z.B. weil der P0.0-Schalter gedrückt wurde, wird das IFG-Bit des betroffenen Peripheriegerätes gesetzt, z.B. POIFG.0 (im SFR-Register). War auch gleichzeitig das GIE-Bit gesetzt ($GIE = 1$), wird die Programmausführung unterbrochen. Der Prozessor lädt den Interrupt-Vektor der Unterbrechung (bei mehreren, die gleichzeitig anstehen, den höchstwertigen) [DB,3-4], speichert die aktuellen Stände des Programmzählers und des Statusregisters ab und springt zur Routine, auf die der Interrupt-Vektor zeigt, die sogenannte Interrupt-Service-Routine, z.B. die POISR-Routine.

Jede Quelle, die eine Unterbrechung verursachen soll, muß freigegeben werden. Dies geschieht durch Setzen des dieser Quelle zugehörigen Bits in dem IE-Register, z.B. POIE.0. Wurde ein IE-Bit für eine mögliche Quelle nicht gesetzt, reagiert der Prozessor auf diese Unterbrechungsanforderung nicht. Wurde das GIE-Bit zurückgesetzt ($DINT$ oder $BIC \#8, SR$), werden alle maskierbaren Unterbrechungsanforderungen ignoriert.

Im Laufe des Programms können die IE-Bits geändert werden. Die IFG-Bits sollen nach jeder Unterbrechungsanforderung gelöscht werden, am besten direkt in der Unterbrechungsroutine. Siehe dazu auch Kap. 7.2.3.

6.2 Die wesentlichen Bestandteile des Prozessors

6.2.1 CPU [DB,5-3]

CPU-central processing unit, dt. Zentraleinheit - verarbeitet die Befehle

6.2.2 Register [DB,5-3]

Interner Speicher der CPU, 16 Word (16*16-Bit). Die Register werden im Programm mit R0 bis R15 bezeichnet bzw. PC für R0, SP für R1, SR oder CG1 für R2 und CG2 für R3. Die ersten vier Register dienen speziell dem Programmablauf und sollen sachgerecht verwendet werden. R4 wird von der STK-PC-Schnittstelle benutzt und soll nicht verwendet werden. R5 bis R15 stehen dem Programmierer voll und ganz zu Verfügung, und sollen so oft wie möglich im Programm verwendet werden; dies erhöht die Leistung eines Programms, da die Zugriffszeit auf die Register viel kürzer ist als auf den Speicher.


Die Abkürzungen, die im Zusammenhang mit den Register [DB,5-6] stehen und deren Lage im Register:

Bezeichnung im Programm		Bit															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PC	R0																
SP	R1																
CG1+SR	R2							V	SCG 1	SCG 0	Osc Off	CPU Off	GIE	N	Z	C	
CG2	R3																
	R4																
																
																
	R15																

- **PC**-Program Counter – Programmzähler, beinhaltet die Adresse des nächsten Programmbefehls
- **SP**-Stack Pointer – Stapelzeiger, beinhaltet (zeigt auf) die aktuelle Adresse auf dem Stapel
- **CG1, CG2** – hier können die häufig im Programm benutzten Konstanten erzeugt werden [SUG,A-6]
- **SR**-Status Register – Flagregister – die Flags (einzelne Bits) sind die Zustandsmerker der CPU, sie geben Auskunft über aktuellen Status des Prozessors und die Ergebnisse der letzten Operationen (Berechnungen, Vergleiche u. ä.) (V, N, Z, C), dienen aber auch der Steuerung des Prozessors (SCG1, SCG0, OscOff, CPUOff) und des Programmablaufs (GIE). Das Statusregister ist nur in der Adressierungsart „Register“ erreichbar, in den anderen Adressierungsarten wirkt es als CG1.

Die Bedeutung der Flags des Statusregisters [DB,5-7]:

Bezeichnung des Bits	Funktion	Bit = 0, nicht gesetzt	Bit = 1, gesetzt
C	Übertrag	Operation erzeugte keinen Übertrag	Operation erzeugte einen Übertrag
Z	Null (Zero)	Ergebnis ungleich Null	Ergebnis gleich Null
N	Negativ (Negative)	Ergebnis positiv	Ergebnis negativ
GIE	Generelle Unterbrechungsfreigabe	alle maskierbaren Interrupts werden ignoriert	alle maskierbaren Interrupts werden angenommen
CPUOff	CPU-Abschaltung	CPU aktiv	CPU deaktiviert, Ende aller Aktivitäten
OscOff	Oszillator-Abschaltung	Oszillator aktiv	Oszillator deaktiviert, Ende aller Aktivitäten
SCG0	Systemtakt-Steuerung	siehe Skript: Betriebsarten	
SCG1			
V	Überlauf	kein Überlauf	Überlauf

	Die Bits C Z N V werden nicht von jedem Befehl beeinflusst (siehe SUG).
---	--

6.2.3 Speicher [DB,4-1] [DB,A-1]

Beinhaltet die Spezialregister, die Register der Peripheriemodule, das Programm selbst, den Stapelspeicher und die Unterbrechungsvektorentabelle (Interruptvektorentabelle). Der Adreßraum liegt zwischen 00000h (Null) und 0FFFFh (65.535), der beschreibbare Speicher in dem STK endet für uns an der Adresse 003FFh. Jede Adresse beinhaltet ein Byte (8 Bit) an Speicherkapazität. Zwei Byte bilden ein Word (16 Bit), der immer an einer geraden Adresse positioniert ist. Aufgrund der verschiedenen Zugriffsmöglichkeiten auf den Speicherinhalt (Byte- bzw. Wordzugriff) ist der Speicher in drei Bereiche aufgeteilt [DB,4-4]:

Zugriffsart	Bereich	
	Anfangsadresse	Endadresse
Byte-Zugriff	0000h	00FFh
Word-Zugriff	0100h	01FFh
beliebig	0200h	03FFh
beliebig	C000h	FFFFh

Die Steuerregister der Peripherien werden im nächsten Absatz beschrieben. Hier eine kurze Beschreibung des Spezial Funktion Registers und der Unterbrechungsvektorentabelle:

- **Spezial-Funktions-Register (SFR)** [DB,4-11][DB,A-2] – Adressen 00000h, 00001h, 00002h und 00003h, dient der Kontrolle der Interrupts, setzt sich zusammen aus: IE- und IFG-Register. Im IE-Register werden die einzelnen Interrupts freigegeben, im IFG-Register werden die Interrupts gemeldet. *Vorstellung:* Schalttafel mit Ein/Aus-Schaltern (IE) und Meldeleuchten (IFG).
- **Unterbrechungsvektorentabelle/Interruptvektorentabelle** – die letzten 16 Word-Adressen im jeweiligen Speicher. In diesen Speicherplätzen sind die Anfangsadressen der entsprechenden Unterbrechungsroutinen gespeichert.

Die Anordnung der Vektoren [DB,3-13]:

Adresse im STK-Speicher	Interrupt Quelle	Zuständiges Flag	Priorität (Wichtigkeit, Vorrang)
3FEh (letzte RAM-Adresse im STK-Speicher)	Spannungsabfall, Reset, Watchdog	RSTI, WDI	15 (höchste)
3FCh	Oszillator fehler	NMIFG, OFIFG	14
3FAh	P0-Schalter	P0.0IFG	13
3F8h	P1 serielle Schnittstelle	P0.1IFG	12
3F6h			11
3F4h	Watchdogtimer	WDTIFG	10
3F2h			9
3F0h			8
3EEh			7
3ECh			6
3EAh	Analog-to-Digital	ADIFG	5
3E8h	Timer Port	im Modul	4
3E6h			3
3E4h			2
3E2h	Basic Timer	BTIFG	1
3E0h	I/O-Port	P0.2..P0.7IFG	0

Eigentlich liegt die Interruptvektorentabelle am oberen Ende des ROM. Durch das Monitor-Programm des STK werden die oberen RAM-Adressen (3E0h bis 3FEh) für Interruptvektorentabelle benutzt, damit man sie programmieren kann.

Weitere Details über die Lage der Statusregister und der einzelnen Bits im Speicher entnehmen Sie bitte dem Anhang.

6.2.4 Peripherie

Peripherien sind alle im Prozessor integrierte Module, die unabhängig voneinander und von der CPU ihre Funktionen ausführen können. Jedem Peripheriemodul sind Register im Speicher zugeordnet. Die Steuerbits, die sich im Steuerregister eines Moduls befinden, ermöglichen es dem Programmierer, den aktuellen Zustand zu erkennen und das Modul zu steuern. Es existieren zwei Gruppen von Peripheriemodulen: 8-Bit- und 16-Bit-Module, sie unterscheiden sich durch die Breite des Datenbusses, über den sie die Daten transferieren. Dieser Unterschied soll beim Programmieren besonders beachtet werden (Byte-, Wordzugriff).

Hier eine kurze Beschreibung der einigen Peripherien:

- **Digital I/O** [DB,8-1][DB,A-3][SUG,6-3]– der Eingang/Ausgang-Port besteht in der MSP430-Prozessor-familie aus maximal fünf 8-Bit-Ports (P0, P1, P2, P3, P4). Bei unserem Prozessor MSP430P325 ist davon nur P0 realisiert, d.h. es stehen uns 8 Pins (Ein/Ausgänge) zur Verfügung (P0.0 bis P0.7). Von jedem einzelnen Pin können über Steuerregister binäre Signale gelesen (PxIN.x) oder an diesen ausgegeben werden (PxOUT.x). Nur die ersten drei Ports (P0, P1, P2) haben auch die Interrupt-Steuerregister (PxIFG, PxIE, PxIES), das bedeutet, diese werden als mögliche Quellen für Unterbrechungen erkannt. Die Bedeutung der einzelnen Steuerregister:

Steuerregister	Funktion	Bit = 0, nicht gesetzt	Bit = 1, gesetzt
PxIN	Eingang	ein Low-Signal steht am Eingang	ein High-Signal steht am Eingang
PxOUT	Ausgang	Low-Signal am Ausgang ausgeben	High-Signal am Ausgang ausgeben
PxDIR	Richtung	Pin als Eingang	Pin als Ausgang
PxIFG	Interruptmerker	kein Interrupt	Interrupt
PxIE	Interruptfreigabe	Pin als Interruptquelle nicht zugelassen	Pin als Interruptquelle zugelassen
PxIES	Flankenauswahl	Interrupt bei Low/High-Übergang	Interrupt bei High/Low-Übergang
PxSEL (nicht bei P0)	Modulwahl	Port selektiert	Modul selektiert

- **LCD-Anzeige** [DB,14-1][DB,A-5][SUG,8-1][SKM,1-9] – Der Prozessor kann maximal 15 Zeichen zu je acht Segmenten bei LCD-Anzeigen bzw. 30 Digitalausgänge oder einer Mischung daraus steuern [DB,14-13,14-24]. Beim STK besteht die Hardware aus 8 Ziffern, die von dem Prozessor gesteuert werden. Das Steuerregister LCDCTL enthält die Steuerbits LCDM0 bis LCDM7, die festlegen, wieviele Ausgänge für LCD verwendet werden und wieviele als Digitalausgänge. Die entsprechenden Daten werden in den 15 LCD-Daten-Registern [DB,8-22,14-12] gespeichert, die auch die aktiven Segmente der einzelnen Digits (Ziffern in der Anzeige) enthalten.
- **Basic Timer** [DB,A-6] – bildet eine niedrige Frequenz, für einige Anwendungen und Peripheriegeräte. Diese entsteht durch Teilen der Grundfrequenz MCLK oder des Uhrenquarzes ACLK (32768 Hz).
- **System Clock Generator** [DB,A-7] – bildet die Taktfrequenz, mit der das System arbeitet, diese entspricht einem Vielfachen der Frequenz des Uhrenquarzes, das in dem SCFQCTL-Register eingestellt wird.
- **A/D-Wandler** [DB,15-1][DB,A-3][SUG,9-1] – wandelt die analogen Signale, die an den Eingängen A0 bis A7 (AIN.0..AIN.7) anstehen, in digitale (12+2)-stellige Werte, die anschließend in dem ADAT-Register gespeichert werden. Die Eingänge können analog (AEN.x-Bit = 0) oder digital (AEN.x-Bit = 1) betrieben werden. Die Bits des Steuerregisters (ACTL) [DB,15-15] des A/D-Wandlers steuern folgende Funktionen (nächste Seite):

Bit	Bezeichnung	Funktion
0	ACTL.0	Starten der Umwandlung
1	ACTL.1	Ein/Ausschalten der internen Referenzspannung
2	ACTL.2	Auswahl des Eingangs (A0..A7)
3	ACTL.3	
4	ACTL.4	
5	ACTL.5	
6	ACTL.6	
7	ACTL.7	Auswahl des Ausganges (A0..A3) für den Quellenstrom
8	ACTL.8	
9	ACTL.9	
10	ACTL.10	Auswahl des Bereichs
11	ACTL.11	
12	ACTL.12	
13	ACTL.13	Ausschalten des A/D-Wandlers (Bit = 1) und der int. Referenzspannung
14	ACTL.14	
15	ACTL.15	
		0 (Reserviert)

- **Watchdog** [DB,A-10][SUG,7-3] – eine äußerst nützliche und energiesparende Funktion des MSP430-Prozessors. Die primäre Aufgabe des Watchdog-Moduls besteht darin, einen kontrollierten Neustart (Reset) des gesamten Systems durchzuführen, nachdem ein Softwareproblem aufgetreten ist. Dies geschieht erst, wenn die eingestellte Zeit abgelaufen ist.

Falls diese Funktion nicht im Programm genutzt wird, kann der Watchdog auch im Timer-Modus arbeiten und nach dem Ablauf der eingestellten Zeit eine Programmunterbrechung (Interrupt) verursachen.

Die Funktionen des Watchdog-Steuerregisters (WDTCTL):

Bit	Bezeichnung	Funktion
0	IS0	Auswahl der Zeit
1	IS1	
2	SSEL	
3	CNTCL	Start der Zeit
4	TMSEL	Modusauswahl: Watchdog (Bit = 0) bzw. Interval timer (Bit = 1)
5	NMI	Auswahl der Funktion des RST/MNI-Pins
6	NMIES	Flankenwahl für NMI
7	HOLD	Watchdog einschalten (Bit = 0) bzw. ausschalten (Bit = 1)
8..15		Paßwort

7. Details eines Assemblerprogramms

Die Programmausschnitte (Beispiele), die Sie in diesem Kapitel finden, stammen aus der Datei evma320.lst .

7.1 Die wesentlichen Bestandteile eines Programms [GS,4-3]

Jedes Programm das von einem Programmierer geschrieben wird muß eine feste Struktur haben, diese ist für die Übersetzungsprogramme genauso wichtig, wie für den Programmierer selbst und alle anderen, die dieses Programm später nachvollziehen müssen und evtl. Änderungen vornehmen wollen.

7.1.1 Kommentare [LUG,3-11]

Die Kommentare ziehen sich durch das ganze Programm und haben keinen Einfluß auf seinen Ablauf. Sie erläutern das Geschehen im Programm. Die Texte hinter dem Semikolon (;) bis zum Ende der Zeile bilden Kommentar.

Eine gute Nachricht für alle, die Beispielprogramme bestehen im größten Teil aus Kommentaren, die meistens auf der rechten Seite stehen.

Ein gute Rat: *gewöhnen Sie sich von Anfang an an das Schreiben der Kommentare.* Ausreichend kommentierte Programme sparen später (spätestens nach eine Woche) eine Menge Zeit und Ärger (bei Dechiffrieren und „Debuggen“).

Ein Tip:

- Kommentar zur Kennzeichnung eines neuen Abschnitts beginnt ganz links,
- Kommentar zur Beschreibung der aktuellen Aktion steht rechts

Beispiel:

```
24          ;--- RAM allocation
25
26 0220      .bss      runoff,1,220h  ; runoff control flag register
27 0221      .bss      lcd_timer,1    ; lcd interval timer
28 0222      .bss      txt_ori,2      ; pointer for LCD text tables origin
29 0224      .bss      txt_idx,2      ; pointer for LCD text tables index
```

7.1.2 Definitionen [LUG,3-12,4-8,4-20]

Die Initialisierungen stehen vor dem Anfang des Hauptprogramms. In diesem Teil des Programms werden:

- die Adressen einiger Register

Beispiel:

```
34          01          IE2          .equ      01h
35          02          IFG1         .equ      02h
74          0120        WDTCTL       .equ      0120h
```

- besonders wichtige Bits

Beispiel:

```
75          80          WDTHold     .equ      80h
78          08          GIE          .equ      08h
```

- oder Bitkombinationen

Beispiel:

```
76          5a00        WDT_wrkey  .equ      05A00h
```

mit Alias-Namen versehen.

7.1.3 Initialisierungen [LUG,3-12,4-8,4-20] [SUG,3-3,3-9]

Ein Teil der Register, Steuerregister und des Speichers muß eingestellt (mit Werten belegt) werden, bevor das eigentliche Programm ausgeführt wird. Z.B. damit der Prozessor das Programm richtig ausführt, muß er wissen:

- wo die Befehle anfangen, d.h. Programmzähler initialisieren

Beispiel:

```
81                ;*****
82                ; Reset : Initialize processor
83                ;*****
84
85 0240            .sect "MAIN",RAM_orig
```

- wo er eventuell Daten ablegen kann, d.h. Stapelzeiger initialisieren

Beispiel:

```
86 0240            RESET
87 0240 403103de  MOV  #SP_orig,SP      ; initialize stackpointer
```

Es werden auch, abhängig vom Programm:

- Register und Steuerregister gelöscht

Beispiel:

```
93 024a 42f20000  MOV.B #08h,IE1      ; ! Monitor !
94 024e 43c20001  CLR.B IE2
95 0252 43c20002  CLR.B IFG1
96 0256 43c20003  CLR.B IFG2
```

- bzw. gesetzt

Beispiel:

```
88 0244 40b25a800120  MOV  #(WDTHold+WDT_wrkey), &WDTCTL
                                ; Stop Watchdog Timer
108 0264 d0f200800005  BIS.B #BTME,ME2 ; Enable basic timer module
110 0270 d0f200800001  BIS.B #BTIE,IE2 ; enable basic timer intrpt
108 0264 d0f200800005  BIS.B #BTME,ME2 ; Enable basic timer module
```

7.1.4 Hauptprogramm [GS,4-3]

Wichtigster Teil eines Programms, hier geschieht das eigentliche Programm. Es besteht aus:

- (vielen) Befehlen
- einigen Sprungmarken (z.B. Schleife) [LUG,3-15] `idle`
- und Routineaufrufen `CALL loesch_LCD, CALL #init_txt_MSP.`

Beispiel:

```
115                ;*****
116                ; Mainloop
117                ;*****
118
119 0280            mainloop
120 0280 +12b002a4  CALL  #init_txt_MSP ; initialize shift text
121
122                ;--- idle loop : check runoff control bits
123
124 0284            idle
125 0284 d232            EINT
126 0286 -b3d0ff98     BIT.B #lcd_req,runoff ; time slice over ?
127 028a +27fc        JZ    idle
128 028c -c3d0ff92     BIC.B #lcd_req,runoff
                                ; yes : clear control bit and execute
129 0290 +12b002b0     CALL  #shift_txt ; shift text 1 digit left
130 0294 +3ff7        JMP   idle
```

7.1.5 Routinen/Unterprogramme [SUG,3-9]

Die Routinen sind, im Grunde, ein Bestandteil des sie aufrufenden Programms, ggf. des Hauptprogramms, und stehen direkt unter dem Hauptteil. Eine Routine fängt immer mit dem Namen an, was nichts anderes als eine Sprungmarke im Programm ist und endet mit dem Befehl RET (Rücksprung in das aufrufende Programm, ggf. das Hauptprogramm). In der Routine findet man die gleichen Programmelemente wie im Hauptprogramm.

Beispiel:

```

167                               ;--- clear LCD
168
169 02e2          show_clr
170 02e2  4035000f  MOV    #15, r5 ; clear display memory
171 02e6          show_clr1
172 02e6  43c50030          MOV.b #0, LCD1-1(r5)
173 02ea  8315          DEC    r5
174 02ec  +23fc          JNZ    show_clr1
175 02ee  4130          RET

```

7.1.6 Interruptroutinen/Unterbrechungsroutinen [SUG,3-9][SKM,4-1][DB,3-3]

Es sind Routinen, die nur dann ausgeführt werden, wenn eine Interruptanforderung vorkommt. Die Interruptroutinen sind den Interruptanforderungen fest zugewiesen, d.h. einer Interruptquelle kann nur eine Interruptroutine zugewiesen werden. Die Verknüpfungsstelle zwischen der Interruptquelle und dem Interrupt bildet die Vektorentabelle. Interruptroutinen enden immer mit dem Befehl RETI (Rücksprung in das unterbrochene Programm). In der Routine findet man die gleichen Programmelemente wie im Hauptprogramm, also auch Unterprogramme.

Beispiel:

```

181 02f0          Int_BT          ; Basic Timer 128Hz (7.8ms)
182 02f0  -83d0ff2f  DEC.B lcd_timer ; decrement SW lcd-timer
183 02f4  +2005          JNZ    Int_BT_end ; !0 : no action
184 02f6  -40f00080ff27 MOV.B #lcd_ival,lcd_timer ;=0 : load again and
185 02fc  -d3d0ff22  BIS.B #lcd_req,runoff ;set lcd request
                                   ; control bit

186 0300          Int_BT_end
187 0300  1300          RETI

```

7.1.7 Tabellen [DB,5-9]

Auch Arrays bzw. Felder genannt, beinhalten Daten (Werte), auf die mit Hilfe eines Zeigers zugegriffen wird (LCD_Tab (R10)). Der Anfang der Tabelle ist mit einem Namen markiert (LCD_Tab).

Beispiel:

```

216 0304          LCD_Tab
217 0304  b7      .byte  a+b+c+d+e+f      ; displays "0"
218 0305  12      .byte  b+c              ; displays "1"
219 0306  8f      .byte  a+b+d+e+g        ; displays "2"
220 0307  1f      .byte  a+b+c+d+g        ; displays "3"
221 0308  3a      .byte  b+c+f+g          ; displays "4"
222 0309  3d      .byte  a+c+d+f+g        ; displays "5"
223 030a  bd      .byte  a+c+d+e+f+g      ; displays "6"
224 030b  13      .byte  a+b+c            ; displays "7"
225 030c  bf      .byte  a+b+c+d+e+f+g    ; displays "8"
226 030d  3f      .byte  a+b+c+d+f+g      ; displays "9"
227 030e  00      .byte  0
260 032f          LCD_Tab_End

```


7.2 Abarbeiten eines Programms durch den Prozessor

Wie wir in dem Kapitel 4 erfahren haben, wird das Programm nach dem Assemblieren über das Terminal-Programm in den Speicher des Prozessors geladen. Die Anfangsadresse des Programms in dem Prozessorspeicher ist für uns meist 240h.

Hier ein kurzer Überblick über die Vorgänge in einzelnen Programmteilen.

7.2.1 Hauptprogramm

Bei Programmstart wird der Programmzähler (PC) auf die 240h (`.sect „MAIN“,RAM_orig`) und der Stapelzeiger (SP) auf 03DEh (`MOV #SP_orig,SP`) und alle Register entsprechend den Vereinbarungen im Initialisierungsteil des Programms eingestellt. Danach wird das Hauptprogramm abgearbeitet, dabei wird immer beim Holen eines Befehlswortes der Programmzähler (PC) um 2 erhöht. *Wieso um 2?* : das Programm wird wortweise gespeichert d.h. ein Befehlswort benötigt 2 Byte (ein Word). Je nach Länge des Befehls wird PC also um 2 (Ein-Wort-Befehl), 4 (Zwei-Wort-Befehl) oder 6 (Drei-Wort-Befehl) erhöht. Nach der Inkrementierung wird der entsprechende Befehl ausgeführt, dann der Programmzähler erhöht, Befehl ausgeführt, Programmzähler erhöht, Befehl ausgeführt u.s.w., u.s.w..

7.2.2 Routinen

Die Routineaufrufe (`CALL #loesch_LCD`) bilden eine besondere Stelle im Programm, da hier in ein Unterprogramm gesprungen wird. Die Vorgänge bei einem Routineaufruf am Beispiel der lösche_LCD-Routine:

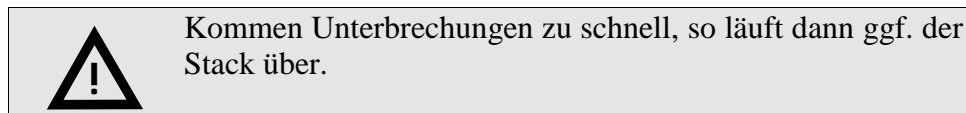
- `CALL #loesch_LCD` im Hauptprogramm
 1. Programmzähler wird erhöht.
 2. Der Inhalt des Programmzählers (PC) wird auf dem Stack gesichert (gespeichert).
 3. Die Adresse `#loesch_LCD` wird in den Programmzähler geschrieben.
 4. Die Befehle werden weiter ausgeführt, jetzt aber innerhalb der Routine.
 5. `RET`-Befehl am Ende der Routine. Der Programmzähler (PC) wird mit dem Inhalt der Spitze des Stacks (die Adresse des nächsten Befehls im Hauptprogramm) überschreiben.
 6. Weiter im Hauptprogramm: Befehl ausführen, Programmzähler erhöhen, usw., usw..

7.2.3 Interrupt/Unterbrechung

Ähnlich wie bei den Routinen ist eine Unterbrechung eine spezielle Stelle im Programm, da dieses Teil des Programms nur dann ausgeführt wird, wenn eine Unterbrechungsanforderung vorliegt. Hier die Vorgänge am Beispiel der P0.0-Unterbrechungsroutine:

- P0.0-Schalter wurde betätigt:
 1. Das `POIFG.0`-Bit im SFR wird gesetzt.
 2. Die Programmausführung wird unterbrochen.
 3. Der Programmzähler wird um 2 erhöht (zeigt jetzt auf den nächsten Befehl im Hauptprogramm).

4. Die Adresse aus dem Programmzähler (PC) wird auf dem Stapel gespeichert.
 5. Der Inhalt des Statusregisters (SR) wird auf dem Stapel gespeichert.
 6. Der Statusregister wird neu initialisiert.
 7. Die Anfangsadresse der Unterbrechungsroutine wird aus der Vektorentabelle in den Programmzähler (PC) kopiert.
 8. Die Unterbrechungsroutine POISR wird ausgeführt.
 9. RETI-Befehl in der Unterbrechungsroutine - beendet diese und kopiert zunächst Statusregisters (SR) und dann die Adresse des nächsten Befehls (im Hauptprogramm) vom Stapel in den Programmzähler (PC) (adversativ zu Punkt 4 und 5).
 10. Weiter im Hauptprogramm: der nächste Befehl wird ausgeführt, usw., usw..
- Jetzt die Vorgänge innerhalb der Unterbrechungsroutine (ISR):
1. Das GIE-Bit (Generelle Unterbrechungsfreigabe) im SR wird (durch Initialisierung des SR) gelöscht, damit während der Ausführung dieser Unterbrechungsroutine keine Unterbrechungen stattfinden. Man kann das GIE-Bit aber auch in der Unterbrechungsroutine (ISR) bei Bedarf wieder setzen, d.h. Unterbrechungen während der Ausführung der Unterbrechungsroutine werden zugelassen.



2. Die Befehle der Routine werden ausgeführt.
3. Innerhalb der Unterbrechungsroutine wird das IFG-Bit, das dieser Unterbrechungsroutine zugeordnet ist, gelöscht. Im Beispielprogramm WD_BSP wird brutal das ganze IFG1-Register gelöscht (CLR.B IFG1), in dem sich auch das POIFG.0-Bit befindetet, Grund: es wurden sonst keine anderen Unterbrechungen zugelassen. Würden mehrere Unterbrechungen zugelassen, muß man mit Bedacht vorgehen und die Bits einzeln löschen (BIC.B #POIFG0, IFG1 bzw. BIC.B #04h, IFG1).
4. Rücksprung in das Hauptprogramm (RETI) mit Rückladen von SR und PC.

8. Die Beispielprogramme

Die drei Beispielprogramme: WD_BSP, P0_0_BSP und AD_BSP sollen dem Leser einige Möglichkeiten, wie auch die Programmierung des MSP430-Prozessors zeigen. Vor allem jedoch finden Sie in den Programmen Beispiele, wie die Programmierung und der Umgang mit den Bestandteilen des Prozessors funktioniert, z.B. wie man auf der LCD-Anzeige über die LCD-Tabelle Zeichen generiert oder wie der Watchdog eingestellt wird.

Die Programme sind ausreichend kommentiert, so daß jeder Leser mit Grundkenntnissen im Assembler relativ leicht nachvollziehen kann, was in den einzelnen Zeilen passiert.

8.1 Die .lst-Dateien [LUG,3-23]

Im Anhang finden Sie den Ausdruck der Dateien WD_BSP.lst, P0_0_BSP.lst und AD_BSP.lst, die gemeinsam mit den .txt- und .obj-Dateien beim Assemblieren der .asm-Datei entstehen. Diese Dateien sind hauptsächlich als Stütze zur Analyse der Programme gedacht. Sie beinhalten zusätzlich, zu dem Quelltext des Programms, noch drei Spalten an der linken Seite.

In der ersten Spalte steht die Zeilennummer, die besonders wichtig für die Bezeichnung der Haltepunkte im Simulator sind.

In der zweiten Spalte stehen die Anfangsadressen der jeweiligen Befehle im Programmspeicher, wichtig beim Setzen der Haltepunkte im STK über EPROM-Monitor. Nun, schauen Sie sich die Adressen genau an, Sie erkennen, daß im jedem Programm der erste Befehl des Programms immer an der Adresse 240h steht und die Interruptvektorentabelle immer an der Adresse 3E0h anfängt (entsprechend den .sect-Anweisungen).

Die dritte Spalte zeigt den Befehl bzw. den definierten Wert in der hexadezimalen Form an. Die kürzesten Befehle, z.B. PUSH oder RETI bestehen aus 4 Hexzahlen, was einem Wort entspricht, und die längsten aus 12 Hexzahlen, was drei Wörtern entspricht. Gut sichtbar ist auch die Abhängigkeit zwischen der Interruptvektorentabelle und den Interruptroutinen, z.B. in dem Programm WD_BSP befindet sich in der Speicheradresse 3F4h (Interruptvektorentabelle) der Wert 2A2h, der nichts anderes als eine Adresse ist, und genau an dieser Adresse fängt die WDISR Interruptroutine an.



Die drei Beispielprogramme, die Sie als Quelltext im Anhang finden, stehen Ihnen als Dateien im Netz der FH (den Prof. bzw. Assistenten fragen) zur Verfügung.

8.2 Beschreibung der Beispielprogramme

WD_BSP

Dieses Programm führt die Funktionsweise des Watchdogs und der LCD-Anzeige vor. Der Watchdog arbeitet im Timer-Modus, d.h. nach Ablauf der eingestellten Zeit verursacht der Watchdog-Timer ein Interrupt und die WDISR-Interruptroutine wird ausgeführt. In der Zwischenzeit (zwischen den Interruptroutinenaufrufen) springt der Prozessor in einer endlosen Schleife und wartet, bis der nächste Interrupt kommt.

In der Interruptroutine wird eine Zahl, die sich im Register 11 befindet, bei jedem Aufruf der Interruptroutine auf der LCD-Anzeige angezeigt und anschließend um 1 erhöht.

Übung:

- Verändern Sie den Wert des Registers 12 (Position der Zahl auf der LCD-Anzeige), (Zeile 121) und beobachten die LCD-Anzeige. Welche Werte kommen in Frage - sind sinnvoll?
- Verändern Sie die Zeit zwischen den Interruptaufrufen (Zeile 116).

P0_0_BSP

Dieses Programm führt die Funktionsweise der LCD-Anzeige, des P0.0-Schalters und der Zeichenübergabe an den PC mit Hilfe des EPROM-Monitors vor.

Die Funktion des Programms ist einfach: die Betätigungen des P0.0-Schalters (auf dem STK) werden gezählt und deren Anzahl auf der LCD-Anzeige angezeigt. Die Stelle, an der die Zahl erscheint, wandert bei jeder Betätigung nach rechts. Der Zähler zählt von 0 bis 9, nachdem die 9 erreicht wird, wird der Zähler gelöscht (Null). Die gleiche Zahl, die auf der LCD-Anzeige erscheint, wird mit Hilfe der UART-Funktionen zum PC gesendet (RS-Routine) und auf dem Monitor des PC's angezeigt.

Das Hauptprogramm besteht aus den Initialisierungen und einer Warteschleife. Das Programm wartet auf einen Interrupt vom P0.0-Schalter; nachdem dieser gedrückt wurde, springt das Programm in die P0ISR-Interruptroutine und führt diese aus.

In der Interruptroutine wird der Wert des Zählers erhöht und die Stelle in der LCD-Anzeige geändert. Dies geschieht über Vergleiche und Sprungbefehle. Dann wird der Wert des Zählers, über eine sehr kurze Routine, auf der LCD-Anzeige ausgegeben. Anschließend wird der Wert des Zählers an PC gesendet (RS-Routine) und auf dem Monitor angezeigt. Als letztes wird eine Entprell-Routine aufgerufen, die eine Zwangspause im Programm verursacht, damit der Zeitabstand zwischen den Betätigungen des P0.0-Schalters größer wird.

AD_BSP

Dieses Programm führt die Funktionsweise des A/D-Wandlers vor. Als Signalquelle wird der auf dem STK integrierte Lichtsensor genutzt, der über die I/O-Ausgänge P0.5, P0.6 und P0.7 mit Strom gespeist wird. Die Spannung, die an dem Lichtsensor entsteht, wird an dem Eingang A3 des A/D-Wandlers gemessen und als dezimale Wert auf der LCD-Anzeige angezeigt. Nach der Betätigung des P0.0-Schalters wird der gleiche Spannungswert als hexadezimale Zahl angezeigt. Nochmalige Betätigung des P0.0-Schalters verursacht eine neue Messung usw..

Im Hauptprogramm werden, wie bei komplexeren Programmen sinnvoll und üblich, hauptsächlich Routinen aufgerufen. Die LICHT_MESS-Routine wandelt die Spannung in einen digitalen Wert, der nach der Wandlung in dem ADAT-Register steht. Dieser Wert wird dann in das Register 12 kopiert und auf diese Weise der BIN2BCD-Routine übergeben, wo eine Umwandlung in eine Dezimalzahl stattfindet. Anschließend gibt die Routine den Dezimalwert auf der LCD-Anzeige aus. Danach springt das Programm in eine Warteroutine, wo es auf die Betätigung des P0.0-Schalters wartet. Nachdem der P0.0-Schalter gedrückt wurde, wird in der P0ISR-Routine ein Flag gesetzt, das es erlaubt, die Warteroutine zu verlassen. Als nächstes wird der Wert aus dem ADAT-Register wieder in das Register 12 kopiert und auf diese Weise der HEXAUS-Routine übergeben, wo eine Umwandlung in eine Hexadezimalzahl stattfindet. Anschließend gibt die Routine den Dezimalwert

auf der LCD-Anzeige aus. Danach springt das Programm wieder in eine Warteroutine, wo es auf die Betätigung des P0.0-Schalters wartet. Nachdem der P0.0-Schalter gedrückt wurde, wird in der P0ISR-Routine ein Flag gesetzt, das es erlaubt, die Warteroutine zu verlassen. Als Letztes springt das Programm an den Anfang des Hauptprogramms zurück.

8.4 Erläuterungen zu den Programmen

Hier einige Stellen aus den Programmen, die nach Meinung des Autors von Bedeutung sind:

→ **Vereinbarungen**

Den Namen werden geeignete Zahlenwerte zugewiesen, die dann im Programm als Adressen oder Bitmuster benutzt werden.

SP_orig	.set	03DEh	Anfangsadresse des Stapels im STK
IE1	.equ	0h	Adresse des IE1-Registers im Speicher
IE2	.equ	01h	Adresse des IE2-Registers im Speicher
LCDM	.equ	30h	Adresse des LCD-Steuerregisters im Speicher
LCD1	.equ	31h	Adresse des LCD-Registers im Speicher
WDTCTL	.equ	120h	Adresse des Watchdog-Steuerregisters im Speicher
WDTHold	.equ	80h	Stoppbit im WDTCTL-Register Bitmuster 1000 0000
WDTPW	.equ	05A00h	Paßwort/Schlüssel zu dem Watchdog Bitmuster 0101 1010 0000 0000
GIE	.equ	08h	Unterbrechungsfreigabe-Bit im SFR Bitmuster 0000 1000
P0IN0	.set	01h	P0.0-Bit im Eingabe-Register Bitmuster 0000 0001
P0IE0	.set	04h	P0.0-Bit im Unterbrechungsfreigabe-Register Bitmuster 0000 0100
P0IFG0	.set	04h	P0.0-Bit im Unterbrechungsmerker-Register Bitmuster 0000 0100

→ **Hauptprogramm**

MOV #SP_orig,SP
Die Zahl, die der SP_orig-Konstanten zugewiesen wurde, wird in das SP-Register (R1) kopiert.
Der Stapelzeiger zeigt jetzt auf die Adresse 3DEh.

MOV #(WDTHold+WDTPW), &WDTCTL
Die Bitmuster 1000 0000 (80h) und 0101 1010 0000 0000 (5A00h) werden addiert. Das Ergebnis 0101 1010 1000 0000 wird in den Speicherplatz unter der Adresse 120h (Watchdog-Steuer-Register) kopiert.

<code>MOV #WDTPW+TMSEL+CNTCL+T250MS, &WDTCTL</code>	Die Bitmuster 0101 1010 0000 0000 (5A00h), 0001 0000 (10h), 1000 (8h) und 0101 (5h) werden addiert. Das Ergebnis 0101 1010 0001 1101 wird in den Speicherplatz unter der Adresse 120h (Watchdog-Kontrol-Register) kopiert.
<code>MOV.B #0, LCD1-1 (R6)</code> oder <code>MOV.B #0, LCD1 (R6-1)</code> oder <code>MOV.B #0, LCDM (R6)</code>	Alle drei Möglichkeiten beschreiben den gleichen Vorgang: die Speicherstelle die sich unter der Adresse LCD1-1+R6 bzw. 31h-1+R6 befindet, wird mit Null überschrieben. Die -1 ist notwendig, da in der Schleife von 8 bis 1 gezählt wird und Speicherplätze der einzelnen Digits der LCD-Anzeige sich in den Adressen 31h+7 bis 31+0 befinden (31h-1+8 bis 31h-1+1).
<code>MOV.B #-1h, LCDM</code> oder <code>MOV.B #FFh, LCDM</code>	Beide Möglichkeiten beschreiben den gleichen Vorgang: die Speicherstelle die sich unter der Adresse LCDM befindet, wird mit FFh (Bitmuster 1111 1111) überschrieben.
<code>MOV.B LCD_Tab (R10), LCDM (R8)</code>	Das Bitmuster aus der LCD-Tabelle, die Position steht im R10, wird in das LCD-Register kopiert, dessen relative Adresse im R8 steht.
<code>MOV.B LCD_Tab (1), LCDM (2)</code>	Das Bitmuster 0001 0010 (02h+10h bzw. b+c aus der LCD_TYPE) wird in die Adresse 30h+2 (zweite Stelle auf der LCD-Anzeige) kopiert.
<code>BIS.B #P0IE0, IE1</code>	Setze das P0IE.0-Bit in dem IE1-Register. Setze das dritte Bit (Bitmuster 0100 (04h)) in dem Byte, an der Adresse 0h.
<code>BIC.B #P0IFG0, IFG1</code>	Lösche das P0IFG.0-Bit in dem IFG-Register. Lösche das dritte Bit (Bitmuster 0100 (04h)) in dem Byte, an der Adresse 02h.
<code>CLR.B IE1</code>	Lösche alle Bits (8 Bits) in dem IE1-Register. Achtung: Bytezugriff.
<code>CLR.B IFG2</code>	Lösche alle Bits (8 Bits) in dem IFG2-Register. Achtung: Wordzugriff.
Schleife <code>NOP</code> <code>JMP Schleife</code>	Eine unendliche Schleife, meistens wird sie in Programmen verwendet, in denen die Ereignisse durch die Interrupts bestimmt werden. D.h. der Befehl NOP bildet eine ABM (Arbeits-Beschaffung-Maßnahme) für den Prozessor, solange er auf einen Interrupt wartet. Man braucht ihn aber nicht, mit <code>JMP \$</code> geht es auch ohne Label.

Literaturverzeichnis

Zur Erstellung dieser Einführung wurden folgenden Bücher benutzt:

- **MSP430 Family:**
 - *Getting Started With The MSP430 Microcontroller.* Texas Instruments 1999
 - *Starter Kit/Evaluation Kit Manual.* Texas Instruments 1996
 - *Application Report.* Texas Instruments 1999
 - *Metering Application Report.* Texas Instruments 1997
 - *Software User's Guide.* Texas Instruments 1994
 - *Architecture User's Guide and Module Library, Data Book User's Guide.* Texas Instruments 1996
 - *Simulation Environment and LCD-Editor Manual.* Texas Instruments 1996
 - *Assembly Language Tools User's Guide.* Texas Instruments 1994
 - *Experiments for the MSP430 Starter Kit (SLAA079).* Texas Instruments 1999
 - *Understanding the MSP430x325 14-Bit ADC (SLAA039).* Texas Instruments 1999
- **Krüger, Tilmann:**
 - Mikrocontroller MSP430, Eine Einführung in die Assemblerprogrammierung energiesparender Mikrocontroller
 - Leitfaden zur Gestaltung von Studien- und Diplomarbeiten. Kleine Handreichung zur besseren Gestaltung. FH Mannheim 1996
- **Backer, Reiner:** Programiersprache Assembler. Eine strukturierte Einführung. rororo computer 1995
- **Syck, Gary:** Turbo Assembler. Biblia uzytkownika. LT&P 1994
- Das Bertelsman Lexikon in 24 Bänden
- **Bünting, Karl-Dieter:** Deutsches Wörterbuch. Isis Verlag 1996
- **Duden 2:** Stilwörter. Dudenverlag 1988
- **Duden 8:** Sinn- und sachverwandte Wörter. Dudenverlag 1997
- **Duden:** Briefe gut und richtig schreiben! Dudenverlag 1987
- Großes Wörterbuch. Synonyme. Buch und Zeit Verlagsgesellschaft mbH 1995
- 40.000 Super-Cliparts. GraphicCorp. 1999

Stichwortverzeichnis

A

A/D-Wandler 28
Anschlüsse 2
Assembler 3, 15, 23

B

Basic Timer 28
Breakpoint 10, 23
Bücher 5
Button 10
Byte-Zugriff 26

C

CG1, CG2 25
COM 9, 16, 20
CPU 25

D

Definitionen 30
Deinstallation 4
Demoprogramm 4, 9
Dialogfenster 11
Digital I/O 27

E

Editor 3, 14, 15, 22
Emulation 20
Emulator 19
EPROM-Monitor 3, 22, 23

F

Fehler 16
Fenster 10

H

Hilfe 4
Haltepunkte 10, 14, 23
Haltezeit 14
Hauptprogramm 30, 32
HyperTerminal 6, 9, 18

I

Initialisierungen 30
Installation 6, 7
Interrupt 24, 32
Interruptroutine 32
I/O-Eingang 2, 27
ISR 24

K

Kommentare 30

L

LCD-Anzeige 2, 9, 14, 23, 28
Lichtsensor 2

M

Maschinensprache 3, 15
Menü 10
Menü-Element 10
Menü-Leiste 11, 21
Monitor 3, 22

P

PC 25
Platine 2
Peripherien 24
Prozessor 2, 13

Q

Quelltext 3

R

RAM 22
Register 10, 24, 25
Reset 2, 20
ROM 22
Routinen 30, 34

S

Schnittstelle 18, 19, 20
Senden e. Datei 21
Setup 7
SFR 26
Simulation 11
SP 25
Speicher 10, 26
SR 25
SR-Register 24
Statuszeile 10, 17
Sterbefälle 23
Steuerregister 25
STK 2
System Clock Generator 28

T

Tabelle 32
Terminal 3, 18, 19, 23
Toolbar 11

U

Übertragen e. Datei 17, 21
Unterbrechung 24
Unterbrechungsrouninen 34
Unterprogramme 30, 34

V

Vektorentabelle 26, 33

W

Watchdog 29
Werkzeugkiste 11
Word-Zugriff 26

Z

Zugriffsart 26

Anhang

```
1 ****
2 *** Zenon Schymiczek
3 *** Einführung in den Umgang mit MSP430
4 *** Studienarbeit
5 *** FH Mannheim
6 *** Fachbereich Automatisierungstechnik
7 *** Insitut für Elektronische Steuerungstechnik
8 ***
9 *** WD_BSP - Programm zu Vorführung der Funktionsweise des Watchdog-Registers
10 *** und der LCD-Anzeige
11 ****
12
13 ;*** Setze die Variable SIM zu '1' um im SIMULATOR zu arbeiten ***
14 00 SIM .set 0 ; 1 = Simulator
15 ; 0 = STK/EVK
16 03de SP_orig .set 003DEh ; Adresse des Stackpointer im STK/EVK
17 0240 RAM_orig .set 00240h ; Anfangsadresse des Programmes im STK/EVK
18 01 lcd_req .set 001h ; lcd request
19
20 .if SIM = 0
21 03ff USER_END .set 003FFh ; Letzte Adresse im RAM-Speicher des STK
22 80 lcd_ival .set 080h ; 128*1/2 lcd shift interval 1/2 sec for STK
23 .else
24 USER_END .set 0FFFFh ; Letzte Adr. im Speicher des sim. Prozessors
25 lcd_ival .set 1 ; 1*1/2 lcd shift interval 1/2 sec for Sim
26 .endif
27
28 ;--- RAM Zuordnung
29
30 0220 .bss runoff,1,220h ; runoff control flag register
31 0221 .bss lcd_timer,1 ; lcd interval timer
32
33 ;--- Definieren der Konstanten
34
35 ;SFR-Special Funktion Register
36 00 IE1 .equ 0h ; Interrupt Freigabe-Register
37 01 IE2 .equ 01h ; Interrupt Freigabe-Register
38 02 IFG1 .equ 02h ; Interrupt Merk-Register
39 03 IFG2 .equ 03h ; Interrupt Merk-Register
40 04 ME1 .equ 04h ; Freigabe-Register der Module
41 05 ME2 .equ 05h ; Freigabe-Register der Module
42
43 ;LCD-Anzeige
44 30 LCDM .equ 030h ; LCD-Anzeige Steuer-Register
45 31 LCD1 .equ 031h ; Anfang der LCD-Tabelle, Digit 1
46
47 ;Basic Timer
48 80 BTME .set 080h ; BT Modul Freigabe-Bit
49 80 BTIE .set 080h ; BT Interrupt Freigabe-Bit
50 80 BTIFG .equ 080h ; BT Interrupt Richtung-Bit
51 40 BTCTL .equ 040h ; Adresse des BT Kontrol-Register
52 42 TCCTL .equ 042h ; Adresse der Timer/Counter Control-Register
53 43 TCPLD .equ 043h ; Adresse der Timer/Counter pre-load-Register
54 44 TCDAT .equ 044h ; Adresse der Timer/Counter
55
56 ;Watchdog Timer
57 0120 WDTCTL .equ 0120h ; Adresse des WDT Kontrol-Register
58 80 WDTHold .equ 80h ; WDT Stop-Bit
59 5a00 WDTPW .equ 05A00h ; Passwort
60 10 TMSSEL .equ 010h ; Watchdog oder Timer Auswahl-Bit
61 08 CNTCL .equ 8h ; Neustart-Bit
62 03 T0_064MS .equ 3h ; 0,064ms-Interval
63 02 T0_5MS .equ 2h ; 0,5ms -Interval
64 07 T1_9MS .equ 7h ; 1,9ms -Interval
65 01 T8MS .equ 1h ; 8ms -Interval
66 06 T16MS .equ 6h ; 16ms -Interval
67 00 T32MS .equ 0h ; 32ms -Interval
68 05 T250MS .equ 5h ; 250ms -Interval
69 04 T1000MS .equ 4h ; 1s -Interval
70
71 ;General Interrupt Enable
72 08 GIE .equ 08h ; Freigabe aller Interrupts-Bit
73
74
75 ;*****
76 ; Initialisierungen der Register
77 ;*****
78
79 0240 .sect "MAIN",RAM_orig
80 0240 RESET
81 0240 403103de MOV #SP_orig,SP ; Initialisieren des Stack Pointers
82 ; Anfangsadresse des Stack Pointers
83
```

Anhang

```

84                                     ;Initialisieren der LCD-Anzeige und Basic Timer-Registers
85
86 0244 43f20030                       MOV.B #-1h,LCDM           ; LCD : Analog generator on
87                                     ;           Low impedance of AG
88                                     ;           4Mux active
89                                     ;           all outputs are Seg
90 0248 40f200570040                   MOV.B #057h,BTCTL       ; Basic-Timer : SSEL=0 DIV=0 Reset=1
91                                     ;           ACLK
92                                     ;           32768/256 = 128Hz (7.8ms debounce time)
93                                     ;           LCD-Grundfrequenz @4Mux: 64Hz
94 024e d0f200800005                   BIS.B #BTME,ME2         ; aktivieren des Basic-Timer-Moduls
95 0254 c0f200400040                   BIC.B #040h,BTCTL       ; löschen des Basic-Timer-Reset-Bits
96 025a d0f200800001                   BIS.B #BTIE,IE2        ; setze Freigabe-Bit der BT-Interruptroutine
97 0260 -40f00080ffbd                 MOV.B #lcd_ival,lcd_timer ; load SW lcd timer
98 0266 +12b00292                       CALL #loesch_LCD        ; Aufruf der LCD-Löschroutine
99
100                                     ;*****
101                                     ; Hauptprogramm
102                                     ;*****
103
104                                     ;Löschen der Special Function Registers
105
106 026a 43c20000                       CLR.B IE1               ; alle maskierbaren Interrupts abschalten
107 026e 43c20001                       CLR.B IE2               ;
108 0272 43c20002                       CLR.B IFG1              ; und alle Interruptflags löschen
109 0276 43c20003                       CLR.B IFG2              ;
110
111 027a                                     mainloop
112 027a d3d20000                       BIS.B #1,IE1           ; setze Bit1 im IE-Register,
113                                     ;           Oszillatorfehler erkennen
114 027e c3d20002                       BIC.B #1,IFG1          ; lösche Bit1 im IFG1-Register,
115                                     ;           und rücksetzen
116 0282 40b25a1d0120                   MOV #WDTPW+TMSEL+CNTCL+T250MS,&WDTCTL
117                                     ;           Watchdog-Einstellungen: 0,25s-Interval
118 0288 d232                           EINT                   ; 1->GIE, Freigabe der Interrupts
119 028a 430b                           MOV #0,R11              ; schiebe 0 in das Register11
120                                     ;           Initialisieren des Zählers
121 028c 432c                           MOV #2,R12              ; schiebe 2 in das Register12,
122                                     ;           Position 2 auf der LCD-Anzeige
123 028e                                     Schleife
124 028e 4303                           NOP                     ; unendliche Schleife bis Interrupt kommt
125 0290 +3ffe                           JMP Schleife            ; springe an den Anfang der Schleife
126
127
128                                     ; * * * * *
129                                     ; Routinen
130                                     ; * * * * *
131
132                                     ;löschen der LCD           ; Löschroutine
133
134 0292                                     loesch_LCD
135 0292 1205                           PUSH R5                 ; retten des Inhalts von Register5(wird auf
136                                     ;           dem STACK abgelegt)
137 0294 4235                           MOV #8, R5              ; schiebe 8 in das Register5, für 8 Durchläufe
138                                     ;           der Schleife (8-Stellen der LCD-Anzeige)
139 0296                                     LCD_Marke
140 0296 43c50030                       MOV.B #0,LCD1-1(R5)    ; 0 in das LCDM-Register, die Position steht
141                                     ;           im Register5
142 029a 8315                           DEC R5                 ; decrementieren des Inhalts von Register5
143 029c +23fc                           JNZ LCD_Marke          ; solange Register5 nicht 0 springe zurück
144                                     ;           zu Sprungmarke
145 029e 4135                           POP R5                 ; Register5 wieder mit dem geretteten Wert
146                                     ;           beschreiben
147 02a0 4130                           RET                     ; Ende der Löschroutine
148
149                                     ; * * * * *
150                                     ; Interruptroutinen
151                                     ; * * * * *
152
153                                     ;Watchdog
154 02a2                                     WDTISR
155 02a2 c232                           DINT                   ; Watchdog Interrupt Service Routine
156 02a4 903b000a                       CMP #10,R11            ; 0->GIE, alle anderen Interrupts werden ignoriert
157 02a8 +2001                           JNZ Marke1             ; Vergleich: ist der Inhalt von Register 11 gleich 10
158                                     ;           ; falls nicht: springe nach Marke1 (die Zahl im
159 02aa 430b                           MOV #0h,R11            ; Register11 wird nicht geändert)
160 02ac                                     Marke1
161 02ac +12b00292                       CALL #loesch_LCD       ; LCD löschen
162 02b0 +4bdc02c20030                   MOV.B LCD_Tab(R11),LCDM(R12) ; Die Zahl aus dem Register11 wird auf
163                                     ;           dem LCD-Display angezeigt
164 02b6 531b                           ADD #01,R11            ; addiere 1 zum Inhalt von Register11
165 02b8 43c20002                       CLR.B IFG1              ; lösche das IFG1 Byte, für die nächste Interrupt
166                                     ;           Meldung
167 02bc d232                           EINT                   ; 1->GIE, Freigabe der Interrupts
168 02be 1300                           RETI                    ; Ende der WDTISR

```

Anhang

```

169
170
171          ;Basic-Timer
172 02c0      BTISR          ; Basic Timer Interrupt Service Routine
173 02c0      1300          RETI          ; Ende der BTISR
174
175
176          ; * * * * *
177          ; LCD Definitionen
178          ; * * * * *
179
180 02c2      LCD_TYPE      ; STK/EVK LCD
181          a          .equ      01h
182          b          .equ      02h
183          c          .equ      10h
184          d          .equ      04h
185          e          .equ      80h
186          f          .equ      20h
187          g          .equ      08h
188          h          .equ      40h
189
190          ;--- Character-Zeichen definieren
191
192 02c2      b7          LCD_Tab .byte  a+b+c+d+e+f      ; "0", Aufruf über LCD_Tab(0)
193 02c3      12          .byte      b+c                ; "1", Aufruf über LCD_Tab(1)
194 02c4      8f          .byte      a+b+d+e+g          ; "2", Aufruf über LCD_Tab(2)
195 02c5      1f          .byte      a+b+c+d+g          ; "3", Aufruf über LCD_Tab(3)
196 02c6      3a          .byte      b+c+f+g            ; "4", Aufruf über LCD_Tab(4)
197 02c7      3d          .byte      a+c+d+f+g          ; "5", Aufruf über LCD_Tab(5)
198 02c8      bd          .byte      a+c+d+e+f+g        ; "6", Aufruf über LCD_Tab(6)
199 02c9      13          .byte      a+b+c              ; "7", Aufruf über LCD_Tab(7)
200 02ca      bf          .byte      a+b+c+d+e+f+g      ; "8", Aufruf über LCD_Tab(8)
201 02cb      3f          .byte      a+b+c+d+f+g        ; "9", Aufruf über LCD_Tab(9)
202 02cc      LCD_Tab_End
203
204
205          ; * * * * *
206          ; Interrupt Vektoren-Tabelle
207          ; * * * * *
208
209 03e0      .sect      "Int_Vect",USER_END-31
210          ; Anfang der Tabelle befindet sich 31 Speicherplätze
211          ; vor dem Ende des Speichers
212 03e0      +0240      .word      RESET
213 03e2      +02c0      .word      BTISR ; Basic Timer
214 03e4      +0240      .word      RESET
215 03e6      +0240      .word      RESET
216 03e8      +0240      .word      RESET
217 03ea      +0240      .word      RESET
218 03ec      +0240      .word      RESET
219 03ee      +0240      .word      RESET
220 03f0      +0240      .word      RESET
221 03f2      +0240      .word      RESET
222 03f4      +02a2      .word      WDTISR ; Watchdog/Timer, Timer mode
223 03f6      +0240      .word      RESET
224 03f8      +0240      .word      RESET
225 03fa      +0240      .word      RESET
226 03fc      +0240      .word      RESET
227 03fe      +0240      .word      RESET
228          .end

```

No Errors, No Warnings

Anhang

```
1 ****
2 *** Zenon Schymiczek
3 *** Einführung in den Umgang mit MSP430
4 *** Studienarbeit
5 *** FH Mannheim
6 *** Fachbereich Automatisierungstechnik
7 *** Insitut für Elektronische Steuerungstechnik
8 ***
9 *** P0_0_BSP - Programm zur Vorführung der Funktionsweise der LCD-Anzeige,
10 *** des P0-Schalters und der Zeichenübergabe an den PC
11 ****
12
13 ;*** Setze die Variable SIM zu '1' um im SIMULATOR zu arbeiten ***
14 00 SIM .set 0 ; 1 = Simulator
15 ; 0 = STK/EVK
16 03de SP_orig .set 003DEh ; Adresse des Stackpointer im STK/EVK
17 0240 RAM_orig .set 00240h ; Anfangsadresse des Programmes im STK/EVK
18 01 lcd_req .set 001h ; lcd request
19
20 .if SIM = 0
21 03ff USER_END .set 003FFh ; Letzte Adresse im RAM-Speicher des STK
22 80 lcd_ival .set 080h ; 128*1/2 lcd shift interval 1/2 sec for STK
23 .else
24 USER_END .set 0FFFFh ; Letzte Adr. im Speicher des sim. Prozessors
25 lcd_ival .set 1 ; 1*1/2 lcd shift interval 1/2 sec for Sim
26 .endif
27
28 ;--- RAM Zuordnung
29
30 0220 .bss runoff,1,220h ; runoff control flag register
31 0221 .bss lcd_timer,1 ; lcd interval timer
32
33 ;--- Definieren der Konstanten
34
35 ;SFR-Special Funktion Register
36 00 IE1 .equ 0h ; Interrupt Freigabe-Register
37 01 IE2 .equ 01h ; Interrupt Freigabe-Register
38 02 IFG1 .equ 02h ; Interrupt Merk-Register
39 03 IFG2 .equ 03h ; Interrupt Merk-Register
40 04 ME1 .equ 04h ; Freigabe-Register der Module
41 05 ME2 .equ 05h ; Freigabe-Register der Module
42
43 ;LCD-Anzeige
44 30 LCDM .equ 030h ; LCD-Anzeige Steuer-Register
45 31 LCD1 .equ 031h ; Anfang der LCD-Tabelle, Digit 1
46
47 ;Basic Timer
48 80 BTME .set 080h ; BT Modul Freigabe-Bit im BTCTL-Register
49 80 BTIE .set 080h ; BT Interrupt Freigabe-Bit im IE-Register
50 80 BTIFG .equ 080h ; BT Interrupt Richtung-Bit im IFG-Register
51 40 BTCTL .equ 040h ; Adresse des BT Kontrol-Registers
52 42 TCCTL .equ 042h ; Adresse des Timer/Counter Kontrol-Registers
53 43 TCPLD .equ 043h ; Adresse des Timer/Counter pre-load-Registers
54 44 TCDAT .equ 044h ; Adresse des Timer/Counters
55
56 ;Watchdog Timer
57 0120 WDTCTL .equ 0120h ; Adresse des WDT Kontrol-Register
58 80 WDTHold .equ 80h ; WDT Stop-Bit im WDTCTL-Register
59 5a00 WDTPW .equ 05A00h ; Passwort mi WDTCTL-Register(High-Byte)
60 10 TMSSEL .equ 010h ; Watchdog oder Timer Auswahl-Bit im WDTCTL-Reg.
61 08 CNTCL .equ 8h ; Neustart-Bit im WDTCTL-Register
62 03 T0_064MS .equ 3h ; 0,064ms-Interval
63 02 T0_5MS .equ 2h ; 0,5ms -Interval
64 07 T1_9MS .equ 7h ; 1,9ms -Interval
65 01 T8MS .equ 1h ; 8ms -Interval
66 06 T16MS .equ 6h ; 16ms -Interval
67 00 T32MS .equ 0h ; 32ms -Interval
68 05 T250MS .equ 5h ; 250ms -Interval
69 04 T1000MS .equ 4h ; 1s -Interval
70
71
72 ;General Interrupt Enable
73 08 GIE .equ 08h ; Freigabe aller Interrupts-Bit
74 ; im Status Register, R2-Low
75
76 ;P0-Ein/Ausgang
77 10 P0IN .equ 010h ; Adresse des Eingabe-Registers
78 11 P0OUT .equ 011h ; Adresse des Ausgabe-Registers
79 12 P0DIR .equ 012h ; Adresse des Richtungs-Registers
80 13 P0IFG .equ 013h ; Adresse des Interrupt-Merker-Registers
81 14 P0IES .equ 014h ; Adresse des Flanken-Auswahl-Registers
82 15 P0IE .equ 015h ; Adresse des Interrupt-Freigabe-Registers
83
84 01 P0IN0 .set 001h ; P0.0 Eingabe-Bit im P0IN-Register
85 04 P0IE0 .set 004h ; P0.0 Interrupt-Freigabe-Bit im IE-Register
```

Anhang

```

86      04      POIFG0      .set      004h      ; P0.0 Interrupt-Merker-Bit im IFG-Register
87      01      PODIRO      .set      001h      ; P0.0 Richtungs-Register-Bit im PODIR-Register
88      01      POIES0      .set      001h      ; P0.0 Flanken-Auswahl-Bit im POIES0-Register
89
90      ;Funktionen der RS-232 Schnittstelle (zwischen STK und PC)
91      ffd2      TXCHAR      .equ      0FFD2h      ; übertrage ein Zeichen in TXData-Register
92      ffd4      TXTABLE      .equ      0FFD4h      ; übertrage eine Tabelle (string)
93      ffd6      PREPRX      .equ      0FFD6h      ; vorbereite Wechselverkehr-Software UART
94      ; für Empfang
95      ffd8      PREPTX      .equ      0FFD8h      ; vorbereite Wechselverkehr-Software UART
96      ; für Senden
97      ffd8      INT_RXTX      .equ      0FFD8h      ; Interrupt zum Empfangen und Senden
98      020e      TXDATA      .equ      020Eh      ; Adresse des Zeichen-Speichers
99      0210      RXBUF      .equ      0210h      ; Adresse des Puffers für Strings
100
101      ;*****
102      ; Initialisierungen des Prozessors und der Register
103      ;*****
104      0240      .sect "MAIN",RAM_orig
105      0240      RESET
106      0240      403103de      MOV      #SP_orig,SP      ; Initialisieren des Stack Pointers
107      ; Anfangsadresse des Stack Pointers
108      0244      40b25a800120      MOV      #(WDTHold+WDTFW),&WDTCTL ; Stop Watchdog Timer
109
110      ;Initialisieren der LCD-Anzeige und Basic Timer-Registers
111
112      024a      43f20030      MOV.B   #-1h,LCDM      ; LCD : Analog generator on
113      ; Low impedance of AG
114      ; 4Mux active
115      ; all outputs are Seg
116      024e      40f200570040      MOV.B   #057h,BTCTL      ; Basic-Timer : SSEL=0 DIV=0 Reset=1
117      ; ACLK
118      ; 32768/256 = 128Hz (7.8ms debounce time)
119      ; LCD-Grundfrequenz @4Mux: 64Hz
120      0254      d0f200800005      BIS.B   #BTME,ME2      ; aktivieren des Basic-Timer-Moduls
121      025a      c0f200400004      BIC.B   #040h,BTCTL      ; löschen des Basic-Timer-Reset-Bits
122      0260      d0f200800001      BIS.B   #BTIE,IE2      ; setze Freigabe-Bit der BT-Interruptroutine
123      0266      -40f00080ffb7      MOV.B   #lcd_ival,lcd_timer ; lade SW lcd_timer
124      026c      +12b0028e      CALL    #loesch_LCD      ; Aufruf der LCD-Löschroutine
125
126      ;*****
127      ; Hauptprogramm
128      ;*****
129
130      0270      mainloop
131      ;Löschen der Special Function Registers
132      0270      43c20000      CLR.B   IE1      ; Bit 0-7 des IE-Registers löschen
133      0274      43c20001      CLR.B   IE2      ; Bit 8-15 des IE-Registers löschen
134      0278      43c20002      CLR.B   IFG1     ; Bit 0-7 des IFG-Registers löschen
135      027c      43c20003      CLR.B   IFG2     ; Bit 8-15 des IFG-Registers löschen
136
137      ;Initialisierungen
138      0280      d2e20000      BIS.B   #POIE0,IE1     ; P0-Interrupt freigeben
139      0284      d232      EINT      ; 1->GIE, Freigabe der maskierbaren Interrupts
140      0286      430a      MOV      #0,R10      ; schiebe 0 in das Register10
141      ; Initialisieren des Zählers
142      0288      4318      MOV      #1,R8      ; schiebe 1 in das Register8
143      ; Initialisieren des Zählers
144      028a      Schleife
145      028a      4303      NOP      ; unentliche Schleife bis Interrupt kommt
146      028c      +3ffe      JMP      Schleife      ; springe an den Anfang der Schleife
147
148
149      ; * * * * *
150      ; Routinen
151      ; * * * * *
152
153      ;löschen der LCD-Anzeige
154      028e      loesch_LCD
155      028e      1206      PUSH    R6      ; retten des Inhalts von Register6(wird auf
156      ; dem STACK abgelegt)
157      0290      4236      MOV      #8,R6      ; schiebe 8 in das Register6, für 8 Durchläufe
158      ; der Schleife (8-Stellen der LCD-Anzeige)
159      0292      LCD_Marke
160      0292      43c60030      MOV.B   #0,LCD1-1(R6) ; 0 in das LCDM-Register, die Position steht
161      ; im Register6
162      0296      8316      DEC      R6      ; decrementiere Inhaht von Register6
163      0298      +23fc      JNZ     LCD_Marke      ; solange Register6 nicht 0, springe zurück
164      ; zu Sprungmarke
165      029a      4136      POP      R6      ; Register6 wieder mit dem geretteten Wert
166      ; beschreiben
167      029c      4130      RET      ; Ende der Löschroutine
168
169      ;nächste Routine

```

Anhang

```

170 029e           Anzeigen_auf_LCD
171 029e +4ad8032c0030   MOV.B LCD_Tab(R10),LCDM(R8) ; Die Zahl aus dem Register10 wird auf
172                                     ; dem LCD-Display angezeigt an der
173                                     ; Position, die von Register8 vorgegeben
174                                     ; wird
175 02a4 4130           RET ; Ende der Routine
176
177 ;und noch eine Routine
178 02a6 Entprellen ;in dieser Routine wird die Zeit vernichtet, damit
179                                     ;die Abstände zwischen den betätigungen von P0
180                                     ;größer werden
181 02a6 1206           PUSH R6 ; retten des Inhalts von Register6(wird auf
182                                     ; dem STACK abgelegt)
183 02a8 4336           MOV #0FFFFh,R6 ; riesige Zahl in den Register schieben
184 02aa Penelope1
185 02aa 8316           DEC R6 ; Zahl um 1 kleiner
186 02ac +23fe         JNZ Penelope1 ; solange noch etwas im Register6 steht
187                                     ;und wieder das Gleiche
188                                     ;noch mehr Zeit vernichten
189 02ae 4336           MOV #0FFFFh,R6 ; riesige Zahl in den Register schieben
190 02b0 Penelope2
191 02b0 8316           DEC R6 ; Zahl um 1 kleiner
192 02b2 +23fe         JNZ Penelope2 ; solange noch etwas im Register6 steht
193                                     ; springe zurück
194 02b4 4136           POP R6 ; Register6 wieder mit dem geretteten Wert
195                                     ; beschreiben
196 02b6 4130           RET ; Ende der Odyssee
197
199 ;Schnittstelle PC-STK, Übergabe eines Zeichens an PC, das Zeichen steht immer
200 ;im Register5
201 02b8 RS ; (RS232 Schnittstelle)
202 02b8 40b2aa5503de   MOV #0AA55h,&03DEh ; Ident-code in die Adresse 03DEh kopieren
203 02be 45c2020e     MOV.B R5,&TXDATA ; Zeichen aus dem Register5 in den Zeichen-
204                                     ; speicher kopieren
205 02c2 c3920200     BIC #01h,&200h ; erstes Bit an der Adresse 200h löschen
206 02c6 1292ffd2     CALL &TXCHAR ; Aufruf einer Routine die ein Zeichen in den
207                                     ; TXDATA Zeichenspeicher überträgt
208 02ca 1292ffd6     CALL &PREPRX ; Aufruf der Empfangroutine
209 02ce 438203de     MOV #0000h,&03DEh ; Löschen des Ident-code
210 02d2 4130           RET
211
212
213 ; * * * * *
214 ; Interruptroutinen
215 ; * * * * *
216
217 02d4 POISR ;diese Routine wird immer aktiviert wenn P0 gedrückt wird
218 02d4 c232           DINT ; 0->GIE, ab jetzt werden alle Interrupts ignoriert
219                                     ;Zähler von 1 bis 9
220 02d6 903a0009     CMP #9,R10 ; Vergleich: ist der Inhalt von Register10 gleich 9?
221 02da +2002         JNZ S1 ; falls nicht: springe nach S1 (die Zahl im
222                                     ; Register10 wird nicht geändert)
223 02dc 430a           CLR R10 ; lösche Register10
224 02de +3c01         JMP S2 ; überspringe die nächste Zeile
225 02e0 531a           ADD #1,R10 ; addiere 1 zum Inhalt des Registers10
226 02e2 S2 ; Sprungmarke
227                                     ;Zähler von 7 bis 1 (Stelle 7 bis 1 auf der LCD-Anzeige)
228 02e2 9318           CMP #1,R8 ; Vergleich: ist der Inhalt von Register8 gleich 1?
229                                     ; (Stelle 1 erreicht)
230 02e4 +2003         JNZ S3 ; falls nicht: springe nach S3 (die Zahl im Register8
231                                     ; wird nicht geändert)
232 02e6 40380007     MOV #7,R8 ; belege Register8 mit 7 (zurück zur Stelle 7)
233 02ea +3c01         JMP S4 ; überspringe die nächste Zeile
234 02ec 8318           DEC R8 ; Register8 -1 (eine Stelle nach rechts)
235 02ee S4 ; Sprungmarke
236                                     ;Zahl auf der LCD-Anzeige anzeigen
237 02ee +12b0028e     CALL #loesch_LCD ; Aufruf der LCD-Löschroutine
238 02f2 +12b0029e     CALL #Anzeigen_auf_LCD; Die Zahl aus dem Register10 wird an der
239                                     ; (Register8)-Stelle des LCD-Displays angezeigt
240                                     ;Senden der gleichen Zahl zum PC
241 02f6 4035000d     MOV #13,R5 ; Im ASCII 13 = CR-Gerätesteuer-Zeichen
242 02fa +12b002b8     CALL #RS ; aufruf der Übertragungsroutine
243                                     ; erstes Zeichen an PC senden, bereite den Datentransfer
244 02fe 4035000a     MOV #10,R5 ; Im ASCII 10 = LF-Zeilenvorschub-Zeichen
245 0302 +12b002b8     CALL #RS ; aufruf der Übertragungsroutine
246                                     ; zweites Zeichen an PC senden, Zeilenvorschub
247 0306 40350020     MOV #32,R5 ; Leerzeichen
248 030a +12b002b8     CALL #RS ; aufruf der Übertragungsroutine
249                                     ; drittes Zeichen an PC senden, Leerzeichen
250 030e 4a05           MOV R10,R5 ; Zahl aus dem Register10 in das Register5 kopieren
251 0310 50350030     ADD #48,R5 ; 48 addieren wegen ASCII
252 0314 +12b002b8     CALL #RS ; aufruf der Übertragungsroutine
253                                     ; Zahl wird Angezeigt
254

```

Anhang

```
255 0318 +12b002a6          CALL    #Entprellen ; die Zeit vernichten, damit die Abstände zwischen
256                                     ; den Betätigungen von P0 größer werden
257
258 031c  43c20002          MOV.B  #0,IFG1      ; lösche das IFG1, für die nächste Interrupt-Meldung
259 0320  d232              EINT      ; 1->GIE, Freigabe der Interrupts
260 0322  1300              RETI      ; Ende der P0ISR
261
262                                     ;Basic-Timer
263 0324          BTISR          ; Basic Timer Interrupt Service Routine
264 0324  1300              RETI      ; Ende der BTISR
265
266 0326          UART
267 0326  4210ffda          BR      &INT_RXTX ; weiteres Programm unter der Adresse 0FFDAh
268 032a  1300              RETI
269
270                                     ; * * * * *
271                                     ; LCD Definitionen
272                                     ; * * * * *
273
274 032c          LCD_TYPE          ; STK/EVK LCD
275          01          a      .equ    01h
276          02          b      .equ    02h
277          10          c      .equ    10h
278          04          d      .equ    04h
279          80          e      .equ    80h
280          20          f      .equ    20h
281          08          g      .equ    08h
282          40          h      .equ    40h
283
284                                     ;--- Character-Zeichen definieren
285
286 032c  b7          LCD_Tab .byte  a+b+c+d+e+f      ; "0", Aufruf über LCD_Tab(0)
287 032d  12          .byte  b+c                    ; "1", Aufruf über LCD_Tab(1)
288 032e  8f          .byte  a+b+d+e+g              ; "2", Aufruf über LCD_Tab(2)
289 032f  1f          .byte  a+b+c+d+g              ; "3", Aufruf über LCD_Tab(3)
290 0330  3a          .byte  b+c+f+g                ; "4", Aufruf über LCD_Tab(4)
291 0331  3d          .byte  a+c+d+f+g              ; "5", Aufruf über LCD_Tab(5)
292 0332  bd          .byte  a+c+d+e+f+g            ; "6", Aufruf über LCD_Tab(6)
293 0333  13          .byte  a+b+c                  ; "7", Aufruf über LCD_Tab(7)
294 0334  bf          .byte  a+b+c+d+e+f+g          ; "8", Aufruf über LCD_Tab(8)
295 0335  3f          .byte  a+b+c+d+f+g            ; "9", Aufruf über LCD_Tab(9)
296 0336          LCD_Tab_End
297
298
299                                     ; * * * * *
300                                     ; Interrupt Vektoren-Tabelle
301                                     ; * * * * *
302
303 03e0          .sect      "Int_Vect",USER_END-31
304                                     ; Anfang der Tabelle befindet sich 31 Speicherplätze
305                                     ; vor dem Ende des Speichers
306 03e0  +0240          .word    RESET
307 03e2  +0324          .word    BTISR ; Basic Timer
308 03e4  +0240          .word    RESET
309 03e6  +0240          .word    RESET
310 03e8  +0240          .word    RESET
311 03ea  +0240          .word    RESET
312 03ec  +0240          .word    RESET
313 03ee  +0240          .word    RESET
314 03f0  +0240          .word    RESET
315 03f2  +0240          .word    RESET
316 03f4  +0240          .word    RESET
317 03f6  +0240          .word    RESET
318 03f8  +0326          .word    UART ; Adresse des UART Steuerprogrammes
319 03fa  +02d4          .word    P0ISR ; P0.0
320 03fc  +0240          .word    RESET
321 03fe  +0240          .word    RESET
322                                     .end
```

No Errors, No Warnings

Anhang

```
1 ****
2 *** Zenon Schymiczek
3 *** Einführung in den Umgang mit MSP430
4 *** Studienarbeit
5 *** FH Mannheim
6 *** Fachbereich Automatisierungstechnik
7 *** Insitut für Elektronische Steuerungstechnik
8 ***
9 *** Programm zu Vorführung der Funktionsweise des AD-Wandlers
10 ***
11 *** AD_BSP - Das Programm nutzt den Lichtsensor der STK, die gemessene
12 *** Lichtintensität wird nach erstem drücken des P0.0-Schalters auf der
13 *** LCD-Anzeige als dezimale Zahl erscheinen, nach zweitem als Hexadezimale
14 *** Zahl. Dieses Programm kann nicht im Simulator betrieben werden
15 ****
16
17 03de SP_orig .set 003DEh ; Adresse des Stackpointer im STK/EVK
18 0240 RAM_orig .set 00240h ; Anfangsadresse des Programmes im STK/EVK
19 01 lcd_req .set 001h ; lcd request
20
21 03ff USER_END .set 003FFh ; Letzte Adresse im RAM-Speicher des STK
22 80 lcd_ival .set 080h ; 128*1/2 lcd shift interval 1/2 sec for STK
23
24 ;--- RAM Zuordnung
25
26 0220 .bss runoff,1,220h ; runoff control flag register
27 0221 .bss lcd_timer,1 ; lcd interval timer
28
29 ;--- Definieren der Konstanten
30 ;SPR-Special Funktion Register
31 00 IE1 .equ 0h ; Interrupt Freigabe-Register
32 01 IE2 .equ 01h ; Interrupt Freigabe-Register
33 02 IFG1 .equ 02h ; Interrupt Merk-Register
34 03 IFG2 .equ 03h ; Interrupt Merk-Register
35 04 ME1 .equ 04h ; Freigabe-Register der Module
36 05 ME2 .equ 05h ; Freigabe-Register der Module
37
38 ;LCD-Anzeige
39 30 LCDM .equ 030h ; LCD-Anzeige Steuer-Register
40 31 LCD1 .equ 031h ; Anfang der LCD-Tabelle, Digit 1
41
42 ;Basic Timer
43 80 BTME .set 080h ; BT Modul Freigabe-Bit im BTCTL-Register
44 80 BTIE .set 080h ; BT Interrupt Freigabe-Bit im IE-Register
45 80 BTIFG .equ 080h ; BT Interrupt Richtung-Bit im IFG-Register
46 40 BTCTL .equ 040h ; Adresse des BT Kontrol-Registers
47 42 TCCTL .equ 042h ; Adresse des Timer/Counter Kontrol-Registers
48 43 TCPLD .equ 043h ; Adresse des Timer/Counter pre-load-Registers
49 44 TCDAT .equ 044h ; Adresse des Timer/Counters
50
51 ;Watchdog Timer
52 0120 WDTCTL .equ 0120h ; Adresse des WDT Kontrol-Register
53 80 WDTHold .equ 80h ; WDT Stop-Bit im WDTCTL-Register
54 5a00 WDTPW .equ 05A00h ; Passwort mi WDTCTL-Register(High-Byte)
55 10 TMSEL .equ 010h ; Watchdog oder Timer Auswahl-Bit im WDTCTL-Reg.
56 08 CNTCL .equ 8h ; Neustart-Bit im WDTCTL-Register
57
58
59 ;General Interrupt Enable
60 08 GIE .equ 08h ; Freigabe aller Interrupts-Bit
61
62
63 ;P0-Eingang
64 10 P0IN .equ 010h ; Adresse des Eingabe-Registers
65 11 P0OUT .equ 011h ; Adresse des Ausgabe-Registers
66 12 P0DIR .equ 012h ; Adresse des Richtungs-Registers
67 13 P0IFG .equ 013h ; Adresse des Interrupt-Merker-Registers
68 14 P0IES .equ 014h ; Adresse des Flanken-Auswahl-Registers
69 15 P0IE .equ 015h ; Adresse des Interrupt-Freigabe-Registers
70
71 01 P0IN0 .set 001h ; P0.0 Eingabe-Bit im P0IN-Register
72 04 P0IE0 .set 004h ; P0.0 Interrupt-Freigabe-Bit im IE-Register
73 04 P0IFG0 .set 004h ; P0.0 Interrupt-Merker-Bit im IFG-Register
74 01 P0DIR0 .set 001h ; P0.0 Richtungs-Register-Bit im P0DIR-Register
75 01 P0IES0 .set 001h ; P0.0 Flanken-Auswahl-Bit im P0IES0-Register
76
77 e0 P_0_567 .equ 0E0h ; Bits 5, 6, und 7 setzen
78 ; 6 und 7 für Sensor und 5 für SVCC
79 ; siehe Schaltplan STK [SKM,1-10]
80
81 ;AD-Wandler
81 0110 AIN .equ 0110h ; Adresse des A/D-Wandler Eingang-Register AIN
82 0112 AEN .equ 0112h ; Adresse des A/D-Wandler A/D-Auswahl-Reg. AEN
83 0114 ACTL .equ 0114h ; Adresse des A/D-Wandler Steuerregister ACTL
84 0118 ADAT .equ 0118h ; Wert-Merk-Register
```

Anhang

```

85      04      ADIFG      .equ      04h      ; A/D-Wandler Interrupt-Merker-Bit im IFG-Reg.
86      01      CS          .equ      01h      ; Start der Konvertierung (Umwandlung)
87
88      08      A3          .equ      08h      ; A3 Eingang-Bit
89      02      SVCC       .equ      02h      ; Ein/Ausschalten der Referenzspannung (SVCC)
90
91      0800    RSELAUT    .equ      0800h    ; Automatische Bereichseinstellung-Bit im ACTL
92
93
94      ;*****
95      ; Initialisierungen der Register
96      ;*****
97
98 0240      .sect "MAIN",RAM_orig
99 0240      RESET
100 0240    403103de      MOV      #SP_orig,SP      ; Initialisieren des Stack Pointers
101
102 0244    40b25a800120  MOV      #(WDTHold+WDTPW),&WDTCCTL ; Anfangsadresse des Stack Pointers
103
104
105      ;Initialisieren der LCD-Anzeige und Basic Timer-Registers
106
107 024a    43f20030      MOV.B   #-1h,LCDM        ; LCD : Analog generator on
108
109
110
111 024e    40f200570040  MOV.B   #057h,BTCTL     ; LCD : Low impedance of AG
112
113
114
115 0254    d0f200800005  BIS.B   #BTME,ME2       ; LCD : 4Mux active
116 025a    c0f200400040  BIC.B   #040h,BTCTL     ; all outputs are Seg
117 0260    d0f200800001  BIS.B   #BTIE,IE2       ; Basic-Timer : SSEL=0 DIV=0 Reset=1
118 0266    -40f00080ffb7  MOV.B   #lcd_ival,lcd_timer ; ACLK
119 026c    +12b002bc      CALL    #loesch_LCD      ; 32768/256 = 128Hz (7.8ms debounce time)
120
121
122
123
124
125
126
127
128
129
130
131
132 0280    40f200e00012  MOV.B   #P_0_567,&P0DIR ; LCD-Grundfrequenz @4Mux: 64Hz
133 0286    40f200e00011  MOV.B   #P_0_567,&P0OUT  ; aktivieren des Basic-Timer-Moduls
134
135
136
137
138
139 0290    42e20001      MOV.B   #ADIFG,&IE2      ; löschen des Basic-Timer-Reset-Bits
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166

```

Anhang

```

167 ; * * * * *
168 ; Routinen
169 ; * * * * *
170
171 ;löschen der LCD ; Löschroutine
172
173 02bc loesch_LCD
174 02bc 1205 PUSH R5 ; retten des Inhalts von Register5(wird auf
175 ; dem STACK abgelegt)
176 02be 4235 MOV #8,R5 ; schiebe 8 in das Register5, für 8 Durchläufe
177 ; der Schleife (8-Stellen der LCD-Anzeige)
178 02c0 LCD_Marke
179 02c0 43c50030 MOV.B #0,LCD1-1(R5) ; Sprungmarke
180 ; 0 in das LCDM-Register, die Position steht
181 02c4 8315 DEC R5 ; im Register5
182 02c6 +23fc JNZ LCD_Marke ; decrementieren des Inhahts von Register5
183 ; solange Register5 nicht 0 springe zurück
184 02c8 4135 POP R5 ; zu Sprungmarke
185 ; Register5 wieder mit dem geretteten Wert
186 02ca 4130 RET ; beschreiben
187 ; Ende der Löschroutine
188
189 ;-----
190 ;Messung der Lichtintensität
191 02cc LICHT_MESS
192 02cc 40b208080114 MOV #A3+RSELAUT,&ACTL ; A/D-Wandler konfigurieren;
193 ; Eingang A3 und Automatische Bereichseinstellung
194 02d2 d3920114 BIS #CS,&ACTL ; Starten der A/D-Umwandlung
195 02d6 430d CLR R13 ; A/D-Flag löschen
196 02d8 WARTEN ; Warten bis der A/D-Wandler die Ausw. beendet
197 02d8 930d CMP #0,R13 ; Nach Beendigung der A/D-Umwandlung wird der
198 02da +27fe JZ WARTEN ; A/D-Flag in der ADISR mit dem Wert #1 belegt
199 ; und beendet die Schleife
200 02dc 4130 RET ; Ende der Routine
201
202 ;-----
203 ;Binäres Wert in eine BCD-Zahl umwandeln
204 02de BIN2BCD
205 02de 120f PUSH R15 ; Inhalt des Registers15 auf den Stack kopieren
206 02e0 120e PUSH R14 ; Inhalt des Registers14 auf den Stack kopieren
207 02e2 120d PUSH R13 ; Inhalt des Registers13 auf den Stack kopieren
208
209 02e4 403f0010 MOV #16,R15 ; R15 mit 16 belegen (16 Bit wandeln)
210 02e8 430e CLR R14 ; Register14 löschen
211 02ea 430d CLR R13 ; Register13 löschen
212 02ec 5c0c BIN1 RLA R12 ; Die Bits im R12 nach Links rotieren,
; nächstes Bit ins Carry
213 02ee ad0d DADD R13,R13 ; R13 mit sich selbst addieren
214 02f0 ae0e DADD R14,R14 ; R14 mit sich selbst addieren
215 02f2 831f DEC R15 ; Noch ein Bit??
216 02f4 +23fb JNZ BIN1 ;
217 ;BCD-Zahl auf der LCD-Anzeige anzeigen
218 02f6 DISBCD5
219 02f6 403f0031 MOV #LCDM+1,R15 ;
220 02fa 120d DIS1 PUSH R13 ; Inhalt des Registers13 auf den Stack kopieren
221 02fc 11110000 RRA 0(SP) ; Den Inhalt des Stacks (vorher R13)
222 0300 11110000 RRA 0(SP) ; um 4 Bits nach Rechts rotieren
223 0304 11110000 RRA 0(SP)
224 0308 11110000 RRA 0(SP)
225 030c f03d000f AND #0Fh,R13 ; mit 0000 0000 0000 1111 ausmaskieren
226 ; d.H. nur die letzten 4 Bits bleiben im R13
227 0310 +4ddf03880000 MOV.B LCD_Tab(R13),0(R15) ; die Zahl im R13 auf LCD anzeigen
228 0316 413d POP R13 ; R13 mit dem Inhalt des Stacks Beschreiben
229 0318 531f INC R15 ; R15 um 1 erhöhen
230 031a 903f0035 CMP #LCDM+5,R15 ; Wurden schon 4 Digits angezeigt?
231 031e +3bed JL DIS1 ; Wenn nicht, dann springe zurück
232 0320 f03e000f AND #0Fh,R14 ;
233 0324 +4edf03880000 MOV.B LCD_Tab(R14),0(R15) ;
234
235 032a 413d POP R13 ; Den Inhalt vom Stack holen
236 032c 413e POP R14 ; Den Inhalt vom Stack holen
237 032e 413f POP R15 ; Den Inhalt vom Stack holen
238 0330 4130 RET
239 ;-----
240 ;Binäres Wert in eine Hex-Zahl umwandeln und auf LCD-Anzeige ausgeben
241 0332 HEXAUS
242 0332 1205 PUSH R5 ; Inhalt des Registers5 auf den Stack kopieren
243 0334 120b PUSH R11 ; Inhalt des Registers11 auf den Stack kopieren
244
245 0336 430b CLR R11 ; Register11 löschen
246 0338 4225 MOV #4,R5 ; Register5 mit der Zahl 4 belegen (4 Stellig)
247 033a c312 CLRC ; Übertrag-Bit löschen
248 033c 5c0c AUSX RLA R12 ; links rotieren arithmetisch, Null ins LSB
249 033e 6b0b RLC R11 ; links rotieren, Carry ins LSB

```

Anhang

```

250 0340 5c0c          RLA    R12          ; ein Bit ins Carry
251 0342 6b0b          RLC    R11          ; Bit aus dem Carry ins R11
252 0344 5c0c          RLA    R12          ; und das alles 4 Mal, da eine Hexzahl (R11)
253 0346 6b0b          RLC    R11          ; aus 4 Bit besteht
254 0348 5c0c          RLA    R12          ; und diese Schleife 4 Mal, da wir 4 Zahlen
255 034a 6b0b          RLC    R11          ; ausgeben.
256 034c +4bd503880030 MOV.B  LCD_Tab(R11), LCDM(R5) ; Zahl aus R11 ausgeben
257 0352 430b          CLR    R11          ; lösche Register11
258 0354 8315          DEC    R5           ; eine Zeile weiter
259 0356 9305          CMP    #0,R5       ; 4 Zeichen ausgegeben?
260 0358 +23f1        JNZ    AUSX         ; wenn nicht, springe zurück zum AUSX
261
262 035a 413b          POP    R11         ; Den Inhalt vom Stack holen
263 035c 4135          POP    R5          ; Den Inhalt vom Stack holen
264 035e 4130          RET
265
266
267 ;-----
268 0360 TASTE_P_0
269 0360 430e          CLR    R14         ; P0-Flag (R14) löschen
270 0362 930e          T      CMP    #0,R14 ; solange P0-Flag nicht gesetzt (<>0)
271 0364 +27fe        JZ     T           ; springe zurück
272 0366 4130          RET
273
274
275 ; * * * * *
276 ; Interrupt Routinen
277 ; * * * * *
278
279 ;P0.0
280 0368 POISR
281 0368 c232          DINT          ; 0->GIE, alle anderen Interrupts werden ignoriert
282 036a 431e          MOV    #1,R14     ; P0-Flag setzen (=1)
283 036c 1205          PUSH   R5         ; Retten des Inhalts von R5 auf den Stack
284 ;Verzögerung um den P0-Schalter abzaprellen
285 036e 4335          MOV    #0FFFFh,R5 ; Große Zahl in R5
286 0370 Verzoegerung
287 0370 8315          DEC    R5         ; und runterzählen
288 0372 +23fe        JNZ    Verzoegerung;bis R5=0
289 0374 4135          POP    R5         ; R5 wieder mit dem Wert vom Stack beschreiben
290 0376 c2e20002        BIC.B  #POIFG0,&IFG1 ; Löschen des P0.0-Bits im Interrupt-Merk-Reg.
291 037a d232          EINT          ; 1->GIE, Freigabe der Interrupts
292 037c 1300          RETI          ; Ende der WDTISR
293
294 ;AD-Wandler
295 037e ADISR
296 037e c232          DINT          ; 0->GIE, alle anderen Interrupts werden ignoriert
297 0380 431d          MOV    #1,R13     ; A/D-Flag setzen (=1)
298 0382 d232          EINT          ; 1->GIE, Freigabe der Interrupts
299 0384 1300          RETI
300
301 ;Basic Timer
302 0386 BTISR
303 0386 1300          RETI
304
305 ; * * * * *
306 ; LCD Definitionen
307 ; * * * * *
308
309 0388 LCD_TYPE          ; STK/EVK LCD
310 01          a      .equ    01h
311 02          b      .equ    02h
312 10          c      .equ    10h
313 04          d      .equ    04h
314 80          e      .equ    80h
315 20          f      .equ    20h
316 08          g      .equ    08h
317 40          h      .equ    40h
318
319 ;--- Character-Zeichen definieren
320
321 0388 b7          LCD_Tab .byte  a+b+c+d+e+f ; "0", Aufruf über LCD_Tab(0)
322 0389 12          .byte  b+c ; "1", Aufruf über LCD_Tab(1)
323 038a 8f          .byte  a+b+d+e+g ; "2", Aufruf über LCD_Tab(2)
324 038b 1f          .byte  a+b+c+d+g ; "3", Aufruf über LCD_Tab(3)
325 038c 3a          .byte  b+c+f+g ; "4", Aufruf über LCD_Tab(4)
326 038d 3d          .byte  a+c+d+f+g ; "5", Aufruf über LCD_Tab(5)
327 038e bd          .byte  a+c+d+e+f+g ; "6", Aufruf über LCD_Tab(6)
328 038f 13          .byte  a+b+c ; "7", Aufruf über LCD_Tab(7)
329 0390 bf          .byte  a+b+c+d+e+f+g ; "8", Aufruf über LCD_Tab(8)
330 0391 3f          .byte  a+b+c+d+f+g ; "9", Aufruf über LCD_Tab(9)
331 0392 bb          .byte  a+b+c+e+f+g ; "A", Aufruf über LCD_Tab(10)
332 0393 bc          .byte  c+d+e+f+g ; "b", Aufruf über LCD_Tab(11)
333 0394 a5          .byte  a+d+e+f ; "C", Aufruf über LCD_Tab(12)

```

Anhang

```
334 0395 9e          .byte  b+c+d+e+g          ; "d", Aufruf über LCD_Tab(13)
335 0396 ad          .byte  a+d+e+f+g          ; "E", Aufruf über LCD_Tab(14)
336 0397 a9          .byte  a+e+f+g            ; "F", Aufruf über LCD_Tab(15)
337
338 0398          LCD_Tab_End
339
340
341                ; * * * * *
342                ; Interrupt Vektoren-Tabelle
343                ; * * * * *
344
345 03e0          .sect      "Int_Vect",USER_END-31
346                ; Anfang der Tabelle befindet sich 31 Speicherplätze
347                ; vor dem Ende des Speichers
348 03e0 +0240      .word    RESET
349 03e2 +0386      .word    BTISR   ; Basic Timer
350 03e4 +0240      .word    RESET
351 03e6 +0240      .word    RESET
352 03e8 +0240      .word    RESET
353 03ea +037e      .word    ADISR           ; EOC from ADC
354 03ec +0240      .word    RESET
355 03ee +0240      .word    RESET
356 03f0 +0240      .word    RESET
357 03f2 +0240      .word    RESET
358 03f4 +0240      .word    RESET
359 03f6 +0240      .word    RESET
360 03f8 +0240      .word    RESET
361 03fa +0368      .word    P0ISR   ; P0.0
362 03fc +0240      .word    RESET
363 03fe +0240      .word    RESET
364                .end
```

No Errors, No Warnings