Anleitungen zum MCT Laborprojekt

Zweck dieses Dokument ist, den Studenten Hinweise für eine erfolgreiche Durchführung des Laborprojekts im Fach Mikrocomputertechnik zu geben.

Inhaltsverzeichnis

1. Voraussetzungen....2
2. Designprozess....2
3. Programmentwicklung und Programmtest....3
   3.1. Die µ-Vision Entwicklungsumgebung....3
   3.2. Organisation : Projekte und Ordner....5
   3.3. Das erste Projekt....5
   3.4. Mitgelieferte Dateien....6
4. Vollständige Simulation....7
   4.1. Simulation des Laborkoffers....7
   4.2. Simulation des Motors....9
   4.3. Beispiel Programmtest....9
5. Projektfahrplan....11
   5.1. Projekt 1 : P1OnOffManuell (siehe auch 3.3 Das erste Projekt)....11
   5.2. Projekt 2 : P1OnOffBlink....11
   5.3. Projekt 3 : PotiToP1....12
   5.4. Projekt 4 : PotiToLCD....12
   5.5. Projekt 5 : T2SimMotor....12
   5.6. Projekt 6 : T2SimToDZ....13
   5.7. Projekt 7 : Kennlinie....13
   5.8. Projekt 8 : Steuerung....14
   5.9. Projekt 9 : Regelung....14
6. Fallstricke und Probleme....14
   6.1. Probleme mit dem Stack....14
   6.2. Fehlende SFR Symboldefinitionen....15
   6.3. Überschreiben des Monitorcodes....15
   6.4. LCD Funktionen in Interrupt Service Routinen....16
   6.5. Endlose ISR Schleifen....16
   6.6. Watchdog Timer....16
   6.7. Warnings beim Link Prozess....16
7. Modularität des Programms....17
   7.1. Schnittstelle zwischen Assembler und C-Code....17
   7.2. Programmstruktur....18
8. Dokumentation....19
   8.1. Dokumentationsbeispiel....20

1. Voraussetzungen

Das Ziel des Laborprojektes ist es, die Drehzahl eines kleinen Motors mit einem Programm zu steuern. Das Programm wird dazu auf einem 80c515c Mikroprozessor ausgeführt, der zusammen mit anderen Komponenten auf einer Platine im sog. Laborkoffer montiert ist.

Für die Erstellung und das Austesten des Steuerprogramms wird ein komfortables PC Programm, die Entwicklungsumgebung **µ-Vision 4** der Firma **KEIL** benutzt. Es ist also ein PC erforderlich, auf dem dieses Programm installiert ist. Der PC muss eine serielle Schnittstelle aufweisen, denn er wird über diese Schnittstelle mit dem Laborkoffer verbunden.

Um das Laborprojekt sinnvoll bearbeiten zu können, ist es einerseits erforderlich, dass Kenntnisse über zu verwendende Einrichtungen des Prozessors, die Möglichkeiten der Programmierung des Prozessors und die Entwicklungsumgebung vorhanden sind :

- Timer 0/1
- Timer 2
- Interruptlogik
- 8051 Assembler Programmierung
- Funktionsweise und Verwendung von Interrupt Service Routinen
- KEIL µ-Vision 4 Entwicklungsumgebung.

Andererseits ist es erforderlich, sich eigenständig Kenntnisse über die nicht in der Vorlesung behandelten Themen anzueignen (mit einer solchen Situation werden Sie auch bei der Ausübung Ihres Berufs konfrontiert werden) :

- A/D Wandler durch Studium des entsprechenden Kapitels im 8051 User's Manual.
- Aufgabenstellung incl. der Funktionsweise des Motors.

2. Designprozess

Ausgehend von der Aufgabenstellung kann und sollte der Design durchgeführt werden, ohne die KEIL Entwicklungsumgebung zu benutzen. Die Erfahrung hat gezeigt, dass Designfehler zeitaufwendiger zu beheben sind als z.B. Codierfehler. Es empfiehlt sich daher, viel Sorgfalt in den Design zu legen. Auch sollte man mit der Codierung erst beginnen, wenn der Design abgeschlossen ist.

Für den Design empfiehlt sich die Top-Down Methode. D.h., man beginnt mit einer groben Struktur des Projekts und kommt dann schrittweise zu immer detaillierteren Festlegungen. Die nachfolgend aufgeführten Schritte zeigen eine mögliche Vorgehensweise für den Design.

1. Identifizieren der notwendigen Funktionsblöcke (z.B. Ermitteln der Solldrehzahl, Anzeige der Drehzahlen, usw.).
2. Identifizieren der Funktionen innerhalb der einzelnen Funktionsblöcke (Wichtig : In dieser Stufe soll festgelegt werden, **was** erledigt werden soll, nicht wie etwas erledigt werden soll!). Einzelheiten über geforderte Funktionen finden sich bereits in der Aufgabenstellung (z.B. die angezeigte Ist-Drehzahl soll der Mittelwert von 32 oder 64 gemessenen Werten sein.)
3. Festlegen der für die Kommunikation zwischen den Funktionsblöcken notwendigen globalen Variablen (z.B. auf welche Weise übermittelt der Funktionsblock 'Ist-Drehzahl ermitteln' den Wert der Drehzahl oder den Wert der Rotationszeit an den Funktionsblock 'Drehzahlen anzeigen' ?)
4. Festlegen der wichtigen Variablen, einschließlich der Angabe der Datentypen (8 Bit Wert, 16 Bit Wert, ...) und Abschätzung des jeweiligen Wertebereichs, die die Variablen annehmen können.
5. Festlegen der Befehlssequenzen und Parameterwerte für die Steuerung der prozessorinternen Komponenten, und zwar für die Initialisierungphase und die Betriebsphase (z.B. die SFR Werte für die Initialisierung des Timer 0, Befehle zur Anzeige einer Drehzahl auf dem LCD, usw.).
6. Erarbeitung der obersten Ebene des Kontrollflusses des Programms (Dokumentation z.B. als Flowchart). Das schließt auch die Angaben mit ein, zu welcher Routine die einzelnen Funktionen zugewiesen werden : zu der Initialisierungsroutine, den ISRs, der Hauptprogrammroutine in Assemblercode oder den C-Code Routinen.
7. Detaildesign der einzelnen Routinen incl. der Eingangs- und Ausgangsvariablen, sowie der in der Routine verwendeten Variablen. Der Verwendungszweck der Variablen sollte erläutert werden. Diese Angaben sollten sich dann im Programm als Kommentare wieder finden.

Es empfiehlt sich, bereits während des Designs am Laborbericht zu arbeiten. Die Festlegungen und Ergebnisse, die sich während der oben genannten Designschritte ergeben, sind neben den Programmlisten die wesentlichen Elemente für den Laborbericht. Eine frühzeitig begonnene Dokumentation hat naturgemäß am Anfang noch viele Lücken. Diese Lücken zeigen aber auch, an welchen Stellen noch Designarbeit erforderlich ist und

helfen so, den Designprozess zu organisieren.

3. Programmentwicklung und Programmtest

3.1. Die µ-Vision Entwicklungsumgebung

Für die Erstellung des Programmcodes und das Testen des Programms wird die Entwicklungsumgebung µ-Vision 4 verwendet. Diese Programmpaket biete zwei Modi an :

- Im **Edit-Modus** kann Programmcode editiert, also neu geschrieben oder geändert werden, und der so erzeugte Quellcode kann durch Assemblieren bzw. Compilieren in ein ausführbares Maschinenprogramm übersetzt werden.

Im **Debug-Modus** kann das Maschinenprogramm zu Testzwecken ausgeführt werden. Dabei gibt es mehrere Optionen der Befehlsausführung :

- ohne anzuhalten (im sog. Run Mode),
- schrittweise, immer ein Befehl nach einem Tastendruck,
- abschnittsweise, durch Setzen von sog. Break Points.

Die Umschaltung zwischen Edit-Modus und Debug Modus erfolgt über

Debug → Start/Stop Debug Session

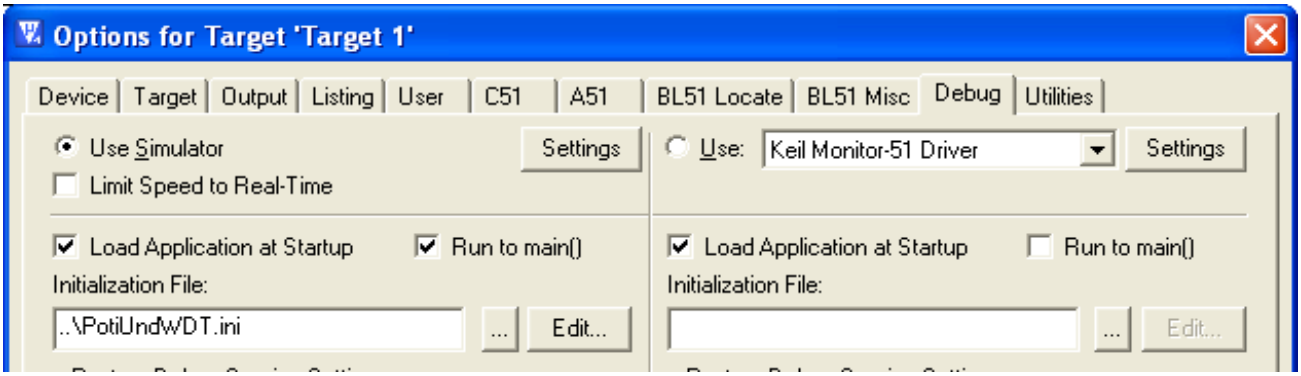
oder die entsprechende Schaltfläche in der Werkzeugleiste.

Die Ausführung des Programms kann im eingebauten **Simulator** oder auf dem **Prozessor des Laborkoffers** erfolgen.

- Der Simulator ist ein Programm innerhalb der µ-Vision 4 Entwicklungsumgebung. Die Programmausführung läuft damit vollständig im PC ab.
- Wird das Programm im Laborkoffer ausgeführt, lädt der PC beim Umschalten in den Debug-Modus das zu testende Maschinenprogramm in den Programmspeicher des Laborkoffers. Danach steuert er nur noch die Programmausführung im Mikroprozessor (Run Mode; Einzelschritt, usw.).

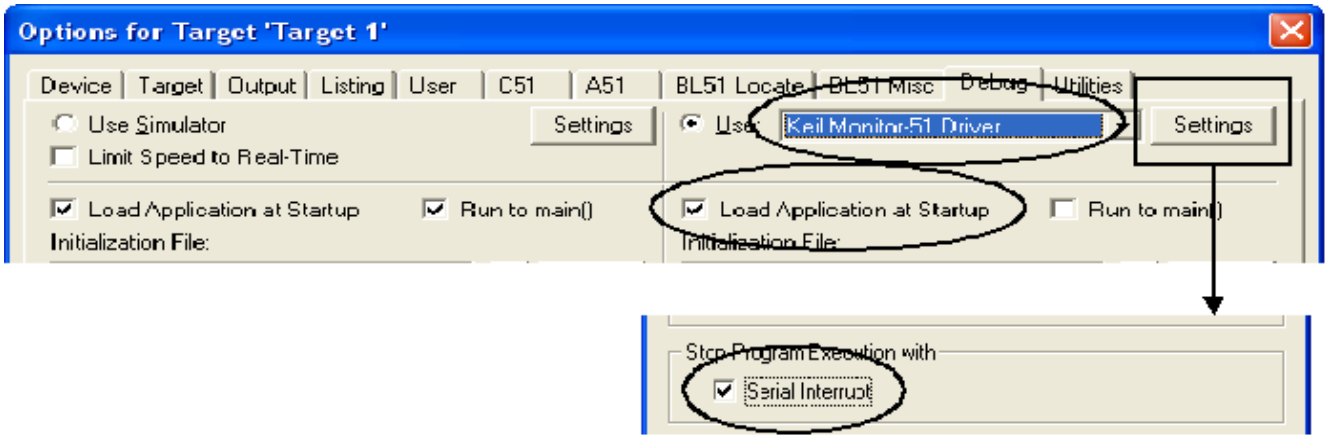
Für den Benutzer sieht die Programmausführung auf dem Simulator und dem Laborkoffer praktisch gleich aus.

Der **Simulatorbetrieb** ist in µ-Vision voreingestellt. Er kann auch aktiviert werden, indem man über '*Project -> Options for Target Target 1*' die Registerkarte '*Debug*' öffnet und '*Use Simulator*' anclickt. (Falls beim Öffnen von 'Project' die Auswahl 'Options for Target 'Target 1' nicht angeboten wird, ist im Project Workspace Fenster zuerst der Eintrag Target1 zu selektieren).



Bei Verwendung des Simulators sollte auch eine .ini-Datei benutzt werden, die den Watchdog Timer deaktiviert (siehe Kapitel Fallstricke und Probleme). Die Datei PotiUndWDT.ini enthält die Spezifikation für die WDT Deaktivierung.

Der **Laborkofferbetrieb** wird aktiviert, indem man über die Selektion '*Project -> Options for Target Target 1*' auf der Registerkarte '*Debug*' die im folgenden Screenshot hervorgehobenen Einstellungen vornimmt.



3.2. Organisation : Projekte und Ordner

Die Komponenten, die auf dem Weg zu einem ausführbaren Programm entstehen, fasst µ-Vision im sog. **Projekt** zusammen. Beim Anlegen eines neuen Projekts erzeugt µ-Vision die Projektdatei, eine Windows-Datei mit der Dateierweiterung .uvproj. Dabei muss der Benutzer angeben, in welchem **Ordner** diese Datei gespeichert werden soll. Mit fortschreitender Arbeit an dem Projekt entstehen weitere Dateien, die µ-Vision in dem Ordner ablegt, in dem auch die Projektdatei liegt. Um den Überblick zu behalten, empfiehlt es sich, für **jedes Projekt einen eigenen Ordner** anzulegen. Sinnvollerweise speichert man auch die Dateien, die den Quellcode enthalten im selben Ordner ab.

Im Falle eines umfangreicheren Projekts erfolgt die Entwicklung in mehreren Stufen. Dabei ist es sinnvoll, jede einzelne Stufe als eigenständiges Projekt durchzuführen, und jedes dieser Projekte in einem eigenen Ordner abzuspeichern. Man arbeitet also an einem

Projekt, bis ein gewünschter Stand erreicht ist, friert dann diesen Stand ein, indem man diesen Stand (dieses Projekt) in einen neuen Ordner kopiert, und arbeitet mit dem Projekt im neuen Ordner weiter. Auf diese Weise lassen sich Fehler durch Vergleich des aktuellen Projekts mit dem letzten Stand eventuell leichter lokalisieren.

Zusammen mit der Aufgabenstellung erhalten Sie auf einem Speicherstick eine Ordnerstruktur, die für die Erledigung der Laboraufgabe vorbereitet ist. In einem Hauptordner (z.B. *MeineProjekte*) sind bereits Ordner für alle Projekte angelegt, die Sie im Zug der Laborarbeit erledigen sollen. Darüber hinaus gibt es noch einige Dateien, die in allen Projekten gleichermaßen gebraucht werden. Diese sind auch im Hauptordner gespeichert und können bei Bedarf in den einzelnen Projektordnern referenziert werden.

3.3. Das erste Projekt

In diesem Kapitel wird anhand eines primitiven Programmbeispiels vorgeführt, wie in der µ-Vision Entwicklungsumgebung ein Programm entsteht, und wie es dann ausgeführt werden kann. Das Programmbeispiel ist die erste Stufe der Projekte, die sie erledigen sollen (siehe 5. Projektfahrplan). Wenn Sie also dieses Beispiel nachvollziehen, sollten Sie es im Projektordner *Port1OnOffManuell* abspeichern.

Mit der Befehlssequenz

```
            CSEG        AT      0
LOOP :
            MOV         P1,#0xFF
            MOV         P1,#0x00
            SJMP        LOOP

            END
```

werden die Bits von Port1 an- und wieder abgeschaltet. Im Simulatorbetrieb kann man sich die Portbits in einem separaten Fenster anzeigen lassen. Im Laborkoffer werden die Portbits über Dioden angezeigt. Durch Ausführen der Sequenz im Einzelschrittverfahren kann überprüft werden, ob das Programm funktioniert.

Über *Projekt -> Neu* in µ-Vision 4 wird nun ein neues Projekt angelegt. Im zugehörigen Dialog

- geben Sie dem Projekt einen beliebigen Namen,
- geben Sie als Speicherplatz für das Projekt den Ordner *Port1OnOffManuell* an,
- wählen Sie als Prozessortyp den **8051-C515C-L** von Infineon aus, und
- lehnen Sie das Angebot, den Standard 8051 Startup Code zu übernehmen, ab.

Damit haben Sie das standardmäßige **Target 1** mit der **Source Group 1** definiert. Die Source Group 1 ist der Container, in dem alle Programmteile des Projekts aufgesammelt werden.

Mit *Datei -> Neu* öffnen Sie ein Editor-Fenster, in dem Programmbefehle eingegeben werden können. Geben Sie die o.g. Befehle in diesem Fenster ein und speichern Sie (Speichern unter ...) das eingegebene Programm z.B. unter dem Dateinamen *Haupt.a51* in den neu erstellten Ordner ab. Durch Angabe der Dateierweiterung .a51 schaltet der Editor die Syntaxhervorhebung für den Assemblercode ein.

Um die Programmdatei in das Projekt zu integrieren, muss sie zur Source Group 1 hinzugefügt werden. Durch Rechtsclick auf Source Group 1 öffnet sich ein Auswahlfenster, in dem Sie den Eintrag *Add Files to Group 'Source Group 1'* auswählen. In dem sich öffnenden Fenster addieren Sie Haupt.a51 zur Source Group und schließen das Fenster wieder. Gegebenenfalls muss die Auswahl des Dateityps von .c auf .a* umgeschaltet werden, um die wählbaren Assemblerprogramme im Auswahlfenster sichtbar zu machen.

Jetzt kann das Programm assembliert werden. Das geschieht z.B. über die Auswahl *Project → Rebuild all Target Files*. Sollte kein Fehler aufgetreten sein, kann man vom Editiermodus in den Debugmodus umschalten und das Programm ausführen.

Im Simulatorbetrieb kann man sich über *Peripherals → I/O Ports → Port1* die Port Bits anzeigen lassen, die sich bei schrittweisem Ausführen des Programms entsprechend ändern sollten. Im Laborkofferbetrieb sollten die Port1 Dioden entsprechend ein- und ausgeschaltet werden.

Wenn das alles funktioniert, haben Sie erfolgreich Ihr erstes 8051 Programm geschrieben und ausgeführt.

3.4. Mitgelieferte Dateien

Um den Einstieg in die Programmierung zu erleichtern, werden auf einem Speicherstick Dateien zu Verfügung gestellt, die auf den privaten Datenbereich der Studenten (H:\) kopiert werden sollten. So stehen sie während des Programmierens permanent zur Verfügung.

Im **Ordner Meine Projekte** sind enthalten :

- **9 Unterordner** für die zu bearbeitenden Projekte (siehe 5. Projektfahrplan)
- **lcd4x20duo.c** : C-Routinen zur Steuerung des LCDs im Laborkoffer und des simulierten LCDs.
- **lcd4x20v2.inc** : EXTRN Deklarationen für die C-Funktionsnamen dieser LCD Funktionen.
- **PotiUndWDT.ini** : Simuliertes Potentiometer + Watchdog Timer Deaktivierung..
- **MotorModell.a51** : Simulationsmodell des Motors

Dazu gibt es noch folgende Dokumentdateien :

- **KeilDemo.doc** : Das verteilte Dokument mit den KEIL EU Screenshots.
- **LaborAufgabe2013.pdf** : Softcopy der Aufgabenstellung des Laborprojekts
- **LaborAnleitung2013.pdf** : Softcopy dieses Dokuments

4. Vollständige Simulation

Für die in der Laboraufgabe geforderte Steuerung des Motors wird außer dem PC mit der Entwicklungsumgebung der Motor selbst und dann noch der Laborkoffer benötigt. Da diese zusätzlichen Geräte nur in den Räumen der DHBW Stuttgart zur Verfügung stehen, kann das Testen des Programms eigentlich nur dort stattfinden. Um von den Geräten unabhängig zu werden, wurde der Simulator durch zusätzliche Funktionen so erweitert, dass die Programmentwicklung auch ohne die Geräte, also vollständig im PC erfolgen kann. Das Ausführen des Programms im Simulator hat den weiteren Vorteil, dass µ-Vision Werkzeuge verwendet werden können, die im Laborkofferbetrieb nicht verfügbar sind (z.B. der Logic Analyzer).

4.1. Simulation des Laborkoffers

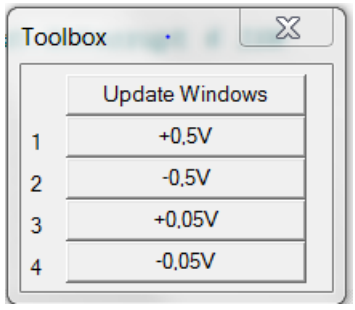
Die physikalischen Komponenten im Laborkoffer, die simuliert werden müssen, sind

- die Port_Leuchtdioden,
- ein Potentiometer,
- das LCD.

Der Wert der Portbits kann direkt über die eingebaute µ-Vision Funktion *Peripherals → I/O Ports → Port x* angezeigt und geändert werden.

µ-Vision ermöglicht den programmierten Zugriff auf den Eingangswert des A/D-Wandlers über die VTREGs AIN0... AIN7. Für unsere Zwecke wird nur das VTREG AIN0 verwendet, das dem **Potentiometer 0** im Laborkoffer entspricht. In der zur Verfügung gestellten Initialisierungsdatei für die Simulation **PotiUndWDT.ini** sind Programmfunktionen enthalten, mit denen der Eingangswert des A/D-Wandlers vergrößert und verkleinert werden kann. Floating Point Werte zwischen 0 und 5 sind gültig. Das entspricht einer Eingangsspannung zwischen 0 V und 5 V und erzeugt im A/D-Wandler Register ADDATH digitale Werte zwischen 0 und 255.

Die Programmfunktionen werden über Buttons aktiviert, die die .ini-Datei im Toolbox-Window (s.u.) anlegt. Das Toolbox-Window kann ggf. über *View → Toolbox-Window* geöffnet werden. Der an den A/D-Wandler angelegte Eingangswert kann über *Peripherals → A/D Converter* sichtbar gemacht werden.



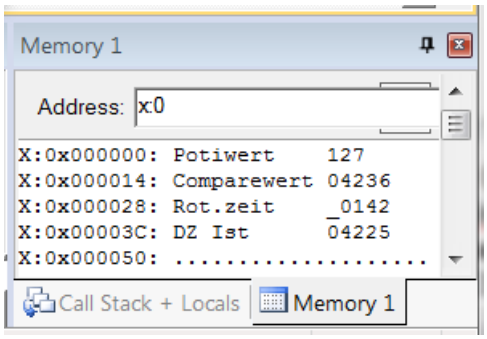
Das **LCD** wird mit Hilfe eines Memory Windows und 80 Bytes im XRAM simuliert. Eine neue Version des LCD Programms (**lcd4x20duo.c**) wird zur Verfügung gestellt. Dieses Programm enthält sowohl die standardmäßigen LCD Funktionen für den

Laborkofferbetrieb, als auch neue LCD Funktionen für den Simulatorbetrieb. Die Schnittstelle ist in beiden Fällen gleich. Das Programm findet heraus, ob im Laborkofferbetrieb oder im Simulatorbetrieb gearbeitet wird und verwendet die entsprechenden Funktionen.

In den neuen Funktionen werden die Anzeigedaten in den ersten 80 Bytes des XRAM gespeichert. Um die Daten in einer ansprechenden Form anzuzeigen,

- öffnet man ein **Memory Window**,
- gibt als Adresse **x:0** ein,
- schaltet **ASCII Mode** Anzeige ein,
  - Rechtsklick irgendwo im Datenbereich
  - Auswahl Ascii
- stellt die **Fensterbreite** auf **20 Zeichen** ein.

Auf diese Weise hat das Memory Window das gleiche Aussehen wie das LCD (s.u.).



Ob mit dem echten Laborkoffer oder mit dem simulierten Laborkoffer gearbeitet wird, ist vollkommen transparent für die Programmentwicklung. Das Umschalten zwischen den beiden Betriebsarten geschieht nach wie vor wie oben beschrieben. Lediglich im Feld Initialization File für den Simulatorbetrieb muss PotiUndWDT.ini eingetragen werden, und das LCD Programm lcd4x20duo.c muss gelinkt werden.

4.2. Simulation des Motors

Der Motor wird mit Hilfe eines regulären Assemblerprogramms simuliert. Dieses Programm benutzt den Timer 1 für die Erzeugung der Tachopulse des simulierten Motors. Das Programm besteht aus einem Initialisierungsteil, einer ISR für den Timer 2 Compare Interrupt und einer ISR für den Timer 1 Interrupt. Wenn der Timer 2 Compare Interrupt auftritt, liest das Programm den Timer 2 Wert aus und berechnet über eine fest eingebaute Kennlinie die dazu gehörige Rotationszeit. Mit diesem Wert wird der Timer 1 betrieben. Dieser erzeugt nach jeder simulierten Umdrehung eine Externen Interrupt 1 Request.

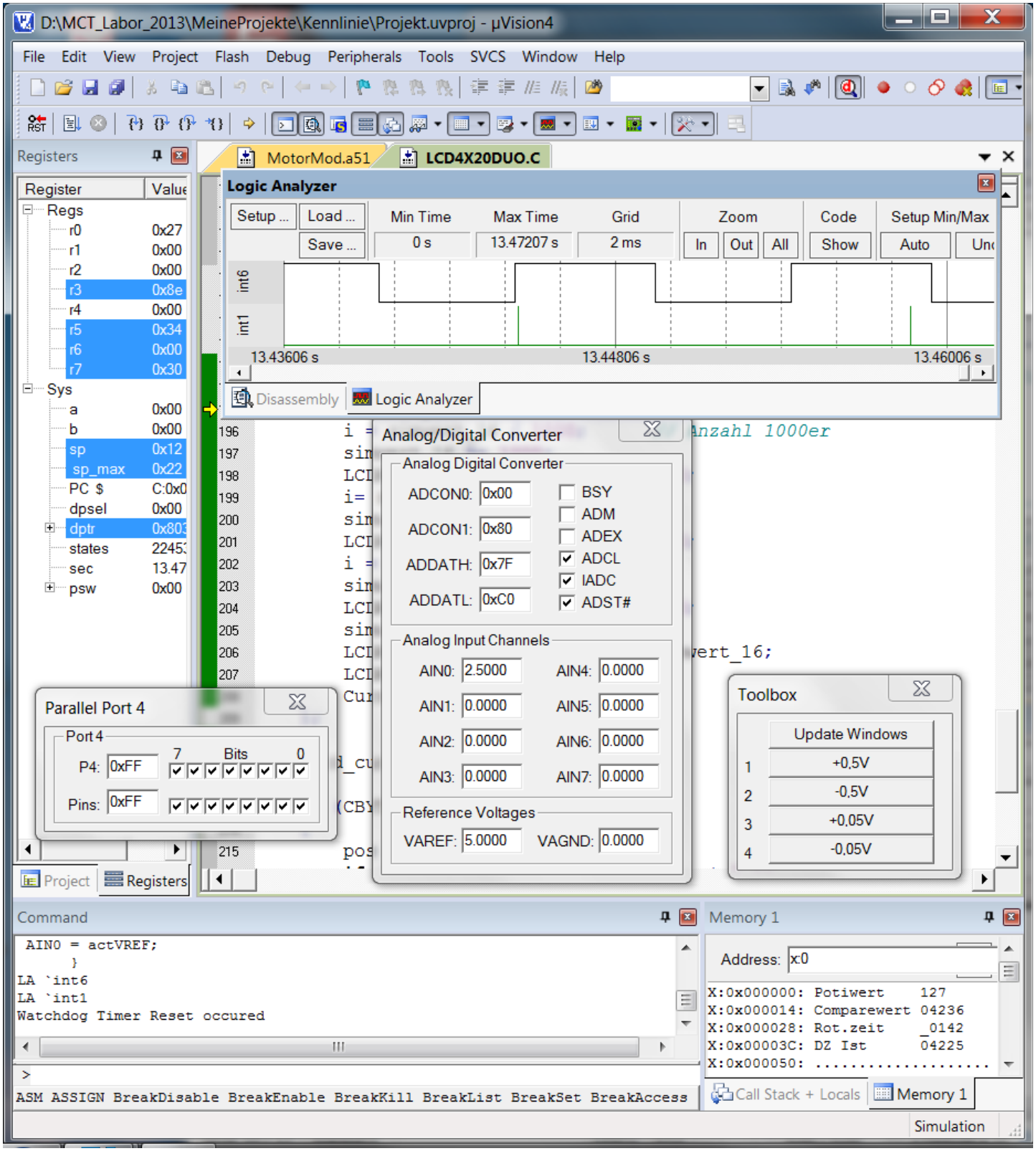
Das Programm, das den Motor simuliert, wird in einer eigenen Assembler Quelldatei gehalten. Dieses Programm muss dann zusätzlich assembliert und mitgelinkt werden. Wegen der Initialisierungsbefehle für den simulierten Motor lässt sich dieser Code leider nicht vollkommen transparent gestalten. An der Adresse 0 (direkt nach RESET) sollte der Befehl 'LJMP MotorModInit' stehen. Damit wird direkt zur Initialisierung des simulierten

Motors verzweigt. Am Ende dieser Initialisierung führt der Code den Befehl 'LJMP NormalInit' aus. Bei diesem Symbol sollte dann die Initialisierung des Testprogramms beginnen.

Der Code für den simulierten Motor überprüft auch, ob im Laborkofferbetrieb oder im Simulatorbetrieb gearbeitet wird und aktiviert den Simulator entsprechend oder auch nicht. Es ergibt sich so noch ein interessanter Mischbetrieb. Es ist möglich mit dem Laborkoffer und einem simulierten Motor zu arbeiten. In diesem Fall wird der Code für den simulierten Motor auf dem Prozessor des Laborkoffers ausgeführt. Dieser Betrieb kann über den Mikroschalter Port 4.7 im Laborkoffer aktiviert werden.

4.3. Beispiel Programmtest

Das folgende Bild zeigt ein µ-Vision Fenster, wie es beim Testen eines Programms im Simulator aussehen könnte. Das Toolbox Window zeigt die Schalter zum Verändern der Eingangsspannung des A/D-Wandlers. Das Memory Window 1 zeigt den LCD Inhalt. Und im Logic Analyzer wird gerade das PWM Signal (int6) und das Tachosignal des Motors (int1) angezeigt.


D:\MCT_Labor_2013\MeineProjekte\Kennlinie\Projekt.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register Value
Regs
r0 0x27
r1 0x00
r2 0x00
r3 0x8e
r4 0x00
r5 0x34
r6 0x00
r7 0x30
Sys
a 0x00
b 0x00
sp 0x12
sp_max 0x22
PC $ C:0x0
dpsel 0x00
dptr 0x80
states 2245
sec 13.47
psw 0x00
MotorMod.a51
LCD4X20DUO.C
Logic Analyzer
Setup ... Load ... Save ...
Min Time 0 s
Max Time 13.47207 s
Grid 2 ms
Zoom In Out All
Code Show
Setup Min/Max Auto
int6
int1
13.43606 s
13.44806 s
13.46006 s
Disassembly
Logic Analyzer
Analog/Digital Converter
Analog Digital Converter
ADCON0: 0x00
ADCON1: 0x80
ADDATH: 0x7F
ADDATL: 0xC0
BSY
ADM
ADEX
ADCL
IADC
ADST#
Analog Input Channels
AIN0: 2.5000
AIN1: 0.0000
AIN2: 0.0000
AIN3: 0.0000
AIN4: 0.0000
AIN5: 0.0000
AIN6: 0.0000
AIN7: 0.0000
Reference Voltages
VAREF: 5.0000
VAGND: 0.0000
Parallel Port 4
Port 4
P4: 0xFF
Pins: 0xFF
7 Bits 0
Toolbox
Update Windows
1 +0,5V
2 -0,5V
3 +0,05V
4 -0,05V
Project Registers
Command
AIN0 = actVREF;
}
LA `int6
LA `int1
Watchdog Timer Reset occured
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess
Memory 1
Address: x:0
X:0x000000: Potiwert 127
X:0x000014: Comparewert 04236
X:0x000028: Rot.zeit 0142
X:0x00003C: DZ Ist 04225
X:0x000050: .....
Call Stack + Locals
Memory 1
Simulation

5. Projektfahrplan

Um eine strukturierte Durchführung des Laborprojekts zu unterstützen, wird in diesem Dokument eine Folge von µVision 4 Projekten vorgeschlagen. Die Reihenfolge dieser Projekte ist so angelegt, dass der Benutzer zunächst in einer Reihe von Miniprojekten, die jeweils nur aus einer kleinen Anzahl von Programmbefehlen bestehen, schrittweise die KEIL Entwicklungsumgebung, die Programmierung der im 8051 Prozessor und der im Laborkoffer vorhandenen Komponenten, sowie der weiteren zur Verfügung gestellten Programmkomponenten kennenlernt. Die so erstellten und getesteten Routinen können dann in den umfangreicheren Projekten, die in der Aufgabenstellung verlangt werden, wieder verwendet werden.

Es wird dringend empfohlen, diesen Fahrplan einzuhalten. Die auf einem Speicherstick gelieferte Ordnerstruktur (siehe 3.2 Projekte und Ordner) ist nach diesem Fahrplan aufgebaut.

5.1. Projekt 1 : P1OnOffManuell (siehe auch 3.3 Das erste Projekt)

- Funktion : Mit der nachfolgenden Befehlsfolge werden die Port1 Bits ein- und wieder ausgeschaltet. Wenn man diese Schleife im Einzelschritt durchführt, kann man im Simulator und im Laborkoffer die Änderung der Port 1 Indikatoren beobachten.

```
        CSEG    AT    0      ;Absolutes Codesegment bei Adresse 0
Start:
        MOV     P1,#0        ;Port1 Dioden aus
        MOV     P1,#0xFF     ;Port1 Dioden ein
        SJMP    Start        ;bildet die Endlosschleife
        END                  ;signalisiert das Ende des Quellcodes
```

- Ziel : Bei dieser Aufgabe geht es um das Kennenlernen der KEIL Entwicklungsumgebung :
  - Ein neues Projekt wird angelegt.
  - Code wird editiert.
  - Ein Programm wird umgewandelt (assembliert und gelinkt).
  - Ein Programm wird ausgeführt.
  - Es wird zwischen Simulator- und Laborkoffermodus umgeschaltet.
  - Anzeige der Port Bits im Simulator (*Peripherals → I/O Ports → Port 1*)

5.2. Projekt 2 : P1OnOffBlink

- Funktion : Die Port 1 Bits sollen im 1 Sekunden Takt ein- und ausgeschaltet werden. Mit dem Timer 0 und der zugehörigen Interrupt Service Routine (ISR) wird ein Bit jede Sekunde umgeschaltet. Im Hauptprogramm wird dieses Bit abgefragt und die Portbits entsprechend ein- bzw. ausgeschaltet.
- Ziel : Kennenlernen der Bestandteile eines Programms : Interrupt Vektor –

Initialisierung – ISRs – Hauptprogramm und die Programmierung von Hardwarekomponenten.

- Benutzung des Stacks.
- Programmierung des Timer 0 : Initialisierung und ISR.
- Programmierung des Interrupt Systems.
- Debug von zeitabhängigem Code mit Hilfe des Logic Analyzers (Signal TF0) (*View → Analysis Windows → Logic Analyzer*)
- Kommunikation zwischen ISR und Hauptprogramm

5.3. Projekt 3 : PotiToP1

- Funktion : Der auf einem Potentiometer eingestellte Wert wird auf den Port 1 Indikatoren angezeigt. Nachdem der A/D Wandler in der Initialisierung betriebsbereit gemacht wurde, wird in jeder Schleife des Hauptprogramms der Potiwert digitalisiert und auf Port 1 angezeigt.
- Ziel :
  - Programmierung des A/D Wandlers : Initialisierung und Betrieb im Hauptprogramm.
  - Benutzung des simulierten Potis (*View → Toolbox Window*).
  - Anzeige des simulierten Potiwertes (*Peripherals → A/D Converter*).

5.4. Projekt 4 : PotiToLCD

- Funktion : Der auf einem Potentiometer eingestellte Wert wird auf dem LCD angezeigt. Das LCD wird mit den Funktionen lcd_init und lcd_clr initialisiert. In jeder Schleife des Hauptprogramms wird dann der Potiwert digitalisiert und samt der Überschrift 'Potiwert :' auf dem LCD angezeigt.
- Ziel :
  - Programmierung des LCDs : Initialisierung und Betrieb im Hauptprogramm.
  - Zusätzliche Benutzung von C Code
    - Parameterübergabe in Registern
  - Benutzung des simulierten LCDs (*View → Memory Window → x:0; Zeichendarstellung ASCII; Fensterbreite 20 Zeichen*).

5.5. Projekt 5 : T2SimMotor

- Funktion : Der Timer 2 wird so Initialisiert, dass er wie ein Motor bei jeder Umdrehung einen Tachopuls erzeugt. Die Periodendauer des Timer 2 PWM Signals stellt dabei die Rotationszeit des Motors dar. Die konstant eingestellte Einschaltzeit des PWM Signals (etwa 2 ms) repräsentiert den Tachopuls. Durch Veränderung des Timer 2 Reloadwerts ändert sich die Periodendauer des PWM Signals und simuliert somit einen Motor mit unterschiedlichen Rotationszeiten. Wenn die Periodendauer

im Bereich von 10 ms bis 60 ms änderbar ist, entspricht das einem Motor mit einem Drehzahlbereich von 6000 U/min bis 1000 U/min.
Die Drehzahl des simulierten Motors wird über ein Poti gesteuert. Zu diesem Zweck wird in jeder Schleife des Hauptprogramms der Potiwert linear in die Timer 2 Periodendauer umgerechnet und damit der Timer 2 Reloadwert neu gesetzt.

- Ziel :
  - Programmierung des Timer 2: nur Initialisierung
  - Debug einer komplizierteren Situation mit Hilfe des Logic Analyzers

5.6. Projekt 6 : T2SimToDZ

- Funktion : Der Code aus Projekt 5 wird erweitert. Der simulierte Tachopuls wird verwendet, um die Rotationszeit des simulierten Motors zu messen. Mit Hilfe einer C-Funktion wird dann aus der Rotationszeit die Drehzahl berechnet, und Potiwert, Rotationszeit und Drehzahl auf dem LCD angezeigt.
  Zur Messung der Rotationszeit wird mit dem Timer 0 in Mode 2 alle 100 µsec ein Tick Signal (Timer 0 Interrupt) erzeugt. Mit diesem Tick Signal wird in der Timer 0 ISR (alle 100 µsec) eine Variable inkrementiert. Diese Variable wird beim Auftreten eines Tachopulses auf 0 gesetzt und beim Auftreten des nächsten Tachopulses ausgelesen. Der Wert der Variablen gibt dann an, wie oft 100 µsec während einer Umdrehung vergangen sind, wie groß also die Rotationszeit ist. Das Auslesen des Wertes der Variablen und das neuerliche Aufsetzen auf 0 wird in der Timer 2 ISR durchgeführt . Die Timer 2 ISR wird aufgerufen durch das Signal TF2. Dieses Signal wird aktiviert, wenn der Timer 2 Zähler den Wert 0xFFFF erreicht und durch den Wert aus dem Reload Register ersetzt wird, wenn also eine neue Periode beginnt
- .Ziel :
  - Behandlung von und Kommunikation zwischen mehreren ISRs.
  - Integration einer eigenen C-Funktion.

5.7. Projekt 7 : Kennlinie

- Funktion : Projekt 7 ist eine Modifikation von Projekt 6. Die Timer 2 Steuerung wird so verändert, dass er das in der Aufgabenbeschreibung vorgestellte PWM Signal erzeugt. Die Ausgangsleitung INT4 (Port P1.1) wird mit dem Motormodell verbunden. Dadurch steuert das PWM Signal das Motormodell. Die Tacholeitung des Motormodells wird mit INT1 (Port P3.3) verbunden. Damit kann der bestehende Code die Rotationszeit messen, allerdings in der External Interrupt 1 ISR anstelle der Timer 2 ISR in Projekt 6.
  Alternativ kann auch das **simulierte Motormodell** verwendet werden, das zusätzlich als Quellcodedatei zur Verfügung gestellt wird. Damit kann dann die komplette Umgebung (Laborkoffer + Motormodell) im PC simuliert werden. Als Zwischenlösung ist es auch möglich, das simulierte Motormodell auf dem 8051 Prozessor im Laborkoffer laufen zu lassen und somit ohne physikalisches Motormodell zu arbeiten.
  Auf dem LCD wird zusätzlich der gerade eingestellte Comparewert des Timer 2 angezeigt. Damit ist es möglich die Kennlinie des Motors aufzunehmen. Zu diesem

Zweck dreht man am Poti bis die Drehzahlen von 1000, 2000, … bis 6000 U/min eingestellt sind und liest dann die zugehörigen Comparewerte ab. Die Kurve der Comparewerte in Abhängigkeit von der Drehzahl stellt die Motorkennlinie dar.

- .Ziel : Kennlinie des Motors aufnehmen

5.8. Projekt 8 : Steuerung

- Funktion : Projekt 8 basiert wiederum auf Projekt 7. Die Bestimmung des Timer 2 Comparewerts aus dem Potiwert wird ersetzt. Der Potiwert wird linear umgerechnet in die Solldrehzahl. Über die Motorkennlinie wird dann der zugehörige Timer2 Comparewert ermittelt. Dieser wird dann zur Steuerung der Motorgeschwindigkeit periodisch in das Timer 2 Compare Register geladen.
  Des weiteren wird, wie in der Aufgabenstellung verlangt, ein Mittelwert der gemessenen Rotationszeiten gebildet, bevor sie auf dem LCD angezeigt wird.

5.9. Projekt 9 : Regelung.

- Ziel : Motorsteuerung entsprechend der Aufgabenstellung
- Funktion : Zu Projekt 8 wird noch die Regelungsfunktion hinzu gefügt, wie in der Aufgabenstellung gefordert.
- Ziel : Vollständige Erledigung der Laboraufgabe.

6. Fallstricke und Probleme

Bei der Programmausführung, insbesondere bei der Benutzung des Laborkoffers, treten zuweilen schwer lokalisierbare Probleme auf. Ein typischer Fehlerfall ist der, dass die Synchronisierung zwischen dem Programm im PC und dem Monitorprogramm im 8051 Prozessor verloren geht. In besonders krassen Fällen kann sogar das PC Programm total abstürzen. Die Synchronisierung kann durch einen Reset im PC Programm und im Laborkoffer häufig wieder hergestellt werden. In seltenen Fällen ist es jedoch erforderlich, die Stromversorgung des Laborkoffers aus- und wieder einzuschalten.

Geht die Synchronisierung einmal verloren, was durch eine entsprechende Fehlermeldung angezeigt wird, wird häufig auch die Option 'Continue' angeboten. Von der Benutzung dieser Option wird jedoch abgeraten. Denn selbst wenn das System wieder zu laufen scheint, so zeigt die Erfahrung, kann es weiterhin Probleme geben, und man jagt eventuell langwierig einem Folgefehler nach. Es wird also dringend geraten, bei Verlust der Synchronisation durch einen Reset zuerst wieder saubere Verhältnisse herzustellen, und dann nach dem Fehler zu suchen.

In der Folge werden einige Fehlerszenarien geschildert und, wenn bekannt, was dagegen unternommen werden kann.

6.1. Probleme mit dem Stack

Wenn in einem Programm Interrupt Service Routinen (ISRs) aufgerufen werden können oder Unterprogramme ausgeführt werden, ist ein funktionierender Stack Mechanismus erforderlich : Ein Stackbereich im internen Datenspeicher muss allokiert sein, und der Stack Pointer (SFR SP) muss bei der Initialisierung so gesetzt werden, dass er auf den Anfang des Stackbereichs zeigt. Mögliche Fehler in diesem Zusammenhang sind :

- Der Stack Mechanismus wurde komplett vergessen.
- Der Stackbereich ist zu klein.
- Die Anzahl der Werte beim Laden und Entladen des Stacks stimmen nicht überein.

In einem solchen Fall können sich Schreibbefehle zum Speicher (MOV) und Stack Ladeoperationen die Daten gegenseitig kaputt schreiben. Die Folgen sind kaum absehbar.

6.2. Fehlende SFR Symboldefinitionen

Es kommt vor, dass beim Assemblieren das in der Dokumentation angegebene Symbol für ein Byte oder ein Bit des SFR Bereichs als Fehler angezeigt wird. Das liegt häufig daran, dass die Definition für ein solches Symbol in der Entwicklungsumgebung fehlt.

µ -Vision enthält standardmäßig bereits die Definition von SFR Symbolen, die für alle Typen des 8051 Prozessors gültig sind. Diese Definitionen enthalten jedoch nicht alle Symbole für den von uns verwendeten Prozessor (z.B. fehlt das Symbol P5 für den Port 5). Für die einzelnen Prozessortypen enthält µ-Vision Dateien mit individuellen Symboldefinitionen. Diese Symbole für einen ausgewählten Prozessor können über einen $INCLUDE Pseudobefehl der entsprechenden Assemblerdatei des Programms hinzugefügt werden. Die Sequenz

```
$NOMOD51
$NOLIST
$INCLUDE (C:\KEIL\C51\ASM\REG515C.INC)
$LIST
```

deaktiviert die eingebauten standardmäßigen SFR Definitionen ($NOMOD51) und aktiviert die Symbole für den in unserem Laborkoffer verwendeten Prozessor C515C-L. Die $NOLIST Assemblerdirektive unterdrückt, dass diese Definitionen in der LIST-Datei erscheinen.

Unglücklicherweise hat KEIL das Symbol für 'Enable All Interrupts', das in der Dokumentation mit 'EAL' abgekürzt wird, in der Datei REG515C.INC 'EA' genannt. Um diese Diskrepanz zu korrigieren, kann dem Programm der Pseudobefehl

```
EAL    EQU    EA
```

hinzugefügt werden.

Um im gesamten Code die selben Symbole verwenden zu können, sollten die o.g. Pseudobefehle am Beginn **jedes** Assemblerprogramms eingefügt werden.

6.3. Überschreiben des Monitorcodes

Der Monitorcode im Laborkoffer residiert zum Teil in Speicherbereichen, in die auch zu testender Code geladen werden kann (z.B der Interrupt Einsprung für das serielle Interface, nachträgliche Korrekturen des Monitorcodes). Deshalb schreibt die Aufgabenstellung vor, den zu testenden Code ab Adresse 0x4000 zu platzieren. Die Segmente für den Programmspeicher müssen entsprechend spezifiziert sein. Wenn der gesamte Code in dem absoluten Programmsegment mit Anfangsadresse 0x0000 gespeichert wird, ist Ärger vorprogrammiert.

6.4. LCD Funktionen in Interrupt Service Routinen

Die LCD Funktionen sind so geschrieben, dass sie beim Aufruf die vorhandenen Werte der Resourcen, die sie selbst verwenden, **nicht** retten und später wieder zurück stellen. Wenn eine LCD Funktion in einer ISR aufgerufen wird, wird sie Register, Akku, DPTR usw. verändern. Wenn dann das unterbrochene Hauptprogramm oder eine andere unterbrochene ISR wieder die Kontrolle bekommt, sind nicht mehr die erwarteten Werte vorhanden, sondern die von der LCD Funktion veränderten Werte. Die Folgen sind nicht absehbar.

Abhilfe : LCD Funktionen nur im Hauptprogramm verwenden.

6.5. Endlose ISR Schleifen

Wenn z.B. auf Grund permanent anstehender Interrupt Requests mit höherer Priorität der Prozessor sich nur noch in diesen ISRs aufhält, und die ISR für die serielle Schnittstelle nicht mehr ausgeführt wird, kann das PC Programm nicht mehr mit dem Monitorprogramm im Prozessor kommunizieren und die Synchronisierung geht verloren. Ein Beispiel für einen permanent anstehenden Interrupt Request ist ein pegelgesteuerter Externer

Interrupt 0 oder 1 (IT0 = 0 oder IT1 = 0), bei dem die Interruptquelle nicht per Programm zurück gesetzt wird.

Findet der permanente Aufenthalt in ISRs mit weniger hoher Priorität statt, ist es möglich, dass zwar die ISR für die serielle Schnittstelle noch Kontrolle bekommt, nicht aber das Hauptprogramm. Die Synchronisierung geht dann nicht verloren, aber das Programm scheint sich aufgehängt zu haben.

6.6. Watchdog Timer

Der Watchdog Timer (WDT) im Laborkoffer ist durch die Verbindung des PE/SWD Pins mit Masse deaktiviert. Im Simulator ist der WDT aktiviert. Lässt man ein Programm im Simulator im Run Mode laufen, passiert periodisch ein unerwünschter WDT Reset. Den WDT im Hauptprogramm regelmäßig zu initialisieren (SETB WDT + SETB SWDT), wie es eigentlich gedacht ist, löst das Problem im Simulator. Wenn man diesen Code allerdings unverändert auf dem Laborkoffer ausführt, steigt das PC Programm mit einer Fehlermeldung aus.

Abhilfe : Den WDT im Simulator durch eine entsprechende Anweisung in der .ini-Datei deaktivieren.

Diese Deaktivierung ist in der Datei PotiUndWDT.ini, die auch das simulierte Potentiometer realisiert, bereits enthalten.

6.7. Warnings beim Link Prozess

Neben Fehlern (Errors), die beim Assemblieren, Compilieren oder Linken auftreten können und immer korrigiert werden müssen, gibt es noch Warnungen (Warnings). Zwei Arten von solchen Warnungen stellen keine Fehler dar und können ignoriert werden. Sie sehen z.B. folgendermaßen aus :

1. *** WARNING L16: UNCALLED SEGMENT, IGNORED FOR OVERLAY PROCESS
   SEGMENT: ?PR?_LCD_CURS?LCD4X20DUO
2. *** WARNING L10: CANNOT DETERMINE ROOT SEGMENT

Diese Warnungen rühren daher, dass wir nicht den STARTUP.A51 Code benutzen.

Dann gibt es noch andere Warnungen, die eigentlich Fehler sind, die µ-Vision aber nicht eindeutig als Fehler identifizieren kann. Diese sehen z.B. folgendermaßen aus :

1. *** WARNING L1: UNRESOLVED EXTERNAL SYMBOL
   SYMBOL: TROT2DZ
   MODULE: Haupt.obj (HAUPT)
2. *** WARNING L2: REFERENCE MADE TO UNRESOLVED EXTERNAL
   SYMBOL: TROT2DZ
   MODULE: Haupt.obj (HAUPT)

Der Grund für eine solche Warnung liegt häufig am Fehlen einer korrespondierenden PUBLIC Spezifikation oder am Fehlen einer ganzen C-Code Datei.

7. Modularität des Programms

Das Programm sollte als Assemblerprogramm angelegt sein und in seinen wesentlichen Teilen durch Assemblerbefehle realisiert werden. Es ist jedoch in der KEIL Entwicklungsumgebung möglich, eine in C geschriebene Unterroutine aus einem Assemblerprogramm heraus aufzurufen, sowie eine Assembler-Unterroutine aus einer C-Routine heraus aufzurufen. Die erste dieser Möglichkeiten soll im Rahmen der Laborarbeit angewendet werden. Es gibt Funktionen, die in Assemblersprache umständlich zu programmieren sind. Ein typisches Beispiel ist die Umrechnung der Motordrehzahl in die Rotationszeit. In diesem Fall muss durch einen zwei Byte langen Binärwert dividiert werden, was in Assemblersprache eine längliche und nicht ganz einfache Routine erfordert, in C jedoch ganz trivial erledigt werden kann. In einem solchen Fall sollte die problematische Funktion in C programmiert werden, und diese C-Routine aus dem Assemblerprogramm heraus aufgerufen werden.

Außerdem werden zum Betrieb des im Laborkoffer eingebauten LCDs eine Reihe von C-Routinen vorgegeben, die natürlich zu verwenden sind. Diese Routinen sind in der Datei **lcd4x20duo.c** zusammengefasst.

7.1. Schnittstelle zwischen Assembler und C-Code

Die Keil Entwicklungsumgebung sieht zwei Methoden der Parameterübergabe beim Aufruf einer C-Routine aus einem Assemblerprogramm heraus vor :

- Parameterübergabe über 8051 Register. Diese Methode wird bei den LCD Routinen angewendet. Details finden sich in der Datei **lcd4x20v2.inc**.
- Parameterübergabe über globale Variablen im Datenspeicher. Diese Methode sollte für selbst geschriebene C-Routinen angewendet werden.

Eine globale Variable (z.B. eine zwei Byte lange Binärzahl) kann innerhalb des Datensegmentes eines Assemblerprogramms durch die Anweisungen

```
Binärzahl : DS    2
            PUBLIC    Binärzahl
```

definiert werden. Um die Variable in einer C-Routine verwenden zu können, muss sie über die Anweisung

```
extern int Binärzahl;
```

in dieser C-Routine deklariert werden. Der Keil C Compiler verwendet die sog. **Big Endian** Methode zur Speicherung von zwei Byte langen Datenelementen. Bei Big Endian ist das erste Byte das Most Significant Byte (MSB) und das letzte Byte das Least Significant Byte (LSB).

Alternativ könnte eine globale Variable auch in der C-Routine deklariert werden, z.B.

```
unsigned char Byte2;
```

Die Eigenschaft 'Public' ist in dieser Deklaration implizit. In der Assemblerroutine müsste dann die Variable über die Anweisung

```
EXTRN     DATA(Byte2)
```

verfügbar gemacht werden.

Um eine C-Rotine per **CALL**-Befehl aus einem Assemblerprogramm heraus aufrufen zu können, muss der Name der C-Routine im Assemblerprogramm über die Anweisung

```
EXTRN     CODE(CRtn);
```

bekannt gemacht werden. Im C-Code wird die Funktion ganz normal programmiert, z.B.:

```
void Crtn void;
void Crtn void
  {
    ........
  }
```

Die Eigenschaft 'PUBLIC' ist auch in dieser Spezifikation implizit.

Die in C übliche Fähigkeit, beim Aufruf einer Funktion oder Prozedur Argumente zu übergeben oder Ergebniswerte zurückzugeben, ist in der KEIL Entwicklungsumgebung stark eingeschränkt. Sie sollten der Einfachheit halber die Kommunikation zwischen C- und Assemblerprogramm ausschließlich über die oben angeführte Methode der globalen Variablen erledigen.

7.2. Programmstruktur

Die KEIL Entwicklungsumgebung bietet die Möglichkeit, ein Programm durch Aufteilung in mehrere Assembler-Quelldateien und mehrere C-Quelldateien zu strukturieren. Die einzelnen Teile können dann separat assembliert bzw. compiliert werden. Im Link Vorgang werden alle in einer sog. Source Group zusammen gefassten Objektdateien zu einem lauffähigen Programm zusammengesetzt.

Für die Laboraufgabe ist vorgeschrieben, dass das Programm in zwei Assembler Quelldateien aufgeteilt wird. Die Datei **INIT.A51** soll die gesamten Initialisierungsfunktionen und die Interrupt Service Routinen enthalten. Die Datei **HAUPT.A51** soll die Funktionen für die Betriebsphase (die Endlosschleife) enthalten. Zur besseren Strukturierung sind abgeschlossene Funktionen in eigene (eventuell auch geschachtelte) Unterroutinen zu packen. In Assembler geschriebene Unterroutinen können optional in einer oder mehreren separaten Assembler Quelldateien zusammengefasst werden.

Die in C geschriebenen eigenen Unterroutinen müssen in eine (oder mehrere) C Quelldateien gepackt werden.

8. Dokumentation

Ein wesentlicher Teil der Laborarbeit ist die Erstellung eines Laborberichts. Diese Kapitel soll Anregungen geben, was der Laborbericht enthalten könnte und wie er aufgebaut sein könnte.

Im nach dem Top-Down Prinzip durchgeführte Design- und Programmentwicklungs-Prozess kommt man von anfangs groben Designentscheidungen schrittweise zu immer detailierteren Strukturen. Die Dokumentation, insbesondere wenn sie parallel zum Design angefertigt wird, sollte eine ähnliche Struktur aufweisen.

Im einzelnen könnten die Dokumentation aus folgenden Kapiteln bestehen :

1. Identifikation der Funktionsblöcke
   - Nennung der ausgewählten Funktionsblöcke
   - Kurze verbale Beschreibung
   - Kommunikation zwischen Funktionsblöcken
2. Beschreibung der Funktionsblöcke im Detail
   - Funktionen innerhalb des Funktionsblocks
   - Art der Kommunikation zwischen den Funktionen
   - Angabe der Variablen und ihrer Wertebereiche
   - Angabe, wann und wie oft Funktionen durchzuführen sind
3. Verwendete Einrichtungen (z.B. Timer, ...)
   - Zweck der Verwendung (Referenz zu 2.)
   - Kommandos zur Initialisierung (SFR Werte)
   - Kommandos während des Betriebs
4. Programmstruktur
   - Was geschieht während der Initialisierung ?
   - Was erledigen die ISRs ?
   - Struktur der Assemblerprogramme Init.a51 und Haupt.a51
   - Unterprogramme
5. Programmlisten
   - Aussagekräftige Kommentare mit Bezug zu 1. bis 4.

8.1. Dokumentationsbeispiel

Am Beispiel des Funktionsblocks **Solldrehzahl ermitteln** soll gezeigt werden, wie die Einträge in Kapitel 1. bis 4. aussehen könnten :

1. Funktionsblock : Solldrehzahl ermitteln

   Dieser Funktionsblock übernimmt vom A/D-Wandler den digitalen Wert der Potentiometereinstellung und berechnet daraus die Solldrehzahl. Der Wert der Solldrehzahl wird vom Funktionsblock 'Drehzahlen anzeigen' und vom Funktionsblock 'Motor steuern' verwendet.

2. Details : Solldrehzahl ermitteln

- Der A/D-Wandler stellt den 10 Bit breiten digitalen Wert in zwei SFRs zur Verfügung:
  - ADDATH : Bits 9 ... 2
  - ADDATL : Bits 1 ... 0
- Das Programm benutzt jedoch nur die 8 Bits aus ADDATH und berechnet daraus die Variable *PotiWert*.
  - *PotiMin* <= *PotiWert* <= *PotiMax*
  - *PotiMin* (möglicher Minimalwert : 0) = 0
  - *PotiMax* (möglicher Maximalwert : 255) = 250
  - Werte für *PotiMin* und *PotiMax* werden eingefügt, wenn die aktuellen Werte für den Laborkoffer vorliegen.
  - *PotiMin* und *PotiMax* können von Laborkoffer zu Laborkoffer variieren.
  - *PotiMin* und *PotiMax* werden im Programm über EQU Pseudobefehle spezifiziert.
- Mit Hilfe des Unterprogramms *Poti2DZ* wird die *SollDZ* ermittelt.
  - Inputwerte des Funktionsblocks : keine
  - Outputwerte des Funktionsblocks : *SollDZ*

3. Analog/Digital-Wandler

   Diese Einrichtung konvertiert die an einem Potentiometer eingestellte Spannung in einen 10 Bit breiten digitalen Wert.

   - Initialisierung :
     - ADEX = 0 – Internal Start of Conversion
     - ADM = 0 – Keine Continuous Conversion
     - MX2, MX1, MX0 = 0, 0, 0 – A/D Konverter 0 ausgewählt.
   - Betriebsphase :
     - ADDATL = 0 – startet die erste Konversion
     - Bevor ein digitaler Wert ausgelesen wird, wird überprüft, ob das Busy Flag BSY (ADCON0.4) = 0 ist. Falls nicht, wird in einer Schleife gewartet, bis das Busy Flag zurückgesetzt ist.
     - Auslesen des digitalen Werts aus SFR ADDATH.

4. .....

   C Unterprogramme :

   - ***Poti2DZ***
     - Um mit Integerzahlen arbeiten zu können, berechnet sich

SollDZ = *PotiDZFaktor* * *PotiWert* + *DZSollMin*

Abhängig von den Werten von *PotiMin* und *PotiMax* werden *PotiDZFaktor* und *DZSollMin* so gewählt, dass *SollDZ* minimal einen Wert von knapp unter 1000 und maximal einen Wert von knapp über 6000 annimmt.
Z.B.: Mit *PotiDZFaktor = 6 und DZSollMin = 500* ergibt sich ein Wertebereich für *SollDZ* von 500 bis 6500, wenn *PotiMin* = 0 und *PotiMax* = 1000 sind.

- *PotiDZFaktor* und *DZSollMin* werden als Konstante deklariert.
- Input : *PotiWert*
- Output : *SollDZ*