Labor Mikrocomputertechnik I

Projekt:

Drehzahlregelung

4. Hj. (März – Mai 2013)

Dozent : Klaus Kraft

1 Zielsetzung der Laborveranstaltung

Kennenlernen eines Mikrocomputer-Entwicklungssystems für den Mikroprozessor 80c515c mit folgenden Dienstprogrammen:

- EDITOR zur Erstellung von Programmquelldateien (Assemblersprache und C)
- ASSEMBLER zur Erzeugung von Objektcode und Listing von Assemblerprogrammen
- COMPILER zur Erzeugung von Objektcode (und/oder Assemblerdateien) aus C Code
- LINKER zum Zusammenbinden der einzelnen Programmmodule.
- SIMULATOR zum Austesten von entwickelten Programmteilen ohne Userhardware.
- MONITOR zum Testen des Gesamtprogrammes mit Userhardware.

Die Programmieraufgabe soll in Programm-Module bzw. Teilprobleme aufgeteilt werden, um eine möglichst praxisnahe Projektbearbeitung zu erreichen.

Die Aufgabe soll selbständig in **Zweiergruppen** bearbeitet werden.

Über den Lösungsweg und die Ergebnisse ist ein Laborbericht auszuarbeiten, der die Fähigkeit zur Erstellung einer nachvollziehbaren Dokumentation nachweist.

ANFORDERUNGEN AN DEN LABORBERICHT:

- Dokumentation der Steuerung der prozessor-internen Komponenten(z.B. Timer, A/D Wandler, Interrupt Logik) und der externen Komponenten (z.B. LCD, Motor).
  - Initialisierung
  - Steuerung während des Betriebs.
- Angabe der Wertebereiche der gewählten variablen Parameter (z.B Rotationszeit, Häufigkeit der Motorregelungen).
- Die im Verlauf der Laborarbeit ermittelte Motorkennlinie.
- Programmstruktur-Überblick.
- Darstellung der Programmmodule durch je ein Struktogramm, Pseudocode oder Flussdiagramm.
- Beschreibung der Schnittstellen von Unterprogrammen (Variablenübergabe, zerstörte Resourcen).
- Programmlisten mit **Kommentaren**, die den Zusammenhang zu den o.g. Struktogrammen usw. herstellen.

2 Aufgabenbeschreibung

Mit dem Programm soll die Drehzahl eines Gleichstrommotors digital geregelt werden.

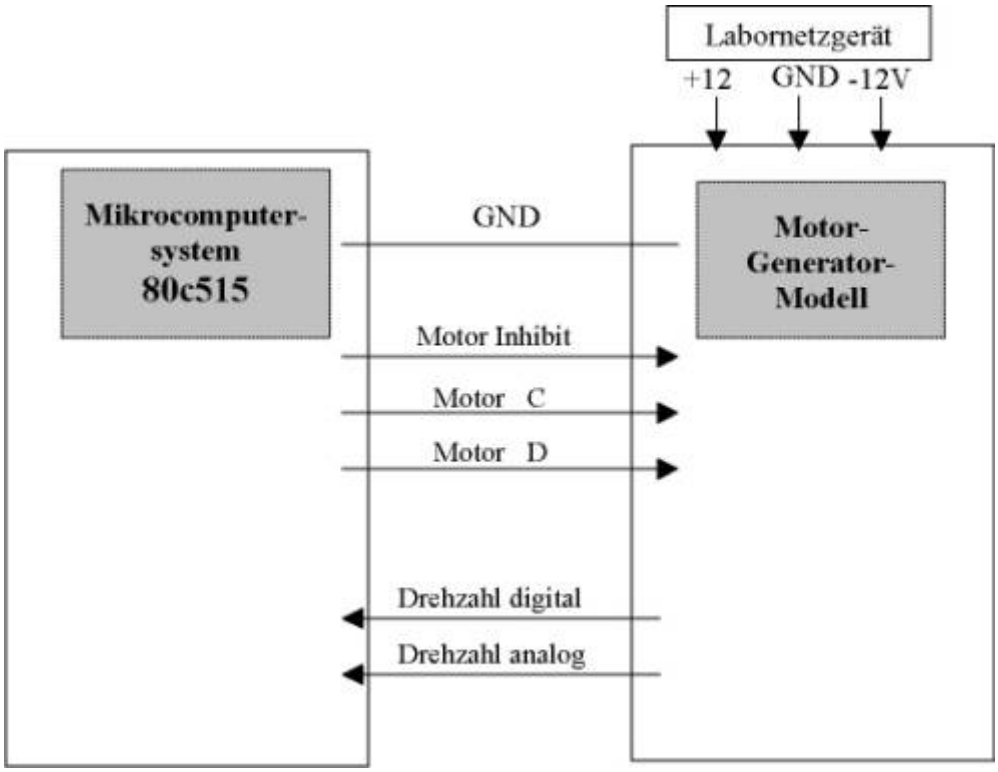

Abbildung 1: Blockschaltbild des Versuches

2.1 Hardwarebeschreibung

Die Hardware gibt die folgenden Rahmenbedingungen vor:

- Steuerung der Drehzahl eines Motors durch programmierte Einstellung des Tastverhältnisses (High / Low-Verhältnis) des Motorstroms. **Die Periodendauer T_{PWM} der Pulsweitenmodulation soll 10 ms betragen!**
  Die Endstufe des DC-Motors wird über die **Motor Inhibit** Leitung angesteuert. Liegt die Motor Inhibit Leitung auf Low Level, wird der Strom durch den Motor eingeschaltet und der Motor wird angetrieben. Liegt die Motor Inhibit Leitung auf High Level, wird der Strom abgeschaltet und der Motor läuft leer.
- Istdrehzahl-Erfassung **digital** über den Externen Interrupt 1 (Buchse P3.3 des Laborkoffers). Der minimale High-Impuls der Reflexionslichtschranke pro Umdrehung beträgt 1,5 ms, wenn sich der Motor mit einer Rotationszeit von 10 ms, entsprechend 6000 U/min dreht.
- Sollwert-Einstellung über ein Potentiometer des Laborkoffers mit nachgeschalteter Analog/Dgital-Wandlung.

- Der Laborkoffer besitzt ein kleines LCD (4 Zeilen à 20 Zeichen), auf dem Soll- und Istdrehzahl (und ggf. andere Informationen) angezeigt werden können.
- Der Motor kann durch einen Generator (und andere äußere Einflüsse) belastet werden. Der Generator ist über eine Achse direkt mit dem Motor verbunden. Er kann über einen digitalen Ausgang per Programm oder mit Hilfe des Kippschalters auf dem Motormodell eingeschaltet werden.
- In dem Laborkoffer befinden sich DIP-Schalter, von denen drei für die Steuerung der Betriebsart eingesetzt werden.
  - DIP-Schalter 1 = 0 : Solldrehzahl nur anzeigen
1 : Solldrehzahl zur Steuerung/Regelung übernehmen
  - DIP-Schalter 2 = 0 : Generatorbremse ausschalten
1 : Generatorbremse einschalten
  - DIP-Schalter 3 = 0 : Motorregelung aus (nur steuern)
1 : Motorregelung ein

2.2 Programmfunktion

Das gesamte Softwarepaket soll aus den folgenden groben Funktionen bestehen:

- Anfangsversion des Programms zur **Ermittlung der Motorkennlinie**.
- Programmstart mit Initialisierung von Hard- und Software.
- Ermitteln der Istdrehzahl.
- Sollwerteinstellung über das Potentiometer AN0 des Mikrocontrollerkoffers.
- Steuern bzw. Regeln der Drehzahl.

Natürlich müssen diese je nach Bedarf in feinere Unterfunktionen unterteilt werden. Gehen Sie beim Entwurf nach der Methode der **Top Down** Verfeinerung vor! Beim Schreiben der Assembler-Routinen und beim Testen sollten Sie die **Bottom Up** Strategie verwenden!

2.2.1 Dienstprogramme

Zur Einübung der Entwicklungsumgebung sollen **vor allen anderen Programmteilen** die nachfolgend gennannten Dienstprogramme entworfen und mit dem Simulator getestet werden. Dabei ist auf strenge Einhaltung der Schnittstellendefinition zu achten. Schreiben Sie jeweils eine C Routine für die eigentliche Umrechnung und rufen Sie diese aus einem Assemblerprogramm heraus auf. Benutzen Sie zur Parameterübergabe und zur Ergebnisrückgabe je ein 16 Bit Feld im internen Datenspeicher.

Da der Keil C Compiler das **Big Endian** Format für Mehrbyte-Daten im Speicher verwendet, sollte generell Big Endian verwendet werden, also : **lower address – most significant byte (MSB); higher address – least significant byte (LSB)**.

n2T : Umrechnung einer Drehzahl n [U/min] in die Rotationszeit T_{rot} [Ticks]
(1 Tick = 100us)

EIN : 16 Bit binäre Variable **UmrDZ** im internen Datenspeicher

AUS : 16 Bit binäre Variable **UmrRotZ** im internen Datenspeicher

zerstört : A, Flags, R0 ... R7

Beispiel : bei n = 6000 U/min ist T_{rot} = 10 ms, d.h. UmrRotZ = 100 Ticks

Speicherabbild :	EIN :	AUS :
	(UmrDZ + 0) = 5888 = 0x17	(UmrRotZ + 0) = 0 = 0x00
	(UmrDZ + 1) = 112 = 0x70	(UmrRotZ + 1) = 100 = 0x64

T2n : Umrechnung einer Rotationszeit T_{rot} [Ticks] in die Drehzahl n [U/min]

EIN : 16 Bit binäre Variable **UmrRotZ** im internen Datenspeicher

AUS : 16 Bit binäre Variable **UmrDZ** im internen Datenspeicher

zerstört : A, Flags, R0 ... R7

2.2.2 Motorkennlinie

Die Abhängigkeit der **Motordrehzahl** vom eingestellten **Tastverhältnis** ist eine stark nichtlineare Funktion. Da die Motormodelle einer Exemplarstreuung unterliegen, ist als Vorarbeit zum eigentlichen Programm die individuelle Kennlinie des vorliegenden Motormodells zu ermitteln. Die Motorkennlinie soll den Bereich von 1000 U/min bis 6000 U/min (bzw. maximales Tastverhältnis) abdecken. Die Motorkennlinie wird durch fünf Geradenstücke (1000 U/min bis 2000 U/min, 2000 U/min bis 3000 U/min, usw.) angenähert.

Zur Ermittlung der Motorkennlinie steuern Sie das Tastverhältnis direkt über das Potentiometer (lineare Abhängigkeit) und zeigen die Istdrehzahl **und** das Tastverhältnis auf dem LCD an. Durch Drehen am Potentiometer stellen Sie bei unbelastetem Motor die Tausenderwerte der Drehzahl ein und lesen das zugehörige Tastverhältnis ab. Die so ermittelten Wertepaare ergeben durch die Verbindung mit Geradenstücken die Motorkennlinie. Bauen Sie dann diese Motorkennlinie in das endgültigen Programm ein und verwenden Sie die so ermittelten Werte zur **Steuerung** des Motors.

Die für dieses Programm notwendigen Routinen werden zum größten Teil auch wieder im endgültigen Programm benötigt. Durch die geringere Funktionalität ist es jedoch einfacher zu überschauen und stellt somit einen besseren Einstieg in die Entwicklungsumgebung dar. Achten Sie trotzdem auch in dieser frühen Phase auf gute Strukturierung und Modularität.

2.2.3 Programmstart

Denken Sie daran, dass jedes Programm eine Einschalt- (= RESET-) Routine benötigt. Nach dem Programmstart sollte auf dem LCD folgender Text erscheinen :

SOLLWERT 0000 U

ISTWERT 0000 U

Für die LCD-Anzeige stehen hierzu verschiedene C-Funktionen zur Ausgabe von Bytes, 16-Bit-Integern, Zeichen im ASCII-Format oder ganzen Strings zur Verfügung (siehe 4.3 Fertige Unterprogramme). Die Aufrufkonventionen für die Verwendung von C-Funktionen aus Assemblercode heraus sind für Interessierte im Compilerhandbuch beschrieben (Kapitel Cx51 Compiler User's Guide – Advanced Programming – Interfacing C to Assembler – Function Parameters).

2.2.4 Motor hochfahren

Bei der Initialisierung des Programms soll der Motor über einen Zeitraum von 3 sec auf die konstante Drehzahl von 3000 U/min hochgefahren werden. Benutzen Sie dazu die Tastverhältniswerte für die glatten Tausender Drehzahlen aus der Motorkennlinie. Denken Sie daran, dass konstante Wertetabellen bei einem Controller im **Programmspeicher** abgelegt werden, und dass der Befehlszähler diese Speicherstellen nie erreichen darf.

2.2.5 Sollwerteingabe

Die Wandlung des Sollwerts vom Potentiometer erfolgt kontinuierlich und soll ebenso kontinuierlich angezeigt werden. Jedoch wird der eingestellte Sollwert nur zur Steuerung bzw. Regelung übernommen, wenn der DIP-Schalter 1 auf '1' steht.

Ist der DIP-Schalter '0', ist die Übernahme gesperrt. In diesem Fall soll die Sollwert-Anzeige blinken (0,5 sec an – 0,5 sec aus) und den am Potentiometer eingestellten Wert anzeigen. Für die Steuerung/Regelung ist die Solldrehzahl zu verwenden, die vor dem Ausschalten des DIP-Schalters 1 gültig war. Wenn das Programm mit dem DIP-Schalter 1 auf '0' gestartet wird, soll die während der Initialisierung eingestellte Solldrehzahl von 3000 U/min beibehalten werden.

2.2.6 Drehzahlerfassung

Parallel zur Ausgabe der Pulsweitenmodulation soll die Istdrehzahl über die digitalen Drehzahlimpulse per Interrupt erfasst und am Display ausgegeben werden.

Zu einfachen Drehzahlerfassung programmieren Sie den Timer0 in Mode 2 mit einer Zykluszeit von **100 us**. Zählen Sie nach jedem Ablauf des Timers in einer entsprechenden ISR eine Variable hoch. Der Wert der Variablen stellt also die verstrichene Zeit in 100 us Einheiten dar. Bei jedem Auftreten des externen Interrupt Requests von der Lichtschranke können Sie die Variable auslesen und als Maß der Rotationszeit mit der Quantisierung 100

us verwenden. Nach dem Auslesen der Variable sollte diese zurückgesetzt werden, damit dann die Zeit für eine neue Umdrehung gemessen wird.

2.2.7 Steuern/Regeln der Drehzahl

Steht der **DIP-Schalter 3 auf '0'** soll die Drehzahl nur **gesteuert** werden. Dazu wird über die Motorkennlinie das Tastverhältnis für die eingestellte Solldrehzahl ermittelt und für die Steuerung der PWM verwendet.

Steht der **DIP-Schalter 3 auf '1'** soll die Drehzahl des Motors **geregelt** werden. D.h., das Tastverhältnis wird so angepasst, dass äußere Einflüsse (z.B. eingeschalteter Generator) kompensiert werden, und die Istdrehzahl immer an die Solldrehzahl angeglichen wird.

Die Drehzahl des Motors (DZ_{Ist}) wird mit der vorgegebenen Solldrehzahl (DZ_{Soll}) verglichen. Diese Differenz wird auf die Solldrehzahl (DZ_{Soll}) bezogen, um die relative Abweichung zu erhalten. Die relative Abweichung sollte in einem Bereich von ± 5 % gehalten werden. Bei einer größeren Abweichung muss das Tastverhältnis nachgeregelt werden. Hierzu sollten Sie einen P-Regler verwenden. Dieses bedeutet, dass der Unterschied zwischen Ist- und Soll-Drehzahl zu einem Korrekturwert für das Tastverhältnis führt. Das neue Tastverhältnis wird nach der folgenden Formel berechnet.

$$TV_t = [1 + k * (DZ_{Soll} – DZ_{Ist}) / DZ_{Soll}] * TV_{t-1}$$

TV_t = Tastverhältnis zum Zeitpunkt t

TV_{t-1} = Tastverhältnis zum Zeitpunkt t-1 (vorherige Regelung)

$(DZ_{Soll} – DZ_{Ist}) / DZ_{Soll}$: Abweichung relativ zu DZ_{Soll}

k : Verstärkungsfaktor für die Abweichung

Die relative Abweichung kann positiv oder negativ sein. Beginnen Sie mit einem Verstärkungsfaktor k von 2. Stellen Sie fest, ob es einen besseren Wert für k gibt, bei der die Änderung zügig ausgeführt wird und trotzdem nicht zum Überschwingen führt.

Bedenken Sie, dass der Motor eine gewisse Zeit braucht, um auf eine Änderung des Tastverhältnisses zu reagieren. Danach sollten Sie den Abstand zwischen zwei Regelungsoperationen wählen.

2.2.8 Displayanzeige

Auf dem Display des Koffers soll jederzeit die Soll- und die Ist-Drehzahl mit folgendem Format abzulesen sein.

SOLLWERT xxxx U

ISTWERT xxxx U

Für die LCD-Anzeige stehen hierzu verschiedene C-Funktionen zur Verfügung (siehe 4.3 Fertige Unterprogramme).

Um eine ruhige Anzeige zu erhalten, sollen mehrere Drehzahlwerte gemittelt (z.B. 32 oder 64 wegen einfacher Division), bevor die Anzeige aufgefrischt wird.

2.2.9 Generator ein/ausschalten

Wird der DIP-Schalter 2 auf '1' gesetzt wird der Generator auf dem Modell an den Motor angekoppelt und erzeugt eine Last. Diese Funktion kann verwendet werden, um die Regelungsoperation zu überprüfen. Mit dem DIP-Schalter 2 auf '0' läuft der Motor unbelastet.

3 Vorgaben zur Modularität der Software

Softwaremodule sollen, wie in einem Industrieprojekt üblich, unabhängig voneinander entwickelt werden. Es sind mindestens ***drei*** Module vorzusehen, auf die die Programmteile verteilt werden.

Die in den Modulen enthaltenen Segmente sollen soweit möglich relokativ angelegt werden (d.h. alles außer Reset- und Interrupt-Einsprüngen).

Die Datensegmente sind im **internen** Datenspeicher anzulegen. Beachten sie bitte, dass der Compiler alle Daten im Big Endian Format ablegt : **higher byte - lower adress**

Der Linker bindet dann alle im Projekt definierten Anteile zu einem Gesamt-Code zusammen. Die relokativen Anteile werden hinter einem Freiraum im Codebereich ab 4000H positioniert. Diese Lücke ist nur durch die Schulversion der Entwicklungssoftware bedingt und kann nicht umgangen werden!!!

3.1 Vorgegebene Module

Vom Projektleiter wurden die folgenden Module einschließlich deren Namen und Programmiersprache verbindlich vorgeschrieben:

- **INIT.A51** in Assembler
  Reset-routine, Interrupteinsprünge samt ISRs und Initialisierung aller Peripherie wie z.B. Timer
- **HAUPT.A51** in Assembler
  Programm zum Betrieb des Systems (endlose Schleife)
- **CRTN.C** in C-Code
  Sammlung der in C geschriebenen Unterroutinen

Bei Bedarf (z.B. zur besseren Übersichtlichkeit) können zusätzliche Module angelegt werden.

4 Erläuterung von Systemkomponenten

4.1 Timer für Pulsweitenmodulation (PWM):

Für die Realisierung der Pulsweitenmodulation besitzt der Prozessor ein geeignetes System von Timern mit zugehörigen Reload- und Vergleichsregistern . Verwenden Sie den Timer 2 (16 Bit) zusammen mit dem Compare-Register CRC (16 Bit) als Reload-Register und dem Compare-Register CC1 (16 Bit) in Compare-Funktion. Damit erhält man das PWM-Signal an Port 1.1 des Prozessors, bzw. auf dem Board des Mikrocontrollerkoffers. Der Timer 2 soll im Mode 0 mit einer Eingangsfrequenz von f_{osc} /12 (= 0,833 MHz) betrieben werden.

4.2 Zentraler Timer

Verwenden Sie für die sonstigen notwendigen Zeiten den Timer 0 mit einer Laufzeit von 100µs. Dieser soll alle 100µs einen Interrupt erzeugen, mit dem dann verschiedene Software-Timer programmiert werden können. Dann kann z.B. bei Vollendung einer Motorumdrehung im zugehörigen Software-Timer abgelesen werden, wie lange eine Umdrehung gedauert hat.

4.3 Fertige Unterprogramme

Zur Vereinfachung der Arbeit dürfen die folgenden Unterprogramme mit verwendet werden. Zur Vermeidung unnötiger Fehlersuche sollten die Schnittstellendefinitionen genau beachtet werden.

Die Hilfsroutinen stehen in Form von C-Unterprogrammen zur Verfügung. Für ihre Verwendung muss die Datei lcd4x20duo.c dem Projekt hinzugefügt werden. Wie C-Unterprogramme von Assembler aus aufgerufen werden können und wie dabei die Parameterübergabe funktioniert, können den KEIL Hilfe-Seiten entnommen werden (Kapitel 6, Interfacing C Programs to Assembler).

Funktion **`lcd_init()`**:

Initialisierung des LCD - Displays an der MC51 Hardware

Prototyp:

`void lcd_init (void);` muss nicht mit aufgerufen werden!

Funktion **`lcd_clr()`**:

Löschen des LCD. Cursor auf Position 0 setzen.

Prototyp:

`void lcd_clr (void);`

Funktion **_lcd_curs(wert)**:

Der Cursor der Anzeige wird auf eine Position zwischen 0 ... 79 gesetzt. Zweite Zeile der Anzeige ab Position 20.

Prototyp:

```
void _lcd_curs ( unsigned char );
```

Funktion **_lcd_byte(wert)**:

Der Übergabewert vom Typ unsigned char wird als 3-stellige Zahl 000 ... 255 ab der aktuellen Cursorposition ausgegeben.

Prototyp:

```
void _lcd_byte ( unsigned char );
```

Funktion **_lcd_int(int wert):**

Der Übergabewert vom Typ unsigned int wird als 4-stellige Zahl 0000 ... 9999 ab der aktuellen Cursorposition ausgegeben. (kein vollständiger Int Bereich!)

Prototyp:

```
void _lcd_int ( unsigned int );
```

Funktion **_asc_out(zeichen)**:

Mit dieser Funktion können einzelne ASCII - Zeichen auf dem Display dargestellt werden. Ausgabe des Zeichens ab der aktuellen Cursorposition.

Prototyp:

```
void _asc_out ( unsigned char );
```

Funktion **_lcd_str(char *ptr)**:

Übergeben wird ein Zeiger auf ein char - Array im ext. RAM. Dieses Array wird als Text ab der aktuellen Cursorposition ausgegeben. Ende der Ausgabe 0-Byte (String-Ende-Kennung).

Prototyp: **_void lcd_str (char *ptr);**

Beispiel:

```
unsigned text[ ] = ( Guten Tag ) ;

lcd_clr( );                     // Anzeige löschen
_lcd_curs ( 3 );                // Cursor setzen
_lcd_str (text );               // Ausgabe Text
```

Laboraufbau:

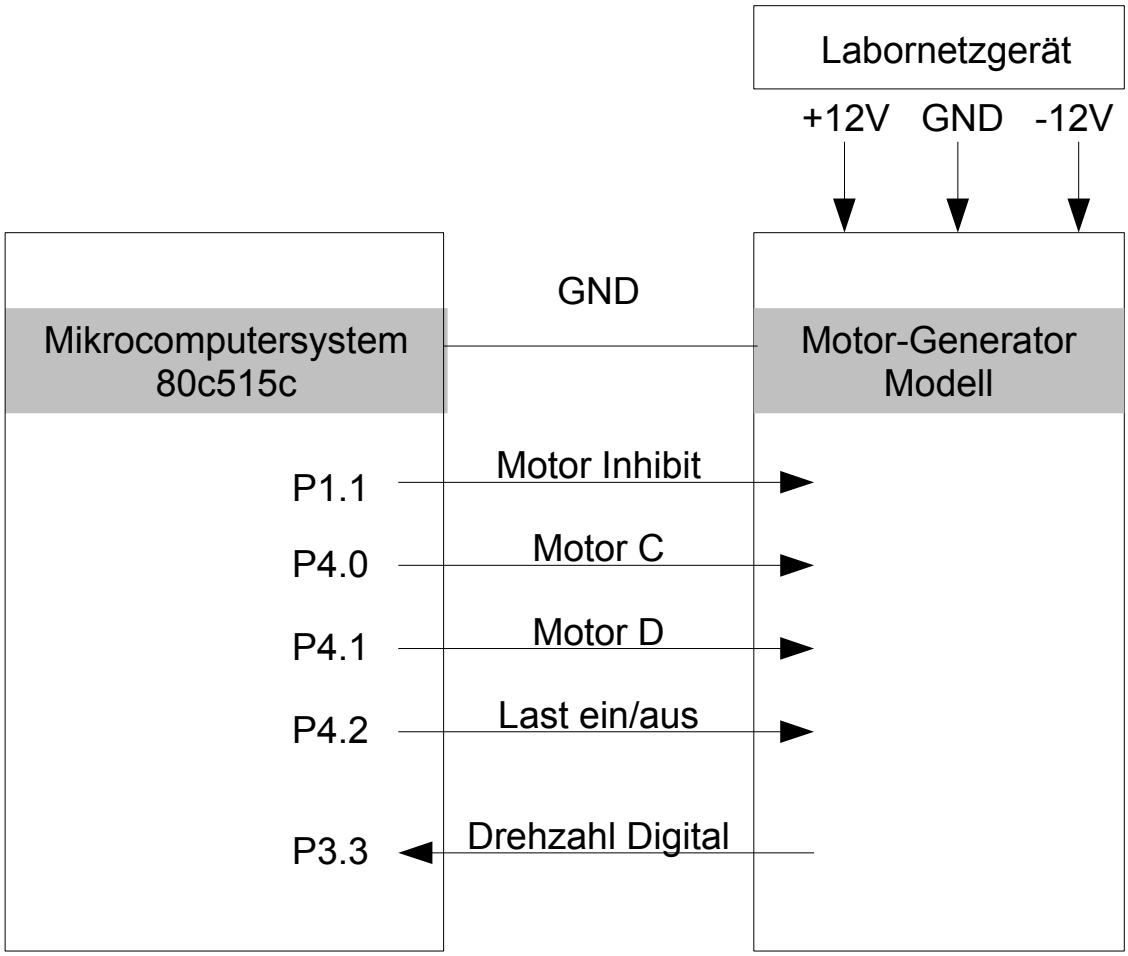

Abbildung 3: Verbindung von Modell mit Mikrocontroller

Die beiden Steuersignale für den Motor **C** und **D** steuern eine Brückenschaltung, mit der der Motor in beiden Richtungen oder Stillstand gesteuert werden kann. Sind die Potenziale unterschiedlich läuft der Motor, sind sie gleich bildet dies eine Kurzschlussbremse. Bei richtiger, konstanter Konfiguration von C und D ist nur Vcc=12V und GND nötig.

Der Steuereingang **Motor Inhibit** schaltet die Motortreiber ab und lässt ihn so im Freilauf weiterdrehen. Deshalb wird die Pulsweitenmodulation über diesen Steuereingang durchgeführt.

Analoge Drehzahl: Die analoge Drehzahlmessung wird nicht benötigt und kann offen bleiben.

GND ist das Bezugspotential.

5 Anhang

5.1 Timer 0/1 für digitale Drehzahlmessung

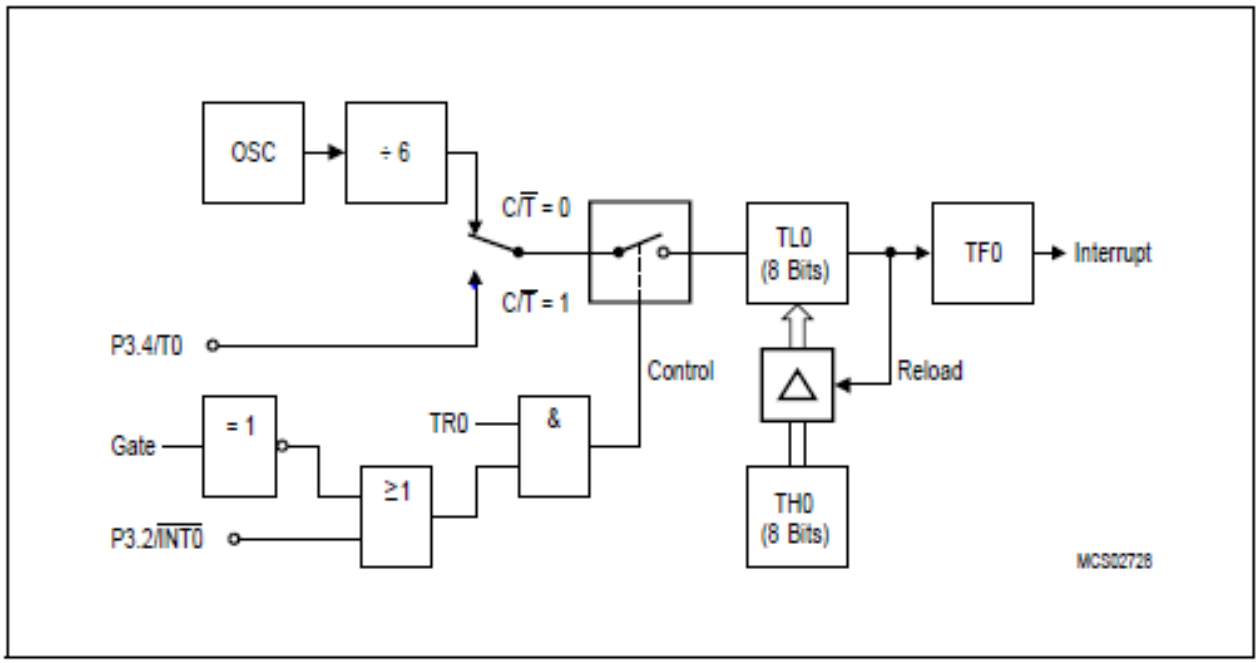

Figure 6-17 Timer/Counter 0,1, Mode 2: 8-Bit Timer/Counter with Auto-Reload

5.2 Timer 2 für Pulsweitenmodulation (PWM)

Der Timer 2 bietet vielfältige Möglichkeiten, die zudem in unterschiedlichen Derivaten verschieden ausgestaltet sind. Der Übersichtlichkeit halber wird hier nur das notwendigste herausgegriffen. Die angegebenen Port-Pins (P1.0 bis P1.3) können beeinflusst werden.

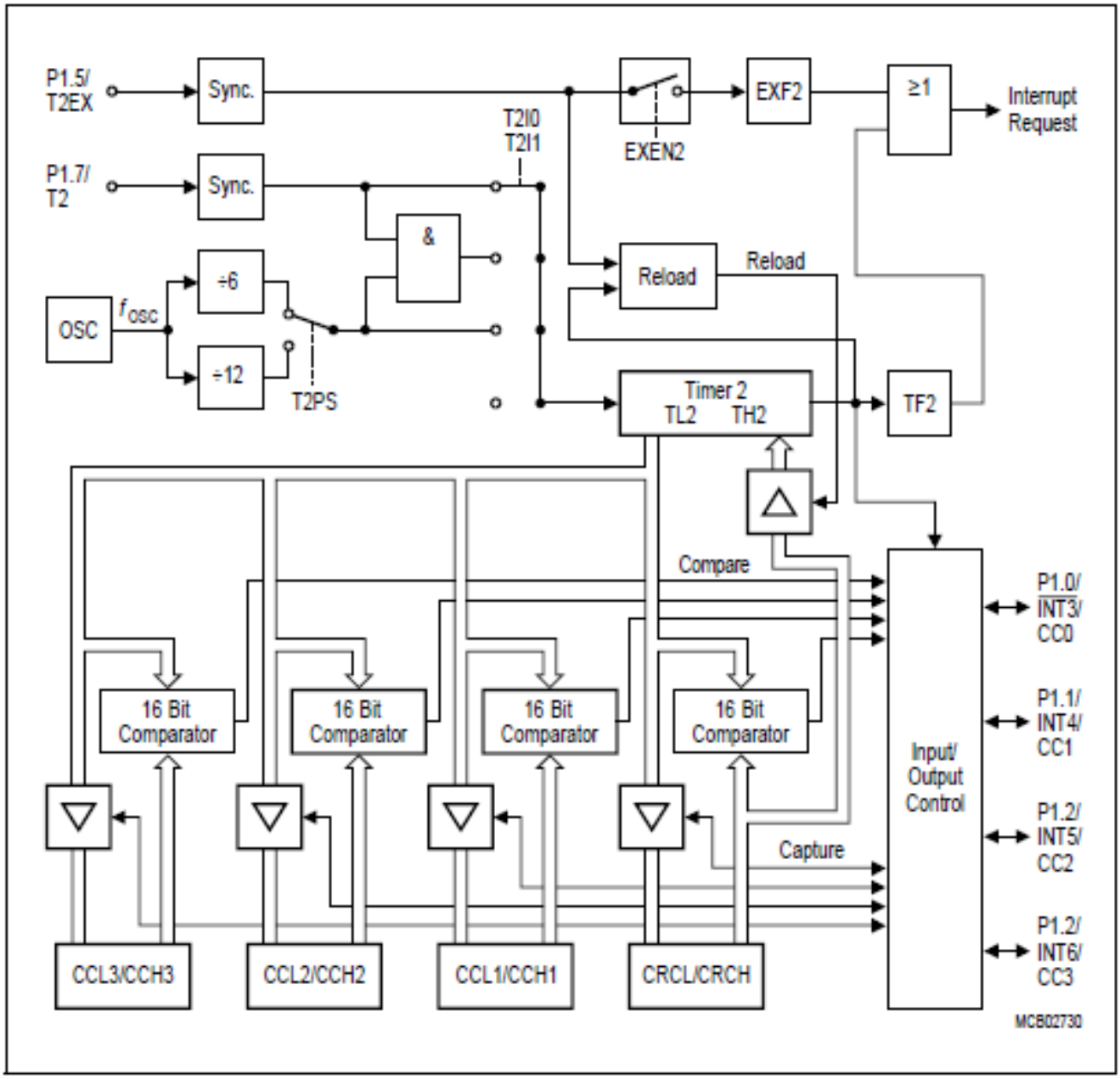

Figure 6-19 Timer 2 Block Diagram

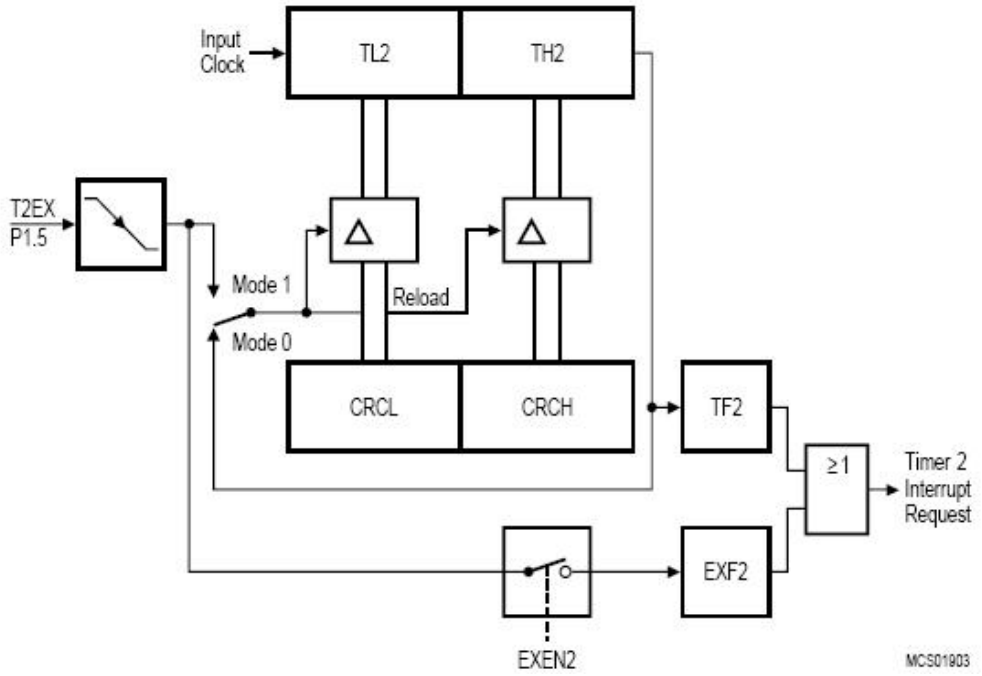

Abbildung 7: Timer 2 Reload Mode

Beim Überlauf des Timer 2 von 0FFFFH zu 0000H gibt es ein Overflow-Signal, welches den Timer neu lädt und gleichzeitig einen Interrupt verursacht. Lässt man den Timer frei laufen, muss man den Interrupt natürlich nicht auswerten (Timer 2 Interrupt disable).

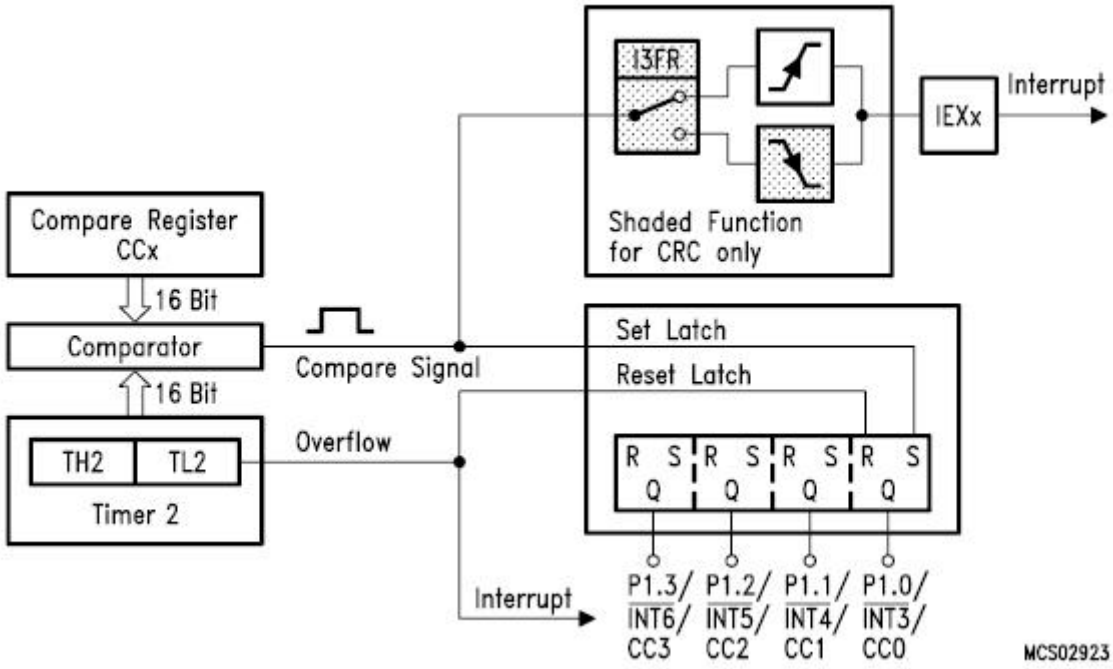

Abbildung 8: Timer 2 Compare Mode 0

Unabhängig vom Reload des Timer 2 kann man ihn mit einem Compare Register verbinden, welches dann innerhalb der Timerlaufzeit einen weiteren Interrupt bzw. ein weiters Steuersignal liefert. Auch dieser Interrupt muss für die Pulsweitenmodulation nicht unbedingt ausgewertet werden.

Die Latches der in Abb. 8 aufgeführten Port-Pins haben zusätzliche Set- und Reset-Eingänge, die durch die Timerfunktionen beeinflusst werden. So wird über das Compare-Signal (Timer = Compare-Register) und das Timer 2 Overflow-Signal z.B. die Pulsweitenmodulation erzeugt.

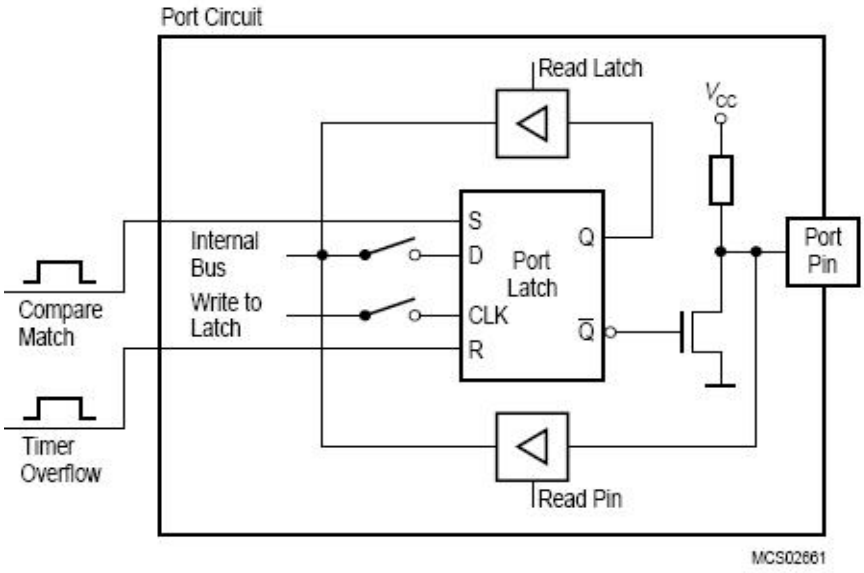

Abbildung 9 Port-Latch in Compare-Mode 0

Der Timer 2 Overflow setzt den Ausgang auf logisch „low“. Die Gleichheit vom Timer 2 mit einem der Compare-Register setzt den Ausgang auf logisch „high“.

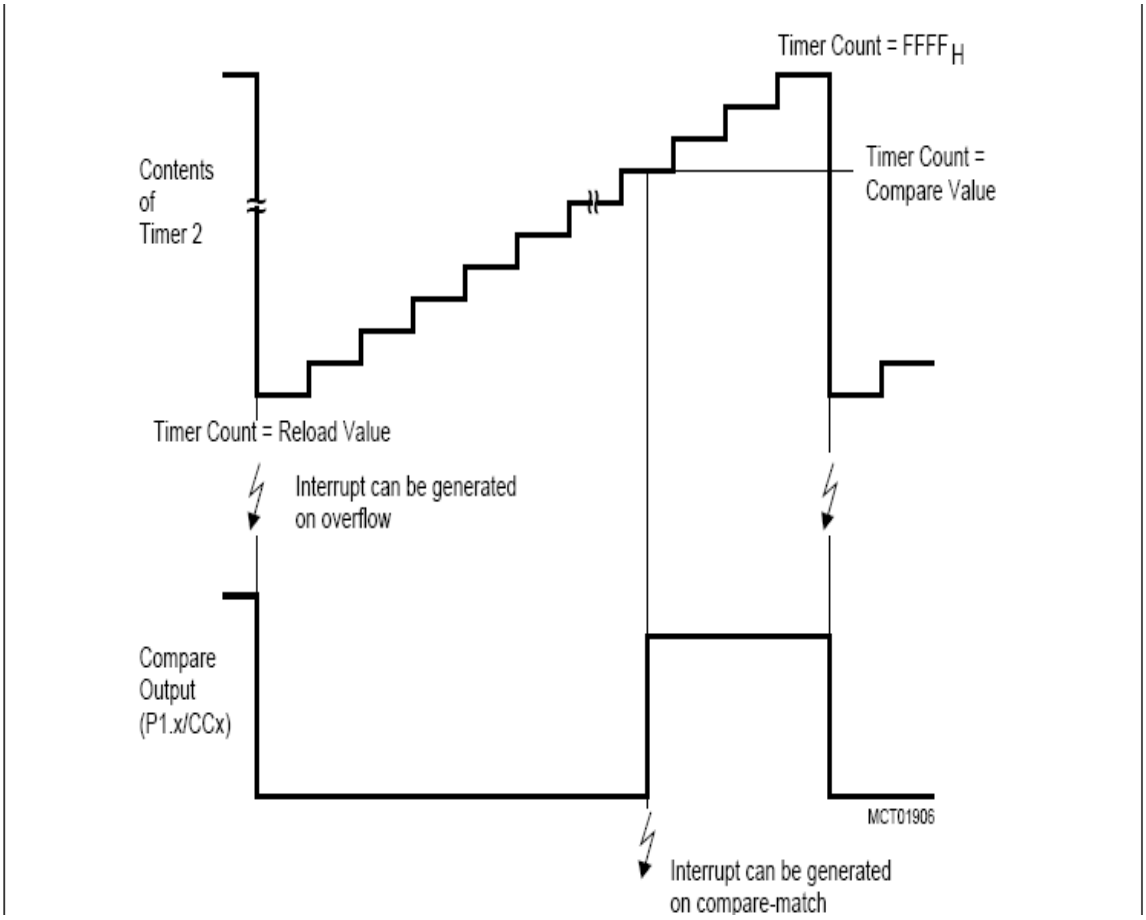

Abbildung 10: PWM Erzeugung