MSP430 -

Intelligenter Stromsparer Teil 2

Nach der Beschreibung der grundlegenden peripheren Module der MSP430-Mikrocontroller Familie im ersten Teil erläutern wir nun den MSP430 Befehlssatz anhand von Beispielen und zeigen, wie ein Programm für die Entwicklungsumgebung IAR KickStart aussieht.

MSP 430 praktisch

Um den Einstieg einfach zu gestalten, soll in diesem und dem nächsten Teil mit Hilfe der über die MSP 430-Internet-Seite kostenlos erhältlichen Entwicklungsumgebung „IAR-KickStart" eine kleine Anwendung zum MSP 430 entwickelt werden. Ziel ist es dabei, nach dem Kennenlernen der Befehle eine einfache Fahrstuhlsteuerung zu programmieren. Dabei soll es möglich sein, über vier Tasten die Etage auszuwählen. Auf einer 7-Segment Anzeige wird dargestellt, in welcher Etage sich der Fahrstuhl gerade befindet. Erreicht er die gewählte Etage, so soll dies ein akustisches Signal anzeigen. Die Bewegung des Fahrstuhls zwischen zwei Etagen wird optisch mittels eines LED-Lauflichtes markiert.

Grundlegender Programm-Aufbau

Jedes Mikrocontrollerprogramm muss ein gewisses Format einhalten, damit der Assembler das Programm in Maschinensprache übersetzen kann. Für die IAR KickStart-Entwicklungsumgebung bedeutet dies, dass in der ersten Spalte des Programmlistings immer nur Sprungmarken, in der zweiten Spalte die MSP430-Befehle, in der dritten Spalte die Operanden und in der letzten Spalte die Kommentare zu finden sind. Die einzelnen Spalten sind durch Leerzeichen oder Tabulatorzeichen voneinander getrennt. Ein Beispiel, wie ein Programm aufgebaut sein kann, ist im Programmlisting 1 gezeigt.

In diesem Beispielprogramm sind die verschiedenen Spalten gut erkennbar.

In der 1. Spalte – immer direkt am Zeilenanfang – befinden sich die Sprungmarken (Start, ReStart, Schleife, ...).

Die zweite Spalte enthält die MSP430-

Programm-Listing 1

```
;***  Programm Beispiel:  Lauflicht
#include „msp430x14x.h“
;--- Hauptprogramm:
      RSEG CODE
Start       MOV   #0A00h,SP         ; Initialisiere Stack Pointers
            MOV   #05A80h,&WDTCTL   ; stoppen des Watchdog
            MOV.b #0h,&P5OUT        ; P5.0 bis P5.7 = L
ReStart     MOV.b #0FFh,&P5DIR      ; P5.0 bis P5.7 = Ausgänge
            MOV.b #01h,&P5OUT       ; LED an P5.0 ist ein
Schleife    CALL  #warte              ; warte Schleife
            RLA.b &P5OUT            ; rotiere P5OUT nach links
            JNZ   Schleife
            JMP   ReStart
;
; Unterprogramm
warte       MOV   #5000h,R5
warte1      DEC   R5                ; Warteschleife
            JNZ   warte1
            RET
;
; Interrupt Vektor
      ASEG 0FFE0h+RESET_VECTOR
            DW    Start             ; Reset Interrupt Vektor

      end
```

Assemblerlisting 1: Lauflicht

Befehle, z. B. MOV, DEC, RET, usw. Man kann drei verschiedene Befehlsformate unterscheiden. Es gibt Sprungbefehle wie „JMP ReStart“ (= Sprung zur Zeile, ReStart’), „JNZ warte1“ (= wenn das Flag Zero gesetzt ist, dann springe zu ‚warte1’ ansonsten zum nächsten Befehl) oder „RET“ (= Rücksprung aus dem Unterprogramm).

Zudem gibt es Befehle mit einem oder zwei Operanden. Befehle mit einem Operanden sind zum Beispiel „RLA &P5OUT“. Dieser hat zur Folge, dass der Inhalt im Control-Register P5OUT um eine Bitstelle nach links verschoben wird. Ein Befehl mit zwei Operanden ist zum Beispiel „MOV #0A00h,SP“. Hierbei ist #0A00h die Quelle und SP das Ziel. Es wird hierdurch also die Konstante 0A00h in das Stack Pointer- Register SP geschrieben. Im Programm fällt auch auf, dass es verschiedene MOV-Befehle gibt, zum einen „MOV“ und zum anderen „MOV.b“. Durch den Zusatz „.b“ kann der Anwender definieren, dass der Ziel-Operand (z. B. Speicherzelle) byteweise beschrieben werden soll. Setzt man den Befehl ohne den „.b“-Zusatz ein, so erfolgt ein wortweises Beschreiben des Ziel-Operanden. Besonders wichtig ist die richtige Adressierung von Peripherie-Control-Registern. Die jeweilige Länge des Peripherie-Control-Registers – ob Byte oder Word – ist im User’s Guide des Controllers beschrieben.

Die vierte Spalte beinhaltet die Kommentare. In jeder Zeile wird alles, was nach einem Semikolon (;) steht, bei der Assemblierung nicht berücksichtigt.

Assembler-Direktiven

Im Programmlisting 1 finden sich neben den MSP430-Befehlen noch andere Anweisungen (#inlcude, RSEG, ASEG, DW,...). Diese Anweisungen sind Assembler-Direktiven. Hierdurch wird dem Assembler mitgeteilt, wie er den nachfolgenden Mikrocontrollercode behandeln soll. Mittels „#include“ erfolgt zum Beispiel das Kopieren einer anderen Datei in das Listing. Mit „RSEG“ und „ASEG“ wird dem Assembler und Linker mitgeteilt, wo er den nachfolgenden Code im Mikrocontroller-Speicher ablegen soll. Die Assembler-Direktive „DW“ ermöglicht es, eine Konstante (16 Bit) direkt in den Mikrocontroller-Speicher zu schreiben. Dies haben wir beispielsweise im Programm mit der Startadresse des Programms realisiert.

Programm-Beispiel Lauflicht

Das Programmlisting 1 zeigt die wohl am häufigsten exerzierte Grundaufgabe beim Kennenlernen eines Mikrocontrollers: ein Lauflicht. Es dient hier zunächst nur als „Trockenübung”, um die Programmstruktur kennen zu lernen, die praktische Realisierung mit einer kleinen Universalplatine folgt im nächsten Teil der Serie.

Mittels der Include-Anweisung werden zu Beginn des Programms die Definitionen der Peripherie-Control-Register eingefügt. Für jeden MSP430-Typ existiert eine eigene Header-Datei - „msp430x14x.h“ zum Beispiel für die MSP430F149-Varianten. Diese Header-Dateien sind im IAR-System-Verzeichnis („Programme/IAR Systems/ew23/430”) von KickStart im Verzeichnis INC zu finden. Mit einem Text-Editor kann man diese Dateien öffnen und anschauen.

Nach der Include-Anweisung wird mittels „RSEG CODE“ angegeben, dass nachfolgender Programmcode im Speicher-Segment CODE (= Flash-Speicher beim MSP430F149) zu platzieren ist. Es erfolgt nun die Initialisierung des Stack Pointers, indem die letzte Adresse des RAMs in SP geschrieben wird. Danach soll das Programm das Watchdog-Modul abschalten. Diese beiden Befehle sollten in jedem Programm durchgeführt werden, um die Funktionalität des Stacks zu gewährleisten und das ständige Generieren eines Resets durch den Watchdog zu verhindern.

Die Realisierung des Lauflichtes selbst ist recht einfach. Zunächst definiert man alle Ein-/Ausgänge des digitalen Ports P 5 als Ausgang (MOV.b #0FFh, &P5DIR). Beachten Sie, dass eine hexadezimale Zahl niemals mit einem Buchstaben beginnen darf. Sie muss immer mit einer Ziffer (0-9) beginnen. Also, wenn man die hexadezimal Zahl FFh verwenden will, so schreibt man im Programm 0FFh.

Mittels des Befehls MOV#01h, &P5 OUT erfolgt das Einschalten der ersten LED. Dabei werden der Pin P5.0 auf „H“-Pegel und alle anderen Port-Pins auf „L“-Pegel gesetzt.

Der CALL-Befehl ruft ein Unterprogramm auf. Bevor in das Unterprogramm gesprungen wird, erfolgt eine automatische Zwischenspeicherung der Rücksprung-Adresse auf dem Stack. Im Unter-

R0= PC=Programmzähler
R1= SP=Stack Pointer
R2=SR=Status Register
R3=reserviert
R4
⇓
R15

Die Register R4 bis R15 können für eigene Zwecke verwendet werden.

Bild 6: MSP-430-CPU-Register

programm wird dann das CPU-Register R5 mit einem Wert geladen und anschließend das Register R5 dekrementiert. Solange der Inhalt von R5 ungleich Null ist, führt der JNZ-Befehl einen Sprung zur Zeile mit der Sprungmarke „warte1“ aus. Ist die Null erreicht, so erfolgt das Abarbeiten des nächsten Befehls (RET). Der RET-Befehl kennzeichnet das Ende des Unterprogramms. Er veranlasst, dass die auf dem Stack gespeicherte Rücksprung-Adresse in den Programmzähler zurück kopiert wird und die Programmabarbeitung im Hauptprogramm in der Zeile „RLA.b &P5OUT“ fortfährt.

Die Zeile „RLA.b &P5OUT“ führt einen Rotierbefehl aus. Dabei erfolgt ein Verschieben des Control-Register-Inhalts P5OUT um eine Bitstelle nach links. In das unterste Bit wird aufgrund des RLA-Befehles eine Null geschoben. Durch mehrmaligen Aufruf dieser Zeile „schiebt” sich so eine leuchtende LED durch unsere LED-Reihe, und damit ist das Lauflicht realisiert. Ein JNZ-Befehl prüft die Anzahl der Schleifendurchläufe, welche die acht LEDs eine nach der anderen aufleuchten lässt. Das Zero Flag ist nämlich solange nicht gesetzt, wie der RLA-Befehl im P5OUT Register mindestens ein Bit gesetzt lässt. Wird also der RLA-Befehl ausgeführt, sowie das gesetzte Bit aus dem P5OUT Register herausgeschoben und dadurch der Inhalt des P5OUT Register gleich Null, so erfolgt auch das Setzen des Zero Flags. Ein gesetztes Zero Flag veranlasst dann auch ein Verlassen der Schleife.

Abschließend definiert man mit Hilfe der Zeilen

```
ASEG 0FFE0h+RESET_VECTOR
DW   Start
```

die Startadresse unseres Programms. Es wird hierdurch die Startadresse, die durch die Sprungmarke Start definiert ist, in die Adresse FFFEh geschrieben. Diese Adresse befindet sich in der Interrupt-Vektor-Tabelle, die im Bereich FFE0h - FFFFh liegt. Die Adresse FFE0h+RESET_ VECTOR (=FFFEh) definiert die Einsprungadresse, falls ein Power-up-Reset (bei Anlegen der Spannung an den Controller), ein externer Reset (verursacht durch Drücken des Reset-Tasters) oder ein Reset, bedingt durch einen Watchdog-Überlauf, ausgelöst wird.

Adressierungsarten

Im ersten Programm haben wir nun den grundlegenden Programmaufbau sowie erste Befehle kennengelernt. Nun wollen wir uns näher mit den verschiedenen Adressierungsarten bei der MSP430- Programmierung befassen. Im besprochenen Programmbeispiel haben wir bereits verschiedene Adressierungsarten benutzt. Unter Adressierungsarten versteht man, wie der Quell-Operand und der Ziel-Operand angewandt werden. So besteht die Möglichkeit, im Quell-Operand eine Konstante und als Ziel-Operand ein CPU Register zu verwenden. Der Befehl könnte wie folgt aussehen:

```
MOV #0A00h,SP
```

Anhand dieses Beispiels ist erkennbar, dass man für den Quell-Operanden und den Ziel-Operanden verschiedene Adressierungsarten einsetzen kann.

Im Quell-Operanden wurde eine Konstante verwendet. Als Kennzeichen für eine Zahl dient das „#“ Zeichen. Diese Adressierungsart nennt man „Immediate Addressing Mode“.

Im Ziel-Operanden ist das CPU-Register SP eingesetzt. Die verschiedenen CPU-Register sind im Bild 6 in einer Übersicht dargestellt. Diese Adressierungsart heißt „Register Addressing Mode“.

Wird eine Adresse im Speicher (RAM oder Flash) angesprochen, kann man dies mittels des „Absolute Addressing Mode“ oder „Symbolic Addressing Mode“ realisieren. Beispiele hierzu sind:

```
MOV   #05A80h,&WDTCTL
; Absolute Addressing Mode
oder
MOV   #05A80h,WDTCTL
; Symbolic Addressing Mode
```

Beide Adressierungsarten wurden dabei jeweils mit dem Ziel-Operanden verwendet. Beide ermöglichen es auch, einen Wert in eine bestimmte Adresse zu schreiben. Bei WDTCTL handelt es sich um ein Symbol, das in der Header-Datei für den Prozessor mit der Adresse 120h definiert ist.

Beispiel 7-Segment-Ansteuerung

Eine weitere Adressierungsart wollen wir uns am Beispiel der Ansteuerung einer 7-Segment-Anzeige ansehen. Die Anzeige besteht aus acht LEDs (inklusive des Punktes). Damit die Anzeige die Ziffern 0 bis 9 darstellt, sind die acht LEDs entsprechend anzusteuern. Wenn man beispielsweise eine Drei dargestellt haben möchte, so müssen

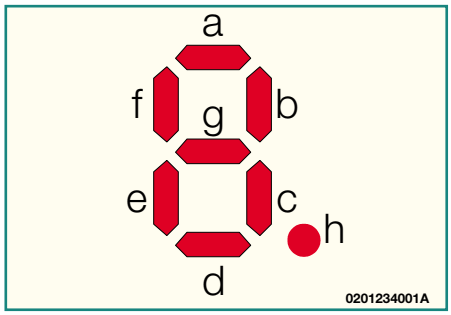

Bild 7: 7-Segment-Anzeige

- gemäss dem Bild 7 - die Segmente a, b, c, d und g in der Anzeige eingeschaltet werden. Es wäre nun wünschenswert, die für die Anzeige aller Ziffern notwendigen Kombinationen für das P4OUT Register aus einer Tabelle entnehmen zu können. Mittels eines Registers, das die darzustellende Zahl (0 bis 9) enthält, soll anhand der Tabelle der korrekte Wert definiert werden. Die Adressierungsart, mit der dies möglich ist, heisst „Indexed Mode“.

Im Programm-Listing 2 ist ein Programm gezeigt, das die eben besprochene Ausgabe über eine 7-Segment-Anzeige ermöglicht. Dabei kommt der „Indexed Mode” zum Einsatz:

```
MOV.b Tabelle(R4),&P4OUT
```

Abhängig von R4 wird aus der Tabelle, die bei der Adresse „Tabelle“ startet, ein Byte ausgelesen und in das Peripherie-Control-Register P4OUT geschrieben. Ist R4 zum Beispiel Null, so wird (Tabelle + 0) also auch eine Null angezeigt. Ist R4 dagegen Drei, so erfolgt (Tabelle + 3) die Darstellung einer Drei.

Damit hätten wir bereits zwei Programmteile unserer Fahrstuhlsteuerung realisiert - die Ansteuerung der 7-Segment-Anzeige und die Realisierung des Lauflichts.

Diese beiden Programmbeispiele können Sie bereits jetzt mit Hilfe der IAR KickStart-Entwicklungsumgebung und dem darin enthaltenen MSP430-Simulator auf Ihrem Computer simulieren.

Erhältlich ist die kostenlose, für Assemblerprogrammierung nicht eingeschränkte Version im Internet. Download unter:

www.ti.com/sc/msp430 unter dem Link „Tool Updates“ und dort „Free Tools“. Hier findet man „KickStart, complete“

Um die besprochenen Programme zu testen, ist die „Embedded Workbench“ der IAR KickStart-Entwicklungsumgebung zu starten. Dort legt man zuerst ein neues Projekt an (File|New|Project), schreibt dann das Assemblerprogramm (File|New|Source Text) und speichert dieses ab.

Bevor das Projekt assembliert werden kann, muss man die Assemblerprogramme dem Projekt hinzufügen (mittels Project|Files) und es sind die korrekten Projekteinstellungen durchzuführen. Drei

Programm-Listing 2

```
;***  Programm Beispiel:  Sieben Segment Anzeige
#include "msp430x14x.h"
;--- Hauptprogramm:
      RSEG CODE
Start       MOV   #0A00h,SP          ; Initialisiere Stack Pointers
            MOV   #05A80h,&WDTCTL    ; stoppen des Watchdog
            MOV.b #0FFh,&P4OUT       ; Sieben-Segment Anzeige aus
            MOV.b #0FFh,&P4DIR       ; P4.0 bis P4.7 = Ausgange
ReStart     MOV   #0h,R4
Schleife    MOV.b Tabelle(R4),&P4OUT ; zeige Nummer in Anzeige
            CALL  #warte
            INC   R4                 ; R4 = R4 + 1
            CMP   #0Ah,R4
            JEQ   ReStart
            JMP   Schleife
;
; Unterprogramm
warte       MOV   #0FFFFh,R5
warte1      NOP                      ; Warteschleife
            DEC   R5
            JNZ   warte1
            RET
;
; Tabelle für Sieben Segment Anzeige
a     EQU   01h   ; da die LEDs an Vcc angeschlossen
b     EQU   02h   ; sind, müssen die Werte hier
c     EQU   04h   ; invertiert werden:
d     EQU   08h   ; damit LED a leuchtet muss
e     EQU   10h   ; xxh in P5OUT geschrieben werden
f     EQU   20h
g     EQU   40h
h     EQU   80h
Tabelle     DB   0FFh-(a+b+c+d+e+f)      ; "0" - Byteweises abspreichern
            DB   0FFh-(b+c)              ; "1" - in einer Tabelle. Dadurch
            DB   0FFh-(a+b+d+e+g)        ; "2" - kann einfach über einen
            DB   0FFh-(a+b+c+d+g)        ; "3" - Zeiger die einzelnen
            DB   0FFh-(b+c+f+g)          ; "4" - Zeilen der Tabelle
            DB   0FFh-(a+c+d+f+g)        ; "5" - angesprochen werden.
            DB   0FFh-(a+c+d+e+f+g)      ; "6"
            DB   0FFh-(a+b+c)            ; "7"
            DB   0FFh-(a+b+c+d+e+f+g)    ; "8"
            DB   0FFh-(a+b+c+d+f+g)      ; "9"
;
; Interrupt Vektor
      ASEG 0FFE0h+RESET_VECTOR
            DW   Start               ; Reset Interrupt Vektor

      end
```

Assemblerlisting 2: 7-Segment-Anzeige

Schritte sind bei der Projekteinstellung wichtig:

- korrektes Linker Command File wählen (Project|Options|XLINK|Include: hier wählen Sie „MSP430F149A.xcl“ im Pfad Programme/IAR Systems/ew23/430/ICC430).
- entsprechenden Debug-Modus wählen (Project|Options|CSpy|Driver: hier wählen Sie „Simulation“)
- und schließlich das richtige Chip Description File (Project|Options|Cspy: hier wählen sie aus dem Verzeichnis Programme/IAR Systems/ew23/430/cw430 die Datei „msp430F14x.ddf“)

Anschließend ist es möglich, mittels Project|Debugger die Simulation der Assemblerprogramme zu starten.

Mehr und sehr ausführliche Informationen über den Befehlssatz sind im „MSP430x1xx Family User’s Guide” zu finden (Internet: www.ti.com/sc/msp430, unter „User Guides“).

Im nächsten Teil der Artikelserie betrachten wir die Programmteile „Tastatur-Auswertung” und „Ansteuerung des Summers” für unser Fahrstuhlprogrannm. Außerdem zeigen wir, wie man das gesamte Fahrstuhlsteuerungs-Projekt in eine entsprechende Hardware lädt und dort testet. ELV

Referenzen:

[1] MSP430 Internet Seite: www.ti.com/sc/msp430

[2] MSP430 Datenblätter: Link „Product List“ auf MSP430 Internet Seite [1]

[3] MSP430x1xx Family User’s Guide, SLAU049A

[4] MSP430x3xx Family User’s Guide, SLAU012

[5] MSP430x4xx Family User’s Guide, SLAU056A