Digitale Signal-Prozessoren DSP Teil 5

Grundlagen und praktische Anwendungen von DSPs

Die Beschreibung praktischer Programmierbeispiele soll zur Vertiefung des Umgangs mit DSPs dienen.

11. Programmbeispiele für den TMS320C50

Der folgende Abschnitt soll anhand einiger Programmbeispiele den Einstieg in die Programmierung erleichtern. In das dargestellte Rahmenprogramm sind die anderen Programmteile einzubinden, und somit sind erste Tests mit dem Debugger möglich.

11.1 Die Grundinitialisierung des Prozessors (Tabelle 8)

Um mit dem Prozessor und der verfügbaren Software arbeiten zu können, sind einige Grundeinstellungen erforderlich. Diese hängen von der umgebenden Hardware, dem gewählten Speichermodell und nicht zuletzt von den Forderungen des Debuggers ab.

Mit diesem Rahmen ist die Arbeitsfähigkeit des Prozessors hergestellt, und es kann der Bereich für die eigenen Programme ausgefüllt werden. Innerhalb der Assemblerliste ist beliebig oft zwischen Daten- und Programmbereich durch die Wahl von **.text** und **.data** umschaltbar. Solange keine Adresse angegeben wird, setzt der entsprechende Bereich jeweils an das alte Ende an.

11.2 Der Aufbau von Schleifen

Für eine effektive Programmgestaltung ist die Möglichkeit der Schleifenkonstruktion ein wichtiger Punkt.

Die einfachste Form ist die Wiederholung eines einzelnen Befehls. Für diesen Fall steht der **Repeat-Befehl** (Tabelle 9) bereit, welcher mit einer konstanten oder variablen Anzahl arbeiten kann. Zu beachten ist (bei allen Schleifenbefehlen), daß eine Runde mehr gearbeitet wird als im Befehl numerisch angegeben. Es ist sinnvoll, diesen Befehl im Zusammenhang mit einer indirekten Adressierung und einem Auto-Increment zu nutzen.

Möchte man mehr als einen Befehl in die Schleife einbeziehen, so kann man den durch die Hardware unterstützten **Block-Repeat-Befehl** (Tabelle 10) verwenden. Er gestattet eine beliebig lange Schleife ohne einen Befehlsaufwand für die Schleifenabarbeitung. Die Zahl der Schleifen-

Tabelle 8: Grundinitialisierung

```
;  Grundrahmen für die Entwicklungskarte mit dem TMS320C5x für 40 MHz internen Takt
       .line    85              ; Zeilenzahl in der Liste
;
; ***************************************************************
;  Definition von Konstanten und Datenbereichen (zentrale Daten)
       .data    0100h           ; Anfang des Datenbereiches im DARAM-B0
esc    .const   27              ; Konstante ESC
tmp1.ber  1                     ; 1 Wort für tmp1
Tabelle .ber  16        ; Tabelle mit 16 Worten
; ---------------------------------------------------------------
;
ROM            .set     0               ; 0 = Debugger-Version
                                        ; 1 = ROM-Version
;
; ---- Haupt-Programm ---------------------------------------------
       .if      rom
       .text    0800h
       ...                      ; Interrupt-Tabelle
       .else
;  Debugger-Version
       .text    0a00h           ; Anfang unseres Programmes auf 0a00h
                                ; (der Debugger steht ab Adresse 0800h)
       .entry                   ; Startpunkt für den Debugger
       .endif
; ---------------------------------------------------------------
start:
;      Konfiguration   (Speicherseite 0 durch Reset)
       .if      rom             ; Definitionen im Debugger vorbelegt
;      splk     #0,imr          ; alle INT aus (nicht im Debugger !!)
       apl      #0f08h,pmst
       .endif
       opl      #0830h,pmst     ; RAM = 1; OVLY = 1; NDX = 0
                                ; IntPTR = 800h (liegt damit in der Tabelle, welche
                                ;                der Debugger bereitstellt)
       clrc     cnf             ; CNF = 0
       clrc     sxm             ; keine Vorzeichenerweiterung
;    Wait-Status
       splk     #0,cwsr; Wait-Mode (0..3 Waits)
       splk     #05555h,iowsr ; I/O - alles 1 Wait
       splk     #0ffffh,pdwsr   ; externer RAM : 3 Wait
;    Vorbereitung der Datenbereiche
       ldp      #100h           ; MEM-Pointer auf DARAM-B0 (0100h)
       lar      ar1,#block1     ; Anfangsadresse RAM-Bereich
       larp     ar1             ; AR1 ist aktiver Pointer
       zap                      ; AC = 0
       rpt      #1ffh           ; Wiederholung 512 mal
        sacl    *+              ; löschen 1/2k Speicher
;
; Init E/A-Komponenten
;      CALL  INI_EA                    ; Init einer peripheren Einheit
;
; ========== Große Arbeits-Schleife ==============================
;
hp1: ldp      #100h             ; MEM-Pointer auf DARAM-B0  (100h)

       eigene Programme
       b        hp1             ; ENDLOS !!
;
; ---------------------------------------------------------------
       .end
```

Tabelle 9: Repeat-Befehl

```
       lar      ar1,#tabelle    ; die Anfangsadresse der Tabelle wird geladen
       larp     ar1             ; der Pointer 1 wird aktiviert
       rpt      #9              ; der Folgebefehl wird 10 mal ausgeführt
       in       *+,tor1         ; 10 Worte werden vom I/O-Kanal gelesen und
                                ; in der Tabelle abgelegt
```

durchläufe ist hierzu in das BRCR-Register zu laden. Wir wollen dazu das oben gezeigte Programm weiter ausbauen.

Diese Schleifenform hat den Nachteil, daß es den dafür erforderlichen Registersatz nur einmal gibt und sie somit nicht schachtelbar ist. Durch eine Softwareschleife oder die Kombination aus beiden kann dies umgangen werden. Der dafür nutzbare Befehl ist der **BANZ** (Tabelle 11) (Springe, wenn das ARX-Register ungleich Null).

Bei diesem Abschnitt sieht man auch deutlich die Arbeit mit der Pointerregisterumschaltung an den Befehlen ADD und BANZ.

11.3 Die Suche eines Minimums und eines Maximums

Da vor allem in der Meßtechnik diese Aufgabe oft zu lösen ist, wollen wir an einem Beispiel die Form der Vergleichsbefehle darstellen.

Außer diesen existiert noch ein „klassischer" Compare-Befehl, der sich jedoch schwer handeln läßt. In solchen Fällen ist eine Subtraktion und anschließende Ergebnisauswertung meist effektiver.

Für die Aufgabe „Min-Max" existieren Befehle, die diese Funktion speziell unterstützen - es ist der **CRGT** und der **CRLT** (**Tabelle 12**), die einen Vergleich von AC mit ACB durchführen. Im Ergebnis wird das CY-Flag gesetzt, und beim CRGT steht das Maximum beider Akkumulatoren im ACB (bzw. das Minimum beim CRLT). Das Programm wird dadurch sehr einfach.

Das Maximum bildet man mit der gleichen Befehlsfolge durch Austausch des CRLT gegen den CRGT und die Änderung des Startwertes auf 0.

11.4 Die Timernutzung im Interrupt (Tabelle 13)

Als letztes Beispiel soll die Nutzung des internen Timers stehen. Dieses Beispiel teilt sich dazu in zwei Komponenten, die Initialisierung und die Interrupt-Service-Routine.

Im Anschluß an die Initialisierung ist der Timer aktiv und löst alle 20µs einen Interrupt aus. Um eine Service-Routine aufzurufen, ist in der Interrupttabelle der Sprung auf diese einzutragen. Diese Tabelle steht am Anfang des Programmbereiches, in unserem Fall verwaltet sie der Debugger ab der Adresse 800 h. Da der Debugger diesen Interrupt nicht belegt, können wir ihn frei nutzen und wie im Programmabschnitt in Tabelle 14 auf unser Programm lenken. Die beiden Programmzeilen müssen dabei vor dem eigentlichen Programmbeginn stehen.

Nach abgeschlossener Initialisierung und Einbindung ist noch eine Service-Routine zu erstellen. Für die Bearbeitung von Interrupts besitzt der Prozessor zusätzliche Register zur Rettung des aktuellen Programmstatus. Dies betrifft solch wichtige Register wie die Akkumulatoren, das Produkt-Register, die Statusregister ST0 und ST1 (die z. B. den Zeiger auf den aktiven Pointer enthalten) und einige andere. Dadurch ist gesichert, daß eine schnelle und effektive Bearbeitung eines Interrupts durchgeführt werden kann. Der logische Schluß ist aber auch, daß dadurch keine Interrupts schachtelbar sind. Die Reorganisation dieser Register erfolgt durch den Befehl zur Rückkehr vom Interrupt. Unser einfaches Beispiel eines Softwarezählers stellt dies in Tabelle 15 anschaulich dar.

Um mit dem vorgestellten DSP-System sinnvoll experimentieren und arbeiten zu können, stehen verschiedene Ergänzungsbaugruppen für unterschiedliche Anwendungen zur Verfügung. Wir beginnen die Vorstellung der Erweiterungsbaugruppen im „ELVjournal" 2/97 mit einem Modul für die digitale und optische Ein- und Ausgabe mit jeweils 8 Stufen.

Tabelle 10: Block-Repeat-Befehl

```
        lar     ar1,#tabelle    ; die Anfangsadresse der Tabelle wird geladen
        lacl    #15             ; Anzahl der Durchläufe -1 in den AC
        samm    brcr            ; Schreibe AC nach BRCR (Zero-Page)
        zap                     ; AC := 0
        larp    ar1             ; der Pointer 1 wird aktiviert
        rptb    loop            ; die Befehle bis einschließlich LOOP werden ausgeführt
        in      *,tor1          ; vom I/O-Kanal lesen und in der Tabelle ablegen
loop:   add     *+              ; gelesenen Wert zum AC addieren, Adresse +1
        bsar    #4              ; AC 4 Bit rechts schieben ( AC:=AC/16)
; in AC steht der Mittelwert der in die Tabelle eingelesenen Daten
```

Tabelle 11: Banz-Befehl

```
        lar     ar1,#tabelle    ; die Anfangsadresse der Tabelle wird geladen
        lar     ar2,#15         ; Anzahl der Durchläufe -1
        zap                     ; AC := 0
        larp    ar1             ; der Pointer 1 wird aktiviert
loop:   in      *,tor1          ; vom I/O-Kanal lesen und in der Tabelle ablegen
        add     *+,0,ar2        ; gelesenen Wert zum AC addieren
        banz    loop,*-,ar1     ; dec(ar2), springe zu loop, wenn ar2<>0
        bsar    #4              ; AC 4 Bit rechts schieben ( AC:=AC/16)
```

Tabelle 12: CRLT-Befehl

```
        lar     ar2,#Tabelle    ; Adresse der Tabelle
        larp    ar2             ; Pointer AR2 aktiv
        lar     ar3,#255        ; Anzahl der Elemente
        lacl    #250            ; Startwert für Minimum
        exar                    ; Minwert in ACB

loop:   lacc    *+,0,ar3        ; Wert nach AC
        crlt                    ; Min in ACB
        banz    loop,*-,ar2     ; Schleife
        exar                    ; ACB nach AC tauschen
        sacl    minx            ; Min-Wert ablegen
```

Tabelle 13: Timernutzung im Interrupt

```
TimIni:
        di                      ; kein Interrupt erlaubt
        ldp     #0              ; Zero-Page einstellen
        splk    #199,prd        ; Timer-Register = 200
                                ; => 10MHz/200 = 50KHz = 20µs
        opl     #8,imr          ; Timer-Int erlauben
        splk    #20h,tcr        ; Start Timer
        splk    #8,ifr          ; clear + pending Interrupt
        ldp     #100h           ; Zeiger der Speicherseite herstellen
        ei                      ; Interrupt erlauben
        ret
```

Tabelle 14: Interrupttabelle

```
        .ps     0808h           ; Adresse für den Timer-Int (im Debugger-Bereich)
        b       timer           ; Timer-Int umlenken
```

Tabelle 15: Beispiel eines Softwarezählers

```
; Interrupt-Routine für Timer  (AC und Status werden automatisch gerettet)
timer:  lacc    cnt0            ; holen Zähler
        add     #1              ; erhöhen um 1
        sacl    cnt0            ; ablegen Zähler
        rete                    ; Interrupt beenden
```