Dr. Sieghard Schicktanz

Multitasking mit dem Z80-EMUF

Mit einigen Einschränkungen im Komfort läßt sich ein Systemkern für Multitasking bereits mit einem Einplatinencomputer minimalen Ausbaus realisieren, wie ihn z. B. der Z80-EMUF darstellt. Damit kann ein vollständiges Anwendungsprogramm in einem EPROM-Bereich von etwa 2 KByte Platz finden. Der Beschreibung des Systemkerns ist eine kurze Besprechung der Grundlagen für ein Multitaskingsystem vorangestellt.

Multitasking, gelegentlich – nicht ganz korrekt – auch Mehrprogrammbetrieb genannt, wird häufig in der Prozeßsteuerungstechnik angewandt. Man versteht darunter die Bearbeitung einer komplexen Aufgabe durch mehrere, quasi parallel ablaufende Teilprogramme, die weitgehend unabhängig voneinander jeweils einen überschaubaren Teil der Gesamtaufgabe ausführen.
Durch die Aufteilung erreicht man eine wesentliche Vereinfachung der Programmerstellung. Eine geschickte Verteilung der Aufgaben ermöglicht es, die Teilprogramme sehr einfach aufzubauen und – wichtig bei industriellem Einsatz – unabhängig voneinander zu erstellen, da die Schnittstellen einfach gehalten werden können.
Außerdem können alle Koordinationsaufgaben in einem zentralen Teilprogramm (Scheduler) zusammengefaßt werden, das unabhängig vom jeweiligen Einsatzfall ausgeführt ist und darüber hinaus weitgehend die Unabhängigkeit der einzelnen anwendungsbezogenen Teilprogramme (Tasks) unterstützt. Der Systemkern stellt alle benötigten Hilfsroutinen zur Verfügung, die zum (quasi) parallelen Bearbeiten der Teilprogramme gebraucht werden, ohne daß der Programmersteller dies besonders berücksichtigen muß.
Bei großen Prozeßrechensystemen stellt diese Funktionen bereits das dort eingesetzte Betriebssystem zur Verfügung. Der Scheduler macht allerdings nur einen kleinen Teil des Betriebssystems aus; mit einigen Einschränkungen im Komfort kann ein Multitaskingsystem, samt Anwendungsprogramm, bereits auf einem Minimalsystem wie dem Z80-EMUF [1] in nur 2 KByte EPROM (und 2 KByte RAM) realisiert werden.

Was ist Multitasking?

Die Grundfunktionen eines Multitaskingsystems sind im folgenden kurz erläutert. Jede Task, d. i. jedes Teilprogramm, das eine abgeschlossene Teilaufgabe bearbeitet, kann verschiedene Zustände einnehmen:

1. Nicht vorhanden
2. Angehalten
3. Bereit (aktiv)
4. Wartend

Der Zustand 1, nicht vorhanden, ist hier trivial, da in einem EPROM-System Tasks nicht neu hinzukommen oder gelöscht werden können. In diesem Fall sind alle möglichen Tasks also immer im System vorhanden, der Zustand 1 kann nicht auftreten.
Eine Task, die den Zustand 3, bereit, hat, wird vom Scheduler in Bearbeitung genommen, sobald dies möglich ist. Im einfachsten Fall wird die Bearbeitung solange durchgeführt, bis aus irgendeinem Grund eine Unterbrechung eintritt (kein Interrupt), z. B. auf Daten aus einer anderen Task gewartet werden muß oder die Bearbeitung beendet ist.
Der Zustand 2, angehalten, bedeutet, daß die Task nicht bearbeitet werden darf. Dies ist z. B. der Fall, wenn die letzte Bearbeitung beendet ist und neue Daten abzuwarten sind.
Der Zustand 4, wartend, schließlich wird dazu benutzt, definierte Zeitverzögerungen zu erreichen. Die Task wird erst dann aufgerufen, wenn ein vom System verwalteter Zähler leergezählt ist. Dieser Zähler wird durch eine zum System gehörige Task in festen Zeitintervallen dekrementiert.
Damit eine Task bearbeitet werden kann, muß sie vom Koordinationsprogramm, dem Scheduler, aufgerufen werden. Dies erfolgt nach einer bestimmten Strategie, die im Schedulerprogramm „eingebaut" ist. Die einfachste Aufrufstrategie ist der zyklische Aufruf, auch (amerikanisch) „Round Robin"-Strategie genannt. Dabei werden die Tasks in einer festen Reihenfolge nacheinander abgefragt und aktive Tasks sofort bearbeitet. Bei Unterbrechung einer Task wird jeweils die nächste aktive Task aufgerufen. Nach der letzten Task in der Reihenfolge wird wieder von vorn begonnen.
Der Scheduler hat dabei folgende Aufgaben:

1. Neustart/Wiederstart von aktiven Tasks
2. Umschalten zwischen den Tasks
3. Anhalten von Tasks

Eine wesentliche Vereinfachung des Systemaufbaus ist erreichbar durch einen „kooperativen" Aufbau der einzelnen Tasks. Das heißt, die Tasks sind so aufgebaut, daß sie nie durch langwierige Bearbeitungsfolgen oder durch Warteschleifen die Bearbeitung der anderen Tasks aufhalten. Dafür stellt der Systemkern die nötigen Hilfsmittel zur Verfügung.

Koordination von Tasks

Zusammen mit diesen sind im Systemkern die folgenden Koordinationsfunktionen verfügbar:

1. PAUSE	Unterbrechen einer Task (Rückkehr zum Scheduler)
2. DELAY	Vorübergehendes Stoppen einer Task (Wartend)
3. STOP	Anhalten einer Task
4. ENDE	Beenden einer Task (mit Reinitialisierung)
5. RESTART	Wiederholung einer Task (Neustart von Beginn an)
6. START	Start einer Task

Lediglich die Funktion 6 (Start einer Task) wirkt auf eine andere als die aktuell bearbeitete Task. Alle anderen Funktionen verändern nur den Zustand der aktuellen Task. Ein Aufruf dieser Funk-

tionen (1 bis 5) bewirkt außerdem jeweils die Rückkehr zum Scheduler und den Aufruf aller anderen momentan aktiven Tasks, bevor die aufrufende Task weiter bearbeitet wird.
Wie ist nun ein solches Koordinationsprogramm zu realisieren?
Es ist klar, daß ein direkter Aufruf der Tasks, z. B. über CALL-Befehle, nicht möglich ist, da die Anzahl der aufzurufenden Tasks unbekannt ist und die einzelnen Tasks so wenig wie möglich von der Existenz der übrigen Tasks beeinflußt werden sollen.
Der einfachste und üblicherweise benutzte Weg zum Feststellen der Task-Anzahl besteht darin, die Information, die das System von jeder Task braucht, in einer Tabelle oder Liste, der Task-Zustandsliste, zusammenzufassen.
Der einfachste Weg, den Tasks möglichst viel Unabhängigkeit zu geben, ist, dafür zu sorgen, daß beliebige Unterprogrammaufrufe, also auch Systemaufrufe, „bunt gemischt" ohne Rücksicht auf die Schachtelungstiefe vorkommen dürfen. Das verlangt aber, jeder Task einen eigenen „privaten" Stack zuzuordnen. Dort kann eine Task Daten zwischenspeichern, ohne andere Tasks oder den Scheduler zu beeinflussen.

Systementwurf

Im ersten Entwurfsschritt ist nun festzulegen, welche Daten der Scheduler für seine Arbeit benötigt und wie sie organisiert werden. Die größte Flexibilität gestattet die Auslegung als Liste von Datenblöcken, die jeweils die Daten für eine einzelne Task enthalten. Jedes Listenelement enthält einen Zeiger auf das nachfolgende Element, somit können die Taskdaten beliebig im Speicher verteilt sein. Ein Listenelement wird häufig als Task-Leitblock bezeichnet.
Zu jeder Task muß sicher die Startadresse in der Liste eingetragen sein, damit die Task überhaupt gestartet werden kann. Des weiteren muß zu jeder Task der Stackpointer abgelegt werden, um dem Scheduler die Umschaltung der Taskstacks zu ermöglichen.

Die Task-Zustandsliste

Natürlich muß der Scheduler den Zustand der Task kennen. Dafür ist ein Eintrag in der Liste nötig, der im hier beschriebenen System wie folgt aussieht:
Für jede Task ist ein sieben Bit breites Identifikationsfeld vorhanden, das den Start einer Task erlaubt, ohne daß etwas anderes als deren Identifikation bekannt sein muß.
Das Identifikationsfeld ist mit einem Flag zusammengefaßt, das die Zustände „angehalten" (Flag = 0) und „bereit" bzw. „wartend" (Flag = 1) unterscheidet. Die Zustände „bereit" und „wartend" werden unterschieden durch den Inhalt eines weiteren Bytes, das gleichzeitig als Wartezeitzähler (DELAY.) dient. Die Wartezeitfelder aller Tasks werden von einer zum System gehörenden Task bearbeitet, die regelmäßig per Interrupt gestartet wird und damit eine Systemzeiteinheit festlegt. Ohne weitere Maßnahmen ist damit eine Verzögerung um max. 255 Systemzeiteinheiten möglich.
Für unser Minimalsystem sieht ein Element der Taskzustandsliste damit also folgendermaßen aus:

1. Wort: (Kalt-)Startadresse der Task
2. Wort: aktueller Stackpointer der Task, bei Initialisierung: Endadresse des Taskstacks
3. Wort: 1. Byte: Task-Identifikation in Bits 0...6, aktiv/inaktiv-Flag in Bit 7
   2. Byte: Verzögerung in Systemzeiteinheiten
4. Wort: Zeiger auf nächstes Taskzustandselement (zyklisch!)

Der „Scheduler"

Nachdem nun die zu bearbeitende Datenstruktur festgelegt ist, kann das Programm entworfen werden, das damit arbeitet. Es gliedert sich in zwei Hauptteile: die Initialisierung und die Task-Aufruf-Schleife. Als Grundstock für ein funktionierendes Steuerprogramm muß unser Scheduler für die Voraussetzungen zum einwandfreien Arbeiten des Systems sorgen.
Als erstes (*Bild 1*) wird deshalb der Stackpointer mit der Anfangsadresse des Scheduler-Stackbereiches geladen. Dann wird die Task-Zustandsliste ins RAM kopiert. Dies ist nötig, weil ihre Elemente während des Programmablaufes verändert werden, was im EPROM nicht möglich wäre. Der ausgeführte Scheduler verwendet das Z80-Register IX als Basisregister für alle Operationen mit den Taskzuständen, deshalb muß es in dieser Programmumgebung (Kontext) immer auf einen gültigen Eintrag der Task-Zustandsliste zeigen. Es wird deshalb zunächst mit der Anfangsadresse

```
                    .MAIN. - Multitasking Scheduler fuer kleine Stand-Alone - Anwendungen
                    Initialisierung, .MAIN.-Modul!

                                .EXTERN TASKLISTE, TASKAREA, .TASKLISTE
                                .EXTERN USR.INIT, IR.INIT, SCHEDULER
                                .INTERN RESET, S.INIT

                                ; Initialisierung der Taskzustands-Infomation
                                ; transportiert alle Taskzustandselemente ins RAM
                                ; (die Werte im ROM muessen bereits die richtigen Links
                                ;  enthalten und bei AKTIV. und DELAY. richtig gesetzt sein!),
                                ; initialisiert die Taskstartadressen und
                                ; setzt den Scheduler-Stackpointer
0000'                       S.INIT:
0000'  31 0040"                 LXI   SP, SCH.STACK          ; Stackpointer setzen
0003'  21 0000:04               LXI   H, TASKLISTE           ; Liste der vorbesetzten
0006'  11 0000:05               LXI   D, TASKAREA            ; Taskzustandselemente
0009'  01 0000:06               LXI   B, .TASKLISTE          ; nach TASKAREA im RAM
000C'  EDB0                     LDIR                         ; kopieren
000E'  DD21 0000:05             LXI   X, TASKAREA            ; Basisadresse fuer SCHEDULER!
0012'  DD4604                   MOV   B, TASK.ID (X)         ; Id der 1. Task merken
0015'                       ..NEXT:
0015'  DD5E02                   MOV   E, TASK.SP (X)         ; Stackpointer in DE laden
0018'  DD5603                   MOV   D, TASK.SP+ 1 (X)      ; DE laden
001B'  CD 003A'                 CALL  RESET                  ; Startadresse setzen
001E'  DD5ED6                   MOV   E, NEXT.T (X)          ; Taskzustandselement-
0021'  DD5607                   MOV   D, NEXT.T+ 1 (X)       ; adresse ...
0024'  D5                       PUSH  D
0025'  DDE1                     POP   X                      ; ... weiterschalten
0027'  DD7E04                   MOV   A, TASK.ID (X)
002A'  B8                       CMP   B                      ; schon einmal rum?
002B'  20EB                     JR    NZ, ..NEXT             ; nochmal
002D'  DDE5                     PUSH  X                      ; Task-Zustands-Zeiger aufheben
002F'  CD 0000:07               CALL  USR.INIT               ; Anwendungs-Initialisierung
0032'  CD 0000:08               CALL  IR.INIT                ; InterRupt-INITialisierung
0035'  DDE1                     POP   X                      ; Zeiger zurueck!
0037'  C3 0000:09               JP    SCHEDULER              ; und los gent's

                    ; Hilfsroutine zum Setzen der (Kalt) Startadresse der aktuellen Task
                    ; als Aufrufadresse fuer den SCHEDULER am Taskstack

003A'               RESET:  ; zerstoert IY und Akku!
                    ; PARAMETER:    Task-Stackpointer in DE
003A'  D5                       PUSH  D                      ; Task-Stackpointer
003B'  FDE1                     POP   Y                      ; nach IY
003D'  DD7E00                   MOV   A, 0 (X)               ; Task-Startadresse
0040'  FD7700                   MOV   0 (Y), A               ; auf Task-Stack
0043'  DD7E01                   MOV   A, 1 (X)               ; als RETurnadresse
0046'  FD7701                   MOV   1 (Y), A               ; ablegen
0049'  C9                       RET
```

Bild 1. Initialisierung einer Task-Verwaltung

des Listenspeicherbereichs (im RAM) geladen. Bevor jedoch der erste Aufruf einer Task ausgeführt werden kann, müssen erst noch alle „privaten" Stacks der Tasks mit deren Kaltstartadressen vorbesetzt werden, da der eigentliche Aufruf per RET geschieht. Die Programmschleife, die das macht, hangelt sich mit Hilfe der Zeiger durch die ganze Taskzustandsliste. In der Schleife wird die Task-Identifikation jedesmal mit der Identifikation der ersten bearbeiteten Task verglichen; weil die Liste zyklisch aufgebaut ist, muß das Programm einmal wieder auf diesen Eintrag stoßen. Damit sind alle Einträge in der Liste bearbeitet. Die Scheduler-Initialisierung ist damit beendet. Normalerweise sind noch weitere vom jeweiligen Einsatzfall abhängige Vorbereitungen auszuführen, die von der Routine USR.INIT ausgeführt werden.

Bevor wir schließlich voll einsteigen können, müssen noch die Ein-/Ausgabeports eingestellt werden und die Interruptbearbeitung muß vorbereitet und freigegeben werden (*Bild 2*). Nach diesen Vorbereitungen ist jetzt endlich alles

```
SCHEDU - Multitasking Scheduler fuer kleine Stand-Alone - Anwendungen
Interrupt-INITialisierung

                        .EXTERN IRVTABLE

0000'                   IR.INIT:

                                ; Interrupt-Vektoren und Ports initialisieren

0000'   21 0000:04              LXI     H, IRVTABLE             ; Adresse der Vektortabelle laden
0003'   7C                      MOV     A, H
0004'   ED47                    STAI                            ; High Byte -> IR-Basis-register
0006'                   ..SETUP:
0006'   21 0019'                LXI     H, INITABLE             ; Tabelle der Portinitialisierungen


0009'   46                      MOV     B, M                    ; Anzahl der Eintraege
000A'   23                      INX     H
000B'                   ..SCHLEIFE:
000B'   C5                      PUSH    B                       ; Anzahl festhalten
000C'   4E                      MOV     C, M                    ; Port und ...
000D'   23                      INX     H
000E'   46                      MOV     B, M                    ; ... Byteanzahl laden
000F'   23                      INX     H
0010'   EDB3                    OUTIR                           ; Rest geht nach draussen
0012'   C1                      POP     B                       ; Zaehler zurueck
0013'   10F6                    DJNZ    ..SCHLEIFE              ; so oft wie angegeben
0015'   ED5E                    IM2                             ; Interrupt spezifizieren
0017'   FB                      EI                              ; und freigeben
0018'   C9                      RET                             ; fertig

                        ; Festdaten:    Port-Initialisierungswerte (Beispielwerte)

0019'                   INITABLE:
0019'   09              .BYTE   9                               ; Anzahl der SETUP-Definitionen

001A'   02050BCFFFB7    .BYTE   ANZEIGE+ CTRL,  5, 8, 11001111B, 11111111B, 10110111B, 00000000B
0021'   03050BCFFFB7    .BYTE   PROGRAMM+ CTRL, 5, 8, 11001111B, 11111111B, 10110111B, 01111111B
0028'   0C030A4F83      .BYTE   LOWTEMP+ CTRL,  3,10, 01001111B, 10000011B
0020'   0D030A4F03      .BYTE   HIGHTEMP+ CTRL, 3,10, 01001111B, 00000011B
0032'   1702A90D        .BYTE   PPICTRL,        2, 10101001B, (6 < 1 ! 1)
0036'   1E0300077D      .BYTE   CTC0,           3, 0, 00000111B, 125.
003B'   1F02C7FA        .BYTE   CTC0+ 1,        2, 11000111B, 250.
003F'   2002C7F0        .BYTE   CTC0+ 2,        2, 11000111B, 240.
0043'   210143          .BYTE   CTC0+ 3,        1, 01000011B    ; wird nicht gebraucht
```

Bild 2. Initialisierung der Ports und Interrupt-Vektoren

```
SCHEDU - Multitasking Scheduler fuer kleine Stand-Alone - Anwendungen
Strategie: Round Robin

                        .EXTERN TASKAREA                ; RAM-Bereich der Taskzustandsliste
                        .EXTERN RESET
                        .INTERN SAVESP
                        .INTERN PAUSE, DELAY, STOP, END, RESTART, START

                        ; Datenstruktur fuer Taskzustandselemente

0007                    AKTIV.  = 7                     ; Flag-Bit
007F                    ID.MASK = 7FH                   ; Maske fuer Taskidentifikation
0002                    TASK.SP = 2                     ; Offset in Taskzustand
0004                    TASK.ID = 4                     ;       "
0005                    DELAY.  = 5                     ;       "
0006                    NEXT.T  = 6                     ;       "

                        ; Ein Taskzustandselement hat folgenden Aufbau:
                        ; 1. Wort: (Kalt-) Startadresse der Task
                        ; 2. Wort: aktueller Stackpointer der Task,
                        ;          bei Initialisierung: Endadresse des Taskstacks
                        ; 3. Wort:
                        ;       1. Byte: Taskidentifikation in Bits 0..6,
                        ;                aktiv/inaktiv-Flag in Bit 7
                        ;       2. Byte: Verzoegerung in Systemzeiteinheiten
                        ; 4. Wort: Zeiger auf naechstes Taskzustandselement (zyklisch!)


0000'                   SCHEDULER:
                        ; Eingang: IX mit Adresse eines Taskzustandselementes besetzt
                        ;          (Taskzustandselement muss richtig besetzt sein!)

                        ; Testen, ob Task aktiv
0000'   DDCB047E                BIT     AKTIV., TASK.ID (X)
0004'   2814                    JR      Z, NEXTTASK

                        ; Testen, ob Task wartend
0006'   DD7E05                  MOV     A, DELAY. (X)
0009'   B7                      ORA     A
000A'   200E                    JR      NZ, NEXTTASK

                        ; Task aufrufen
000C'   DDE5                    PUSH    X
000E'   ED73 0000"              SSPD    SAVESP                  ; Speicherplatz fuer Stackpointer
0012'   DD6E02                  MOV     L, TASK.SP (X)
0015'   DD6603                  MOV     H, TASK.SP+ 1 (X)       ; Task-Stackpointer laden
0018'   F9                      SPHL                            ; Stack umschalten
0019'   C9                      RET                             ; Bearbeitung aufnehmen

                        ; Rueckkehr aus der Task:       je nach Bearbeitungszustand ueber
                        ;                               verschiedene Eintrittspunkte

                        ; Fortschalten auf naechste Task
001A'                   NEXTTASK:       ; durch Pointer weiterhangeln
001A'   DD5E06                  MOV     E, NEXT.T (X)           ; Taskzustandselement-
001D'   DD5607                  MOV     D, NEXT.T+ 1 (X)        ; adresse nach
0020'   D5                      PUSH    D
0021'   DDE1                    POP     X                       ; Indexregister
0023'   18DB                    JR      SCHEDULER               ; weiter mit diesem

                        ; Routine zur Kontext-Umschaltung
0025'                   SWITCH:
0025'   FDE1                    POP     Y                       ; Rueckkehradresse holen
0027'   2A 0000"                LHLD    SAVESP                  ; Scheduler-Stackpointer retten
002A'   ED73 0000"              SSPD    SAVESP                  ; Task-Stackpointer
002E'   ED5B 0000"              LDED    SAVESP                  ; zwischenspeichern
0032'   F9                      SPHL                            ; Stack umschalten
0033'   DDE1                    POP     X                       ; Taskelementadresse zurueck
0035'   DD7302                  MOV     TASK.SP (X), E          ; Task-Stackpointer
0038'   DD7203                  MOV     TASK.SP+ 1 (X), D       ; ablegen
003B'   FDE9                    PCIY                            ; fertig, a la RETurn weiter

                        ; Von Anwendungsprogramm aufrufbare Eintrittspunkte:

003D'                   PAUSE:  ; Rueckgabe an Scheduler, wenn Warten noetig
                                ; keine Parameter, Rueckkehr wie aus Subroutine
                                ; alle benoetigten Register sind zu retten!
003D'   CD 0025'                CALL    SWITCH
0040'   18DB                    JR      NEXTTASK

0042'                   DELAY:  ; wie PAUSE, aber fuer spezifizierte Zeit
                        ; PARAMETER:    Verzoegerung in Akku
0042'   CD 0025'                CALL    SWITCH
0045'   DD7705                  MOV     DELAY. (X), A           ; Verzoegerung ablegen
0048'   18D0                    JR      NEXTTASK

004A'                   STOP:   ; Task inaktivieren, z.B. fuer Intertask-kommunikation
                                ; sonst wie PAUSE
004A'   CD 0025'                CALL    SWITCH
004D'   DDCB048E                RES     AKTIV., TASK.ID (X)     ; Task inaktiv setzen
0051'   18C7                    JR      NEXTTASK

0053'                   END:    ; Beenden einer Task mit Reinitialisierung der Startadresse
                                ; Stack  m u s s  leer sein !!!!!
0053'   CD 0025'                CALL    SWITCH                  ; Task-Stackpointer noch in DE
0056'   DDCB048E                RES     AKTIV., TASK.ID (X)     ; Task inaktiv setzen
005A'   CD 0000:05              CALL    RESET                   ; Eintrittsadresse reinitialisieren
005D'   18BB                    JR      NEXTTASK

005F'                   RESTART:; Wiederstart einer Task von Anfang an, evtl. mit Verzoegerung
                                ; Stack  m u s s  leer sein !!!!!
                        ; PARAMETER:    Verzoegerung in Akku
005F'   CD 0025'                CALL    SWITCH
0062'   DD7705                  MOV     DELAY. (X), A           ; Verzoegerung ablegen
0065'   CD 0000:05              CALL    RESET                   ; Eintrittsadresse reinitialisieren
0068'   18B0                    JR      NEXTTASK
```

Bild 3. Routinen zur Verwaltung der Tasks

getan, um die reguläre Programmbearbeitung aufnehmen zu können (*Bild 3*). Das Basiszeigerregister IX zeigt auf das erste Element der Task-Zustandsliste. Im ersten Schritt wird nun geprüft, ob die aktuelle Task angehalten ist. Wenn dies der Fall ist, wird sofort zur nächsten Task weitergegangen. Ansonsten wird im zweiten Schritt geprüft, ob der Wartezeitzähler läuft. Nur wenn dieser auf Null steht, ist die Task aktiv und kann aufgerufen werden.

Dazu wird zuerst das Basiszeigerregister auf den Scheduler-Stack gerettet. Der momentane Stackpointerinhalt wird dann an einer dafür reservierten Stelle abgespeichert. Jetzt kann das Stackpointerregister mit dem Task-Stackpointer geladen werden. Im Task-Stack steht als letzter Eintrag die Adresse des nächsten zu bearbeitenden Befehls der Task (am Anfang steht dort die Startadresse der Task). Ein einfaches RET startet jetzt die Bearbeitung der Task.
Eine Rückkehr zum Scheduler findet mit jedem Aufruf einer Systemroutine in der laufenden Task (mit Ausnahme der Start-Routine) statt. Dabei kommt ein Trick zur Auswirkung: der CALL-Befehl beim Aufruf der Systemroutine hat automatisch die nächste Befehlsadresse der Task auf deren Stack abgelegt. Die Schedulerroutine SWITCH, die bei der Rückkehr immer als erstes aufgerufen wird, „friert" diesen Zustand des Taskstacks ein und führt eine Kontextumschaltung in den Scheduler-Kontext durch (d. h. die gesamte Programmumgebung des Schedulers, die auch den Stack beinhaltet, wird zugänglich gemacht). Dazu holt SWITCH zunächst die überflüssige eigene Rückkehradresse vom Stack in IY. Dann lädt sie – einigermaßen umständlich – den Task-Stackpointerwert in das Registerpaar DE und schaltet das Stackpointer-Register auf den Schedulerstack um. Von diesem wird das Basiszeigerregister zurückgeholt und damit ist der letzte Task-Zustand wieder zugänglich. Dort wird nun der aktuelle Task-Stackpointerwert abgelegt. Über die in IY stehende Rückkehradresse wird danach die Bearbeitung im Systemkontext fortgesetzt.
Das damit erfolgte „Einfrieren" des Taskstacks erlaubt, jede beliebige andere Aktivität zwischendurch auszuführen, ohne den Kontext (die Umgebung) der gerade verlassenen Task zu stören. In unserem Fall werden nun erst einmal alle anderen Tasks geprüft und, wenn aktiv, aufgerufen, bis nach einem vollständigen Umlauf die eben verlassene Task wieder an die Reihe kommt.

Doch weiter der Reihe nach:
Nach der Rückkehr von SWITCH wird evtl. noch der Zustand der letzten Task verändert, z. B. eine Wartezeit eingetragen, dann schaltet NEXTTASK das IX-Register zum nächsten Element der Task-Zustandsliste weiter. Damit wird dann genauso verfahren wie gerade beschrieben.

Der geschilderte Ablauf zeigt auch, warum die einzelnen Tasks „kooperativ" programmiert werden müssen. Da der Scheduler keine Eingriffsmöglichkeit in eine laufende Task hat, kann eine Schleife, in der keine Systemroutine aufgerufen wird, den ganzen Ablauf anhalten. Deshalb sind Warteschleifen nicht erlaubt!

Neustart einer Task

Der bis hier behandelte Teil des Systems erlaubt bereits den quasi-parallelen Ablauf mehrerer Programme durch verschachtelte Abarbeitung. Jedoch gibt es damit keine Möglichkeit, eine angehaltene Task zum Laufen zu bringen.

Dazu ist ein Zugriff auf den Taskzustand einer anderen als der aktuellen Task nötig. Dies leistet die Prozedur START (*Bild 4*). Ein direkter Zugriff auf einen bestimmten Eintrag der Task-Zustandsliste ist normalerweise nicht möglich, weil die Reihenfolge der Einträge und evtl. sogar ihre Lage im Speicher nicht bekannt sind. Wir müssen also mit einem bekannten Eintrag anfangen und die Liste anhand der Zeigerverkettung verfolgen, bis der gesuchte Eintrag gefunden ist. Die Vorgehensweise ist bereits bekannt: sie ist dieselbe wie bei der Stack-Initialisierung. Lediglich der Vergleich der Task-Identifikation muß verfeinert werden: die Identifikation muß unabhängig vom Zustand des höchstwertigen (Flag-)Bits gefunden werden (es ist aber auch die Beschränkung der Suche nur auf inaktive Tasks möglich.

Wenn die gesuchte Task bereits aktiv oder wartend ist, wird sie nicht beeinflußt; die Suchstrategie ist dann entsprechend anzupassen). Zusätzlich ist in der gezeigten Routine noch eine Sicherung eingebaut: wenn die gesuchte Task-Identifikation nicht vorhanden sein sollte, bricht die einfache Suchschleife nicht ab und das System „hängt sich auf". Das wird verhindert, indem geprüft wird, ob die Liste bereits einmal ganz abgesucht worden ist. In diesem Fall wird die Suche abgebrochen und über das Carry-Flag Fehlanzeige zurückgemeldet. Außerdem zeigt das Zero-Flag an, ob die gestartete Task wirklich angehalten war. Mit dieser Routine können wir nun eine inaktive Task „aufwecken", und eine Task kann sich am Ende ihrer Aktivitä-

```
SCHEDU - Multitasking Scheduler fuer kleine Stand-Alone - Anwendungen
Strategie: Round Robin

006A'                 START:  ; Start einer Task, evtl. mit Verzoegerung
                      ; PARAMETER:   Verzoegerung in Akku
                      ;              Taskidentifikation in C-Register
                      ; ERGEBNIS:    Z  gesetzt, wenn Task inaktiv war
                      ;              CY gesetzt, wenn Task nicht gefunden
006A'  FDE5                   PUSH  Y
006C'  D5                     PUSH  D
006D'  C5                     PUSH  B
006E'  F5                     PUSH  PSW                  ; Register retten
006F'  FD21 0000:04           LXI   Y, TASKAREA          ; Scheduler-Kontext verfuegbar machen
0073'  3E7F                   MVI   A, ID.MASK
0075'  FDA604                 ANA   TASK.ID (Y)          ; Taskidentifikation holen
0078'  47                     MOV   B, A                 ; erste merken
0079'                 ..SUCHE:
0079'  B9                     CMP   C                    ; mit gesuchter vergleichen
007A'  2818                   JR    Z, ..DIESE           ; gefunden?
007C'  FD5E06                 MOV   E, NEXT.T (Y)        ; naechstes Taskzustands-
007F'  FD5607                 MOV   D, NEXT.T+ 1 (Y)     ; element ...
0082'  D5                     PUSH  D
0083'  FDE1                   POP   Y                    ; .... ansteuern
0085'  3E7F                   MVI   A, ID.MASK
0087'  FDA604                 ANA   TASK.ID (Y)          ; naechste Taskid holen
008A'  B8                     CMP   B                    ; wieder am Anfang?
008B'  20EC                   JR    NZ, ..SUCHE          ; weitersuchen
008D'  F1                     POP   PSW
008E'  C1                     POP   B
008F'  D1                     POP   D
0090'  FDE1                   POP   Y
0092'  37                     SCF                        ; melde nicht gefunden
0093'  C9                     RET

0094'                 ..DIESE:
0094'  F1                     POP   PSW
0095'  FDCB047E               BIT   AKTIV., TASK.ID (Y)  ; melde (in)aktiv/wartend
0099'  FDCB04FE               SET   AKTIV., TASK.ID (Y)  ; setze aktiv/wartend
0090'  FD7705                 MOV   DELAY. (Y), A        ; setze Verzoegerung
00A0'  C1                     POP   B                    ; hole restliche Register zurueck
00A1'  D1                     POP   D
00A2'  FDE1                   POP   Y
00A4'  C9                     RET
```

Bild 4. Routine zum Start einer inaktiven Task

ten ruhig „schlafenlegen" (anhalten), bis sie wieder gebraucht wird.

Das Aktivieren einer Task kann von beliebiger Stelle aus geschehen. Es muß lediglich die Adresse von START und die Identifikation der zu startenden Task bekannt sein. Der Start kann auch von einer Interruptroutine ausgeführt werden. Die Interruptroutine bleibt dadurch kurz und belastet das System nicht. Die Hauptarbeit – die für eine Interruptroutine zu komplex sein kann – wird dann im normalen Taskmodus ausgeführt. Ein Beispiel für eine solche Task ist die zum System gehörige Wartezeitbearbeitung (TICK, *Bild 5*).

Die Interruptroutine IR.TICK startet lediglich die Task TICK; erst diese dekrementiert die Wartezeitzähler aller Tasks. Die Bearbeitung läuft wieder nach dem Schema der Stackinitialisierung ab. Wie dort wird die gesamte Liste der Taskzustände einmal durchlaufen, und bei jeder Task wird der DELAY-Zähler dekrementiert, wenn er nicht bereits den Wert Null hat. Da das nicht im Interruptmodus geschieht, kann es keinen Konflikt mit dem Setzen oder Löschen des Zählers von anderer Stelle (START) geben. TICK zeigt auch den Aufruf einer Systemroutine. Der Aufruf von ENDE gibt die Bearbeitung wieder an den Scheduler zurück und setzt als Rückkehradresse für den nächsten Taskaufruf die Startadresse von TICK ein. Damit fängt TICK bei jedem Aufruf von vorn an und bearbeitet die gesamte Task-Zustandsliste.

```
SCHEDU - Multitasking Scheduler fuer kleine Stand-Alone - Anwendungen
Strategie: Round Robin - Beispiel-Task DRUCK
0003                $READY  == 3                        ; READY-Bit Drucker-Steuerung

                    .EXTERN DRUCKZEIGER
0000'               DRUCK:
0000'  DB22                 IN      READY
0002'  CB5F                 BIT     $READY, A
0004'  280F                 JR      Z, ..WARTE          ; Drucker nicht bereit?
0006'  2A 0000"             LHLD    DRUCKZEIGER
0009'  7E                   MOV     A, M
000A'  FEFF                 CPI     $END                ; Textende testen
000C'  CC 0000:04           CALL    Z, ENDE             ; fertig?
000F'  23                   INX     H
0010'  22 0000"             SHLD    DRUCKZEIGER         ; Druckzeiger weiterstellen
0013'  D320                 OUT     DRUCKER             ; Zeichen ausgeben
0015'                  ..WARTE:
0015'  AF                   CALL    RESTART             ; eine Runde aussetzen
0016'  CD 0000:05
                    ; Hilfsroutine, die signalisiert, ob Drucktask verfuegbar ist

0019'               BEREIT: ; gibt erst zurueck, wenn Drucktask frei ist
                            ; zerstoert alle Register!
0019'  3EFF                 MVI     A, $END
001B'  2A 0000"             LHLD    DRUCKZEIGER
001E'  BE                   CMP     M
001F'  C8                   RET     Z
0020'  CD 0000:06           CALL    PAUSE               ; eine Runde aussetzen
0023'  18F4                 JR      BEREIT              ; wieder probieren
```

Bild 6. Beispiel aus einem Protokolldrucker

Ein Anwendungsbeispiel

Schließlich soll noch ein Beispiel aus einer Anwendung des vorgestellten Systems gezeigt werden (*Bild 6*, vgl. [2]). Es handelt sich hier um eine Task, die einen von einer anderen Task vorbereiteten Text auf einem Drucker ausgibt. Der Grundaufbau der Task wird zumindest denen nicht fremd sein, die sich schon einmal mit Schnittstellen und deren Programmierung befaßt haben. Hier sind jedoch zwei Stellen besonders zu beachten:

Es gibt keine echte Warteschleife für den Fall, daß der Port noch keine neuen Daten aufnehmen kann, da das System nicht blockiert werden darf. Statt dessen wird die Sytemfunktion RESTART aufgerufen, die zum Scheduler – und damit zu den anderen Tasks – zurückschaltet und außerdem dafür sorgt, daß beim nächsten Aufruf die Task bei ihrer Startadresse wieder aufgenommen wird. Damit kommen bei jeder vergeblichen Portabfrage alle anderen aktiven Tasks zur Bearbeitung.

Dasselbe geschieht, wenn ein Zeichen ausgegeben wurde, da dann der Drucker erst einmal beschäftigt ist bzw. das Zeichen übertragen werden muß.

Wenn jedoch das letzte Zeichen des Textes ausgegeben ist und der Druckzeiger auf eine Endemarke zeigt, beendet sich das Druckprogramm durch den Aufruf von ENDE und wartet inaktiv auf den nächsten Start (wie TICK). So, wie die letzte Abfrage hier programmiert ist, würde in einem Einprogrammsystem Zeit verschenkt. Im vorliegenden Fall ist dies aber bedeutungslos, da ja ein Warten auf den Port die anderen Tasks nicht am Ablaufen hindert. Die gezeigte Form spart dagegen etwas Programmspeicher, da der Vergleich gleich nach dem sowieso nötigen Holen des Ausgabezeichens steht.

```
SCHEDU - Multitasking Scheduler fuer kleine Stand-Alone - Anwendungen
Strategie: Round Robin
                    ; Task-Identifikation:
0054                $TICK   == 'T'

0034'               IR.DELAY:       ; Verzoegerungsbearbeitung (Systemzeit)
0034'  08                   EXAF
0035'  D9                   EXX
0036'  0E54                 MVI     C, $TICK
0038'  AF                   XRA     A
0039'  CD 0000:05           CALL    START
003C'  D9                   EXX
003D'  08                   EXAF
003E'  FB                   EI
003F'  ED4D                 RETI
SCHEDU - Multitasking Scheduler fuer kleine Stand-Alone - Anwendungen
Strategie: Round Robin

                    .EXTERN TASKAREA                    ; RAM-Bereich der Taskzustandsliste

0000'               TICK:   ; dekrementiert Verzoegerungszaehler <> 0 aller Tasks
0000'  FD21 0000:04         LXI     Y, TASKAREA         ; Scheduler-Kontext verfuegbar machen
0004'  3E7F                 MVI     A, ID.MASK
0006'  FDA604               ANA     TASK.ID (Y)         ; Taskid holen
0009'  47                   MOV     B, A                ; erste merken
000A'               ..SCAN:
000A'  FD7E05               MOV     A, DELAY. (Y)       ; Verzoegerung holen
000D'  B7                   ORA     A
000E'  2803                 JR      Z, ..NULL           ; = 0 ?
0010'  FD3605               DCR     DELAY. (Y)          ; dekrementieren
0013'               ..NULL:
0013'  FD5E06               MOV     E, NEXT.T (Y)       ; naechstes Taskzustands-
0016'  FD5607               MOV     D, NEXT.T+ 1 (Y)    ; element ...
0019'  D5                   PUSH    D
001A'  FDE1                 POP     Y                   ; ... ansteuern
001C'  3E7F                 MVI     A, ID.MASK
001E'  FDA604               ANA     TASK.ID (Y)         ; naechste Taskid holen
0021'  B8                   CMP     B                   ; einmal rum?
0022'  CC 0000:05           CALL    Z, ENDE             ; dann fertig
0025'  18E3                 JR      ..SCAN              ; sonst weiterarbeiten
```

Bild 5. Wartzeit-Bearbeitung mit der Task TICK

In Bild 6 ist zusätzlich zur eigentlichen Drucktask noch eine Dienstprozedur (BEREIT) angegeben, die anderen Tasks zur Verfügung steht. Durch einen Aufruf von BEREIT kann eine Task ihren Ablauf mit der Druckausgabe synchronisieren. Ein solcher „Export" von Funktionen fördert ebenfalls die Übersichtlichkeit eines Programms, da alle Funktionen, die ein Modul betreffen, in diesem selbst ausgeführt werden und ein „fremdes" Modul keine Kenntnis über deren Aufbau braucht. Die hier gezeigte Routine BEREIT entspricht in ihrem Aufbau völlig einer einfachen Warteschleife in einem Einprogrammsystem. Da aber einfache Warteschleifen in unserem Multitaskingsystem verboten sind – sie sind nicht kooperativ – muß innerhalb der Schleife mindestens einmal eine Systemfunktion aufgerufen werden, die den Ablauf anderer Tasks erlaubt. Im Beispiel leistet dies die Prozedur PAUSE. Diese Routine ist, im Unterschied zu RESTART und ENDE, genauso zu verwenden wie eine beliebige andere Subroutine. D. h. die Bearbeitung der Task geht wie bei einer solchen mit dem dem Aufruf folgenden Befehl weiter, ohne daß die Programmumgebung verändert ist. Die Wirkung der Systemroutinen ist im Bild 4 bei den einzelnen Routinen selbst angegeben.

Für viele Zwecke ausreichend

Mit dem vorgestellten „Micro-Multitasking-Scheduler" können bereits mit kleinsten Konfigurationen die Programmierverfahren der Prozeßrechentechnik verwendet werden. Es fehlen natürlich noch viele Möglichkeiten der großen Systeme. So ist hier keine Unterstützung von Ein-/Ausgabevorgängen auf Systemebene vorhanden, und auch der Datenaustausch zwischen den Tasks wird nicht unterstützt. Aber bei Computern der Ausbaustufe des Z80-EMUF sind die Programme wohl noch einfach genug, daß zur Organisation dieser Vorgänge keine Klimmzüge nötig sind. Außerdem kann man den hier vorgestellten Kern ja noch um die zusätzlich gebrauchten Funktionen erweitern.

Mit dem vorhandenen Systemkern wurde ein Protokolldrucker ausgeführt, der interruptgesteuert ein Dutzend Eingänge überwacht, ihre Zustände – teils in festen Zeitintervallen – zusammen mit einem Meßwert protokolliert und das Protokoll mit Datum, Uhrzeit und diversen anderen Zusatzinformationen versieht. Datum und Uhrzeit werden ebenfalls interruptgesteuert intern geführt und können mit einer eingebauten Tastatur überprüft und eingestellt werden. Insgesamt enthält das Betriebsprogramm sechs Tasks und ist in knapp als 2 KByte Programmspeicher untergebracht. Der Systemkern benötigt davon etwa 300 Byte.

Die Listings wurden mit einem Z80-Assembler erstellt, dessen Befehlssyntax weitgehend der von Intel für den 8080 eingeführten entlehnt ist. Lediglich die Erweiterungen des Z80-Befehlssatzes ohne 8080-Entsprechungen sind der Zilog-Spezifikation entnommen.

Der Assembler erstellt relokatiblen (verschiebbaren) und linkfähigen (bindefähigen) Code. Er ermöglicht damit ein einfaches Aufspalten des Quellenprogramms in Module durch Angabe von internen Symbolen (die in anderen Modulen verwendbar sein sollen) und von externen Symbolen, die in anderen Modulen definiert sind.

Literatur

z1] *Kanis, W.*: Der Z80-EMUF. mc 1983, Heft 4, Seite 112.

[2] *Gaulke, E.*: Z80-EMUF als Spooler. mc 1983, Heft 10, Seite 98.

Mehr Platz auf Pascal-Disketten

In Heft 4/84 stellten wir eine Möglichkeit vor, wie man das Apple-DOS 3.3 an den Betrieb mit 40-Spur-Floppy-Laufwerken anpaßt. Hier folgt nun die entsprechende Modifikation des UCSD-Pascal-Systems. Damit wird es z. B. möglich, eine bootfähige Disk „Apple 0" zu erstellen, auf der dann – wenn sich „SYSTEM.APPLE" auf der Diskette befindet – immer noch acht Blocks mehr frei bleiben als auf einer 35-Spur-Disk-„Apple 0".

Im Unterschied zum DOS 3.3, wo sich ja auf jeder Diskette das komplette Betriebssystem befindet, sind auf einer Pascal-Diskette nur die ersten sechs Blöcke (zwölf Sektoren) fest vom System belegt. Hier befindet sich das Directory, in dem unter anderem auch steht, wieviele Blöcke auf der Diskette insgesamt verfügbar sind. Dieser Information bedienen sich zur Verwaltung der Files alle Teile des Systems, die auf die Diskette schreiben, z. B. der Filer oder der Editor. Auf jeder Diskette steht die Blockanzahl in den Bytes $0E, 0F des Sektors $0B auf der Spur $00. Bei einer 35-Spur-Diskette sind es 280 Blöcke. In $0E steht also $18, und in $0F steht $01. Das Directory wird beim Formatieren einer Pascal-Diskette angelegt. Will man also mit 40 Spuren (320 Blöcke) arbeiten, so muß der Formatierer dahingehend geändert werden, daß er zum einen auf der Diskette 40 Spuren anlegt und zum anderen die 320 Blöcke auch im Directory vermerkt. Diese Änderungen übernimmt das Programm im *Bild*, das als Unterprogramm die USERWTS-Routine aus mc 6/83, S. 53, verwendet.

Zur Anwendung des Programms muß man zunächst den Formatierer auf eine formatierte, ansonsten aber leere (!) Diskette kopieren, so daß FORMATTER-.CODE das erste und FORMATTER.DATA das zweite File auf der Diskette ist, wobei keine unbenutzten Blöcke vor oder zwischen den beiden Files liegen dürfen. Nachdem man das Programm eingegeben hat, kann man es mit 9500G vom Monitor aus starten, wenn man die oben beschriebene Pascal-Diskette eingelegt hat. Das Programm ändert genau zwei Bytes des Formatierers: Eines sorgt dafür, daß Disketten von nun an mit 40 Spuren formatiert werden; das andere bewirkt, daß im Directory dieser Disketten die Anzahl der Blöcke auf 320 gesetzt wird.

Thomas Schilz

```
9500- A9 01 85 19 A9 01 85 1A
9508- 85 1B 85 1C 20 34 95 A9
9510- 40 8D 0E 10 A9 02 85 1C
9518- 20 34 95 A9 0B 85 1B A9
9520- 01 85 1C 20 34 95 A9 28
9528- 8D 9C 10 A9 02 85 1C 20
9530- 34 95 60 00 20 E3 03 84
9538- 06 85 07 A0 01 A9 60 91
9540- 06 C8 A5 19 91 06 C8 A9
9548- 00 91 06 C8 A5 1A 91 06
9550- C8 A5 1B 91 06 A0 08 A9
9558- 00 91 06 C8 A9 10 91 06
9560- A0 0C A5 1C 91 06 A0 0E
9568- A9 00 91 06 C8 A9 60 91
9570- 06 A4 06 A5 07 20 D9 03
9578- A9 00 85 4B A0 0D B1 06
9580- 85 1D 60
```

Das Programm erzeugt einen 40-Spur-Formatierer für das UCSD-Pascal-System des Apple-II