AN018

Application Note 018

Echtzeit-Programme in C

Autor: HK AN018.DOC (13 Seiten)

1. Aufgabenstellung

Die Erstellung von Multi-Tasking-Programmen für die MODULAR-4 soll mit Hilfe eines C-Cross-Compilers und eines Cross-Assemblers erfolgen. Dabei soll der prinzipielle Aufbau von C-Programmen für das Multi-Tasking-Betriebssystem erläutert werden. Beschrieben werden auch die von SORCUS zur Verfügung gestellten C-Routinen und #include-Dateien.

2. C-Routinen, #define Anweisungen und #include-Dateien

Nachfolgend werden die Möglichkeiten beschrieben, die der Anwender hat, um die vom Multi-Tasking-Betriebssystem der MODULAR-4 Karte zur Verfügung gestellten Sub-Routinen aufzurufen.

2.1. C-Routinen und #define Anweisungen

Es werden zu dieser Application Note zwei Dateien auf Diskette mitgeliefert, und zwar die Datei **m4p-usr.c** und die Datei **m4p-sub.c.**
Die Datei **m4p-usr.c** enthält die #define-Anweisungen, die die Adressen der Sub-Routinen des Betriebssystems zur Verfügung stellen. Die Datei muß am Anfang eines Echtzeitprogramms mit #include eingebunden werden, damit dem Programm die Adressen der Sub-Routinen bekannt sind.

Die Datei **m4p-sub.c** enthält Funktionen in C, die einen Aufruf dieser Sub-Routinen ermöglichen. Außerdem sind als Beispiel weitere Modul-spezifische Prozeduren enthalten. Diese Datei muß am Ende eines Multi-Tasking-Programms mit #include eingebunden werden. Ein Einbinden am Anfang des Programms würde die vorgeschriebene Struktur eines Echtzeit-Programms zerstören!

2.1.1. #define Anweisungen für die Sub-Routinen des Betriebssystems

Die Datei **m4p-usr.c** enthält die #define-Anweisungen in C für die aufrufbaren Sub-Routinen des Betriebssystems. Die Beschreibung zu diesen Sub-Routinen finden sie im Technischen Handbuch. Die bei der Beschreibung angegebenen Register werden der Funktion **subroutine(adr,regs)** übergeben, die dann die Sub-Routine des Betriebssystems aufruft (s.u.).

2.1.2. C-Routinen

Die C-Routinen für den Aufruf der Sub-Routinen des Betriebssystem und weitere modulspezifische Routinen sind in der Datei **m4p-sub.c** enthalten.

2.1.2.1. C-Routine für den Aufruf einer Betriebssystem-Routine

Die Funktion **subroutine** ruft eine Sub-Routine des Betriebssystems auf, deren Adresse in **adr** und deren Parameter in **regs** übergeben werden.

```
subroutine(adr,regs)
int adr;
union _int_byte *regs;

wobei
        union _int_byte
        {
        struct _register_int
        {
                int af;             /* Z80-Register AF */
                int bc;             /* Z80-Register BC */
                int de;             /* Z80-Register DE */
                int hl;             /* Z80-Register HL */
                int ix;             /* Z80-Register IX */
                int iy;             /* Z80-Register IY */
        }i;
    struct _register_byte
        {
                char f;             /* Z80-Register AF */
                char a;
                char c;             /* Z80-Register BC */
                char b;
                char e;             /* Z80-Register DE */
                char d;
                char l;             /* Z80-Register HL */
                char h;
        }b;
    } regs;
```

2.1.2.2. Aufruf eines System-Call

Die Funktion **syscall** führt einen System-Call aus, wobei die Nr. des System-Call in **snr** und die Funktions-Nr. in **fnr** übergeben werden. Die Beschreibung aller z.Zt. zur Verfügung stehenden System-Calls befindet sich im Technischen Handbuch.

```
syscall(snr,fnr)
char snr;
unsigned int fnr;
```

2.1.2.3. Lies Task-Nr.

Diese Funktion liefert die Task-Nr. der Task, unter der das C-Programm installiert worden ist. Sie muß als erste Anweisung in einer Task-Prozedur (z.B. Auto-Init-Prozedur) stehen.

```
gettask()
```

2.1.2.4. Lies Adresse des Parameterbereichs

Diese Funktion liefert die Adresse des Parameterbereichs der in **Task_Nr** angegebenen Task.

```
getpar_adr(Task_Nr)
```

2.1.2.5. Lies Adresse des Datenbereichs

Diese Funktion liefert die Adresse des Datenbereichs der in **Task_Nr** angegebenen Task.

```
getdat_adr(Task_Nr)
```

2.1.2.6. Setze Multiplexer und starte Wandlung (Modul M-AD16-3)

Diese Funktion setzt den Multiplexer auf dem SPB-Modul M-AD16-3, das sich auf dem Steckplatz befindet, der über **Modul_Adr** adressiert wird, auf den Kanal, der in **kanal** angegeben wird und startet die A/D-Wandlung. Berücksichtigt wird dabei die Settle-Time in **Delay** (Zählerwert).

```
setmuxst(kanal,Modul_Adr,Delay)
char kanal,Modul_Adr,Delay;
```

2.1.2.7. Ergebnis der A/D-Wandlung abholen, nächste Wandlung starten und Multiplexer neu setzen (Modul M-AD16-3)

Diese Funktion holt das Ergebnis der letzten Wandlung von dem SPB-Modul M-AD16-3 ab, das sich auf dem Steckplatz befindet, der über **Modul_Adr** adressiert wird, startet die nächste Wandlung und setzt den Multiplexer auf den in **kanal** angegebenen Kanal.

getwandels(Modul_Adr,kanal)
char Modul_Adr,kanal;

2.2. #include-Dateien erzeugen

Diese Dateien werden für das Freigeben (EI) und Sperren (DI) des CPU-Interrupts benötigt.

di.c Es wird der CPU-Interrupt gesperrt.

ei.c Es wird der CPU-Interrupt freigegeben.

Wie Sie eigene #include-Dateien mit Assemblercode erzeugen können, wird an folgendem Beispiel kurz erläutert:

```
/* Aufruf eines Systemcall */

syscall(snr,fnr)          /* Definition der C-Funktion */
char snr;                 /* enthält Systemcall */
unsigned int fnr;         /* enthält Funktion, hier Task-Nr.*/
{
#ASM                      ; Einbinden von Assembler-Code
      POP    HL           ; HL enthält die Rückkehr-Adresse
      POP    BC           ; C enthält die Nr. des System-Call
      POP    DE           ; E enthält die Task_Nr.
      PUSH   DE           ; Stack wieder restaurieren
      PUSH   BC           ;
      PUSH   HL           ;
      LD     A,C          ; Nr. des System-Call nach A laden
      RST    08H          ; System-Call ausführen (Restart 8)
      EX     DE,HL        ;Resultat von DE nach HL laden
                          ; (Funktionswerte werden immer
                          ; in HL übergeben)
#ENDASM                   ; Ende des Assembler-Codes
}                         /* Ende der C-Funktion */
```

3. Aufbau von Anwendungsprogrammen in C

Multi-Tasking-Programme, die in C geschrieben werden, können nur durch Einbindung von Assembler-Routinen Einfluß auf die Wahl des Registersatzes nehmen. Ein in C geschriebenes Programm (ohne Einbindung von Bibliotheken), das mit dem Cross-C-Compiler (MIC) zu einem Assembler-Programm übersetzt wird, verwendet nur den aktiven Registersatz (AF, BC, DE, HL). Die immer aktiven Register IX, IY, I und R werden von dem erwähnten C-Cross-Compiler nicht verwendet, ebenso der nicht aktive Registersatz AF', BC', DE' und HL'.

Werden die Objekt-Bibliotheken CLIBZ und CZLIB verwendet, wird Z80-Code eingebunden, sonst 8080-Code.

Die von SORCUS gelieferten Dateien enthalten immer Z80-Code.

3.1. Programme für NI-Tasks (Nicht-Interrupt-Tasks)

Bei Aufruf der Auto-Init-Prozedur, der Haupt-Prozedur und der allgemeinen Prozeduren kann über folgende Prozeduren auf Daten und Parameter zugegriffen werden:

gettask() liefert die Task-Nr.,
getpar_adr(Task_Nr) liefert die Adresse des Parameterbereichs der Task,
getdat_adr(Task_Nr) liefert die Adresse des Datenbereichs der Task.

Die Adressen von Parameter- und Datenbereich müssen Pointern zugewiesen werden, die auf eine Struktur oder ein Feld zeigen (z.B. int *daten, Pointer auf den Datenbereich). Soll der CPU-Interrupt gesperrt bzw. freigegeben werden, können die #include-Dateien **di.c** und **ei.c** verwendet werden. Retten von Registern oder Umschalten der Registersätze ist nicht erforderlich.

3.2. Programme für II-Tasks (Indirekte Interrupt-Tasks)

Bei den Tasks 16 bis 31 (II-Tasks) gilt das gleiche wie für die NI-Tasks. Zusätzlich sorgt das Betriebssystem beim Eintritt in die Interrupt-Service-Routine (Haupt-Prozedur) dafür, daß der Registersatz umgeschaltet wird. Der Anwender kann also eine Standard-C-Funktion schreiben, die dann bei Auftreten eines Interrupts ausgeführt wird. Der CPU-Interrupt darf, außer in Ausnahmefällen, nicht freigegeben werden. Dies geschieht automatisch nach Verlassen der C-Funktion durch das Betriebssystem. Außerdem wird abschließend vom Betriebssystem der Registersatz wieder umgeschaltet und ein Return from Interrupt (RETI) durchgeführt.

4. Prinzipieller Aufbau einer Programm-Datei (NI- und II-Task)

```
/*  Programm-Datei (für NI- und II-Tasks)  */

struct _kopftabelle
kopf = { 0x54,          /* Programm-Nr. */
         p_anzahl,      /* Anzahl der von diesem Programm benötigten Parameter (Anzahl Byte) */
         &main,         /* AdressederHaupt-Prozedur */
         &_auto_init,   /* Adresse einer Auto-Initialisierungs-Prozedur, die beim Installieren des
                           Programms aufgerufen wird */
         &_proc_6,      /* Adresse der Prozedur 6 (falls benötigt) */
         &_proc_8,      /* Adresse der Prozedur 8 (falls benötigt) */

         .....          /* weitere Prozeduren, max. 122 */
         &_proc_n,      /* Adresse der letzten Prozedur */
         0xffff,        /* Endezeichen */
         vers,rel,      /* Angabe der Version ("1" bis "9") und Release ("A" bis "Z") des
                           Programms */
         0,             /* Länge der Datei (wird von SNW ermittelt und eingetragen, ab Vers. 1J,
                           12.07.1990*/
         &kopf.pgm_nr/* Anfangsadresse: wird von SNW beim Laden und Installieren für
                        Kontrollzwecke verwendet, wenn vers > 1 ist. */
      };

_auto_init()            /* Auto-Initialisierungs-Prozedur */

{
 ...
 ...
}

_proc_6()               /* Prozedur 6 */

{
 ...
 ...
}

_proc_8()               /* Prozedur 8 */
{
 ...
 ...
}


main()                  /* Haupt-Prozedur */
{
 ...
 ...
}           /* wird mit RET beendet */
```

4.1. Listing Programm 84, Beispiel für II-Task (Datei M4P54C.C)

```
/* Letzte Bearbeitung: 20.07.1990          13:00
   Listing des Programms 84, realisiert in "C". Eine Beschreibung des Programms finden Sie im
   Benutzer-Handbuch zur MODULAR-4/Z80. Das Assembler-Listing steht im Technischen
   Handbuch */

#include m4p-usr.c      /* Einbinden von "C" - #define- Anweisungen */

extern main();          /* Vorab-Definition */
extern _auto_init();    /* verwendeter Prozeduren */
extern _proc_6();
extern _proc_8();

/* weitere Prozedurdefinitionen möglich, falls im Anwenderprogramm verwendet */

#define p_anzahl 22

/* Definition der Struktur des Parameterbereichs des Programms 84 (ueber diese Struktur werden
die Parameter zur Laufzeit angesprochen) */

static struct _parameter
{
char status;            /* Statusbyte */
char TimerB_Data;       /* Timer-B Data */
char TimerB_Code;       /*       Code fuer Vorteiler */
char Soft_Teiler_1;     /* Timer-B Teiler */
char Soft_Teiler_2;     /* Timer_B Teiler */
char Trigger;           /* Start-Trigger: 0 bis 7: INT-0 bis INT-7, 255: PC */
char Flanke;            /*Flanke für Trigger (INT-0 bis INT-7) */
char KAnzahl;           /* Anzahl der Analogkanaele */
char Erster_Kanal;      /* Erster zu messender Kanal */
int  MAnzahl;           /* Anzahl der Meß-Werte */
char settle;            /* Settle-Time fuer 1. Wandlung */
};
/* Definition des Parameter-Pointers */

struct _parameter *para_point;

/* Aufbau der Kopftabelle für das Programm 84 s.a. Technisches Handbuch */
struct _kopftabelle
{
char pgm_nr;            /* Programm-Nr. */
char p_anz;             /* Anzahl Parameter */
int  *main;             /* Adresse Hauptprogramm */
int  *auto_init;        /* Adresse Auto-Init-Prozedur */
int  *proc_6;           /* Adresse Proezdur 6 */
int  *proc_8;           /* Adresse Proezdur 8 */
int  trenn;             /* Trennzeichen */
char version;           /* Version */
char revision;          /* Revision */
```

```
int    *laenge;          /* Programmlaenge */
int    *kopf;            /* Anfangsadresse */
} kopf = {0x54,p_anzahl,&main,&_auto_init,&_proc_6,&_proc_8,0xffff,'3','A',0,&kopf.pgm_nr};

/* Definition der Struktur des Parameterbereichs des Programms 84 (ueber diese Struktur werden
die Parameter vorinitialisiert innerhalb des Programm-Codes des Programms 84) */

struct _cparameter
{
char p_anz;              /* Anzahl der Parameter */
struct _parameter par;   /* Vordefinierte Parameter */
} pardef = {p_anzahl,0,50,4,1,1,255,0,1,48,16,1};

/* Lokale Variable */

int *paktuell = {0};     /* Pointer auf aktuellen Wert */
int Modul_Adr = {0};     /* Modul-Basisadresse */
int plast = {0};         /* Pointer auf letzten Wert + 1 */
int first = {0};         /* Erster Kanal */
int count = {0};         /* Laufvariable in der ISR */
int lkanal = {0};
int Delay = {0};         /* Settle-Time */
int Task_Nr = {0};       /* Task-Nr. */
int Anzahl = {0};        /* Anzahl Kanäle */

_auto_init()      /* Auto-Initialisierungs-Prozedur */
{
  Task_Nr = gettask();
  para_point = getpar_adr(Task_Nr);    /* Anfangs-Adresse Parameterbereich zuweisen */
  regs.i.hl = pardef;                  /* Initialisiere Parameter */
  regs.i.iy = para_point;
  subroutine(PRINIP,regs);
  subroutine(TBSTOP,regs);             /* Stop Timer-B */
  regs.i.de = 0x100;                   /* Mask Timer-B */
  subroutine(MASK,regs);
}

_proc_6()         /* Prozedur 6 */
{
  paktuell = getdat_adr(Task_Nr);      /* Anfangs-Adresse Datenbereich zuweisen */
  /* End-Adresse Datenbereich ermitteln und zuweisen */
  plast = paktuell + (para_point->MAnzahl*para_point->KAnzahl)-1;

  first = para_point->Erster_Kanal;    /* 1. Kanal zuweisen */
  Anzahl = para_point->KAnzahl;        /* Anzahl Kanäle zuweisen */
  /* Modul-Basis-Adresse errechnen */
  Modul_Adr = ((first << 1) & 0x60) | 0x80;
  Delay = para_point->settle;          /* Settle-Time zuweisen */
  para_point->status = 1;              /* Status = 1 setzen */

  if (para_point->Trigger == 255) {    /* PC als Trigger: Timer-B initialisieren und starten */
```

```
    regs.b.e = para_point->TimerB_Data;
    regs.b.d = para_point->TimerB_Code;
    subroutine(TBSET,regs);          /* Timer-B setzen */
    regs.i.de= 0x100;
    subroutine(CLEAR,regs);          /* Clear Timer-B */
    subroutine(UNMASK,regs);         /* Unmask Timer-B */
    para_point->status = 2;          /* Status = 2 setzen */
    }
  else {
    regs.b.e = para_point->Flanke;   /* Aktive Flanke setzen */
    regs.b.d = para_point->Trigger;
    subroutine(EDGE,regs);
    regs.b.a = para_point->Trigger;  /* Maske erzeugen */
    subroutine(TRIMASKE,regs);
    subroutine(CLEAR,regs);          /* Clear Interreupt */
    subroutine(UNMASK,regs);         /* Unmask Interrupt */
    }
}

_proc_8()        /* Prozedur 8 */
{
  subroutine(TBSTOP,regs);           /* Stop Timer-B */
  regs.b.a = para_point->Trigger;
  subroutine(TRIMASKE,regs);         /* Maske erzeugen */
  regs.b.d += 1;
  subroutine(MASK,regs);             /* Maskiere Trigger und Timer-B */
}

main()           /* Haupt-Prozedur (Interrupt-Service-Routine) */
{
  lkanal = first;                    /* Laufvariable auf 1. Kanal setzen */
  /* Multiplexer auf 1. Kanal setzen, Wandlung starten und Multiplexer auf nächsten Kanal setzen */
  setmuxst(lkanal++,Modul_Adr+1,Delay);
  for (count = 2;count <= Anzahl;count++) {
    /* Gewandelten Wert speichern, nächste Wandlung starten und Multiplexer neu setzen */
    paktuell++[0] = getwandels(Modul_Adr+2,++lkanal);
  }
  if (paktuell == plast) {           /* Check, ob Ende der Daten */
    regs.i.de = 0x100;               /* Mask Timer-B */
    subroutine(MASK,regs);
    subroutine(TBSTOP,regs);         /* Stop Timer-B */
    para_point->status = 3;          /* Status = 3 setzen */
    }
  /* Gewandelten Wert speichern */
  paktuell++[0] = getwandels(Modul_Adr+2,first);
}

#include m4p-sub.c
```

5. Erzeugen von M4Pxx.LAB Dateien

Nach Erstellen eines Multi-Tasking-fähigen C-Programms (z.B. M4P54C.C) wird dieses Programm mit Hilfe des C-Cross-Compilers CCZ.EXE zu einer Assembler-Source übersetzt:

1. Aufruf: CCZ /JJJ1SCV M4P54C

wobei die Optionen bedeuten:

JJJ1 alle initialisierten statischen Daten werden ins Codesegment abgelegt. Dadurch wird die Erstellung von Programmen, die in ein ROM gebracht werden sollen, vereinfacht. Dadurch wird ermöglicht, daß die Kopftabelle am Anfang der Datei steht.

S Der Wert eines char liegt bei dieser Option zwischen 0 und 255 und nicht wie sonst zwischen -128 und 127.

C Das C-Quellprogramm wird als Kommentar mit in die Assemblerdatei ausgegeben. So kann später verfolgt werden, welcher Assemblercode aus den jeweiligen C-Quellzeilen erzeugt wurde.

V Der Compiler unterscheidet 16 statt 8 Zeichen bei Namen.

Für andere Anwendungen können andere Optionen erforderlich sein.
Die Option C ist optional.

Die erzeugte Datei heißt M4P54C.ASM. Aus dieser Datei wird dann mit Hilfe des Cross-Assemblers eine Objekt-Datei erzeugt:

2. Aufruf: CA /SV M4P54C

Die erzeugte Datei heißt M4P54C.OBJ. Aus dieser Datei wird dann mit Hilfe von SNW eine .LAB Datei erzeugt. Damit zusätzliche C-Bibliotheksfunktionen mit dazugelinkt werden können, benötigt SNW die Datei MLIBC.LIB.

Diese Datei muß mit dem Bibliotheksverwalter CV.EXE (siehe Beschreibung (MI-C, Seite AF-1) erzeugt werden. Damit können alle für dieses Anwendungsprogramm benötigten Routinen in eine Bibliothek übertragen werden. Diese Bibliothek muß für ein Anwendungsprogramm nur einmal erzeugt werden.

3. Aufruf: SNW /L:M4P54C

Die Datei M4P54C.INS eine Parameterdatei ist, die vor dem erstmaligen Erzeugen einer .LAB Datei mit SNW erzeugt werden muß.

4. Aufruf: SNW /I:M4P54C

Die Datei M4P54C.INS eine Parameterdatei ist, die alle Informationen enthält, die für das Laden und Installieren erforderlich sind. Sie wurde vorher mit SNW erzeugt.

5. Beispiel für eine Parameterdatei M4P54C.INS

```
LINKTIME 23.07.93 16:54:02
M4DEVICE 0380 TIMEOUT=10 RESET
USES MLIBC.LIB
M4P54C 54 3A FCAF FFFE 17 1000 17
```