MODULAR-4/486 Basiskarte

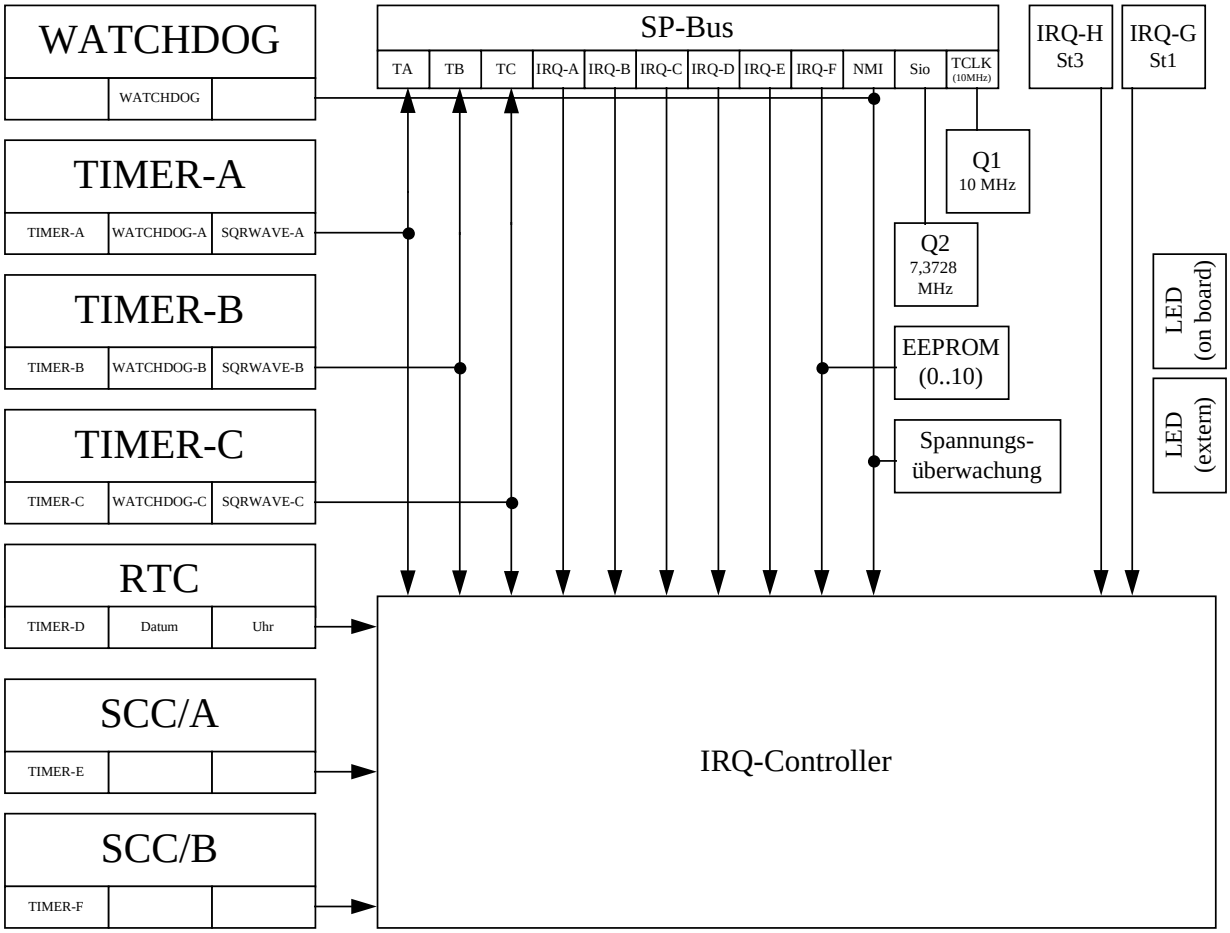
Installationsparameter

Parameter	Wert
Dateiname:	ML8D0000.LIB
Programmnummer:	4FFh
Tasknummer:	1 bis 20
Interruptnummer:	90h
Länge des Datenbereichs:	0
Flags:	802h

Kanaleigenschaftsstruktur CPS_ML8

Offset	Strukturelement	Datentyp	Bedeutung
0	*.usDevice*	USHORT	Device-Typ
2	*.usIndexFirst*	USHORT	Index des ersten Device
4	*.usIndexLast*	USHORT	Index des letzten Device
6	*.usFlags*	USHORT	Flags für zusätzliche Kanaleigenschaften
8	*.usReadMode*	USHORT	Lesemodus
10	*.usWriteMode*	USHORT	Schreibmodus
12	*.ulTimeout*	DWORD	Timeoutzeit für Watchdog
16	*.usMode*	USHORT	Betriebsmodus

Device Ressourcen im Überblick



Die Devices sind zum Teil nicht unabhängig und verwenden gemeinsame Ressourcen. Wenn z.B. ein Kanal zu Timer-E geöffnet wurde (zur Generierung von Interrupts), steht die Kommunikationsfunktion des SCC/A nicht mehr zur Verfügung, da Timer-E den Baudratengenerator zur Intervallgenerierung benötigt.

Leuchtdioden (LED)

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_LED* = 0402h
2	*.usIndexFirst*	Device-Index: *ML8_EXTERNAL_LED* = 0, *ML8_LOCAL_LED*= 1
4	*.usIndexLast*	= *.usIndexFirst*
6	*.usFlags*	Mögliche Flags: Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv) Alle anderen Bits = 0.

Alle anderen Strukturelemente werden nicht ausgewertet.

Setzen einer Leuchtdiode (0 = aus, 1 = ein):

mdd8_write_channel_byte(handle, bLED)

Aktuelle Zustand einer Leuchtdiode lesen:

bLED = mdd8_read_channel_byte(handle)

Uhr

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_CLOCK* = 0701h
2	*.usIndexFirst*	= 0
4	*.usIndexLast*	= 0
6	*.usFlags*	Mögliche Flags: Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv) Alle anderen Bits = 0.
16	*.usMode*	Reserviert, = 0 setzen

Alle anderen Strukturelemente werden nicht ausgewertet.

Uhrzeit lesen:

ulTime = mdd8_read_channel_dword(handle);

Uhrzeit setzen:

mdd8_write_channel_dword(handle, ulTime);

Format von ulTime (32 Bit):

31 24	23 16	15 8	7 0
Stunden	Minuten	Sekunden	Sekunden/100

Kalender (Datum)

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_CALENDAR* = 0702h
2	*.usIndexFirst*	= 0
4	*.usIndexLast*	= 0
6	*.usFlags*	Mögliche Flags: Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv) Alle anderen Bits = 0.
16	*.usMode*	Reserviert, = 0 setzen

Alle anderen Strukturelemente werden nicht ausgewertet.

Datum lesen:

dDate = mdd8_read_channel_dword(handle);

Datum setzen:

mdd8_write_channel_dword(handle, dDate);

Format von dDate (32 Bit):

31 24	23 16	15 8	7 0
Jahr (0..99)	Monat (1..12)	Tag (1..31)	Wochentag (0=Sonntag, 1=Montag, ...)

Timer

Die MODULAR-4/486 Basiskarte verfügt über sechs Timer, die z.B. zur Generierung von Abtastraten verwendet werden können. Der Timer löst zyklisch Interrupts in einem programmierbaren Zeitraster aus.

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_TIMER* = 0501h
2	*.usIndexFirst*	Device-Index:
		ML8_TIMER_A (16 Bit) = 0 *ML8_TIMER_B (16 Bit)* = 1 *ML8_TIMER_C (16 Bit)* = 2 *ML8_TIMER_D* = 3 *ML8_TIMER_E (16 Bit)* = 4 *ML8_TIMER_F (16 Bit)* = 5
4	*.usIndexLast*	= *.usIndexFirst*
6	*.usFlags*	Mögliche Flags:
		Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv)
		Alle anderen Bits = 0.
16	*.usMode*	Reserviert, = 0 setzen

Alle anderen Strukturelemente werden nicht ausgewertet.

Das Intervall zwischen zwei Interrupts wird beim Setzen des Timers mit einem Schreibzugriff in µs angegeben. Der aktuelle Timerstand (in µs) kann mit einem Lesezugriff abgefragt werden:

Setzen: **mdd8_write_channel_dword(handle, ulInterval);**

Lesen: **ulTimer = mdd8_read_channel_dword(handle);**

Gestartet bzw. angehalten wird der Timer mit:

mdd8_send_channel_command(handle, CMD_START);

bzw.

mdd8_send_channel_command(handle, CMD_STOP);

Nach dem Öffnen eines Timer-Kanals befindet sich dieser im STOP-Zustand.

Die Timer-D, -E und -F können nur zur Interruptgenerierung verwendet werden. Das Lesen des aktuellen Timerwertes ist nicht möglich. Timer-E und -F können nur verwendet werden, wenn die seriellen Schnittstellen der MODULAR-4/486 nicht verwendet werden. Timer-D unterstützt nur folgende Frequenzen: 64 Hz (15,625 ms), 1 Hz (1 Sekunde), 0,0166 Hz (1 Minute) und 0,000277 Hz (1 Stunde).

Rechteckgenerator

Mit diesem Device ist es möglich, Rechtecksignale zu generieren. Das Puls- Pausen-Verhältnis kann programmiert werden. Das Rechtecksignal kann z.B. mit dem SPB-Modul M-D40-2 ausgegeben werden.

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_SQRWAVE* = 0901h
2	*.usIndexFirst*	Device-Index:
		ML8_SQRWAVE_A (Timer-A) = 0 *ML8_SQRWAVE_B (Timer-B)* = 1 *ML8_SQRWAVE_C (Timer-C)* = 2
4	*.usIndexLast*	= *.usIndexFirst*
6	*.usFlags*	Mögliche Flags:
		Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv)
		Alle anderen Bits = 0.
16	*.usMode*	Reserviert, = 0 setzen

Alle anderen Strukturelemente werden nicht ausgewertet.

Ausgabefrequenz (Periodendauer in µs) setzen:

mdd8_write_channel_dword(handle, ulInterval);

Aktiviert und deaktiviert wird der Kanal mit dem Sonderdienst **mdd8_send_channel_command(handle, CMD_START);**

bzw.

mdd8_send_channel_command(handle, CMD_STOP);

Nach dem Öffnen des Kanals befindet sich dieser im STOP-Zustand.

Zur Erzeugung des Rechtecksignals werden die Devices Timer-A, -B und -C der MODULAR-4/486 verwendet. Sind bereits Timer-Kanäle geöffnet, ist das Device nicht mehr als Rechteckgenerator einsetzbar. Das Puls/Pausenverhältnis beträgt bei MODULAR-4/486 immer 1:1.

!

Watchdog

Ein Watchdog kann zur Überwachung der Software auf der MODULAR-4/486 verwendet werden. Ist der Watchdog aktiviert, muß er von der Software regelmäßig (vor Ablauf einer Timeout-Zeit) nachgetriggert werden, andernfalls erfolgt ein Interrupt. Unter dem Interrupt muß eine Interrupt Task installiert werden, die auf das Überschreiten der Timeoutzeit reagiert.

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_WATCHDOG* = 0503h
2	*.usIndexFirst*	Device-Index:
		ML8_WATCHDOG (NMI)[1] = 0 *ML8_WATCHDOG_A (Timer-A Interrupt)* = 1 *ML8_WATCHDOG_B (Timer-B Interrupt)* = 2 *ML8_WATCHDOG_C (Timer-C Interrupt)* = 3
4	*.usIndexLast*	= *.usIndexFirst*
6	*.usFlags*	Mögliche Flags:
		Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv)
		Alle anderen Bits = 0.
12	*.ulTimeout*	Timeoutzeit in µs
16	*.usMode*	Reserviert, = 0 setzen

Alle anderen Strukturelemente werden nicht ausgewertet.

Die Timeoutzeit des Watchdogs wird beim Öffnen des Kanals festgelegt (Angabe in µs).

Aktiviert und deaktiviert wird der Watchdog mit dem Sonderdienst **mdd8_send_channel_command(handle, CMD_START);**

bzw.

mdd8_send_channel_command(handle, CMD_STOP);

[1] Die Timeout-Zeit wird mit Jumper J4 fest eingestellt. Möglich sind 100 ms oder 1,6 s. Der CPS-Parameter .ulTimeout hat keine Bedeutung.

Das Nachtriggern des Watchdog erfolgt durch Aufruf der Funktion

mdd8_trigger_channel(handle);

Der aktuellen Zustand des Watchdogs (0 = deaktiviert, 1 = aktiviert, 2 = Timeout) wird mit

status = mdd8_get_channel_info(handle, *INFO_DEVICE* = 48h);

abgefragt.

Serielle Schnittstellen

Zur Verwendung der seriellen Schnittstellen der Basiskarte (RS-232, asynchron) öffnen Sie den Kanal wie folgt:

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_SCC* = 0801h
2	*.usIndexFirst*	Device-Index:
		ML8_SCC_A = serielle Schnittstelle A = 0 *ML8_SCC_B* = serielle Schnittstelle B = 1
4	*.usIndexLast*	= *.usIndexFirst*
6	*.usFlags*	Mögliche Flags:
		Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv)
		Alle anderen Bits = 0.
16	*.usMode*	Betriebsart, s.u.

Alle anderen Strukturelemente werden nicht ausgewertet.

Des Senden und Empfangen der Nutzdaten erfolgt mit den Datentransferdiensten des Kanals. Die Modemsteuerleitungen können mit den Sonderdiensten des Kanals gesetzt bzw. abgefragt werden.

Vor dem Senden von Zeichen ist es erforderlich, den Status des Sendepuffers zu ermitteln. Nur wenn dieser leer ist, darf ein neues Zeichen gesendet werden.

Ob Zeichen empfangen wurden, sollte vor dem Lesen des Empfangspuffers durch prüfen des Puffers ermittelt werden.

Den Status von Sende- und Empfangspuffer kann man ebenfalls mit Sonderdiensten des Kanals abfragen.

Das CPS-Strukturelement *.usMode* bestimmt die Betriebsart (Baudrate, Datenbits, Stopbits und Parität) des seriellen Kanals. Die Parameter werden oder-verknüpft:

Baudraten:

_ML8_COM_110	= 0000h	110 Baud
_ML8_COM_150	= 0020h	150 Baud
_ML8_COM_300	= 0040h	300 Baud
_ML8_COM_600	= 0060h	600 Baud
_ML8_COM_1200	= 0080h	1200 Baud
_ML8_COM_2400	= 00A0h	2400 Baud
_ML8_COM_4800	= 00C0h	4800 Baud
_ML8_COM_9600	= 00E0h	9600 Baud
_ML8_COM_19200	= 0100h	19200 Baud
_ML8_COM_38400	= 0200h	38400 Baud

Datenbits:

_ML8_COM_CHR7	= 0002h	7 Datenbits
_ML8_COM_CHR8	= 0003h	8 Datenbits

Stoppbits:

_ML8_COM_STOP1	= 0000h	1 Stoppbit
_ML8_COM_STOP2	= 0004h	2 Stoppbits

Parität:

_ML8_COM_NO_PARITY	= 0000h	ohne Parität
_ML8_COM_EVEN_PARITY	= 0018h	gerade Parität
_ML8_COM_ODD_PARITY	= 0008h	ungerade Parität

Soll z.B. ein Kanal mit den Einstellungen: 9600 Baud, 8 Datenbits, 1 Stoppbit, ohne Parität verwendet werden, setzen Sie

```
.usMode = _ML8_COM_9600 + _ML8_COM_CHR8 + _ML8_COM_STOP1 +
          _ML8_COM_NO_PARITY;
```

Es stehen folgende Sonderdienste zur Verfügung:

ulStatus = mdd8_get_channel_info(handle, *INFO_DEVICE* = 48h);

liest den Status zurück. Die Bits haben folgende Bedeutung:

Bit 0: Sendepuffer-Status

1 (TRUE) : Zeichen wurde gesendet, Sendepuffer leer

Bit 1: Empfangspuffer-Status

1 (TRUE) : Zeichen wurden empfangen, Empfangspuffer voll

Bit 2: CTS-Status

Zustand der CTS Leitung

Bit 3: DCD-Status

Zustand der DCD Leitung

mdd8_send_channel_control(handle, *CTRL_DEVICE*, *ML8_SET_RTS* = 0);

RTS-Leitung setzen (=1)

mdd8_send_channel_control(handle, *CTRL_DEVICE*, *ML8_CLR_RTS* = 1);

RTS-Leitung löschen (=0)

mdd8_send_channel_control(handle, *CTRL_DEVICE*, *ML8_SET_DTR* = 2);

DTR-Leitung setzen (=1)

mdd8_send_channel_control(handle, *CTRL_DEVICE*, *ML8_CLR_DTR* = 3);

DTR-Leitung löschen (=0)

Hardware-Interrupts

Zur Programmierung von Interrupt-Tasks ist es notwendig, den verwendeten Interrupt und seine aktive Flanke festzulegen. Dazu verwenden Sie das Device HW_INT.

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_HW_INT* = 0B01h
2	*.usIndexFirst*	Device-Index:
		ML8_INT_NMI (SP-Bus, Spannungsüberw., Watchdog) = 0
		ML8_INT_IRQ_A (SP-Bus) = 1
		ML8_INT_IRQ_B (SP-Bus) = 2
		ML8_INT_IRQ_C (SP-Bus) = 3
		ML8_INT_IRQ_D (SP-Bus) = 4
		ML8_INT_IRQ_E (SP-Bus) = 5
		ML8_INT_IRQ_F (SP-Bus) = 6
		ML8_INT_IRQ_G (extern, St1) = 7
		ML8_INT_IRQ_H (extern, St3) = 8
		ML8_INT_TIMER_A = 9
		ML8_INT_TIMER_B = 10
		ML8_INT_TIMER_C = 11
		ML8_INT_TIMER_D = 12
		ML8_INT_TIMER_E = 13
		ML8_INT_TIMER_F = 14
		ML8_INT_SCC_A = 15
		ML8_INT_SCC_B = 16
4	*.usIndexLast*	= *.usIndexFirst*
6	*.usFlags*	Mögliche Flags:
		Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv)
		Alle anderen Bits = 0.
16	*.usMode*	Reserviert (=0)

Alle anderen Strukturelemente werden nicht ausgewertet.

Nur bei den externen Interrupts IRQ-A bis -H kann die aktive Flanke programmiert werden.

Der Sonderdienst CTRL_DEVICE stellt folgende Steuerbefehle zur Verfügung:

ML8_UNMASK_INT	= 0	Interrupt maskieren
ML8_MASK_INT	= 1	Interrupt demaskieren
ML8_POS_INT_EDGE	= 2	Positive Interruptflanke setzen
ML8_NEG_INT_EDGE	= 3	Negative Interruptflanke setzen:
ML8_CLR_INT	= 4	Interruptanforderung löschen
ML8_END_INT	= 5	Interrupt beenden (EOI)

Aufruf:

mdd8_send_channel_control(handle, *CTRL_DEVICE*, Steuerbefehle);

EEPROM

Ein Kanal zu einem EEPROM Device ermöglicht es, auf einen Eintrag im EEPROM der Basiskarte oder eines Moduls zuzugreifen. Der Zugriff auf die EEPROM-Inhalte kann direkt oder über eine Kopie im RAM der Karte erfolgen (schneller). In die Kopie geschriebene Daten gehen nach dem Abschalten der Karte verloren.

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_EEPROM* = 0A01h
2	*.usIndexFirst*	Device-Index = Modulsteckplatz (0 = Basiskarte)
4	*.usIndexLast*	= *.usIndexFirst*
6	*.usFlags*	Mögliche Flags:
		Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv)
		Alle anderen Bits = 0.
8	*.usReadMode*	Lesemodus:
		IO_MODE_DIRECT = 1 (direkt) *IO_MODE_RAM* = 4 (aus RAM)
10	*.usWriteMode*	Schreibmodus:
		IO_MODE_DIRECT = 1 (direkt) *IO_MODE_RAM* = 4 (aus RAM)

Alle anderen Strukturelemente werden nicht ausgewertet.

EEPROM-Wort anwählen:

mdd8_send_channel_control(handle, *CTRL_RW_PTR*, ulPointer);

EEPROM-Wort lesen bzw. schreiben:

wEEPROM = mdd8_read_channel_word(handle);

mdd8_write_channel_word(handle, wEEPROM);

Dabei wird der Lese-/Schreibzeiger automatisch um 1 inkrementiert!

Programmier-Beispiele

Timergesteuertes LED-Blinkprogramm (Echtzeit-Task)

Es sollen die on-board Leuchtdiode und Timer-B verwendet werden.

Öffnen Sie die Kanäle wie folgt:

```
ML8CPS.usDevice  = DEVICE_LED;
ML8CPS.usIndexFirst   = ML8_LOCAL_LED;
ML8CPS.usIndexLast    = ML8_LOCAL_LED;
ML8CPS.usFlags  = 0;
LED_Handle = mdd8_open_channel(wTask, sizeof(ML8CPS), &ML8CPS);

ML8CPS.usDevice  = DEVICE_TIMER;
ML8CPS.usIndexFirst   = ML8_TIMER_B;
ML8CPS.usIndexLast    = ML8_TIMER_B;
ML8CPS.usFlags  = 0;
TimerB_Handle = mdd8_open_channel(wTask, sizeof(ML8CPS), &ML8CPS);

ML8CPS.usDevice  = DEVICE_HW_INT;
ML8CPS.usFlags  = 0;
ML8CPS.usIndexFirst   = ML8_INT_TIMER_B;
ML8CPS.usIndexLast    = ML8_INT_TIMER_B;
IRQ_TimerB_Handle = mdd8_open_channel(wTask, sizeof(ML8CPS), &ML8CPS);
```

Starten Sie den Timer und geben Sie den Interrupt frei:

```
mdd8_write_channel_dword(TimerB_Handle, 1000); // Intervall = 1000µs
mdd8_send_channel_command(TimerB_Handle, CMD_START);
mdd8_send_channel_control(IRQ_TimerB_Handle, CTRL_DEVICE, UNMASK_INT);
```

Damit wird die Task gestartet. Alle 1000 µs wird die Hauptprozedur der Task per Interrupt aufgerufen.

In der Hauptprozedur der Task, die unter Timer-B Interrupt als II oder DI Task installiert werden muß, schaltet man die Leuchdiode ein bzw. aus.

```
mdd8_write_channel_byte(LED_Handle, 1); // LED einschalten
mdd8_write_channel_byte(LED_Handle, 0); // LED ausschalten
```

Wurde das Programm als direkte Interrupt-Task (DI) installiert, so muß in der Hauptprozedur der aufgetretene Interrupt gelöscht werden:

```
mdd8_send_channel_control(IRQ_TimerB_Handle, CTRL_DEVICE, END_OF_INT);
```

Verwendung der seriellen Schnittstellen (Echtzeit-Task)

Es sollen über die serielle Schnittstelle A der MODULAR-4/486 Daten empfangen und gesendet werden (asynchrone Kommunikation). Die Einstellungen für die Schnittstelle sollen sein: 4800 Baud, 7 Datenbits, 1 Stopbit, keine Parität.

Öffnen Sie den Kanal wie folgt:

```
ML8CPS.usDevice  = DEVICE_SCC;
ML8CPS.usIndexFirst   = ML8_SCC_A;
ML8CPS.usIndexLast    = ML8_SCC_A;
ML8CPS.usFlags   = 0;
ML8CPS.usMode    = _COM_4800 | _COM_CHR7 | _COM_STOP1 | _COM_NOPARITY;
SCC_Handle = mdd8_open_channel(wTask, sizeof(ML8CPS), &ML8CPS);
```

In der Hauptprozedur der Task, die als NI-Task zyklisch vom Betriebssystem aufgerufen wird, werden die empfangenen Zeichen eingelesen:

```
                              // prüfen, ob Receive Buffer Full
if(mdd8_get_channel_info(SCC_Handle, INFO_DEVICE) & CHECK_RBF)
                              // Zeichen einlesen
      bData = mdd8_read_channel_byte(SCC_Handle);
```

In einer weiteren Prozedur der Task, können Zeichen gesendet werden:

```
                              // prüfen, ob Transmit Buffer Empty
if(mdd8_get_channel_info(SCC_Handle, INFO_DEVICE) & CHECK_TBE)
                              // Zeichen senden
      mdd8_write_channel_byte(SCC_Handle, bData);
```

Sollen die Modemsteuerleitungen gesetzt werden, verwenden Sie folgende Aufrufe:

```
mdd8_send_channel_control(SCC1Handle, CTRL_DEVICE, SET_DTR);
mdd8_send_channel_control(SCC1Handle, CTRL_DEVICE, CLR_DTR);
mdd8_send_channel_control(SCC1Handle, CTRL_DEVICE, SET_RTS);
mdd8_send_channel_control(SCC1Handle, CTRL_DEVICE, CLR_RTS);
```