Ausschankstation

Andreas Spirka, 702789
Sven Eisenhauer, 707173

Stand: 25.01.2006

Praktikum Mikroprozessorsysteme 2
Termin 6 und 7

1. Einleitung

Dieses Dokument beschreibt die Software des Systems Ausschankstation.
Dabei handelt es sich um ein System zum Abfüllen einer bestimmten Menge Flüssigkeit in einen Behälter.
Zum Einsatz kommt dabei ein ARM-Prozessor mit der Peripherie zur Ansteuerung einer Waage und einer Pumpe. Die Waage liefert zwei Schingungsfrequenzen, die sich je nach Belastung der Waage ändern. Die eine wird größer, der andere kleiner. Durch die Differenz beider Periodendauern lässt sich das entsprechende Gewicht berechnen.
Die Pumpe wird durch Anlegen eines 50 Hz Signals aktiviert, durch ein dauerhaftes Low-Signal wird die Pumpe deaktiviert.
Die Ausgaben des Programms an den Benutzer erfolgen über die serielle Schnittstelle auf eine serielle Konsole.
Die Bedienung des Programms durch den Benutzer erfolgt über zwei Tasten (SW1 und SW2) direkt am System.
Laut Anforderungsdefinition sollen 50 ml bzw. g pro Abfüllvorgang abgefüllt werden.

2. Bedienungsanleitung

Dieser Abschnitt beschreibt den Ablauf eines Abfüll-Vorgangs.

1. Nach dem Start des Programms erscheint ein Begrüßungstext auf der seriellen Konsole.

2. Stellen Sie einen Becher oder ein Glas auf die Waage.

3. Drücken Sie die Taste SW1 zum tarieren der Waage.

4. Drücken Sie die Taste SW2 zum Start des Abfüllvorgangs.

5. Es werden 50 ml bzw. g abgefüllt und das aktuelle Nettogewicht wird auf der Konsole ausgegeben.

6. Auf der seriellen Konsole erscheint das Gesamtgewicht auf der Waage. (Behälter plus Inhalt)

7. Entfernen Sie den Becher.

8. Starten Sie einen weiteren Abfüllvorgang mit Schritt 2.

3. Funktionsbeschreibung

Dieser Abschnitt erläutert die Funktionsweise der einzelnen Programmteile.
Das Hauptprogramm sowie die meisten Hilfsroutinen sind in C geschrieben. Daneben gibt es noch einige Routinen, die in Assembler geschrieben sind.

- Hauptroutine
  Name: main
  Deklaration: int main (void)
  Übergabeparameter: keine
  Rückgabewert: int 0
  Datei: Termin6Aufgabe1.c
  Beschreibung: Diese Funktion initialisiert zuerst die benötigte Peripherie mit den Funktionen pmcinit, inits, Timer_Init, pioinit und gibt den Begrüßungstext

durch die Funktion greeting aus.
Danach läuft sie in einer Endlosschleife und reagiert auf die Änderungen der Variablen aktion durch den Interrupt-Handler.
Wird SW1 gedrückt, setzt der Interrupt-Handler aktion auf den Wert 1. In diesem Fall führt main einen Wiege-Vorgang aus und setzt das Tara-Gewicht auf diesen Wert.
Wird danach SW2 gedrückt, startet main die Pumpe durch die Funktion pumpestart. Jetzt mißt main zyklisch das Gewicht, bis das Abfüllgewicht von 50 g erreicht ist. Nun stoppt main die Pumpe.
Sobald der Behälter von der Waage genommen wird, startet ein weiterer Durchlauf.

- Interrupt-Handler
  Name: taste_irq_handler
  Deklaration: void taste_irq_handler (void) __attribute__ ((interrupt))
  Übergabeparameter: keine
  Rückgabewert: keiner
  Datei: Termin6Aufgabe1.c
  Beschreibung: Setzt die globale Variable aktion auf den entsprechenden Wert , je nachdem, von welcher Taste der Interrupt ausgelöst wurde. Dadurch wird die Hauptroutine gesteuert.
- Initialisierung des Power Managment Controllers
  Name: pmcinit
  Deklaration: void pmcinit(void)
  Übergabeparameter: keine
  Rückgabewert: keine
  Datei: Termin6Aufgabe1.c
  Beschreibung: Schaltet den Takt für die benötigte Peripherie ein. Hier: PIOA, PIOB, Timer5, Timer4, Timer3, USART0
- Timerinitialisierung
  Name: Timer_init
  Deklaration: void Timer_init(void)
  Übergabeparameter: keine
  Rückgabewert: keine
  Datei: Termin6Aufgabe1.c
  Beschreibung: Stellt die Timerblöcke des Systems entsprechend ihrer Verwendung ein.
  Timer 3: Wave-Mode zur Generierung des 50 Hz Signals für die Pumpe
  Timer 4: Capture-Mode zum messen der niedrigeren Schwingungsfrequenz der Waage
  Timer 5: Capture-Mode zum messen der höheren Schwingungsfrequenz der Waage
- Pioinitialisierung
  Name: pioinit
  Deklaration: void pioinit(void)
  Übergabeparameter: keine
  Rückgabewert: keine
  Datei: Termin6Aufgabe1.c
  Beschreibung: Diese Routine schaltet die Tasten SW1 und SW2 an Pio B für Interrupts frei, trägt die Funktion taste_irq_handler als Interrupt-Handler für Pio B als Interrupt-Quelle im Source Vector Register des AIC ein und ermöglicht den Interrupt.
- Initialisierung der seriellen Schnittstelle (Assembler)
  Name: inits
  Deklaration: inits()

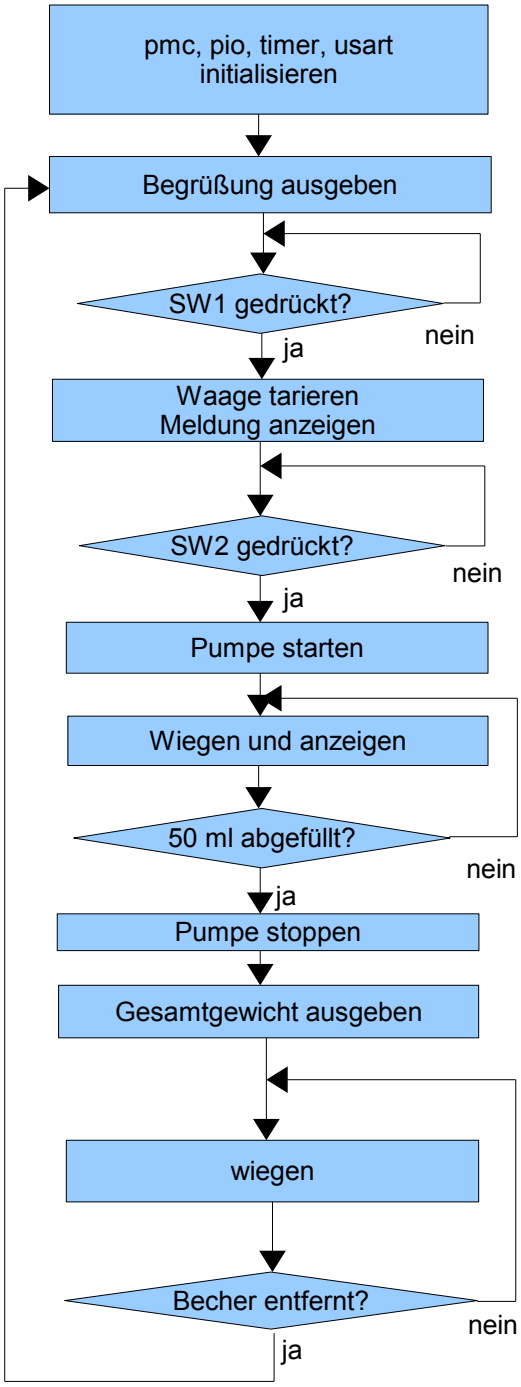
Übergabeparameter: keine
Rückgabewert: keine
Datei: ser_io.S
Beschreibung: Diese Routine führt per Software-Interrupt die Assembler-Routine init_ser aus. Diese initialisiert die USART0 auf eine Baudrate von 38400, 8 bit, 1 Stop bit, keine Parity.

- Routine für einen Wiege-Vorgang
  Name: massemessen
  Deklaration: int massemessen(void);
  Übergabeparameter: keine
  Rückgabewert: int masse
  Datei: Termin6Aufgabe1.c
  Beschreibung: Diese Routine berechnet aus den jeweiligen Capture-Registern A und B der Timer 4 und 5 die Werte aus und berechnet daraus unter Einbeziehung der spezifischen Konstanten der Waage die Masse, die sich aktuell auf der Waage befindet. Das Ergebnis dieser Berechnung wird zurück gegeben.
- Routine für zehn Wiege-Vorgänge
  Name: genaumessen
  Deklaration: int genaumessen(void)
  Übergabeparameter: keine
  Rückgabewert: int masse
  Datei: Termin6Aufgabe1.c
  Beschreibung: Diese Funktion führt zehn mal die Funktion massemessen aus und berechnet das arithmetische Mittel der 10 Messergebnisse. Damit sollen eventuelle Messungenauigkeiten kompensiert werden.
- Ausgabe einer Zeichenkette (Assembler)
  Name: puts
  Deklaration: void puts(char*)
  Übergabeparameter: Auszugebende Zeichenkette
  Rückgabewert: keine
  Datei: ser_io.S
  Beschreibung: Die Routine puts bekommt die Startadresse der auszugebenden Zeichenkette übergeben. Von dieser Adresse liest sie byteweise ein Zeichen und übergibt dieses per Software-Interrupt an die Assembler-Routine putchar. Diese gibt das Zeichen auf der seriellen Schnittstelle aus. Liest puts den aritmetischen Wert Null ('\0') interpretiert sie diesen als Ende der Zeichenkette und kehrt zur aufrufenden Funktion zurück.
- Software-Interrupt-Handler (Assembler)
  Name: SWIHandler
  Deklaration: SWIHandler
  Übergabeparameter: Offset der auszuführenden Routine
  Rückgabewert: kein
  Datei: swi.S
  Beschreibung: Diese Routine wird durch den Assembler Befehl swi ausgeführt. Die Zahl nach dem swi Befehl gibt den Offset der Routine an, die ausgeführt werden soll. SWIHandler berechnet diesen Offset und springt zur entsprechenden Routine.
- Ausgabe eines Zeichens (Assembler)
  Name: putchar
  Deklaration: putchar
  Übergabeparameter: Zu sendendes Zeichen
  Rückgabewert: keine
  Datei: seriell.S

Beschreibung: Diese Routine verschickt ein Zeichen über die serielle Schnittstelle

- Ausgabe von Zahlenwerten
  Name: Int2Str
  Deklaration: void Int2Str(int);
  Übergabeparameter: int Zahl
  Rückgabewert: keine
  Datei: Termin6Aufgabe1.c
  Beschreibung: Diese funktion wandelt die übergebene Zahl in eine Zeichenkette um. Dies geschieht zur Zahlenbasis 10 für jede Stelle der Zahl. Anschliessend wird die Zeichenkette mit der Funktion puts auf der seriellen Schnittstelle ausgegeben.
- Starten der Pumpe
  Name: pumpestart
  Deklaration: void pumpestart(void)
  Übergabeparameter: keine
  Rückgabewert: keine
  Datei: Termin6Aufgabe1.c
  Beschreibung: Die Funktion startet die Pumpe, in dem sie dem Timer 3, der ein 50 Hz Signal generiert den Pin zuweist, an dem die Pumpe angeschlossen ist.
- Stoppen der Pumpe
  Name: pumpestop
  Deklaration: void pumpestop(void)
  Übergabeparameter: keine
  Rückgabewert: keine
  Datei: Termin6Aufgabe1.c
  Beschreibung: Diese Routine weist den Pin, an dem die Pumpe angeschlossen ist, der Pio zu. Danach wird der Pin als Output konfiguriert und das Output Data Register für den Pin gelöscht. Dadurch liegt ein Low-Signal an dem Pin und die Pumpe stopt.
- Ausgabe des Begrüßungstextes
  Name: greeting
  Deklaration: void greeting(void)
  Übergabeparameter: keine
  Rückgabewert: keine
  Datei: Termin6Aufgabe1.c
  Beschreibung: Diese Funktion gibt mittels der Routine puts einen Begrüßungstext auf der seriellen Schnittstelle aus.
- Ausgabe des Endzustands
  Name: ausgabe
  Deklaration: void ausgabe(int m)
  Übergabeparameter: int Gesamtgewicht
  Rückgabewert: keine
  Datei: Termin6Aufgabe1.c
  Beschreibung: Diese Funktion gibt das Gesamtgewicht nach Beendigung des Abfüllvorgangs auf der seriellen Konsole aus. Dazu verwendet sie die Routinen puts und Int2Str.

4. Programm-Ablauf-Plan



5.Sourcecode Listings

```
// Lösung zu Termin6&7
// Aufgabe 1...
// Namen: Andreas Spirka; Sven Eisenhauer
// Matr.: 702789; 707173
// vom :  25.01.2006



#include "../h/pmc.h"
#include "../h/tc.h"
#include "../h/pio.h"
#include "../h/aic.h"

void pmcinit(void);
void pioinit(void);
void Timer_init(void);
int massemessen(void);
void pumpestart(void);
void pumpestop(void);
void greeting(void);
void ausgabe(int);
int genaumessen(void);
void Int2Str(int);
void taste_irq_handler (void) __attribute__ ((interrupt));

volatile int aktion=0;
char lf[2];
char cr[2];

// für die Initialisierung der Zähler TC4 und TC5

#define TC4_INIT  TC_CLKS_MCK2 | TC_LDBSTOP | TC_CAPT | TC_LDRA_RISING_EDGE |
TC_LDRB_RISING_EDGE
#define TC5_INIT  TC_CLKS_MCK2 | TC_LDBSTOP | TC_CAPT | TC_LDRA_RISING_EDGE |
TC_LDRB_RISING_EDGE

// Interruptserviceroutine für die Tasten SW1 und SW2
void taste_irq_handler (void)
{
  StructPIO* piobaseB   = PIOB_BASE;                // Basisadresse PIO B
  StructAIC* aicbase  = AIC_BASE;                   // Basisadresse Advanced
Interrupt Controller
```

```
  int taste = piobaseB->PIO_PDSR;
          if (!(taste & KEY1))            // falls Schalter 1 gedrückt
                    aktion=1;
          if (!(taste & KEY2))            // falls Schalter 2 gedrückt
                    aktion=2;


          aicbase->AIC_EOICR = piobaseB->PIO_ISR;   // AIC End of Interrupt
Command Register = 1
}


void pmcinit(void)
{
          StructPMC* pmcbase = PMC_BASE;
          pmcbase->PMC_PCER = 0x06f84;   // Clock PIOA, PIOB, Timer5, Timer4,
Timer3,US0 einschalten
}


void pioinit(void)
{
          StructPIO* piobaseB   = PIOB_BASE;                  // Basisadresse
PIO B
          StructAIC* aicbase  = AIC_BASE;                     // Basisadresse
Advanced Interrupt Controller
          // disable all interrupt sources of pio
          piobaseB->PIO_IDR = 0xFFFFFFFF;
          piobaseB->PIO_PER = KEY1|KEY2;            // Schalter 1-2 enabeled
          // Interrupt Initialisierung für PIOB (0x4000)
          aicbase->AIC_IDCR = (1<<PIOB_ID);                   // Interrupt
disabled
          aicbase->AIC_ICCR = (1<<PIOB_ID);                   // Interrupt
clear
          aicbase->AIC_SMR[PIOB_ID] = 0x1;          // An Stelle 14 Level
Sensitive / Höchste Priorität
          aicbase->AIC_SVR[PIOB_ID] = (unsigned int)taste_irq_handler;   //
Adresse der Interrupt Service Routine in Vektor-Tabelle
          aicbase->AIC_IECR = (1<<PIOB_ID);                   // enable PIOB
Interrupt in AIC
          piobaseB->PIO_IER = KEY1|KEY2;            // Schalter 1+2 lösen
Interrupts aus
}


void Timer_init(void)
{
  StructTC* tcbase3 = TCB3_BASE;                      // Basisadressse TC Block
3 Pumpe
  StructTC* tcbase4 = TCB4_BASE;                      // Basisadressse TC Block
4 Waage
  StructTC* tcbase5 = TCB5_BASE;                      // Basisadressse TC Block
```

```
5 Waage
  StructPIO* piobaseA   = PIOA_BASE;                // Basisadresse PIO A

          tcbase3->TC_CCR = TC_CLKDIS;                          // Disable Clock

  // Initialize the mode of the timer 3
  tcbase3->TC_CMR =
    TC_ACPC_CLEAR_OUTPUT  |    //ACPC    : Register C clear TIOA
    TC_ACPA_SET_OUTPUT    |    //ACPA    : Register A set TIOA
    TC_WAVE               |    //WAVE    : Waveform mode
    TC_CPCTRG             |    //CPCTRG  : Register C compare trigger enable
    TC_CLKS_MCK1024;           //TCCLKS  : MCKI / 1024

  // Initialize the counter:
  tcbase3->TC_RA = 244;
  tcbase3->TC_RC = 488;                                                 // RA
= RC/2 für symmetrisches Signal

  // Start the timer :
  tcbase3->TC_CCR = TC_CLKEN ;                                  //__
  tcbase3->TC_CCR = TC_SWTRG ;                                  //__
  piobaseA->PIO_PER  = (1<<PIOTIOA3) ;    // Pio herrscht über pin
  piobaseA->PIO_OER  = (1<<PIOTIOA3) ;    // Output an pin
  piobaseA->PIO_CODR = (1<<PIOTIOA3) ;    // clear output -> low signal, pumpe
aus

// Periodendauer der Waagensignale messen
// Signal aud TIOA4 ca. 16kHz entspricht ca. einer Periodendauer von 62,5us
// durch den Teiler von 32 ergeben sich ca. 2ms
// Zähler mit positiver Flanke starten

          //piobaseA->PIO_PDR  =         0x090;
          piobaseA->PIO_PDR    =         (1<<PIOTIOA4)|(1<<PIOTIOA5);
          tcbase4->TC_CCR                =         TC_CLKDIS;
          tcbase4->TC_CMR                =         TC4_INIT;
          tcbase4->TC_CCR                =         TC_CLKEN;
          tcbase4->TC_CCR                =         TC_SWTRG;

          tcbase5->TC_CCR                =         TC_CLKDIS;
          tcbase5->TC_CMR                =         TC5_INIT;
          tcbase5->TC_CCR                =         TC_CLKEN;
          tcbase5->TC_CCR                =         TC_SWTRG;
}
```

```
int massemessen(void)
{
          StructTC* tcbase4 = TCB4_BASE;
          StructTC* tcbase5 = TCB5_BASE;
          volatile int         captureRA1;
          volatile int         captureRB1;
          volatile int         capturediff1;
          volatile float Periodendauer1;
          volatile int         captureRA2;
          volatile int         captureRB2;
          volatile int         capturediff2;
          volatile float Periodendauer2;
          volatile int         c1=18030;
          volatile int         c2=40;
          volatile int         masse;

          tcbase4->TC_CCR      =          TC_SWTRG;
          tcbase5->TC_CCR      =          TC_SWTRG;
          while (!( tcbase4->TC_SR & TC_LDBSTOP));            // Capture
Register B wurde geladen Messung abgeschlossen
                    captureRA1           = tcbase4->TC_RA;
                    //
                    captureRB1           = tcbase4->TC_RB;
                    capturediff1         =          abs(captureRB1) -
abs(captureRA1);
                    Periodendauer1 = abs(capturediff1);
          while (!( tcbase5->TC_SR & TC_LDBSTOP));            // Capture
Register B wurde geladen Messung abgeschlossen
                    captureRA2           = tcbase5->TC_RA;
                    //
                    captureRB2           = tcbase5->TC_RB;
                    capturediff2         =          abs(captureRB2) -
abs(captureRA2);
                    Periodendauer2 = abs(capturediff2);

                    masse = c1 * ((Periodendauer1 / Periodendauer2) -1) -c2;
                    return masse;
}

void pumpestart(void)
{
          StructPIO* piobaseA = PIOA_BASE;
          piobaseA->PIO_PDR = (1<<PIOTIOA3);        // Timer herrscht über
Bit (Taktsignal)
```

```
}

void pumpestop(void)
{
          StructPIO* piobaseA = PIOA_BASE;
          // Vermeiden, daß dauerhaftes High Signal (PIO herrscht über das BIT)
          piobaseA->PIO_PER  = (1<<PIOTIOA3) ;      // PIN der PIO zuweisen
          piobaseA->PIO_OER  = (1<<PIOTIOA3) ;      // wird auf Output gesetzt
          piobaseA->PIO_CODR = (1<<PIOTIOA3) ;      // Clear Output (low
Signal)
}

void greeting(void)
{
          cr[0]=0x0d;
          lf[0]=0x0a;
          lf[1]='\0';
          cr[1]='\0';
          puts("AUSSCHANKSTATION\0");
          puts(&cr[0]);
          puts(&lf[0]);
          puts("Bitte Becher auf die Waage stellen\0");
          puts(&cr[0]);
          puts(&lf[0]);
          puts("SW1 zum tarieren drücken\0");
          puts(&cr[0]);
          puts(&lf[0]);
}

void ausgabe(int m)
{
          cr[0]=0x0d;
          lf[0]=0x0a;
          lf[1]='\0';
          cr[1]='\0';
          puts("Gemessene Gesamtmasse: \0");
          Int2Str(m);
          puts(&cr[0]);
          puts(&lf[0]);
          puts("Becher kann entfernt werden\0");
          puts(&cr[0]);
          puts(&lf[0]);

}
```

```
void Int2Str(int m)
{
          int a,i=7;
          char string[9];
          string[8] = '\0';

          while(m != 0)
          {
                    a = m%10;
                    string[i] = a + '0';
                    m = m/10;
                    i--;
          }
          puts(&string[i+1]);
}

int genaumessen(void)
{
          volatile int durchlaeufe=10;
          volatile int summe=0;
          volatile int retVal=0;
          volatile int i=0;
          for (i=0;i<durchlaeufe;i++)
          {
                    summe+=massemessen();
          }
          retVal= summe/durchlaeufe;
          return retVal;
}

int main(void)
{
          volatile int masse=0;
          volatile int tara=0;
          volatile int netto=0;

          cr[0]=0x0d;
          lf[0]=0x0a;
          lf[1]='\0';
          cr[1]='\0';

          inits();
```

```
	greeting();
	pmcinit();
	Timer_init();
	pioinit();

	for(;;)
	{
		if(aktion == 1)
		{
			puts("Tara: \0");
			tara=genaumessen();
			netto=0;
			Int2Str(tara);
			puts(" SW2 startet die Pumpe");
			puts(&cr[0]);
			aktion=0;
		}
		if (aktion == 2 && tara != 0)
		{
			puts(&cr[0]);
			puts(&lf[0]);
			pumpestart();
			while(netto <= 50)
			{
				masse=genaumessen();
				netto=masse-tara;
				puts("Fuellgewicht: \0");
				Int2Str(netto);
				puts(&cr[0]);
			}
			pumpestop();
			ausgabe(masse);
			do {
				masse=genaumessen();
			}
			while (masse > 10);
			// Gewicht hat sich geändert, d. h. Becher wurde
entfernt
			// D. h. von vorne starten
			greeting();
			aktion=0;
		}
	}
	// disable piob command register in AIC
```

```
            //aicbase->AIC_IDCR = (1<<PIOB_ID);
            // disable all interrupt sources of pio
            //piobaseB->PIO_IDR = 0xFFFFFFFF;
            // Timer (0x200), PIOA (0x2000), PIOB (0x4000) ausschalten
            //pmcbase->PMC_PCDR = (1<<PIOA_ID)|(1<<PIOB_ID)|(1<<TC3_ID);
            return 0;
}
```

Termin6Aufgabe1.c

```
@------------------------------------------------------------------------------
@ File Name           : ser_io.S
@ Object              : Ein- Ausgabe-Funktionen der seriellen Schnittstelle
@
@ Namen :
                      Matr.-Nr.:
@                                        :
                                                              Matr.-Nr.:
@
@------------------------------------------------------------------------------

@ Debuginformationen
            .file     "ser_io.S"

@ Funktion
            .text
            .align    2
            .global   inits
            .type     inits,function
inits:
            stmfd sp!,{lr}       @ Retten der Register
            swi 0
            ldmfd sp!,{pc}       @ Rücksprung

@ Funktion
            .text
            .align    2
            .global   puts
            .type     puts,function
puts:
            stmfd sp!,{lr}       @ Retten der Register

// Hier muß Ihr Code eingefügt werden.
            mov       r5, r0                                    @ Anfangsadresse
```

```
des kompletten Strings von r0 nach r5 kopieren
loop:     ldrb      r0, [r5], #1                              @ Holen des
Zeichens der Adresse in r5 nach r0 und erhöhe dann Adresse in r5
          cmp       r0, #0                                    @ Ende des
Strings? entspricht binärer Null
          beq       L1                                        @ Wenn
Stringende erreicht
          swi 1                                               @ Auslösen der
SWI Methode putchar (swi + offset)
          b         loop                                      @ weiter bei
loop solange Stringende nicht erreicht
L1:
          @mov      r0, #0x0d                       @ Carriage Return
          @swi 1                                              @ Auslösen der
SWI Methode putchar (swi + offset)
          @mov      r0, #0x0a                       @ Linefeed
          @swi 1                                              @ Auslösen der
SWI Methode putchar (swi + offset)
          ldmfd sp!,{pc}                                      @ Rücksprung


@ Funktion
          .text
          .align    2
          .global   gets
          .type     gets,function
gets:
          stmfd     sp!,{lr}   @ Retten der Register


// Hier könnte Ihr Code eingefügt werden!


          ldmfd sp!,{pc}       @ Rücksprung


.end
```

ser_io.S

```
@-------------------------------------------------------------------------
@ File Name                                                              :
seriell.S
@ Object                                                                 :
Grundfunktionen der seriellen Schnittstelle
@
@ 1.0 27/10/02 GR              : Creation
@-------------------------------------------------------------------------


          .file     "seriell.S"


#include "../h/pmc.inc"
#include "../h/pio.inc"
#include "../h/usart.inc"

DEFAULT_BAUD = 38400
CLOCK_SPEED = 25000000
US_BAUD =  0x29      @ CLOCK_SPEED / (16*(DEFAULT_BAUD))

@ Funktion
          .text
          .align    2
          .global   init_ser
          .type     init_ser,function
init_ser:
          stmfd               sp!, {r0-r3, lr}               @ Register
retten
          adr                           r0,L1
          adr                           r1,L1_end
init_ser_loop:
          ldmia               r0!, {r2-r3}
          cmp                           r0, r1
          str                           r3, [r2]
          bne                           init_ser_loop
          ldmfd               sp!, {r0-r3, pc}               @ Rücksprung
L1:
          .word     PMC_BASE+PMC_PCER, 0x4
          .word     PIOA_BASE+PIO_PDR, 0x18000
          .word     USART0_BASE+US_CR, 0xa0
          .word     USART0_BASE+US_MR, 0x8c0
          .word     USART0_BASE+US_BRGR, US_BAUD
          .word     USART0_BASE+US_CR, 0x50
L1_end:
```

```
@ Funktion
          .text
          .align    2
          .global   putchar
          .type     putchar,function
putchar:
          stmfd               sp!, {r0-r2, lr}              @ Register
retten
          ldr                           r2, =USART0_BASE
1:
          ldr                           r1, [r2, #US_CSR]
          tst                           r1, #US_TXRDY       @ ist
Transmitter frei
          beq                           1b
          str                           r0, [r2,#US_THR]
          ldmfd               sp!, {r0-r2, pc}              @ Rücksprung

@ Funktion
          .text
          .align    2
          .global   getchar
          .type     getchar,function
getchar:
          stmfd               sp!, {r1, r2, lr}             @ Register
retten
          ldr                           r2, =USART0_BASE

1:
          ldr                           r1, [r2, #US_CSR]
          ands                r1, r1, #US_RXRDY
          beq                           1b
          ldr                           r0, [r2, #US_RHR]
          ldmfd               sp!, {r1, r2, pc}             @ Rücksprung

.end
```

seriell.S

```
@-------------------------------------------------------------------------
@ File Name           : swi.S
@ Object              : SoftwareInterruptHandler
@
@ 1.0 27/10/02 GR      : Creation
@
@-------------------------------------------------------------------------



          .global SWIHandler
          .text
SWIHandler:


          ldr                             sp, STACK           @ den Wert von
STACK (0x78c) auf sp kopieren
          stmfd               sp!, {lr}                       @ retten der
Rücksprungadresse
          ldr                             ip,[r14, #-4]                   @ hole
"swi X" Aufruf in ip (lr steht auf Aufruf nach swi Befehl im aufrufenden
Programm)
          bic                             ip, ip, #0xff000000  @ maskiere X aus
und speichere in ip
          mov                             ip, ip, lsl #2                  @ X
ist in ip, X * 4 (entspricht lsl #2) ist Offset des swi
          ldr                             lr, =SWIJumpTable    @ Lade Adresse
von SWIJumpTable in lr
          ldr                             ip, [lr, ip]                    @
Addiere ip (Offset des SWI) zu Adresse SWIJumpTable und speichere in ip
          mov                             lr, pc                          @
Speichere pc in lr für Rücksprung
          mov                             pc, ip                          @ Lade
ip nach pc, entspricht Sprung in jeweilige SWI-Routine
          ldmfd               sp!, {pc}^                              @
Rücksprung
STACK:
          .word 0x78c


SWIJumpTable:
          .word init_ser
          .word     putchar
          .word     getchar
.end
```

swi.S