MICROCONTROLLER - BEISPIELE

Übung 1 - Das erste Assembler Programm

Aufgabe 2 - Register setzen und subtrahieren

```
/*
 * U1A2.asm
 *
 *  Created: 22.02.2012 17:06:54
 *   Author: mmeschenmoser, kilkow
 */

.INCLUDE "usb1287def.inc"

.ORG    0                       //set start address of code
start:                          //initialize
        ldi r16,5               //load imediate: Wert 5 in Register 16

loop:
        subi r16,3              //subtrahiere den Wert 3 im Register 16
        cpi r16,-8              //compare imediate: if r16 < -8 dann s -> true
        brge loop               //wenn s ungleich true, gehe zu loop:

end:                            //endless loop
        rjmp end                //return to jump
```

Aufgabe 3 - Bitmuster

```
start:                          //initialize
        ldi r16,255             //lauter 1'en

loop:
        lsr r16                 //null von links reinschieben
        cpi r16,0               //compare imedaite with 0
        brne loop               //springe falls keine ☼/span>bereinstimmung

        jmp start               //wiederhole alles
```

Aufgabe 4 - Bitmuster 2

```
start:                          //initialize
        ldi r16,255             //lade wert 255 in Register r16
        clc                     //clear carry flag: c -> 0

loop:
        ror r16                 //rotiere nach rechts
        rjmp loop               //re-jump
```

Aufgabe 5 - Register invertieren

```
start:                          //initialize
        ldi r16,15              //load imediate 15 in Register 16
        com r16                 //Einerkomplement (alles umkehren)
        rjmp start              //re-jump
```

Übung 2 - SRAM, Hardware, IO-Ports

Aufgabe 1 - dito 1.2, aber mit Zwischenspeicherung

```
.INCLUDE "usb1287def.inc"
.equ var_a = 0x100              //definiert eine Variable

.ORG    0                       //set start address of code
start:                          //initialize
        ldi r16,5               //load imediate Wert 5 in Register 16
        sts var_a,r16           //speichere den Wert von r16 in var_a
loop:
        clr r16                 //clear r16 (optional)
        lds r16,var_a           //lädt den Wert von var_a in r16 rein
        subi r16,3              //subtrahiere den Wert 3 im Register 16
        cpi r16,-8              //if r16 <-8 dann true
        sts var_a,r16           //speichere r16 wieder in var_a
        brge loop               //wenn ungleich true, gehe zu loop:

end:                            //endless loop
        rjmp end                //return to jump
```

Aufgabe 2 - LED ansteuern

```
start:                          //initialize
    ldi r16,0x01                //acht-bit vector mit "0000 0001"
    out DDRA,r16                //definiere DDRA(0) als output
    out PORTA,r16               //sende zu PORTA eine 1

end:                            //endless loop
    rjmp end                    //return to end
```

Aufgabe 3 - Tasten zu LED (mit Polling - ständiges Prüfen in Schleife)

```
start:                          //initialize
    ldi r16,0b00001111          //setze Register mit vier Ausgängen
    out DDRA,r16                //definiere DDRA(0) als output

loop:
    in r17,PINB                 //speichere Ist-werte der Schalter in r17
    swap r17                    //tauscht 0-3 mit 4-7
    out PORTA,r17               //gibt diese wieder aus
    clr r17                     //clear r17
    jmp loop
```

Aufgabe 4 - PortB Pin 6 -> PortA Pin 1

```
start:                          //initialize
    ldi r16,0b00000001          //setze Register mit vier Ausgängen
    out DDRA,r16                //definiere DDRA(0) als output

loop:
    in r17,PINB                 //speichere Ist-werte der Schalter in Register
    bst r17,6                   //lese das sechste bit und schreibe in T
    clr r17
    bld r17,0                   //schreibe von T in letztes Bit
    out PORTA,r17               //gibt diese wieder aus
    jmp loop
```

Aufgabe 5 - Nach 5 Klicks den Zustand des LED zu wechseln

```
start:                  //initialize
    ldi r16,0b00000001  //setze Register mit vier Ausgängen
    ldi r22,0b00000000  //setze Register mit vier Ausgängen (letzte aus)
    out DDRA,r16        //definiere DDRA(0) als output
    clr r18             //letzer Zustand
    clr r19             //aktueller Zustand
    clr r20             //Anzahl Veränderungen
    clr r21             //hat das LED geleuchtet???
    clr r23             //letztes Bit von r21
    in r17,PINB         //speichere Ist-werte der Schalter in Register 17
    bst r17,6           //lese das sechste bit und schreibe in T
    bld r18,0           //schreibe von T in letztes Bit

loop:
    in r17,PINB         //speichere Ist-werte der Schalter in Register
    bst r17,6           //lese das sechste bit und schreibe in T
    bld r19,0           //schreibe von T in letztes Bit
    cp r18,r19
    breq loop           //spring nach oben, falls noch gleicher Zustand
                        //Jetzt ist etwas geändert worden
    mov r18,r19         //schreiber19 in r18
    inc r20             //erhöhe r20 um 1
    cpi r20,5
    brlo loop           //spring nach oben, wenn r20<5
                        //jetzt sind fünf Änderungen geschehen
    clr r20             //counter nullsetzen
    in r21,PINA
    bst r21,0
    bld r23,0           //letztes Bit in r23 schreiben
    cpi r23,1           //muss das LED ein- oder ausgeschaltet werden?
    breq LEDaus
    out PORTA,r16       //schalte LED ein
    jmp loop

LEDaus:
    out PORTA,r22       //schalte LED aus
    jmp loop
```

Aufgabe 6 - Nach 3 positiven Flanken, LED wechseln

```
fast dito A5

cpi r19,1               //aufsteigende Flanke
```

Übung 3 - Adressierungsarten, Delay

Aufgabe 1 - Data Direct/Indirect Adressing

```
.INCLUDE "usb1287def.inc"
.EQU    var_a=0x0320            //Dezimal 800
.ORG    0                       // set start address of code

start:
                                //Data Direct Addressing
    ldi r16,0xA6                //lade 166 in Register r16
    STS var_a,r16               //lade Inhalt von r16 in var_a (0x0320)

                                //Data Indirect Adressing (X-Register)
    LDI XL,LOW(0x0321)          //X die Speicheradresse angeben, "21"
    LDI XH,HIGH(0x0321)         //X die Speicheradresse angeben, "03"
    ST X,r16                    //lade Inhalt von r16 in Register X (0x0321)
```

Aufgabe 2 - Datenspeicher belegen

```
start:                          //initialize
    ldi r16,10
    ldi r17,3
    ldi r18,0
    LDI XL,LOW(0x0320)          //X die Speicheradresse angeben, "20"
    LDI XH,HIGH(0x0320)         //X die Speicheradresse angeben, "03"
loop:
                                //Data Indirect Adressing
    ST X+,r16                   //store r16 in data space 0x0320 (with pos inc)
    add r16,r17                 //addiere 3 zu r16 dazu und speichere es in r16
    inc r18                     //increase r18

    cpi r18,50                  //vergleiche r18 mit 50
    brlo loop                   //bei kleiner, macht er branch/springe
```

Aufgabe 4 - Blinklicht mit 20Hz

```
start:                          //initialize
    ldi r16,0b00000001          //setze Register mit einem Ausgang
    out DDRA,r16                //definiere DDRA(0) als output
    ldi r17,0                   //letzer Zustand

start1:
                                //Z swetzen (Zähler)
    LDI ZL,LOW(50000)           //200000/4
    LDI ZH,HIGH(50000)          //200000/4

loop:
    sbiw ZH:ZL, 1               //subtrahiere 1 vom Z-Pointer
    brne loop
                                //Status wechseln
    com r17                     //0 zu 1, 1 zu 0 (Einerkomplement =invertieren)
    out PORTA,r17               //schalte LED aus/ein, wechsle Zustand
    rjmp start1
```

Übung 4 - Stack, Subroutinen, Clock

Lernaufgabe - Analyse mit brsh

```
start:                          // initialize
    clc
    ldi r16,5
loop:
    subi r16,3                  //subtrahiert den Wert 3 und schreibt ihn ins Register
    cpi r16, -8                 //subtrahiert den Wert 3 ohne ihn ins
                               Register zu schreiben und setzt das carry
    brsh loop                   // same or higher:        schaut aufs carry-bit
                               //brpl loop       // plus:                  schaut aufs
negativ-bit
```

Aufgabe 1 - Delay von 10us mithilfe von Subroutine

```
start:                          // initialize
    ldi r17,0b00000001          //setze Register mit einem Ausgang
    out DDRA,r17                //definiere DDRD(0) als output
    ldi r18,0
                               //Initialisierung des Stacks
    ldi r16, HIGH(RAMEND)
    out sph, r16
    ldi r16, LOW(RAMEND)
    out spl, r16

loop:
                               //LED ein
    com r18
    out PORTA,r18               //schalte LED aus

                               //subroutine fr den Delay
    call delay10us

                               //LED aus
    com r18                     //0 zu 1, 1 zu 0 (Einerkomplement)
    out PORTA,r18               //schalte LED aus

                               //subroutine fr den Delay
    call delay10us

    rjmp loop                   //Endlosschleife

                               //subroutine
delay10us:
                               //Z swetzen (Zähler)
        LDI ZL,LOW(17)
        LDI ZH,HIGH(17)         //80 werden benötigt
    subloop:
        sbiw ZH:ZL, 1           //hinabzählen
        brne subloop

        ret                     //return

                               //5xcall, 1x2xLDI            einmalig = 7
                               //2xsbiw, 2xbrne,            loop = ? = 68/4 = 17
                               //5xret                      einmalig = 5
```

Aufgabe 3 - togglen

```
     //1x com, 1x out, 1x com, 1x out, 2x rjmp = 6
```

Übung 5 - Interrupts, Timer0

Aufgabe 1 - Blinklicht mit Interrupts

```
.include "usb1287def.inc"       //Import fr die Konstanten
.EQU toStart = 178              //konstante um start des zähler zu setzen
.EQU cntStart = 25              //wo soll er beginnen
.EQU cntAddr = 0x100            //Speicheradresse fr Speicher
.ORG 0                          //Programmzähler
   rjmp start                   //Sprung zum Programm (min zwei Zeilen abstand)
.ORG OVF0addr                   //Overflow
    rjmp T0ISR
.ORG 0x50

start:                          //Programmbeginn
    ldi r16, HIGH(RAMEND)       //Initialisierung des Stacks
    out sph, r16
    ldi r16, LOW(RAMEND)
    out spl, r16
    ldi r17,0b00000001          //PORT A als Ausgang intialisieren
    out DDRA,r17                //definiere DDRA(0) als output
    out PORTA,r17               //schalte LED ein
    ldi r19,cntStart            //lade Startwerte in Zähler
    STS cntAddr, r19
                               //Counter 0 auf enable setzen (GIE)
    ldi r20,0b00000001
    STS TIMSK0, r20
    SEI                         //Setze das interrupt bit im Statusregister
                               //Initialisiere den Timer 0
    ldi r21, 0b00000000
    ldi r22, 0b00000101
    ldi r23, toStart
    out TCCR0A, r21
    out TCCR0B, r22
    out TCNT0, r23

loop: rjmp loop
T0ISR:
                               //Register und Statusregister sichern
    in r24, SREG
    push r24
    push r18
    push r19
                               // Eigentlicher Code
    lds r18,cntAddr             //Zählvariable in r18 laden
    dec r18                     //1 abzählen
    STS cntAddr,r18
    cpi r18,0
    brne ende                   //springe zurck wenn noch nicht 25
    ldi r19,cntStart
    STS cntAddr, r19
    com r17                     //ein-> aus      aus-> ein
    out PORTA,r17               //schalte LED um

ende:
    out TCNT0, r23
    pop r19                     //Register und Statusregister laden
    pop r18
    pop r24
    out SREG, r24

RETI
```

Aufgabe 2 - Blinklicht ein- und ausschalten

```
.include "usb1287def.inc"       //Import fr die Konstanten
.EQU toStart = 178              //konstante um start des zähler zu setzen
.EQU cntStart = 25              //wo soll er beginnen
.EQU cntAddr = 0x100            //Speicheradresse fr Speicher

.ORG 0                          //Programmzähler
   rjmp start                   //Sprung zum Programm (immer zwei Zeilen abstand)
.ORG PCI0addr                   //Taste gedrckt
    rjmp PUSHERISR
.ORG OVF0addr                   //Overflow
    rjmp T0ISR
.ORG 0x50

start:                          //Programmbeginn
                               //Initialisierung des Stacks
    ldi r16, HIGH(RAMEND)
    out sph, r16
    ldi r16, LOW(RAMEND)
    out spl, r16

                               //PORT A als Ausgang intialisieren
    ldi r17,0b0000001           //setze Register mit einem Ausgang
    out DDRA,r17                //definiere DDRA(0) als output
    out PORTA,r17               //schalte LED ein

                               // lade Startwerte in Zähler
    ldi r19,cntStart
    STS cntAddr, r19

                               // PinChange Interrupt einstellen
    ldi r24,0b00000001
    ldi r25,0b11111111
    STS PCICR, r24
    STS PCMSK0, r25

                               // Setze das interrupt bit im Statusregister
    SEI

                               //Initialisiere den Timer 0
    ldi r21, 0b00000000
    ldi r22, 0b00000101
    ldi r23, toStart
    out TCCR0A, r21
    out TCCR0B, r22
    out TCNT0, r23

loop: rjmp loop



...
```

```
T0ISR:

                                //Register und Statusregister sichern
    in r24, SREG
    push r24
    push r18
    push r19
    push r20

    lds r18,cntAddr             //Zählvariable in r18 laden
    dec r18                     //1 abzählen
    STS cntAddr,r18
    cpi r18,0
    brne ende                   //springe zurck wenn noch nicht 25

    ldi r19,cntStart
    STS cntAddr, r19
    com r17                     //ein-> aus      aus-> ein
    out PORTA,r17               //schalte LED um

ende:
    out TCNT0, r23

                               //Register und Statusregister laden
    pop r20
    pop r19
    pop r18
    pop r24
    out SREG, r24

RETI


PUSHERISR:
                               //Register und Statusregister sichern
    in r24, SREG
    push r24
    push r20

                               // Counter 0 toggeln
    LDS r20,TIMSK0
    LDI r24, 1
    EOR r20,r24
    STS TIMSK0, r20

                               //Register und Statusregister laden
    pop r20
    pop r24
    out SREG, r24
RETI
```

Übung 6 - Sleep Modus, Arithmetik

Aufgabe 1 - Sleep Modus

```
.include "usb1287def.inc"       //Import fr die Konstanten
.EQU toStart = 178              //konstante um start des zähler zu setzen
.EQU cntStart = 25              //wo soll er beginnen
.EQU cntAddr = 0x100            //Speicheradresse fr Speicher

.ORG 0                          //Programmzähler
   rjmp start                   //Sprung zum Programm (immer zwei Zeilen abstand)
.ORG OVF0addr                   //Overflow
    rjmp T0ISR
.ORG 0x50

start:                             //Programmbeginn
                                //Initialisierung des Stacks
    ldi r16, HIGH(RAMEND)
    out sph, r16
    ldi r16, LOW(RAMEND)
    out spl, r16

                                //PORT A als Ausgang intialisieren
    ldi r17,0b00000011             //setze Register mit einem Ausgang
    out DDRA,r17                   //definiere DDRA(0) als output
    out PORTA,r17                  //schalte LED ein

                                // lade Startwerte in Zähler
    ldi r19,cntStart
    STS cntAddr, r19

                                // Counter 0 auf enable setzen (GIE)
    ldi r20,0b00000001
    STS TIMSK0, r20
                                // Setze das interrupt bit im Statusregister
    SEI                            //Es darf Unterbrechungen geben
                                //Initialisiere den Timer 0
    ldi r21, 0b00000000
    ldi r22, 0b00000101
    ldi r23, toStart
    out TCCR0A, r21
    out TCCR0B, r22
    out TCNT0, r23
                                //Stelle sleep mode ein
    ldi r27, 0b00000001            //idle mode
    out SMCR, r27

loop:
    sleep                          //CPU geht schlafen
        cli                                 //disable interrupts globally
        ldi r25,0x02                        //toggle output bit 1
        in r26,PINA
        eor r26,r25
        out PORTA,r26
        sei                                 //enable interrupts globally
    rjmp loop
```

```
T0ISR:
                            //Register und Statusregister sichern
    in r24, SREG
    push r24
    push r18
    push r19


                            // Eigentlicher Code
    lds r18,cntAddr            //Zählvariable in r18 laden
    dec r18                    //1 abzählen
    STS cntAddr,r18
    cpi r18,0
    brne ende                  //springe zurck wenn noch nicht 25

    ldi r19,cntStart
    STS cntAddr, r19
    com r17                    //ein-> aus      aus-> ein
    out PORTA,r17              //schalte LED um

ende:
    out TCNT0, r23             //Timer Count Register: Timer zurcksetzen


                            //Register und Statusregister laden
    pop r19
    pop r18
    pop r24
    out SREG, r24

RETI
```

Aufgabe 2 - 16-Bit Addition

```
  start:                          //300+150
                                 //erstes Register r17:r16
    ldi r17,0b00000001            //vorne, highbyte
    ldi r16,0b00101100            //hinten, lowbyte
                                 //0000'0001'0010'1100
                                 //zweites Register r18:r19
    ldi r19,0b00000000            //vorne, highbyte
    ldi r18,0b10010110            //hinten, lowbyte
                                 //0000'0000'1001'0110


add16:
    add r16, r18                  //zwei 16-bit Opperanden addieren
    brcc gump                     //springe zu gump,
                                 wenn carry nicht gesetzt (cleared carry)
                                  //addiere 1 dazu, da carry gesetzt wurde
    ldi r18,1
    add r17,r18                   //addiere 1 dazu

gump:
    add r17, r19                  //addiere highbyte

                                 //es sollte nun 450 = 0000 0001 1100 0010 im r17:r16 sein
```

Aufgabe 3 - schnellere 16-Bit Addition Variante mit adc

```
add16:
     //zwei 16-bit Opperanden addieren
    add r16,r18      //lowbyte addieren und carry setzen (0/1)
    adc r17,r19      //highbyte addieren mit carry
```

Aufgabe 4 - Multiplikation

```
start:
    ldi r16,0b00101000            //X im Register 16
    ldi r18,0b01001001            //Y im Register 18

mult16:
                                  //P r17:r16
    ldi r17,0b00000000            //highbyte von P mit Nullen fllen

    ldi r20,7                     //Zähler (8 durchläufe)
    loop:
    SBRC r16,0                    //springe wenn Bit 0 im Register r16 cleared
    add r17,r18                   //addition
                                  //schieben
    clc                           //lpan>sche carry
    ror r17                       //schiebe das carry von links rein
    ror r16                       //schiebe das carry von links rein

    dec r20                       //Zähler -1
    brne loop                     //springe zurck wenn noch nicht null

    ror r17                       //schiebe das carry von links rein
    ror r16                       //schiebe das carry von links rein
```

Übung 7 - C-Programmierung

Aufgabe 1 - Variable setzen und subtrahieren

```
/*
 * U7A1.c
 *
 * Created: 04.04.2012 16:00:52
 *  Author: mmeschenmoser und kilkow
 */

#include <avr/io.h>

int main(void)
{
    char a = 30;

    while(a > 10)
    {
        a-=5;
    }

    while (1);                          //Endlosschleife

    return 0;
}
```

Aufgabe 2 - Taster -> LED mit Polling (immer wieder abfragen)

```
int main(void)
{
    DDRA |= 0x0F;                       //Ausgang 0000 1111

    while(1)
    {
                                    //Setze die LED (3-0) gemäss den Schaltern (7-4) =
Schiebe um 4
        PORTA = ~(PINB>>4);
    }

    return 0;
}
```

Aufgabe 3 - PortB Pin6 -> PortA Pin1

```
int main(void)
{
    DDRA |= 0x02;                       //Ausgang 0000 0010

    while(1)
    {
        if (PINB & 0x40)                //PINB AND 0100 0000 = 0000 0000 bei 0
            PORTA = 0x00;               //Schalter nicht gedrückt
        else
            PORTA = 0x02;               //Schalter gedrückt
    }

    return 0;
}
```

Aufgabe 4 - Nach fünfmaligem Drücken wechseln

```
int main(void)
{
    DDRA |= 0x02;                   //Ausgang 0000 0010
    char zaehler = 5;
    char zustand = 0;               //Zustand: 0=nicht gedrückt, 1=gedrückt

    while(1)
    {
        if (PINB & 0x40)            //PINB AND 0100 0000 = 0000 0000 bei 0
            if (zustand==1){        //Schalter nicht gedrückt
                zaehler-=1;
                zustand = 0;
            }else{;}
        else
            if (zustand==0){        //Schalter gedrückt
                zaehler-=1;
                zustand = 1;
                }else{;}

        if (zaehler==0) {
            zaehler = 5;
            PORTA = PINA^0x02;
        }
    }

    return 0;
}
```

Aufgabe 5 - Blinklicht mit Delay

```
void delay250ms()
{
    int i;
    for(i = 0; i<30000;i++);
}

int main(void)
{
    DDRA |= 0x01;                       //Ausgang 0000 0001

    while(1)
    {
        delay250ms();
        PORTA = PINA^0x01;              //toggle
    }
}
```

Aufgabe 6 - Blinklicht mit Interrupt

```
#include <avr/io.h>
#include <avr/interrupt.h>

int zaehler = 25;
```

```
ISR(TIMER0_OVF_vect)              //once per 10ms
{
    zaehler--;
    if (zaehler == 0){
        PORTA = PINA^0x01;        //toggle
        zaehler =25;
    }
}

int main(void)
{
    DDRA |= 0x01;                 //Ausgang 0000 0001
    sei();                        //damit man unterbrechen kann
    TCNT0 = 178;                  // Startwert Timer0 auf 178
    TCCR0A = 0x00;
    TCCR0B = 0x05;
    TIMSK0 = 0x01;
    while(1){}
    return 0;
}
```

Aufgabe 7 - Ein- und Ausschalten mit Interrupt auf PB4

```
int zaehler = 10;
char bool = 0;

ISR(TIMER0_OVF_vect)              //once per 10ms
{
    sei();    // damit Pin-Change-Interrupt auch in dieser Schleife wirkt!!
    zaehler--;
    if (zaehler == 0){
        PORTA = PINA^0x01;        //toggle
        zaehler =10;
    }
}

ISR(PCINT0_vect){                 //Pin-change-interrupt
    static char bool = 0;
    if (bool){                    //Zustand: 0 = Interrupt bit nicht gesetzt
        bool = 0;
        TIMSK0 = 0x00;;           //Timer0 ausschalten
    }else{
        bool = 1;
        TIMSK0 = 0x01;;           //Timer0 einschalten
    }
}

int main(void)
{
    DDRA |= 0x01;                 //Ausgang 0000 0001
    TCNT0 = 178;                  // Startwert Timer0 auf 178
    TCCR0A = 0x00;
    TCCR0B = 0x05;
    PCICR = 0x01;                 // PinChangeInterrupt
    PCMSK0 = 0x10;
    sei();                        // Interrupts generell

    while(1){}
    return 0;
}
```

Aufgabe 8 - Sleep-Modus

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>

int zaehler = 10;


ISR(TIMER0_OVF_vect) //once per 10ms
{
    zaehler--;
    if (zaehler == 0){
        PORTA = PINA^0x01; //toggle
        zaehler =25;
    }
}

int main(void)
{
    DDRA |= 0x03; //Ausgang 0000 0001

    sei(); //damit man unterbrechen kann

    // Timer0 einschalten
    TCNT0 = 1; // Startwert Timer0 auf 178
    TCCR0A = 0x00;
    TCCR0B = 0x05;
    TIMSK0 = 0x01;

    // Sleep einschalten
    set_sleep_mode(SLEEP_MODE_IDLE);

    while(1){
        sleep_enable();
        PORTA = PINA^0x02;
    }
    return 0;
}
```

Aufgabe 9 - 16Bit Addition

```
int main(void)
{
// summe von 2 int
   int integer1 = 5;                  // erstes int wird gespeichert
   int integer2 = 5;                  // zweites int wird gespeichert
   int resultInt = integer1+integer2; // Addiert zwei int werte

// summe von zwei long
   long langer1 = 5;                  // erstes Long wird gespeichert
   long langer2 = 5;                  // zweites Long wird gespeichert
   long resultLong=langer1+langer2;   // Addiert zwei long werte

    while (1);
}
```

Übung 8 - UART

Aufgabe 1 - Initialisierung

```
#include <avr/io.h>

int main(void)
{
    initUART1();
    UDR1='u';                                   //Senden
    while(1) ;                                  //Endlosschleife
}

void initUART1()
{
    UBRR1 = 51;      //baud rate setzen (UBBRH und UBBRL setzen) (Buch 18-1)
    UCSR1B = (1<<RXEN1) | (1<<TXEN1);    // Empfangen und Senden einschalten
    UCSR1C = (0<<UMSEL10) | (0<<UMSEL11);     //Asynchroner USART (optional)
    UCSR1C = (0<<UPM10) | (0<<UPM11);  // Parity Mode ausschalten (optional)
    UCSR1C = (1<<UCSZ10) | (1<<UCSZ11) | (0<<UCSZ12);
}
```

Aufgabe 2 - writeChar

```
#include <avr/io.h>

int main(void)
{
    initUART1();
    while(1)
    {
        writeCharUART1('b');
    }
}
void initUART1(){}                              siehe Aufgabe 1 !!
void writeCharUART1(const char ch){
    while( !(UCSR1A & (1<<UDRE1))) //solange USART Data Register nicht leer
    { }
    UDR1 = ch;
}
```

Aufgabe 3 - write String

```
void initUART1(){ }                             siehe Aufgabe 1 !!
void writeCharUART1(const char ch){ }     siehe Aufgabe 1 !!
void writeStringUART1(const char str[]){
    int i = 0;
    char a = '1';
    while(a){
        a = str[i];
        writeCharUART1(a);
        i++;
    }
}

int main(void)
{
    initUART1();
    char str[16];
    char val = 23;
    writeStringUART1("T1 = 23\n\r");
    while(1) { }
}
```

Aufgabe 4 - int 2 str

```
#include <avr/io.h>

void initUART1() //initialisieren
{
    UBRR1 = 51; //baud rate setzen (UBBRH und UBBRL setzen) (Buch 18-1)
    UCSR1B = (1<<RXEN1) | (1<<TXEN1); // Empfangen und Senden einschalten
```

```
    UCSR1C = (0<<UMSEL10) | (0<<UMSEL11); //Asynchroner USART (optional)
    UCSR1C = (0<<UPM10) | (0<<UPM11); // Parity Mode ausschalten (optional)
    UCSR1C = (1<<UCSZ10) | (1<<UCSZ11) | (0<<UCSZ12);
}

void writeCharUART1(const char ch)
{
    while( ! (UCSR1A &(1<<UDRE1))) //solange USART Data Register nicht leer
    { }
    UDR1 = ch;
}

void writeStringUART1(const char str[])
{
    int i = 0;
    char a = '1';

    while(a)
    {
        a = str[i];
        writeCharUART1(a);
        i++;
    }
}

void int2str(char str[], int i, unsigned char length)
//wandle einen int in einen str um. (str = Ausgabe)
{
    str[length] = 0; //schreibe ein Null ganz rechts
    char neg = 0;
    if (i<0) {neg = 1; i = i*(-1);} //falls negativ
    do
    {
        length--;
        str[length] = (i%10)+48; //schreibe aktuelles Zeichen mit Modulo
        i = i/10; //nehme n酧stes Zeichen mit Division
    }while(i);

    if(neg){ length--; str[length] = '-'; }//fge Minuszeichen ein
    while(length) //fge Leerzeichen ein
    {
        length--;
        str[length] = ' ';
    }
}

int main(void)
{
    initUART1(); //initialisiere
    char str[20];
    int2str(str,-55,5);
    writeStringUART1(str);
    while(1) { }
}
```

Übung 9 - Treiber für UART

Aufgabe 3 - Vollständiger Treiber (Musterlösung)

U9A3.c - file

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "UART1.h"          //lokale Headerfiles

// defines and declarations for transmit and receive buffers
#define txBufLen 32
#define rxBufLen 16              //Konstante (oder const ... = 16;)

struct {
    char buf[txBufLen];
    int out, len;
} txBuf;

struct {
    char buf[rxBufLen];
    int out, len;
} rxBuf;

// isr for transmitting from buffer to UART1
ISR(USART1_UDRE_vect) { // tx register empty
    if (txBuf.len > 0) {
        UDR1 = txBuf.buf[txBuf.out];
        txBuf.out = (txBuf.out + 1) % txBufLen;
        txBuf.len--;
    } else UCSR1B &= ~0x20; // switch of tx interrupts
}

// isr for receiving from UART1 to buffer
ISR(USART1_RX_vect) {   // rx register full
    if (rxBuf.len < rxBufLen - 1) {
        rxBuf.buf[(rxBuf.out + rxBuf.len) % rxBufLen] = UDR1;
        rxBuf.len++;
    }
}

// transmits a single char over UART1, nonblocking
void writeCharUART1(const char ch) {
    char status = SREG;     // make atomic
    cli();
    if (txBuf.len < txBufLen) {
        txBuf.buf[(txBuf.out + txBuf.len) % txBufLen] = ch;
        txBuf.len++;
    }
    UCSR1B |= 0x20; // reenable tx interrupts
    SREG = status;
}

// transmits a string over UART1
void writeStringUART1(const char str[]){
    int i = 0;
    while (str[i] != 0) {
        writeCharUART1(str[i++]);
    }
}
```

```
// reads all characters from buffer and returns them as a string, nonblocking
void readStringUART1(char str[]) {
    char status = SREG;     // make atomic
    cli();
    while (rxBuf.len > 0) {
        *str++ = rxBuf.buf[rxBuf.out];
        rxBuf.out = (rxBuf.out + 1) % rxBufLen;
        rxBuf.len--;
    }
    SREG = status;
    *str = 0;               // terminate the string
}

// initialize UART1
void initUART1() {
    UBRR1 = 51;     // 9600 baud @ 8MHz
    UCSR1B = 0xb8;  // enable rx and tx, enable rx and tx interrupts
    UCSR1C = 0x06;  // no parity, 1 stop, 8 bit frame
}
```

U9A2.h - file

```
void writeStringUART1(const char []);
void writeCharUART1(const char ch);
void readStringUART1(char []);
void initUART1();
```

Test.c - file

```
#include "UART1.h";
#include <avr/interrupt.h>

int main(void)
{
    initUART1(); //initialisiere
    sei();  //global interrupt bit setzen

    char strrx[20];
    strrx[18] = 25;
    strrx[19]=0;

    while(1)
    {
        readStringUART1(strrx);
        writeStringUART1(strrx);
    }
}
```

Übung 11 - Temperaturmessgerät Version 1, Ablaufsteuerung und ADC

Aufgabe 1 - Timer3 und Sleep-Modus

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include "UART1.h"

void initTimer3_1s()                        //Timer 3 auf 1s initialisieren
{
    TCCR3A = 0;
    TCCR3B = 0;
    TCCR3B |= (1<<WGM32);                   //CTC Mode
    TCCR3B |=(1<<CS32);                     //prescaler = 256
    TIMSK3 |= (1<<OCIE3A);                  //OC Interrupt enable

    OCR3A=31249;                            //Interrupt Takt = 1sec
}

ISR(TIMER3_COMPA_vect)                      //
{

}

int main(void)
{
    set_sleep_mode(SLEEP_MODE_IDLE);        // Sleep einschalten
    initTimer3_1s();
    initUART1();
    sei();                                  //set global interrupt Bit

    while(1)
    {
        writeCharUART1('a');
        sleep_mode();
    }
}
```

Aufgabe 2&3 - String - Kombination mit ADC

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include "UART1.h"

int go = 0;
short val = 0;
char str[20];

void initTimer3_1s() //Timer 3 auf 1s initialisieren
{
    TCCR3A = 0;
    TCCR3B = 0;
    TCCR3B |= (1<<WGM32);               //CTC Mode
    TCCR3B |=(1<<CS32);                 //prescaler = 256
    TIMSK3 |= (1<<OCIE3A);              //OC Interrupt enable

    OCR3A=31249;                        //Interrupt Takt = 1sec
}
```

```
void initADC()
{
    ADCSRA |= (1<<ADIE);                //interrupt einschalten
    ADCSRA |= (1<<ADPS0);               //prescailor auf 128 setzen
    ADCSRA |= (1<<ADPS1);
    ADCSRA |= (1<<ADPS2);
    ADMUX |= (1<<REFS0);                //setze Referenzspannung auf AV_cc
    ADMUX |= (0<<REFS1);
}

ISR(ADC_vect) //ISR vom ADC
{
}

ISR(TIMER3_COMPA_vect) //ISR vom Timer
{
    go = 1;
}

int main(void)
{
    //set_sleep_mode(SLEEP_MODE_IDLE);  // Sleep einschalten
    initTimer3_1s();
    initUART1();
    initADC();
    sei();                              //set global interrupt Bit
    str[0]=0;

    while(1)
    {
        set_sleep_mode(SLEEP_MODE_IDLE);
        ADCSRA &= ~0X80;                //disable ADC
        while(!go) sleep_mode();
        go = 0;

        ADCSRA |= 0X80; //enable ADC
        set_sleep_mode(SLEEP_MODE_ADC);
        sleep_mode();

        val = ADCW;
        sprintf(str,"ADC0 = %4d\r\n",val);
        writeStringUART1(str);
    }
}
```

Übung 12 - Temperaturmessgerät Version 2, Interpolation

Aufgabe 3&4 - Lookup Tabelle - Interpolation

```
/*
 * U12A4 Interpolation.c
 *
 * Created: 23.05.2012 16:35:35
 *  Author: mmeschenmoser
 */


#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include "UART1.h"

int go = 0;
short val = 0;
char str[40];
short ADC_Value[13]={67,90,119,157,201,253,312,376,443,512,578,641,699};
short i;
short temp;


void initTimer3_1s() //Timer 3 auf 1s initialisieren
{
    TCCR3A = 0;
    TCCR3B = 0;
    TCCR3B |= (1<<WGM32);               //CTC Mode
    TCCR3B |=(1<<CS32);                 //prescaler = 256
    TIMSK3 |= (1<<OCIE3A);              //OC Interrupt enable

    OCR3A=31249;                        //Interrupt Takt = 1sec
}

void initADC()
{
    ADCSRA |= (1<<ADIE);                //interrupt einschalten
    ADCSRA |= (1<<ADPS0);               //prescailor auf 128 setzen
    ADCSRA |= (1<<ADPS1);
    ADCSRA |= (1<<ADPS2);
    ADMUX |= (1<<REFS0);                //setze Referenzspannung auf AV_cc
    ADMUX |= (0<<REFS1);
}

ISR(ADC_vect) //ISR vom ADC
{

}

ISR(TIMER3_COMPA_vect) //ISR vom Timer
{
    go = 1;
}
```

```
int main(void)
{
    //set_sleep_mode(SLEEP_MODE_IDLE);  // Sleep einschalten
    initTimer3_1s();
    initUART1();
    initADC();
    sei();                              //set global interrupt Bit
    str[0]=0;

    while(1)
    {
        set_sleep_mode(SLEEP_MODE_IDLE);
        ADCSRA &= ~0X80;                //disable ADC
        while(!go) sleep_mode();
        go = 0;

        ADCSRA |= 0X80; //enable ADC
        set_sleep_mode(SLEEP_MODE_ADC);
        sleep_mode();

        val = ADCW;

        //calculate index in the ADC_Value_Table
        i=0;
        while(val>ADC_Value[i]){
            i++;
        }
        i--;

        if(i>=12)
        {
            sprintf(str,"Fehler!! Zu hohe Temperatur\r\n");
        } else {
            //calculate Temperature
            temp = (-20+5*i)+5*(val-ADC_Value[i])/(ADC_Value[i+1]-ADC_Value[i]);
            sprintf(str,"Temperatur gerundet = %5dC\r\n",temp);
        }
        writeStringUART1(str);
    }
}
```

Übung 13 - Temperaturmessgerät Endversion

Aufgabe 2 - mit Transmitter und Receiver

```
...
val = ADCW;

// search in LUT
for (i=0; (i < 12) && (val > (table[i])); i++);
i--;
tx = -20 + 5*i;

// interpolate
temp = tx + 5*((table[i])-val)/((table[i])-(table[i+1]));

// send result to rf
sendTempRf(temp);

// output to UART1
sprintf(str, "local temp: = %4d%cC\t", temp, 176);
writeStringUART1(str);
temp = receiveTempRf();
sprintf(str, "remote temp: = %4d%cC\r\n", temp, 176);
writeStringUART1(str);
...
```

Übung 14 - Abschluss des Projektes

Aufgabe 1 - CRC-Check

```
#include <avr/io.h>

int main(void)
{
    char temp = 22;

    int crc16 = calcCRC16(&temp, 1);
    //crc16 msste null sein.
}

int calcCRC16(char str[], int len){
    int rem = 0;

    for (int i = 0; i<len; i++)
    {
        rem = rem ^ (str[i]<<8);
        for(int j = 1; j<=8;j++){
            if ((rem & 0x8000) == 0x8000)
            { //gleich -> mst = 1
                rem = (rem<<1) ^ 0x1021;
            } else { //ungleich -> mst = 0
                rem = (rem<<1);
            }
        }
    }
    return rem;
}
```