Vorgaben für die Softwareentwicklung

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

Vorgaben für die Softwareentwicklung

Codierung und Programmierung

Generelles

- Es empfiehlt sich für alle definierten und deklarierten Namen die Englische Sprache zu verwenden. Für Variablen, Funktionen und Kommentare darf die Deutsche Sprache genutzt werden.
- Eine detailliertere Liste ist im Embedded System development Coding Reference Guide zu finden.
- Neben der Struktur beim Programmieren zählt auch eine interessante und schöne Umsetzung für den Nutzer in die Bewertung.

Kommentare

- Stellen Sie Ihrem Programm einen beschreibenden Kommentar voran.
- Dieser sollte in der Form sein, wie die Beschreibung in den Übungsprogrammen.
- Beschreiben Sie darin, ob
  - weitere c- oder h-files eingebunden müssen
  - Jumper gesetzt / geöffnet werden müssen
  - spezielle Hardware genutzt werden muss

Beschreibung

```
/*
================================================================================

mein Programm:     Programmbeispiel für mich
=============    ============================================================

Dateiname:       MEXLE_MeinProgramm.c

Autoren:         Max Integer (Hochschule Heilbronn)

Version:         0.5 vom 29.02.2019

Hardware:        z.B. MiniMEXLE Ver. 3.0 (oder angepasste Version 2.0) oder
MEXLE2020
                 AVR-USB-PROGI Ver. 2.0

Software:        Atmel Studio Ver. 7.0.1417

Funktion:        Diese Programm sol eine einfaches Beispiel der Ein- und
Ausgabe am MiniMEXLE sein.
                 Es wird ein einfacher Zähler hoch- oder heruntergezählt

Displayanzeige:        Start (fuer 2s):        Betrieb:
                     +----------------+        +----------------+
                     | Mein Programm  |        |Counter 0       |
```

```
                    |   Zaehler      |        | Up Down        |
                    +----------------+        +----------------+

Tastenfunktion:   S2:    Up    (zaehlt Counter aufwaerts. Überlauf bei 255)
                  S3:    Down    (zaehlt Counter abwaerts. Unterlauf bei 0)

Jumperstellung:   keine Auswirkung

Fuses im uC:      CKDIV8: Aus    (keine generelle Vorteilung des Takts)

Header-Files:     lcd_lib_de.h    (Library zur Ansteuerung LCD-Display Ver.
1.2)

Module:           1) get_switch_state: Schalter einlesen
                  2) set_display_values: Werte ausgeben

    1) get_switch_state:    ...
    3) set_display_values:    ...

================================================================================
=*/
```

- Es empfiehlt sich die Code-Kommentierung zeilenweise durchzuführen. Schreiben Sie dabei nicht, was im Code bereits steht, sondern was Sie mit dem Code bezwecken.

Beispiel für Code-Kommentierung

SCHLECHT	`if(i==0) output=0; // wenn i = 1, output=0`
GUT	`if(i==0) output=0; // nur für erstes Element wird der Output zurückgesetzt`

- Während der Entwicklungsphase kann es sich anbieten Code testweise auszukommentieren. Für die finale Version sollten die Kommentare aber “sauber” sein.
- Falls es alternative Werte gibt, welche optional sinnvoll sind, können diese und deren Konsequenzen in ein Kommentar gepackt werden.

Beispiel für auskommentierten Code

SCHLECHT	`...`
	`    if (i==1) output("eins");    // ToBeChanged: noch an Zähler anpassen`
	`//    if (i==3) output("null");`
	`//    if (i==4) montagA();`
	`//    if (i==5) ??;`
	`...`

GUT	`...`
	`    if (i==STARTWERT) LCDoutput(startAusgabe);   // nur bei i=1 erfolgt eine Ausgabe`
	`                          // bei den anderen Werten erübrigt sich die`
	`                          // Ausgabe, weil i <=1`
	`...`

Makros

- Nutzen Sie für die Manipulation von Bits die vorgegebenen Makros.
- Beim Erstellen von eigenen Makros sollte auf Querwirkungen geachtet werden, da ein Makro eine Codeersetzung vor dem Compiler durchführt.

Beispiel für die vordefinierten Makros

SCHLECHT

```
#define    TWICE(x)    2*x // Port-Bit Setzen

void main()
{
    ...
    PORTD = PORTD & 64;
    alterWert = 4;
    neuerWert = TWICE(alterWert+2); // durch das Makro wird der Code nur ersetzt
                        // es ergibt sich also neuerWert = 2*alterWert+2
                        // der Compiler wertet dieses über Punkt vor Strich aus
                        // es ergibt sich also 2*4+2=10 und nicht 2*(4+2)=12
    ...
}
```

GUT

```
// Makros

#define SET_BIT(PORT, BIT)    ((PORT) |=  (1 <<(BIT))) // Port-Bit Setzen
#define CLR_BIT(PORT, BIT)    ((PORT) &= ~(1 <<(BIT))) // Port-Bit Loeschen
#define TGL_BIT(PORT, BIT)    ((PORT) ^=  (1 <<(BIT))) // Port-Bit Toggeln

void main()
{
    ...
    CLR_BIT(PORTD, ENABLE);
    alterWert = 4;
    neuerWert = 2*(alterWert+2);
    ...
}
```

Konstanten

- Konstanten per `#define` sollten z.B. für die feste Größe von Arrays verwendet werden. Sie können (bzw. werden) auch für hardwarenahe Werte, wie Portnummern, genutzt werden.
- Auch Werte in `enum` sind Konstanten. Für Konstanten mit ähnlichem Hintergrund (z.B. Tage, s.u.) sollten `enum` genutzt werden. Damit können Variablen auch als `enum`-Typ definiert werden, was den Code leserlicher macht.
- Bei Defines wird keine Typisierung überwacht. Ist dies notwendig empfiehlt es sich `const` Variablen zu nutzen.
- Konstanten per `#define` oder `enum` sind komplett in **Großbuchstaben** zu schreiben, um diese von Variablen zu unterscheiden.
- Falls Sie aus mehreren Wörtern zusammengefügt sind, sollten Sie **mit Unterstrich** getrennt

werden.

```
enum tage {MONTAG, DIENSTAG, MITTWOCH, DONNERSTAG, FREITAG, SAMSTAG,
SONNTAG};

for (enum tage aktuellerTag = MONTAG; aktuellerTag <= SONNTAG;
aktuellerTag++)
{...};
```

Beispiel für Konstanten

SCHLECHT	`// Konstanten`
	
	`#define CONST1 1.414 // Korrekturwert`
	`#define PORT1 4 // Erster Port`
	`#define gameIntro 0 // Spiel Zustand: aktuell wird das Intro gezeigt`
	`#define gameStarted 1 // Spiel Zustand: aktuell ist das Spiel gestartet`
	`#define gamePaused 2 // Spiel Zustand: aktuell ist das Spiel pausiert`
	`#define gameEnded 3 // Spiel Zustand: aktuell ist das Spiel zu Ende. Highscore etc wird angezeigt`
	
	`...`
	
	`int ZeichenAufLCD[2][16]=0;`
	
	`...`

GUT	`// Konstanten`
	
	`#define SQRT_OF_2 1.414 // Wurzel aus 2`
	`#define FIRST_PORT 4 // Erster Port für die Eingabe`
	`#define XMAX_LCD 2 // Anzahl der Zeilen`
	`#define YMAX_LCD 16 // Anzahl der Spalten`
	
	`enum gameStates{`
	`GAMESTATE_INTRO,`
	`GAMESTATE_STARTED,`
	`GAMESTATE_PAUSED,`
	`GAMESTATE_ENDED};`
	
	`...`
	
	`int ZeichenAufLCD[XMAX_LCD][YMAX_LCD]=0;`
	
	`...`

Variablen - leserlich, initialisiert und separat

- Wenn sich Werte im Programm zur Laufzeit ändern, so sollten diese als Variable angelegt werden.
- Nutzen Sie soweit es geht const Variablen für alle Werte im Programm, welche zur Laufzeit nicht mehr geändert werden. **Wichtig:** Das gilt z.B. für Grenzen von Schleifen (for(int i=0; i<iMax;i=i+1)) oder für Sonderzeichen. Bei größeren Programmen biete es sich an die const

Variablen mit in einem separaten header zu pflegen.
- Variablen beginnen mit Kleinbuchstaben.
- Falls Sie aus mehreren Wörtern zusammengefügt sind, so werden die folgenden Wörter **ohne Unterstrich** direkt angefügt, aber groß geschrieben. Dies wird auch als “BinnenMajuskel” oder “camelCase” bezeichnet.
- Vermeiden Sie zu allgemeine Namen, wie anzahl, uebergabewert oder string. Sinnvoller sind Namen, wie anzahlBuchstaben, stunden, ausgabeString. Durch die Autovervollständigung (Vorschläge unter dem eingegebenen Text) sind auch längere Namen schnell einzugeben, bzw mit Cursortasten und TAB auswählbar.
- Nutzen sie auch bei Zählvariablen aussagekräftige Namen.
- Auch kann eine zu allgemeine Deklaration kann zu Problemen führen. Schlecht ist z.B. “int a;”.
- Es bietet sich an bei der Definition bereits zu initialisieren. Gut ist also “bool a=1;”.

Beispiel für Variablen

SCHLECHT

```
// Variablen

int    spieler = 2;         // unklar, ob Konstante
int    gem_Lae_1;           // unklar, ob es ein Vorzeichen besitzt;
                            // unklar, ob es nur 8bit sein sollten
                            // unklar, welche Länge gemeint ist
char    gem_Lae_2;          // unklar, ob es ein Vorzeichen besitzt;
                            // unklar, welche Länge gemeint ist
char    wasKopie2;          // unklar,Was was ist
```

GUT

```
// Variablen

const int       maxAnzSpieler   = 2;    // Maximale Anzahl der
Spieler
uint8_t         gemesseneLaenge = 0;    // gemessene Länge in Meter
unsigned char   gemesseneBreite = 0;    // gemessene Breite in
Meter
bool            zeichenAusgabe  = 1;    // Wahrheitswert zur
Anzeige, ob
                                        // ein Zeichen ausgegeben
werden darf
```

Anweisungsblöcke und Funktionen

- Teilen Sie Ihr Projekt in sinnvolle Unterstrukturen. Diese sind meist Funktionen. Die Unterstrukturen sollten nicht zu groß werden, um die Übersichtlichkeit zu bewahren.
- Bei größeren Programmen ist auch die Aufteilung in mehrere Dateien sinnvoll, also z.b. main.c, LED.c, motorDriver.c. Dabei sollte darauf geachtet werden, dass globalen Variablen und Konstanten jeweils nur im Kontext der einzelnen Dateien genutzt werden und, dass header-Dateien angelegt werden. Das ermöglicht ein separates Testen der unterschiedlichen Dateien (z.B. mit einer Datei testLED.c, welche LED.h inlcude't).
- Nutzen Sie den Zeileneinschub den AtmelStudio automatisch anbietet.
- Für die Benennung von Funktionen bietet sich - wie bei Variablen - camelCase an. Zum leichteren Verständnis sollten die Funktionsnamen aus Objekt(e) und Verb zusammengesetzt werden (z.B. bool isI2cMessageNotSent() oder void sentI2cMessage()). Damit wird der Code besser lesbar, Z.B. `if (isI2cMessageNotSent()) sentI2cMessage()`
- Vermeiden Sie zu viele Leerzeilen.
- Stellen Sie auch jeder Funktion eine kurze Beschreibung voran. Aus dieser sollte hervorgehen, was Sinn und Zweck der Funktion ist.

Beispiel für Anweisungsblöcke

SCHLECHT

```
uint8_t unter2_neu(uint8_t Was)
{

    int a;
for(a=10;a<20;a++)

{
if(arr[a]==Was)
{
return a;
};
    };

return 0;
}
```

GUT

```
uint8_t schluesselPositionFinden(uint8_t schluessel)
/*    Das Array schluesselArray wird nach dem übergebenen Schlüssel durchsucht.
      Wird der Schlüssel gefunden, so wird die Position zurückgegeben.
      Wird der Schlüssel nicht gefunden, so wird 0 zurückgegeben.
*/
{
    for( int aktuelleSchluesselPosition=ERSTE_SCHLUESSEL_POSITION; a<=LETZTE_SCHLUESSEL_POSITION; aktuelleSchluesselPosition++)
    {                           // durchlaufe alle Schlüsselpositionen
        if(schluesselArray[aktuelleSchluesselPosition]==schluessel) return aktuelleSchluesselPosition;
                               // falls Schlüssel gefunden,
    };                           // gib die erste Position zurück
    return 0;
}
```

In diesem Beispiel wäre der Funktionsname schluesselPosition statt schluesselPositionFinden auch geeignet gewesen.
Weiterhin bietet es sich an hier auch einen Zeiger auf das Array und die Werte für erste und letzte Position als Parameter der Funktion zu übergeben, um diese flexibler anwenden zu können.

Arrays, Schleifen und Abfragen

- Es ist sinnvoll Deklaration und Definition der Schleifenvariable direkt in den if-Befehl zu packen. Also: `for(uint8_t i=0; i<MAX; i)''. Damit wird der Code etwas kompakter. * Vermeiden Sie Zugriffe auf Bereiche von Arrays, welche nicht definiert wurden. So erhält man bei einem Array ''uint8_t array[10];'' mit Zugriff auf ''array[-1]'', ''array[10]'' oder ''array[21]'' keine Werte des eigentlichen Array, sondern Werte von anderen Variablen. Ein Beschreiben dieser Bereich kann zu unerwarteten Werten in anderen Variablen führen. ++++Beispiel für Arrays, Schleifen und Abfragen |SCHLECHT|`

```
#define MAX 10

uint8_t array[MAX];
```

```
...
    for(unit8_t a = 0 ; a<=MAX ; a++)
    {                          // durchlaufe alle Schlüsselpositionen
        array[a*2] = array[a*2-1];
    };
```

| |**GUT**|

```
#define MAX 10

uint8_t array[MAX];

...
    for(unit8_t a = 1 ; a<=MAX/2 ; a++)
    {                          // durchlaufe alle notwendigen
Schlüsselpositionen
        array[a*2] = array[a*2-1];
    };
```

| ==== Programmoptimierung - kurz und übersichtlich ==== * Ziel ist ein leicht lesbarer und wartbarer Code. Halten Sie deswegen alle Funktionen schlank - auch void main(). Als Faustformel wären 100 Zeilen für eine Funktion zu groß, 20...50 Zeilen gut. * Versuchen Sie sinnvolle Unterfunktionen zu programmieren. Trennen Sie Eingabe, Verarbeitung und Ausgabe. * Überlegen Sie sich immer wenn Sie im Code Copy-Paste nutzen, warum dies nicht als Unterfunktion lösbar ist. ++++Beispiel für ähnliche Zeilen |**SCHLECHT**|

```
...
    temp = hunderter;
    lcd_goto(1,0);
    lcd_putc(0x30 + temp%10);

    temp = zehner;
    lcd_goto(1,1);
    lcd_putc(0x30 + temp%10);

    temp = einser;
    lcd_goto(1,2);
    lcd_putc(0x30 + temp%10);
...
```

| |**GUT**|

```
#define ASCII_ZERO 0x30
#define MOD_TEN 10

#define DISP_LINE1  1

#define DISP_POS0  0
#define DISP_POS1  1
#define DISP_POS2  2
```

```
...
void printDecimalDigit(int x, int y, int DigitToBePrint)
{
    lcd_goto(x,y);
    lcd_putc(ASCII_ZERO + DigitToBePrint%MOD_TEN );
};
...
    printDecimalDigit(DISP_LINE1, DISP_POS0, hunderter);
    printDecimalDigit(DISP_LINE1, DISP_POS1, zehner);
    printDecimalDigit(DISP_LINE1, DISP_POS2, einer);
...
```

| * Prüfen Sie, ob aufeinanderfolgende, ähnliche if-Anweisungen sich nicht direkt über Arrays lösen lassen (Beispiel Verzweigungen 1). Wählen Sie bei Verzweigungen statt vielen if-Anweisungen mit ähnlichen Bedingungen Switch-Case-Anweisungen (Beispiel Verzweigungen 2). Falls diese nicht möglich sind, eine For-Schleife und Arrays (Beispiel Verzweigungen 3). * Auch Switch case kann auch durch verschiedene Vereinfachungen noch verbessert werden. ++++Beispiel für Verzweigungen 1 - Umwandlung in Array |**SCHLECHT**|

```
...
    if (i==0) output("null");    // wenn 0 dann null
    if (i==1) output("eins");    // wenn 1 dann eins
    if (i==2) output("zwei");    // wenn 2 dann zwei
    if (i==3) output("drei");    // wenn 3 dann drei
    if (i==4) output("vier");    // wenn 4 dann vier
    if (i==5) output("fünf");    // wenn 5 dann fünf
...
```

| |**GUT**|

```
#define MAX_ANZ_AUSGABE 6
#define MAX_ZEICHEN_AUSGABE 4

...
    char AusgabeZahl[MAX_ANZ_AUSGABE][MAX_ZEICHEN_AUSGABE] = {
            "null",
            "eins",
            "zwei",
            "drei",
            "vier",
            "fünf"
        };
...
    outputToLCD(AusgabeZahl[i]);
...
```

| ++++Beispiel für Verzweigungen 2 - Umwandlung in Switch-Case |**SCHLECHT**|

```
...
    if (i==0) doZero;    // wenn 0 dann null
    if (i==1) doOne;     // wenn 1 dann eins
```

```
    if (i==2) doTwo;     // wenn 2 dann zwei
    if (i==3) doThree;   // wenn 3 dann drei
    if (i==4) doFour;    // wenn 4 dann vier
    if (i==5) doFive;    // wenn 5 dann fünf
...
```

| |**GUT**|

```
...
    switch(i) {
        case 1: doOne;   break;    // könnte alternativ auch
        case 2: doTwo;   break;    // über Pointer auf Funktionen
        case 3: doThree; break;    // wie Beispiel 3 gelöst
        case 4: doFour;  break;    // werden
        case 5: doFive;  break;
        default: break;
    };
...
```

| ++++Beispiel für Verzweigungen 3 - Optimierung von Switch-Case |**OPTIMIERBAR**|

```
...
    switch(i) {
        case 1: doOne;   break;                // - die ersten beiden
Cases
        case 2: doOne;   break;                //   haben die gleichen
Konsequenzen
        case 3: doThree;doFour;doFive; break;  // - hier werden - je
nach Zahl -
        case 4: doFour;doFive;         break;  //   nach und nach
aufeinanderfolgende
        case 5: doFive;                break;  //   Funktionen
aufgerufen
        case 6: doOne;   break;                // - hier haben mehrere
Cases
        case 7: doOne;   break;                //   haben die gleichen
Konsequenzen
        case 8: doOne;   break;                //
        case 9: doOne;   break;                //
        default: break;
    };
...
```

| |**BESSER**|

```
...
    switch(i) {
        case 1, 2, 6...9 : doOne;   break;    // Einzelne Zahlen können
mit Komma getrennt und Gruppen mit '...' kombiniert werden
        case 3:            doThree;           // ohne Break werden alle
folgenden Befehle bis zum nächsten Break ausgeführt
```

```
        case 4:             doFour;
        case 5:             doFive;
        default:            break;
    };
...
```

| ++++Beispiel für Verzweigungen 4 - Umwandlung in For-Next |**SCHLECHT**|

```
...
    if (( 0<i) && (i<= 7)) j=j+2;
    if (( 7<i) && (i<=12)) j=j+5;
    if ((12<i) && (i<=20)) j=j+3;
    if ((20<i) && (i<=22)) j=j+10;
    if ((22<i) && (i<=60)) j=j+7;
    if ((60<i) && (i<=85)) j=j+1;
...
```

| |**GUT**|

```
...
    int maxSteps             = 6;
    int Grenze[maxSteps+1]   = { 0, 7,12,20,22,60,85};
    int jSummand[maxSteps]   = { 2, 5, 3,10, 7, 1};

    for(int steps; steps<maxSteps+1; steps++) {
        if( (Grenze[steps] < i) && (i <= Grenze[steps+1]) ) j = j +
jSummand[steps];
    };
...
```

| * Falls Sie if-Ausdrücke nutzen, für welche vorherige Fälle nicht gelten, so überprüfen Sie folgende Punkte. Wenn die if-Ausdrücke ausschließlich gegenseitig ausschließende Bedingungen beinhalten, so nutzen Sie "else if" (Beispiel Verzweigungen 4). Falls unabhängig von den Bedingungen Anfangs- oder Endanweisungen immer ausgeführt werden, so sollten diese nicht im if-Ausdruck stehen (Beispiel Verzweigungen 5). ++++Beispiel für Verzweigungen 4 - Verwenden von Else if |**SCHLECHT**|

```
...
    if (( 0<i) && (i<= 7)) j=j+2;
    if (( 7<i) && (i<=12)) {
        j=j+5;
        DoOne;
    }
    if ((12<i) && (i<=20)) j=j+3;
...
```

| |**GUT**|

```
...
    if      (( 0<i) && (i<= 7)) { j = j + 2;}
```

```
    else if (( 7<i) && (i<=12)) {
        j=j+5;
        DoOne;
    }
    else if ((12<i) && (i<=20)) { j = j + 3;};
...
```

| ++++Beispiel für Verzweigungen 5 - Reduzieren der Anweisungen |**SCHLECHT**|

```
...
    if      (i<=7) {
        j=j+2;
        DoOne;
    }
    else if (( 7<i) && (i<=12)) {
        j=j+5;
        DoZero;
        DoOne;
    }
    else if (12<i){
        j=j+3;
        DoZero;
        DoOne;
    };
...
```

| |**GUT**|
noch leserlich:

```
...
    if      (i<=7) {
        j=j+2;
    }
    else if (( 7<i) && (i<=12)) {
        j=j+5;
        DoZero;
    }
    else if (12<i){
        j=j+3;
        DoZero;
    };
    DoOne;
...
```

auch möglich, aber etwas schwerer leserlich:

```
...
    if           (i<=7)              j=j+2;
    else{if     (( 7<i) && (i<=12))  j=j+5;
        else if  (12<i)              j=j+3;
        DoZero;
```

```
    };
    DoOne;
...
```

| * Nutzen Sie im main() immer eine Endlosschleife, um an den Anfang zurückzukehren. Bitte verwenden Sie dazu nicht den Aufruf von main() in main()! Der Mikrocontroller legt dabei jedesmal neu Rücksprungadresse und Variablenzustände im Speicher ab und füllt diesen so auf. Korrekt wäre die Verwendung einer Endlosschleife. * Verwenden Sie nie den Goto-Befehl. Wird durch diesen eine Schleifenende u.ä. übersprungen, so werden die Speicherbereiche für die nur dort verwendeten Variable nicht freigegeben. * Wenn Sie aus verschachtelten Schleifen zurückkehren wollen, sollten Sie break und ein Flag nutzen. ++++Beispiel für Schleifen 1 - main() |**SCHLECHT**|

```
void main()
{
    initAll;
    while(1){            // es wäre auch for(;;){} möglich
        Eingabe;
        Verarbeitung;
        if (CancelButton==1) main;
        Ausgabe;
    }
}
```

| |**GUT**|

```
void main()
{
    initOneTimeFunctions;

    while(1){                          // äußere Endlos-Schleife
        initOtherFunctions;
        CancelButton = 0;
        while(!CancelButton){          // innere Schleife mit
Abbruchbedingung
            Eingabe;
            Verarbeitung;
            if (!CancelButton) Ausgabe;
        };
    }
}
```

| ++++Beispiel für Schleifen 2 - Abbrechen von verschachtelten Schleifen |**SCHLECHT**|

```
    for(int xpos=0;xpos<10;xpos++){
        initYPos;
        for(int ypos=0;ypos<20;ypos++){
            Eingabe;
            Verarbeitung;
            if (CancelButton) goto Abbruch;
        };
```

```
    }
Abbruch:
...
```

| |**GUT**|

```
    int xposMax=10, yposMax=20;

    for(int xpos=0 ; xpos<xposMax ; xpos++){
        initYPos;
        for(int ypos=0 ; ypos<yposMax ; ypos++){
            Eingabe;
            Verarbeitung;
            if (CancelButton) break;    // bricht nur die ypos-Schleife
ab!
        };
        if (CancelButton) break;        // bricht die xpos-Schleife ab
                                        // CancelButton ist hier ein
Flag
    }
...
```

Beachten Sie, dass in diesem Fall CancelButton eine Variable sein muss und sich zwischen den beiden if-Bedingungen nicht ändern darf (z.B. durch Interrupts). | * _delay_ms() und _delay_us() sind zu 99% nicht notwendig. Verwenden Sie stattdessen Interrupts, bzw. Timer. Es können z.B. durch Interrupts Takte angelegt werden: takt10ms, takt100ms, takt1s. Diese können dann im main() Verzweigungen in einer Zustandsmaschine auslösen. * Wenn Sie Zahlen in Variablen speichern und diese auch mathematisch weiterverwenden, so wandeln Sie diese Variable erst bei der Ausgabe in das ASCII-Format um. ++++Beispiel für Variablen mit Ausgabe |**SCHLECHT**|

```
...
    Zahlenwert = (ADC_Wert/10)%10 + 0x30;
    Zahlenwerte[i] = Zahlenwert;
    Flaeche = (Zahlenwert - 0x30 ) * Breite;
    LCD_putc(Zahlenwerte[i]);
...
```

| |**GUT**|

```
#define ASCII_ZERO 0x30
#define MOD_TEN 10

...
    Laenge     = (ADC_Wert/MOD_TEN)%MOD_TEN;
    Laengen[i] = Laenge;
    Flaeche    = Laenge * Breite;
    LCD_putc(Laengen[i]+ ASCII_ZERO);
...
```

| ++

Bewertung

Zur Bewertung lege ich diese Checkliste (xls-File) als Maßstab an.