

Vorgaben für die Softwareentwicklung

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

Allgemeine Tipps	2
Dokumentation	2
Lastenheft	2
Offene Punkte Liste	2
Modulblockbild/Blockschaltbild	3
Codierung und Programmierung	3
Generelles	3
Kommentare	3
Makros	5
Konstanten	6
Variablen - leserlich, initialisiert und separat	7
Anweisungsblöcke und Funktionen	7
Programmoptimierung - kurz und übersichtlich	8
Bewertung	13
abzugebende Work Products	13
Bewertung	14

Allgemeine Tipps

- Wenn Sie erst eine Woche vor der Abgabe mit dem Programmieren beginnen, wird es in der Regel eng. Besonders, wenn es Unklarheiten und Nachfragen gibt.
- Versuchen Sie Spaß am Programmieren zu finden. Es ist wie Puzzeln, nur dass Sie selbst das Bild bestimmen!

Dokumentation

Lastenheft

In realen Projekten würden Sie zunächst ein Lastenheft erhalten, dass Sie in ein Pflichtenheft, Systembeschreibung, Hardware- und Software-Systembeschreibung herunterbrechen, wobei parallel dazu die Systemtests und Hardware-/Softwaresystemtests beschrieben werden.

Dieses Vorgehen ist für die Arbeiten in EST zu umfänglich und wird wie folgt reduziert. Die Angaben auf der Wikiseite Ihres Projekts, sowie die während der Projektgespräche dienen als Lastenheft. Daraus ist ein Pflichtenheft zu erstellen, welches die Projektziele in obligatorische ("Muss"), gewünschte ("Soll") und fakultative ("Kann") Aufgaben untergliedert. Zusätzlich ist es auch möglich Projektziele explizit auszuschließen ("Nicht benötigt").

Um das Pflichtenheft gut zu strukturieren, bietet sich folgende Vorlage an:

Wiki-Format

h1. Pflichtenheft

_.	Nr	_.	Projektziel	_.	Beschreibung	_.	Status	_.	Priorität
	1		1. Teilaspekt des Zielzustands		Hier sollte eine kurze Beschreibung stehen		offen		Muss
	2		2. Teilaspekt des Zielzustands		Falls notwendig, kann auch auf eine Fußnote 1) referenziert werden		in Bearbeitung		Soll
	3		weiterer Teilaspekt des Zielzustands		...		erledigt		Kann
	4		-noch ein Teilaspekt des Zielzustands-		...		nicht mehr relevant		ausgenommen

1) Diese Fußnote dient nur dem Beispiel.

(Alternativ finden Sie eine Vorlage für das Pflichtenheft in Excel hier: [opl_ph.xlsx](#))

Offene Punkte Liste

Als weitere Aufgabe sollen Sie eine Offene-Punkte-Liste (OPL) anlegen und führen. Darin sollten wichtige Informationen und Aufgaben (z.B. aus Projektgesprächen) zusammengefasst sein. Ziel ist dabei nicht zwangsläufig, dass am Ende Ihres Projekts alle Punkte abgearbeitet sind. Die OPL kann auch Punkte enthalten, welche zukünftig noch bearbeitet werden müssen oder Ideen, welche noch umgesetzt werden könnten. Sie soll auch dazu dienen, dass andere - wie z.B. Ihre Nachfolger - Ihre Entscheidungen verstehen können.

Die Offene Punkte Liste sollte bevorzugt über das Ticketsystem von Redmine geführt werden.

Modulblockbild/Blockschaltbild

Für die Software ist ein Modulblockbild/Blockschaltbild beizufügen. Dies soll die Unterfunktionen und ihre gerichteten Schnittstellen zueinander (z.B. globale Variablen, Parameter) darstellen.

Ich würde Sie bitten aus Gründen der Wiederverwendbarkeit dafür die Vorlage in Ihrem Projekt im Wiki nutzen ([hier](#)). Dort ist dies unter Softwareübersicht zu finden. Auch wenn Sie keine State Machine verwenden, bietet es sich an in ähnlicher Weise die Zusammenhänge der einzelnen (Unter)funktionen zu beschreiben.

~~Ich würde Sie bitten aus Gründen der Wiederverwendbarkeit dafür die Homepage draw.io zu nutzen. Eine geeignete Vorlage finden Sie in den ILIAS Kursen [\(134142\) Labor Elektronik](#) bzw. [\(134274\) Elektronische Systeme](#). Falls Sie die alte Darstellung bereits genutzt haben, können Sie auch diese einbinden.~~

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.

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:          MiniMEXLE_MeinProgramm.c

Autoren:            Max Integer (Hochschule Heilbronn)

Version:            0.5 vom 29.02.2019

Hardware:           MiniMEXLE Ver. 3.0 (oder angepasste Version 2.0)
```

AVR-USB-PROGI Ver. 2.0

Software: Atmel Studio Ver. 7.0.1417

Funktion: Diese Programm soll 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 Verteilung des Takts)

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

Module: 1) Schalter einlesen
2) Werte ausgeben

Modul 1: ...

Modul 2: ...

=====
=*/

- 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	<pre> ... if (i==1) output("eins"); // ToBeChanged: noch an Zähler anpassen // if (i==3) output("null"); // if (i==4) montagA(); // if (i==5) ??; ... </pre>
GUT	<pre> ... if (i==STARTWERT) LCDoutput(startAusgabe); // nur bei i=1 erfolgt eine Ausgabe // bei den anderen Werten erübrigt sich die // Ausgabe, weil i <=1 ... </pre>

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	<pre> #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 ... } </pre>
-----------------	---

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.
- Für andere Konstanten empfiehlt es sich const Variablen zu nutzen. Dadurch werden auch die Typisierung überwacht.
- Konstanten per #define werden komplett in Großbuchstaben geschrieben.
- Falls Sie aus mehreren Wörtern zusammengefügt sind, sollten Sie **mit Unterstrich** getrennt werden.

Beispiel für Konstanten

SCHLECHT	<pre>// Konstanten #define CONST1 1.414 // Korrekturwert #define PORT1 4 // Erster Port ... int ZeichenAufLCD[2][16]=0; ...</pre>
GUT	<pre>// Konstanten #define SQR2_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 ... int ZeichenAufLCD[XMAX_LCD][YMAX_LCD]=0; ...</pre>

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

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 include't).
- Nutzen Sie den Zeileneinschub den AtmelStudio automatisch anbietet.

- Vermeiden Sie zu viele Leerzeilen.
- Für die Benennung von Funktionen bietet sich - wie bei Variablen - camelCase an. Zum leichten Verständnis sollten die Funktionsnamen aus Objekt(e) und ggf. Verb zusammengesetzt werden (z.B. bool istGesendet() oder void I2CBotschaftSenden()).
- Stellen Sie auch jeder Funktion eine kurze Beschreibung voran. Aus dieser sollte hervorgehen, was Sinn und Zweck der Funktion ist.

Beispiel für Anweisungsblöcke

SCHLECHT	<pre>uint8_t unter2_neu(uint8_t Was) { int a; for(a=10;a<20;a++) { if(arr[a]==Was) { return a; }; } return 0; }</pre>
GUT	<pre>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. */ { int a; for(a=ersteSchluesselPosition ; a<=letzteSchluesselPosition ; a++) { // durchlaufe alle Schlüsselpositionen if(schluesselArray[a]==schluessel) return a; // falls Schlüssel gefunden, }; // gib die erste Position zurück return 0; }</pre> In diesem Beispiel wäre der Funktionsname schluesselPosition statt schluesselPositionFinden auch geeignet gewesen.

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	<pre> ... 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); ... </pre>
GUT	<pre> ... void printDecimalDigit(int x, int y, int DigitToBePrint) { lcd_goto(x,y); lcd_putc(0x30 + DigitToBePrint%10); }; ... printDecimalDigit(1, 0, hunderter); printDecimalDigit(1, 1, zehner); printDecimalDigit(1, 2, einer); ... </pre>

- 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).

Beispiel für Verzweigungen 1 - Umwandlung in Array

SCHLECHT	<pre> ... 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 ... </pre>
GUT	<pre> ... char AusgabeZahl[6][4] = { "null", "eins", "zwei", "drei", "vier", "fünf" }; ... outputToLCD(AusgabeZahl[i]); ... </pre>

Beispiel für Verzweigungen 2 - Umwandlung in Switch-Case

SCHLECHT	<pre> ... 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 ... </pre>
GUT	<pre> ... 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; }; ... </pre>

Beispiel für Verzweigungen 3 - Umwandlung in For-Next

SCHLECHT	<pre> ... 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; ... </pre>
GUT	<pre> ... 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]; }; ... </pre>

- 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	<pre> ... 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; ... </pre>
GUT	<pre> ... 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;}; ... </pre>

Beispiel für Verzweigungen 5 - Reduzieren der Anweisungen

SCHLECHT	<pre> ... 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; }; ... </pre>
GUT	

`</code>` 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	<pre>void main() { initAll; while(1){ // es wäre auch for(;;){} möglich Eingabe; Verarbeitung; if (CancelButton==1) main; Ausgabe; } }</pre>
GUT	<pre>void main() { initOneTimeFunctions; while(1){ // äußere Endlos-Schleife initOtherFunctions; CancelButton = 0; while(!CancelButton){ // innere Schleife mit Abbruchbedingung Eingabe; Verarbeitung; if (!CancelButton) Ausgabe; }; } }</pre>

Beispiel für Schleifen 2 - Abbrechen von verschachtelten Schleifen

SCHLECHT	<pre>for(int xpos=0;xpos<10;xpos++){ initYPos; for(int ypos=0;ypos<20;ypos++){ Eingabe; Verarbeitung; if (CancelButton) goto Abbruch; }; } Abbruch: ...</pre>
-----------------	---

GUT	<pre> 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 } ... </pre> 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	<pre> ... Zahlenwert = (ADC_Wert/10)%10 + 0x30; Zahlenwerte[i] = Zahlenwert; Flaeche = (Zahlenwert - 0x30) * Breite; LCD_putc(Zahlenwerte[i]); ... </pre>
GUT	<pre> ... Laenge = (ADC_Wert/10)%10 ; Laengen[i] = Laenge; Flaeche = Laenge * Breite; LCD_putc(Laengen[i]+ 0x30); ... </pre>

Bewertung

abzugebende Work Products

Bitte legen Sie folgende Unterlagen in einem Zip-File unter ILIAS ab:

- Lastenheft und OPL als xls
- Modulblockbild/Blockschaltbild als xml
- Programm mit `main.c` sowie weitere für Ihr Projekt benötigte c- und h-Files

Die ATMEL-spezifischen Dateien (*.cproj, *.atsln, Debug-Ordner, etc.) nicht mit ablegen. Überprüfen Sie, ob aus diesen Files lauffähiger Code erstellt werden kann. Wenn dabei etwas zu beachten ist (z.B. Anlegen neuer Compiler Symbole), sollte dies in die Programmbeschreibung aufgenommen werden.

Bewertung

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

From:

<https://wiki.mexle.org/> - **MEXLE Wiki**

Permanent link:

https://wiki.mexle.org/microcontrollertechnik/vorgaben_fuer_die_softwareentwicklung?rev=1561069463

Last update: **2021/05/09 10:08**

