

<b>1. NAMENSGBUNG</b>	<b>2</b>
1.1. Variablenamen	2
1.2. Funktionsnamen	2
1.3. Modulnamen	2
<b>2. VARIABLENTYPEN</b>	<b>3</b>
<b>3. VARIABLEN-DEKLARATION</b>	<b>4</b>
3.1 Deklaration im C-File	4
3.2 Extern Deklaration im H-File	4
3.3 Fixe Werte	4
3.4 Globale Variablen	5
3.5 Zugriff auf Variablen	5
<b>4. KOMMENTARE</b>	<b>5</b>
4.1 Modul-Header	5
4.2 Funktions-Header	6
4.3 Block-Kommentare	6
4.4 Zeilen-Kommentare	6
4.5 Fußzeile	6
<b>5. PROGRAMMIERSTIL</b>	<b>7</b>
5.1 Textformat	7
5.2 Gliederung des Codes (C-Programme)	7
5.3 Gliederung des Codes (ASM-Programme)	7
5.4 Funktions-Schalter	8
5.5 Parameter-Index	8
<b>6. HISTORY</b>	<b>9</b>

# Codierungsrichtlinien

Diese Codierungsrichtlinien sind im „embedded Bereich“ auf 8 und 16Bit Prozessoren gültig. Um die Software später auf ein andere Prozessorsysteme portieren zu können, ist es zwingend notwendig, folgende Richtlinien bei der Code-Erstellung zu berücksichtigen. Alle Codeteile, die durch Wartung oder Erweiterungen verändert werden, sind ebenso nach diesen Richtlinien zu ändern.

## 1. Namensgebung

Die Namensgebung ist für die weitere Wartung und Pflege des Programms von besonderer Bedeutung. Aus den Namen soll man auf einem Blick erkennen, worum es geht. So soll der Type, das Modul in dem es Registriert ist und eine logischer Bezeichnung zum Namen erkennbar sein.

### 1.1. Variablennamen

Variablennamen bestehen aus dem folgenden Teilen :

**Type**  
**Modulnamen (nur bei externen Variablen)**  
**Bezeichner**

z.B.: `gabyGDPConfigSystem[8]`

Dieses Beispiel ist ein ...

GLOBAL ARRAY of BYTES im Modul GDP und der Name ist ConfigSystem.

### 1.2. Funktionsnamen

Der Funktionsname ist ähnlich aufgebaut und besteht aus folgenden Teilen :

**Rückgabetype**  
**Modulname**  
**Funktionsbezeichnung**

z.B.: `pbyGetObjectPnt(byte byPage, byte byObject)`

Dieses Beispiel ist eine interne Funktion ...

Rückgabetype ist ein POINTER of BYTES, die Funktion heißt GetObjectPnt

### 1.3. Modulnamen

Aus dem Modulnamen soll die eindeutige Funktion des Moduls sowie eine eventuelle Gliederung mehrerer Layer erkennbar sein.

z.B.: `lcddrv_L0_int.c`

Dieses Beispiel ist ein Modul im LCD-Driver, Layer 0, internes File, C-Syntax.  
**Ein internes File wird nur im Modul selbst inkludiert.**

## 2. Variablentypen

Der Type der Variable kann aus folgenden Abkürzungen bestehen:

ANSI	W-Breite	im Code (defined)	kann alleine stehen	Abkürzung	Priorität
#define (macro)	-	#define (macro)	Ja, Funktionen	m	1
static	-	static	Nein	st	2
(global Variable)	-	--	Nein	g	2
const	-	const	Ja, Strings, Tabellen	co	2
struct	-	struct	Ja, in Verb.mit struct	s	3
union	-	union	Ja, in Verb. mit struct	u	3
* (pointer)	-	*	Nein	p	4
array	-	[ ]	Nein	a	5
unsigned char	8	byte	Ja	by	6
unsigned int	16	word	Ja	w	6
bit	1	bit	Ja	b	6
integer	16	int	Ja	i	6
long	32	long	Ja	l	6
void	-	void	Ja, bei Funktionen	v	6
(string, zero term.)	-	char[], char *	Ja	sz	6
unsigned long	32	ulong	Ja	ul	6

Werden Datentypen mit #TYPEDEF deklariert, so wird zusätzlich zum Namen des neuen Datentyps die jeweilige Abkürzung mit einem Underline „\_“ vorangestellt.

Hier einige Beispiele zur Erläuterung:

### VARIABLE :

```

s_ObjectMovingTable    xdata    saObjectMovingTable[256];
u_BoxCoords            xdata    guaDisplayViews[8];
s_ExchangeBD          idata    gspBD;
ulong                 xdata    gaulDisplayBufferAddress[8];
byte                  idata    gbyTextCursorX;
byte                  data     gabyConfigSystem[8];
bit                   data     bFirstPixel;

```

### STRUCT:

```

typedef struct
{
    byte byHeaderSize;
    byte byFixedSizeX;
    byte byFixedSizeY;
    byte byOffset;
    byte byEnd;
    byte code * pbyObject[256];
} s_Object;

```

#### UNION :

```
typedef union
{
    struct
    {
        word        wHi;
        word        wLo;
    } Words;
    ulong          ulValue;
} u_LongSplit2Word;
```

#### MACRO :

```
#define mbyLowOfWord(w)      (((byte *) &(w)) + 1)
```

#### CONST :

```
const byte code coText[] = "Hello World";
```

*Hier genügt der Hinweis "co", da damit feststeht, das der Text im CODE liegt bzw. nicht veränderbar ist.*

## 3. Variablen-Deklaration

### 3.1 Deklaration im C-File

Die Variable wird ausschließlich im C-File angelegt. Die Variable kann im Code-File selbst angelegt werden bzw. kann es in einer zusätzlichen Datei (.c-File) erfolgen.

### 3.2 Extern Deklaration im H-File

Um anderen Modulen die Möglichkeit zu bieten, auf Variablen zugreifen zu können, ist eine EXTERN Deklaration in einem H-File notwendig. Wird die Variable zwar von anderen Modulen verwendet, jedoch soll sie nicht für alle global verwendbar sein, wird die Deklaration in einem H-File vorgenommen, welches im Namen noch zusätzlich die Endung **int** (INTERN) hat. z.B. **lcddrv\_L0\_int.h**. Die globale H-Datei heißt dann **lcddrv\_L0.h**.

### 3.3 Fixe Werte

Im Code selbst sollen keine „hardcodierten“ Werte vorkommen. Diese werden in einem gesonderten File mittels **#define** angelegt. So können später die Werte bequem in einem zentralen File geändert werden.

```
z.B.: #define UNSIGNED          0x00
      #define SIGNED           0x80
      #define NO_FEATURES      0
      #define OBJECT_ROM_SMALL 0x00
      #define OBJECT_ROM_LARGE 0x40
```

## 3.4 Globale Variablen

Variablen, die im gesamten Projekt gültig sind, werden in einer zentralen Datei deklariert. Zusätzlich sollte eine Gliederung sowie Kommentare verwendet werden.

## 3.5 Zugriff auf Variablen

Zugriffe auf Teile einer Variable, erfolgen ausnahmslos über definierte Macros. Dies gewährleistet die Möglichkeit, den Code auf andere Prozessoren zu Portieren, die andere Wortbreite bzw. eine andere Byteorder haben.

```
z.B.: #define mbyLowOfWord(w)          (*(( (byte *) &(w)) + 1))
      #define mbyHighOfWord(w)       (*(( (byte *) &(w)) + 0))
      #define mwLowOfLong(l)         (*(( (word *) &(l)) + 1))
      #define mwHighOfLong(l)        (*(( (word *) &(l)) + 0))
      #define mbyByteOfLong(l, b)    (*(( (byte *) &(l)) + b))
      #define mwMiddleWordOfLong(l)  (*(( (word *) ((( (byte *) &(l)) + 1))))
```

Unions sollten so sparsam wie möglich verwendet werden, da UNIONS bei Portierungen immer extra angepasst werden müssen. Wenn kein Mangel an Speicher besteht, so werden diese auf jeden Fall vermieden.

# 4. Kommentare

Kommentare werden generell in Englisch verfasst. Dabei ist auf eine kurze, prägnante Ausführung zu achten. Die Maximale Breite einer Zeile beträgt 110 Zeichen.

## 4.1 Modul-Header

Zu Beginn eines jeden Files steht ein Modul-Header. Er leitet das jeweilige Modul ein.

### Beispiel für einen Modul-Header :

```
/******
*
*   © 2003 - 2004, Max Mustermann AG
*
******
*
* File Name           : LCDDRV_L0_int.c
* Module Name         : Level 0 Commands for Display
* Module Registration : LCDDRV
* Author              : Philipp Caha
* Company             : Max Mustermann AG
* Date (DD.MM.YYYY)  : 01.01.2004
* Module/File Description : Low Level Commands for DISPLAY with T6963
*
* History:
*
*   $Log: lcddrv.h,v $
*   Revision 1.00 2004/01/01 08:00:00 caha
*   Initial Version
*
******/
```

## 4.2 Funktions-Header

Jede Funktion besitzt einen Funktions-Header. Er leitet die Funktion ein und gibt Auskunft über die darin abgehandelte Aktion.

### Beispiel für einen Funktionsheader :

```
/******
 *
 * © 2003 - 2004, Max Mustermann AG
 *
 ******
 *
 * Function Name      : byLCDDRV_L0_Plot (0x01)
 * Author            : Philipp Caha
 * Date (DD.MM.YY)  : 2.10.2003
 * Parameters        : byte*      pbyCmdBuffer
 *
 *                   Buffer :
 *
 *                   | M1 | Y10 | Y9 | Y8 | M0 | X10 | X9 | X8 |
 *                   |---|-----|
 *                   | X7-X0 |
 *                   |---|-----|
 *                   | Y7-Y0 |
 *                   |---|-----|
 *
 * Returns           : Length of Command-String
 * Reference         : --
 * Description       : Set a Pixel
 ******/
```

Werden der Funktion Parameter übergeben, so werden diese im Header vermerkt und deren Type angegeben. Die Funktion bzw. Wertigkeit der Parameter müssen dabei vermerkt werden.

## 4.3 Block-Kommentare

Block-Kommentare dienen zur übersichtlichen Erklärung der Befehle in den nächsten Zeilen. z.B.:

```
/* Draw a Segment of a Circle */
vGetCoordsOfAngle(uDegreeStart.wValue,
                 uR.wValue,
                 &uGraphicCursorX.iValue,
                 &uGraphicCursorY.iValue);

uGraphicCursorX.iValue = iMx + uGraphicCursorX.iValue;
uGraphicCursorY.iValue = iMy - uGraphicCursorY.iValue;
```

## 4.4 Zeilen-Kommentare

Wenn möglich und sinnvoll, sollte jede Zeile kurz kommentiert werden. Bei den Kommentaren genügt oft nur ein Stichwort (siehe darunter liegendes Beispiel). Der Anteil der „inline – Documentation“ sollte jedoch zumindest 50% des gesamten Quellcodes umfassen.

```
iX = miGetXCoord(pbyCmdBuffer + 1);           // new Position X
```

## 4.5 Fußzeile

Am Ende des Moduls kommt eine Fußzeile.

```
/****** End of File *****/
```

## 5. Programmierstil

### 5.1 Textformat

Wenn im Editor TAB's verwendet werden, so müssen diese bei Übergabe in SPACES umgewandelt werden. In Oberflächen wie µVision oder ULTRAEDIT kann dies automatisch eingestellt werden.

### 5.2 Gliederung des Codes (C-Programme)

Um eine überschaubare Gliederung zu ermöglichen, wird bei der Strukturierung um 4 Zeichen eingerückt.

#### Struktur bei IF :

Die Klammern werden immer in einer neuen Zeile positioniert, um die Struktur besser erkennen zu können.

```
if (byValue == 1)
{
    /* Origin is set */
    iX += uOriginX.iValue;
    iY += uOriginY.iValue;
}
```

#### Struktur bei SWITCH-CASE :

```
switch (*pbyCmdBuffer)
{
    case REMOVE :

        /* Remove Object */
        byValue = 1;
        break;

    case CLAEER :

        /* Clear Object */
        byValue = 0;
        break;
}
```

### 5.3 Gliederung des Codes (ASM-Programme)

Um eine überschaubare Gliederung zu ermöglichen und Sprungmarken positionieren zu können, wird in der Strukturierung um mindestens 8 Zeichen eingerückt. z.B.:

```
jNoGlobalDrawMode2:    mov     b, r7                ; save r7 in b
                       mov     r7, #DATARD
                       call    _vDisplayWriteCmd    ; write cmd
                       call    byDisplayReadData    ; read data-byte
                       mov     a, r7
                       xch     a, b
                       mov     r7, a
```

Die Sprungmarken werden immer mit einem „j“ eingeleitet.

## 5.4 Funktions-Schalter

Um nach der Codierungsphase bzw. für Testzwecke nicht benötigte Funktionen ausblenden zu können, muß jede Funktion und die dazugehörige Querverweise mit einem #IF - #ENDIF eingeschlossen werden. Die dazugehörigen Schalter werden in einer eigenen Datei vermerkt.

### Beispiel :

```
#if _LCDDRV_L0_MOVE_OBJECT == USED
byte byLCDDRV_L0_MoveObject(byte pdata* pbyCmdBuffer)
{
    .
    .
    .
}
#endif
```

### Dazugehörig die FCT-Datei :

```
.
.
#define _LCDDRV_L0_MOVE_OBJECT          USED
#define _LCDDRV_L0_CLEAR_BUFFER        NOT_USED
```

Der Dateiname sollte mit der Kurzbezeichnung FCT im Namen darauf hinweisen. Pro Projekt ist eine FCT-Datei anzulegen.

## 5.5 Parameter-Index

Werden einer Funktion Parameter als Pointer auf einen String übergeben werden, so müssen für den Index, also den Zugriff auf die Elemente, #defines in der Funktion definiert werden.

### Beispiel :

```
byte byLCDDRV_L0_DrawObject(byte pdata* pbyCmdBuffer)
{
    #define byObjPage      (*(pbyCmdBuffer + 0))
    #define byObjNumber   (*(pbyCmdBuffer + 1))

oder

    #define byObjPage      pbyCmdBuffer[0]
    #define byObjNumber   pbyCmdBuffer[1]
```

Wichtig ist nur, daß für die Parameter sinnvolle Namen im Code verwendet werden.

## 6. History

Um Zwischenversionen, Releases bzw. einfach nur den Ablauf des Projektes nachvollziehbar zu machen, muss ein Versionsmanagement eingesetzt werden. Der Code sollte jeden Tag eingecheckt werden.

Zusätzlich werden periodisch (zumindest monatlich) Versionen „eingefroren“ welche der Projektdokumentation dienen.

Als Programme zum Versionsmanagement kommen in Frage :

- CVS
- Visual Source Safe
- oder händisch eintragen

Die Kommentare in der History müssen aussagekräftig die Änderungen beschreiben.