Einführung in die Programmiersprache C und in den C166-Compiler

Die vorliegenden Unterlagen sollen einen kurzen Überblick über die Software-Entwicklung in C geben. Diese Unterlagen erheben keinen Anspruch auf Vollständigkeit sondern dienen nur zur Demonstration.

Beispiel einer Controller Programmgliederung

A.) Compiler-Steueranweisungen

B.) Datendefinitionen

C.) Functions

D.) Interrupt-functions

E.) Main-Programm

Beschreibung der einzelnen Programmabschnitte

A) Compiler-Steueranweisungen

Der Aufruf von Compiler-Steueranweisungen ist natürlich nicht ausschließlich auf den Programmbeginn beschränkt. Normalerweise befinden sich am Programmanfang aber eine Reihe von diesen Befehlen. Sie werden auch durch die Programmieroberfläche eingefügt.

Beispiel einer Compileranweisung: #pragma debug code pl(60)

debug

Aufnahme von Debug-Informationen in die Objektdatei, damit ein Programm symbolisch getestet werden kann. (z.b. für den Simulator Adresseninformation, Datentypen, Zeileninformationen etc.)

code

Mit dieser Anweisung wird der Compiler veranlaßt, den erzeugten Programm-Code in der Form von Assembler-Mnemonics in die List-Datei zu übernehmen.

pl(x)

Dieser Befehl stellt die Zeilenzahl pro Seite im List-File ein.

define

Mit define-Anweisungen können häufig verwendete Konstanten durch Namen ersetzt werden.

Beispiel: #define DIGIT 0x34

B.) Datendefinitionen

In diesem Abschnitt werden Variable definiert und dafür Speicherstellen reserviert.
Folgende Unterscheidungen sind zu machen:

a) Unterteilung nach der Verfügbarkeit:

Globale Variablen (extern)

Diese müssen außerhalb aller functions definiert werden und gelten ab diesem Zeitpunkt im ganzen Programm. Falls sie zusätzlich static definiert werden, gelten sie nur in diesem Modul.

Lokale Variablen (static, automatic)

Diese werden in der jeweiligen function definiert und stehen nur innerhalb der jeweiligen function zur Verfügung. Der Linker überlagert, nachdem er überprüft hat, ob sich zwei functions gegenseitig nicht aufrufen, diese Variablen, um Platz speziell im kleinen internen RAM des Controllers zu sparen. Mit der zusätzlichen Deklaration "static" kann dies verhindert werden, sodaß die jeweilige Variable ihren Wert behält. Wichtig für Interrupts, hier kann keine Überprüfung zur Linkzeit erfolgen!

b) Unterteilung nach dem Speicherort:

Der Compiler unterstützt mehrere Speichermodelle, die den Speicherort automatisch bestimmen.
Man kann aber auch den Speicherort definieren, um eine für die Applikation optimale Speicherverteilung zu erreichen.

near	16 bit pointer; 16 bit Adressenberechnung 16 KB für Variable in der NDATA Gruppe, 16 KB für constants in der NCONST Gruppe, 16 KB für die system area in der SDATA Gruppe.
idata	On-chip RAM ; schnellster Zugriff.
bdata	bit-adressierbares On-chip RAM.
sdata	SYSTEM area (Adressenbereich 0C000h-0FFFFh); PEC adressierbar.
far	32 bit pointer; gesamter Adressenbereich Das einzelne Objekt (array oder structure) ist auf 16 KB limitiert.
huge	32 bit pointer; gesamter Adressenbereich Das einzelne Objekt (array oder structure) ist auf 64 KB limitiert.
xhuge	32 bit pointer; 32 bit Adressen für Objekte, die innerhalb des Speichers unlimitiert sind.

c) Unterteilung nach Typen:

void	leer	
bit	1 bit	0 or 1
signed char	1 byte	-128 to +127
unsigned char	1 byte	0 to 255
signed int	2 bytes	-32768 to +32767
unsigned int	2 bytes	0 to 65535
signed long	4 bytes	-2147483648 to +2147483647
unsigned long	4 bytes	0 to 4294967295
float	4 bytes	±1,176E-38 to ±3,40E+38
double	8 bytes	±1,7E-308 to ±1,7E+308
pointer	2 / 4 bytes	Adresse von Objekten

SFR Zugriff:

sbit	1 bit 0 or 1
sfr	2 bytes 0 to 65535

Arrays

alle Datentypen, ausgenommen bit, Dimension in eckiger Klammer.

Strukturen

mehrere Variablen können in einer Struktur zusammengefaßt werden.

Unions

Es können verschiedene Datentypen in einer union Platz finden. Sie werden dann ab der gleichen Speicheradresse abgelegt. Der größte Datentyp bestimmt die Größe der union.

Beispiele von Datendefinitionen:

```
unsigned char idata var1,var2;
bit secflag;
const unsigned char text[5];
unsigned long far temp[100];
unsigned char *ptr;
signed char test_tab [4] [3];
float  idata x,y,z;


 struct  {
         unsigned int day;
         unsigned int month;
         unsigned int year;
         }d;

         d.year = 2000;
```

```
union {
        unsigned char access_byte[2];
        unsigned int access_int;
        }measure;
```

Zugriff: measure.access_byte[0] = 0x21; measure.access_byte[1] = 0x43; a = measure.access_int ergibt für das unsigned int a gleich 0x4321.

Mit dieser Definition kann auf zwei bytes als integer zugegriffen werden und man erspart sich die Umrechnung 256*high byte + low byte

C) Functions

Tätigkeiten, die an mehreren Stellen im Programm abgearbeitet werden sollten in sogenannte 'functions' zusammengefaßt werden. Die Definition für eine function muß sich vor dem Aufruf befinden. Befindet sich die function in einem anderen Modul, muß sie als Prototyp (1. Zeile der function definition) deklariert werden.

Struktur einer function:

Rückgabetyp function-Name (Übergabevariablen mit Typendeklaration)

{
Definition lokaler function-Variablen

Realisierung der function-Tätigkeit

eventuelle Wertrückgabe
}

function-Name: beliebig

Übergabevariablen: Liste von Werten,die an die function übergeben werden sollen.
Die Übergabewerte werden an lokale Variablen übergeben, die in der function definiert werden müssen. Die function arbeitet nur mit lokalen Kopien der Übergabevariablen, d.h. nach Beendigung der function sind diese Variablen im aufrufenden Programm unverändert. Will man eine Veränderung, so kann man an die function einen Pointer auf die Variable übergeben oder einfach mit globalen Variablen arbeiten.

Wertrückgaben: Jede function kann durch den 'return(Variable)'-Befehl einen Wert zurückgeben.

Beispiel von functions und ihrem Aufruf:

```
unsigned char getkey()

input=getkey();     /* function getkey() liefert einen Wert */

pointerset(anfang); /* An die function pointerset() wird ein Wert übergeben */

value=select(digit);/* An die function select() wird ein Wert übergeben, und ein entsprechender Wert
wird von der function rückgeliefert */
```

D) Interrupt functions
Allgemeine Form:

```
function name () interrupt x using y {
        Interrupt-Behandlung;
}
```

function name beliebig
An interrupt functions können keine Werte übergeben werden und sie liefern keinen Wert zurück.

Interruptquellen x : x bestimmt den Einsprungsvektor (aus dem User Manual des Controllers zu entnehmen)

z. B 0x20 für den Timer 0
Registerbank y : Name der Registerbank

Beispiel:

```
void periodint(void) interrupt 0x11 using period_bank
```

E) Hauptprogramm main()

Das Hauptprogramm: main() stellt den Kern jedes C-Programms dar. Das Hauptprogramm beginnt mit der Initialisierung und setzt dann mit dem abzuarbeitenden Programm fort.

Beispiel:

```
void main(void) {

    S0CON=0x8011;          /* Serielle 10-Bit-Rahmen */
    T0IC=0x44;             /* Timer0 Interrupt erlauben und Priorität setzen */

/* Rest des Hauptprogrammes */
while(1) {


        }
}
```

Um nun in den function's oder im main()-Programm verschiedene Tätigkeiten durchzuführen bedarf es verschiedener elementarer Operationen.

a) Arithmetische Operatoren:

Addition	+	Subtraktion	-	Multiplikation	*
Division	/	Modulo	%		

Beispiel: x=y+z;

b) Vergleichsoperatoren:

größer	>	größer gleich	>=
kleiner	<	kleiner gleich	<=
gleich	==	ungleich	!=
log.AND	&&	log.OR	||

Beispiel: if (x == y) z=1;

c) Inkrementieren, Dekrementieren um Eins:

Erhöhen vor	++i	nach	i++	einer Zuweisung
Erniedrigen vor	- - i	nach	i - -	einer Zuweisung

Beispiel: x = ++n;

d) Bit-Manipulationen

log.AND	&	log.OR	|	log.EXOR	^
Shift left	<<	Shift right	>>	Komplement	~

Beispiel: x=x & mask;

e) Zuweisungen:

Kompakte Formulierung:
statt Operand1 = Operand1 Operator Operand2;
kann Operand1 Operator = Operand2;
geschrieben werden.

Beispiel: a = a + b; x = x * y;
a += b; x *= y;

Zuweisungen laut einer Bedingung mit der Form:
Variable = (Bedingung) ? (Zuweisung 1) : (Zuweisung 2);

Beispiel: z=(a==5) ? 2 : 3;

f) Pointer-Operatoren:

Adresse einer Variable	&
Wert einer Adresse	*

Beispiel: y = x; entspricht: px = &x; y = *px;

g) Sonstige Symbole:

Abschluß eines Statements	;
Blockbegrenzer	{ }
Array's	[ ]

h) Kontrollstrukturen:

1) if-Anweisung:

```
if ( Bedingung)
        Statement 1 bzw. Block 1
else
        Statement 2 bzw. Block 2
```

```
Beispiel:  if (a==b)
                        x=1;
                else  {
                        k=1;
                        x=0;
                }
```

2) switch-Struktur:

```
switch ( Variable) {
        case Wert 1 : Statement 1, bzw. Block 1; break;
        case Wert 2 : Statement 2, bzw. Block 2; break;
        .       .       .
        default: Statement wenn keine Bedingung erfüllt; break;
        }
```

Beispiel:

```
switch (select)
{
case 1:  function1();      break;
case 5:  function2();      break;
default: function0();      break;
}
```

3) Schleifen (while und for):

```
for (Startwert;Laufbedingung;Inkrement)
        Statement bzw. Block

while (Laufbedingung)
        Statement bzw. Block

do
        Statement bzw. Block
while (Laufbedingung)
```

Die Schleifenkonstruktion mit do-while unterscheidet sich von der while-Konstruktion dadurch, daß die Laufbedingung erst am Schleifenende getestet wird.

Beispiel:

```
a=15;
while (a>5) {

        for(i=0; i<12; i+=5)  a- = 4;
}
```

4) Schleifenausstieg (break, continue)

Die break-Anweisung dient dazu, Schleifen vorzeitig zu verlassen. (Anwendung siehe Switch-Struktur) Die continue-Anweisung hat eine ähnliche Funktion, es wird nur dabei die Schleife nicht verlassen, sondern sofort wieder mit dem Schleifenanfang begonnen.

Interface zu ASSEMBLER-Programmen

Das Interface zu Assemblerprogrammen ist von der Compilerversion abhängig. Die an die function übergebenen Werte und der Rückgabewert der function werden am Userstack oder in Registern übergeben. Um eine Kompatibilität zu den C functions zu erreichen ist am besten, die gewünschte function in C zu schreiben und mit der Compileranweisung SRC zu kompilieren.
Damit erhält man ein Assemblerprogramm, das alle Übergaben richtig behandelt und die Namen der Programm- und Datensegmente richtig erstellt. Danach kann der Assemblercode optimiert und mit dem Assembler übersetzt werden.