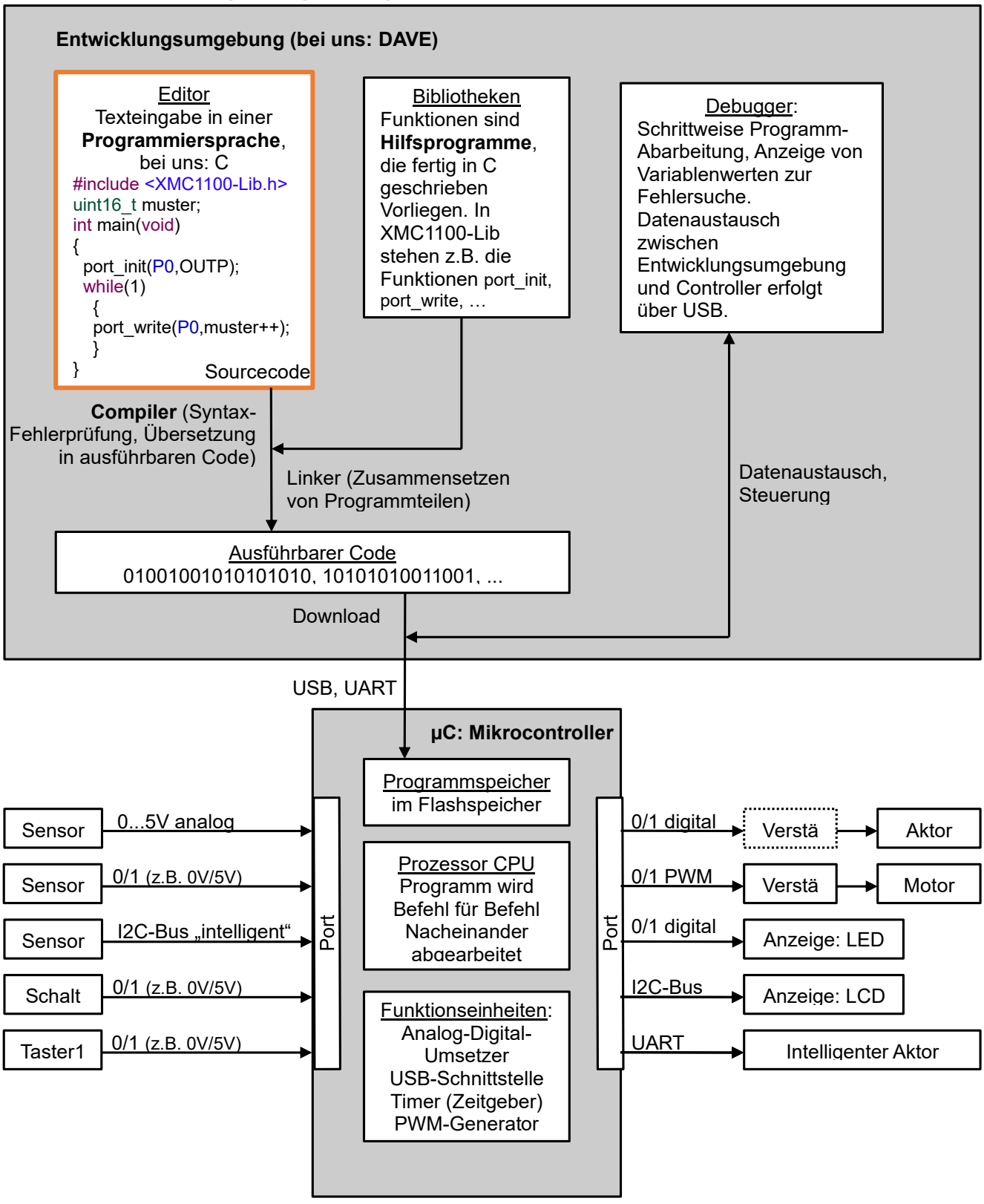
Inhaltsverzeichnis

1 Entwicklungsumgebung um Mikrocontroller

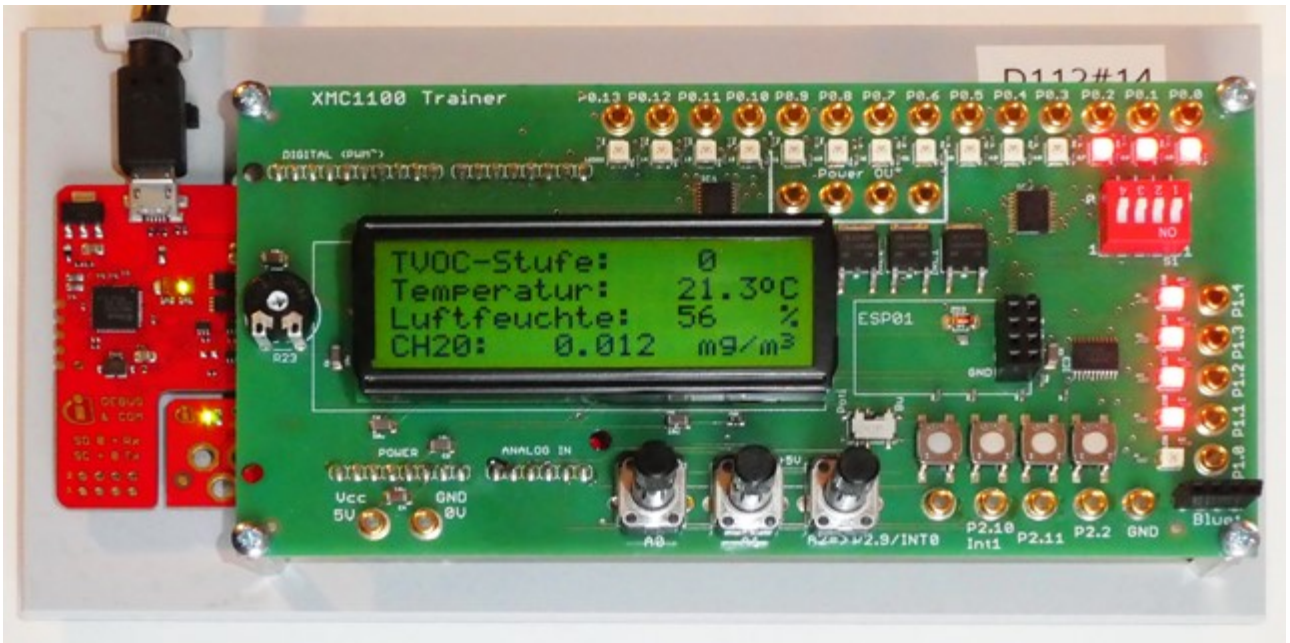


EVA-Prinzip: Eingabe → Verarbeitung → Ausgabe

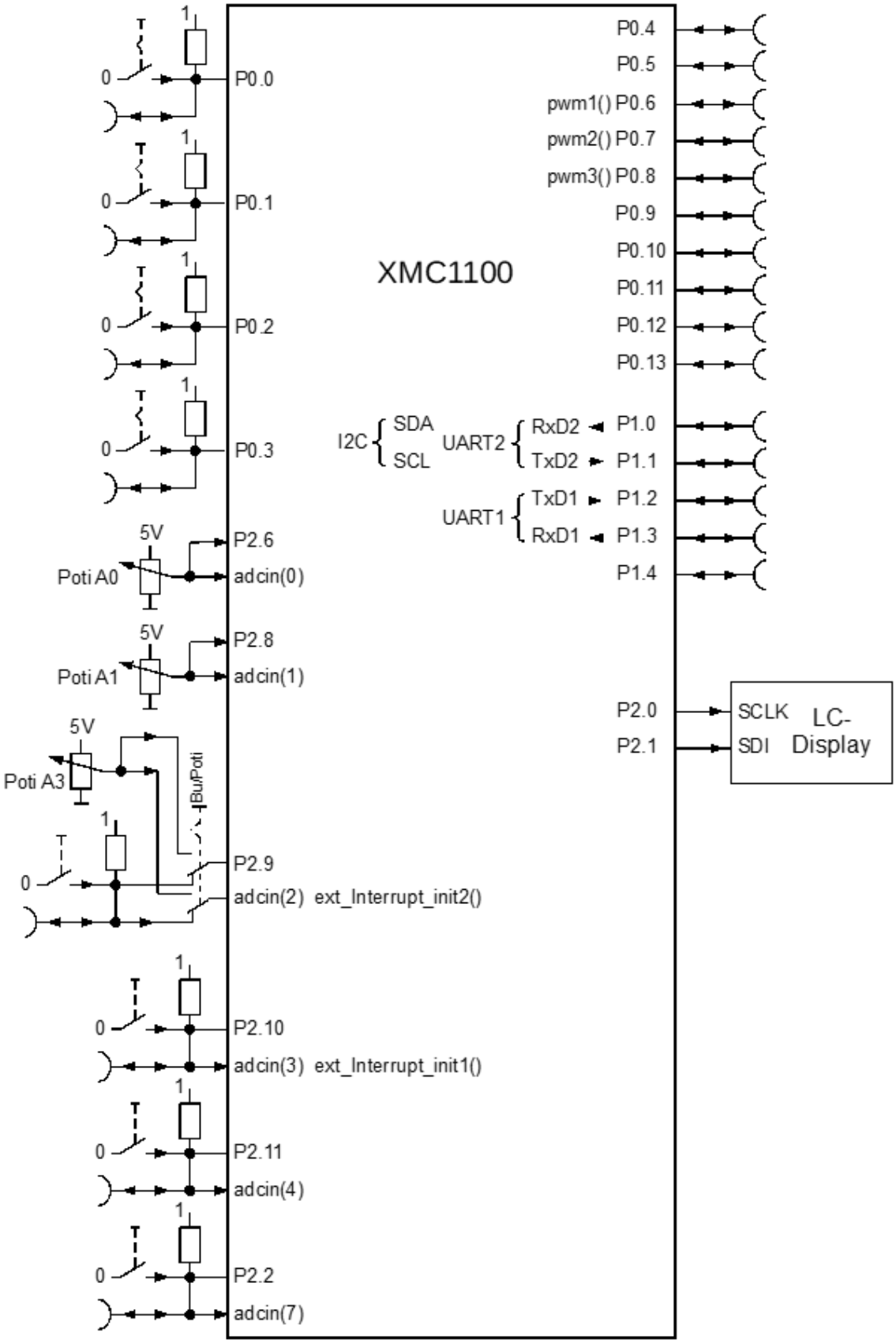
2 XMC1000-Trainer

2.1 Pinbelegungen

Portbit		Buchse	beachte!	Schalter /Taster	Sonderfunktion	Sonderfunktionen
P0.0	LED	Digital IN / OUT	0 oben, 1 unten	Schalter lowaktiv		
P0.1	LED	Digital IN / OUT	0 oben, 1 unten	Schalter lowaktiv		
P0.2	LED	Digital IN / OUT	0 oben, 1 unten	Schalter lowaktiv		
P0.3	LED	Digital IN / OUT	0 oben, 1 unten	Schalter lowaktiv		
P0.4	LED	Digital IN / OUT				
P0.5	LED	Digital IN / OUT				
P0.6	LED	Digital IN / OUT			pwm1	
P0.7	LED	Digital IN / OUT			pwm2	
P0.8	LED	Digital IN / OUT			pwm3	
P0.9	LED	Digital IN / OUT				
P0.10	LED	Digital IN / OUT				
P0.11	LED	Digital IN / OUT				
P0.12	LED	Digital IN / OUT				
P0.13	LED	Digital IN / OUT				
P1.0	LED	Digital IN / OUT / RxD2	auch Bluetooth, ESP01		SDA (I2C)	RxD2 (UART2)
P1.1	LED	Digital IN / OUT / TxD2	auch Bluetooth, ESP01		SCL (I2C)	TxD2 (UART2)
P1.2	LED	TxD1	Kommunikation über USB mit PC		TxD1 (UART1)	
P1.3	LED	RxD1	Kommunikation über USB mit PC		RxD1 (UART1)	
P1.4	LED	Digital IN / OUT				
P2.0					SCLK LCD	
P2.1					SDI LCD	
P2.2		Analog / Digital IN	<=Taster nicht gedrückt	Taster lowaktiv	adcin(7)	
P2.6				Poti A0	adcin(0)	
P2.8				Poti A1	adcin(1)	
P2.9		Analog / Digital IN	Schiebeschalterstellung	Poti A3 / Taster	adcin(2)	ext_Interrupt2_init(Flanke)
P2.10		Analog / Digital IN	<=Taster nicht gedrückt	Taster lowaktiv	adcin(3)	ext_Interrupt1_init(Flanke)
P2.11		Analog / Digital IN	<=Taster nicht gedrückt	Taster lowaktiv	adcin(4)	



2.2 Blockschaltbild



3 Formelsammlung C/C++

3.1 Datentypen

Datentyp	stdint.h type	Bits	Sign	Wertebereich
(unsigned) char	**uint8_t**	**8**	Unsigned	0 ... 255
signed char	**int8_t**	**8**	Signed	-128 ... 127
unsigned short	**uint16_t**	**16**	Unsigned	0 ... 65.535
short	**int16_t**	**16**	Signed	-32.768 ... 32.767
unsigned int	**uint32_t**	**32**	Unsigned	0 ... 4.294.967.295
(signed) int	**int32_t**	**32**	Signed	-2.147.483.648 ... 2.147.483.647
unsigned long long	**uint64_t**	**64**	Unsigned	0 ... 18.446.744.073.709.551.615
long long	**int64_t**	**64**	Signed	-9.223.372.036.854.775.808 ... 9.223.372.036.854.775.807
Datentyp	**IEE754 Name**	**Bits**		**Wertebereich**
float	**Single Precision**	**32**		**-3,4E38 ... 3,4E38**
double	**Double Precision**	**64**		**-1,7E308 ... 1,7E308**
pointer		**32**		**Adresse einer Variablen**

3.2 Operatoren

Da Gleichheitszeichen ist in C ein Zuweisungsoperator, d.h. einer Variablen einen Wert zuweisen, z.B. x = 10;

Mathematische Operatoren		Priorität	Verhältnis- und logische Operatoren	
++	Inkrement	Höchste	!	NOT
–	Dekrement			
-	Vorzeichen		>	Größer
			>=	Größer gleich
			<	Kleiner
			<=	Kleiner gleich
*	Multiplikation		==	Gleich
/	Division		!=	Ungleich
%	Modulo, Rest der Division			
+	Plus		&&	AND
-	Minus			
		niedrigste	||	OR
+=	x += 3; wie x = x + 3			
-=	x -= 3; wie x = x - 3;			
*=	x *=5; wie x = x * 5;			
/=	x /= 7; wie x = x / 7;			

Bitweise Operatoren		Beispiele	Ergebnisse
&	UND	X = 10;	
|	ODER	Y=++X;	Y=11
^	EXOR	Y=X++;	Y=10
~	Einerkomplement	Y=0x11;	
<<	Nach links schieben	Y=Y<<1;	Y=0x22
>>	Nach rechts schieben	(bitweise um 1 nach links schieben)	

3.3 Aufbau eines C-Programms

C unterscheidet zwischen Groß- und Kleinschreibung!

3.3.1 Kommentareingabe

```
// Kommentar für eine Zeile
/* Kommentar für einen Block von einer
oder mehreren Zeilen */
```

3.3.2 Befehlsblock

```
{     // Anfang eines zusammengehörigen Befehlsblocks (begin)
}     // Ende eines zusammengehörigen Befehlsblocks (end)
```

3.3.3 Compileranweisung über zusätzliche Quellcodes mit Funktionen und Deklarationen:

```
#include <XMC1100-Lib.h>  // Funktionen-Bibliothek fuer XMC1100
```

3.3.4 Konstantendeklaration

```
#define muster1 0x0F      // LED-Bitmuster 1 unter 4 LEDs an
```

3.3.5 Deklaration von globalen Variablen (vor main!)

```
uint16_t adc_wert;        // 16-Bit-Wert ganze Zahl ohne Vorzeichen
```

3.3.6 Deklaration von Funktionen

```
void time(uint32_t zeit); // eigene Funktion zur Zeitverzögerung s.u.
```

Deklaration einer eigenen Funktion, die unter main steht, damit der Name in main bekannt ist. Alternative: die gesamte Funktion hier angeben statt unter main zu schreiben.

3.3.7 Hauptprogramm (Hauptfunktion)

```
int main(void)            // Hauptprogramm
{
}
```

3.3.8 Beispiel

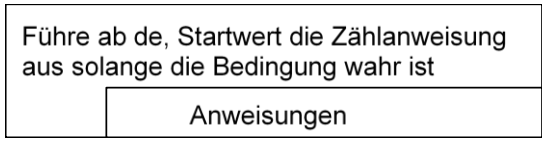
```
//  Muster zeitveränderbares Blinklicht
#include <XMC1100-Lib.h>  // Funktionen-Bibliothek fuer XMC1100
#define muster1 0x0F      // LED-Bitmuster 1 unter 4 LEDs an
#define muster2 0xF0      // LED-Bitmuster 2 obere 4 LEDs an
uint16_t adc_wert;        // 16-Bit-Wert ganze Zahl ohne Vorzeichen
void time(uint32_t zeit); // eigene Funktion zur Zeitverzögerung
int main(void)            // Hauptprogramm ---------------------
{
  port_init(P0,OUTP);      // Port0 auf Ausgabe -> LED-Anzeige
  adc_init();              // Analog-Digital-Converter
                              initialisieren
  while(1U)                // Hauptprogramm-Endlosschleife
  {
     port_write(P0,muster1);// Muster1 an Port0 ausgeben
     adc_wert = adc_in(0);  // Wert vom ADC -> Zeitverzoegerung
     time (adc_wert*0xff);  // Zeitverzögerung
     port_write(P0,muster2);// Muster2 an Port0 ausgeben
     time (adc_wert*0xff);  // Zeitverzögerung
  }//while
}//main
// eigene Funktion zur Zeitverzögerung ---------------------------
void time(uint32_t zeit)
{
      uint32_t i;            // lokale Zählvariable
      for (i=0;i<zeit;i++);  // Schleife zur Zeitverzögerung
}
```

3.4 Schleifen

3.4.1 For-Schleife (zählergesteuerte Schleife)

Erzwingt eine genau berechenbare **Zahl der Wiederholungen**

```
for(<startwert>;<Bedingung>;<Schrittweite>)
{     //Anweisungen
}
```



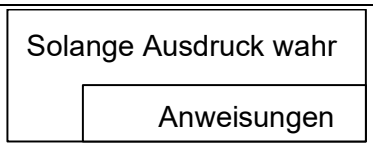
startwert:	Anfangswert der Variablen
Bedingung:	Schleife wird so lange durchlaufen, wie die Bedingung ‚wahr' ist.
Schrittweite:	Anweisung zum Erhöhen oder Erniedrigen der Variablen

```
// Beispiel Ausgang 10x invertieren
    for (x=10; x>0; x--)
    {
       ausgang =~ausgang;
    }
// Beispiel Zeitverzögerung
    uint32_t i;                 // Zählvariable
    for (i=0;i<0xff000;i++); // Schleife zur Zeitverzögerung
```

3.4.2 While-Schleife (kopfgesteuerte Schleife)

Wird **nur so lange** wiederholt, wie eine am **Schleifenanfang** stehende Bedingung **erfüllt** ist.

```
while(<Bedingung>)
{     //Anweisungen
}
```



Wenn die am Schleifenanfang stehende **Bedingung nicht gilt,** dann wird die gesamte Schleife übersprungen.
Solange die am Schleifenanfang stehende **Bedingung gilt**, wird die Schleife wiederholt.
Die Prüfbedingung steht vor den Anweisungen. Sie heißt deshalb "kopfgesteuerte Schleife".

```
// Beispiel: Solange der lowaktive Taster an P2.9 gedrückt ist,
// wird der Ausgang invertiert.
    while (bit_read (P2,9) == 0)
    {
       ausgang = ~ausgang;
    }
```

3.4.3 Do-While-Schleife (fußgesteuerte Schleife)

Die Schleife wird prinzipiell erst mal durchlaufen. Am Ende des Durchganges steht eine Prüfbedingung, die entscheidet, ob die Schleife wiederholt wird.

```
do
{      //Anweisungen
}
while(<Bedingung>);
```

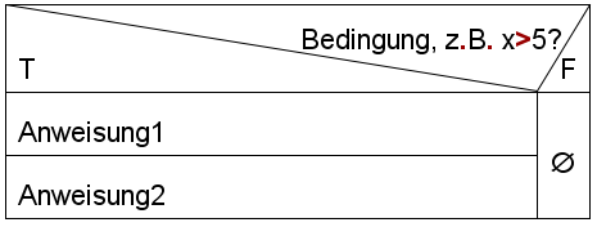
```
// Beispiel: Die Schleife wird maximal 100 mal und mindestens 1 mal
// durchlaufen. Sie wird frühzeitig abgebrochen, wenn der lowaktive
// Taster an P2.9 gedrückt (= 0) wird.
    X = 100;
    do
    {
       x--;
    }
    while ((x > 0) && (bit_read (P2,9) == 1));
```

3.5 Programmverzweigungen

3.5.1 Verzweigung mit if

Bei der ‚if' - Anweisung werden die folgende Anweisung (oder ein ganzer Anwendungsblock) **nur dann ausgeführt**, wenn die **hinter ‚if' stehende Bedingung wahr** ist.

```
if(<Bedingung>)
{     <Anweisung1>;
      <Anweisung2>;
      ...
}
```

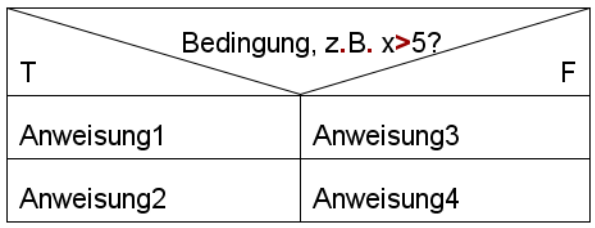



```
// Beispiel: Wenn taster1 gedrückt ist, wird ausgang1 zu eins und
ausgang2
// zu null. Drückt man dagegen taster2, wird ausgang3 zu eins.
  if (taster1 == gedrueckt)
  {                              // Block mit mehreren Anweisungen
     ausgang1 = 1;               // wird ausgeführt, wenn die Bedingung
     ausgang2 = 0;               // hinter if wahr ist
  }
  if (taster2 == gedrueckt) ausgang3 = 1; // keine { } nötig, da 1
                                             Anweisung
```

3.5.2 Verzweifung mit if – else

Mit ‚**if – else**' kann **nur zwischen zwei Alternativen** gewählt werden.

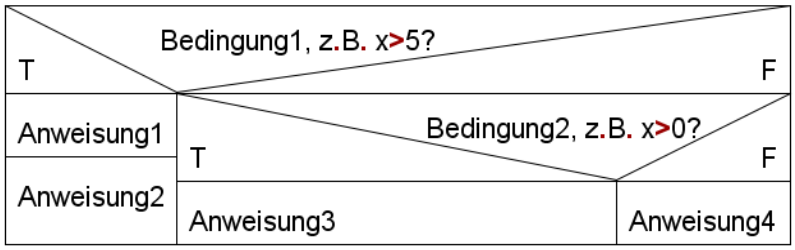
```
if(<Bedingung>)
{     <Anweisung1>;
      <Anweisung2>;
}
else
{     <Anweisung3>;
      <Anweisung4>;
}
```




```
// Beispiel: Wenn taster1 gedrückt ist, soll ausgang1 eins und
ausgang2 null
// werden, andernfalls soll ausgang1 null und ausgang2 eins werden.
  if (taster1 == 1)
  {                             // Block mit mehreren Anweisungen
    ausgang1 = 1;               // wird ausgeführt, wenn die Bedingung
    ausgang2 = 0;               // hinter if wahr ist
  }
  else
  {                             // Block mit mehreren Anweisungen
    ausgang1 = 0;               // wird ausgeführt, wenn die Bedingung
    ausgang2 = 1;               // hinter if nicht wahr ist
  }
```

3.5.3 Mehrfach – Verzweigung

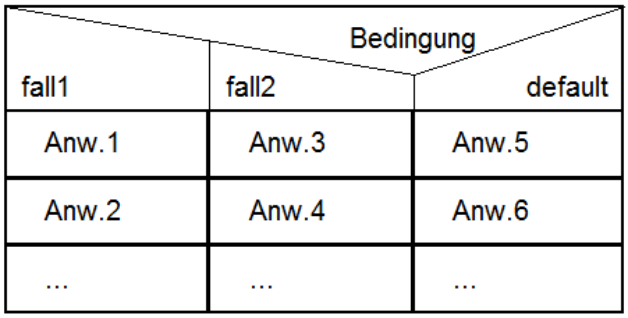
```
if(<Bedingung1>)
{     <Anweisung1>;
      <Anweisung2>;
}
else if((<Bedingung2>)
{     <Anweisung3>;
      ...
}
else
{     <Anweisung4>;
      ...
}
```



3.5.4 Fallauswahl mit switch

Mit der ‚**switch‘ –** Anweisung kann aus einer **Reihe von Alternativen** ausgewählt werden.
Es ist zulässig, dass mehrere Möglichkeiten gültig sind und dieselbe Wirkung haben.
Sie werden einfach nacheinander aufgelistet.
Passt **keine der Möglichkeiten**, dann wird die ‚**default‘** – Einstellung ausgeführt.
Achtung! **Auf keinen Fall break vergessen!!!**

```
switch(<Vergleichswert>)
{     case <Wert1>:
            <Anw.1>;
            <Anw.2>;
            ...
            break;
      case <Wert2>:
            <Anw.3>;
            <Anw.4>;
            ...
            break;
      ...
      default:
            break;
}
```




```
// Beispiel: In der Variablen ergebnis ist ein Messergebnis oder
eine Zahl gespeichert.
// abhängig vom genauen Wert sollen nun bestimmte Reaktionen
erfolgen.
  switch (ergebnis)
  {
    case 0x00:
    case 0x10:
    case 0x20:
        ausgang1 = 1; // wenn ergebnis 0x00 oder 0x10 oder 0x20 ist
        break;
    case 0x30:
        ausgang1 = 0; // wenn ergebnis 0x30 ist
        break;
    case 0x40:
        ausgang1 = ~ ausgang1; // wenn ergebnis 0x40 ist
        break;
    default:
        ausgang2 = 1; // in allen anderen Fällen
        break;
  } //switch
```

Hinweise: Für die ‚switch – Variable' darf man nur einfache Datentypen verwenden. Hinter case müssen Konstanten stehen.
Diese können mit **#define** am Anfang des Programms deklariert werden.

```
#define rechts  0x10        // ohne Semikolon!!
#define links   0x20
uint8_t richtung;
....
  switch (richtung)
  {
     case rechts:      motor = rechtskurve; break;
     case links:       motor = linkskurve; break;
     default:          motor = vorwaerts; break;
}
```

3.6 Funktionen

```
<Datentyp Rückgabewert> <funktionsname>(<Datentyp> Parameter1, <Datentyp>
Parameter2,...);
```

```
// Beispiele:
void pwm_init1 (void);          // ohne Rückgabewert, ohne Parameter
void lcd_byte (uint8_t val);    // ohne Rückgabewert, mit einem
                                   Parameter
uint8_t adc_in (uint8_t kanal);// mit Rückgabewert, mit Parameter
```

3.6.1 Definition von Funktionen

```
<Datentyp> funktionsname(Datentyp> <Parameter1, <Datentyp> Parameter2,...)
{    //Anweisungen
     return <Rückgabewert>;
}
```

```
// Beispiel1: Funktion ohne Übergabewert und ohne Rückgabewert
void time_100ms(void)
{
      uint32_t i;               // lokale Zählvariable
      for (i=0;i<0xff000;i++); // Schleife zur Zeitverzögerung
}
```

```
// Beispiel2: Funktion mit Übergabewert und ohne Rückgabewert
void time(uint32_t zeit)
{
      uint32_t i;               // lokale Zählvariable
      for (i=0;i<zeit;i++);     // Schleife zur Zeitverzögerung
}
```

```
// Beispiel3: Funktion mit Übergabewert und mit Rückgabewert
  int32_t addieren( int32_t z1, int32_t z2)
  {
     int32_t  result;     // lokale Variable
      result = z1 + z2;
      return (result);    // Rückgabewert
  }
```

Achtung: Wenn eine Funktion **definiert** wird, folgen **direkt hinter dem Funktionsnamen nur zwei runde Klammern, dahinter aber nix mehr (also NIE ein Strichpunkt)!!**

3.6.2 Funktionsaufruf

```
<funktionsname>(<Paramter1>, Paramter2, ...);
```

```
// Beispiel:
uint32_t ergebnis, zahl1, zahl2;
int main(void)
{  …
   time_100ms();                          // Funktion ohne
                                             Übergabewert
   time (0x3fff);                         // Funktion mit Übergabewert
   ergebnis = addieren (zahl1,zahl2);     // Funktion mit Rückgabewert
}
```

4 Funktionsbibliothek für den Controller XMC1100

Die Header-Datei mit allen Funktionsnamen wird durch `#include <xmc1100-lib.h>` eingebunden.

4.1 Verzögerungsfunktionen

Delay Verzögert den Programmablauf für die angegebene Zeitdauer

Funktion	Kommentar
delay_10us (uint8_t mikrosekunden10);	**// millisec = 0…65535**
delay_100us (uint8_t mikrosekunden100);	**// microsec100 = 0…255 → Zeitverzögerung 0 bis 255 mal 100us**
delay_ms (uint16_t millisekunden);	**// microsec10 = 0…255 → Zeitverzögerung 0 bis 255 mal 10us**

4.2 Bit und Byte Ein-/Ausgabe

Parameterliste:

Parameter	Eintrag Parametervariablen
port	P0, P1, P2
bitnr	0 … 7
direction	INP = 0, OUTP = 1

Achtung: P2 nur Input möglich

4.2.1 Initialisierung einzelner Portbits

Funktion	Kommentar
bit_init (uint8_t **port**, uint8_t **bitnr**, uint8_t **direction**);	**// Initialisieren einzelnes Bit für Ein- oder Ausgabe**
uint8_t **bit_read** (uint8_t **port**, uint8_t **bitnr**);	**// Einzelnes Portbit einlesen**
bit_write (uint8_t **port**, uint8_t **bitnr**, uint8_t value);	**// Einzelnes Portbit ausgeben**

4.2.2 Initialisierung gesamter Ports

Funktion	Kommentar
port_init (uint8_t **port**, uint8_t **direction**);	**// Initialisieren gesamtes Port für Ein- oder Ausgabe**
uint16_t **port_read** (uint8_t **port**);	**// Gesamtes Port einlesen**
port_write (uint8_t **port**, uint16_t value);	**// Gesamtes Port ausgeben**

4.2.3 Initialisierung gesamtes Port P0 mit IO-Bitmaske

Funktion	Kommentar
port0_init_maske(uint16_t io_maske);	**// Bit der Maske 0 → Input, 1 → Output** **// z.B. 0x00ff rechte 8 Bit Ausgabe, linkte 8 Bit Eingabe** **// (nur 6 sichtbar) 0b0000000011111111 = 0x00ff**

```c
// Beispiel: Bit-Ein-Ausgabe
#include <xmc1100-lib.h>  // Hilfsfunktionen
uint8_t temp;             // Zwischenspeicher
int main(void)
{
  bit_init(P2,11,OUTP);     // Taster
  bit_init( P0, 0, OUTP ); // LED
  while(1U)
  {
    temp = bit_read(P2,11);// Taster lesen
    bit_write(P0,0,temp);  // an LEDs anzeigen
  }
}
```

4.3 Analog – Digital – Konverter

Sechs analoge Eingänge, je 12 Bit Auflösung

Parameterliste:

Parameter	Eintrag Parametervariablen
kanal	0 = AN0: P2.6 (Poti links) 1 = AN1: P2.8 (Poti Mitte) 2 = AN2: P2.9 (Poti rechts) [Int0: Schiebeschalter beachten!] 3 = AN3: P2.10 [Int1] 4 = AN4: P2.11 7 = AN7: P2.2 **Rückgabewert:** **value**: 0 … 4095 (12 Bit Auflösung)

Achtung: ***alternative*** Verwendung der Portbits: P2.9 (AN2) , P2.10 (AN3) als ext. Interrupt
P2.9 (AN2) , P2.10 (AN3), P2.11 (AN4), P2.2 (AN7) als digitaler Input.

Funktion	Kommentar
adc_init (void);	// Analog-Digital-Konverter für alle Kanäle 0 … 7 initialisieren
uint16_t **adc_in** (uint8_t kanal);	// 12-Bit-Wert vom Analogeingang einlesen

```c
// Beispiel:
#include <xmc1100-lib.h>   // Hilfsfunktionen
int16_t wert;
int main(void)
{
  adc_init( );
  port_init(P0,OUTP);        // LEDs an P0
  while(1)
  {
    wert = adc_in(0);        // Poti einlesen
    port_write(P0,wert);     // 12 Bit-Wert anzeigen
    // port_write(P0,wert>>4); // 8 Bit-Wert anzeigen
  }
}
```

4.4 Funktionen für die LCD-Anzeige

Parameterliste:

Parameter	Eintrag Parametervariablen
row	Zeile 1 … 4
column	Spalte 1 … 16 (Display abhängig!)

Funktion	Kommentar
lcd_init (void);	// Initialisierung des Displays
lcd_clear (void);	// LCD-Anzeige löschen
lcd_setcursor (uint8_t row, uint8_t column);	// Setzen der LCD-Cursorposition
lcd_print (char text[ ]);	// Textausgabe an LCD ab gewählter Cursorposition bis '\0' [\0-terminiert]
lcd_char (uint8_t value);	// Ausgabe eines Zeichens an das LCD
lcd_uint8 (uint8_t value);	// Ausgabe eines positiven Byte 'value' als 3-stelligen Dezimalwert aufs LCD Führende Nullen werden zu 'blank'. Wertebereich: 0 ... 255
lcd_int8 (int8_t value);	// Ausgabe eines positiven und negativen Byte 'value' als 3-stelligen Dezimalwert aufs LCD Führende Nullen werden zu 'blank'. Wertebereich: -128 ... 0 ... +127
lcd_uint16 (uint16_t value);	// Ausgabe eines positiven 16-Bit-Integerwert 'value' als 5-stelligen Dezimalwert aufs LCD Führende Nullen werden zu 'blank'. Wertebereich: 0 ... 65535
lcd_int16 (int16_t value);	// Ausgabe eines positiven und negativen 16-Bit-Integerwert 'value' als 5-stelligen Dezimalwert + evtl. Minuszeichen aufs LCD Führende Nullen werden zu 'blank'. Wertebereich: -327680 ... 32767
lcd_hex8 (uint8_t value);	// Ausgabe einer 8-Bit-Variable 'value' als 2-stellige Hexzahl aufs LCD Wertebereich: 00 … FF
lcd_hex16 (uint16_t value);	// Ausgabe einer 16-Bit-Variable 'value' als 4 stellige Hexzahl aufs LCD Wertebereich: 0000 ... FFFF

```
// Beispiel:
#include <XMC1100-Lib.h>   // Bibliothek
uint8_t zaehler8=0;
uint16_t zaehler16=0;
uint8_t text1[]="ab 255: ";
uint8_t text2[]="bis 65535: ";
int main(void)
{
  lcd_init();              // LCD initialisieren
  while(1)                 // Endlos
  {
    lcd_setcursor(1,1);    // links oben
    lcd_print (text1);
    lcd_uint8(zaehler8);   // 8 Bit
    lcd_setcursor(2,1);    // Zeile 2
    lcd_print (text2);
    lcd_uint16(zaehler16); // 16 Bit
    lcd_char ('!');        // Zeichen
    delay_ms(10);
    zaehler8--;
    zaehler16++;
  }
}
```

4.5 Externe Interrupts

Parameterliste:

Parameter	Eintrag Parametervariablen	
flanke	RE = 1	Rising Edge [ansteigende Flanke]
	FE = 0	Falling Edge [abfallende Flanke]

Funktion	Kommentar
ext_interrupt1_init (uint8_t **flanke**);	**// Ext. Interrupt mit Portpin P2.10 initialisieren**
ext_interrupt2_init (uint8_t **flanke**);	**// Ext. Interrupt mit Portpin P2.9 initialisieren**
ext_interrupt1_enable (**void**);	**// Freigabe externen Interrupt 1 von P2.10**
ext_interrupt2_enable (**void**);	**// Freigabe externen Interrupt 2 von P2.9**
ext_interrupt1_disable (**void**);	**// Sperren externen Interrupt 1 von P2.10**
ext_interrupt2_disable (**void**);	**// Sperren externen Interrupt 2 von P2.9**

Interrupt-Service-Routine (ISR) Jeweiliger Interrupt-Handler muss ins eigene Programm kopiert werden!

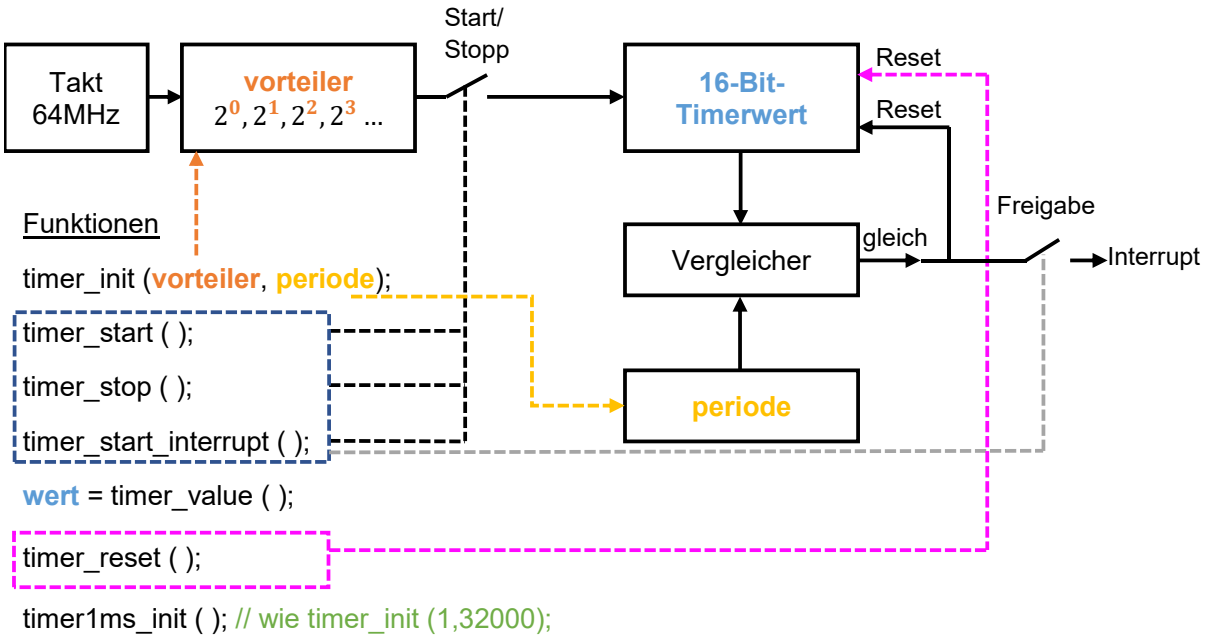
Funktion	Kommentar
void ERU0_2_IRQHandler(**void**) // ext. Int an P2.10 { // Interrupt-Service }	**// Interrupt-Service-Routine (ISR) für ext. Interrupt 1 von P2.10**
void ERU0_3_IRQHandler(**void**) // ext. Int an P2.9 { // Interrupt-Service }	**// Interrupt-Service-Routine (ISR) für ext. Interrupt 2 von P2.9**

```
// Beispiel: Test ext. Interrupt, Taster an P2.10
#include <XMC1100-Lib.h>// Funktionsbibliothek für XMC1100
uint8_t zaehler;
int main(void)
{
  ext_interrupt1_init(RE);  // steigende Flanke
  ext_interrupt1_enable( ); // Freigabe
  port_init(P0,OUTP);       // LEDs
  while(1)                  // Endlos
  {
    port_write(P0,zaehler);     // LEDs zählen Interrupts
  }
}
// steigende Flanke an P2.10
// (Taster loslassen, da 0-aktiv)
void ERU0_2_IRQHandler (void )
{
   zaehler++;
}
```

4.6 Timer

4.6.1 Blockschaltbild

Vereinfachte Timer-Darstellung



4.6.2 Timer-Funktionen

Parameterliste:

Parameter	Eintrag Parametervariablen
vorteiler	0 … 15 ergibt Teilung mit $2^{vorteiler}$
periode	1 … 65535, 64 MHz-Takt

Funktion	Kommentar
timer_init (uint8_t vorteiler, uint16_t periode);	// Timer initialisieren
timer1ms_init (void);	// Timer initialisieren für 1ms, wie timer_init (1,32000);
timer_start (void);	// Timer *ohne* Interrupt starten
timer_start_interrupt (void);	// Timer *mit* Interrupt starten
timer_stop (void);	// Timer stoppen ohne Reset
uint16_t **timer_value** (void);	// Timerwert lesen
timer_reset (void);	// Timer rücksetzen
timer_waitFlag (void);	// Warten bis Timer abgelaufen ist, Polling)

Interrupt-Service-Routine (ISR)

Interrupt-Handler muss ins eigene Programm kopiert werden!
Hinweis: Alle Interrupts haben gleiche Priorität!

Funktion	Kommentar
void CCU40_3_IRQHandler (void) // Timer-Interrupt nach eingestellter Periodendauer { // Interrupt-Service }	// Interrupt-Service-Routine (ISR) für Timer

```
// Beispiel: LEDs zählen im Sekundentakt
#include <XMC1100-Lib.h>     // Funktionsbibliothek
uint16_t zaehler;
int main(void)
{
  timer_init(10,62500);      // 64MHz/2^10/62500=1Sek
  timer_start_interrupt();
  port_init(P0,OUTP);        // LEDs
  while(1)                   // Endlos
  {
    port_write(P0,zaehler); // LEDs zählen
  }                          // im Sekundentakt
}
// jede Sekunde:
void CCU40_3_IRQHandler (void )
{
   zaehler++;
}
```

4.6.3 Zeitmessung

Vorteiler gemäß Tabellenauswahl

Funktion	Kommentar
timer_init (vorteiler, 0xFFFF);	// Timer initialisieren

Wert	Takt nach Vorteiler	Timertakt	Periode max.
0	64 MHz / 1	15,625 nSek	1, 024 mSek
1	64 MHz / 2	31,25 nSek	2,1 mSek
2	64 MHz / 4	62,5 nSek	4,1 mSek
3	64 MHz / 8	125 nSek	8,2 mSek
4	64 MHz / 16	250nSek	16,4 mSek
5	64 MHz / 32	500 nSek	32,8 mSek
6	64 MHz / 64	1 uSek	64,5 mSek
7	64 MHz / 128	2 uSek	134,2 mSek
8	64 MHz / 256	4 uSek	268,4 mSek
9	64 MHz / 512	8 uSek	536,9 mSek
0xA	64 MHz / 1024	16 uSek	1,05 Sek
0xB	64 MHz / 2048	32 uSek	2,1 Sek
0xC	64 MHz / 4096	64 uSek	4,2 Sek
0xD	64 MHz / 8192	128 uSek	8,4 Sek
0xE	64 MHz / 16384	256 uSek	16,8 Sek
0xF	64 MHz / 32768	512 uSek	33,6 Sek

Formel für berechnen der Zeit, bis Interrupt ausgelöst wird:

$$Zeit_{Bis\ Interrupt} = \frac{1}{F} \rightarrow F = \frac{64MHz}{2^{vorteiler} \cdot Periode}$$

```
// Beispiel 1: Berechnen der Zeit 1 Sekunde
```

$$Zeit_{Bis\ Interrupt} = \frac{1}{F} \rightarrow F = \frac{64MHz}{2^{vorteiler} \cdot Periode} = \frac{64MHz}{2^{10} \cdot 63500} = 0{,}999\ Hz \rightarrow 1\ Sekunde$$

```
// Beispiel 2: Berechnen der Periode für 2,5 Sekunden mit Vorteiler=10
```

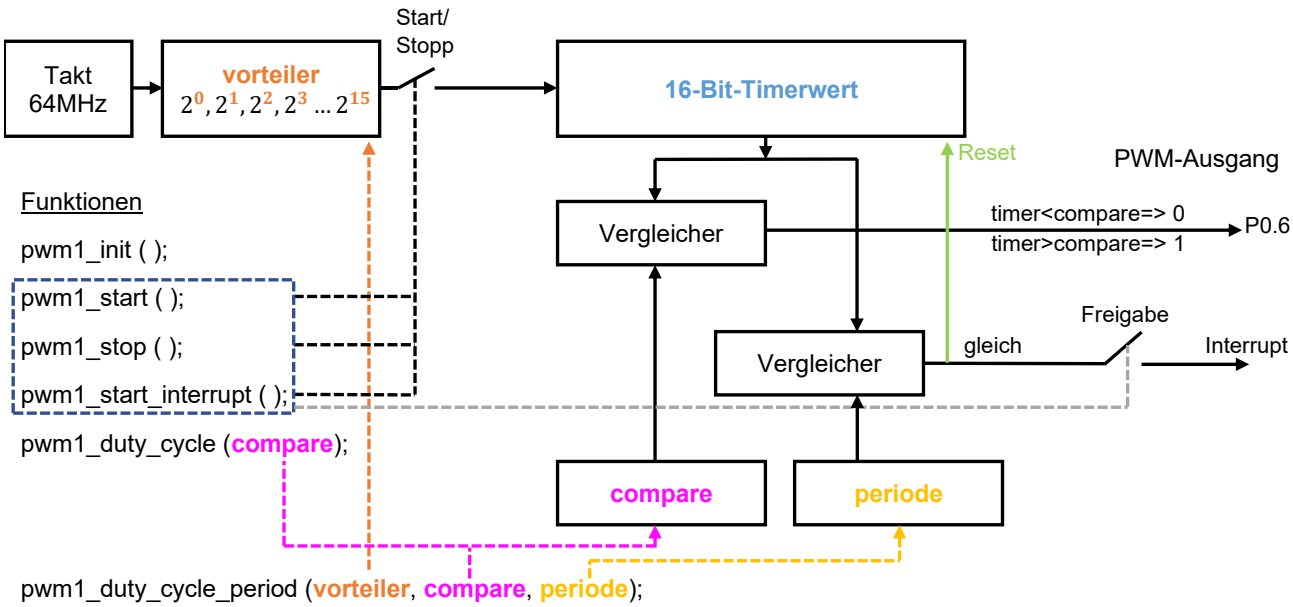
$$F = \frac{1}{t} = 0{,}4Hz \rightarrow Periode = \frac{64MHz}{2^{vorteiler} \cdot F} = \frac{64MHz}{2^{12} \cdot 0{,}4Hz} = 39062{,}5 \rightarrow 39063$$

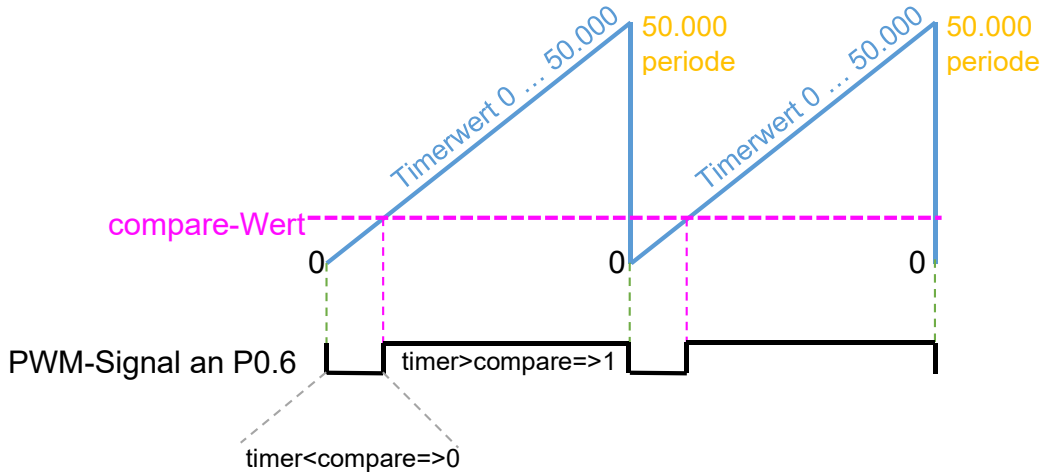
```
// Beispiel: Timer Messfunktion
#include <xmc1100-lib.h>
#define gedrueckt 0                    // lowaktive Taster
uint16_t timer_wert,zeit;
int main(void){
bit_init(P2,9,INP);                   // Start-Taster
bit_init(P2,2,INP);                   // Stopp-Taster
timer_init(0xC,0xFFFF);               // 16uSek Timertakt; 4,2 Sek Periode max!!!
while(1){
   timer_reset();                      // rücksetzen
  while (bit_read(P2,9)!= gedrueckt); // warten start
  timer_start();                       // Timer starten
  while (bit_read(P2,2)!= gedrueckt); // warten stopp
  timer_stop();                        // Timer anhalten
  timer_wert = timer_value();          // Wert auslesen
  zeit = timer_wert*64/1000;           // Zeit in ms
  }
}
```

4.7 Pulsweitenmodulation PWM

3 digitale PWM-Ausgänge (Timer CCU40_CC40 bis CCU42_42 werden als 16-Bit-Timer verwendet!)

Vereinfachte Darstellung: Timer im 16-Bit-PWM-Betrieb





Parameterliste:

Parameter	Eintrag Parametervariablen
vorteiler	0 … 15 ergibt Teilung mit $2^{vorteiler}$
periode	1 … 65535, 64 MHz-Takt
compare	0 … **periode**

4.7.1 PWM1 Ausgang P0.6

Funktion	Kommentar
pwm1_init (void);	// PWM1 initialisieren
pwm1_init_8 (void);	// PWM1 8-Bit-Ausgabe initialisieren
pwm1_start_interrupt (void);	// Ausgabe starten mit Interrupt nach jeder Periode.
pwm1_start (void);	// Ausgabe starten ohne Interrupt
pwm1_stop (void);	// Ausgabe stoppen
pwm1_duty_cycle (uint16_t compare);	// 16-Bit-Comparewert einstellen, der den Tastgrad des PWM-Signals einstellen
pwm1_duty_cycle_period (uint8_t vorteiler, uint16_t compare, uint16_t periode);	// 16-Bit-Comparewert (Tastgrad) und Periodendauer des PWM-Signals einstellen
pwm1_duty_cycle_period_8 (uint8_t vorteiler, uint16_t compare, uint16_t periode);	// 8-Bit-Comparewert (Tastgrad) und Periodendauer des PWM-Signals einstellen

Interrupt-Service-Routine (ISR) — Interrupt-Handler muss ins eigene Programm kopiert werden!
Hinweis: Alle Interrupts haben gleiche Priorität!

Funktion	Kommentar
void CCU40_0_IRQHandler (void) // Timer-Interrupt nach eingestellter Periodendauer { // Interrupt-Service }	// Interrupt-Service-Routine (ISR) für PWM1

```
// Beispiel: 16-Bit-PWM-Signal an P0.6, Tastgrad mit Poti einstellen
#include <xmc1100-lib.h> // Funktionsbibliothek
int main(void)
{
  uint16_t wert;
  pwm1_init();  // PWM1 an P0.6 initialisieren
  pwm1_start(); //Ausgabe starten
  adc_init();  // ADC initialisieren für Poti
while(1U)
  {
    wert = (adc_in(0)<<4);// 12-Bit => 16 Bit
    pwm1_duty_cycle(wert);// 16-Tastgrad
  }//while
}//main
```

4.7.2 PWM2 Ausgang P0.7

Funktion	Kommentar
pwm2_init (**void**);	**// PWM2 initialisieren**
pwm2_init_8 (**void**);	**// PWM2 8-Bit-Ausgabe initialisieren**
pwm2_start_interrupt (**void**);	**// Ausgabe starten mit Interrupt nach jeder Periode.**
pwm2_start (**void**);	**// Ausgabe starten ohne Interrupt**
pwm2_stop (**void**);	**// Ausgabe stoppen**
pwm2_duty_cycle (uint16_t **compare**);	**// 16-Bit-Comparewert einstellen, der den Tastgrad des PWM-Signals einstellen**
pwm2_duty_cycle_period (uint8_t **vorteiler**, uint16_t **compare**, uint16_t **periode**);	**// 16-Bit-Comparewert (Tastgrad) und Periodendauer des PWM-Signals einstellen**
pwm2_duty_cycle_period_8 (uint8_t **vorteiler**, uint16_t **compare**, uint16_t **periode**);	**// 8-Bit-Comparewert (Tastgrad) und Periodendauer des PWM-Signals einstellen**

Interrupt-Service-Routine (ISR)	Interrupt-Handler muss ins eigene Programm kopiert werden! Hinweis: Alle Interrupts haben gleiche Priorität!

Funktion	Kommentar
void CCU40_1_IRQHandler (**void**) // Timer-Interrupt nach eingestellter Periodendauer { // Interrupt-Service }	**// Interrupt-Service-Routine (ISR) für PWM2**

4.7.3 PWM3 Ausgang P0.8

Funktion	Kommentar
pwm3_init (**void**);	**// PWM3 initialisieren**
pwm3_init_8 (**void**);	**// PWM3 8-Bit-Ausgabe initialisieren**
pwm3_start_interrupt (**void**);	**// Ausgabe starten mit Interrupt nach jeder Periode.**
pwm3_start (**void**);	**// Ausgabe starten ohne Interrupt**
pwm3_stop (**void**);	**// Ausgabe stoppen**
pwm3_duty_cycle (uint16_t **compare**);	**// 16-Bit-Comparewert einstellen, der den Tastgrad des PWM-Signals einstellen**
pwm3_duty_cycle_period (uint8_t **vorteiler**, uint16_t **compare**, uint16_t **periode**);	**// 16-Bit-Comparewert (Tastgrad) und Periodendauer des PWM-Signals einstellen**
pwm3_duty_cycle_period_8 (uint8_t **vorteiler**, uint16_t **compare**, uint16_t **periode**);	**// 8-Bit-Comparewert (Tastgrad) und Periodendauer des PWM-Signals einstellen**

Interrupt-Service-Routine (ISR)	Interrupt-Handler muss ins eigene Programm kopiert werden! Hinweis: Alle Interrupts haben gleiche Priorität!

Funktion	Kommentar
void CCU40_2_IRQHandler (**void**) // Timer-Interrupt nach eingestellter Periodendauer { // Interrupt-Service }	**// Interrupt-Service-Routine (ISR) für PWM3**

4.8 Serielle Schnittstelle UART

4.8.1 UART1

UART1 RxD auf P1.3 (nur zur Beobachtung, kein Empfang)
TxD auf P1.2
über USB als virtuelle Schnittstelle COM x am PC verwendbar

Parameterliste:

Parameter	Eintrag Parametervariablen
baudrate	300,1200,9600...115000, Standard 9600, 8N1
enable_interrupt	enable_interrupt = RE_INT = 1 Interrupt bei Receive, Empfang enable_interrupt = NO_RE_INT = 0 kein Interrupt bei Receive, Empfang

Funktion	Kommentar
uart1_init (uint32_t baudrate, uint8_t enable_interrupt);	// Serielle Schnittstelle UART1 initialisieren
uint8_t **uart1_get** (void);	// Abrufen [UART1] des ersten empfangenen Byte (0 ... 255) 0 => Byte = 0 oder kein Zeichen empfangen!
uart1_put (uint8_t value);	// Schreiben von 1 Byte an P1.2
uart1_print (uint8_t *text);	// Schreiben eine nullterminierte Zeichenfolge
uint8_t **uart1_char_received** (void);	// Abfrage von UART1 an P1.3 bzw. vom PC, ob ein Zeichen vorhanden ist, wenn kein Interruptbetrieb 1 = Zeichen empfangen 0 = kein Zeichen empfangen

Interrupt-Service-Routine (ISR) Interrupt-Handler muss ins eigene Programm kopiert werden!
Hinweis: Alle Interrupts haben gleiche Priorität!

Funktion	Kommentar
void USIC0_1_IRQHandler // 1 Byte empfangen von UART1 { // Interrupt-Service }	// Interrupt-Service-Routine (ISR) für UART1

```
// Beispiel: UART1-Test ohne Interrupt
#include <XMC1100-Lib.h>
uint8_t wert;
int main(void)
{
  port_init(P0,OUTP);                  // LEDs
  uart1_init(9600,NO_RE_INT);          // 9600 Bps, kein Interrupt
  while(1U)
  {
    if (uart1_char_received()==1) {
       wert = uart1_get();             // empf. Zeichen speichern
       uart1_put(wert);                // und wieder zurück senden
    }//if
    port_write(P0,wert);               // Zeichen dual anzeigen
  }//while
}//main
```

4.8.2 UART2

UART2 RxD auf P1.0
TxD auf P1.1

Parameterliste:

Parameter	Eintrag Parametervariablen
baudrate	300,1200,9600...115000, Standard 9600, 8N1
enable_interrupt	enable_interrupt = RE_INT = 1 Interrupt bei Receive, Empfang enable_interrupt = NO_RE_INT = 0 kein Interrupt bei Receive, Empfang

Funktion	Kommentar
uart2_init (uint32_t baudrate, uint8_t enable_interrupt);	// Serielle Schnittstelle UART2 initialisieren
uint8_t **uart2_get** (void);	// Abrufen [UART2] des ersten empfangenen Byte (0 ... 255) 0 => Byte = 0 oder kein Zeichen empfangen!
uart2_put (uint8_t value);	// Schreiben von 1 Byte an P1.1
uart2_print (uint8_t *text);	// Schreiben eine nullterminierte Zeichenfolge
uint8_t **uart2_char_received** (void);	// Abfrage von UART2 an P1.0 bzw. vom PC, ob ein Zeichen vorhanden ist, wenn kein Interruptbetrieb 1 = Zeichen empfangen 0 = kein Zeichen empfangen

Interrupt-Service-Routine (ISR) Interrupt-Handler muss ins eigene Programm kopiert werden!
Hinweis: Alle Interrupts haben gleiche Priorität!

Funktion	Kommentar
void USIC0_0_IRQHandler // 1 Byte empfangen von UART12{ // Interrupt-Service }	// Interrupt-Service-Routine (ISR) für UART2

```
// Beispiel: UART2-Test mit Interrupt
#include <XMC1100-Lib.h>        // Funktionsbibliothek
uint8_t wert;
int main(void)
{
  port_init(P0,OUTP);           // LEDs
  uart2_init(9600,RE_INT);      // Interrupt bei Empfang
  while(1U)
  {
    port_write(P0,wert);        //empf. Zeichen dual anzeigen
  }
}
// Interrupt-Service-Routine 1 Byte empfangen UART1
void USIC0_0_IRQHandler(void)
  {
   wert = uart2_get();          // 1 Zeichen aus Puffer abholen
   uart2_put(wert);             // und wieder zurück senden
  }
```

4.9 I²C-Bussystem

Schnittstellenroutine zur Kommunikation mit I²C-Bus Komponenten
Nicht gleichzeitig mit UART2 verwendbar!

SDA auf P1.0
SCL auf P1.1

Funktion	Kommentar
i2c_init (void)	// Initialisierung I2C-Bus
i2c_delay (void)	// Zeitverzögerung zur Verlangsamung der Datenübertragungsrate i=2 bis i=100. Wählen je nach I2C-IC und Pull-Up-Widerstand
i2c_start (void)	// Startbedingung I2C-Bus
i2c_stop (void)	// Stoppbedingung I2C-Bus
i2c_write (uint8_t value)	// Byte ausgeben an I2C-Bus, Rückgabewert: ack = ACK/NACK
i2c_read (uint8_t ack)	// Byte einlesen von I2C-Bus

```
// Beispiel: I2C-Test
#include <XMC1100-Lib.h>      // Funktionsbibliothek
#define adresse   0b01110010
#define ack_anzeige 4
uint8_t ack_bit;                  // Ack-Bit Empfangen-> 0
int main(void)                    // Hauptprogramm
{
  i2c_init();                     // I2C initialisieren
  bit_init(P1,ack_anzeige,OUTP);  // ACK LED aus, NACK LED an
  bit_write(P1,ack_anzeige,1);    // zuerst NACK
  while(1U)                       // Endlosschleife
  {
     i2c_start();                 // Startbedingung
     i2c_write(adresse);          // PortExpander TypA Adr 001
     ack_bit=i2c_write(0x55);     // Bit-Kombination anzeigen
     i2c_stop();                  // Stoppbedingung
     bit_write(P1,ack_anzeige,ack_bit); // ACK/NACK anzeigen an LED
  }//while
}//main
```