Inhaltsverzeichnis

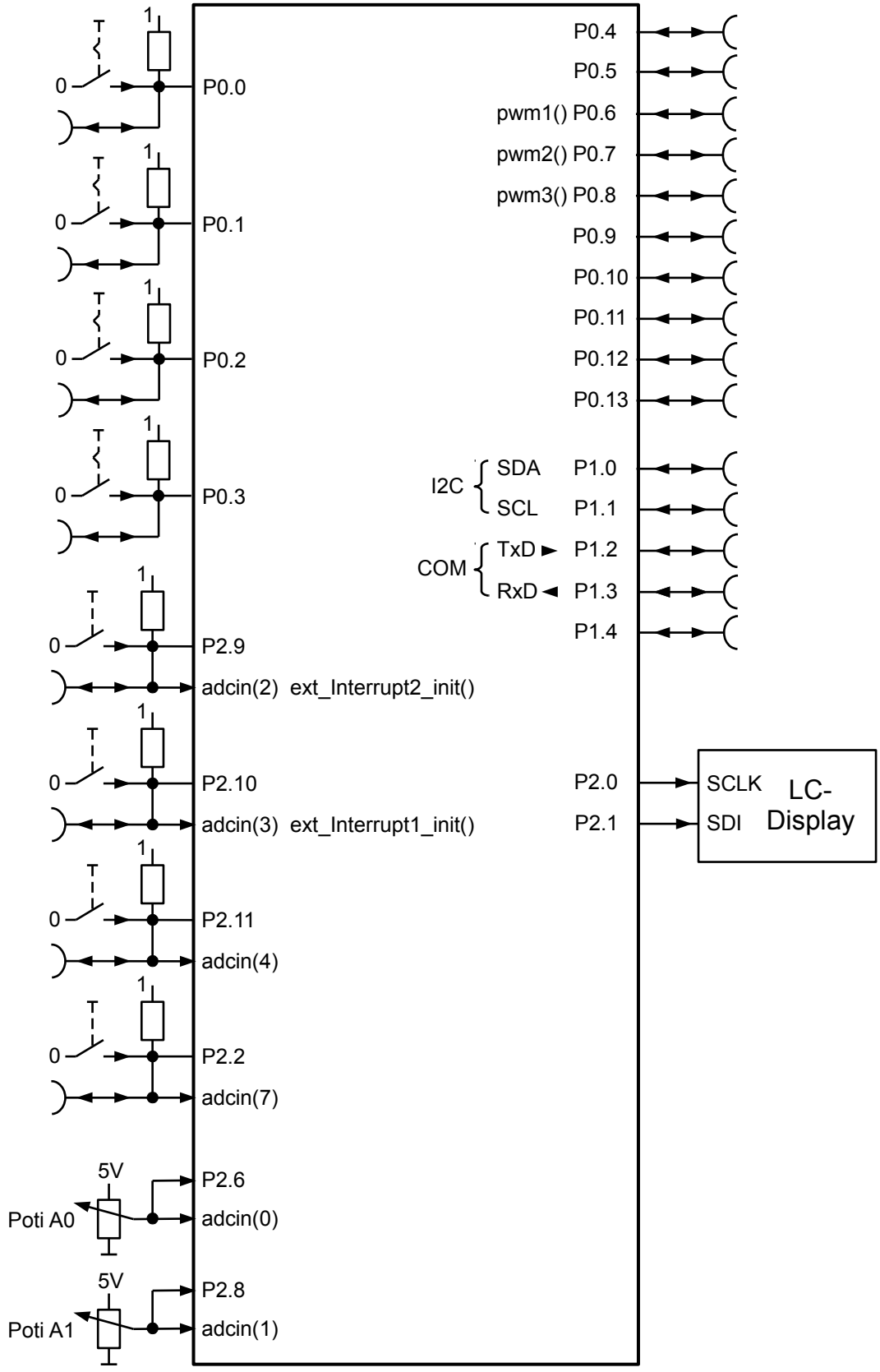
Vielen Dank an Reinhold Birk, Gottlieb-Daimler-Schule 2 für die Bereitstellung der Experimentierplatine und der Bibliothek XMC1100-Lib sowie das Expertenwissen in jeder Situation.

1 XMC1100-Trainer

1.1 Pinbelegungen

Portbit		Buchse	beachte!	Schalter /Taster	Sonderfunktion	Sonderfunktionen
P0.0	LED	Digital IN / OUT	<= Schalter auf 1!	Schalter lowaktiv		
P0.1	LED	Digital IN / OUT	<= Schalter auf 1!	Schalter lowaktiv		
P0.2	LED	Digital IN / OUT	<= Schalter auf 1!	Schalter lowaktiv		
P0.3	LED	Digital IN / OUT	<= Schalter auf 1!	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			SDA (I2C)	
P1.1	LED	Digital IN / OUT			SCL (I2C)	
P1.2		TxD	Kommunikation über USB		TxD (COM)	
P1.3		RxD	Kommunikation über USB		RxD (COM)	
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	<=Taster nicht gedrückt	Taster lowaktiv	adcin(2)	ext_Interrupt2_init()
P2.10		Analog / Digital IN	<=Taster nicht gedrückt	Taster lowaktiv	adcin(3)	ext_Interrupt1_init()
P2.11		Analog / Digital IN	<=Taster nicht gedrückt	Taster lowaktiv	adcin(4)	

1.2 Blockschaltbild



2 Formelsammlung C/C++

2.1 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

2.2 Operatoren

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

Mathematische Operatoren		Priotität	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)	

2.3 Aufbau eines C-Programms

C unterscheidet zwischen Groß- und Kleinschreibung!

2.3.1 Kommentareingabe

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

2.3.2 Befehlsblock

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

2.3.3 Compileranweisung über zusätzliche Quellcodes mit Funktionen und Deklarationen:

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

2.3.4 Konstantendeklaration

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

2.3.5 Deklaration von globalen Variablen (vor main!)

```
uint16_t adc_wert;            // Wert vom AD-Converter 16 Bit unsigned
```

2.3.6 Deklaration von Funktionen

Entweder gesamte Funktion hier angeben
oder unter main schreiben und dann hier deklarieren, damit der Name in mailn bekannt ist.

2.3.7 Hauptprogramm (Hauptfunktion)

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

2.3.8 Beispiel

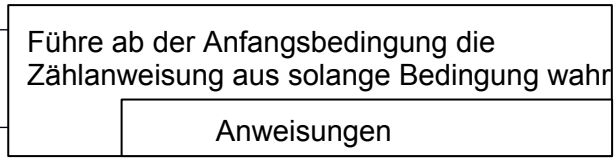
```
// Projectname: 1_Blinklicht_Zeit_veraenderbar
// Blinklicht mit druch Poti veraenderbarer Zeitverzoegerung

#include <XMC1100-Lib.h>          // Bibliothek fuer XMC1100
#define muster1 0x0F              // LED-Bitmuster 1 unter 4 LEDs
#define muster2 0xF0              // LED-Bitmuster 2 obere 4 LEDs

uint16_t adc_wert;                // Wert vom AD-Converter 16 Bit unsigned
int main(void)                    // Hauptprogramm
{
  port_init(P0,OUTP);             // Port0 auf Ausgabe
  adc_init();                     // Analog-Digital-Converter initialisieren
  while(1U)                       // Endlosschleife, immer bei Controllern
  {
        port_write(P0,muster1);   // Muster1 an Port0 ausgeben
        adc_wert = adc_in(0);     // Wert vom ADC -> Zeitverzoegerung
        delay_ms (adc_wert);      // Zeitverzoegerung
        port_write(P0,muster2);   // Muster2 an Port0 ausgeben
        delay_ms (adc_wert);      // Zeitverzoegerung
  }//while
}//main
```

2.4 Schleifen

2.4.1 For-Schleife (zählergesteuerte Schleife)

Erzwingt eine genau berechenbare **Zahl der Wiederholungen**

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



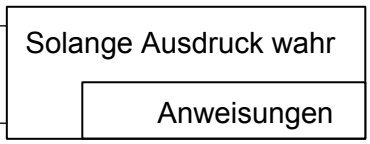
startwert: Anfangswert der Variablen
Bedingung: Schleife wird solange durchlaufen wie die Bedingung wahr ist
Schrittweite: Anweisung zum Erhöhen oder Erniedrigen der Variablen

```
// Beispiel Ausgang 10x invertieren
for (x=10; x!=0; x--)
        {
                ausgang =~ausgang;
        }
// Beispiel Zeitverzögerung
uint32_t  x;
        for (x=100000; x!=0; x--);      // Zeitverzoegerung
```

2.4.2 While-Schleife (kopfgesteuerte Schleife)

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

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



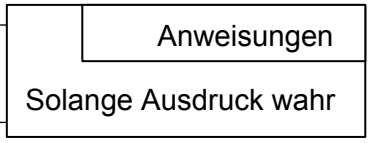
Wenn die am Schleifenanfang stehende **Bedingung nicht gilt,** dann wird die gesamte Schleife übersprungen
Solange die am Schleifenanfang stehende **Bedingung gilt**, wird die Schleife wiederholt.
Die Prüfbedingung steht vor den Anweisungen. Sie heißt deshalb “kopfgesteuerte Schleife”.

```
// Beispiel: Solange der lowaktive Taster an P2.9 gedrückt ist, wird der Ausgang invertiert.

while (bit_read (P2,9) == 0)
        {
                ausgang = ~ausgang;
        }
```

2.4.3 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 Taster an P2.9 gedrückt (= 0) wird.
X = 100;
do
{
        x--;
}
while ((x > 0) && (bit_read (P2,9) == 1));
```

2.5 Programmverzweigungen

2.6 Verzweigung mit if

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

Bedingung J	N
Anweisung1	
Anweisung2	
...	

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

```
// Beispiel: Wenn “taster1” gedrückt ist, soll “ausgang1” eins und „ausgang2“ null werden.
// Drückt man dagegen “taster2”, wird nur “ausgang2” zu eins.
        if (taster1 == 1)
        {                               // Block mit mehreren Anweisungen
            ausgang1 = 1;               // wird ausgeführt, wenn die Bedingung
            ausgang2 = 0;               // hinter if wahr ist
        }

        if (taster2 == 1) ausgang2 = 1;     // nur eine Anweisung, keine { } nötig
```

Mit **“if - else “** kann **nur zwischen zwei Alternativen** wählen.

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

Bedingung J	N
Anweisung1	Anweisung3
Anweisung2	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
        }
```

2.6.1 Mehrere if-Anweisungen

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

Bedingung1 J	N	
Anweisung1	Bedingung2 J	N
Anweisung2	Anweisung3	Anweisung4
...	...	...

2.7 Fallauswahl mit switch

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

Bedingung: fall1	fall2	default
Anw.1	Anw.3	Anw.5
Anw.2	Anw.4	Anw.6
...	...	...

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!!!**

```
// 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;
                    break;
            case 0x30:
                    ausgang1 = 0;
                    break;
            case 0x40:
                    ausgang1 = ~ ausgang1;
                    break;
            default:
                    ausgang2 = 1;
                    break;
        }
```

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
        unsigned char richtung;
        ....
        switch (richtung)
        {
            case rechts:    motor = rechtskurve; break;
            case links:     motor = linkskurve; break;
            default:        motor = vorwaerts; break;
        }
```

2.8 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
```

2.8.1 Definition von Funktionen

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

```
// Beispiel:
int addieren( uint8_t z1, uint8_t z2)
{       uint16_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)!!**

2.8.2 Funktionsaufruf

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

```
// Beispiel:
int main(void)
{       ...
        value1 = adc_in (1);    // Wert von Kanal1 des ADC holen
}
```

2.8.3 Funktion mit Übergabewert

```
void zeitms (uint32_t sec)                  // 32-Bit-Übergabevariable sec
{
    uint32_t t1,t2;                         // lokale Variable
    for (t1 = sec; t1 != 0; t1--)           // äußere Schleife ms * 1Millisekunde
    {
        for (t2 = 0xFFFFFF; t2 != 0; t2--);  // Wert für 1ms noch testen!
                                            // erzeugt Zeitverzögerung ca. 1s
    }
}

void main (void)
{
    ....                                    // Programm
    zeitms (2);                             // Funktionsaufruf mit Werteübergabe
                                            // hier: 2ms Zeitverzögerung
}
```

3 Funktionsbibliothek für den Controller XMC1100

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

3.1 Verzögerungsfunktionen

Delay
Verzögert den Programmablauf für die angegebene Zeitdauer

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

3.2 Bit- und Byte Ein-/Ausgabe

bit_init (uint8_t **port**, uint8_t **bitnr**, uint8_t **direction**)
uint8_t **value** = bit_read (uint8_t **port**, uint8_t **bitnr**)
bit_write (uint8_t **port**, uint8_t **bitnr**, uint8_t **value**)

port_init (uint8_t **port**, uint8_t **direction**)
uint16_t **value** = port_read (uint8_t **port**)
port_write (uint8_t **port**, uint8_t **value**)

Pararameterliste :
port : P0 , P1 ,P2

bitnr : 0…7
direction : INP = 0, OUTP =1
Achtung : P2 nur Input möglich

Bsp.: Bit-Ein-Ausgabe

```
#include <xmc1100-lib.h>
uint8_t  temp;

void main(void)
{
  bit_init( P2, 11, INP ); // Taster
  bit_init( P0, 0, OUTP ); // LED

  while (1)
    {
    temp = bit_read(P2,11);
    bit_write(P0,0,temp );
    }
}
```

3.3 Analog-Digital Konverter

Sechs analoge Eingänge, je 12 Bit Auflösung

adc_init ( );
uint16_t **value** = adc_in (kanal-nr)

kanal-nr : 0 = AN0 (Poti) → P2.6
1 = AN1 (Poti) → P2.8
2 = AN2 (Buchse) → P2.9
3 = AN3 (Buchse) → P2.10
4 = AN4 (Buchse) → P2.11
7 = AN7 (Buchse) → P2.2

Rückgabewert :
value : 0 … 4095 (12 Bit Auflösung)

Bsp.: ADC

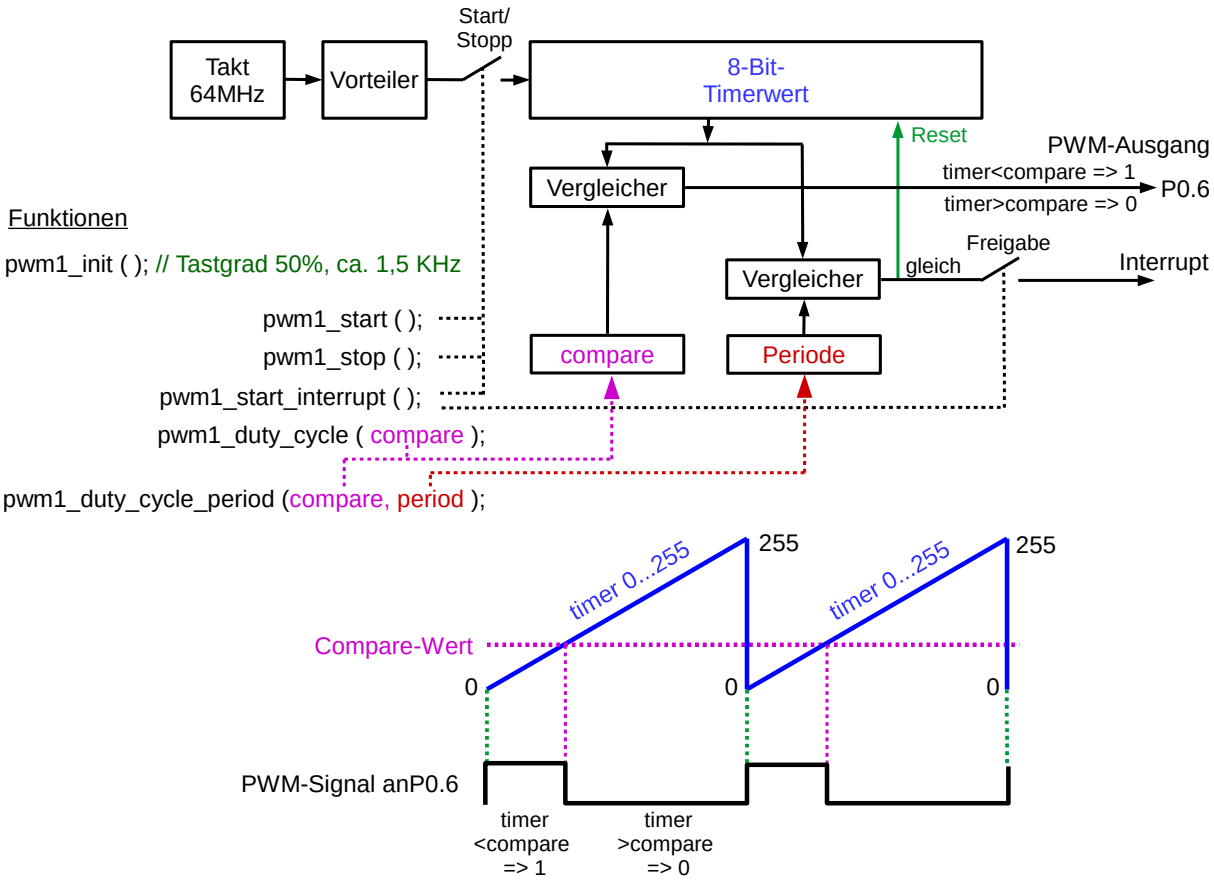
```
#include <xmc1100-lib.h>
uint16_t wert;
void main(void)
{
  adc_init( );
  byte_init( PORT0, OUT );

  while(1)
  {
    wert = adc_in(1);// ADC einlesen
    byte_write( P0, wert >> 4 );
              // 8 Bit OUT
  }
}
```

Achtung : AN2 .. AN7 können ***alternativ*** als ext. Interrupt oder digitaler Input verwendet werden.

3.4 Pulsweitenmodulation PWM

Ein digitaler Ausgang (PWM Out) zur Ausgabe eines pulsweitenmodulierten Signals.
(Timer CCU40_CC40 bis CCU40_CC42 werden als 8-Bit-Timer verwendet!

Vereinfachte Darstellung: Timer im 8-Bit-PWM-Betrieb




```
pwm1_init ( )      // Init für Tastgrad 50% , Ausgabe an P0.6
pwm1_start ( )
pwm1_start_interrupt ( )   // Interrupt nach jeder Periode
pwm1_stop ( )
pwm1_duty_cycle ( uint8_t value )
pwm1_duty_cycle_period (uint8_t compare, uint8_t period)

pwm2_init ( )      // Init für Tastgrad 50% , Ausgabe an P0.7
pwm2_start ( )
pwm2_start_interrupt ( )   // Interrupt nach jeder Periode
pwm2_stop ( )
pwm2_duty_cycle ( uint8_t value )
pwm2_duty_cycle_period (uint8_t compare, uint8_t period)

pwm3_init ( )      // Init für Tastgrad 50% , Ausgabe an P0.8
pwm3_start ( )
pwm3_start_interrupt ( )   // Interrupt nach jeder Periode
pwm3_stop ( )
pwm3_duty_cycle ( uint8_t value )
pwm3_duty_cycle_period (uint8_t compare, uint8_t period)
```

```
Bsp.: PWM
#include <xmc1100-lib.h>

void main(void)
{
  pwm1_init( );

  pwm1_duty_cycle(64);//Tastgrad 25%
  pwm1_start( );
  delay_ms(5000);
  pwm1_stop( );

  while(1);
}
```

Pararameterliste :
value, compare : 0 … 255 (entspricht Tastgrad 0…100%) . Initialwert: 127
period: 0 … 255 (entspricht Periodendauer, compare<period)

3.5 Externer Interrupt

Zwei externe Interrupteingänge: ***P2.9***
P2.10

Achtung: werden ext. Interrupt benutzt können diese Eingänge nicht gleichzeitig Analogeingänge sein.

```
// bei fallender Flanke an P2.10, steigende Flanke an P2.9:
ext_interrupt_init ();        // beide Interrupts initialisieren

// Flanken wählbar wenn einzeln initialisiert:
// flanke=RE=1 Rising Edge ansteigende Flanke
// flanke=FE=0 Falling Edge abfallende Flanke
ext_interrupt1_init ( uint8_t flanke );
ext_interrupt2_init ( uint8_t flanke );

// Interrupts freigeben und sperren:
ext_interrupt1_enable ();          // Int. an P2.10 freigeben
ext_interrupt2_enable ();          // Int an P2.9 freigeben
ext_interrupt1_disable ();         // Int. an P2.10 sperren
ext_interrupt2_disable ();         // Int an P2.9 sperren

// Interruptfunktionen : (ISR)
void ERU0_2_IRQHandler(void) // P2.10
{
}
void ERU0_3_IRQHandler(void) // P2.9
{
}
```

Bsp.: ext. Interrupt

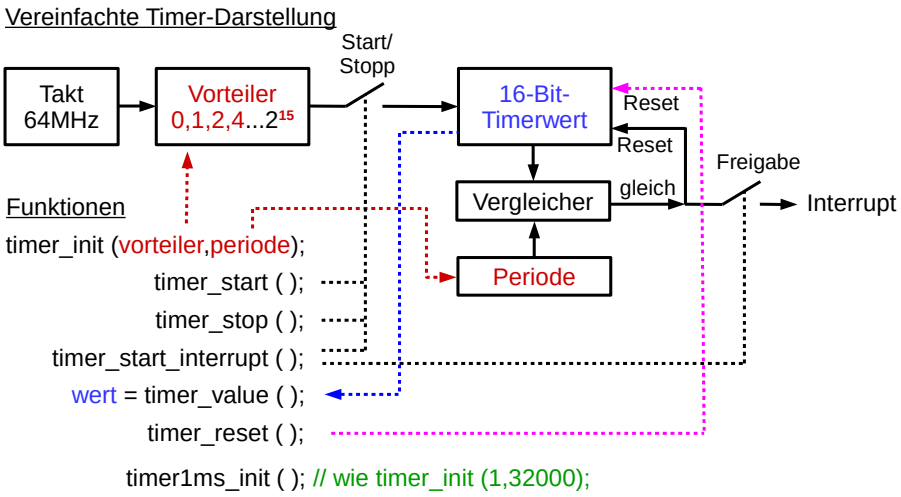
```
#include <xmc1100-lib.h>
uint8_t zaehler;

void main(void)
{
  ext_interrupt1_init(RE);
  ext_interrupt1_enable ( );
  while(1);   // Endlos
}
// steigende Flanke an P2.10:
void ERU0_2_IRQHandler (void )
{
   zaehler++;
}
```

Achtung: die Funktionsnamen. ***void ERU0_3_IRQHandler(void)*** und ***void ERU0_2_IRQHandler(void)*** sind so vom System vordefiniert.

3.6 Timer

3.6.1 Blockschaltbild

benutzt wird CCU40_CC43



3.6.2 Funktionen

```
void timer_init (uint8_t vorteiler, uint16_t periode);     // Timer initialisieren
                                                           // vorteiler 0...15 ergibt Teilung mit 2^vorteiler
                                                           // periode 1...65536, 64 Mhz-Takt
void timer1ms_init ( void );                               // Timer initialisieren für 1ms, wie timer_init (1,32000);
void timer_start(void);                                    // Timer ohne Interrupt starten
void timer_start_interrupt(void)                           // Timer mit Interrupt starten
void timer_stop(void)                                      // Timer stoppen ohne Reset
uint16_t timer_value(void);                                // Timerwert lesen
void timer_reset(void);                                    // Timer rücksetzen

// festgelegter ISR-Funktionsname:
void CCU40_3_IRQHandler (void) // Timer ISR
{
}
```

Hinweis: alle Interrupts haben gleiche Priorität!

Bsp.: Zeitgeber mit Timer-Interrupt

```
#include <xmc1100-lib.h>
uint8_t zaehler;

void main(void)
{
  timer1ms_init( );
  timer_start_interrupt ( );
  while(1);   // Endlos
}
// jede Millisekunde:
void CCU40_3_IRQHandler (void )
{
   zaehler++;
}
```

3.6.3 Zeitmessung

Vorteiler gemäß Tabelle wählen
timer_init (vorteiler, 0xFFFF); **// 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

Bsp.: 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
  }
}
```

3.7 Funktionen für die LCD Anzeige

LC-Display
Routinen zur Ansteuerung eines Textdisplays

```
lcd_init ( )
lcd_clear ( )
lcd_setcursor ( uint8_t row, uint8_t column )
                                    // row (zeile) = 1, 2, … ;
                                    // column (spalte) = 1, 2, …
lcd_print ( uint8_t text[ ] )       // Bsp.: "Hallo Welt"
lcd_char ( uint8_t value )          // 'A', 'B', ':', '?', ….
lcd_byte ( uint8_t value )          // 0 … 255
lcd_int ( uint16_t value )          // 0 … 65535
```

Bsp.: LCD

```
#include <xmc1100-lib.h>
uint8_t meinText[ ] =
"Controller sind toll!";

void main(void)
{
  lcd_init();
  lcd_setcursor(1,1);
  lcd_print(meinText);
  lcd_setcursor(2,1);
  lcd_byte(155);
  lcd_char('V');

  while(1);   // Endlos
}
```

3.8 Funktionen zur Kommunikation

3.8.1 RS232

Serielle Schnittstellenroutinen zur Kommunikation mit einem PC
Es wird ***P1.3 ➔ RxD Controller***
P1.2 ➔ TxD Controller verwendet

```
rs232_init ( )        // 9600 Baud, 1 Startbit, 8 Datenbit, 1 Stopbit

uint8_t value = rs232_get ( )   // Liefert \0, wenn kein Zeichen
                                // empfangen
uint8_t value = rs232_wait_get ( )  // wartet solange, bis ein
                                // Zeichen empfangen wurde
                                // empfangen
uint8_t  rs232_char_received ( ) // liefert 1 wenn Zeichen
                                // empfangen, sonst 0
rs232_put ( uint8_t value )     // Ausgabe eines Bytes
rs232_print ( uint8_t text[ ] )  // Ausgabe eines Strings
```

Pararameterliste :
value: Byte (8 Bit)
text [ ]: Zeichenkette
Bsp.: "Hallo Welt!"

Bsp.: RS232

```
#include <xmc1100-lib.h>
uint8_t  meinText[ ] = "Controller sind toll!";

void main(void)
{
 rs232_init( );

 rs232_print(meinText);
 rs232_print("…aber klar!\n");

 while (rs232_get( ) == '\0');  // Warten auf Zeichen
 rs232_put('A');

 while(1);          // Endlos
}
```

3.8.2 I2C-Bussystem

Schnittstellenroutinen zur Kommunikation mit I2C-Bus Komponenten , es wird ***P1.0 ➔ SDA***
P1.1 ➔ SCL verwendet

```
i2c_init ( )
i2c_start ( )
i2c_stop ( )
uint8_t ack = i2c_write ( uint8_t value )
uint8_t value = i2c_read ( uint8_t ack )
```

Pararameterliste :
ack : ACK = 0, NACK =1
value: Byte (8 Bit)

Bsp.: I2C

```
#include <xmc1100-lib.h>
#define ADDR_R  0x41    //
#define ADDR_W  0x40

uint8_t wert = 0x34;

void main(void)
{
 i2c_init( );
 i2c_start( ) ;
 i2c_write(ADDR_W);
 i2c_write(wert);
 i2c_stop( );

 while(1);          // Endlos
}
```

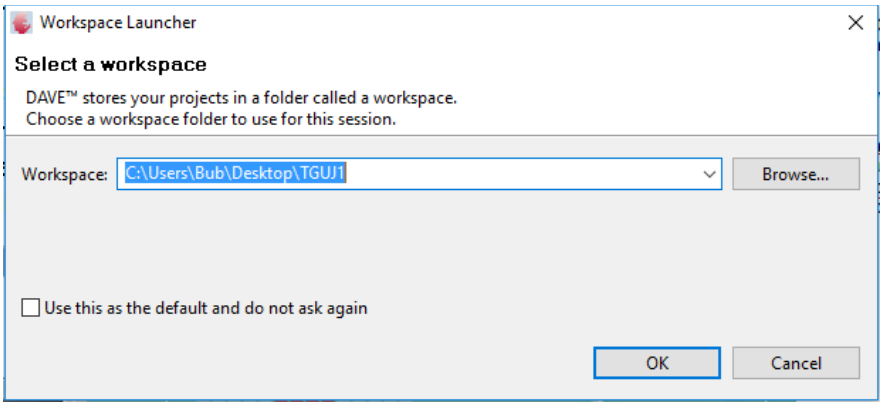
4 Eintwicklungsumgebung Dave4 Einführung

4.1 Projekt anlegen

Die aktuelle Version DAVE4.x.x eignet sich gut für alle Aufgaben mit den XMC – Controllern. DAVE besteht aus einem frei verfügbaren GNU C-Compiler, eingebettet in eine „Eclipse“ Oberfläche.

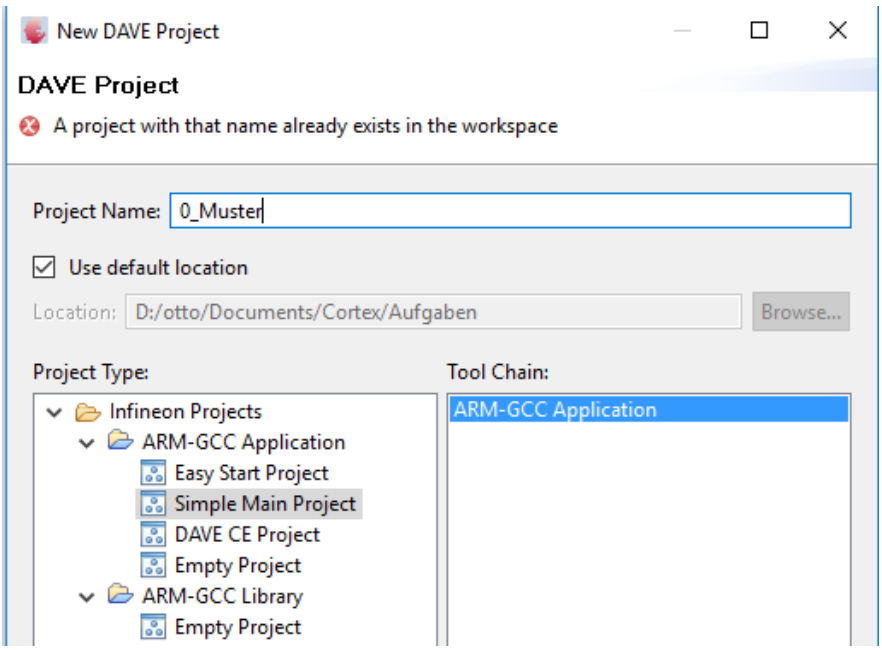
Ein Download der jeweils aktuellen Version ist von der Webseite „www.infineon.com/dave“ möglich.

Beim Start des Programmes muss zuerst ein „Workspace“ angegeben werden. Dies ist ein Ordner auf Ihrem PC, worin anschließend alle Programme, Bibliotheken … abgelegt werden. Legen sie für den Workspace einen Ordner an, welcher nicht über das Netzwerk erreichbar ist, dies erhöht die Arbeits-geschwindigkeit.



In der Schule: Ordner TGUJ1 auf dem Desktop. Nach der Stunde auf eigenen Stick kopieren!

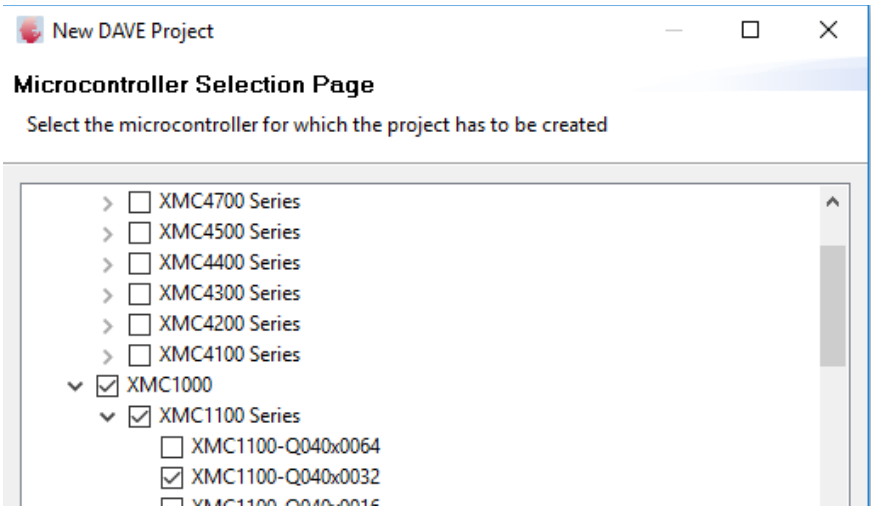
Nach der Angabe des „Workspace“ muss mit
File → New → Dave-Project
ein neues Projekt angelegt werden.
Wir arbeiten zuerst mit „Simple Main Project“ d.h. ohne Unterstützung von „DAVE-App’s“



Geben Sie einen sinnvollen Projektnamen z.B. mit Nummer an.

Achtung: keine Umlaute Ä,Ü

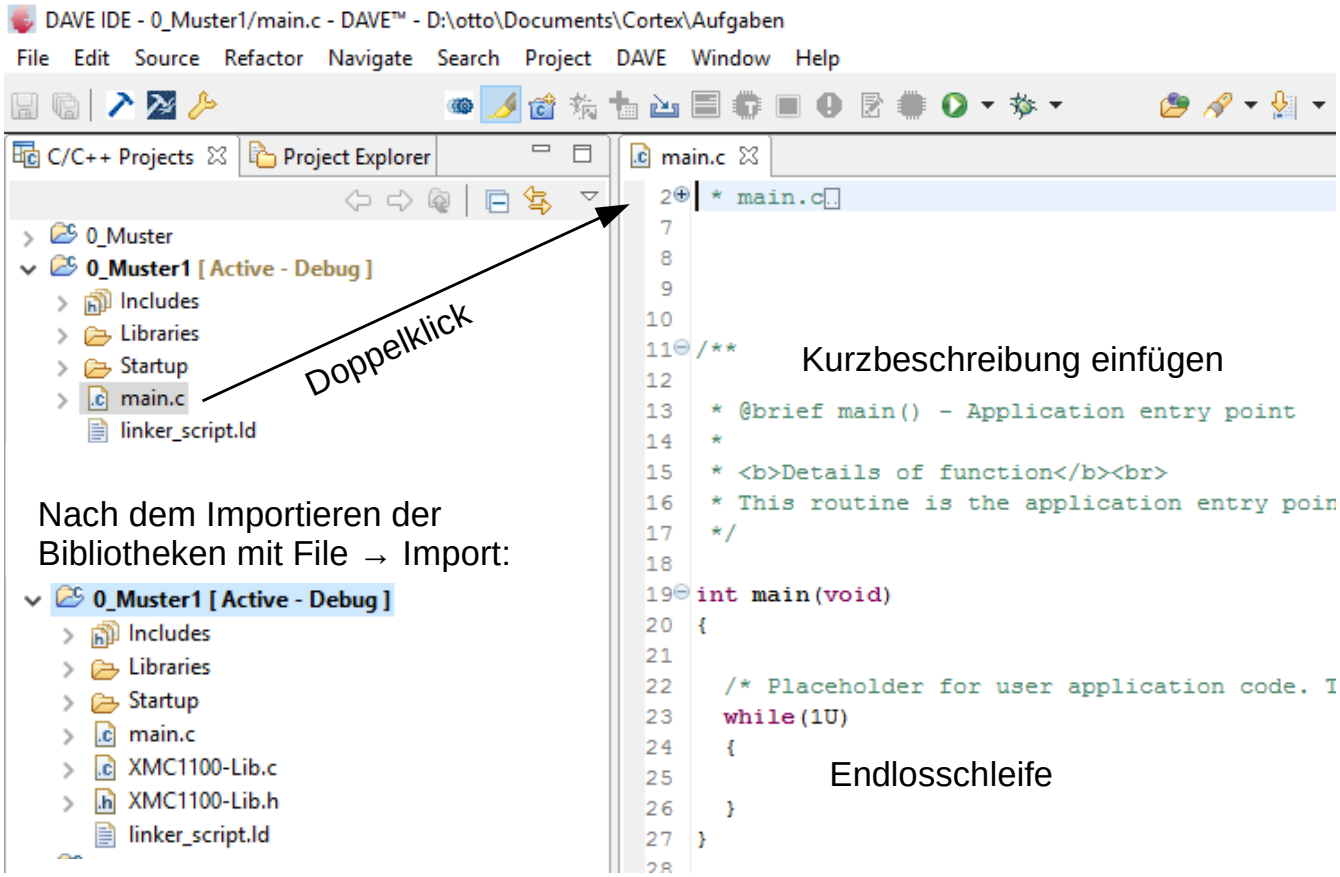
→ Next →
Auswahl des Controllertyps :
XMC1100 Series,
in der Serie 2. von oben



→ Finish

Es wir ein Projekt erzeugt mit

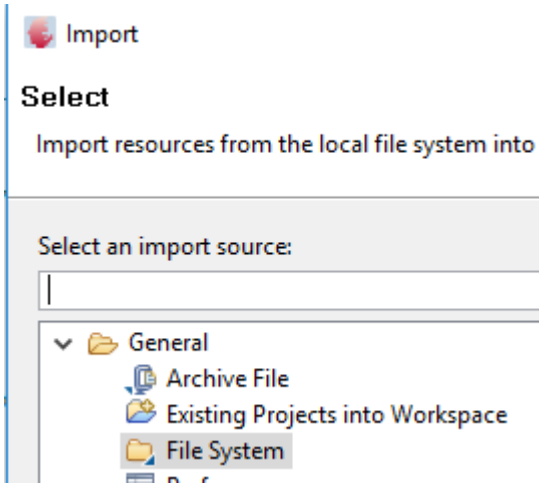
wichtigen Voreinstellungen und Bibliotheken und einem leeren Hauptprogramm main.c:



Wir arbeiten zunächst mit Hilfsfunktionen, da für den Anfänger der Zugriff auf die Komponenten des Controller (z.B. die Ports) „unanschaulich“ ist.

Diese Hilfsfunktionen sind in der Bibliothek XMC1100-Lib zusammengefasst.

4.2 Dateien zum Projekt hinzufügen



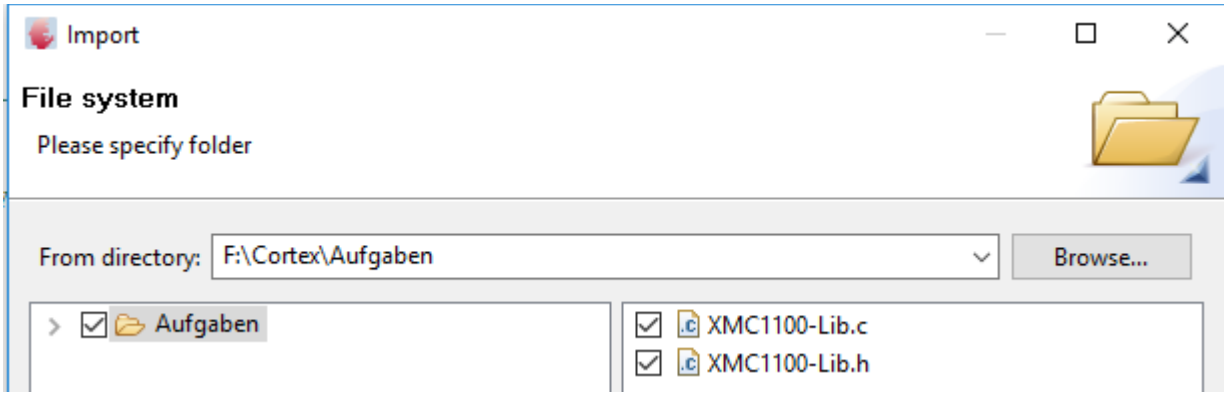
Links auf den Projektnamen klicken, dann:

File → Import → General → File System

→ Next

→ Browse

→ Ordner mit den hinzuzufügenden Dateien suchen



→ hinzuzufügende Dateien auswählen

XMC1100-Lib.c

XMC1100-Lib.h

→ Finish

Nun gehören die Hilfsfunktionen in der Bibliothek XMC1100.c

zum Projekt hinzu.

Im Programm kann man nach #include <XMC1100.h> darauf zugreifen.

4.3 Projekt mit neuem Namen kopieren

Rechtsklick auf Projektname → Copy
Rechtsklick auf freie Fläche unter den Projekten → Paste, neuen Projektnamen angeben

4.4 Aktives Projekt

Nur ein Projekt ist immer aktiv und wird übersetzt und downgeloadet.
Rechtsklick auf den Projektname → Set Active Project

4.5 Programm eingeben

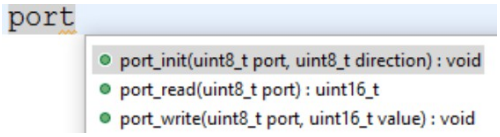
Doppelkick auf main.c des aktiven Projekts.

main.c

```
1  /* Muster Blinklicht
3
4  #include <XMC1100-Lib.h>          // Hilfsfunktionen für XMC1100
5
6  int main(void)                    // Hauptprogramm
7  {
8    port_init(P0,OUTP);             // Port0 auf Ausgabe
9
10   while(1U)                       // Endlosschleife, immer bei Controllern
11   {
12       port_write(P0,0x0F);        // untere 4 LEDs von Port0 einschalten
13       delay_ms (500);             // Zeitverzögerung 0,5s
14       port_write(P0,0xF0);        // obere 4 LEDs von Port0 einschalten
15       delay_ms (500);             // Zeitverzögerung 0,5s
16   } //while(1U)
17 } //main
```

Die Liste der Hilfsfunktionen in XMC1100-Lib finden Sie z.B. duch Doppleklick auf XMC1100.h im Projekt.

Wenn Sie einen Teil der Funktion eingegeben haben, z.B. „port“ und dann <STRG><LEER> drücken, erhalten Sie eine Auswahl aller zur Verfügung stehenden Funktionen
=> auswählen durch Doppelklick
=> einzugebende Parameter werden angezeigt



Sobald Sie die Funktionen im Editor eingegeben haben und mit der Maus über den Text fahren, wird Ihnen die in der Datei XMC1100-Lib.c angegebene Erklärung angezeigt:

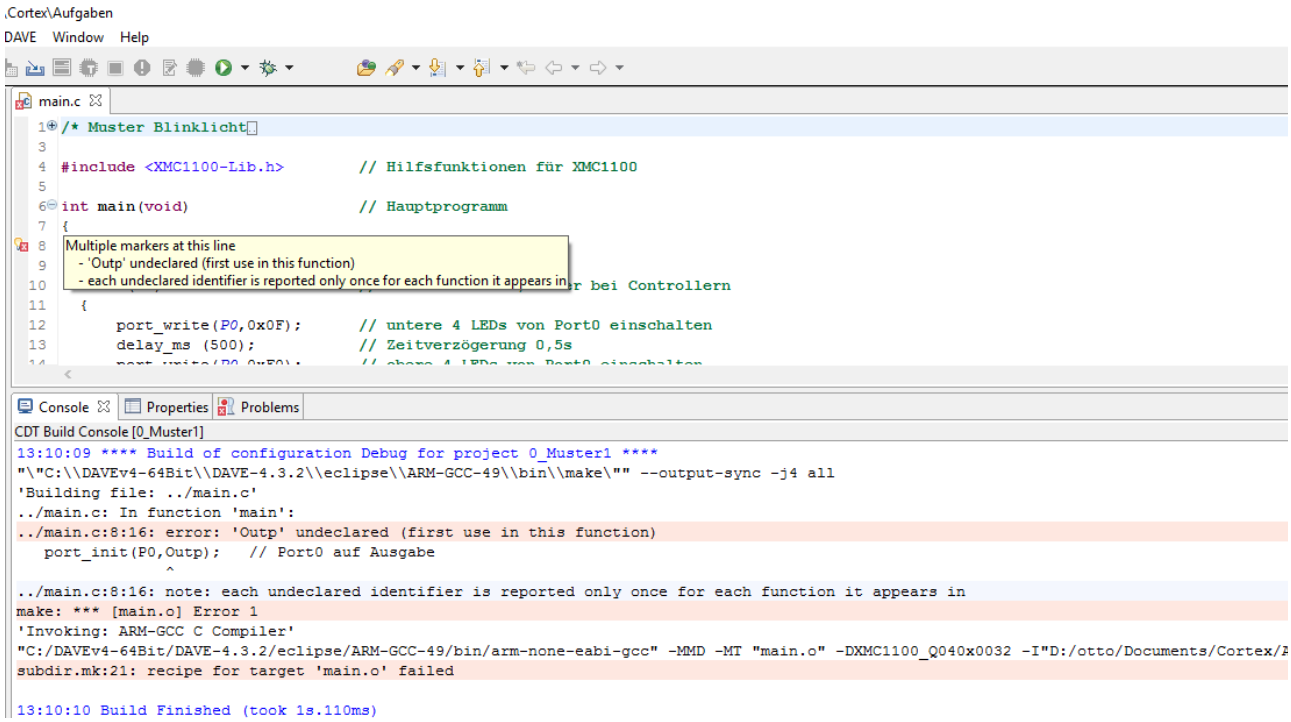
main.c

```
1  /* Muster Blinklicht
3
4  #include <XMC1100-Lib.h>          // Hilfsfunktionen für XMC1100
5
6  int main(void)                    // Hauptprogramm
7  {
8    port_init(P0,OUTP);             // Port0 auf Ausgabe
9    //---------------------------------------------------------------
10   //Gesamtes Port für Ein- oder Ausgabe initialisieren
11   // port: P0,P1,P2 direction: INP 0, OUTP 1
12   //---------------------------------------------------------------
13    void port_init(uint8_t port, uint8_t direction)
14
```

4.6 Compilieren (Übersetzen von C in Maschinensprache)

Compiler-Aufruf mit

Falls ein Fehler auftritt, wird dieser durch ein kleines, rotes Kreuz und einer Wellenlinie in der entsprechenden Zeile angezeigt. Wenn Sie mit der Maus über das Kreuz fahren, erscheint eine Fehlererklärung. Diese und alle weiteren Fehler sehen Sie auch im Consolen-Fenster unten.



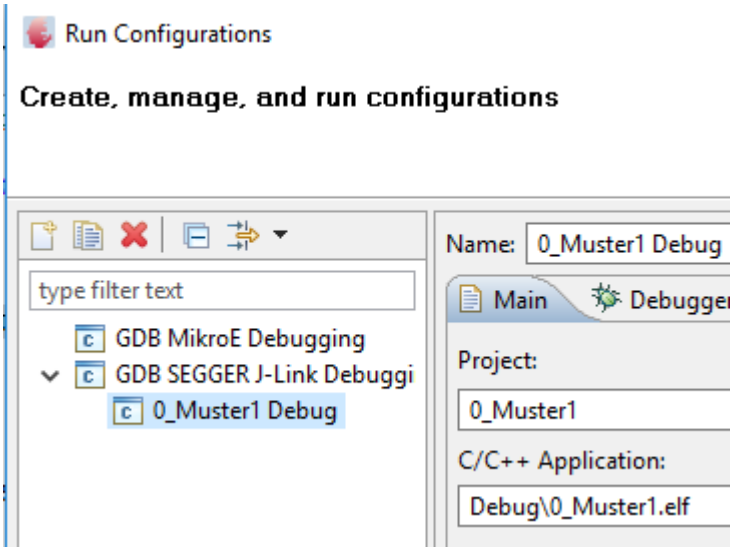
Hier war Outp klein statt OUTP groß geschrieben.

4.7 Download

!!Vor dem Download immer erst compilieren!!

Beim ersten Download des Programms mit 

müssen Sie mit Doppelklick auf GDB Segger J-Link Debugging die Art des Downloads wählen, nichts weiter eintragen → Run



Fall der Download auch nach USB-Stecker ziehen und wieder verbinden und erneutem Download nicht funktionieren sollten, gehen Sie folgendermaßen vor:

Dieses Fenster erhalten sie jederzeit über den kleinen Pfeil neben grünen Pfeil → Run Configurations

Rechtsklick auf den Projektnamen unter GDB Segger J-Link → Delete

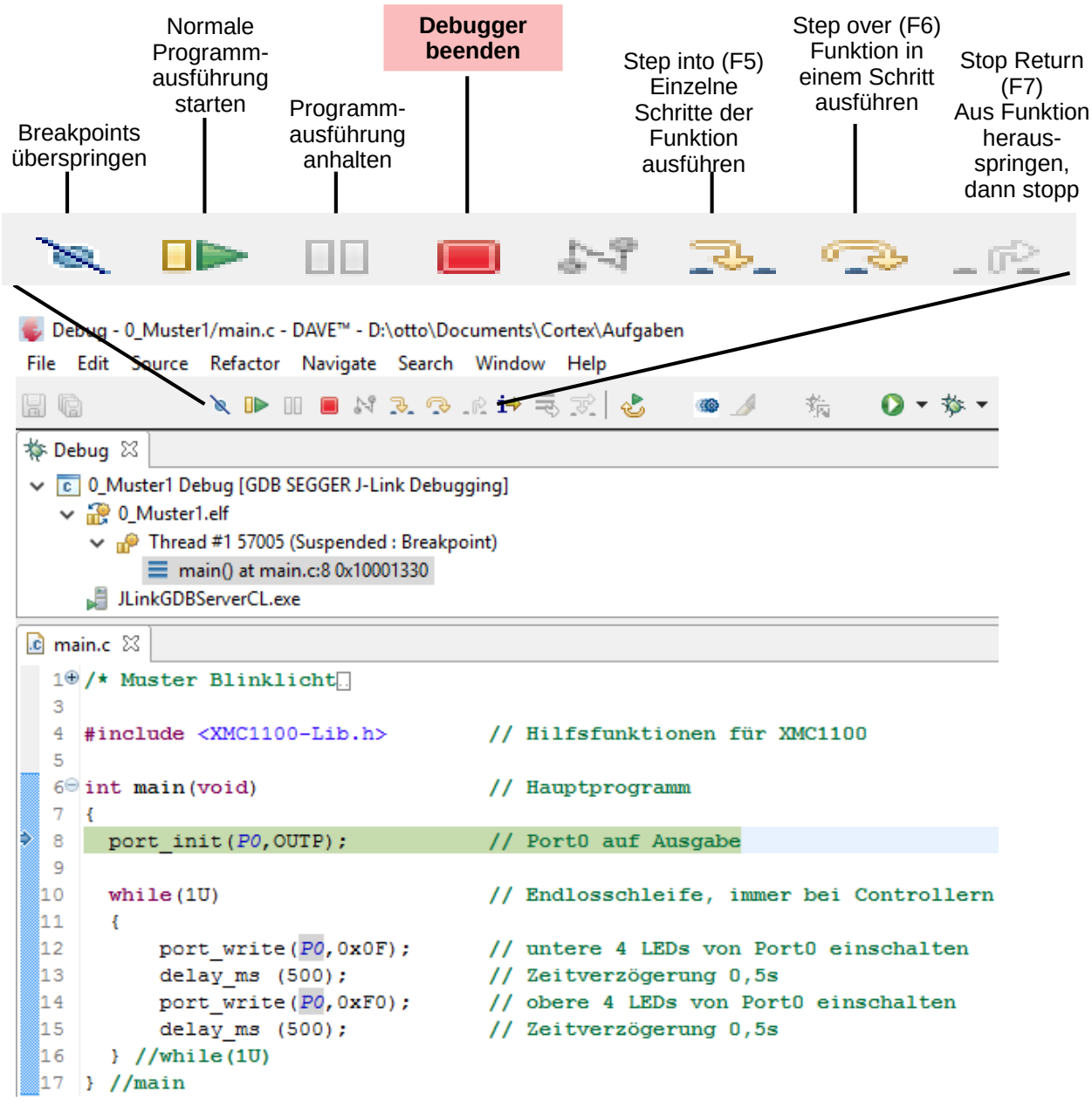
Dann wieder Doppelklick auf GDB Segger J-Link → Run

4.8 Debuggen

Debugger öffnen mit

Beim ersten Mal öffnet sich wie beim Download ein Fester, bei dem die Kommunikationsart gewählte werden muss mit Doppelklick auf GDB Segger J-Link Debugging.

Die Fenster werden nun im Debugger-Modus angeordnet.



Pfeil links und farblich hinterlegte Zeile: Dieser Befehl wird als nächstes ausgeführt.
Stoppstelle einfügen: Rechtsklick auf linken Rand → Toggle Breakpoint

Wichtig: Vor erneutem Download und bei Programmänderung: **Debugger immer beenden**

Zur Änderung des Programms vom Debugger in die IDE wechseln: