Implementierung der Technischen Richtlinie FA205 mit AtmelStudio 7 für ATmega-Mikrocontroller

Inhalt

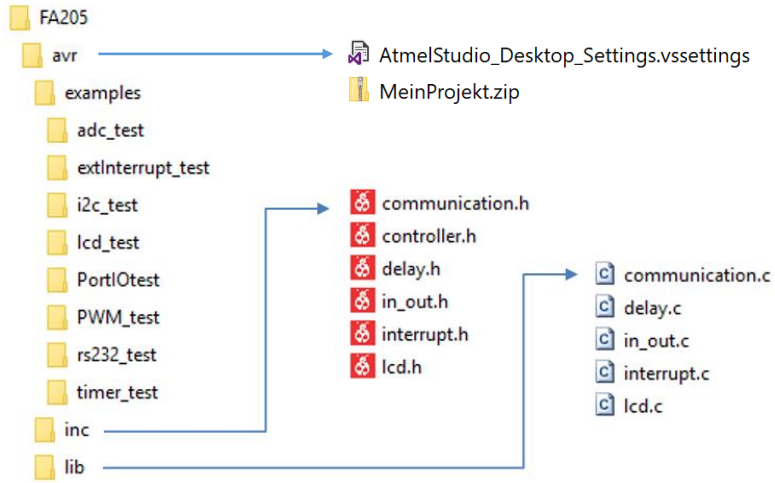
Rolf Rahm

Reichenbach, Februar 2016 (Letzte Änderung: 07.06.2021)

1 Allgemeine Hinweise zu Dateien und Verzeichnissen

Diese Anleitung dient zur Installation von Atmel-Studio inklusive der FA205-Bibliotheken auf einem einzelnen (privaten) Windows-PC[1]. Es empfiehlt sich, diese Datei gut durchzulesen und Schritt für Schritt abzuarbeiten. Gegebenenfalls sollte das Dokument ausgedruckt werden.

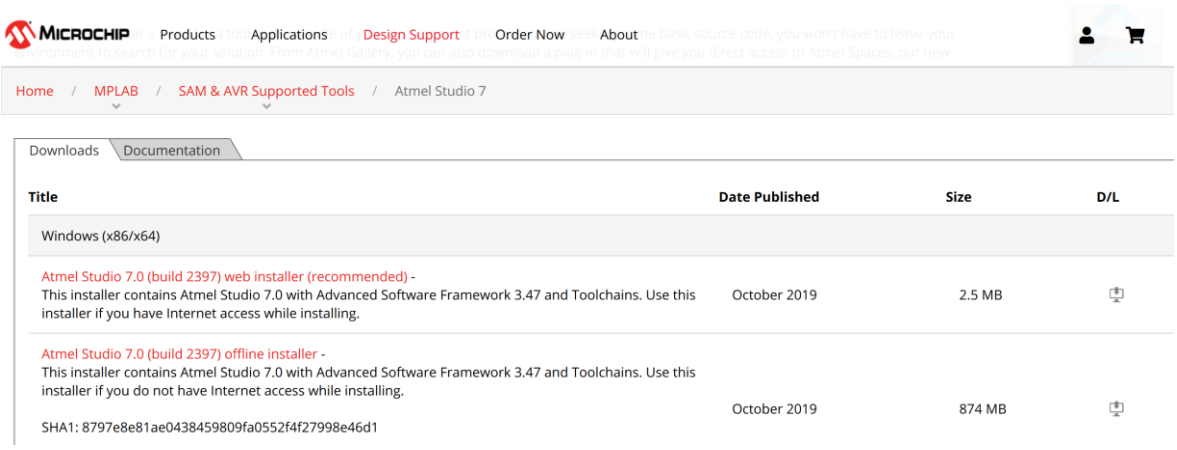
Ein neues Projekt, basierend auf den Vorgaben der Technischen Richtlinie FA205, benötigt eine festgelegt Verzeichnisstruktur. Die Bibliotheksverzeichnisse (inc, lib, example) sollten dazu immer im Stammverzeichnis (C:\FA205\...) auf dem lokalen Rechner abgelegt werden:



In der Projekt-Vorlage ist insbesondere der Pfad für die Header-Files im Verzeichnis …\inc hinterlegt. Liegt der Include-Pfad nicht auf C:\FA205\inc, muss dies im Projekt angepasst werden, oder eine eigene Vorlagen-Datei erstellt werden (siehe Kapitel 3.4).

2 Hinweise zur Installation von Atmel Studio7

Der Installer für AtmelStudio ist kostenlos von der Microchip-Homepage (https://www.microchip.com/mplab/avr-support/atmel-studio-7) erältlich. Es wird empfohlen, den Web-Installer zu verwenden.



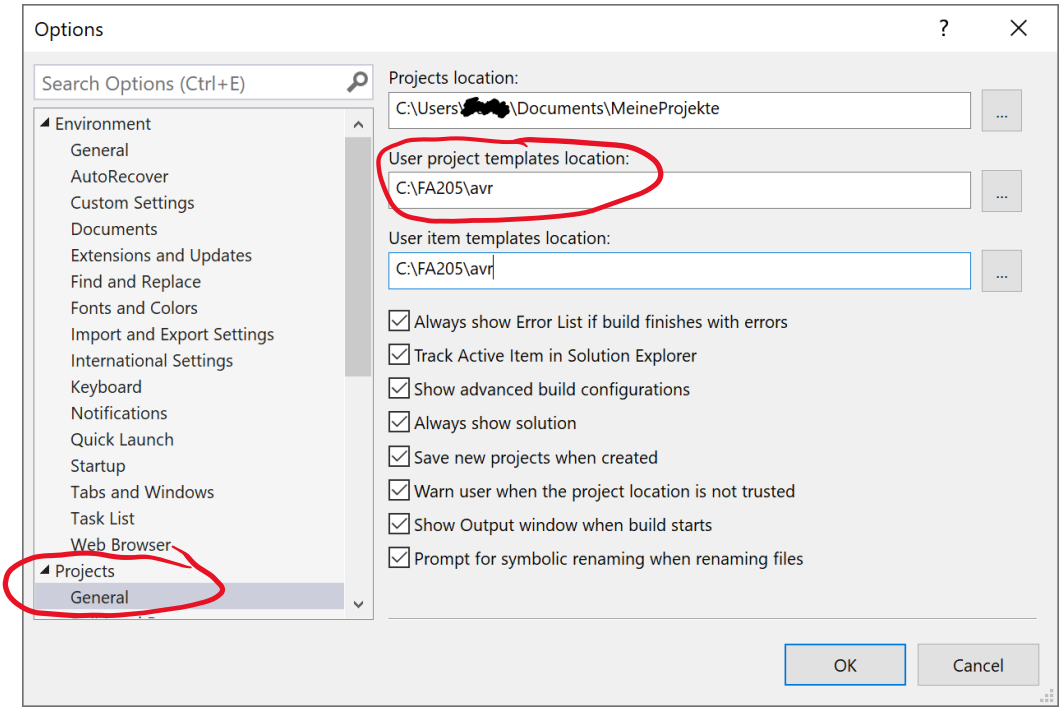
Eine Registrierung ist nicht (mehr) erforderlich. Während der Installation möchte der Installer verschiedene Frameworks installieren. Benötigt wird lediglich die ATmega (8-Bit)-Toolchain. Die USB-Treiber sollten alle installiert werden. Sonstige Komponenten die eventuell angeboten werden, benötigen nur Plattenplatz und verlängern die Installationsprozedur.

[1] Auf Apple oder Linux Systemen lässt sich AtmelStudio auch in einer virtuellen Maschine eher nicht installieren/benutzen.

3 Erstellen eines richtlinienkonformen Projekts in AtmelStudio 7

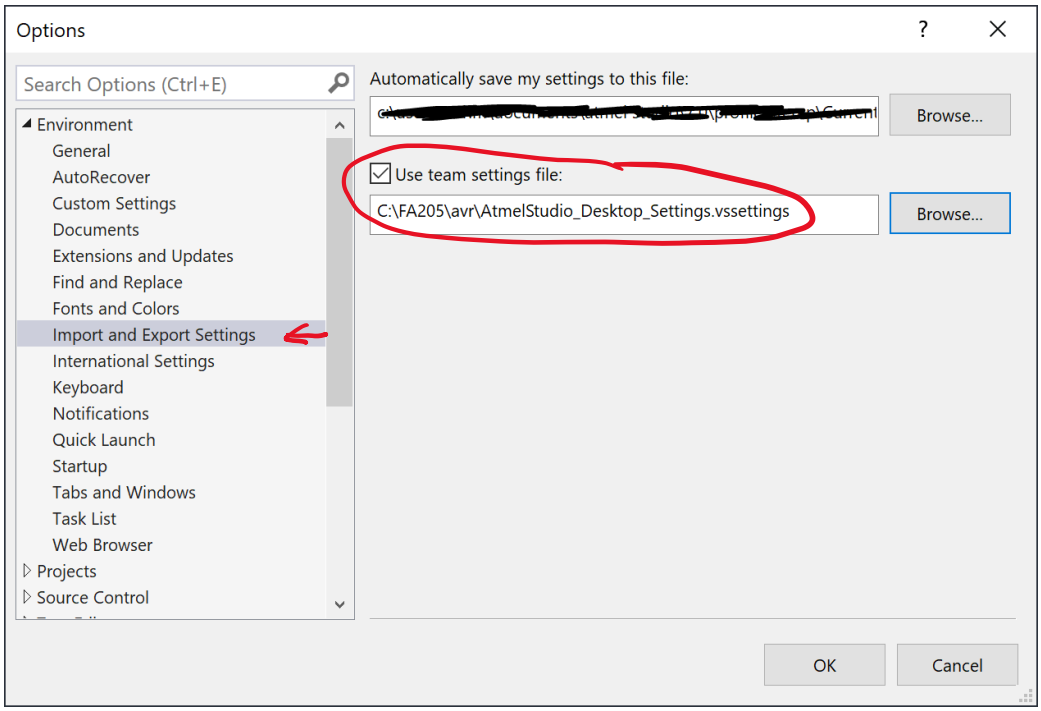
3.1 Einbinden der Projektvorlage (MeinProjekt.zip)

Eine fertige Projektvorlage für FA205-Projekte liegt im Ordner C:\FA205\avr. Um das Template zu nutzen, gehen Sie ins Menü: „Tools→ Options“. Stellen Sie den angegebenen Pfad zum Projekt-Template ein.

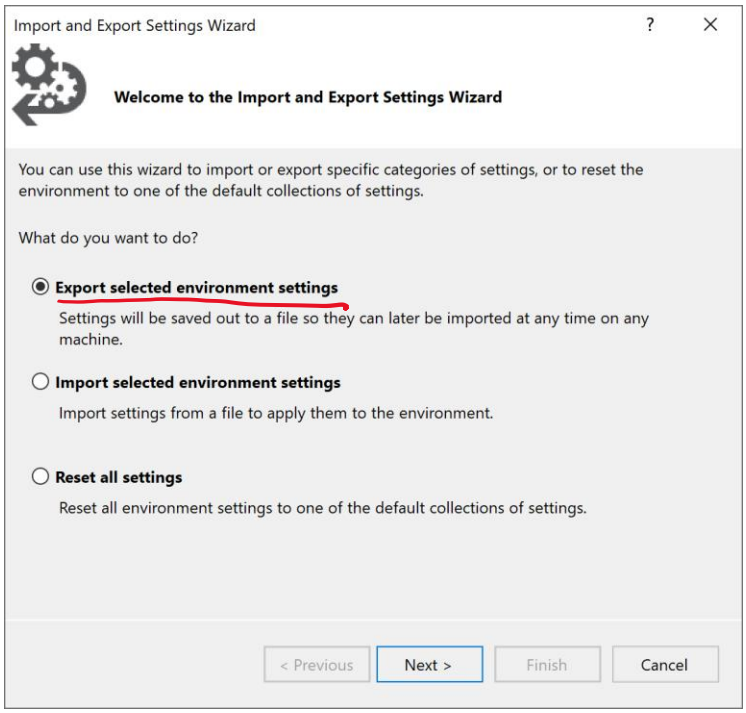


3.2 Einheitliche Desktop-Einstellungen laden („Team Settings“-Datei)

Um zunächst mit dem gleichen Aussehen und der Funktionalität wie im Labor zu starten, empfiehlt es sich die Teamsettings-Datei zu laden. Die Einstellung erfolgt ebenfalls im Menü „Tools→ Options“.



Es können auch leicht eigene Settings in der Team-Settings-Datei gespeichert werden. Den entsprechenden Eintrag findet man in „Tools→Import and Export Settings…“

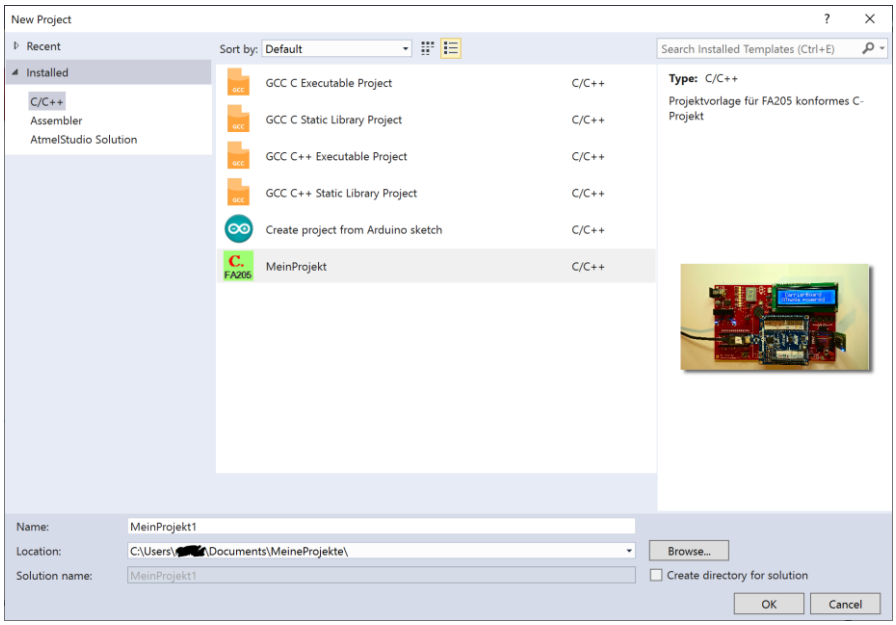


3.3 Erstellen eines FA205-Projekts

Wenn alle Einstellungen korrekt sind, können Sie über das Menü „File->New->Project“, oder direkt aus dem Startfenster ein neues Projekt erstellen. Geben Sie dem Projekt einen sinnvollen Namen und kontrollieren Sie die Pfadangaben.

Wichtig:

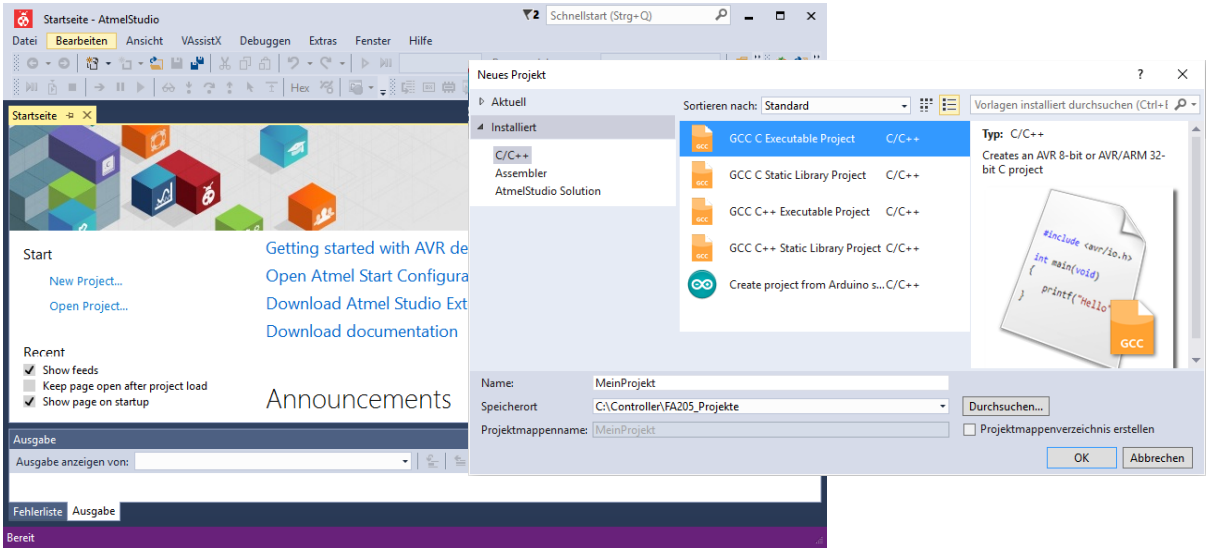
Verwenden Sie im Pfad und Dateinamen keine Sonderzeichen und Umlaute (ä,ö, ß, %, #, usw.), da der Compiler damit i.d.R. Probleme bekommt.
Erlaubt sind z.B. Unterstriche (Bsp.: 3_1_Beleuchtungssensor).



3.4 Eigene Projektvorlage erzeugen

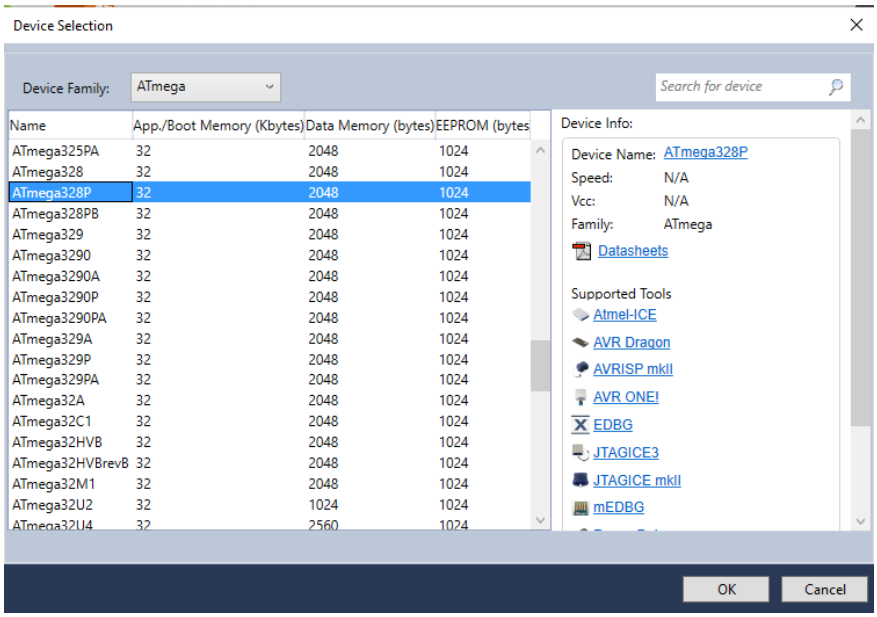
Diese Anleitung ist für den Fall, dass Sie eine eigene Projektvorlage erstellen möchten[2].

Starten Sie AtmelStudio 7 mit installierter ATmega (8 Bit)-Toolchain. Wählen Sie *Start→New Project…* . Erstellen Sie das Projekt (hier: MeinProjekt) im Homeverzeichnis ihres Netzordners.



3.4.1 Controller auswählen

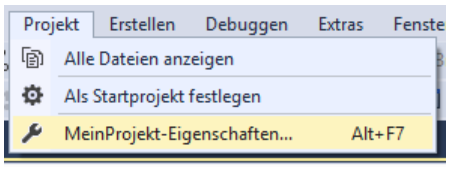
Wählen Sie im folgenden Dialog die Mikrocontroller-Familie und anschließend den Mikrocontroller (ATmega328p, Atmega328PB oder ATmega8). Sie können den Controller in den Projekteigenschaften im Abschnitt Device jederzeit anpassen.



3.4.2 Projekteinstellungen

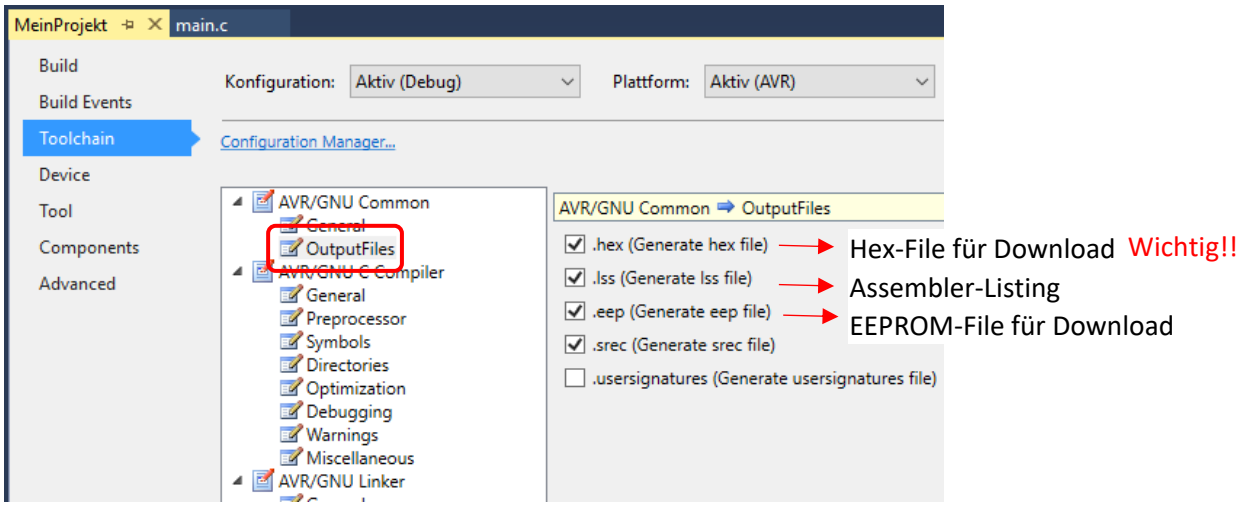
Die folgenden Einstellungen zum Projekt sollten vorgenommen oder kontrolliert werden. Öffnen Sie den Dialog: "*Projekt>MeinProjekt-Eigenschaften*" (Alt+F7).

[2] Die Menü-Sprache ist im Folgenden Deutsch. Der dazu notwendige Aufwand lohnt sich aber nicht wirklich und ist daher aus der Anleitung rausgefolgen. Die entsprechenden Englischen Einträge sind leicht zu finden.



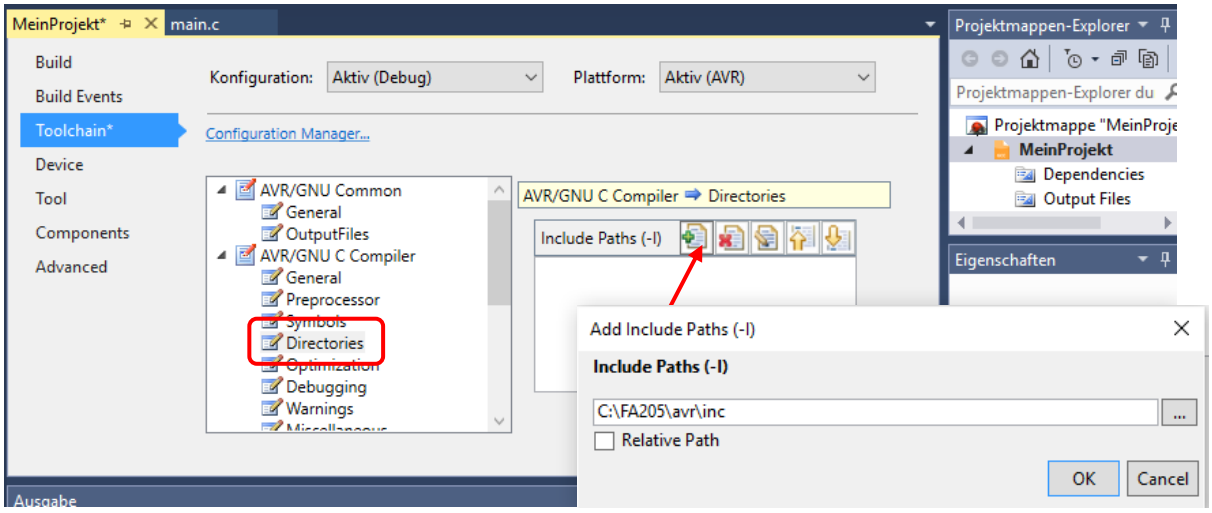
3.4.2.1 Ausgabeformat festlegen

Bei jedem Make wird eine Hex-Datei erzeugt, die für den Download zum Mikrocontrollersystem notwendig ist.



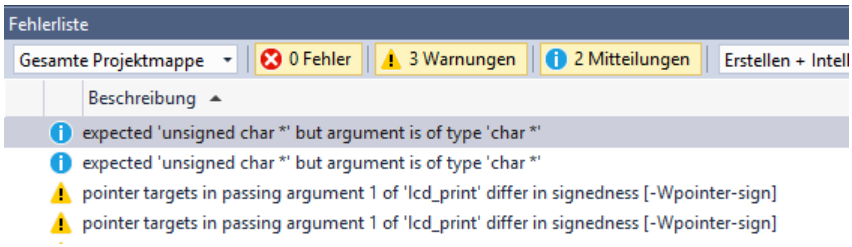
3.4.2.2 Einstellung des Suchpfades für die FA205 Header-Dateien

Der Suchpfad für die Header-Dateien (Include Paths) sollte absolut angegeben werden (z.B. "*C:\FA205\avr\inc*"). Der Compiler sucht die Include-Dateien dann in der Reihenfolge: 1. Include-Pfad, 2. Projekt-Verzeichnis, 3. Include-Pfad im AtmelStudio Installationsverzeichnis.

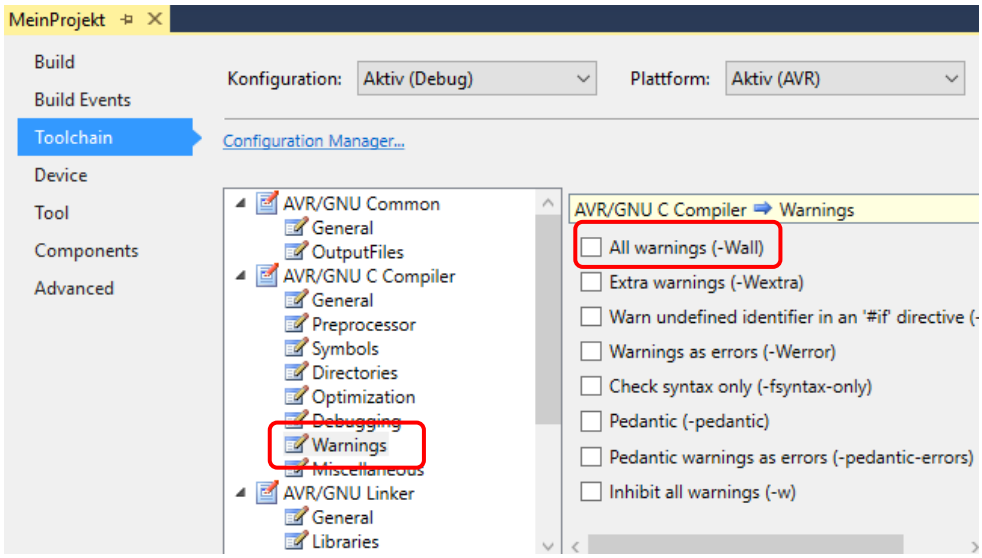


3.4.2.3 Unterdrücken von Linker-Warnmeldungen

Der AVR-GCC-Compiler zeigt beim Compilieren eines richtlinienkonformen Projektes eventuell folgende Warnmeldungen. Dies kommt daher, dass GCC für String-Argumente den Datentyp char verwendet, während in der Technischen Richtlinie explizit unsigned char verwendet wird.

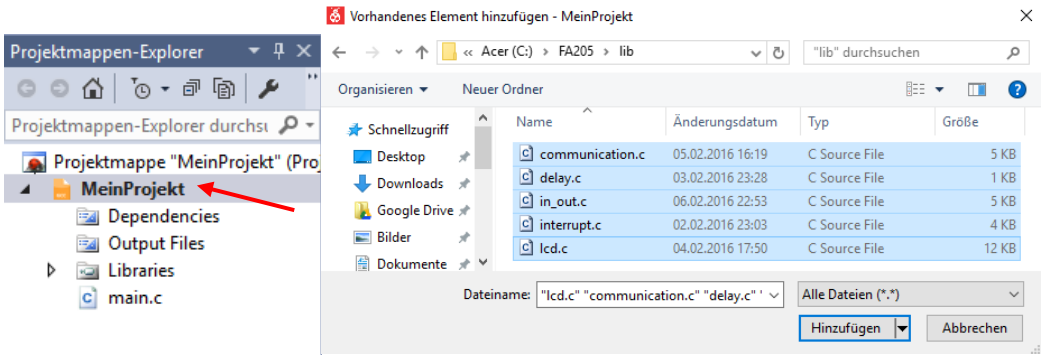


Wenn es stört, können die Warnungen unterdrückt werden:

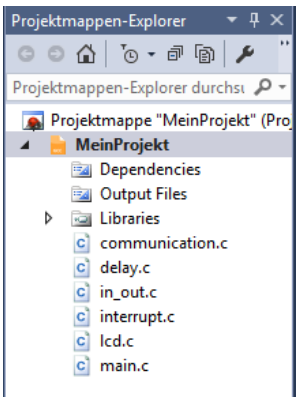


3.4.3 Hinzufügen der Bibliotheksfunktionen zum Projekt

Damit die Funktionen der Technischen Richtlinie vom Linker gefunden werden, müssen sie dem Projekt hinzugefügt werden. Dazu öffnen Sie im Projektmappen-Explorer mit Rechtsklick auf "*MeinProjekt>Hinzufügen>Vorhandenes Element*" (Umschalt+Alt+A) den Datei-Auswahl-Dialog.



Suchen Sie den Bibliotheksordner ("*C:\FA205\avr\lib*") und markieren Sie alle fünf Bibliotheksdateien. Fügen Sie sie mit '*Hinzufügen*' dem Projekt hinzu.

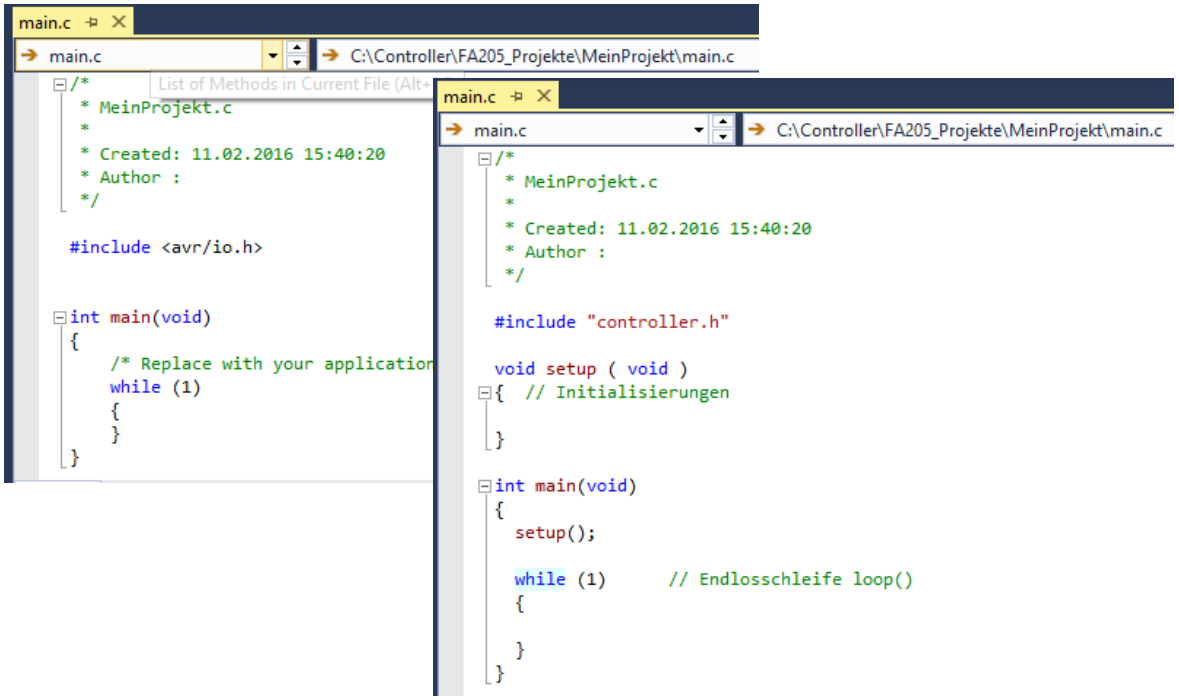


Grundsätzlich müssen nur die erforderlichen Bibliotheken hinzugefügt werden. Da der avr/gcc-Compiler nicht limitiert ist, ist eine Überschreitung der vorhandenen Programmspeichergröße, i.d.R. nicht zu erwarten.

3.4.4 Erstellen einer C-Quelldatei

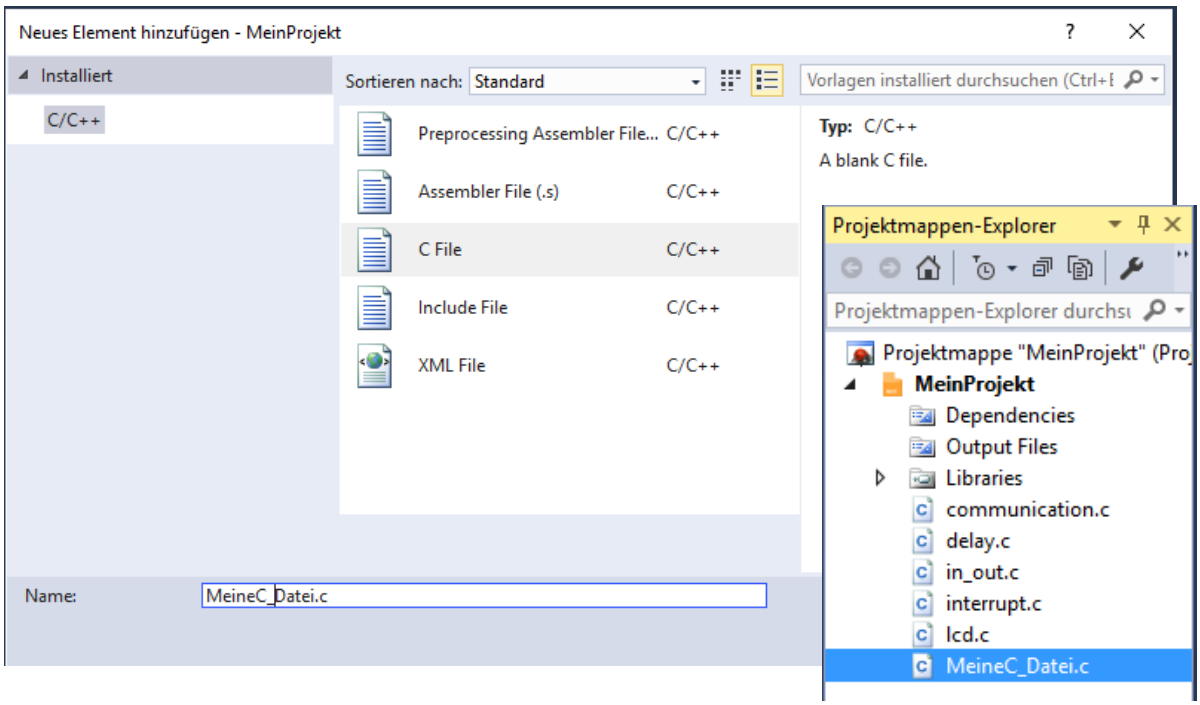
3.4.4.1 Anpassen der Datei main.c

Um eigene Funktionen dem Projekt hinzuzufügen, benötigen Sie eine C-Quelldatei. Die standardmäßig erstellte Datei ***main.c,*** kann wie dargestellt einfach richtlinienkonform abgeändert werden. Empfohlen wird der rechts abgebildete Grundaufbau:



3.4.4.2 Erzeugen einer neuen C-Datei

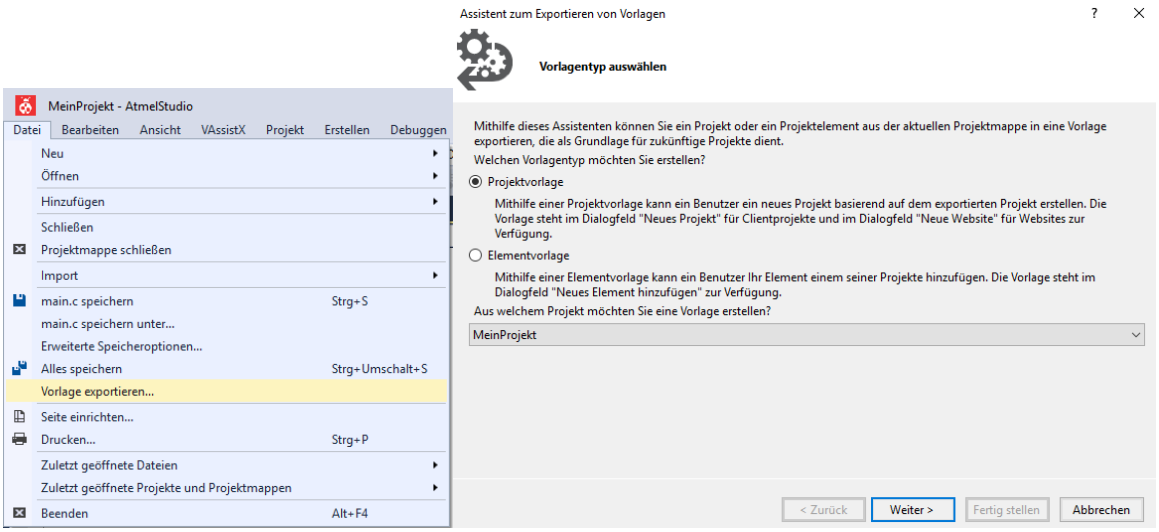
Alternativ kann eine neue C-Datei erstellt werden, die dann automatisch im Projektverzeichnis abgelegt wird. Gehen Sie im Projektmappen(Solution)-Explorer mit Rechtsklick auf "*MeinProjekt>Hinzufügen> Neues Element…*" (Strg+Umschalt+A). Geben Sie der Datei einen Namen.



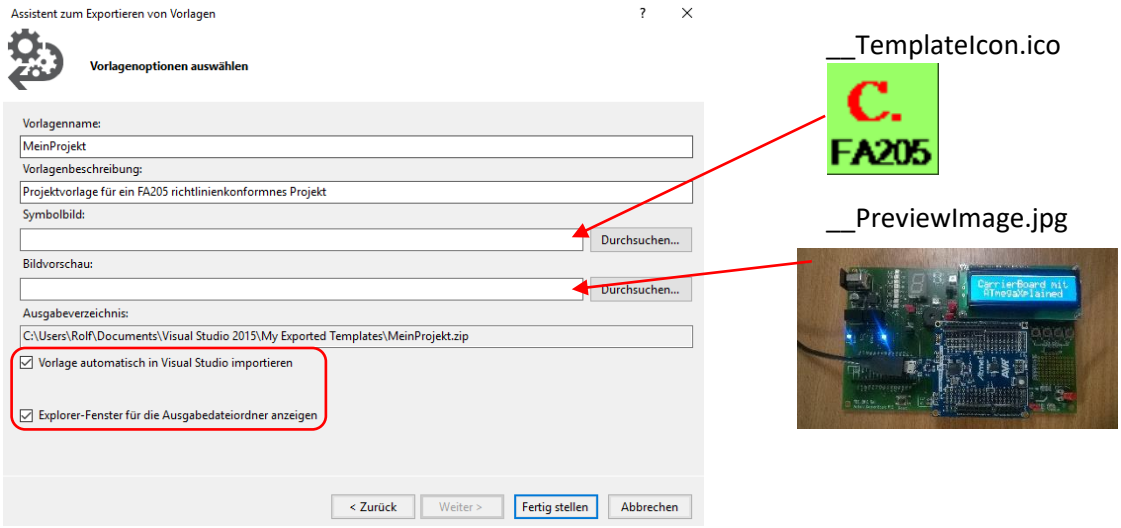
3.4.5 Projektvorlage erstellen

Damit nicht jedesmal alle Projekteinstellungen gemacht werden müssen, können Sie in AtmelStudio auch eine Projektvorlage (Template) erstellen.

Wählen Sie im Menü "*Datei>Vorlage exportieren…*" und anschließend "*Projektvorlage*" aus "*MeinProjekt*".

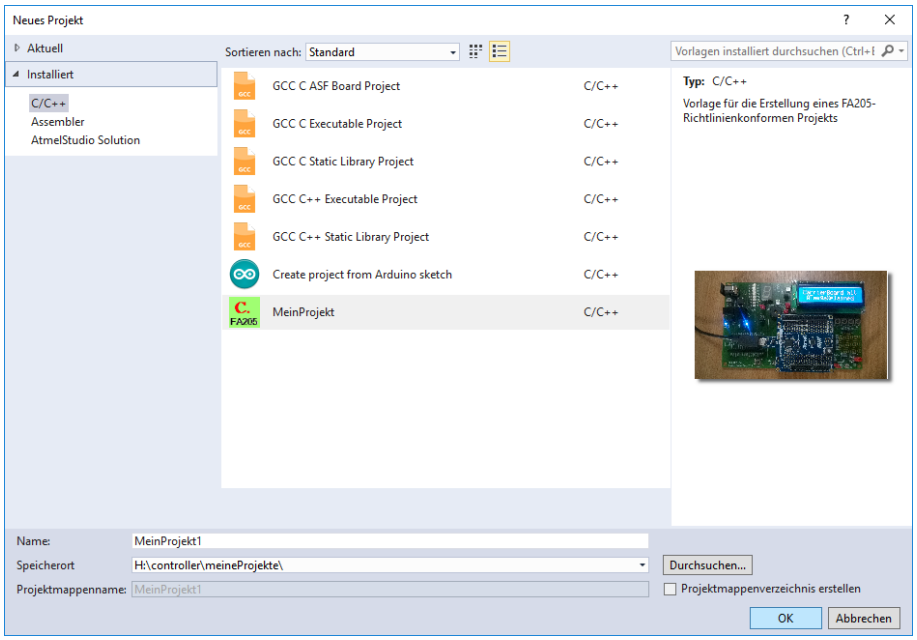


Im Vorlagenassistenten Vergeben Sie einen Vorlagenamen und eine Vorlagenbeschreibung. Diese wird dann im "*Start>New Projekt …*"-Dialog angezeigt. Haben Sie VisualStudio installiert, wird



dessen Tempate-Verzeichnis automatisch gesetzt. Wählen Sie die Einstellungen so wie vorgegeben und setzen Sie die Haken wie unten gezeigt.

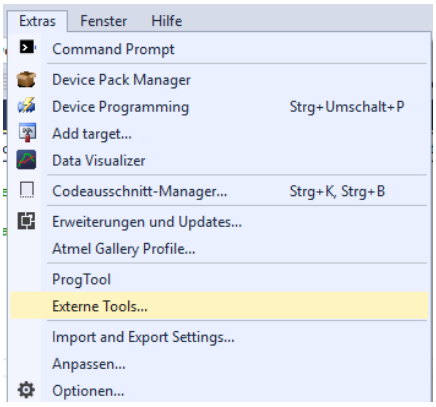
Nach der Erstellung des Templates befindet sich im VS-Template-Verzeichnis (C:\Users*xxxx*\Documents\Visual Studio 2015\Templates\ProjectTemplates) die Datei **MeinProjekt.zip**. Nun sollte beim Erstellen eines neuen Projekts das Template "MeinProjekt" angeboten werden. (Falls nicht, kopieren Sie das Template nach: C:\Users*xxxx*\Documents\Atmel Studio\7.0\Templates\ProjectTemplates). Geben Sie dem Projekt einen sinnvollen Namen. Wählen Sie einen beliebigen Speicherort (sinnvoll ist das Homeverzeichnis).

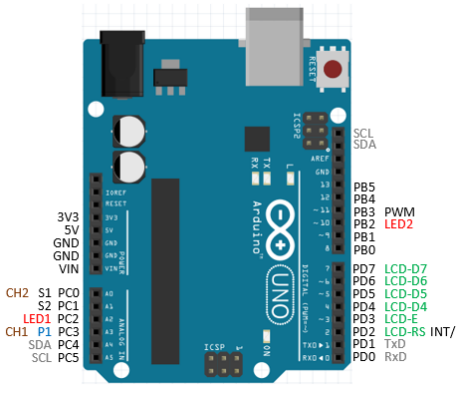


4 Download des Hex-Files auf Arduino

Da der ArduinoUno einen ATmega328p-Controller besitzt, kann er auch direkt aus AtmelStudio heraus programmiert werden. Dies geht z.B. über die auf dem Uno vorhandene 6-polige AVR-ISP-Schnittstelle und einen externen AVR-Programmer. Einfacher ist es jedoch die Im UNO implementierte Programmierschnittstelle über den USB-Port zu verwenden. Bei der Installation der Arduino-IDE wird dazu ein virtueller COM-Port eingerichtet, den die IDE zur Programmierung und zur seriellen Kommunikation verwendet.

Um das Hex-File eines in AtmelStudio erstellten Projekts direkt auf den ATmega328 herunterzuladen, muss das Programm **avrdude.exe** als externes Tool in AtmelStudio eingetragen werden. Dieses befindet sich (etwas versteckt) im Arduino Installationsverzeichnis. Wählen Sie "*Extras>Externe Tools…*":



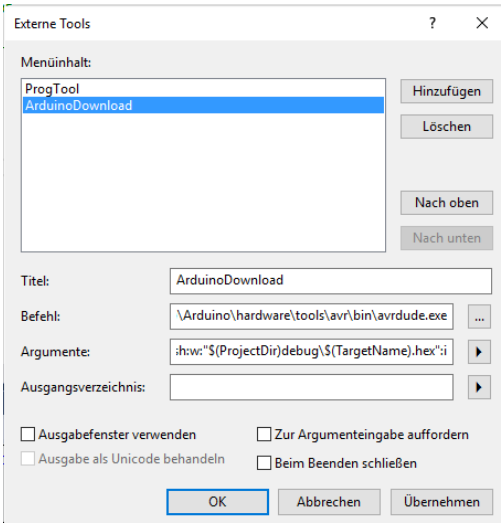


Klicken Sie im erscheinenden Dialog auf "*Hinzufügen*" und geben Sie dem Tool einen Namen (Titel), z.B. **ArduinoDownload**. Im Feld "*Befehl*"muss der Pfad und Dateiname zu avrdude eingetragen werden. Z.B.: *C:\Program Files (x86)\Arduino\hardware\tools\avr\bin\avrdude.exe*

Als **Argumente** tragen Sie die folgende Zeichenkette ohne Zeilenumbruch ein (bzw. passen Sie vorher den Arduino Installationspfad an):

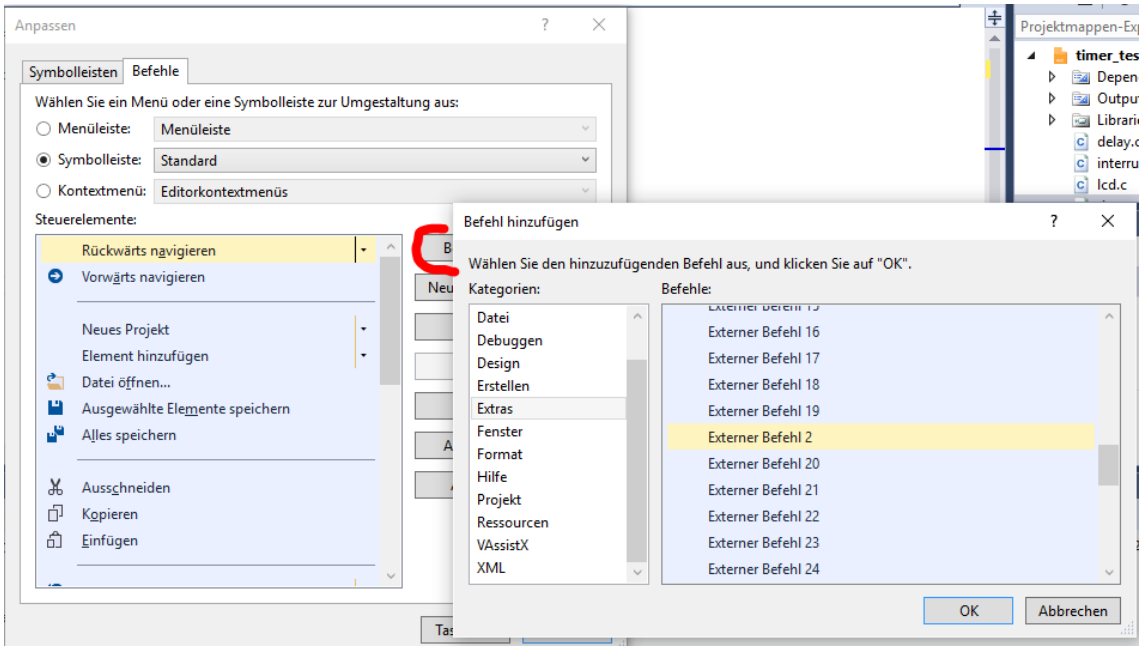
-C"C:\Program Files (x86)\Arduino\hardware\tools\avr/etc/avrdude.conf" -v -patmega328p -carduino -PCOM6 -b115200 -D -Uflash:w:"$(ProjectDir)debug\$(TargetName).hex":i

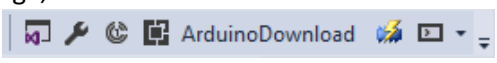
Beim Eintrag –PCOM6 ändern Sie bitte die Schnittstellennummer auf die entsprechende UART-Schnittstelle für den Arduino ab. Sie finden die korrekte Nummer im Gerätemanager oder in der Arduino-IDE unter "*Werkzeuge>Port: "COM**x** (Arduino Uno)*"".



Sie sollten den Haken bei "*Beim Beenden schließen*" entfernen, sonst erhalten Sie keine Rückmeldung, falls bei der Einrichtung etwas schief gegangen ist.

Nach *OK* erscheint das Tool im "*Extras-Menü*". Über den Dialog "*Extras>Anpassen…*" kann das Tool auch auf die Symbolleiste gelegt werden.



Die Nummer bei "*Externer Befehl x*" ist die Reihenfolge, in der das Tool im Menü "*Extras*" erscheint. Die Symbolleiste sieht dann etwa so aus: 

Das Tool sollte nun für alle Projekte gültig sein. Denken Sie daran, den COM-Port anzupassen, falls der Arduino auf einem anderen virtuellen COM-Port liegt, z.B. wenn ein anderer USB-Port verwendet wird.

5 Anpassen der Bibliotheksfunktionen an die eigene Hardware

Der folgende Abschnitt ist nur wichtig, wenn die FA205-Bibliotheken an eigene Hardware mit Atmega-Controller angepasst werden soll !

Damit die eigene Mikrocontroller-Hardware mit den Bibliotheksfunktionen zusammenarbeiten, müssen in der Regel Anpassungen vorgenommen werden. In der Header-Datei werden Einstellungen vorgenommen, die keinen direkten Eingriff in die Bibliotheksfunktionen erfordern. Bei geänderter Hardware, müssen die Anpassungen aber direkt in den Bibliotheksfunktionen erfolgen. Aufgrund der Fülle an Mikrocontrollern und Lehrsystemen, sind die folgenden Angaben naturgemäß nicht vollständig, sollen aber die Haupteingriffspunkte in die Bibliotheksfunktion aufzeigen. Dabei werden nacheinander alle Bibliotheken besprochen.

5.1 controller.h

In der Header-Datei 'controller.h' werden Controller-spezifische Einstellungen festgelegt. Wählen Sie hier den Controllertyp (_ATMEGA328_ oder _ATMEGA8_) und die Systemfrequenz F_CPU.

```
// Controllerspezifische Registerdefinitionen
#include "avr/io.h"

//#define _ATMEGA8_      // z.B. MyAVR
#define _ATMEGA328_    // z.B. Arduino UNO

// F_CPU beeinflusst timer1ms, delay, rs232, pwm
//#define F_CPU 3686400UL     // z.B. myAVR
#define F_CPU 16000000UL    // z.B. Arduino UNO
```

5.2 in_out.h

In der in_out-Bibliothek sind Port-I/O, PWM und ADC-Funktionen definiert.

PORT-IO

Bei den Port Ein-/Ausgabe-Funktionen sollten keine Veränderungen vorgenommen werden. Für weitere Ports (z.B. ATmega32) müssen Änderungen auch in der Bibliothek **in_out.c** erfolgen.

PWM

Die AVR-Mikrocontroller haben unterschiedliche PWM-Ausgänge. Die beiden hier unterstützten Controller (ATmega8, ATmega328p) und Kompatible besitzen beide einen PWM Ausgang an PortB.3. Dieser ist standardmäßig als PWM-Port konfiguriert. Alternativ kann beim ATmega328p der PortD.3 als PWM-Ausgang aktiviert werden (Schalter aktivieren). Andere sind hier nicht implementiert.

```
// ... für PWM
// Der PWM-Ausgang ist standardmäßig PORTB.3 (PWM mit Timer2)
// Wenn in controller.h der Schalter auf _ATMEGA328_ steht,
// kann alternativ der PORTD.3 als PWM-Ausgang gewählt werden.
//#define _PWM_PORT_PD3_             // PWM-Ausgang PORTD.3
```

ADC

Die ATmega-Controller haben 6 ADC-Eingangskanäle. In der Technischen Richtlinie sind zwei ADC-Funktionen (adc_in***1***() und adc_in***2***()) definiert. Hier können die 6 Kanäle einer der beiden Funktionen zugeordnet werden.

```
// ... für ADC
// Mögliche ADC-Ausgänge: PORTC  .0 | .1 | .2 | .3 | .4 | .5
//                              0x00 0x01 0x02 0x03 0x04 0x05
#define CH1 0x03    // Kanal 1: PORTC.3
#define CH2 0x04    // Kanal 2: PORTC.4
```

5.3 delay.h

Die Delay-Bibliothek enthält die Funktionen um softwaregesteuerte Wartezeiten zu realisieren. Damit die Delay-Zeiten möglichst genau sind, muss im Schlüssel F_CPU in controller.h die korrekte Prozessorfrequenz eingetragen sein.

5.4 lcd.h

In der LCD-Bibliothek sind die Ausgabefunktionen für ein Standard Text-Display definiert.

Die Definition der Anfangsadressen der Textzeilen hängt ab von der Anzahl darstellbarer Zeichen pro Zeilen. Verwenden Sie bei Zeilenlängen bis 16 Zeichen die Einstellung LCD_LEN 16. Bei Zeilenlängen ab 20 Zeichen LCD_LEN 20.

```
//**** Schalter *******************************************************
#define LCD_LEN 16              // Display-Zeilenlänge 20 oder 16 (default)
```

Bei der LCD-Schnittstelle kann zwischen einer standardmäßigen 4-Bit-Schnittstelle und einer Anbindung über den I²C-Bus gewählt werden. Aktivieren Sie den Schalter LCD_I2C, wenn Sie ein LC-Display-Modul mit PCF8574-Porterweiterung haben. Für die 4 Bit-Schnittstelle deaktivieren Sie den Schalter.

```
#define LCD_I2C                 // Aktiviert das I2C-Interface
```

Wird das Display im 4-Bit Interface Modus betrieben, müssen die 4 Datenleitungen immer auf 4 aufeinanderfolgenden Bits auf dem LCD_PORT liegen. Die Zuordnung der Datenleitungen zu den Portbits wird dabei in LCD_PORT_MASK vorgenommen. Es ergeben sich 5 mögliche Werte:

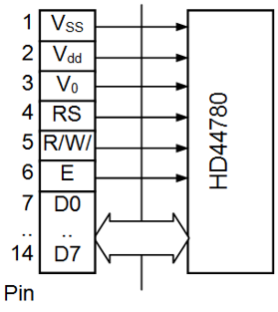
```
// * |D.7|D.6|D.5|D.4|D.3|D.2|D.1|D.0|    |B.0|
// * | 7 | 6 | 5 | 4 | EN| RS| x | x |    |R/W|      MyAVR-Board
#define LCD_PORT        PORTD   // LCD-Display an PortD im -4-Bit-Modus
#define LCD_PORT_CFG    DDRD
#define LCD_PORT_MASK   0xf0    // Maskiert die 4 Datenbit (4 Bit-Modus)
                                // Mögliche Werte: f0, 78, 3c, 1e, 0f
```

Die Steuerleitungen EN, RS und RW können auf beliebige Portbits gelegt werden.

```
#define LCD_PORT_EN     PORTD   // LCD-Display EN an PortD.3
#define LCD_PORT_EN_CFG DDRD
#define EN       3

#define LCD_PORT_RS     PORTD   // LCD-Display RS an PortD.2
#define LCD_PORT_RS_CFG DDRD
#define RS       2


// (D.7) Wenn Busy-Flag nicht abgefragt werden kann/soll,
// dann LCD_BUSY auskommentieren
//#define LCD_BUSY 7
 #ifdef LCD_BUSY
  #define LCD_PORT_RW     PORTB   // LCD-Display RW an PortB.0
  #define LCD_PORT_RW_CFG DDRB
  #define RW       0
  #define LCD_PORT_BUSY   PIND    // Busy-Flag wird von Bit D7 gelesen!
 #endif
```



Die Steuerleitung RW ist nur erforderlich, wenn das Busy-Flag abgefragt werden soll. Ansonsten bleiben die entsprechenden Schlüssel (LCD_PORT_RW, LCD_PORT_CFG, RW) unberücksichtigt. Der RW-Pin am Display (Pin 5) muss dann auf GND (0V) gelegt werden.

Die Abfrage des Busy-Flags beschleunigt die Display-Zugriffe. Dazu muss der Schlüssel LCD_BUSY aktiviert werden und die Nummer dem Portbit am Controller entsprechen, das mit dem Bit D7 des

Displays verbunden ist. LCD_PORT_BUSY muss daher immer das Ausgangsregister des gewählten LCD_PORT sein.

Beispiel:

LCD_PORT	PORTB		PORTB	.7	.6	.5	.4	.3	.2	.1	.0
LCD_PORT_MASK	0x3c	➔	Maske	0	0	1	1	1	1	0	0
			Display			D7	D6	D5	D4		
LCD_PORT_BUSY	PINB										
LCD_BUSY	5		Busy-Flag			B.5					

Wird die I²C-Schnittstelle mit PCF8574 verwendet, können die Steuerleitungen auf den Port-Bits veteilt werden. Die Datenleitungen sind fest auf P7 … P4 gelegt. Je nach Hardware ergeben sich beim PCF8574 acht mögliche Schreib- und Leseadressen am I²C-Bus (siehe Datenblatt). Eventuell muss hier probiert werden. Die abgebildete Konfiguration ist gültig für das I²C-Display-Modul, welches mit dem **Allnet-Starterkit für Arduino Uno** geliefert wird.

```
#else   // Für I2C-Display mit PCF8574 (D7..D4 => P7..P4)!!!
 #define RS       0
 #define RW       1
 #define EN       2
 #define BL       3   //Backlight
 //***** I2C-Adressen und Kontrollbyte für den PCF8574 ***************
 #define LCD_ADDR_R  0x4f   // 01001111 ->  0100 fest, 111 durch Jumper, 1 Wert lesen
 #define LCD_ADDR_W  0x4e   // 01001110 ->  0100 fest, 111 durch Jumper, 0 Wert schreiben
#endif
```

5.5 Interrupt.h

Der externe Interrupt ist standardmäßig mit dem externen Interrupt 0 an PORTB.2 des AVR-Mikrocontrollers realisiert. Durch Ändern des Schalters nach _EXT_INT1_ wird dagegen der externe Interrupt 1 an PORTB.3 verwendet.

```
#define _EXT_INT0_     // externer Interrupt 0 oder 1
```

Anm.: Diese Umschaltung ist zur Zeit noch nicht implementiert (Rahm, 12.2.16)

5.6 communication.h

I2C

Die I2C-Funktionen sind für den Masterbetrieb definiert. Es wird der I2C-Port des AVR-Controllers verwendet. Die Busfrequenz kann im Schlüssel _I2C_FREQUENZ_ eingestellt werden. Beachten Sie: Je höher die Busfrequenz, desto störanfälliger wird die Übertragung. Das Produkt aus Pullup-Widerstand und kapazitiver Belastung der Busleitung (R*C) darf nicht zu groß sein.

```
// ... für I2C-Funktionen
// SDA  = PC.4
// SCL  = PC.5
#define _I2C_FREQUENZ_ 40000 // 40kHz     @16MHz:   40kHz...100kHz
                             //           @3.16MHz: 8kHz ...100kHz
```

RS232

Die Standard-Baudrate für die Technische Richtlinie beträgt 9600Baud. Sie wird abhängig von F_CPU berechnet. Prinzipiell können auch andere Baudraten eingestellt werden.

```
// ... für RS232-Funktionen
#define BAUD 9600
```