

Arduino Carrier Board mit AtmegaXplained und AtmelStudio 7

Inhalt

1	Das FES-Arduino Carrier Board.....	2
1.1	Baugruppen des Carrier-Board.....	2
1.2	Pinbelegungen auf dem Arduino-Header	2
1.3	Mögliche Systemkonfigurationen	3
1.3.1	Carrier-Board mit Arduino-Nano	3
1.3.2	Carrier-Board mit ATmega328P Xplained	3
1.3.3	Carrier-Board mit Pretzel IOT-Controller	3
1.3.4	Carrier-Board mit ArduinoUno	4
2	Atmega328pXplained-mini mit ATmega328P-Controller	5
2.1	Aufbau und Baugruppen des ATmegaXplained	5
2.2	Vergleich: ATmegaXplained vs. ArduinoUno	5
3	Xplained-Board mit AtmelStudio	6
3.1	Anpassen der FA205-Bibliotheken für I2C-Display.....	6
3.2	Programmierung über die DebugWire-Schnittstelle.....	7
3.2.1	Inbetriebnahme des Controller-Boards	7
3.2.2	Aufspielen der neuesten Firmware	7
3.2.3	ISP-Programmierung	8
3.2.4	In-System-Debugging mit debugWire	8

Rolf Rahm

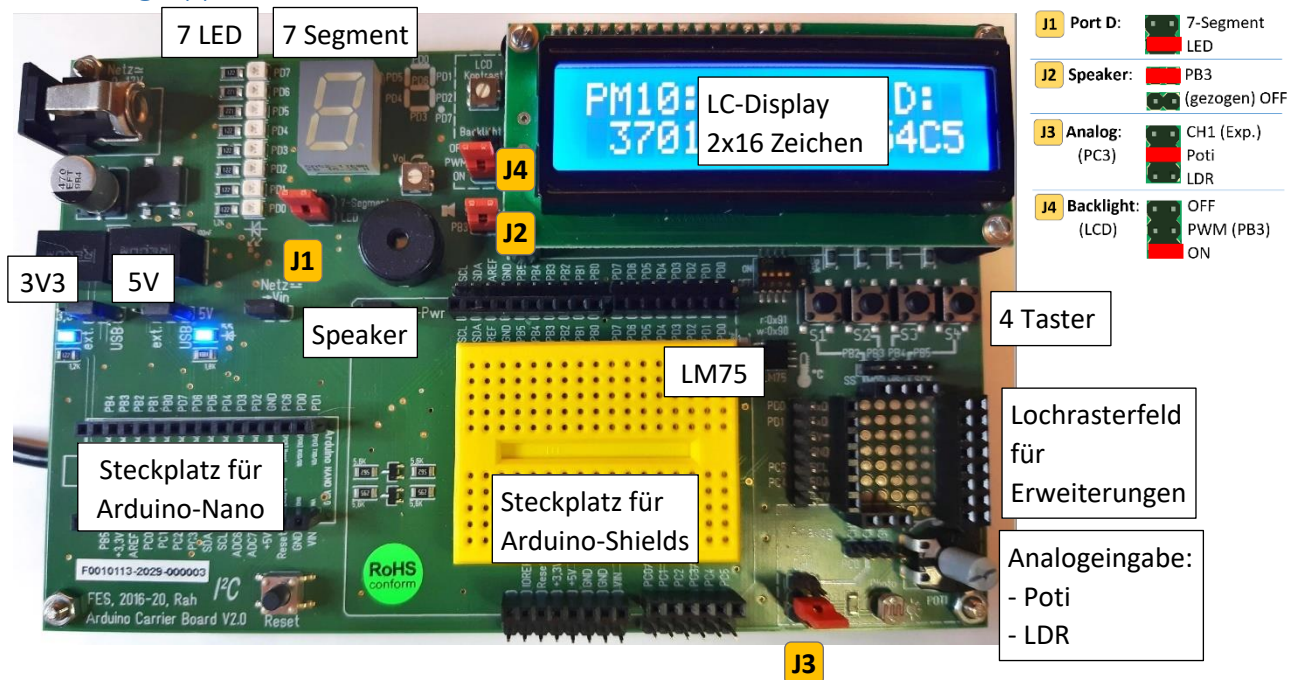
Reichenbach, August 2016

(Letzte Änderung: 19.11.2021)

1 Das FES-Arduino Carrier Board

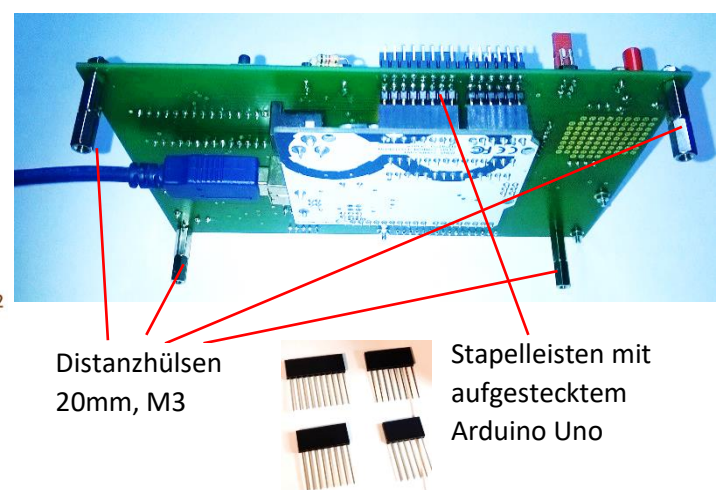
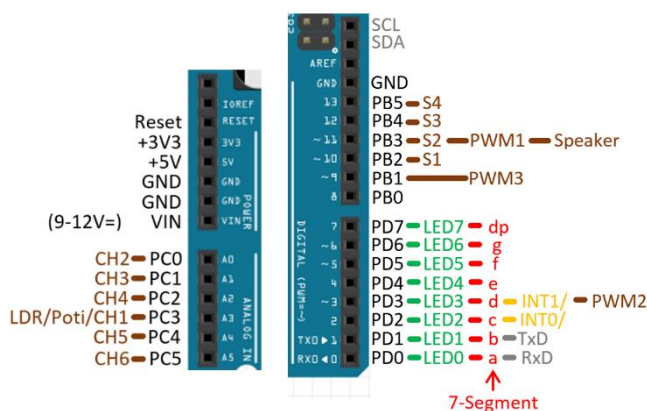
Das Arduino Carrier Board ist eine fertige Hardwareplattform, mit der die gängigen Programmiersuche zu Mikrocontrollern ohne (fehleranfällige) Steckbretter durchgeführt werden können. Ebenso ist es für arduinokompatible Shields voll erweiterungsfähig. Das Board kann mit Arduino und ArduinoNano kompatiblen Mikrocontroller-Modulen verwendet werden und bietet damit eine universelle Lösung für die unterschiedlichsten unterrichtlichen Anforderungen. Das Board kann in verschiedenen Ausstattungsvarianten von ASE-Elektronik bestellt werden: <https://www.ase-schlierbach.de/produkte/>

1.1 Baugruppen des Carrier-Board



1.2 Pinbelegungen auf dem Arduino-Header

Bestückung mit Stapel-Buchsenleisten für Arduino Uno



1.3 Mögliche Systemkonfigurationen

1.3.1 Carrier-Board mit Arduino-Nano



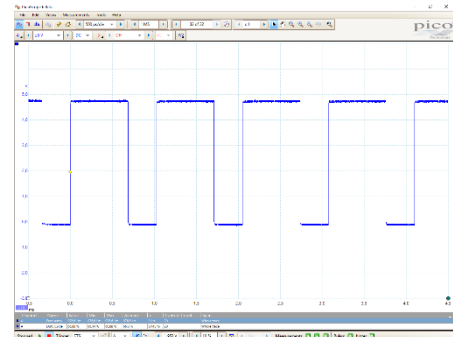
```
lcl_test_fa205 | Arduino 1.6.5
Datei Bearbeiten Sketch Werkzeuge Hilfe
lcl_test_fa205 $
#include "controller.h"

void setup()
{
  lcd_init();
  delay_ms(20);
  lcd_setcursor(1,1);
  lcd_print(" CarrierBoard");
  lcd_setcursor(2,1);
  lcd_print("mit ArduinoNano");
}

void loop()
{
}

Hochladen abgeschlossen.
Arduino Nano, ATmega328 on COM3
```

Beispiel mit Motorshield (L298) und PicoScope zur Aufnahme der PWM



1.3.2 Carrier-Board mit ATmega328P Xplained



Spezielle Features:

- Mit Stapelleisten bestückbar
- Programmierung und Debugging über debugWire-Schnittstelle.
- PD0 und PD1 sind frei für Byteausgabe über PORTD (z.B. SiebenSegment)
- Arduino Shields können aufgesteckt werden.

1.3.3 Carrier-Board mit Pretzel IOT-Controller



- Mit ESP8266 WLAN-Controller
- Arduino Nano kompatibel
- Bibliotheken für Arduino-IDE

1.3.4 Carrier-Board mit ArduinoUno



Bestückung mit Arduino Stapel-Buchsenleisten



Um den ArduinoUno mit dem Carrier-Board zu verwenden, müssen die Arduino-Pinheader als Stapel-Buchsenleisten ausgeführt sein. Der Uno wird dann von unten auf das Carrier-Board gesteckt.

Somit muss das Carrier-Board auf Abstandsbolzen (>20mm) gesetzt werden.

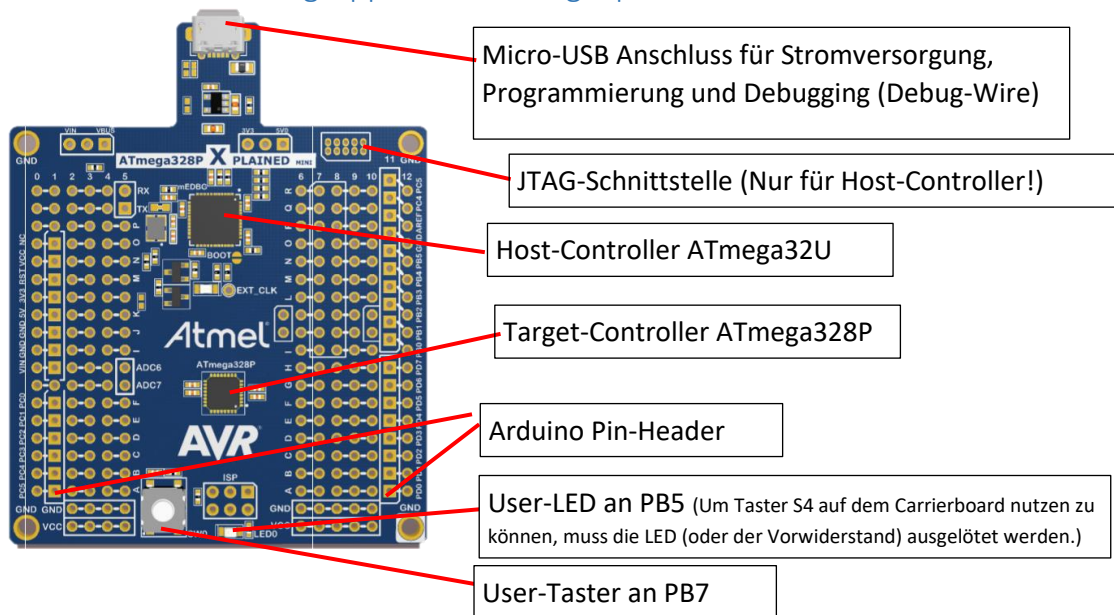
Die Programmierung erfolgt dann in der Arduino IDE oder alternativ in AtmelStudio.¹

Die FA205-Richtlinienbibliotheken müssen analog zu Abschnitt 3.1 auch für die Arduino-IDE angepasst werden.

¹ Zum Einbinden von AVR-Dude in AtmelStudio siehe Anleitung „Implementierung der Technischen Richtlinie mit AtmelStudio für ATmega-Mikrocontroller“.

2 Atmega328pXplained-mini mit ATmega328P-Controller

2.1 Aufbau und Baugruppen des ATmegaXplained



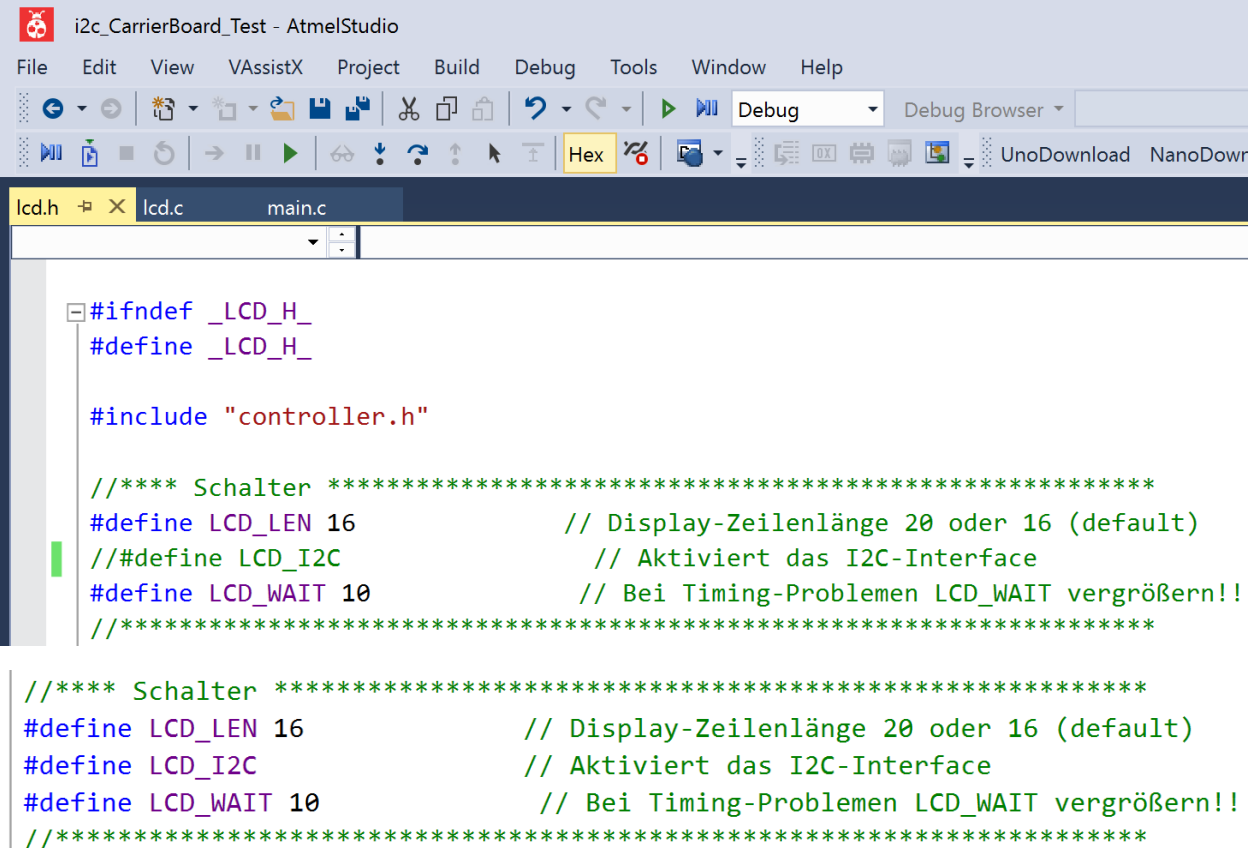
2.2 Vergleich: ATmegaXplained vs. ArduinoUno

	ATmegaXplained	ArduinoUno
Spannungsregler	3,3V	5V; 3,3V
Spannung an Vin	max. 5V (kein 5V-Regler onboard)	max. 12V (5V-Regler onboard)
Spannungsversorgung	USB; +5V	USB; Netzbuchse (Vin)
User LED	PB5	PB5 (Arduino Pin 13)
User Taster	PB7	-
InSystem-Debugging	Debug-Wire	-
IDE	AtmelStudio, ArduinoIDE (Mit speziellem Bootloader)	ArduinoIDE, AtmelStudio
Kontaktierung	Arduino-Pin-Header, Signale mehrfach zugänglich, Stapelbuchsenleiste einlötbar	Arduino-Pin-Header
RS232-Schnittstelle	PD0/PD1 frei verwendbar.	PD0/PD1 (RxD/TxD) durch Kommunikation mit Host blockiert.

3 Xplained-Board mit AtmelStudio

3.1 Anpassen der FA205-Bibliotheken für I2C-Display

In AtmelStudio die Datei lcd.h (Standardmäßig im Ordner C:\FA205\avr\inc) öffnen und den Schalter LCD_I2C entkommentieren.



```
i2c_CarrierBoard_Test - AtmelStudio
File Edit View VAssistX Project Build Debug Tools Window Help
Debug
Hex
UnoDownload NanoDownr

lcd.h x lcd.c main.c

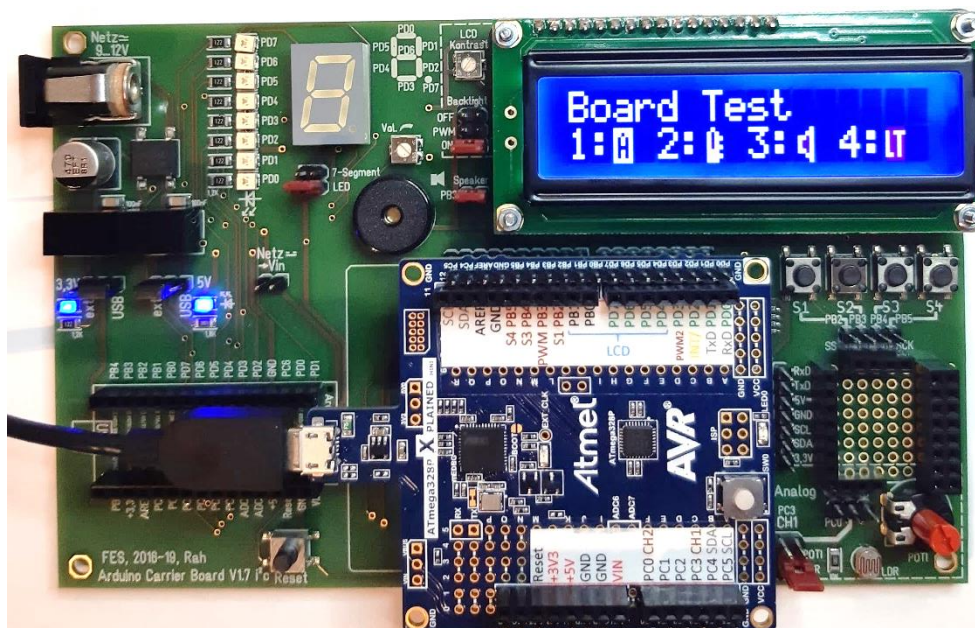
#ifndef _LCD_H_
#define _LCD_H_

#include "controller.h"

//**** Schalter ****
#define LCD_LEN 16 // Display-Zeilenlänge 20 oder 16 (default)
// #define LCD_I2C // Aktiviert das I2C-Interface
#define LCD_WAIT 10 // Bei Timing-Problemen LCD_WAIT vergrößern!!
// ****

//**** Schalter ****
#define LCD_LEN 16 // Display-Zeilenlänge 20 oder 16 (default)
#define LCD_I2C // Aktiviert das I2C-Interface
#define LCD_WAIT 10 // Bei Timing-Problemen LCD_WAIT vergrößern!!
// ****
```

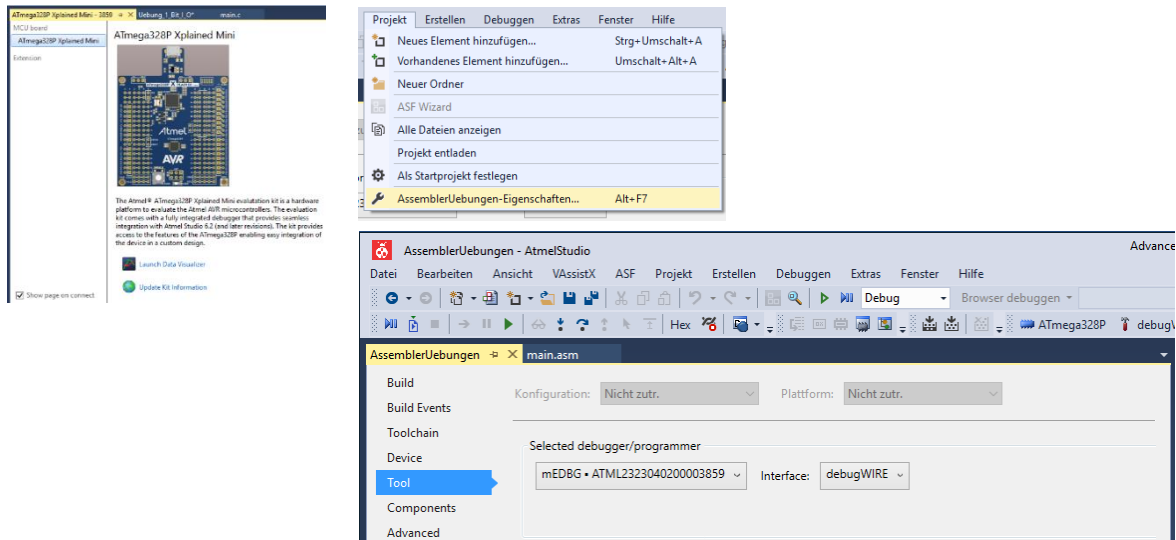
Nun kann das Display parallel mit den LEDs/SiebenSegment genutzt werden.



3.2 Programmierung über die DebugWire-Schnittstelle

3.2.1 Inbetriebnahme des Controller-Boards

Erstellen Sie zunächst ein Projekt in AtmelStudio⁷² und schließen Sie das Board mit einem MicroUSB-Kabel am PC an. Im Hauptfenster erscheint der Tab „ATmega328P Xplained Mini – xxxx“. Darüber haben Sie Zugriff auf Tools und Online-Informationen rund um das Xplained-Board. Der Tab wird meist nicht benötigt und kann geschlossen werden. Öffnen Sie nun den Tab "Projekt>projektname-Eigenschaften (Alt+F7)". Als Tool wählen Sie hier "mEDBG • ATMLxxxxxxx" und als Interface: "debugWire".

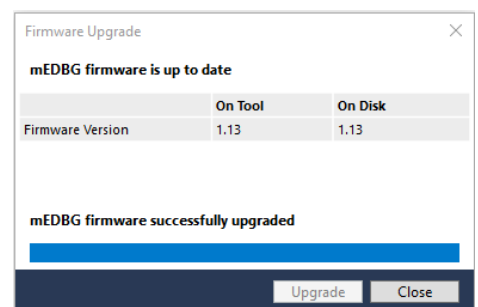
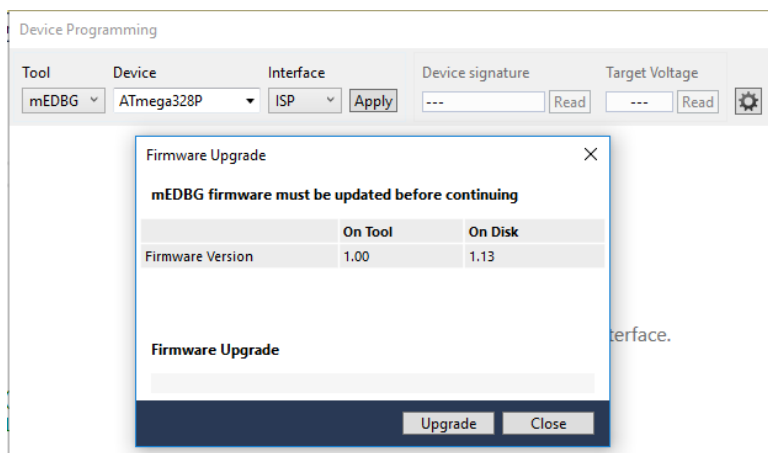
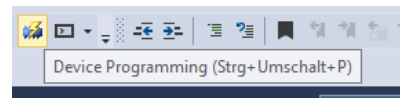


Schreiben Sie nun den Quellcode Ihres Programms.

3.2.2 Aufspielen der neuesten Firmware

Vor dem ersten Programmiervorgang des XplainedBoards, oder nach Updates von Atmel Studio, muss eventuell die Firmware upgedatet werden. Öffnen Sie zunächst das Programmiertool mit "Extras>Device Programming" oder über die Symbolleiste:

Stellen Sie als Tool "mEDBG" ein. Nach Betätigen von Apply erscheint automatisch die Upgrade-Aufforderung.



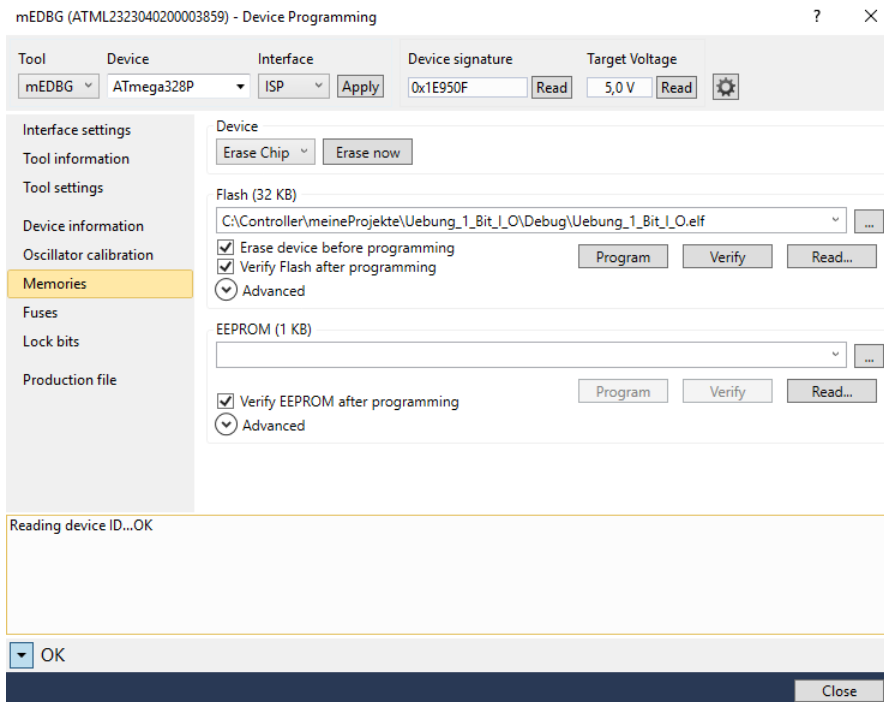
⁷² siehe auch Labor-Projekt: "Programmierung des Mikrocontrollers mit AtmelStudio 7"

3.2.3 ISP-Programmierung

Um ein Programm auf die Xplained-Hardware zu übertragen gibt es zwei Möglichkeiten. Die In-System-Programmierung (ISP) und das In-System-Debugging.

Beim ISP wird die Hex-Datei mit dem Maschinencode (*.hex, *.elf) mit dem Programmiertool direkt in den Controller geschrieben. Erstellen Sie zunächst die Hexdatei mit (Make) (F7).

Öffnen Sie dann wieder mit "Extras>Device Programming" das Programmiertool und nehmen Sie die in der Abbildung unten gezeigten Einstellungen vor. Achten Sie darauf, dass die Target-Voltage nicht unter 4,5V angezeigt wird, da sonst der Controller beim Programmieren Schaden nehmen



kann. Im Abschnitt "Memories" wählen Sie die erstellte Hex- oder ELF-Datei aus und betätigen "Program". Anschließend wird das Programm übertragen und ein Reset auf dem Controller ausgeführt. Schließen Sie nun das Programmier-Tool wieder.

Hinweis: Im Abschnitt „Fuses“ und „Lock bits“ nur Änderungen

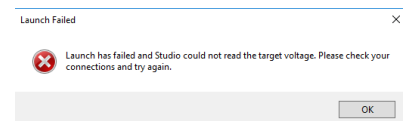
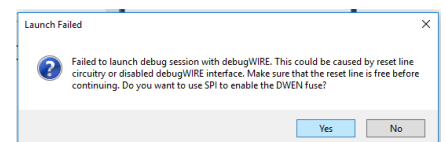
vornehmen, wenn man genau die Wirkung kennt.

3.2.4 In-System-Debugging mit debugWire

Beim In-System-Debugging kann der Programmablauf, Variablen, interne Register, IO-Ports, des Controllers, usw. während der Programmausführung beobachtet werden. Er eignet sich besonders zum Suchen von Programmfehlern, aber auch um die Arbeitsweise eines Programms zu analysieren und zu verstehen. Es ist daher die favorisierte Programmiermethode im Mikrocontroller-Labor.

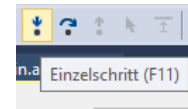


Zur Einrichtung muss, wie vorher beschrieben, das debugWire-Interface aktiviert sein³. Im Menü „Debuggen“ starten Sie nun das Debugging (Alt+F5 oder F5). Eine Meldung zeigt an, dass auf dem Controller die Fuse DWEN (debugWire Enable – Bit) gesetzt werden muss. Atmel Studio bietet an dies selbst zu tun. Quittieren Sie mit "Yes".



³ Sollte das Board gewechselt werden, muss normalerweise das neue Board per Hand im Tab „Projekteigenschaften“ als Tool eingestellt werden.

Eine evtl. erscheinende Fehlermeldung klicken Sie bitte weg und starten die Einzelschritt-Ausführung (F11) (evtl. mehrfach versuchen).



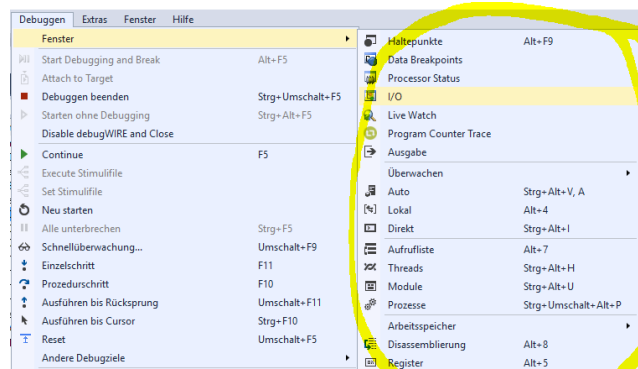
Bei wiederholten Problemen hilft es oft, das Board von der USB-Schnittstelle zu trennen und neu einzustecken⁴.

Hinweise zum Debugging mit debugWire:

- DebugWire verwendet zur Programmierung und Steuerung des Debugvorgangs (Unterbrechung, Haltepunkte, Abrufen von Registerinhalten, usw.) die Resetleitung des ATmega328p. Die normale Resetfunktion und damit auch der Reset-Taster auf dem Carrier-Board kann daher nicht verwendet werden. Der Reset-Taster sollte während einer Datenübertragung nicht betätigt werden, da dies die Kommunikation zwischen Target- und Host-Controller unterbricht.
- Das Programm befindet sich auch nach dem Beenden des Debug-Vorgangs (Strg+Umschalt+F5) noch im Programmspeicher und wird automatisch neu gestartet. Wenn man auf den Reset-Taster verzichten kann, kann das Board somit normalerweise im Debug-Modus bleiben.
- Variablen-Inhalte können während eines Programmhalts (Breakpoint) beobachtet werden. Da der Compiler diese während der Codeerzeugung oft „wegoptimiert“, sollte man Variablen die man beobachten möchte immer als „volatile“ deklarieren. Z.B.:

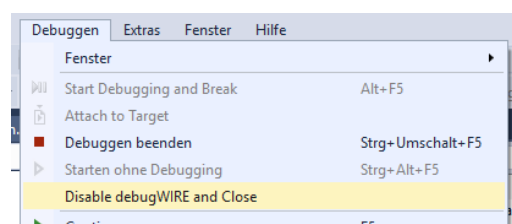
```
volatile uint_8t buffer;
```

- Beobachtungsfenster einrichten mit Debuggen→Fenster→I/O (usw.)



Achtung!!

Die einzige Möglichkeit in den ISP-Modus zurückzukehren besteht darin, während laufendem Debugging den Menüeintrag "**Debuggen>Disable debugWire and close**" zu wählen!



⁴ In seltenen Fällen muss die Fuse DWEN mit dem Programmierwerkzeug per Hand gesetzt werden. Achtung: Danach besteht keine ISP-Verbindung mehr.

Bsp.: Debugging eines Assemblerprogramms

Debugging beenden Einzelschritt Prozedurschritt Reset

Breakpoint

Nächster auszuführender Befehl

z.B. PORTB-Register

The screenshot shows the Atmel Studio IDE in 'Advanced Mode'. The main window displays the assembly file 'main.asm' with the following code:

```

#include "m328pdef.inc"
.equ LED = 5

start:
    ldi r16, 1 << LED
    out DDRB, r16

loop:  ldi r16, 1 << LED
       out PORTB, r16

       ldi r16, 0
       out PORTB, r16

       rjmp loop
    
```

A red dot on the line 'out PORTB, r16' indicates a breakpoint. The I/O window on the right shows the 'I/O Port (PORTB)' selected, with its address 0x25 and value 0x00. The Processor Status window shows the Program Counter at 0x0000018A. The Project Explorer on the far right shows the project 'AssemblerUebungen' with files 'main.asm' and 'main.c'.

Bsp.: Debugging eines C-Programms

The screenshot shows the Atmel Studio IDE in 'Advanced Mode'. The main window displays the C file 'main.c' with the following code:

```

uint8_t temp;

void setup() {
    // ... initialization ...
}

void loop() {
    while (1) // Endlosschleife loop()
    {
        temp = PINB & (1 << S1); // Abfragen der Eingänge mit
        //temp = bit_read(Taster, S1); // Mit Richtlinienfunktion
        if (!temp) // Wenn Taster betätigt (0)
        {
            //LED einschalten:
            // PORTD = PORTD | 0x20; // 1. Möglichkeit
            // PORTD |= 0x20; // 2. Möglichkeit
            // PORTD |= (1 << H5); // 3. Möglichkeit
            bit_write(LED, H5, 1); // 4. Möglichkeit

            delay_ms(500);

            //LED ausschalten:
            // PORTD = PORTD & ~0x20; // 1. Möglichkeit
            // PORTD &= ~0x20; // 2. Möglichkeit
            // PORTD &= ~(1 << H5); // 3. Möglichkeit
            bit_write(LED, H5, 0); // 4. Möglichkeit

            delay_ms(100);
        }
    }
}
    
```

A red dot on the line 'while (1)' indicates a breakpoint. The I/O window on the right shows the 'I/O Port (PORTB)' selected, with its address 0x25 and value 0x00. The Processor Status window shows the Program Counter at 0x0000018A. The Project Explorer on the far right shows the project 'Uebung_1_Bit_I_O' with files 'main.c' and 'main.h'.