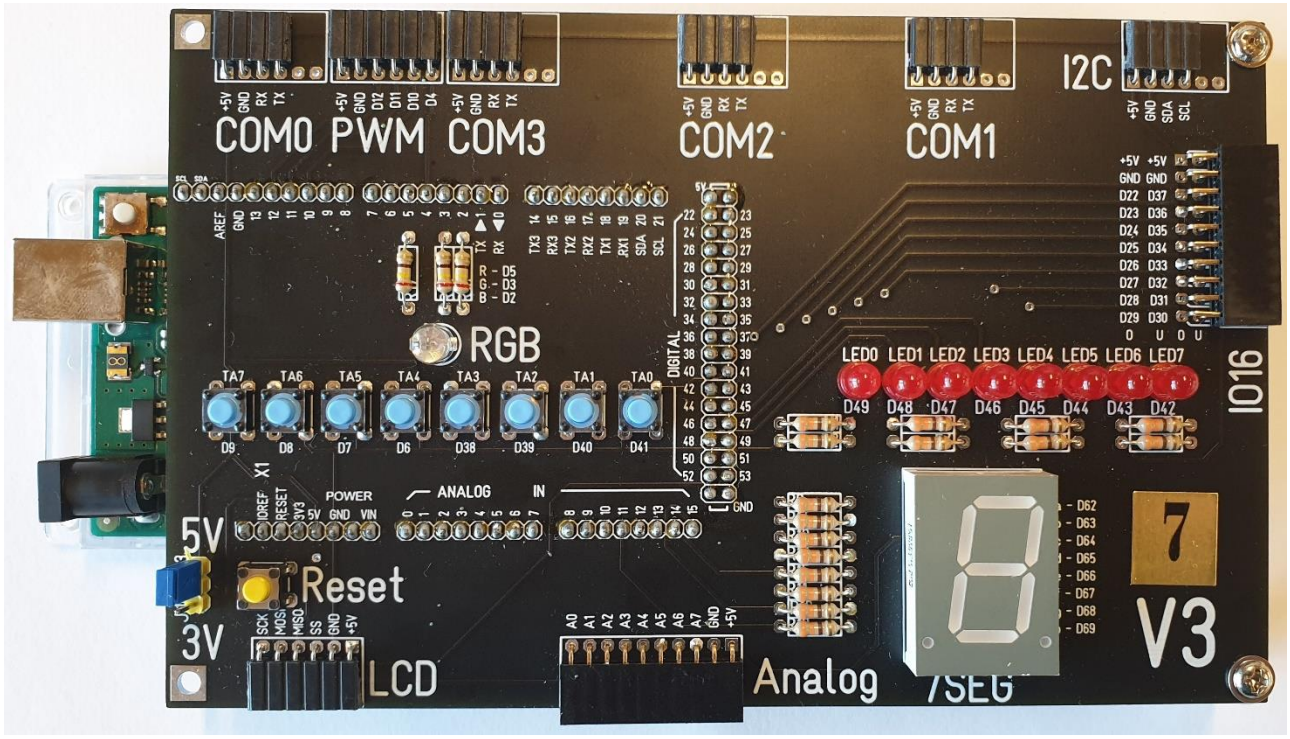
ÜBUNGSÜBERSICHT

Nr.	Übungskurzbeschreibung
1	Blink LED0
2	Lauflicht
3	pinMode() und digitalWrite() analysieren
4	Lauflicht2
5	Leds und Tasten
6	Blinken → Oszilloskop
7	7 Segment Anzeige
8	RGB Led
9	Modul UTIL (UTIL.CPP und UTIL.H) erstellen
10	Stoppuhr (Tasten, Leds, 7 Segmentanzeige und serielle Schnittstelle)
11	Einfache Ampelsteuerung
12	Serielle Schnittstelle (Strings empfangen)
13	AD-Wandler
14	Ampel4Ser
15	7 Segment Anzeige Multiplexing
16	Daten Collector
17	Sensor- und Relaisstation

BOARD



RGB Led	RGB/R	RGB/G	RGB/B
	(PE3) [5]	(PE5) [3]	(PE4) [2]

KEYs	KEY0	KEY 1	KEY 2	KEY 3	KEY 4	KEY 5	KEY 6	KEY 7
	(PG0) [41]	(PG1) [40]	(PG2) [39]	(PD7) [38]	(PH3) [6]	(PH4) [7]	(PH5) [8]	(PH6) [9]

LEDs	LED0	LED1	LED2	LED3	LED4	LED5	LED6	LED7
	(PL0) [49]	(PL1) [48]	(PL2) [47]	(PL3) [46]	(PL4) [45]	(PL5) [44]	(PL6) [43]	(PL7) [42]

7SEG	SEG7a	SEG7b	SEG7c	SEG7d	SEG7e	SEG7f	SEG7g	SEG7dp
	(PK0) [62]	(PK1) [63]	(PK2) [64]	(PK3) [65]	(PK4) [66]	(PK5) [67]	(PK6) [68]	(PK7) [69]

LED0 … D1 … PORTL Bit0 … Arduino Pin 42
KEY0 … S1 … PORTG Bit 0 … Arduino Pin 41

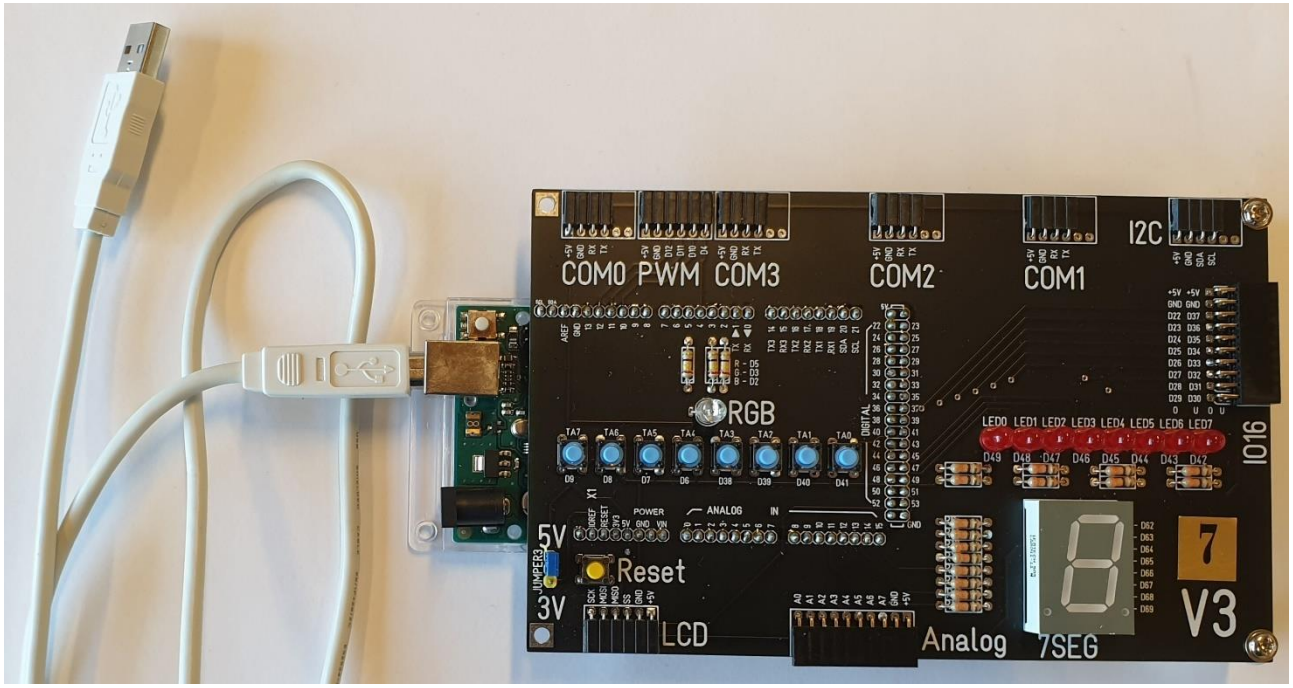
VORBEREITENDE ARBEITEN

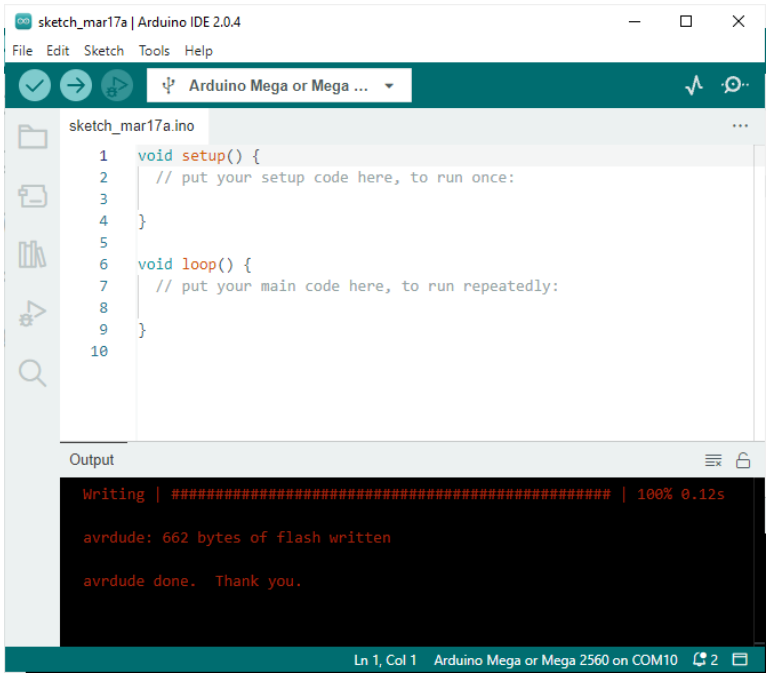
Legen Sie ein Verzeichnis an, in dem alle Arduino-Sketches gespeichert werden.
Beim Speichern eines Sketches wird automatisch ein weiteres Verzeichnis mit dem Sketchnamen erstellt.

z.B.: D:\Work\Arduino\Labor

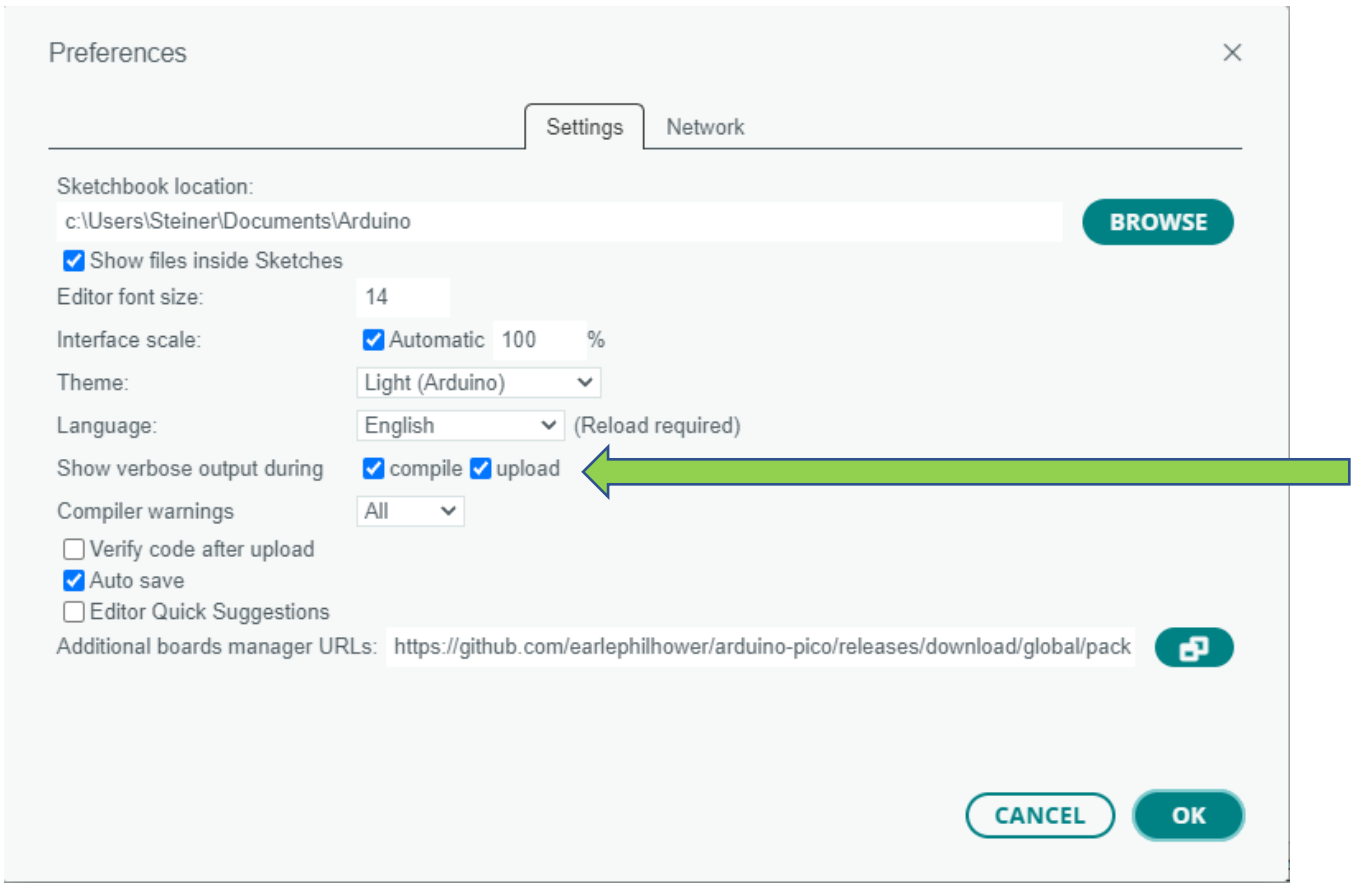
→ Beim Speichern des Sketches ue00_blink wird das Verzeichnis ue00_blink erstellt und in dieses Verzeichnis die Datei ue00_blink.ino gespeichert.
D:\Work\Arduino\Labor\ue00_blink\ue00_blink.ino

Verbinden Sie das Board mit dem Computer und starten Sie die Arduino-IDE.

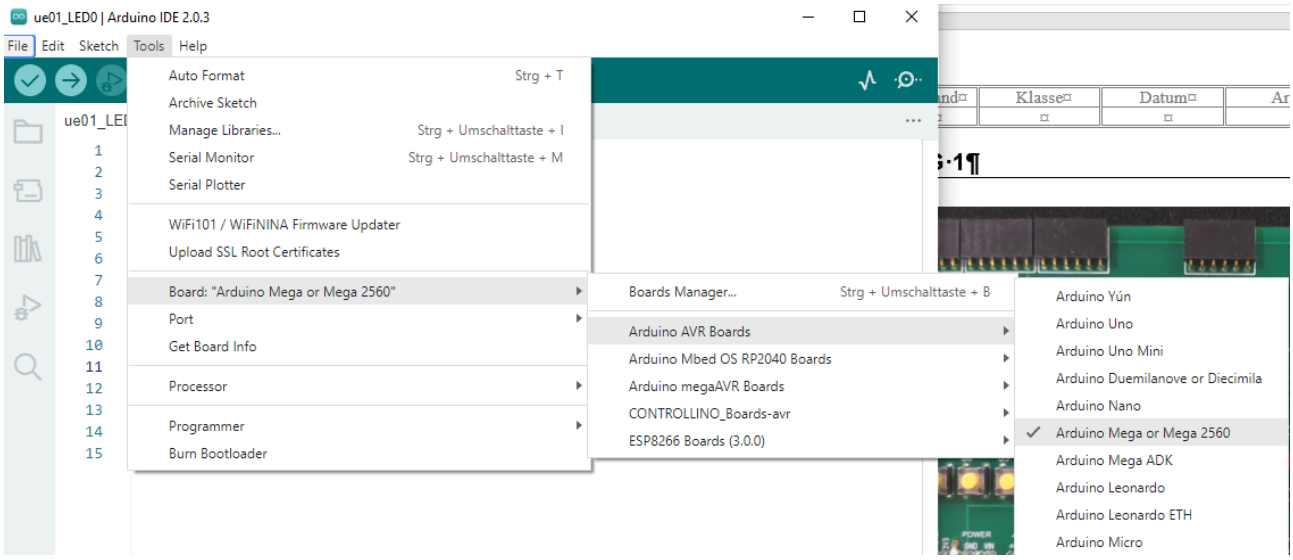




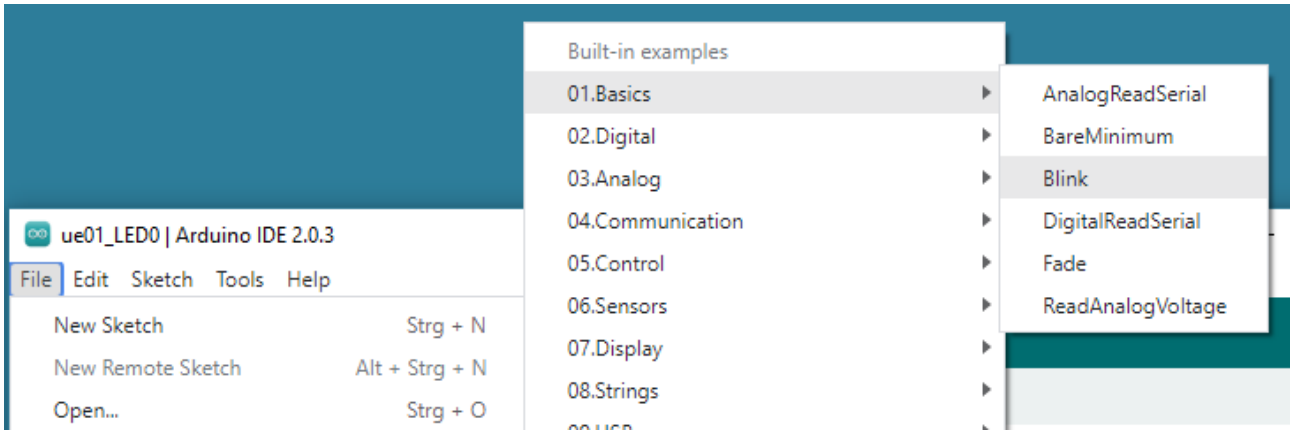
Bei den Voreinstellungen (File – Preferences …)
Show verbose output during … einschalten



Stellen Sie sicher, dass das Board „Arduino Mega or Mega 2560“ eingestellt ist.



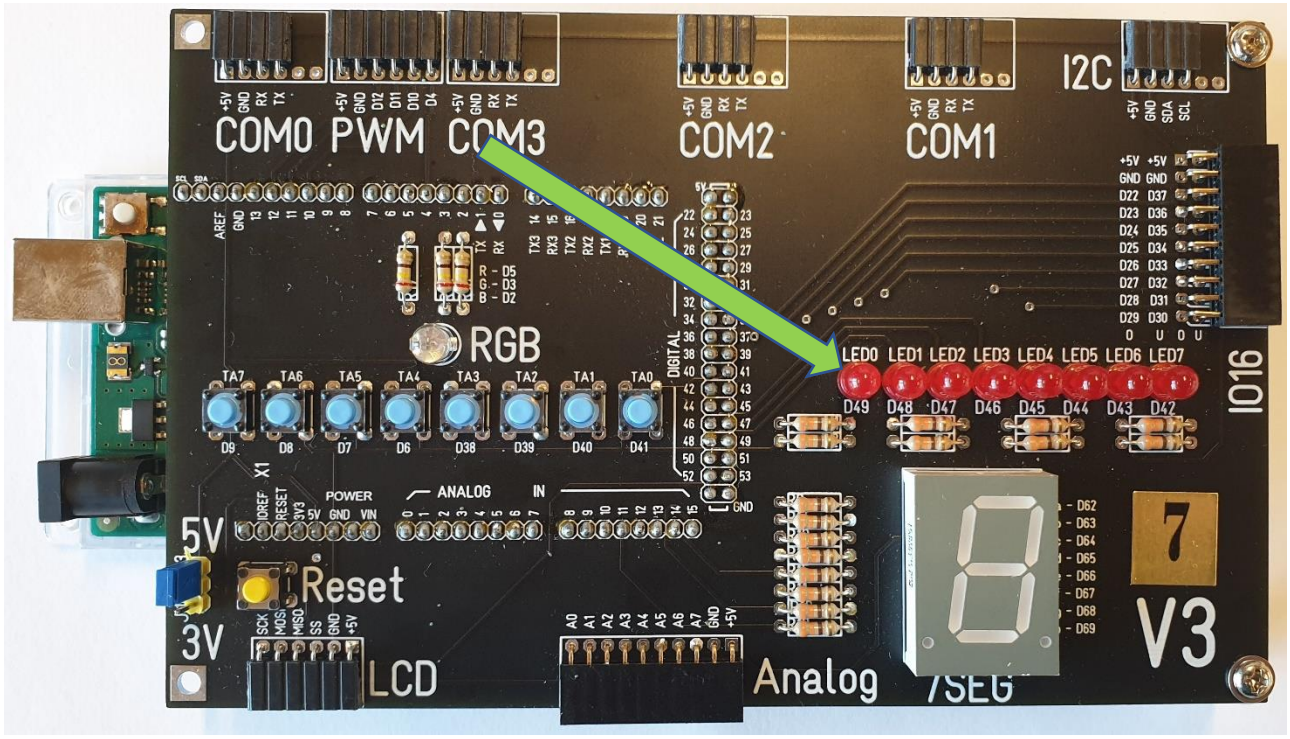
Anschließend öffnen Sie das Blink-Beispiel und speichern Sie es unter ….\ue00_blink.



Nach dem Compilieren und Linken (→ Upload) blinkt am Arduino Board die eingebaute (built in) LED mit einer Frequenz von 0,5 Hz.

```
/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

ÜBUNG 1



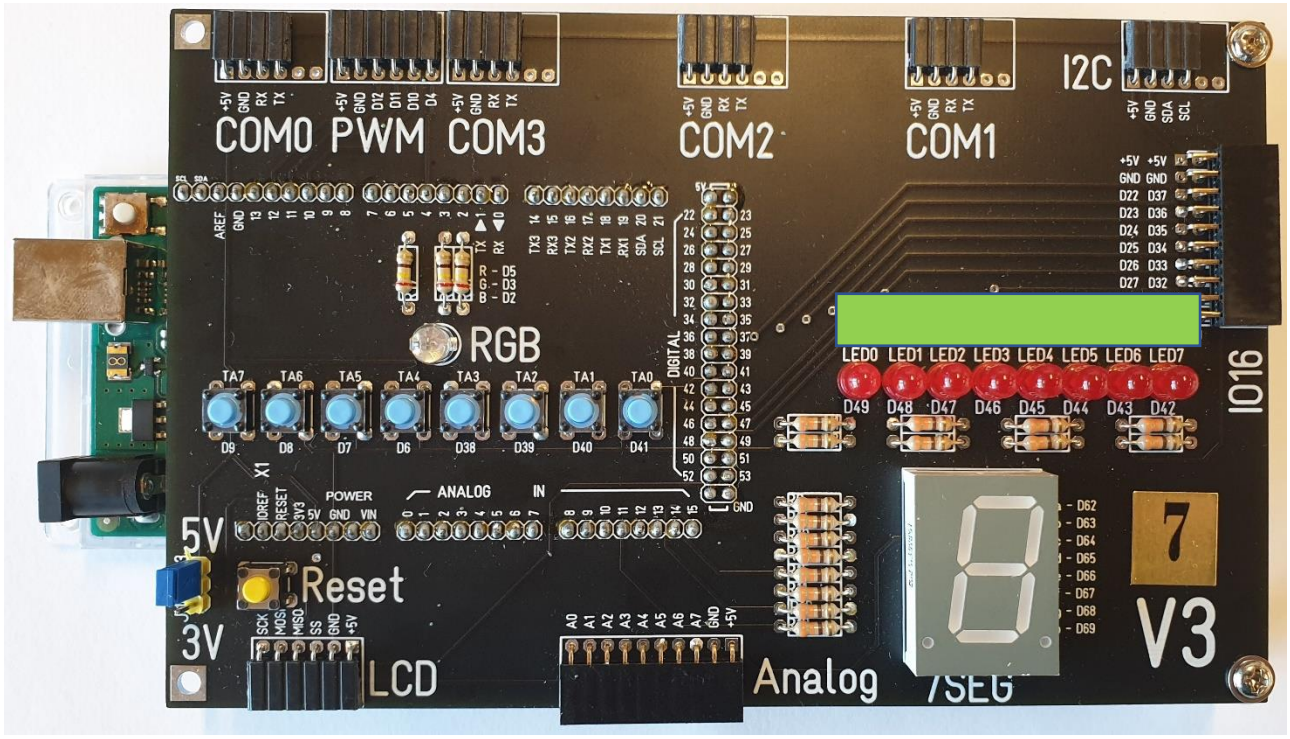
Speichern Sie das erste Beispiel unter ue01_LED0.
Entfernen Sie alle Kommentarzeilen und ändern Sie die blinkende LED auf LED0 mit der korrekten Nummer (siehe Seite 1).

#define LED0 ___

```
#define LED0 __

void setup()
{
  pinMode(LED0, OUTPUT);
}

void loop()
{
  digitalWrite(LED0, HIGH);
  delay(1000);
  digitalWrite(LED0, LOW);
  delay(1000);
}
```

→ LED0 (D1) blinkt mit 0,5 Hz.

ÜBUNG 2



Speichern Sie ue01_LED0 unter ue02_Lauflicht.
Schreiben Sie ein Lauflichtprogramm, LED0 – LED1 – LED2 … LED7 – LED0 mit jeweils 200 ms Leuchtzeit.

a) einfache Lösung
b) verbesserte Lösung (mit Schleife)
c) Annahme: die Led Nummern sind nicht sortiert (49-42), sondern 8 beliebige Zahlen zwischen 0 und 69. Wie sieht dann die Lösung mit Schleife aus?
d) Lösung direkt mit PORT-Zugriff → Übung 4

ÜBUNG 3:

Analysieren Sie die Datei **main.cpp** und in der Datei **wirig_digital.c** die Funktionen ***pinMode()*** und ***digitalWrite()***.
Suchen Sie den vollständigen Ablauf der Funktionen pinMode() und digitalWrite() im Anhang 1.

main.cpp

```
/*
  main.cpp - Main loop for Arduino sketches
  Copyright (c) 2005-2013 Arduino Team.  All right reserved.
  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.
  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.
  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

#include <Arduino.h>

// Declared weak in Arduino.h to allow user redefinitions.
int atexit(void (* /*func*/ )()) { return 0; }

// Weak empty variant initialization function.
// May be redefined by variant files.
void initVariant() __attribute__((weak));
void initVariant() { }

void setupUSB() __attribute__((weak));
void setupUSB() { }

int main(void)
{
     init();
     initVariant();

#if defined(USBCON)
     USBDevice.attach();
#endif

     setup();

     for (;;)
     {
          loop();
          if (serialEventRun)
               serialEventRun();
     }
     return 0;
}
```

ue01_LED0.ino

```
#define LED0 49

void setup()
{
  pinMode(LED0, OUTPUT);
}

void loop()
{
  digitalWrite(LED0, HIGH);
  delay(1000);
  digitalWrite(LED0, LOW);
  delay(1000);
}
```

FRAGEN

I) Erklären Sie die Funktionsweise von main.cpp.

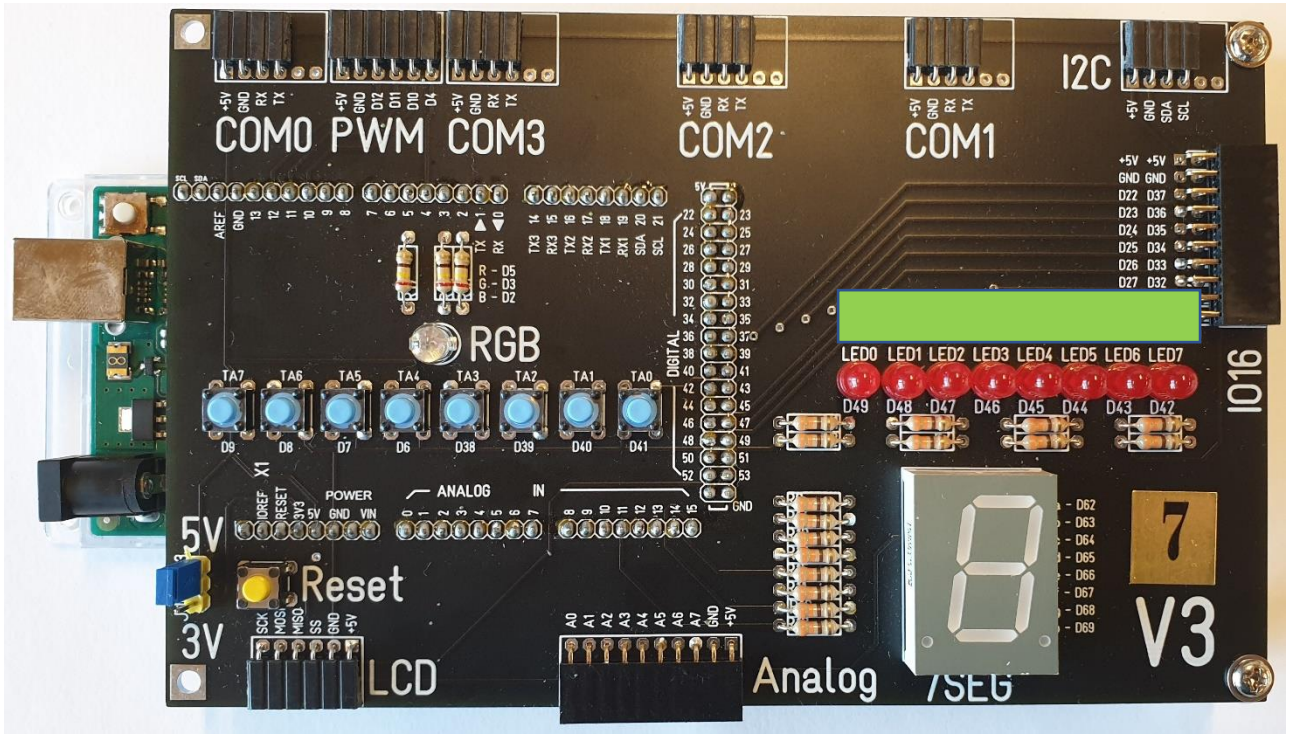
II) Analysieren Sie die Funktion setup() bis ins letzte Detail (inkl. aller Funktionsaufrufe und Feldzugriffe). [Siehe Anhang 2]

III) Schreiben Sie die Funktion setup() mit optimiertem Portzugriff neu.

IV) Analysieren Sie die Funktion loop() bis ins letzte Detail (inkl. aller Funktionsaufrufe und Feldzugriffe außer delay()).

V) Schreiben Sie die Funktion loop() mit optimiertem Portzugriff neu.

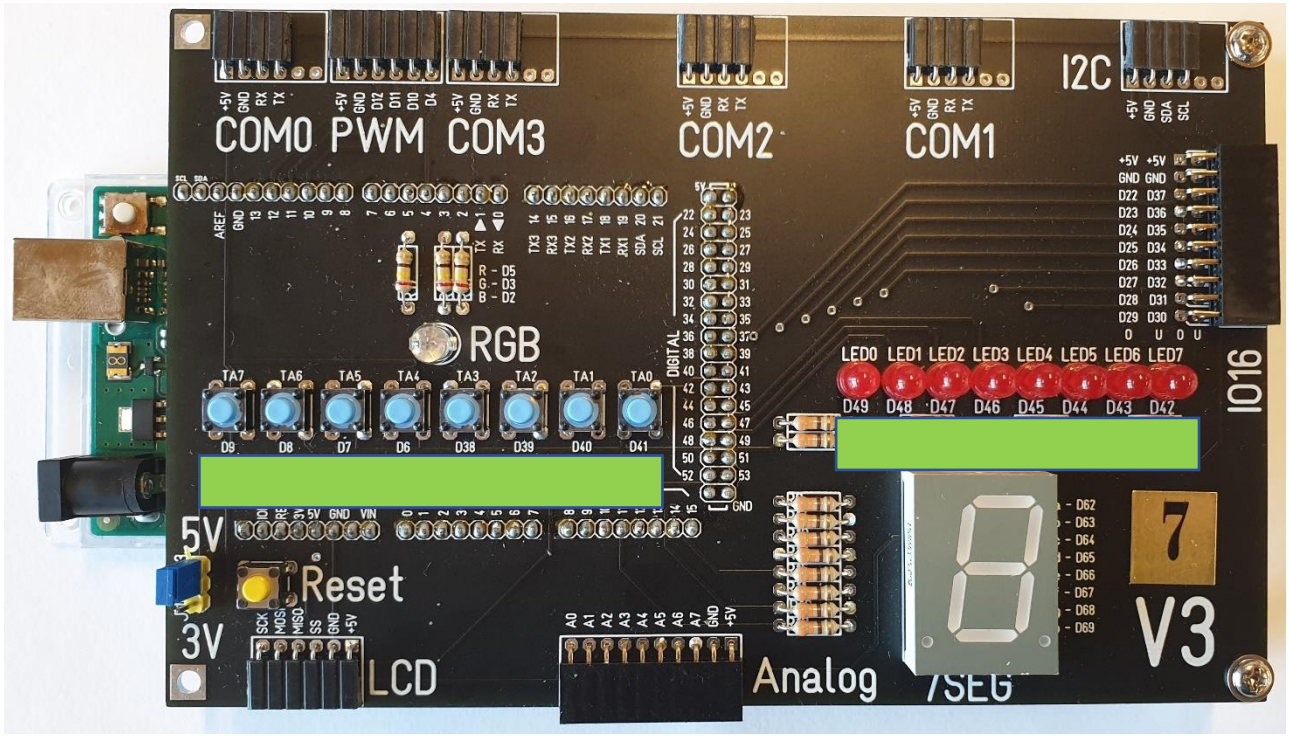
ÜBUNG 4



Schreiben Sie ein Lauflichtprogramm, LED0 – LED1 – LED2 … LED7 – LED0 mit jeweils 200 ms Leuchtzeit direkt mit Portzugriff.

a) einfache Lösung
b) verbesserte Lösung: Die Funktion delay darf nur 1x im Programm aufgerufen werden.

ÜBUNG 5



Schreiben Sie ein Programm, das die Tasten auf den Leds abbildet.
Taste 0 (S1) → LED0 (D1)
:
Taste 7 (S8) → LED7 (D8)

Verwenden Sie die Funktion int digitalRead(uint8_t pin).
Für die Initialisierung der Input-Pins wird

```
pinMode(pin, INPUT_PULLUP);
```

benötigt.

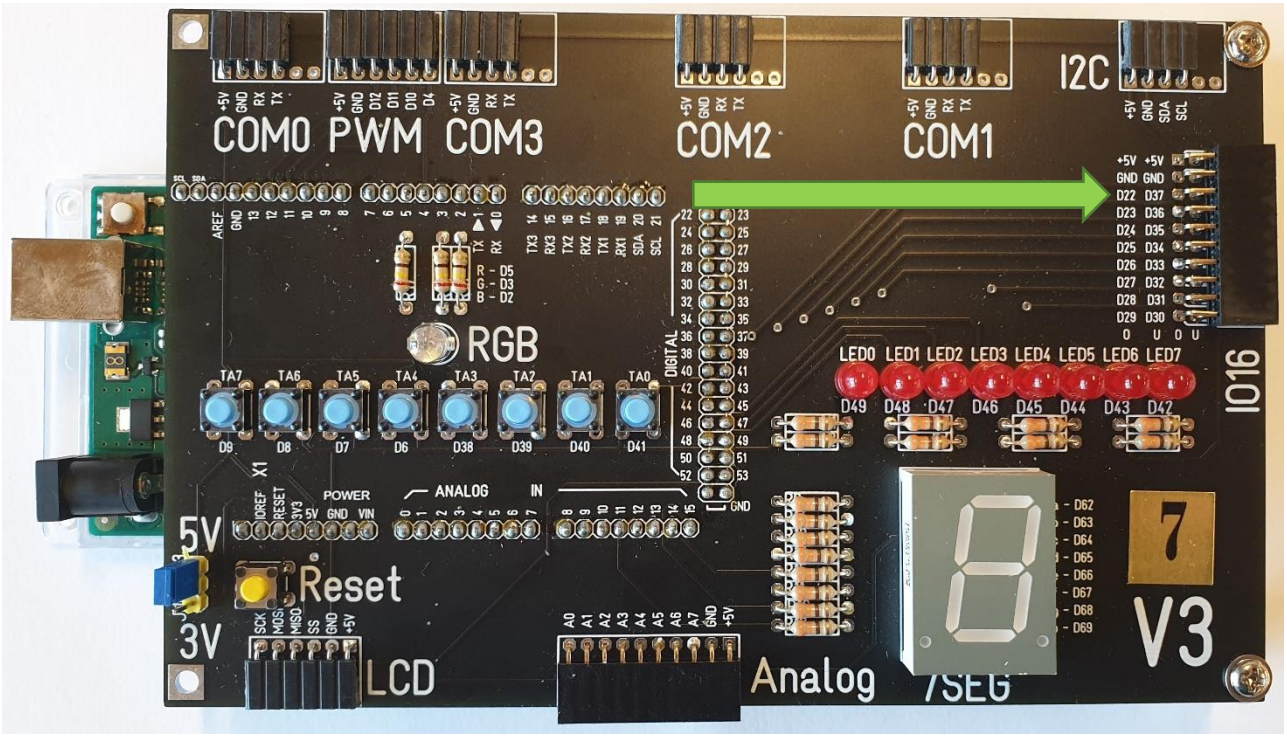
a) direktes Programmieren
b) Schreiben Sie 3 Funktionen

bool kbhit(void)	…	gibt zurück ob eine Taste gerdrückt ist
uint8_t getch(void)	…	wartet auf Tastendruck und liefert die Nummer der gedrückten Taste zurück, und
setLed(uint8_t ledNr, uint8_t on)	…	schaltet eine Led ein oder aus

c) Erweitern Sie uint8_t getch(void) → Erklärung notwendig

d) Optimieren Sie die Funktionen durch direkten Speicherzugriff (PINx – Register).

ÜBUNG 6



a) Ändern Sie das Programm in Übung 1 so, dass der Pin statt LED0 auf IO16O0 (IO16 Port – obere Reihe – Pin 0; PA0 bzw. Arduino Pin 22) gesetzt wird und löschen Sie die zwei delay() Zeilen.
Messen Sie mit einem Oszilloskop die Frequenz (Maximalfrequenz) des Augsgangssignals.
Gibt es Auffälligkeiten im Signalverlauf?
f_{max} = _____

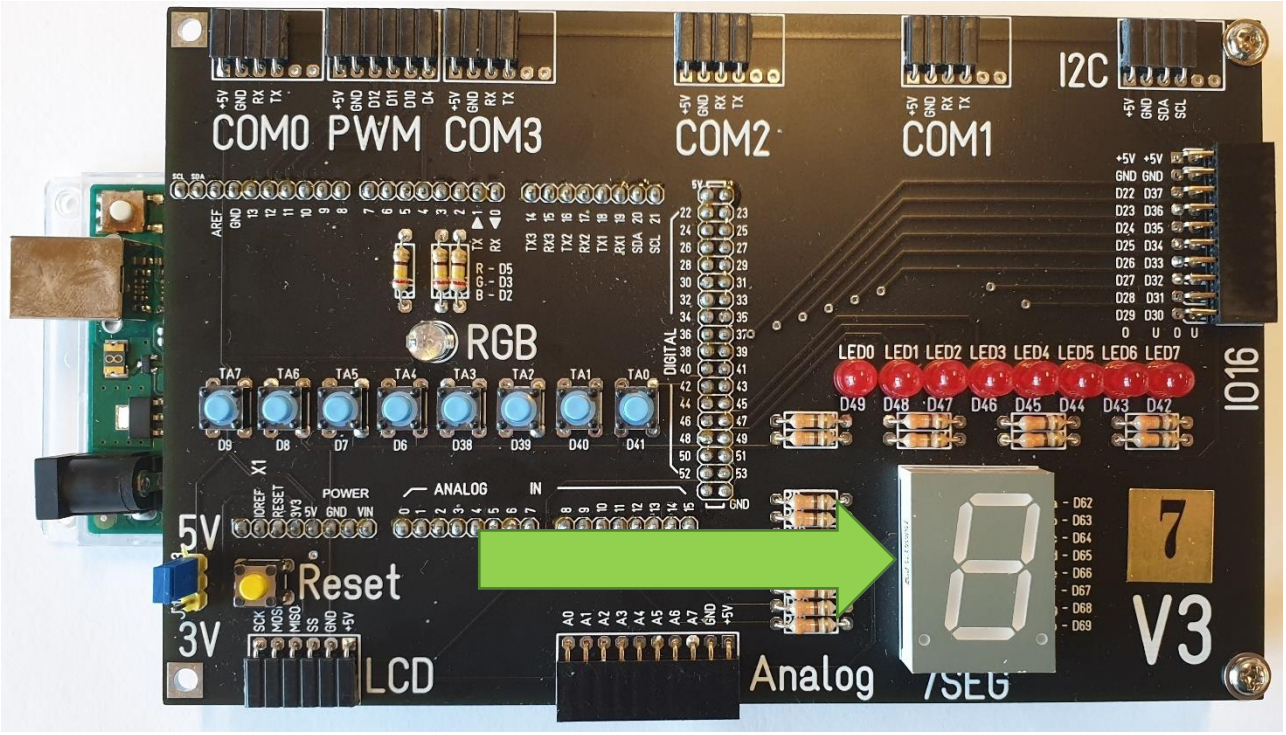
b) Download des fertigen Hex-Files la06_PA0.hex mit Hilfe von AVRDUDE.HEX
Messen Sie mit einem Oszilloskop die Frequenz (Maximalfrequenz) des Ausgangssignals.
Gibt es Auffälligkeiten im Signalverlauf?
f_{max} = _____

SYNTAX VON AVRDUDE.EXE

```
D:\avrdude\bin\avrdude -C D:\avrdude\etc\avrdude.conf -v -v -p atmega2560 -c wiring -P COM3 -b 115200 -D -Uflash:w:la06_PA0.hex:i
```

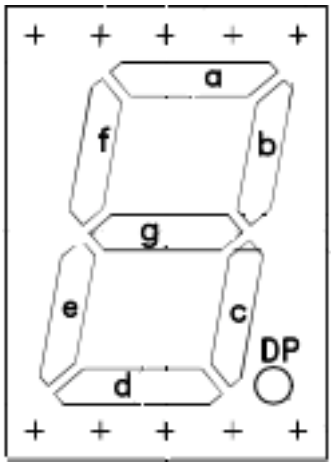
D:\avrdude\bin\avrdude	---	Programmaufruf
-C D:\avrdude\etc\avrdude.conf	---	Konfigurationsdatei von avrdude
-v -v	---	Verbose verbose Mode: umfangreiche Ausgabe
-p atmega2560 -c wiring	---	ArduinoMega2560
-P COM3	---	Schnittstelle (ggf. anpassen)
-b 115200	---	Baudrate
-D -Uflash:w:la06_PA0.hex:i	---	Pfad der Hexdatei

ÜBUNG 7



a) Schreiben Sie ein Programm, das im Sekundentakt die Ziffern 0 bis 9 auf der 7-Segmentanzeige darstellt. Nach der Ziffer 9 soll mit der Ziffer 0 fortgesetzt werden.

	SEG7a	SEG7b	SEG7c	SEG7d	SEG7e	SEG7f	SEG7g	SEG7dp
7SEG	(PK0) [62]	(PK1) [63]	(PK2) [64]	(PK3) [65]	(PK4) [66]	(PK5) [67]	(PK6) [68]	(PK7) [69]



b) Schreiben Sie eine Funktion
void setSeg7(uint8_t nr),
die die Zahl auf der 7 Segmentanzeige ausgibt.

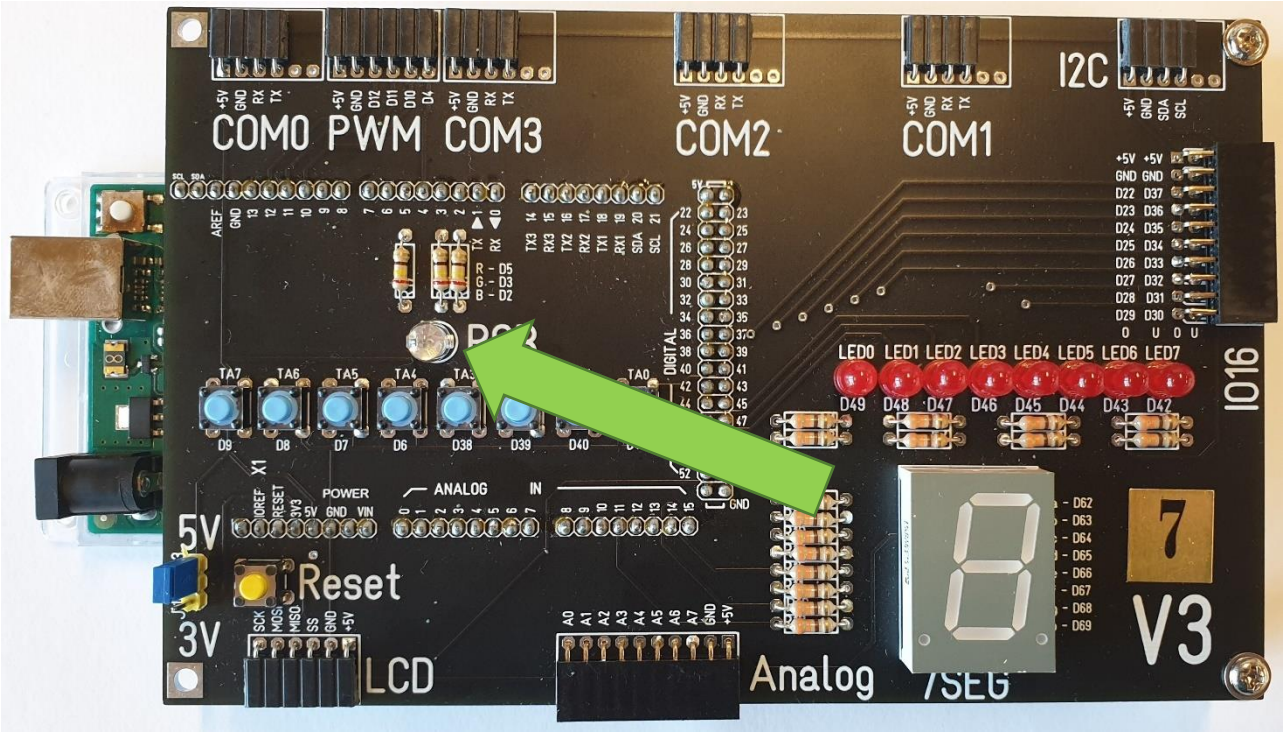
c) Ändern Sie das Programm so ab, dass zusätzlich zum Zählen Übung 5 implementiert wird, also die Nummer der gedrückten Taste auf den Leds abgebildet wird.

d) Verbessern Sie das Programm so, dass die gedrückte Taste unmittelbar angezeigt wird und nicht erst mit einer Verzögerung von bis zu einer Sekunde.

Tipp: Verwenden Sie die Funktion millis() statt delay()

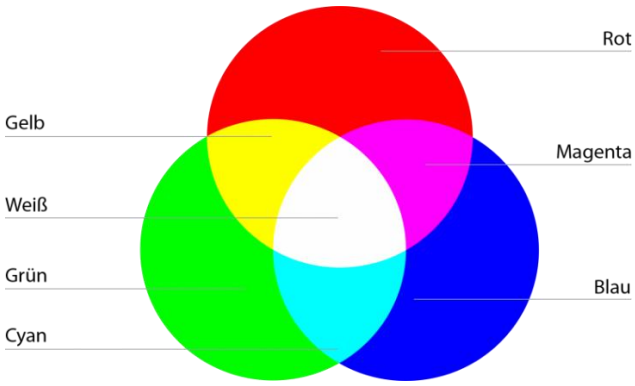
Variante 1: Lösung mit Arduino Funktionen
Variante 2: Lösung mit direktem Zugriff auf die Ports.

ÜBUNG 8



a) Schreiben Sie ein Programm, das die Farben der RGB-LED hintereinander im Sekundentakt darstellt.
Die Farbnummer soll auf der 7 Segmentanzeige ausgegeben werden.

	RGB/R	RGB/G	RGB/B
RGB Led	(PE3) [5]	(PE5) [3]	(PE4) [2]



b) Erweitern Sie das Programm so, das die Farbe nach dem Tastendruck dargestellt wird (KEY0 → Farbe 0, KEY1 → Farbe 1, … , KEY7 → Farbe 7).

Verwenden Sie die zuvor erstellten Funktionen (kbhit, getch und setSeg7).

ÜBUNG 9

Erstellen Sie aus den selbst programmierten Funktionen ein Modul "UTIL", bestehend aus UTIL.CPP und UTIL.H.
In UTIL.H stehen nur die Funktionsprototypen und in UTIL.CPP die vollständigen Funktionen.
Für die Verwendung des Moduls sind beide Dateien ins aktuelle Verzeichnis zu kopieren und UTIL.H ist zu inkludieren.

Funktionen:

```
    bool kbhit(void);
    uint8_t getch(void);
    void setLed(uint8_t ledNr);
    void setSeg7(uint8_t nr);

Weiters sind folgende Funktionen für die Initialisierung sinnvoll
    void initKeys(void);
    void initLeds(void);
    void init7Seg(void);
```

UTIL.H (Damit UTIL.H nur einmal inkludiert wird)

```
    #ifndef _UTIL_H
        #define _UTIL_H
        : (Funktionsprototypen)
    #endif
```

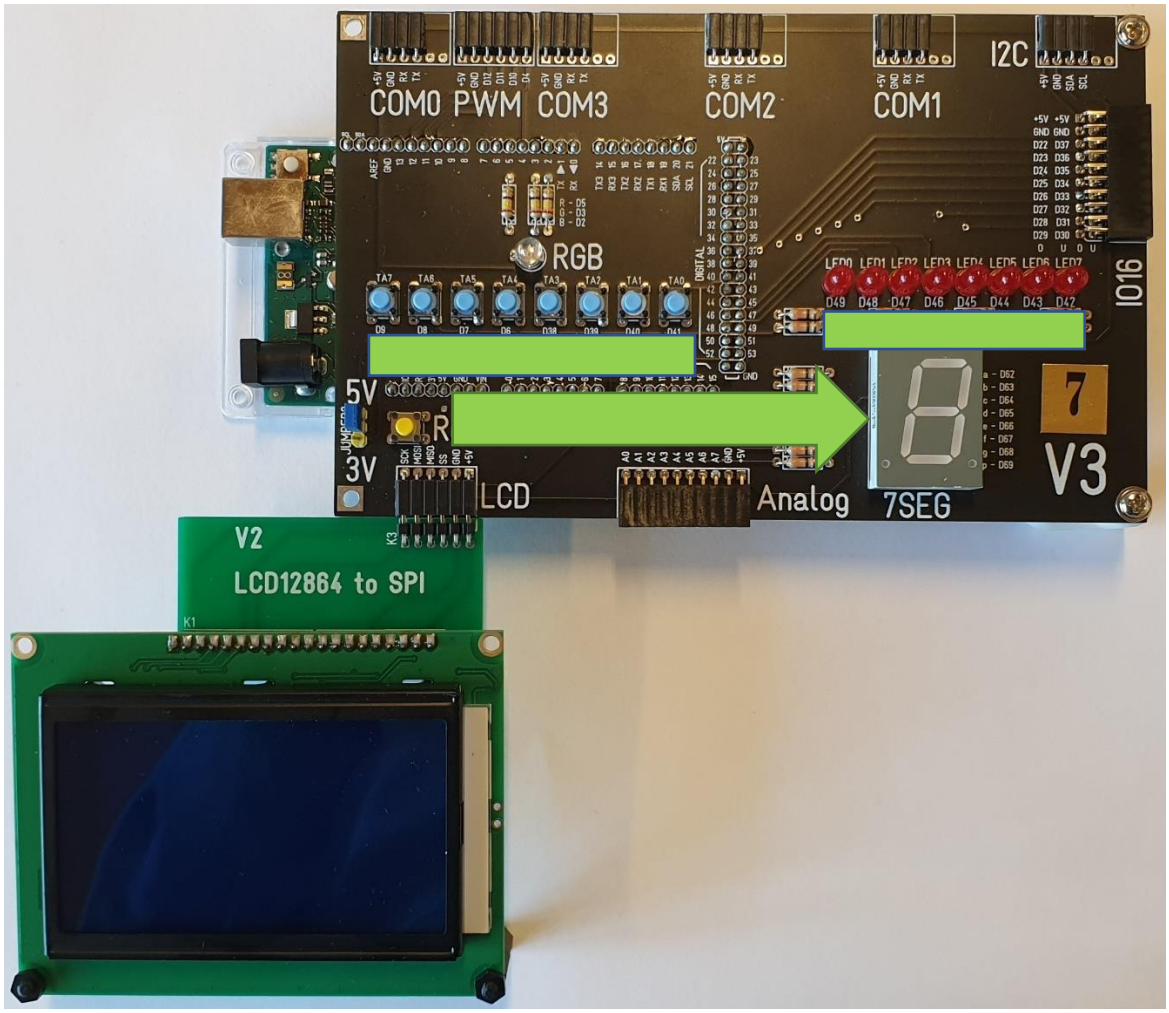
UTIL.CPP

```
    #include "UTIL.H"

    : (Funktionen)
```

Ändern Sie alle Übungsbeispiele so, dass das Modul UTIL zur Anwendung kommt.
Übung 1, Übung 2, Übung 4, Übung 5, Übung 7 und Übung 8
mit zum Beispiel ue05a_keys → ue05a_keys_util

Tipp:
Öffnen Sie das aktuelle Verzeichnis (Sketch – Show Sketch Folder) und erstellen Sie die beiden Dateien UTIL.CPP und UTIL.H.
Wenn die beiden Dateien nicht direkt in der Arduino-IDE angezeigt werden, müssen sie mit Sketch-Add File… eingebunden werden.

ÜBUNG 10



Programmieren Sie eine Stoppuhr mit folgenden Funktionen:

a) Die Auflösung der Stoppuhr ist 1/100 Sekunde
Die 1/10 Sekunden werden auf der 7 Segmentanzeige und die Sekunden auf den Leds (binär oder BCD codiert) ausgegeben.

Tasten:

START	KEY0	…	Starten der Stoppuhr
STOPP	KEY1	…	Stoppen der Stoppuhr
RESET	KEY2	…	Stoppen der Stoppuhr und zurücksetzten
ZZSKI	KEY3	…	Die Anzeige wird für 2 Sekunden angehalten, die Stoppuhr läuft weiter
ZZSTD	KEY4	…	Die Anzeige wird bis zum nächsten Drücken der Taste angehalten, die Stoppuhr läuft im Hintergrund weiter

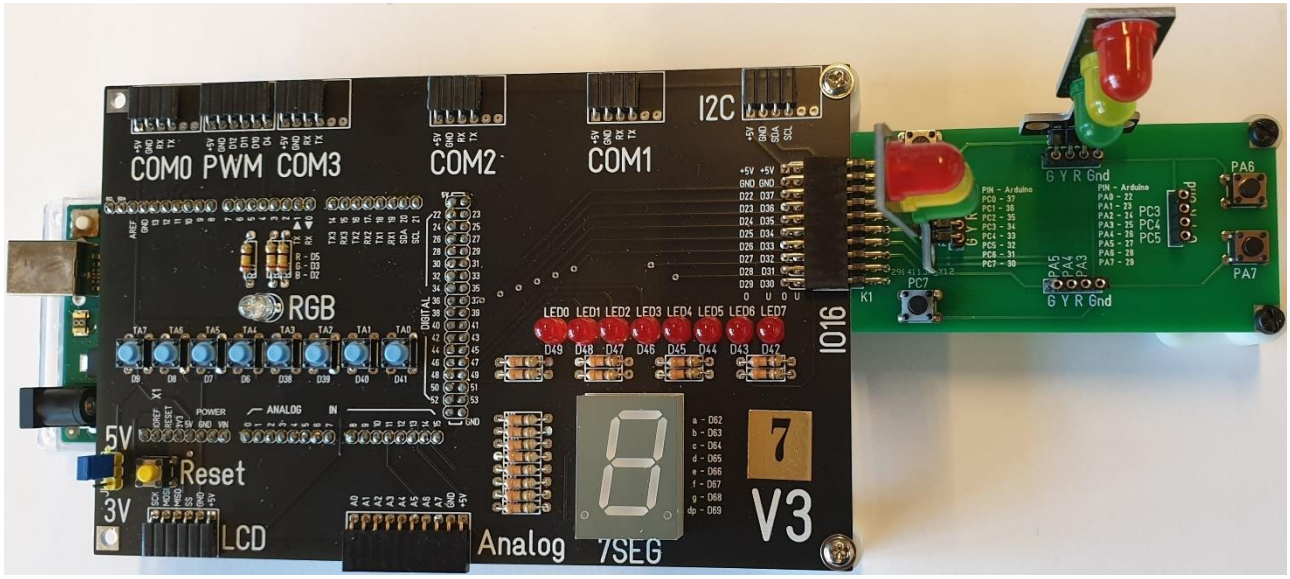
b) Erweitern Sie die Stoppuhr so, dass bei ZZSKI die Zeit (mm:ss.hh) am LCD ausgegeben wird.

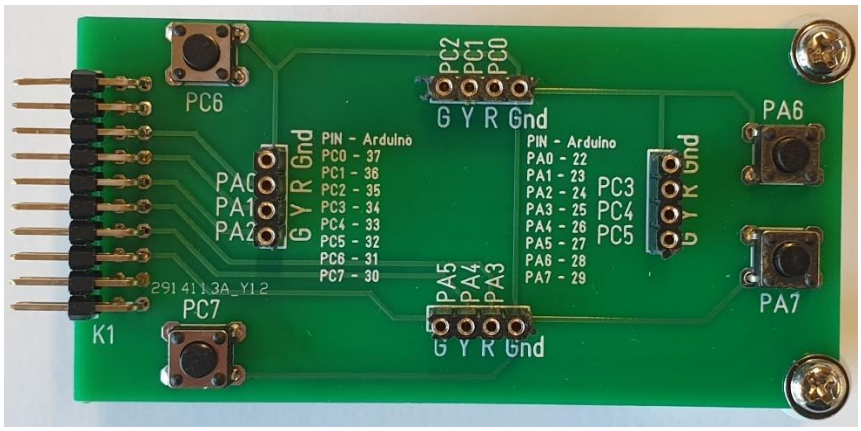
c) Die letzten 4 Zwischenzeiten sollen am LCD dargestellt werden.

d) Beim Drücken von STOP wird die aktuelle Zeit über sie serielle Schnittstelle (9600,N,8,1) in der Form „Zeit: mm:ss.hh“ übertragen.

Verwenden Sie die Module UTIL und LCD1264 für die Lösung der Probleme.
Die Genauigkeit der Stoppuhr kann durch Verwendung von millis() statt delay() erhöht werden.
Überprüfen Sie den gesendeten String mit dem SerialMonitor und mit der Terminalprogramm TeraTerm.

ÜBUNG 11





Programmieren Sie eine Ampelsteuerung

a) Standard Phase mit Ampel Nord
Grün 5s, 4xBlinken 500ms,
Gelb 3s, Rot 5s, Rot-Gelb 3s

b) Ampel Nord leuchtet Grün, bei Tastendruck (PC6) startet eine Standard Phase.

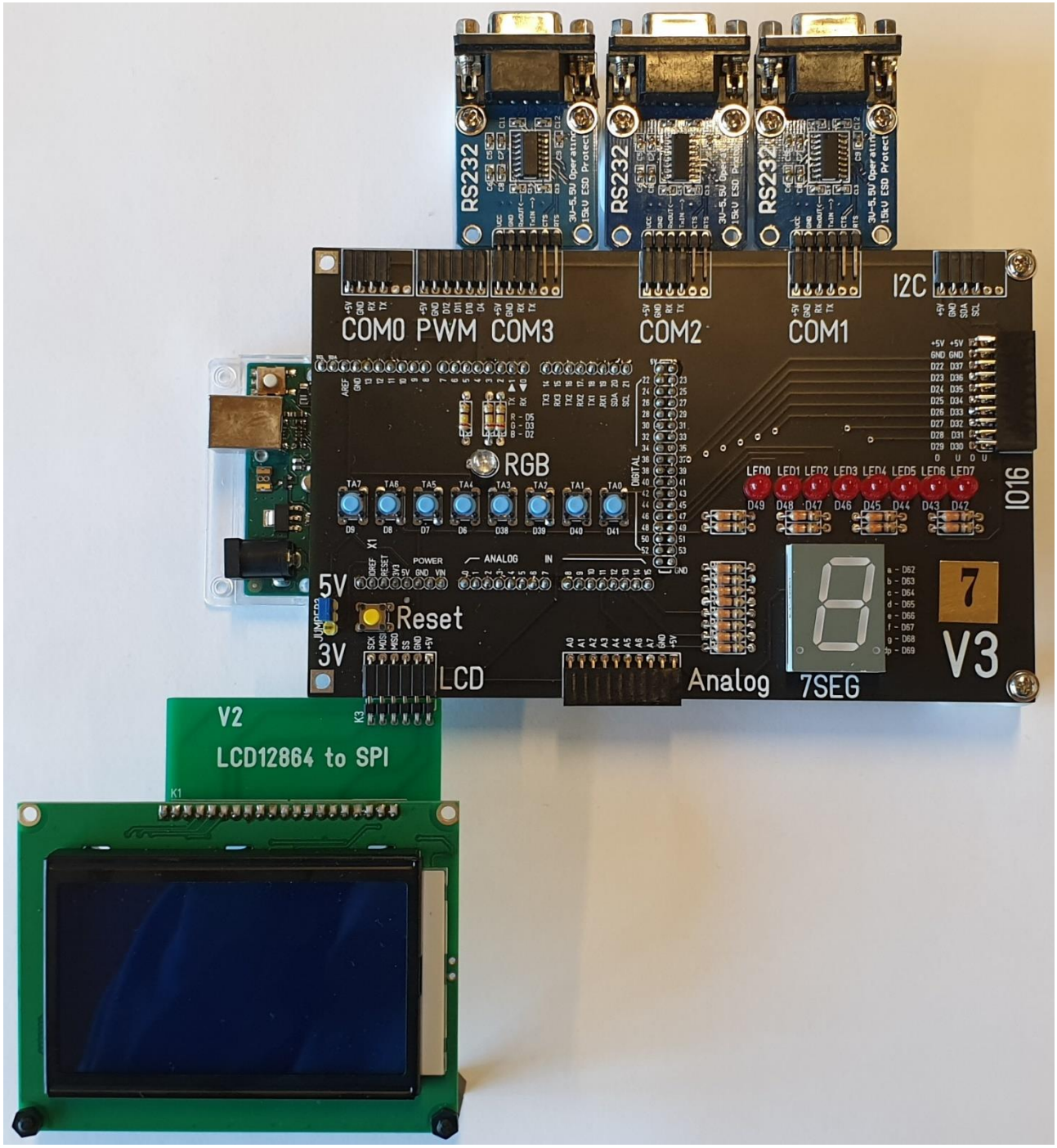
c) Erweitern von b) auf Ampel Nord und Ampel West mit Taster PC7.

d) Beide Ampeln sollen unabhängig voneinander laufen.

e) Ergänzung der Unabhängigkeit auf 4 Ampeln

Tipp: Nachdenken über die Datenstruktur der Ampeln kann nicht schaden!!!

ÜBUNG 12

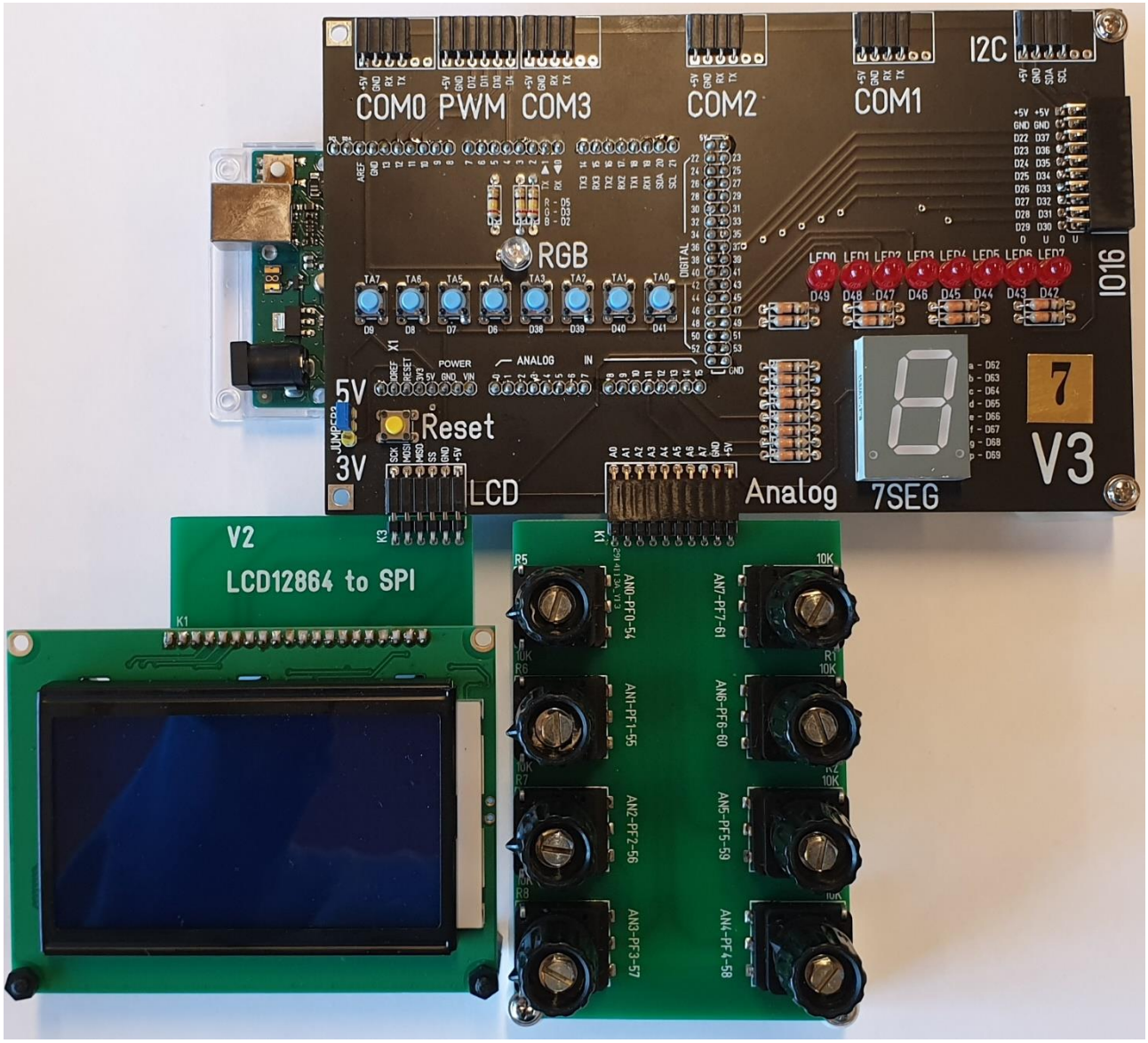


a) Schreiben Sie eine Programm, das einen String über die serielle Schnittstelle empfängt, den String in Großbuchstaben umwandelt, am LCD darstellt und wieder über die serielle Schnittstelle zurücksendet.

b) Erweitern Sie das Programm, so das alle 4 serielle Schnittstellen verwendet werden können.

c) Entwerfen Sie ein Datenübertragungsprotokoll (DÜP), das die Steuerung der Stoppuhr aus Übung 10 übernimmt.
- Steuerung mit einzelnen Zeichen
- Steuerung mit ganzen Strings

ÜBUNG 13



a) Schreiben Sie ein Programm, das die Spannung am Analog Eingang 0 (A0) zyklisch einliest (Zahlenwert 0-1023) und auf den Leds (0-255) ausgibt.
Die Funktion zum Einlesen eines AD-Wertes ist (keine Initialisierung notwendig):

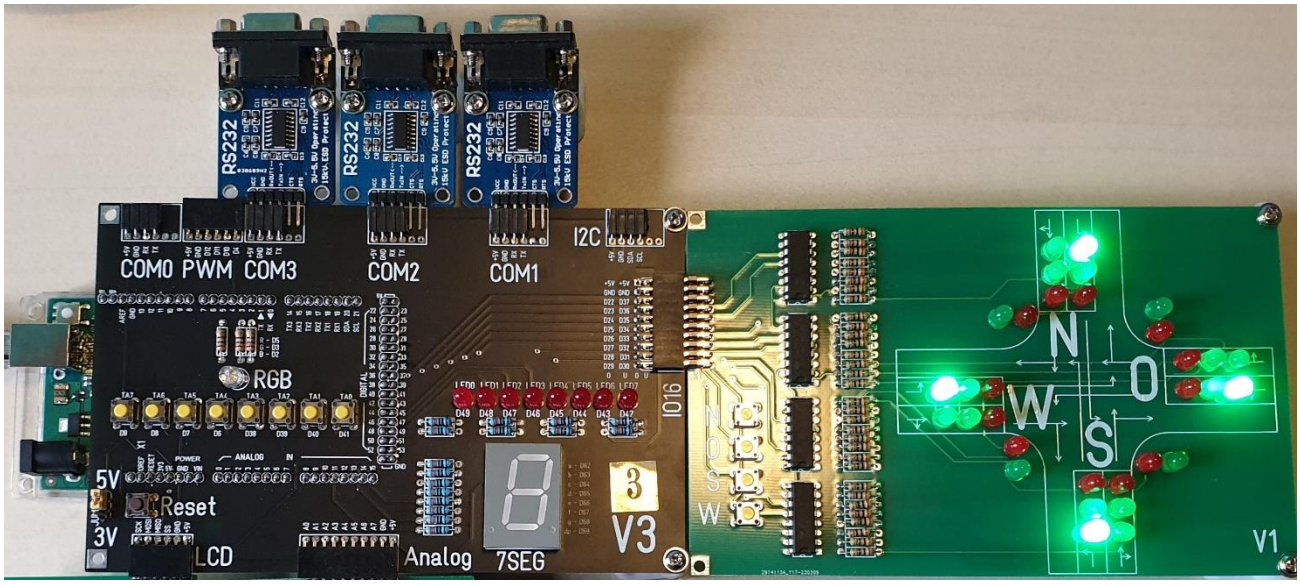
```
uint16_t analogRead(uint8_t analogpin);
```

b) Erweitern Sie das Programm so, dass über die Tasten ein Kanal ausgewählt werden kann, die Nummer des Kanals auf der 7-Segmentanzeige angezeigt und der Analogwert wieder auf den Leds dargestellt wird.

c) Alle 8 Analogkanäle (A0 bis A7) sollen gleichzeitig auf dem LCD dargestellt werden.

d) DÜP erstellen und das Programm so erweitern, dass ein Analogwert über die serielle Schnittstelle angefordert werden kann. Zusätzlich ist das DÜP mit einem Telegramm zu ergänzen, das alle Analogwerte über die serielle Schnittstelle sendet.

ÜBUNG 14



a) Schreiben Sie ein Programm, das bei allen Ampeln die gelben Lichter im Sekundentakt blinken. Verwenden Sie dazu die Funktionen initAmpelSer() und updateAmpelSer() im Modul AmpelSer.

b) Schreiben Sie eine einfache Ampelsteuerung für die Ampel Nord

c) Erweiterung für die Ampeln Nord und Süd

d) Vervollständigen Sie das Programm für alle Ampeln

e) Erweiteung Sie Aufgabenstellung a) so, dass durch einen beliebigen Tastendruck alle Autofahrer im Zyklus Gelb/Rot, …, gezeigt bekommen und die Füßgänger zu richtigen Zeitpunkt Grün. Nach dem Ablauf soll wieder auf Gelb blinken (siehe a)) geschaltet werden.

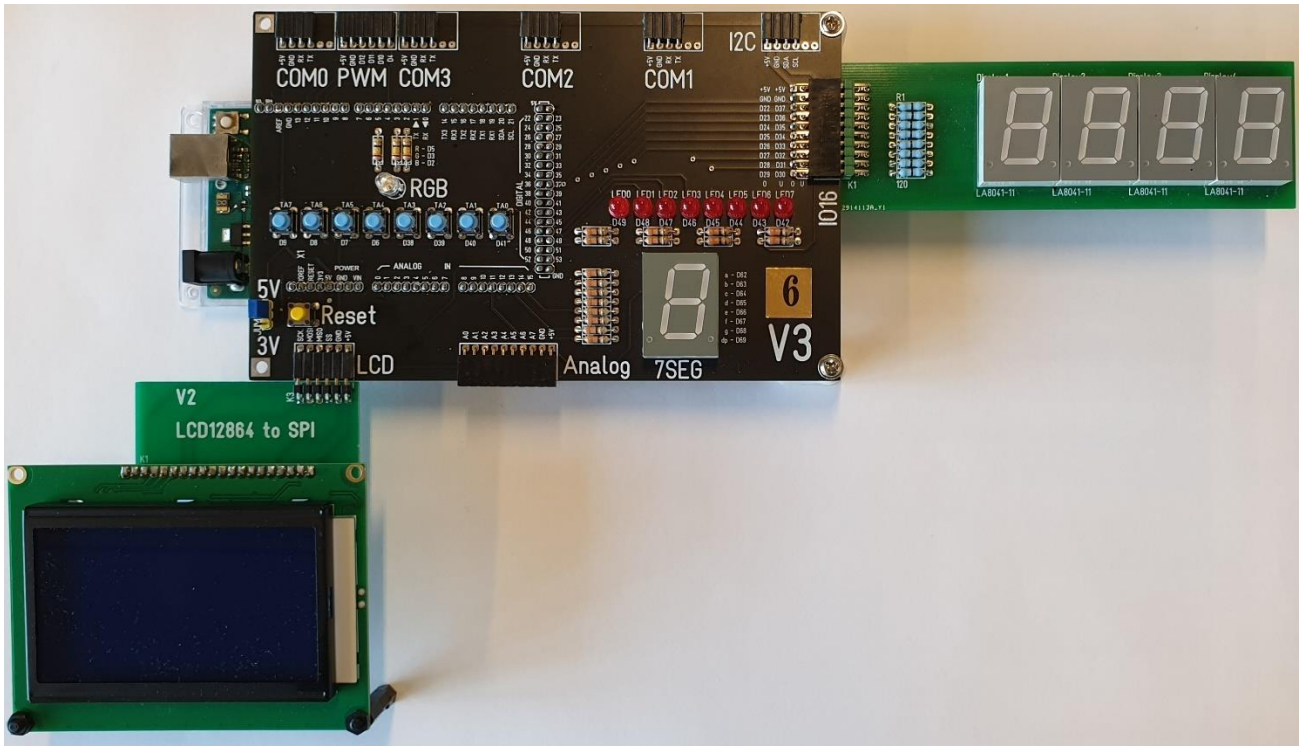
f) Erweitern Sie die Aufgabenstellung d) so, dass das Drücken eines Tasters VORRANG bedeutet. d.h.: Wenn die Taste Nord gedrückt wird, werden (im beschleunigten Zyklus) die Fahrtrichtung Nord auf Grün und alle Anderen auf Rot gesetzt. Nach einer bestimmten Zeit (z.B. 5 Sekunden) soll der Standardzyklus weiter laufen.

g) Entwerfen Sie ein DÜP, das die Ampeln steuern (schreiben und lesen) und die Tasten Abfragen kann

h) Schreiben Sie ein Java-Programm, das die Steuerung einer ApmpelSer durchführt.

i) Erweitern Sie das Java-Programm so, dass beliebig viele Ampeln gleichzeitig gesteruet werden können.

ÜBUNG 15



7SEG	SEG7a	SEG7b	SEG7c	SEG7d	SEG7e	SEG7f	SEG7g	SEG7dp
	(PC0) [37]	(PC1) [36]	(PC2) [35]	(PC3) [34]	(PC4) [33]	(PC5) [32]	(PC6) [31]	(PC7) [30]

Digits	DIG0	DIG1	DIG2	DIG3				
	(PA3) [25]	(PA2) [24]	(PA1) [23]	(PA0) [22]				

Die Segmente sind aktiv LO und die Digits sind aktiv HI

a) Schreiben Sie ein Programm, das im Sekundentakt die Ziffern 0 bis 9 auf der 4 stelligen 7-Segmentanzeige (7Seg4) darstellt. Nach der Ziffer 9 soll mit der Ziffer 0 fortgesetzt werden.

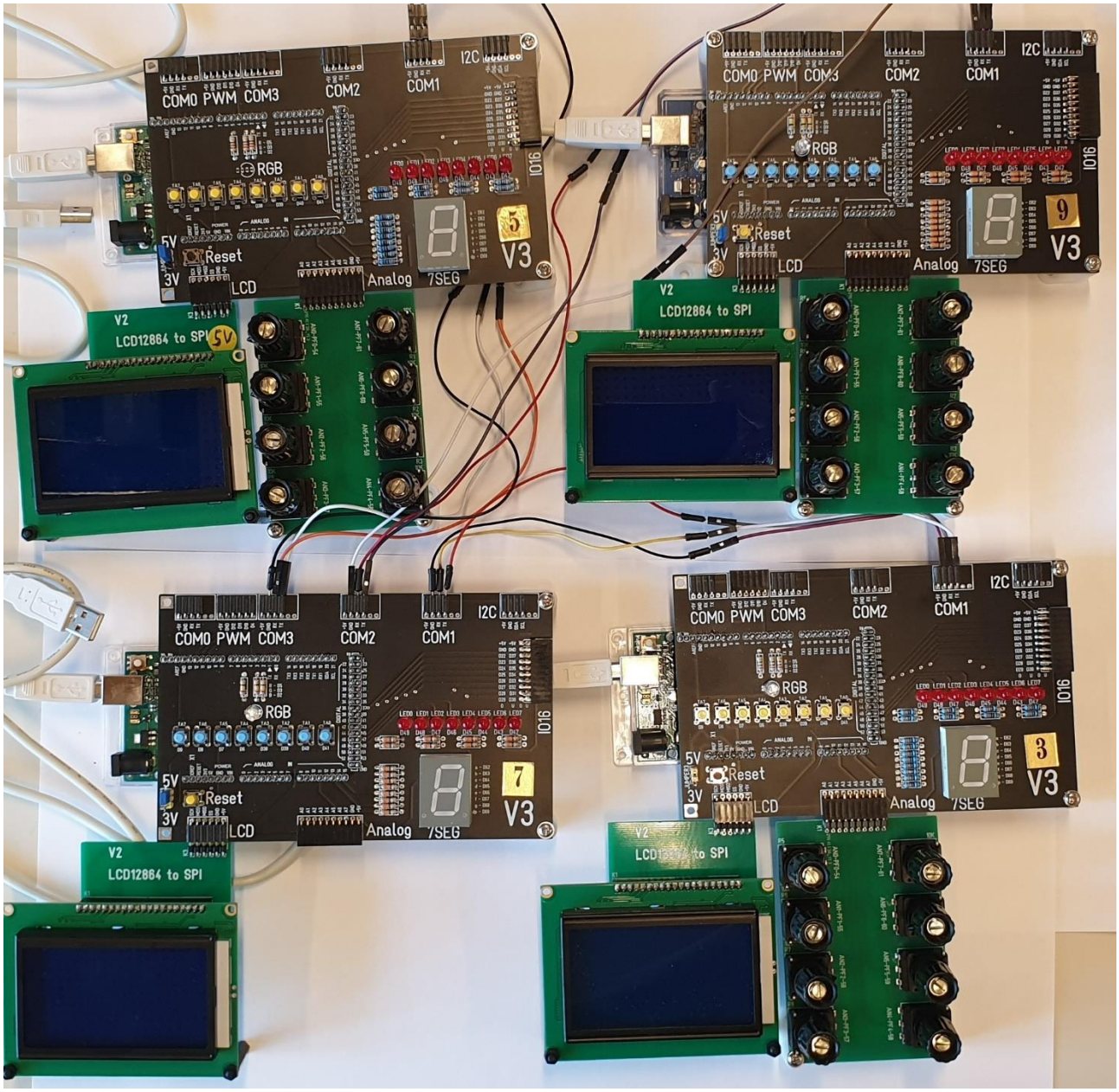
b) Erweitern Sie das Programm so, dass mit den Tasten 0 bis 3 die Stelle (Digit) aktiviert und mit den Tasten 4-7 die Stellen 0-3 wieder deaktiviert werden.

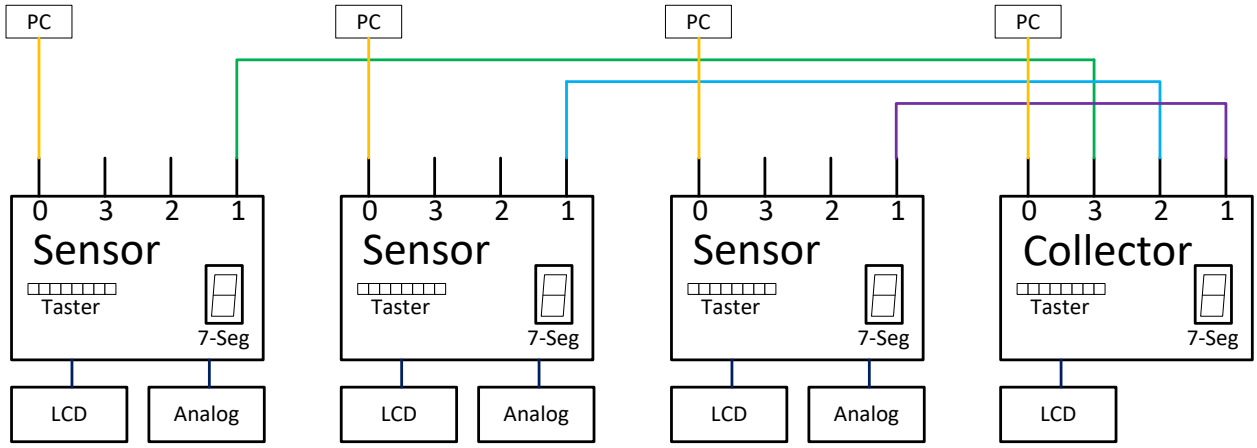
c) Erweitern Sie das Programm so, dass beliebige Zahlenwerte (0000-9999) am 7Seg4 dargestellt werden können. Dazu wird ein Timerinterrupt (z.B.: Timer 1) benötigt.
Timer und Interrupts werden erklärt.

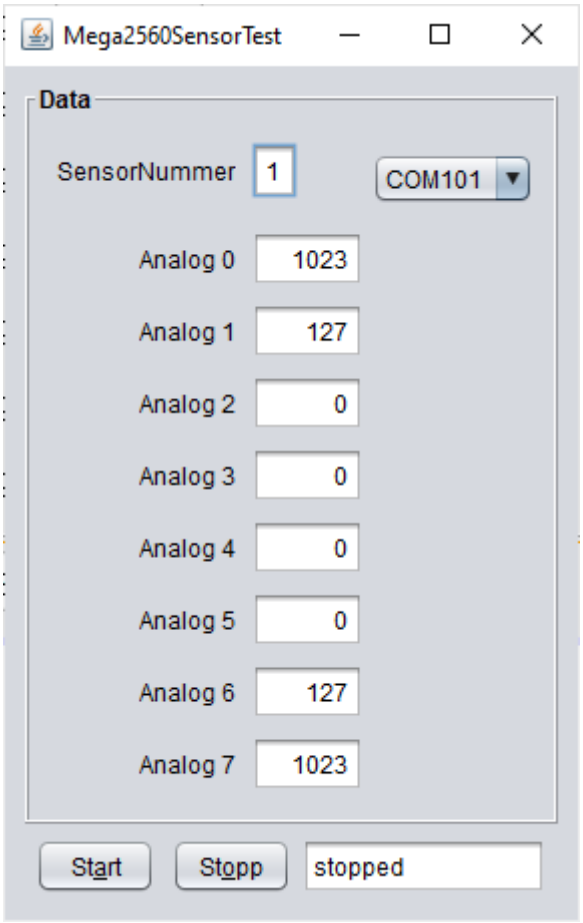
d) Ändern Sie das Programm von Aufgabenstellung 10 so, dass die Sekunden und Hundertstelsekunden auf 7Seg4 und die Minuten auf der 7 Segmentanzeige am Board dargestellt werden.

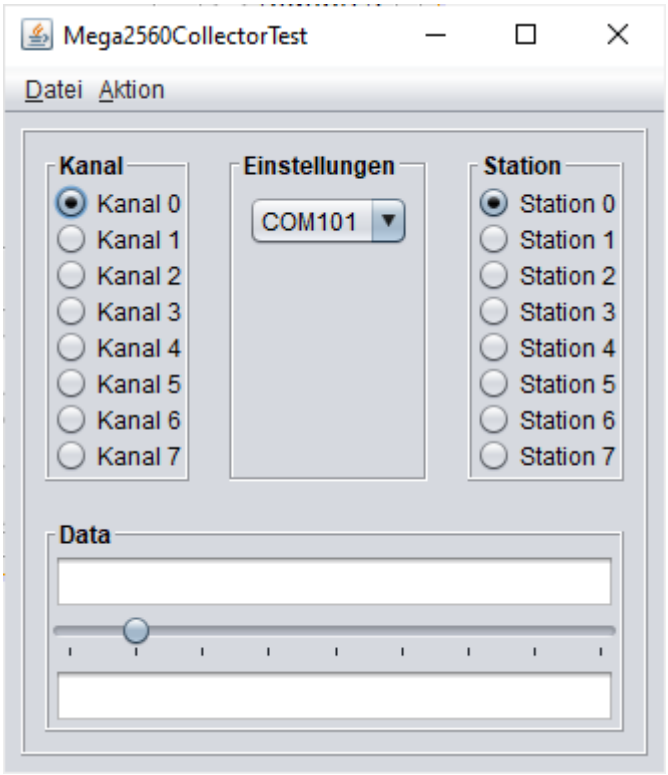
d) Implementieren Sie zusätzlich die Steuerung der Stoppuhr über die serielle Schnittstelle aus Übung 12 (alle 4 Schnittstellen).

ÜBUNG 16


COM0 PWM COM3
COM2
COM1
I2C
RGB
5V
3V
Reset
LCD
Analog
7SEG
V3
V2
LCD12864 to SPI


PC
0 3 2 1
Sensor
Taster
7-Seg
LCD
Analog
Collector





Programmieren Sie folgende Aufgabenstellungen:

a) Nach dem PowerOnReset (POR) wird über die Tasten die Stationsnummer der Sensorstation gewählt und auf den 7 Segmentanzeige ausgegeben. Anschließend sendet die Sensorstation im Sekundentakt die Ststionsnummer und alle Sensordaten über die Schnittstellen Serial und Serial1 aus. Weiters werden alle Sensordaten am LCD ausgegeben.
Zusätzlich soll die Stationsnummer (0-7) über die serielle Scnittstelle einstellbar sein

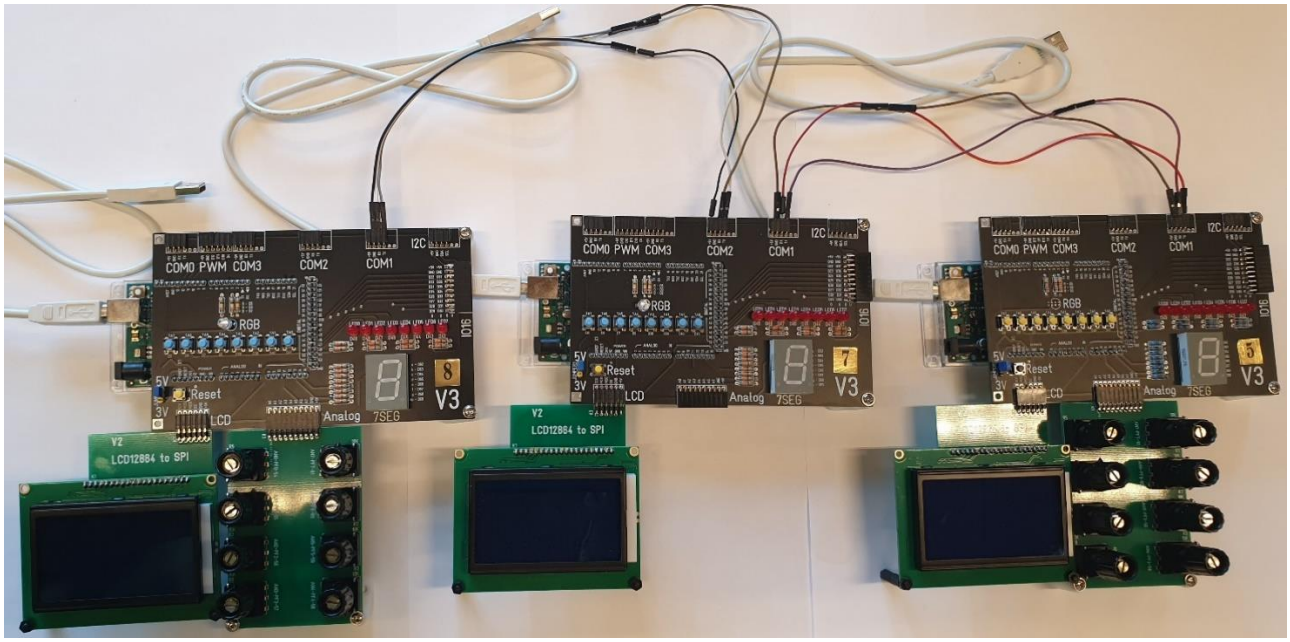
Das Datenübertragungsprotokoll (DÜP) ist seblst zu definieren.
Tipp1: Stringbasiert mit Trennzeichen und mit <CR> (\r) als Abschluss.
Tipp2: Schnittstelle einstellen und Daten auslesen kann über verschiedene Schnittstellen erfolgen.

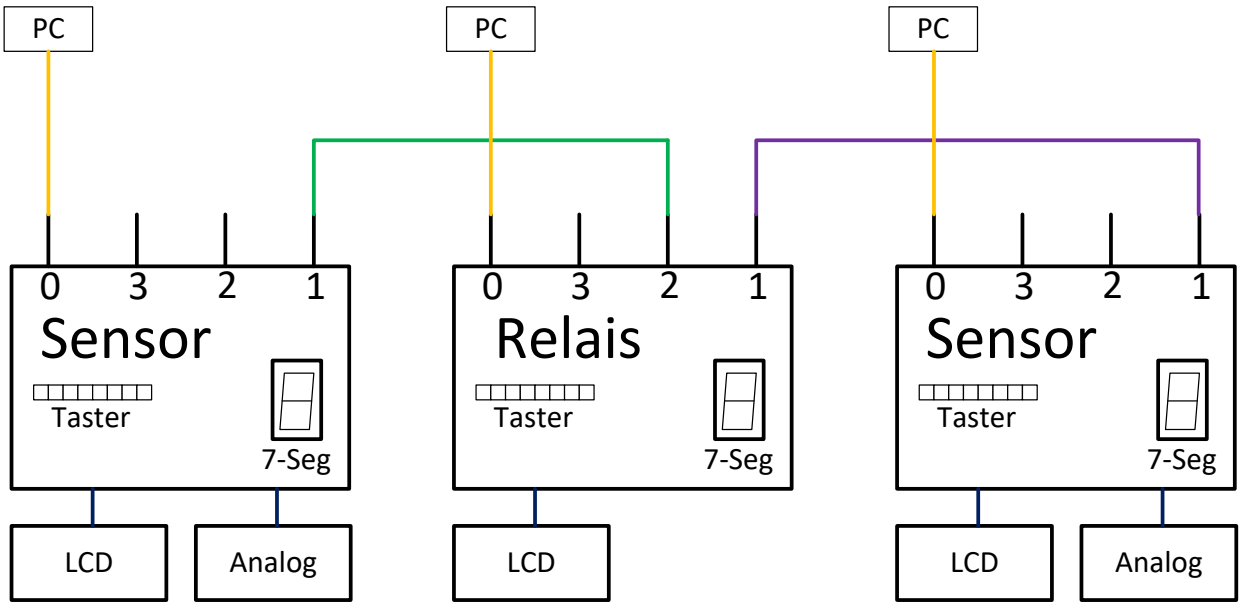
b) Die gesendeten Daten werden von der Collectorstation empfangen und gespeichert.
Die Nummer der zuletzt empfangenen Station wird auf den Leds ausgegeben.
Mit den Tasten kann eine Stationsnummer gewählt werden.Alle Sensorwerte dieser Sensorstation werden am LCD zyklisch (500ms) ausgegeben. Die gewählte Station wird mit der 7 Segmentanzeige dargestellt.

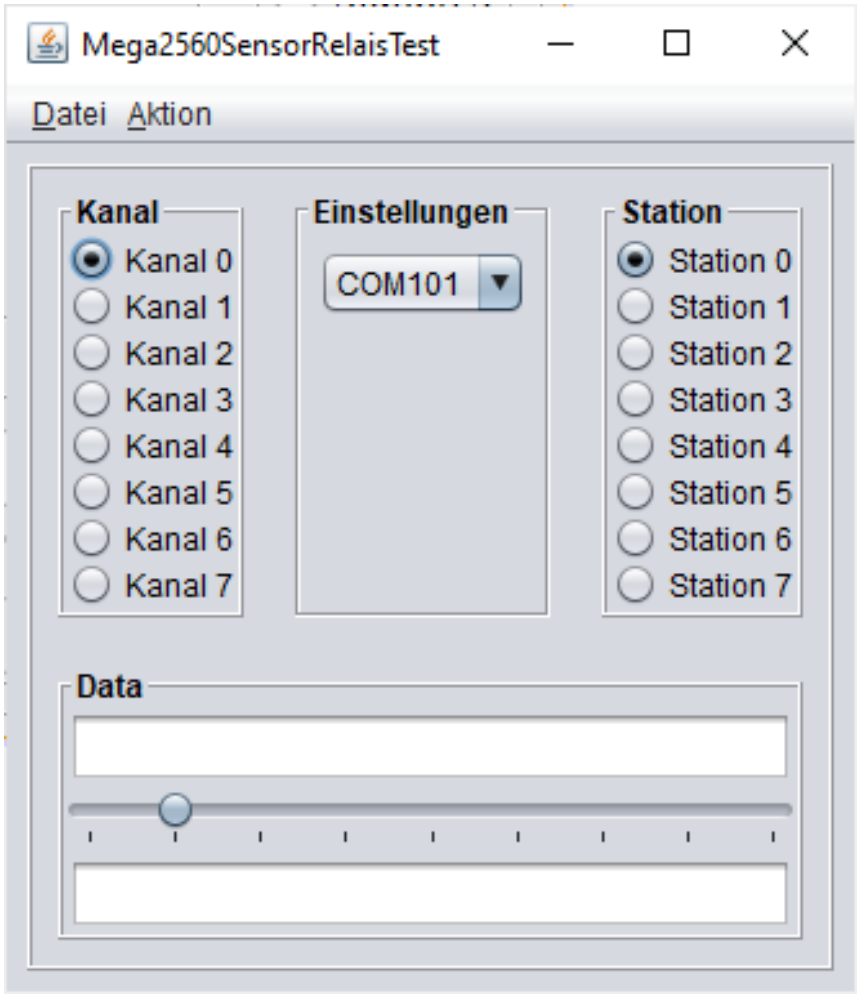
c) Uber COM0 können alle Sensordaten auf Anforderung ausgelesen werden,
Das DÜP ist zu definieren.

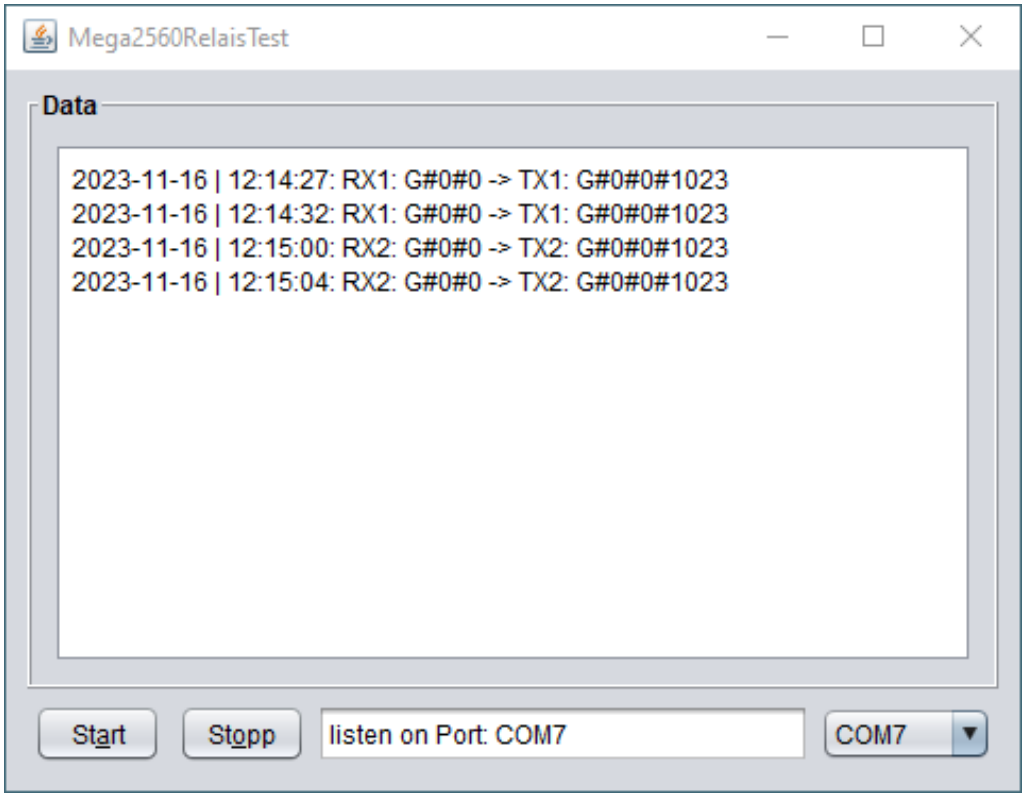
Tipp: Einzrlne Sensorwerte werden auf Anforderung gesendet.
Zyklisches senden einzelner Sensorwerte
Alle Sensorwerte werden auf Anforderung gesendet
Alle Sensorwerte werdne zyklisch gesendet

ÜBUNG 17


COM0 PWM COM3
COM2
COM1
I2C
RGB
Reset
LCD
Analog
7SEG
V3
LCD12864 to SPI


PC
PC
PC
0 3 2 1
Sensor
Taster
7-Seg
LCD
Analog
0 3 2 1
Relais
Taster
7-Seg
LCD
0 3 2 1
Sensor
Taster
7-Seg
LCD
Analog


Mega2560SensorRelaisTest
Datei Aktion
Kanal
Kanal 0
Kanal 1
Kanal 2
Kanal 3
Kanal 4
Kanal 5
Kanal 6
Kanal 7
Einstellungen
COM101
Station
Station 0
Station 1
Station 2
Station 3
Station 4
Station 5
Station 6
Station 7
Data


Mega2560RelaisTest
Data
2023-11-16 | 12:14:27: RX1: G#0#0 -> TX1: G#0#0#1023
2023-11-16 | 12:14:32: RX1: G#0#0 -> TX1: G#0#0#1023
2023-11-16 | 12:15:00: RX2: G#0#0 -> TX2: G#0#0#1023
2023-11-16 | 12:15:04: RX2: G#0#0 -> TX2: G#0#0#1023
Start
Stopp
listen on Port: COM7
COM7

Programmieren Sie folgende Aufgabenstellungen:

a) Nach dem PowerOnReset (POR) wird über die Tasten die Stationsnummer der Sensorstation gewählt und auf den 7 Segmentanzeige ausgegeben. Anschließend sendet die Sensorstation im Sekundentakt die Ststionsnummer und alle Sensordaten über die Schnittstelle Serial1 aus. Weiters werden alle Sensordaten am LCD ausgegeben.
Zusätzlich soll die Stationsnummer (0-7) über die serielle Scnittstelle einstellbar sein

Das Datenübertragungsprotokoll (DÜP) ist seblst zu definieren.
Tipp: Stringbasiert mit Trennzeichen und mit <CR> (\r) als Abschluss.

b) Die gesendeten Daten werden von der Relaisstation empfangen und gespeichert.
Die Nummer der rechten Station (COM1) wird auf den Leds ausgegeben.
Die Nummer der linken Station (COM2) wird auf der 7-Segmentanzeige ausgegeben.
Alle Sensorwerte werden am LCD zyklisch (500ms) ausgegeben.

c) Wenn über COM0 eine Leseaufforderung für einen Sensorwert einer Sensorstation empfangen wird, soll diese Anforderung über COM1 an die Relaisstation weitergegeben werden. Das empfangene Telegramm wird konvertiert und über COM0 weitergesendet.

d) Wenn über COM1 oder COM2 eine Leseanforderung für einen Sensorwert einer Sensorstation empfangen wird, soll ein Antworttelegramm generiert und über die selbe Schnittstelle zurückgesendet werden. Weiters soll die RGB-Led bei COM1 kurz grün und bei COM2 kurz BLAU aufleuchten.

e) Der Wert des mit den Tasten ausgewählten Sensors wird auf der 4 stelligen 7 Segmentanzeige dargestellt (siehe Übung 15).

ANHANG 1: ERGÄNZUNGEN

AD ÜBUNG 3

Der Zugriff auf die Hardware erfolgt über sogenannte Register, das sind beim ATMega2560 8-Bit breite Speicherstellen mit definierten Namen.
Das Arduino System nimmt dem Programmierer die Arbeit ab, die entsprechenden Port-Bits zu identifizieren und stellt die I/O-Pins als Nummern (0 - 69) zur Verfügung.
Die Funktion pinMode(uint8_t pin, uint8_t mode) setzt die entsprechenden Bits für Input oder Output und die Funktion digitakWrite(uint8_t pin, uint8_t val) setzt den Pin auf HIGH oder LOW, wenn der Pin auf Output gesetzt ist.

In unserem Fall bedeutet pinMode(LED0, OUTPUT) → pinMode(49,1)
Das Bit 0 von PORT L muss auf Output gesetzt werden.

		7	6	5	4	3	2	1	0
Port-Struktur	DDRL								1
	PORTL								1 / 0
	PINL								

DDRx … Data Direction Register vom Port x (0 … Input; 1 … Output)
PORTx … Port Register vom Port x @Output: 0 … LOW; 1 … HIGH
@Input: 0 … Pullup aus; 1 … Pullup ein
PINx … Pin Register vom Port x @Input: einlesen des Zustandes
@Output: invertieren des Zusatndes beim Schreiben

Einzelne Bits in den Registern können mit den Bit-Operatoren in C Realisiert werden.
Bitweises Oder: |
Bitweises Und: &
Bitweises XOR: ^

ODER			
A	M	Q	Q|m
0	0	0	A
1	0	1	
0	1	1	1
1	1	1	

UND			
A	M	Q	Q|m
0	0	0	0
1	0	0	
0	1	0	A
1	1	1	

XOR			
A	M	Q	Q|m
0	0	0	A
1	0	1	
0	1	1	~A
1	1	0	

Die logischen Elemente können so gesehen werden, dass ein Eingangs- und ein Steuerpin (Maske) zur Verfügung stehen. Mit der Oder-Verknüpfung kann gesteuert auf 1 gesetzt, mit der Und-Verknüpfung gesteuert auf 0 gesetzt und mit dem XOR gesteuert invertiert werden.

D.h. das Bit 0 in DDRL wird mit
DDRL |= 0x01
auf 1 gesetzt.

Für das ‚Schieben' von Bits nach links oder rechts gibt es in C sog. Schiebeoperatoren.

>> nach rechts schieben
<< nach ninks schieben

Bsp:
```
uint8_t val = 0x55;
uint8_t val2 = val << 1
```

>>		7	6	5	4	3	2	1	0
	val	0	1	0	1	0	1	0	1
	val2	0	0	1	0	1	0	1	0

Beim Schieben nach rechts werden alle Bits nach rechts verschoben und von links wird mit 0 aufgefüllt.

Verkürzte Schreibweise mit Zuweisungsoperator:

```
uint8_t val = 0x55;
val >>= 1;
```

Bedeutung: Integerdivision durch 2

Bsp:
```
uint8_t val = 0x05;
uint8_t val2 = val << 2
```

<<		7	6	5	4	3	2	1	0
	val	0	0	0	0	0	1	0	1
	val2	0	0	0	1	0	1	0	0

Beim Schieben nach rechts werden alle Bits nach rechts verschoben und von links wird mit 0 aufgefüllt.

Verkürzte Schreibweise mit Zuweisungsoperator:

```
uint8_t val = 0x05;
val <<= 2;
```

Bedeutung: Multiplikation mit 2^2

C-Ergänzung: bedingte Bewertung mit ?: Operator

statt
```
if(a==3)
    b = 5;
else
    b = 7;
```
kann
```
b = a==3 ? 5 : 7;
```
geschrieben werden!

AD ÜBUNG 10

Die srielle Schnittstelle kann beim Arduino-System einfach angewendet werden. Es existiert eine vorgefertigte Klasse (Serial für COM0).

Anwendung:

in setup():

Serial.begin(9600); //Initialisieren mit 9600,N,8,1

in loop():

Die Ausgabe erfolgt über Serial.write(), Serial.print() oder Serial.println()
Einfach mit sprintf(…) einen String ‚zusammenbasteln' und diesen String über die Schnittstelle senden.

z.B.:

```
char str[80]="";
sprintf(str, "Wert:%02d", wert);
Serial.println(str);
```

Referenz:
https://www.arduino.cc/reference/de/language/functions/communication/serial (24.04.2023)

Funktionen für das Empfangen von Zeichen über die serielle Schnittstelle:

Serial.available() … sind Zeichen verfügbar
Serial.read() … liest ein Zeichen ein

serialEvent() … Diese Funktion wird bei jedem Empfang eines Zeichens aufgefufen

AD ÜBUNG12

Beispielcode für den Empfang eines Strings über die serielle Schnittstelle

```
#define MAXSTR 80

char rx0Str[MAXSTR] = "";
bool rx0Flag = false;
```

```
void setup()
{
    Serial.begin(9600);
}
```

```
void loop()
{
    if(rx0Flag)
    {
        rx0Flag = false;
        // ToDo
    }
}
```

```
void serialEvent()
{
    static char tempStr[MAXSTR] = "";
    static uint8_t cnt = 0;

    while(Serial.available())  // solange Zeichen vorhanden sind
    {
        char ch = Serial.read();       // Zeichen einlesen
        if(ch == '\n' || ch == '\r')  // wenn String beendet
        {                              // fertig
            tempStr[cnt] = 0;          // Stringabschluss
            cnt = 0;                   // vorbereiten fuer naechsten String
            strcpy(rx0Str, tempStr);   // kopieren
            rx0Flag = true;            // Flag setzen
        }
        else
        {
            tempStr[cnt++] = ch;  // Zeichen speichern
            if(cnt >= MAXSTR)     // Sicherheitsprogrammierung
            {
                cnt = 0;
            }
        }
    }
}
```

AD ÜBUNG14

Die Funktionen

```
void initAmpelSer(void)
void updateAmpelSer(uint8_t ampelNr, uint8_t value)
```

und

```
uint8_t getAmpelSer(void)
```

sind im Modul ampelSer (ampelSer.h und ampelSer.cpp) zu finden.

Für die Ampelnummern steht ein enum mit den Werten (NONE, NORD, OST, SUED, WEST) zur Verfügung.

Der Wert der Lichter ist abgebildet wie folgt:

Bit	7	6	5	4	3	2	1	0
Licht	RTL	GEL	GRL	RTR	GER	GRR	RTF	GRF

GRF … Grün für Fußgänger
RTF … Rot für Fußgänger
GRR … Grün für Gerade/Rechts
GER … Gelb für Gerade/Rechts
RTR … Rot für Gerade/Rechts
GRL … Grün für Links
GEL … Gelb für Links
RTL … Rot für Links

z.B.: beide gelbe Lichter der Ampel Nord einschalten (und alle Anderen aussschalten)

Bit	7	6	5	4	3	2	1	0	
Licht	RTL	GEL	GRL	RTR	GER	GRR	RTF	GRF	
	0	1	0	0	1	0	0	0	→ 0x48

Aufruf von

```
updateAmpelSer(NORD, 0x48);
```

`AmpelButton:`
Die Fuktion `getAmpelSer()` liefert die Nummer der gedrückten Taste oder 0 zurück (siehe enum). Falls mehrere Taste gedrückt sind wird die niedrigste Nummer laut enum zurückgeliefert.

z.B.:

```
if(getAmpelSer() == NORD)
{
    // Button Nord ist gedrückt
}
```

ANHANG 2: ARDUINO DATEIEN

ctags_target_for_gcc_minus_e.cpp

```
# 1 "...\\ue01_LED0\\ue01_LED0.ino"


void setup()
{
  pinMode(49, 0x1);
}

void loop()
{
  digitalWrite(49, 0x1);
  delay(1000);
  digitalWrite(49, 0x0);
  delay(1000);
}
```

wiring_digital.c

```
/*
  wiring_digital.c - digital input and output functions
  Part of Arduino - http://www.arduino.cc/

  Copyright (c) 2005-2006 David A. Mellis

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General
  Public License along with this library; if not, write to the
  Free Software Foundation, Inc., 59 Temple Place, Suite 330,
  Boston, MA  02111-1307  USA

  Modified 28 September 2010 by Mark Sproul
*/

#define ARDUINO_MAIN
#include "wiring_private.h"
#include "pins_arduino.h"
```

```
void pinMode(uint8_t pin, uint8_t mode)
{
     uint8_t bit = digitalPinToBitMask(pin);
     uint8_t port = digitalPinToPort(pin);
     volatile uint8_t *reg, *out;

     if (port == NOT_A_PIN)
          return;

     // JWS: can I let the optimizer do this?
     reg = portModeRegister(port);
     out = portOutputRegister(port);

     if (mode == INPUT)
     {
          uint8_t oldSREG = SREG;
          cli();
          *reg &= ~bit;
          *out &= ~bit;
          SREG = oldSREG;
     }
     else if (mode == INPUT_PULLUP)
     {
          uint8_t oldSREG = SREG;
          cli();
          *reg &= ~bit;
          *out |= bit;
          SREG = oldSREG;
     }
     else
     {
          uint8_t oldSREG = SREG;
          cli();
          *reg |= bit;
          SREG = oldSREG;
     }
}
```

```
// Forcing this inline keeps the callers from having to push their own stuff
// on the stack. It is a good performance win and only takes 1 more byte per
// user than calling. (It will take more bytes on the 168.)
//
// But shouldn't this be moved into pinMode? Seems silly to check and do on
// each digitalread or write.
//
// Mark Sproul:
// - Removed inline. Save 170 bytes on atmega1280
// - changed to a switch statement; added 32 bytes but much easier to read
and maintain.
// - Added more #ifdefs, now compiles for atmega645
//
//static    inline    void    turnOffPWM(uint8_t    timer)    __attribute__
((always_inline));
//static inline void turnOffPWM(uint8_t timer)
static void turnOffPWM(uint8_t timer)
{
    switch (timer)
    {
        #if defined(TCCR1A) && defined(COM1A1)
        case TIMER1A:   cbi(TCCR1A, COM1A1);    break;
        #endif
        #if defined(TCCR1A) && defined(COM1B1)
        case TIMER1B:   cbi(TCCR1A, COM1B1);    break;
        #endif
        #if defined(TCCR1A) && defined(COM1C1)
        case TIMER1C:   cbi(TCCR1A, COM1C1);    break;
        #endif

        #if defined(TCCR2) && defined(COM21)
        case  TIMER2:   cbi(TCCR2, COM21);      break;
        #endif

        #if defined(TCCR0A) && defined(COM0A1)
        case  TIMER0A:  cbi(TCCR0A, COM0A1);    break;
        #endif

        #if defined(TCCR0A) && defined(COM0B1)
        case  TIMER0B:  cbi(TCCR0A, COM0B1);    break;
        #endif
        #if defined(TCCR2A) && defined(COM2A1)
        case  TIMER2A:  cbi(TCCR2A, COM2A1);    break;
        #endif
        #if defined(TCCR2A) && defined(COM2B1)
        case  TIMER2B:  cbi(TCCR2A, COM2B1);    break;
        #endif

        #if defined(TCCR3A) && defined(COM3A1)
        case  TIMER3A:  cbi(TCCR3A, COM3A1);    break;
        #endif
        #if defined(TCCR3A) && defined(COM3B1)
        case  TIMER3B:  cbi(TCCR3A, COM3B1);    break;
        #endif
        #if defined(TCCR3A) && defined(COM3C1)
        case  TIMER3C:  cbi(TCCR3A, COM3C1);    break;
        #endif
```

```
        #if defined(TCCR4A) && defined(COM4A1)
        case  TIMER4A:  cbi(TCCR4A, COM4A1);    break;
        #endif
        #if defined(TCCR4A) && defined(COM4B1)
        case  TIMER4B:  cbi(TCCR4A, COM4B1);    break;
        #endif
        #if defined(TCCR4A) && defined(COM4C1)
        case  TIMER4C:  cbi(TCCR4A, COM4C1);    break;
        #endif
        #if defined(TCCR4C) && defined(COM4D1)
        case TIMER4D: cbi(TCCR4C, COM4D1);   break;
        #endif

        #if defined(TCCR5A)
        case  TIMER5A:  cbi(TCCR5A, COM5A1);    break;
        case  TIMER5B:  cbi(TCCR5A, COM5B1);    break;
        case  TIMER5C:  cbi(TCCR5A, COM5C1);    break;
        #endif
    }
}
```

```
void digitalWrite(uint8_t pin, uint8_t val)
{
     uint8_t timer = digitalPinToTimer(pin);
     uint8_t bit = digitalPinToBitMask(pin);
     uint8_t port = digitalPinToPort(pin);
     volatile uint8_t *out;

     if (port == NOT_A_PIN)
          return;

     // If the pin that support PWM output, we need to turn it off
     // before doing a digital write.
     if (timer != NOT_ON_TIMER)
          turnOffPWM(timer);

     out = portOutputRegister(port);

     uint8_t oldSREG = SREG;
     cli();

     if (val == LOW)
     {
          *out &= ~bit;
     }
     else
     {
          *out |= bit;
     }

     SREG = oldSREG;
}

int digitalRead(uint8_t pin)
{
     uint8_t timer = digitalPinToTimer(pin);
     uint8_t bit = digitalPinToBitMask(pin);
     uint8_t port = digitalPinToPort(pin);

     if (port == NOT_A_PIN)
          return LOW;

     // If the pin that support PWM output, we need to turn it off
     // before getting a digital reading.
     if (timer != NOT_ON_TIMER)
          turnOffPWM(timer);

     if (*portInputRegister(port) & bit)
          return HIGH;
     return LOW;
}
```

pins_arduino.h

```
/*
  pins_arduino.h - Pin definition functions for Arduino
  Part of Arduino - http://www.arduino.cc/

  Copyright (c) 2007 David A. Mellis

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General
  Public License along with this library; if not, write to the
  Free Software Foundation, Inc., 59 Temple Place, Suite 330,
  Boston, MA  02111-1307  USA
*/

#ifndef Pins_Arduino_h
#define Pins_Arduino_h

#include <avr/pgmspace.h>

#define NUM_DIGITAL_PINS            70
#define NUM_ANALOG_INPUTS           16
#define analogInputToDigitalPin(p)  ((p < 16) ? (p) + 54 : -1)
#define digitalPinHasPWM(p)         (((p) >= 2 && (p) <= 13) || ((p) >= 44
                                      && (p)<= 46))

#define PIN_SPI_SS    (53)
#define PIN_SPI_MOSI  (51)
#define PIN_SPI_MISO  (50)
#define PIN_SPI_SCK   (52)

static const uint8_t SS   = PIN_SPI_SS;
static const uint8_t MOSI = PIN_SPI_MOSI;
static const uint8_t MISO = PIN_SPI_MISO;
static const uint8_t SCK  = PIN_SPI_SCK;

#define PIN_WIRE_SDA        (20)
#define PIN_WIRE_SCL        (21)

static const uint8_t SDA = PIN_WIRE_SDA;
static const uint8_t SCL = PIN_WIRE_SCL;

#define LED_BUILTIN 13

#define PIN_A0   (54)
#define PIN_A1   (55)
#define PIN_A2   (56)
#define PIN_A3   (57)
#define PIN_A4   (58)
#define PIN_A5   (59)
#define PIN_A6   (60)
```

```
#define PIN_A7   (61)
#define PIN_A8   (62)
#define PIN_A9   (63)
#define PIN_A10  (64)
#define PIN_A11  (65)
#define PIN_A12  (66)
#define PIN_A13  (67)
#define PIN_A14  (68)
#define PIN_A15  (69)

static const uint8_t A0 = PIN_A0;
static const uint8_t A1 = PIN_A1;
static const uint8_t A2 = PIN_A2;
static const uint8_t A3 = PIN_A3;
static const uint8_t A4 = PIN_A4;
static const uint8_t A5 = PIN_A5;
static const uint8_t A6 = PIN_A6;
static const uint8_t A7 = PIN_A7;
static const uint8_t A8 = PIN_A8;
static const uint8_t A9 = PIN_A9;
static const uint8_t A10 = PIN_A10;
static const uint8_t A11 = PIN_A11;
static const uint8_t A12 = PIN_A12;
static const uint8_t A13 = PIN_A13;
static const uint8_t A14 = PIN_A14;
static const uint8_t A15 = PIN_A15;

// A majority of the pins are NOT PCINTs, SO BE WARNED (i.e. you cannot use
// them as receive pins)
// Only pins available for RECEIVE (TRANSMIT can be on any pin):
// (I've deliberately left out pin mapping to the Hardware USARTs - seems
// senseless to me)
// Pins: 10, 11, 12, 13,  50, 51, 52, 53,  62, 63, 64, 65, 66, 67, 68, 69

#define digitalPinToPCICR(p)    ( (((p) >= 10) && ((p) <= 13)) || \
                                  (((p) >= 50) && ((p) <= 53)) || \
                                  (((p) >= 62) && ((p) <= 69)) ? (&PCICR) : \
                                  ((uint8_t *)0) )

#define digitalPinToPCICRbit(p) ( (((p) >= 10) && ((p) <= 13)) || \
                                  (((p) >= 50) && ((p) <= 53)) ? 0 : \
                                  ( (((p) >= 62) && ((p) <= 69)) ? 2 : \
                                    0 ) )

#define digitalPinToPCMSK(p)    ( (((p) >= 10) && ((p) <= 13)) || (((p)
                                 >= 50) && ((p) <= 53)) ? (&PCMSK0) : \
                                  ( (((p) >= 62) && ((p) <= 69)) ? (&PCMSK2)
                                    : ((uint8_t *)0) ) )

#define digitalPinToPCMSKbit(p) ( (((p) >= 10) && ((p) <= 13)) ? ((p) - 6) :
\
                                ( ((p) == 50) ? 3 : \
                                  ( ((p) == 51) ? 2 : \
                                    ( ((p) == 52) ? 1 : \
                                      ( ((p) == 53) ? 0 : \
                                        ( (((p) >= 62) && ((p) <= 69)) ? \
                                          ((p) - 62) : 0
                                ) ) ) ) ) )
```

```
#define digitalPinToInterrupt(p) ((p) == 2 ? 0 : ((p) == 3 ? 1 : \
                                  ((p) >= 18 && (p) <= 21 ? 23 - (p) : \
                                    NOT_AN_INTERRUPT)))

#ifdef ARDUINO_MAIN

const uint16_t PROGMEM port_to_mode_PGM[] = {
     NOT_A_PORT,
     (uint16_t) &DDRA,
     (uint16_t) &DDRB,
     (uint16_t) &DDRC,
     (uint16_t) &DDRD,
     (uint16_t) &DDRE,
     (uint16_t) &DDRF,
     (uint16_t) &DDRG,
     (uint16_t) &DDRH,
     NOT_A_PORT,
     (uint16_t) &DDRJ,
     (uint16_t) &DDRK,
     (uint16_t) &DDRL,
};

const uint16_t PROGMEM port_to_output_PGM[] = {
     NOT_A_PORT,
     (uint16_t) &PORTA,
     (uint16_t) &PORTB,
     (uint16_t) &PORTC,
     (uint16_t) &PORTD,
     (uint16_t) &PORTE,
     (uint16_t) &PORTF,
     (uint16_t) &PORTG,
     (uint16_t) &PORTH,
     NOT_A_PORT,
     (uint16_t) &PORTJ,
     (uint16_t) &PORTK,
     (uint16_t) &PORTL,
};

const uint16_t PROGMEM port_to_input_PGM[] = {
     NOT_A_PIN,
     (uint16_t) &PINA,
     (uint16_t) &PINB,
     (uint16_t) &PINC,
     (uint16_t) &PIND,
     (uint16_t) &PINE,
     (uint16_t) &PINF,
     (uint16_t) &PING,
     (uint16_t) &PINH,
     NOT_A_PIN,
     (uint16_t) &PINJ,
     (uint16_t) &PINK,
     (uint16_t) &PINL,
};

const uint8_t PROGMEM digital_pin_to_port_PGM[] = {
     // PORTLIST
     // -------------------------------------------
     PE   , // PE 0 ** 0 ** USART0_RX
     PE   , // PE 1 ** 1 ** USART0_TX
```

```
    PE   , // PE 4 ** 2 ** PWM2
    PE   , // PE 5 ** 3 ** PWM3
    PG   , // PG 5 ** 4 ** PWM4
    PE   , // PE 3 ** 5 ** PWM5
    PH   , // PH 3 ** 6 ** PWM6
    PH   , // PH 4 ** 7 ** PWM7
    PH   , // PH 5 ** 8 ** PWM8
    PH   , // PH 6 ** 9 ** PWM9
    PB   , // PB 4 ** 10 ** PWM10
    PB   , // PB 5 ** 11 ** PWM11
    PB   , // PB 6 ** 12 ** PWM12
    PB   , // PB 7 ** 13 ** PWM13
    PJ   , // PJ 1 ** 14 ** USART3_TX
    PJ   , // PJ 0 ** 15 ** USART3_RX
    PH   , // PH 1 ** 16 ** USART2_TX
    PH   , // PH 0 ** 17 ** USART2_RX
    PD   , // PD 3 ** 18 ** USART1_TX
    PD   , // PD 2 ** 19 ** USART1_RX
    PD   , // PD 1 ** 20 ** I2C_SDA
    PD   , // PD 0 ** 21 ** I2C_SCL
    PA   , // PA 0 ** 22 ** D22
    PA   , // PA 1 ** 23 ** D23
    PA   , // PA 2 ** 24 ** D24
    PA   , // PA 3 ** 25 ** D25
    PA   , // PA 4 ** 26 ** D26
    PA   , // PA 5 ** 27 ** D27
    PA   , // PA 6 ** 28 ** D28
    PA   , // PA 7 ** 29 ** D29
    PC   , // PC 7 ** 30 ** D30
    PC   , // PC 6 ** 31 ** D31
    PC   , // PC 5 ** 32 ** D32
    PC   , // PC 4 ** 33 ** D33
    PC   , // PC 3 ** 34 ** D34
    PC   , // PC 2 ** 35 ** D35
    PC   , // PC 1 ** 36 ** D36
    PC   , // PC 0 ** 37 ** D37
    PD   , // PD 7 ** 38 ** D38
    PG   , // PG 2 ** 39 ** D39
    PG   , // PG 1 ** 40 ** D40
    PG   , // PG 0 ** 41 ** D41
    PL   , // PL 7 ** 42 ** D42
    PL   , // PL 6 ** 43 ** D43
    PL   , // PL 5 ** 44 ** D44
    PL   , // PL 4 ** 45 ** D45
    PL   , // PL 3 ** 46 ** D46
    PL   , // PL 2 ** 47 ** D47
    PL   , // PL 1 ** 48 ** D48
    PL   , // PL 0 ** 49 ** D49
    PB   , // PB 3 ** 50 ** SPI_MISO
    PB   , // PB 2 ** 51 ** SPI_MOSI
    PB   , // PB 1 ** 52 ** SPI_SCK
    PB   , // PB 0 ** 53 ** SPI_SS
    PF   , // PF 0 ** 54 ** A0
    PF   , // PF 1 ** 55 ** A1
    PF   , // PF 2 ** 56 ** A2
    PF   , // PF 3 ** 57 ** A3
    PF   , // PF 4 ** 58 ** A4
    PF   , // PF 5 ** 59 ** A5
    PF   , // PF 6 ** 60 ** A6
```

```
    PF   , // PF 7 ** 61 ** A7
    PK   , // PK 0 ** 62 ** A8
    PK   , // PK 1 ** 63 ** A9
    PK   , // PK 2 ** 64 ** A10
    PK   , // PK 3 ** 65 ** A11
    PK   , // PK 4 ** 66 ** A12
    PK   , // PK 5 ** 67 ** A13
    PK   , // PK 6 ** 68 ** A14
    PK   , // PK 7 ** 69 ** A15
};

const uint8_t PROGMEM digital_pin_to_bit_mask_PGM[] = {
    // PIN IN PORT
    // -------------------------------------------
    _BV( 0 ) , // PE 0 ** 0 ** USART0_RX
    _BV( 1 ) , // PE 1 ** 1 ** USART0_TX
    _BV( 4 ) , // PE 4 ** 2 ** PWM2
    _BV( 5 ) , // PE 5 ** 3 ** PWM3
    _BV( 5 ) , // PG 5 ** 4 ** PWM4
    _BV( 3 ) , // PE 3 ** 5 ** PWM5
    _BV( 3 ) , // PH 3 ** 6 ** PWM6
    _BV( 4 ) , // PH 4 ** 7 ** PWM7
    _BV( 5 ) , // PH 5 ** 8 ** PWM8
    _BV( 6 ) , // PH 6 ** 9 ** PWM9
    _BV( 4 ) , // PB 4 ** 10 ** PWM10
    _BV( 5 ) , // PB 5 ** 11 ** PWM11
    _BV( 6 ) , // PB 6 ** 12 ** PWM12
    _BV( 7 ) , // PB 7 ** 13 ** PWM13
    _BV( 1 ) , // PJ 1 ** 14 ** USART3_TX
    _BV( 0 ) , // PJ 0 ** 15 ** USART3_RX
    _BV( 1 ) , // PH 1 ** 16 ** USART2_TX
    _BV( 0 ) , // PH 0 ** 17 ** USART2_RX
    _BV( 3 ) , // PD 3 ** 18 ** USART1_TX
    _BV( 2 ) , // PD 2 ** 19 ** USART1_RX
    _BV( 1 ) , // PD 1 ** 20 ** I2C_SDA
    _BV( 0 ) , // PD 0 ** 21 ** I2C_SCL
    _BV( 0 ) , // PA 0 ** 22 ** D22
    _BV( 1 ) , // PA 1 ** 23 ** D23
    _BV( 2 ) , // PA 2 ** 24 ** D24
    _BV( 3 ) , // PA 3 ** 25 ** D25
    _BV( 4 ) , // PA 4 ** 26 ** D26
    _BV( 5 ) , // PA 5 ** 27 ** D27
    _BV( 6 ) , // PA 6 ** 28 ** D28
    _BV( 7 ) , // PA 7 ** 29 ** D29
    _BV( 7 ) , // PC 7 ** 30 ** D30
    _BV( 6 ) , // PC 6 ** 31 ** D31
    _BV( 5 ) , // PC 5 ** 32 ** D32
    _BV( 4 ) , // PC 4 ** 33 ** D33
    _BV( 3 ) , // PC 3 ** 34 ** D34
    _BV( 2 ) , // PC 2 ** 35 ** D35
    _BV( 1 ) , // PC 1 ** 36 ** D36
    _BV( 0 ) , // PC 0 ** 37 ** D37
    _BV( 7 ) , // PD 7 ** 38 ** D38
    _BV( 2 ) , // PG 2 ** 39 ** D39
    _BV( 1 ) , // PG 1 ** 40 ** D40
    _BV( 0 ) , // PG 0 ** 41 ** D41
    _BV( 7 ) , // PL 7 ** 42 ** D42
    _BV( 6 ) , // PL 6 ** 43 ** D43
    _BV( 5 ) , // PL 5 ** 44 ** D44
```

```
    _BV( 4 ) , // PL 4 ** 45 ** D45
    _BV( 3 ) , // PL 3 ** 46 ** D46
    _BV( 2 ) , // PL 2 ** 47 ** D47
    _BV( 1 ) , // PL 1 ** 48 ** D48
    _BV( 0 ) , // PL 0 ** 49 ** D49
    _BV( 3 ) , // PB 3 ** 50 ** SPI_MISO
    _BV( 2 ) , // PB 2 ** 51 ** SPI_MOSI
    _BV( 1 ) , // PB 1 ** 52 ** SPI_SCK
    _BV( 0 ) , // PB 0 ** 53 ** SPI_SS
    _BV( 0 ) , // PF 0 ** 54 ** A0
    _BV( 1 ) , // PF 1 ** 55 ** A1
    _BV( 2 ) , // PF 2 ** 56 ** A2
    _BV( 3 ) , // PF 3 ** 57 ** A3
    _BV( 4 ) , // PF 4 ** 58 ** A4
    _BV( 5 ) , // PF 5 ** 59 ** A5
    _BV( 6 ) , // PF 6 ** 60 ** A6
    _BV( 7 ) , // PF 7 ** 61 ** A7
    _BV( 0 ) , // PK 0 ** 62 ** A8
    _BV( 1 ) , // PK 1 ** 63 ** A9
    _BV( 2 ) , // PK 2 ** 64 ** A10
    _BV( 3 ) , // PK 3 ** 65 ** A11
    _BV( 4 ) , // PK 4 ** 66 ** A12
    _BV( 5 ) , // PK 5 ** 67 ** A13
    _BV( 6 ) , // PK 6 ** 68 ** A14
    _BV( 7 ) , // PK 7 ** 69 ** A15
};

const uint8_t PROGMEM digital_pin_to_timer_PGM[] = {
    // TIMERS
    // ------------------------------------------
    NOT_ON_TIMER  , // PE 0 ** 0 ** USART0_RX
    NOT_ON_TIMER  , // PE 1 ** 1 ** USART0_TX
    TIMER3B  , // PE 4 ** 2 ** PWM2
    TIMER3C  , // PE 5 ** 3 ** PWM3
    TIMER0B  , // PG 5 ** 4 ** PWM4
    TIMER3A  , // PE 3 ** 5 ** PWM5
    TIMER4A  , // PH 3 ** 6 ** PWM6
    TIMER4B  , // PH 4 ** 7 ** PWM7
    TIMER4C  , // PH 5 ** 8 ** PWM8
    TIMER2B  , // PH 6 ** 9 ** PWM9
    TIMER2A  , // PB 4 ** 10 ** PWM10
    TIMER1A  , // PB 5 ** 11 ** PWM11
    TIMER1B  , // PB 6 ** 12 ** PWM12
    TIMER0A  , // PB 7 ** 13 ** PWM13
    NOT_ON_TIMER  , // PJ 1 ** 14 ** USART3_TX
    NOT_ON_TIMER  , // PJ 0 ** 15 ** USART3_RX
    NOT_ON_TIMER  , // PH 1 ** 16 ** USART2_TX
    NOT_ON_TIMER  , // PH 0 ** 17 ** USART2_RX
    NOT_ON_TIMER  , // PD 3 ** 18 ** USART1_TX
    NOT_ON_TIMER  , // PD 2 ** 19 ** USART1_RX
    NOT_ON_TIMER  , // PD 1 ** 20 ** I2C_SDA
    NOT_ON_TIMER  , // PD 0 ** 21 ** I2C_SCL
    NOT_ON_TIMER  , // PA 0 ** 22 ** D22
    NOT_ON_TIMER  , // PA 1 ** 23 ** D23
    NOT_ON_TIMER  , // PA 2 ** 24 ** D24
    NOT_ON_TIMER  , // PA 3 ** 25 ** D25
    NOT_ON_TIMER  , // PA 4 ** 26 ** D26
    NOT_ON_TIMER  , // PA 5 ** 27 ** D27
    NOT_ON_TIMER  , // PA 6 ** 28 ** D28
```

```
    NOT_ON_TIMER  , // PA 7 ** 29 ** D29
    NOT_ON_TIMER  , // PC 7 ** 30 ** D30
    NOT_ON_TIMER  , // PC 6 ** 31 ** D31
    NOT_ON_TIMER  , // PC 5 ** 32 ** D32
    NOT_ON_TIMER  , // PC 4 ** 33 ** D33
    NOT_ON_TIMER  , // PC 3 ** 34 ** D34
    NOT_ON_TIMER  , // PC 2 ** 35 ** D35
    NOT_ON_TIMER  , // PC 1 ** 36 ** D36
    NOT_ON_TIMER  , // PC 0 ** 37 ** D37
    NOT_ON_TIMER  , // PD 7 ** 38 ** D38
    NOT_ON_TIMER  , // PG 2 ** 39 ** D39
    NOT_ON_TIMER  , // PG 1 ** 40 ** D40
    NOT_ON_TIMER  , // PG 0 ** 41 ** D41
    NOT_ON_TIMER  , // PL 7 ** 42 ** D42
    NOT_ON_TIMER  , // PL 6 ** 43 ** D43
    TIMER5C   , // PL 5 ** 44 ** D44
    TIMER5B   , // PL 4 ** 45 ** D45
    TIMER5A   , // PL 3 ** 46 ** D46
    NOT_ON_TIMER  , // PL 2 ** 47 ** D47
    NOT_ON_TIMER  , // PL 1 ** 48 ** D48
    NOT_ON_TIMER  , // PL 0 ** 49 ** D49
    NOT_ON_TIMER  , // PB 3 ** 50 ** SPI_MISO
    NOT_ON_TIMER  , // PB 2 ** 51 ** SPI_MOSI
    NOT_ON_TIMER  , // PB 1 ** 52 ** SPI_SCK
    NOT_ON_TIMER  , // PB 0 ** 53 ** SPI_SS
    NOT_ON_TIMER  , // PF 0 ** 54 ** A0
    NOT_ON_TIMER  , // PF 1 ** 55 ** A1
    NOT_ON_TIMER  , // PF 2 ** 56 ** A2
    NOT_ON_TIMER  , // PF 3 ** 57 ** A3
    NOT_ON_TIMER  , // PF 4 ** 58 ** A4
    NOT_ON_TIMER  , // PF 5 ** 59 ** A5
    NOT_ON_TIMER  , // PF 6 ** 60 ** A6
    NOT_ON_TIMER  , // PF 7 ** 61 ** A7
    NOT_ON_TIMER  , // PK 0 ** 62 ** A8
    NOT_ON_TIMER  , // PK 1 ** 63 ** A9
    NOT_ON_TIMER  , // PK 2 ** 64 ** A10
    NOT_ON_TIMER  , // PK 3 ** 65 ** A11
    NOT_ON_TIMER  , // PK 4 ** 66 ** A12
    NOT_ON_TIMER  , // PK 5 ** 67 ** A13
    NOT_ON_TIMER  , // PK 6 ** 68 ** A14
    NOT_ON_TIMER  , // PK 7 ** 69 ** A15
};

#endif

// These serial port names are intended to allow libraries and architecture-
// neutral sketches to automatically default to the correct port name for a
// particular type of use. For example, a GPS module would normally connect
// to SERIAL_PORT_HARDWARE_OPEN, the first hardware serial port whose RX/TX
// pins are not dedicated to another use.
//
// SERIAL_PORT_MONITOR        Port which normally prints to the Arduino
//                            Serial Monitor
//
// SERIAL_PORT_USBVIRTUAL     Port which is USB virtual serial
//
// SERIAL_PORT_LINUXBRIDGE    Port which connects to a Linux system via
//                            Bridge library
//
```

```
// SERIAL_PORT_HARDWARE       Hardware serial port, physical RX & TX pins.
//
// SERIAL_PORT_HARDWARE_OPEN  Hardware serial ports which are open for use.
//                            Their RX & TX pins are NOT connected to
//                            anything by default.
#define SERIAL_PORT_MONITOR         Serial
#define SERIAL_PORT_HARDWARE        Serial
#define SERIAL_PORT_HARDWARE1       Serial1
#define SERIAL_PORT_HARDWARE2       Serial2
#define SERIAL_PORT_HARDWARE3       Serial3
#define SERIAL_PORT_HARDWARE_OPEN   Serial1
#define SERIAL_PORT_HARDWARE_OPEN1  Serial2
#define SERIAL_PORT_HARDWARE_OPEN2  Serial3

#endif
```

Arduino.h

```
/*
  Arduino.h - Main include file for the Arduino SDK
  Copyright (c) 2005-2013 Arduino Team.  All right reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

#ifndef Arduino_h
#define Arduino_h

#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>

#include <avr/pgmspace.h>
#include <avr/io.h>
#include <avr/interrupt.h>

#include "binary.h"

#ifdef __cplusplus
extern "C"{
#endif

void yield(void);

#define HIGH 0x1
#define LOW  0x0

#define INPUT 0x0
#define OUTPUT 0x1
#define INPUT_PULLUP 0x2

#define PI 3.14159265358979323846264338327950
#define HALF_PI 1.57079632679489661923132169163975
#define TWO_PI 6.28318530717958647692528676655900
#define DEG_TO_RAD 0.017453292519943295769236907684886
#define RAD_TO_DEG 57.295779513082320876798154814105
#define EULER 2.71828182845904523536028747135266

#define SERIAL  0x0
#define DISPLAY 0x1

#define LSBFIRST 0
#define MSBFIRST 1
```

```
#define CHANGE 1
#define FALLING 2
#define RISING 3

#if     defined(__AVR_ATtiny24__)     ||     defined(__AVR_ATtiny44__)     ||
defined(__AVR_ATtiny84__)
  #define DEFAULT 0
  #define EXTERNAL 1
  #define INTERNAL1V1 2
  #define INTERNAL INTERNAL1V1
#elif    defined(__AVR_ATtiny25__)     ||     defined(__AVR_ATtiny45__)     ||
defined(__AVR_ATtiny85__)
  #define DEFAULT 0
  #define EXTERNAL 4
  #define INTERNAL1V1 8
  #define INTERNAL INTERNAL1V1
  #define INTERNAL2V56 9
  #define INTERNAL2V56_EXTCAP 13
#else
#if    defined(__AVR_ATmega1280__)    ||    defined(__AVR_ATmega2560__)    ||
defined(__AVR_ATmega1284__)      ||      defined(__AVR_ATmega1284P__)      ||
defined(__AVR_ATmega644__)       ||       defined(__AVR_ATmega644A__)       ||
defined(__AVR_ATmega644P__) || defined(__AVR_ATmega644PA__)
#define INTERNAL1V1 2
#define INTERNAL2V56 3
#else
#define INTERNAL 3
#endif
#define DEFAULT 1
#define EXTERNAL 0
#endif

// undefine stdlib's abs if encountered
#ifdef abs
#undef abs
#endif

#define min(a,b) ((a)<(b)?(a):(b))
#define max(a,b) ((a)>(b)?(a):(b))
#define abs(x) ((x)>0?(x):-(x))
#define                                                  constrain(amt,low,high)
((amt)<(low)?(low):((amt)>(high)?(high):(amt)))
#define round(x)     ((x)>=0?(long)((x)+0.5):(long)((x)-0.5))
#define radians(deg) ((deg)*DEG_TO_RAD)
#define degrees(rad) ((rad)*RAD_TO_DEG)
#define sq(x) ((x)*(x))

#define interrupts() sei()
#define noInterrupts() cli()

#define clockCyclesPerMicrosecond() ( F_CPU / 1000000L )
#define clockCyclesToMicroseconds(a) ( (a) / clockCyclesPerMicrosecond() )
#define microsecondsToClockCycles(a) ( (a) * clockCyclesPerMicrosecond() )

#define lowByte(w) ((uint8_t) ((w) & 0xff))
#define highByte(w) ((uint8_t) ((w) >> 8))

#define bitRead(value, bit) (((value) >> (bit)) & 0x01)
```

```
#define bitSet(value, bit) ((value) |= (1UL << (bit)))
#define bitClear(value, bit) ((value) &= ~(1UL << (bit)))
#define bitToggle(value, bit) ((value) ^= (1UL << (bit)))
#define bitWrite(value, bit, bitvalue) ((bitvalue) ? bitSet(value, bit) :
bitClear(value, bit))

// avr-libc defines _NOP() since 1.6.2
#ifndef _NOP
#define _NOP() do { __asm__ volatile ("nop"); } while (0)
#endif

typedef unsigned int word;

#define bit(b) (1UL << (b))

typedef bool boolean;
typedef uint8_t byte;

void init(void);
void initVariant(void);

int atexit(void (*func)()) __attribute__((weak));

void pinMode(uint8_t pin, uint8_t mode);
void digitalWrite(uint8_t pin, uint8_t val);
int digitalRead(uint8_t pin);
int analogRead(uint8_t pin);
void analogReference(uint8_t mode);
void analogWrite(uint8_t pin, int val);

unsigned long millis(void);
unsigned long micros(void);
void delay(unsigned long ms);
void delayMicroseconds(unsigned int us);
unsigned long pulseIn(uint8_t pin, uint8_t state, unsigned long timeout);
unsigned long pulseInLong(uint8_t pin, uint8_t state, unsigned long timeout);

void shiftOut(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t
val);
uint8_t shiftIn(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder);

void attachInterrupt(uint8_t interruptNum, void (*userFunc)(void), int mode);
void detachInterrupt(uint8_t interruptNum);

void setup(void);
void loop(void);

// Get the bit location within the hardware port of the given virtual pin.
// This comes from the pins_*.c file for the active board configuration.

#define analogInPinToBit(P) (P)

// On the ATmega1280, the addresses of some of the port registers are
// greater than 255, so we can't store them in uint8_t's.
extern const uint16_t PROGMEM port_to_mode_PGM[];
extern const uint16_t PROGMEM port_to_input_PGM[];
extern const uint16_t PROGMEM port_to_output_PGM[];

extern const uint8_t PROGMEM digital_pin_to_port_PGM[];
```

```
// extern const uint8_t PROGMEM digital_pin_to_bit_PGM[];
extern const uint8_t PROGMEM digital_pin_to_bit_mask_PGM[];
extern const uint8_t PROGMEM digital_pin_to_timer_PGM[];

// Get the bit location within the hardware port of the given virtual pin.
// This comes from the pins_*.c file for the active board configuration.
//
// These perform slightly better as macros compared to inline functions
//
#define digitalPinToPort(P) ( pgm_read_byte( digital_pin_to_port_PGM + (P) )
)
#define digitalPinToBitMask(P) ( pgm_read_byte( digital_pin_to_bit_mask_PGM +
(P) ) )
#define digitalPinToTimer(P) ( pgm_read_byte( digital_pin_to_timer_PGM + (P)
) )
#define analogInPinToBit(P) (P)
#define  portOutputRegister(P)  (  (volatile  uint8_t  *)(  pgm_read_word(
port_to_output_PGM + (P))) )
#define  portInputRegister(P)  (  (volatile  uint8_t  *)(  pgm_read_word(
port_to_input_PGM + (P))) )
#define  portModeRegister(P)  (  (volatile  uint8_t  *)(  pgm_read_word(
port_to_mode_PGM + (P))) )

#define NOT_A_PIN 0
#define NOT_A_PORT 0

#define NOT_AN_INTERRUPT -1

#ifdef ARDUINO_MAIN
#define PA 1
#define PB 2
#define PC 3
#define PD 4
#define PE 5
#define PF 6
#define PG 7
#define PH 8
#define PJ 10
#define PK 11
#define PL 12
#endif

#define NOT_ON_TIMER 0
#define TIMER0A 1
#define TIMER0B 2
#define TIMER1A 3
#define TIMER1B 4
#define TIMER1C 5
#define TIMER2  6
#define TIMER2A 7
#define TIMER2B 8

#define TIMER3A 9
#define TIMER3B 10
#define TIMER3C 11
#define TIMER4A 12
#define TIMER4B 13
#define TIMER4C 14
#define TIMER4D 15
```

```
#define TIMER5A 16
#define TIMER5B 17
#define TIMER5C 18

#ifdef __cplusplus
} // extern "C"
#endif

#ifdef __cplusplus
#include "WCharacter.h"
#include "WString.h"
#include "HardwareSerial.h"
#include "USBAPI.h"
#if defined(HAVE_HWSERIAL0) && defined(HAVE_CDCSERIAL)
#error "Targets with both UART0 and CDC serial not supported"
#endif

uint16_t makeWord(uint16_t w);
uint16_t makeWord(byte h, byte l);

#define word(...) makeWord(__VA_ARGS__)

unsigned long pulseIn(uint8_t pin, uint8_t state, unsigned long timeout =
1000000L);
unsigned long pulseInLong(uint8_t pin, uint8_t state, unsigned long timeout =
1000000L);

void tone(uint8_t _pin, unsigned int frequency, unsigned long duration = 0);
void noTone(uint8_t _pin);

// WMath prototypes
long random(long);
long random(long, long);
void randomSeed(unsigned long);
long map(long, long, long, long, long);

#endif

#include "pins_arduino.h"

#endif
```

Common.h

```
COMMON.H

#pragma once
#include <stdint.h>
#include <stdbool.h>

#ifdef __cplusplus
extern "C"{
#endif

void yield(void);

typedef enum {
  LOW     = 0,
  HIGH    = 1,
  CHANGE  = 2,
  FALLING = 3,
  RISING  = 4,
} PinStatus;

typedef enum {
  INPUT             = 0x0,
  OUTPUT            = 0x1,
  INPUT_PULLUP      = 0x2,
  INPUT_PULLDOWN    = 0x3,
  OUTPUT_OPENDRAIN = 0x4,
} PinMode;

typedef enum {
  LSBFIRST = 0,
  MSBFIRST = 1,
} BitOrder;

#define PI           3.1415926535897932384626433832795
#define HALF_PI      1.5707963267948966192313216916398
#define TWO_PI       6.2831853071795864769252867665590
#define DEG_TO_RAD   0.017453292519943295769236907684886
#define RAD_TO_DEG   57.295779513082320876798154814105
#define EULER        2.7182818284590452353602874713527

#define SERIAL       0x0
#define DISPLAY      0x1

#ifndef constrain
#define constrain(amt,low,high) ((amt)<(low)?(low):((amt)>(high)?(high):(amt)))
#endif

#ifndef radians
#define radians(deg) ((deg)*DEG_TO_RAD)
#endif

#ifndef degrees
#define degrees(rad) ((rad)*RAD_TO_DEG)
#endif

#ifndef sq
#define sq(x) ((x)*(x))
#endif

typedef void (*voidFuncPtr)(void);
typedef void (*voidFuncPtrParam)(void*);
```

```
// interrupts() / noInterrupts() must be defined by the core

#define lowByte(w) ((uint8_t) ((w) & 0xff))
#define highByte(w) ((uint8_t) ((w) >> 8))

#define bitRead(value, bit) (((value) >> (bit)) & 0x01)
#define bitSet(value, bit) ((value) |= (1UL << (bit)))
#define bitClear(value, bit) ((value) &= ~(1UL << (bit)))
#define bitToggle(value, bit) ((value) ^= (1UL << (bit)))
#define bitWrite(value, bit, bitvalue) ((bitvalue) ? bitSet((value), (bit)) :
bitClear((value), (bit)))

#ifndef bit
#define bit(b) (1UL << (b))
#endif

/* TODO: request for removal */
typedef bool      boolean;
typedef uint8_t   byte;
typedef uint16_t  word;

void init(void);
void initVariant(void);

#ifndef HOST
int atexit(void (*func)()) __attribute__((weak));
#endif
int main() __attribute__((weak));

#ifdef EXTENDED_PIN_MODE
// Platforms who want to declare more than 256 pins need to define
EXTENDED_PIN_MODE globally
typedef uint32_t pin_size_t;
#else
typedef uint8_t pin_size_t;
#endif

void pinMode(pin_size_t pinNumber, PinMode pinMode);
void digitalWrite(pin_size_t pinNumber, PinStatus status);
PinStatus digitalRead(pin_size_t pinNumber);
int analogRead(pin_size_t pinNumber);
void analogReference(uint8_t mode);
void analogWrite(pin_size_t pinNumber, int value);

unsigned long millis(void);
unsigned long micros(void);
void delay(unsigned long);
void delayMicroseconds(unsigned int us);
unsigned long pulseIn(pin_size_t pin, uint8_t state, unsigned long timeout);
unsigned long pulseInLong(pin_size_t pin, uint8_t state, unsigned long timeout);

void shiftOut(pin_size_t dataPin, pin_size_t clockPin, BitOrder bitOrder,
uint8_t val);
uint8_t shiftIn(pin_size_t dataPin, pin_size_t clockPin, BitOrder bitOrder);

void attachInterrupt(pin_size_t interruptNumber, voidFuncPtr callback, PinStatus
mode);
void attachInterruptParam(pin_size_t interruptNumber, voidFuncPtrParam callback,
PinStatus mode, void* param);
void detachInterrupt(pin_size_t interruptNumber);
```

```
void setup(void);
void loop(void);

#ifdef __cplusplus
} // extern "C"
#endif

#ifdef __cplusplus
  template<class T, class L>
  auto min(const T& a, const L& b) -> decltype((b < a) ? b : a)
  {
    return (b < a) ? b : a;
  }

  template<class T, class L>
  auto max(const T& a, const L& b) -> decltype((b < a) ? b : a)
  {
    return (a < b) ? b : a;
  }
#else
#ifndef min
#define min(a,b) \
   ({ __typeof__ (a) _a = (a); \
       __typeof__ (b) _b = (b); \
     _a < _b ? _a : _b; })
#endif
#ifndef max
#define max(a,b) \
   ({ __typeof__ (a) _a = (a); \
       __typeof__ (b) _b = (b); \
     _a > _b ? _a : _b; })
#endif
#endif

#ifdef __cplusplus

/* C++ prototypes */
uint16_t makeWord(uint16_t w);
uint16_t makeWord(byte h, byte l);

#define word(...) makeWord(__VA_ARGS__)

unsigned long pulseIn(uint8_t pin, uint8_t state, unsigned long timeout =
1000000L);
unsigned long pulseInLong(uint8_t pin, uint8_t state, unsigned long timeout =
1000000L);

void tone(uint8_t _pin, unsigned int frequency, unsigned long duration = 0);
void noTone(uint8_t _pin);

// WMath prototypes
long random(long);
long random(long, long);
void randomSeed(unsigned long);
long map(long, long, long, long, long);

#endif // __cplusplus
```

ANHANG 3: PINBELEGUNGEN

Name	Gehäuse	Pin									
		1	2	3	4	5	6	7	8	9	10
COM0	K1X06	+5V	GND	RXD0 (PE0) [0]	TXD0 (PE1) [1]	---	---				
COM1	K1X06	+5V	GND	RXD1 (PD2) [19]	TXD1 (PD3) [18]	---	---				
COM2	K1X06	+5V	GND	RXD2 (PH0) [17]	TXD2 (PH1) [16]	---	---				
COM3	K1X06	+5V	GND	RXD3 (PJ0) [15]	TXD3 (PJ1) [14]	---	---				
PWM	K1X06	+5V	GND	PWM0 (PB6) [12]	PWM1 (PB5) [11]	PWM2 (PB4) [10]	PWM3 (PG5) [4]				
IO16	K2X10 (O)	+5V	GND	PA0 [22]	PA1 [23]	PA2 [24]	PA3 [25]	PA4 [26]	PA5 [27]	PA6 [28]	PA7 [29]
	K2X10 (U)	+5V	GND	PC0 [37]	PC1 [36]	PC2 [35]	PC3 [34]	PC4 [33]	PC5 [32]	PC6 [31]	PC7 [30]
Analog	K1X10	+5V	GND	AN7 (PF7) [61]	AN6 (PF6) [60]	AN5 (PF5) [59]	AN4 (PF4) [58]	AN3 (PF3) [57]	AN2 (PF2) [56]	AN1 (PF1) [55]	AN0 (PF0) [54]
LCD	K1X06	+3,3V	GND	---	---	MOSI (PB2) [21]	SCK (PB1) [20]				
RGB Led	On-board	RGB/R (PE3) [5]	RGB/G (PE5) [3]	RGB/B (PE4) [2]							
KEYs	On-board	KEY0 (PG0) [41]	KEY1 (PG1) [40]	KEY2 (PG2) [39]	KEY3 (PD7) [38]	KEY4 (PH3) [6]	KEY5 (PH4) [7]	KEY6 (PH5) [8]	KEY7 (PH6) [9]		
LEDs	On-board	LED0 (PL0) [49]	LED1 (PL1) [48]	LED2 (PL2) [47]	LED3 (PL3) [46]	LED4 (PL4) [45]	LED5 (PL5) [44]	LED6 (PL6) [43]	LED7 (PL7) [42]		
7SEG	On-board	SEG7a (PK0)[62]	SEG7b (PK1) [63]	SEG7c (PK2) [64]	SEG7d (PK3) [65]	SEG7e (PK4) [66]	SEG7f (PK5) [67]	SEG7g (PK6) [68]	SEG7dp (PK7) [69]		

(...) PortBit
[...] ArduinoNr

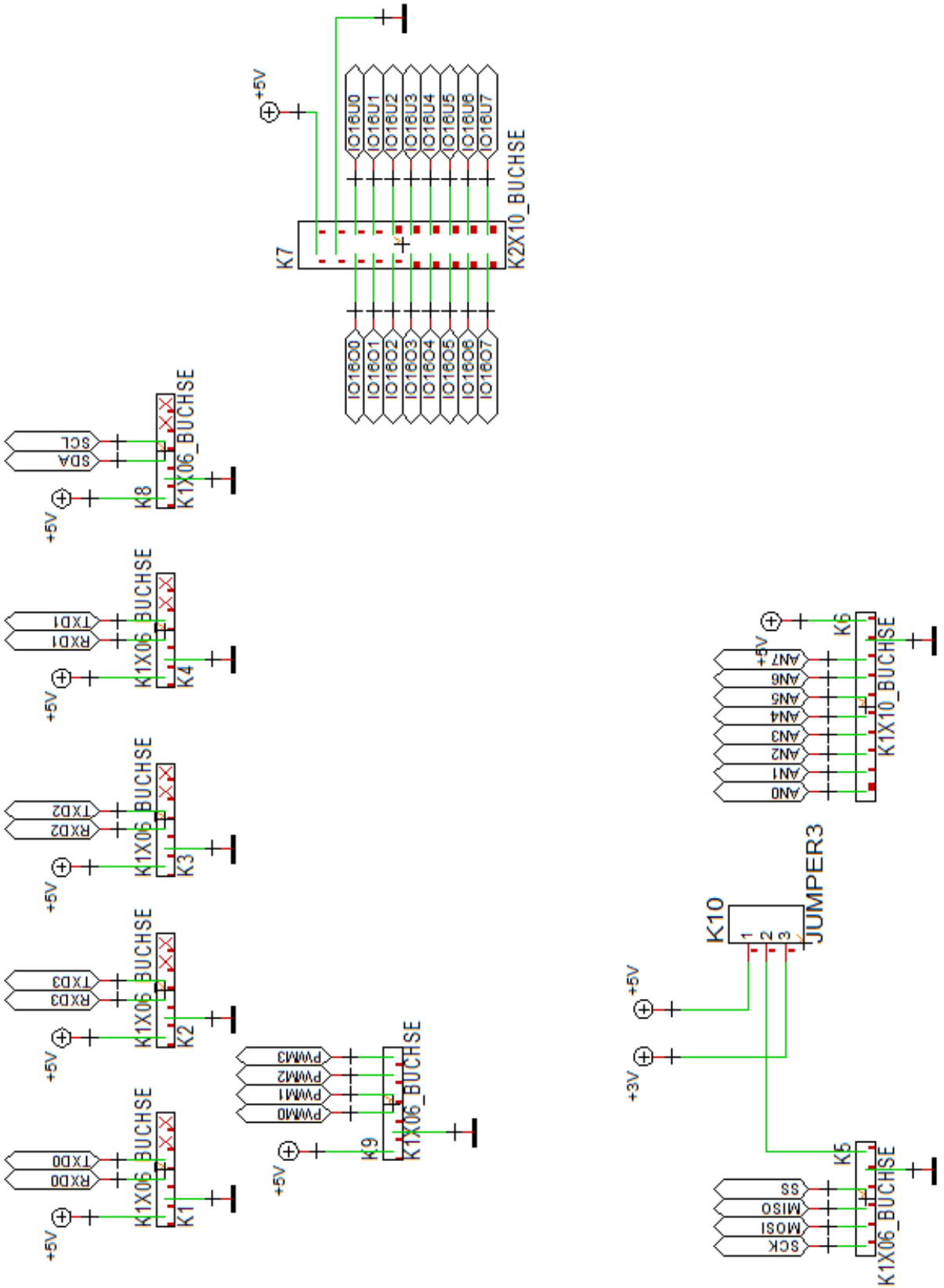
Nr	Pin
0	PE0
1	PE1
2	PE4
3	PE5
4	PG5
5	PE4
6	PH3
7	PH4
8	PH5
9	PH6
10	PB4
11	PB5
12	PB6
13	PB7
14	PJ1
15	PJ0
16	PH1
17	PH1
18	PD3
19	PD2
20	PD1
21	PD0
22	PA0
23	PA1
24	PA2
25	PA3
26	PA4
27	PA5
28	PA6
29	PA7
30	PC7
31	PC6
32	PC5
33	PC4
34	PC3

Nr	Pin	
35	PC2	
36	PC1	
37	PC0	
38	PD7	
39	PG2	
40	PG1	
41	PG0	
42	PL7	
43	PL6	
44	PL5	
45	PL4	
46	PL3	
47	PL2	
48	PL1	
49	PL0	
50	PB3	
51	PB2	
52	PB1	
53	PB0	
54	PF0	A0
55	PF1	A1
56	PF2	A2
57	PF3	A3
58	PF4	A4
59	PF5	A5
60	PF6	A6
61	PF7	A7
62	PK0	A8
63	PK1	A9
64	PK2	A10
65	PK3	A11
66	PK4	A12
67	PK7	A13
68	PK6	A14
69	PK7	A15

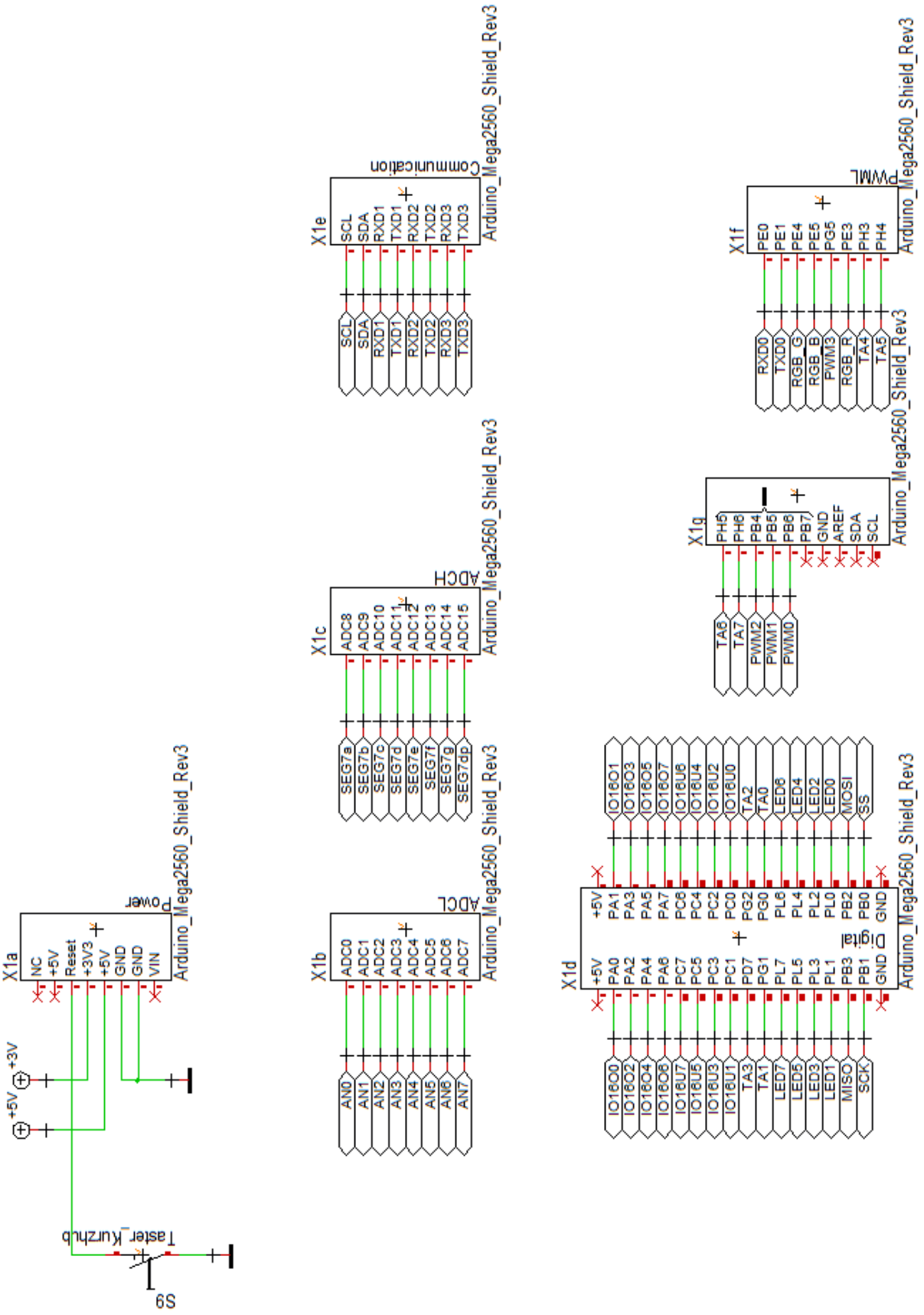
PORT	7	6	5	4	3	2	1	0
A	---	---	---	---	---	---	---	---
	IO16O7	IO16O6	IO16O5	IO16O4	IO16O3	IO16O2	IO16O1	IO16O0
B	OC0A/OC1C PCINT7	OC1B PCINT6	OC1A PCINT6	OC2A PCINT4	MISO PCINT3	MOSI PCINT2	SCK PCINT1	~SS PCINT0
	BUILD_IN	PWM0	PWM1	PWM2	(MISO)	MOSI	SCK	(SS)
C	---	---	---	---	---	---	---	---
	IO16U7	IO16U6	IO16U5	IO16U4	IO16U3	IO16U2	IO16U1	IO16U0
D	T0				INT5	INT4	INT3	INT2
	KEY3	xxx	xxx	xxx	TXD1	RXD1	SDA	SCL
E			OC3C INT1	OC3B INT0	OC3A		---	PCINT8
	xxx	xxx	RGB/B	RGB/G	RGB/R	xxx	TXD0	RXD0
F	TDI	TDO	TMS	TCK	---	---	---	---
	A7	A6	A5	A4	A3	A2	A1	A0
G			OC0B			---	---	---
	XXX	XXX	PWM3	xxx	Xxx	KEY2	KEY1	KEY0
H		OC2B	OC4C	OC4B	OC4A		---	---
	xxx	KEY7	KEY6	KEY5	KEY4	Xxx	TXD2	RXD2
J							PCINT10	PCINT9
	xxx	Xxx	xxx	xxx	Xxx	Xxx	TXD3	RXD3
K	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16
	SEG7dp	SEG7g	SEG7f	SEG7e	SEG7d	SEG7c	SEG7b	SEG7a
L	---	---	OC5C	OC5B	OC5A	T5	ICP5	ICP0
	LED7	LED6	LED5	LED4	LED3	LED2	LED1	LED0

ANHANG 4: SCHALTPLAN

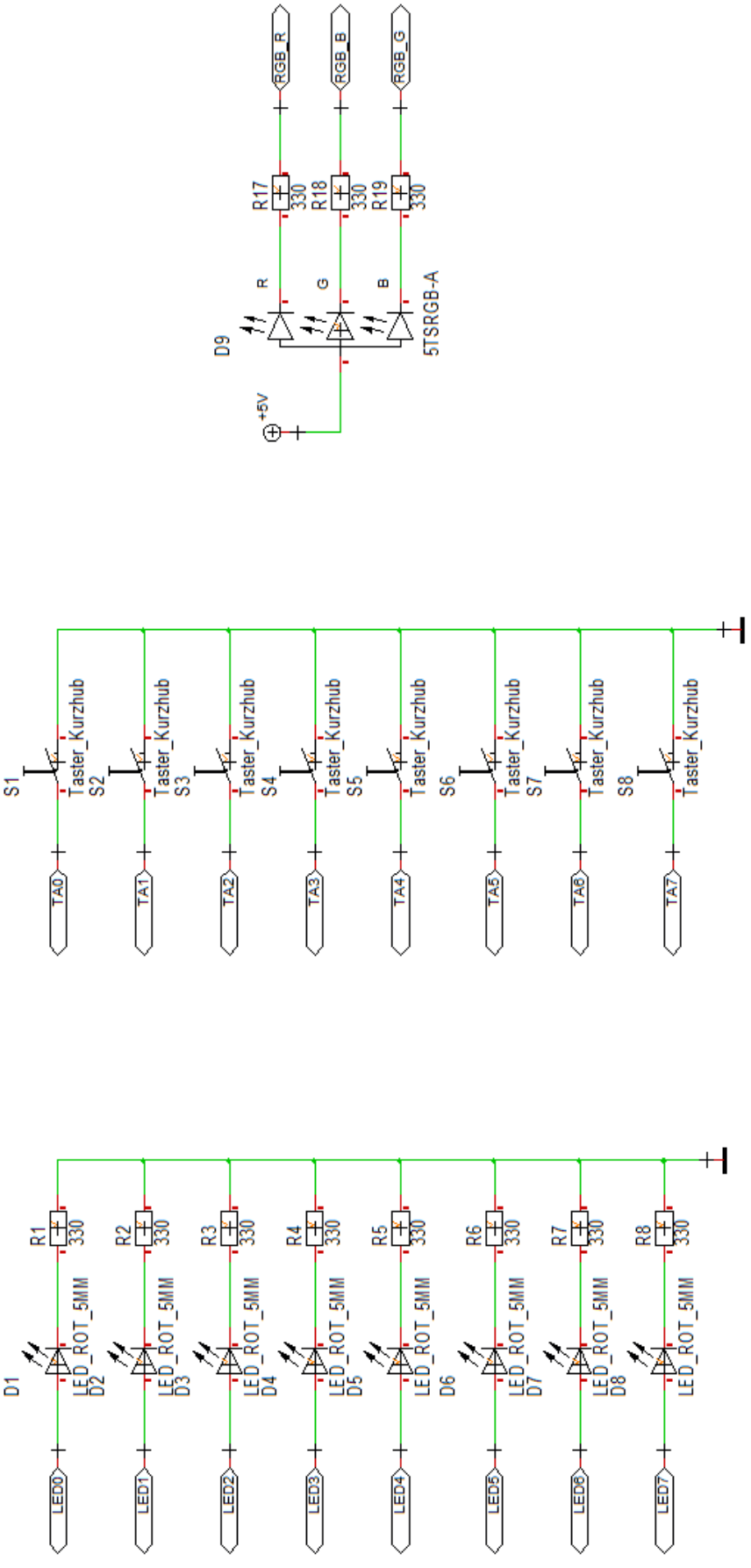
Input-Output



Shield


X1a
Power
Arduino_Mega2560_Shield_Rev3
NC
+5V
Reset
+3V3
+5V
GND
GND
VIN
+3V
+5V
Taster_Kurzhub
S9
X1b
ADCL
ADC0
ADC1
ADC2
ADC3
ADC4
ADC5
ADC6
ADC7
AN0
AN1
AN2
AN3
AN4
AN5
AN6
AN7
X1c
ADCH
ADC8
ADC9
ADC10
ADC11
ADC12
ADC13
ADC14
ADC15
SEG7a
SEG7b
SEG7c
SEG7d
SEG7e
SEG7f
SEG7g
SEG7dp
X1d
Digital
+5V
PA1
PA3
PA5
PA7
PC6
PC4
PC2
PC0
PG2
PG0
PL6
PL4
PL2
PL0
PB2
PB0
GND
IO16O1
IO16O3
IO16O5
IO16O7
IO16U6
IO16U4
IO16U2
IO16U0
TA2
TA0
LED6
LED4
LED2
LED0
MOSI
SS
+5V
PA0
PA2
PA4
PA6
PC7
PC5
PC3
PC1
PD7
PG1
PL7
PL5
PL3
PL1
PB3
PB1
GND
IO16O0
IO16O2
IO16O4
IO16O6
IO16U7
IO16U5
IO16U3
IO16U1
TA3
TA1
LED7
LED5
LED3
LED1
MISO
SCK
X1e
Communication
SCL
SDA
RXD1
TXD1
RXD2
TXD2
RXD3
TXD3
X1f
PWML
PE0
PE1
PE4
PE5
PG5
PE3
PH3
PH4
RXD0
TXD0
RGB_G
RGB_B
PWM3
RGB_R
TA4
TA5
X1g
PH5
PH6
PB4
PB5
PB6
PB7
GND
AREF
SDA
SCL
TA6
TA7
PWM2
PWM1
PWM0

onBoard


RGB_R
RGB_B
RGB_G
R17
330
R18
330
R19
330
R
G
B
D9
5TSRGB-A
+5V
S1
Taster_Kurzhub
S2
S3
S4
S5
S6
S7
S8
TA0
TA1
TA2
TA3
TA4
TA5
TA6
TA7
R1
R2
R3
R4
R5
R6
R7
R8
D1
LED_ROT_5MM
D2
D3
D4
D5
D6
D7
D8
LED0
LED1
LED2
LED3
LED4
LED5
LED6
LED7
SEG7dp
R16
Display1
LA8041-11
R9
SEG7a
SEG7b
SEG7c
SEG7d
SEG7e
SEG7f
SEG7g

ANHANG X: HINWEISE

ARDUINO VERZEICHNISSE

System:
C:\Users\Steiner\AppData\Local\Arduino15\packages\arduino\hardware\avr\1.8.6\
Usercode:→ Ableitungen vom Sketch
C:\Users\Steiner\AppData\Local\Temp\arduino-sketch-<32 Hex Stellen>\
Sketch:
Eigendefinition: → wo die *.INO Dateien gespeichert sind

AVRDUDE

avrdude ist eine Programmiersoftware auf Kommandozeilen Ebene

```
D:\avrdude\bin>avrdude
Usage: avrdude [options]
Options:
  -p <partno>                Required. Specify AVR device.
  -b <baudrate>              Override RS-232 baud rate.
  -B <bitclock>              Specify JTAG/STK500v2 bit clock period (us).
  -C <config-file>           Specify location of configuration file.
  -c <programmer>            Specify programmer type.
  -D                         Disable auto erase for flash memory
  -i <delay>                 ISP Clock Delay [in microseconds]
  -P <port>                  Specify connection port.
  -F                         Override invalid signature check.
  -e                         Perform a chip erase.
  -O                         Perform RC oscillator calibration (see AVR053).
  -U <memtype>:r|w|v:<filename>[:format]
                             Memory operation specification.
                             Multiple -U options are allowed, each request
                             is performed in the order specified.
  -n                         Do not write anything to the device.
  -V                         Do not verify.
  -u                         Disable safemode, default when running from a script.
  -s                         Silent safemode operation, will not ask you if
                             fuses should be changed back.
  -t                         Enter terminal mode.
  -E <exitspec>[,<exitspec>] List programmer exit specifications.
  -x <extended_param>        Pass <extended_param> to programmer.
  -y                         Count # erase cycles in EEPROM.
  -Y <number>                Initialize erase cycle # in EEPROM.
  -v                         Verbose output. -v -v for more.
  -q                         Quell progress output. -q -q for less.
  -l logfile                 Use logfile rather than stderr for diagnostics.
  -?                         Display this usage.

avrdude version 6.3-20190619, URL: <http://savannah.nongnu.org/projects/avrdude/>
```

AVR-OBJDUMP

Avr-objdump kann aus einer Binärdatei eine Assemblerdatei erstellen.

```
avr-objdump.EXE
Usage: avr-objdump <option(s)> <file(s)>
 Display information from object <file(s)>.
 At least one of the following switches must be given:
  -a, --archive-headers    Display archive header information
  -f, --file-headers       Display the contents of the overall file header
  -p, --private-headers    Display object format specific file header contents
  -P, --private=OPT,OPT... Display object format specific contents
  -h, --[section-]headers  Display the contents of the section headers
  -x, --all-headers        Display the contents of all headers
  -d, --disassemble        Display assembler contents of executable sections
  -D, --disassemble-all    Display assembler contents of all sections
  -S, --source             Intermix source code with disassembly
  -s, --full-contents      Display the full contents of all sections requested
  -g, --debugging          Display debug information in object file
  -e, --debugging-tags     Display debug information using ctags style
  -G, --stabs              Display (in raw form) any STABS info in the file
  -W[lLiaprmfFsoRt] or
  --dwarf[=rawline,=decodedline,=info,=abbrev,=pubnames,=aranges,=macro,=frames,
          =frames-interp,=str,=loc,=Ranges,=pubtypes,
          =gdb_index,=trace_info,=trace_abbrev,=trace_aranges,
          =addr,=cu_index]
                           Display DWARF info in the file
  -t, --syms               Display the contents of the symbol table(s)
  -T, --dynamic-syms       Display the contents of the dynamic symbol table
  -r, --reloc              Display the relocation entries in the file
  -R, --dynamic-reloc      Display the dynamic relocation entries in the file
  @<file>                  Read options from <file>
  -v, --version            Display this program's version number
  -i, --info               List object formats and architectures supported
  -H, --help               Display this information
```