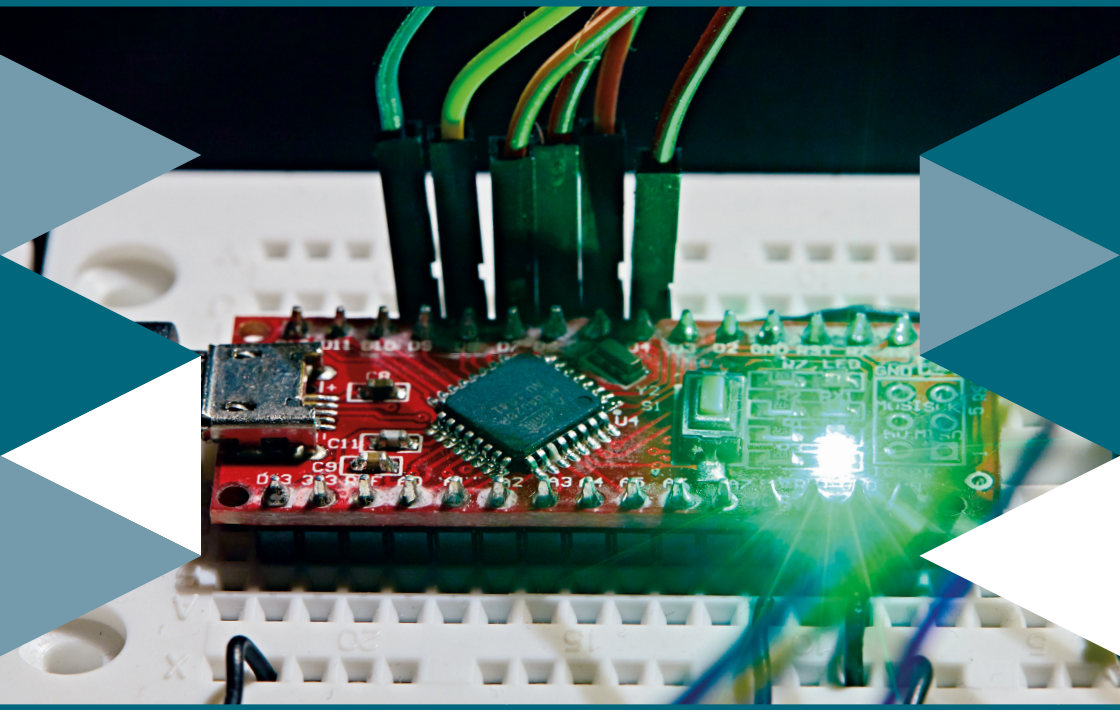
FRANZIS

BONUS-KAPITEL

MACH'S EINFACH

MAKER KIT für ARDUINO®



MAKER KIT
für
ARDUINO®
Bonuskapitel

Inhalt

1 Eigene Zeichen für das LCD-Modul erstellen

Spezielle grafische Sonderzeichen fehlen im Zeichensatz des LCD-Moduls. Er bietet aber die Möglichkeit, bis zu acht Zeichen frei zu programmieren und dann wie Buchstaben anzuzeigen.

Eigene Zeichen werden als Byte-Array aus acht fünfstelligen Binärzahlen gebildet. Jede 1 steht für ein eingeschaltetes Pixel, jede 0 für ein ausgeschaltetes.

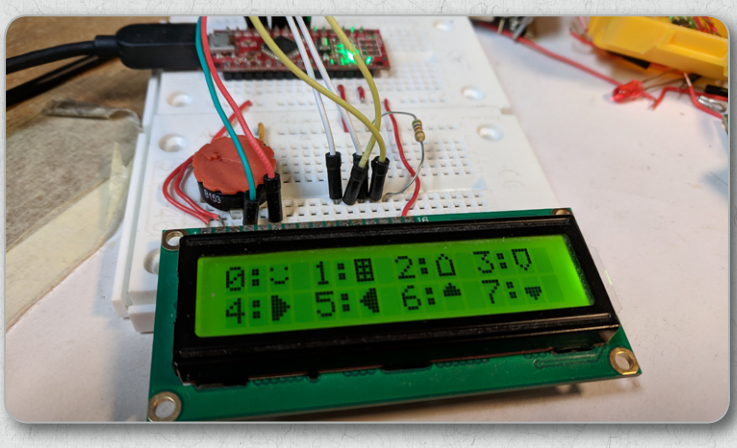


Abb. 1.1: *Eigene Zeichen auf dem LCD-Modul.*

Der Sketch `zeichen02.ino` zeigt acht verschiedene selbst erstellte Zeichen auf dem LCD-Modul.

```
#include <LiquidCrystal.h>
//lcd(RS, E, D4, D5, D6, D7);
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

byte z0[8] = {
  B00000,
  B10001,
  B00000,
  B00000,
  B10001,
  B01110,
  B00000,
  B00000
};
byte z1[8] = {
  B11111,
  B10101,
  B11111,
  B10101,
  B11111,
  B10101,
  B11111,
  B00000
};
byte z2[8] = {
  B00100,
  B01010,
  B10001,
  B10001,
  B10001,
  B10001,
```

```
  B11111,
  B00000
};
byte z3[8] = {
  B11111,
  B10001,
  B10001,
  B10001,
  B10001,
  B01010,
  B00100,
  B00000
};
byte z4[8] = {
  B11000,
  B11100,
  B11110,
  B11111,
  B11110,
  B11100,
  B11000,
  B00000
};
byte z5[8] = {
  B00011,
  B00111,
  B01111,
  B11111,
  B01111,
  B00111,
  B00011,
  B00000
};
byte z6[8] = {
  B00100,
  B01110,
  B11111,
```

```
  B11111,
  B00000,
  B00000,
  B00000,
  B00000
};
byte z7[8] = {
  B00000,
  B00000,
  B00000,
  B11111,
  B11111,
  B01110,
  B00100,
  B00000
};

void setup() {
  lcd.createChar(0, z0);
  lcd.createChar(1, z1);
  lcd.createChar(2, z2);
  lcd.createChar(3, z3);
  lcd.createChar(4, z4);
  lcd.createChar(5, z5);
  lcd.createChar(6, z6);
  lcd.createChar(7, z7);
  lcd.begin(16, 2);
  lcd.print("0:");
  lcd.write(byte(0));
  lcd.print(" 1:");
  lcd.write(byte(1));
  lcd.print(" 2:");
  lcd.write(byte(2));
  lcd.print(" 3:");
  lcd.write(byte(3));
  lcd.setCursor(0,1);
  lcd.print("4:");
```

```
  lcd.write(byte(4));
  lcd.print(" 5:");
  lcd.write(byte(5));
  lcd.print(" 6:");
  lcd.write(byte(6));
  lcd.print(" 7:");
  lcd.write(byte(7));
}

void loop() {
}
```

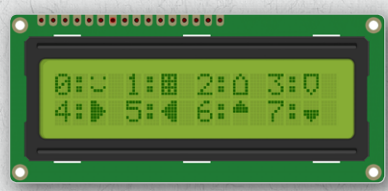


Abb. 1.2: *Das Ergebnis auf dem LCD-Modul.*

So funktioniert es

Am Anfang werden in den Byte-Arrays `z0[8]` bis `z7[8]` die Grafiken für die Sonderzeichen definiert.

```
byte z0[8] = {
  B00000,
  B10001,
  B00000,
  B00000,
  B10001,
  B01110,
  B00000,
  B00000
};
```

Jedes Array enthält acht fünfstellige Binärzahlen. Jede davon stellt eine Pixelzeile des Zeichens dar. Binärzahlen werden durch ein führendes `B` gekennzeichnet. Die fünf Bits jeder Zahl geben den Einschaltzustand der fünf Pixel in jeder Zeile eines Zeichens an.

In der Prozedur `void setup()` werden dann die Sonderzeichen angelegt.

```
lcd.createChar(0, z0);
```

Die Funktion lcd.createChar()

Die Funktion `lcd.createChar()` legt ein Sonderzeichen an. Der erste Parameter gibt die Nummer des Zeichens an. Diese kann eine Zahl von 0 bis 7 sein. Der zweite Parameter gibt das Byte-Array an, in dem die Zeichengrafik gespeichert ist.

Auf diese Weise werden alle acht Sonderzeichen definiert.

```
lcd.begin(16, 2);
lcd.print("0:");
lcd.write(byte(0));
```

Nachdem das LCD-Modul über `lcd.begin()` initialisiert wurde, folgen acht Zeilenpaare, in denen jeweils zuerst eine Nummer zwischen 0 und 7 angezeigt wird und danach das jeweilige Sonderzeichen. Die Funktion `lcd.write(byte())` schreibt ein durch seine Nummer festgelegtes Sonderzeichen an der aktuellen Cursorposition.

Da alle Zeichen nur einmal angezeigt werden und der Sketch danach nichts weiter tun muss, bleibt die Prozedur `void loop()` leer. Sie muss aber in einem Arduino-Sketch trotzdem vorhanden sein.

2 Menüsteuerung auf dem LCD-Modul

Mit dem LCD-Modul und ein paar Tastern lässt sich eine komplette Benutzeroberfläche für verschiedenste Arduino-Projekte bauen. Man braucht, wenn der Sketch einmal läuft, keinen PC mit Monitor und Tastatur mehr, sondern kann auch komplexe Arduino-Projekte über ein interaktives Menü auf dem LCD-Modul bedienen.

Eine derartige Steuerung brauchen Sie nicht einmal mehr selbst zu entwickeln. Der Arduino-Bibliotheksverwalter bringt die Bibliothek `LCDMenuLib2` mit allen Komponenten mit, die man für ein Steuerungsmenü mit mehreren Ebenen benötigt.

- 254 Menüpunkte maximal.
- 254 Menüpunkte pro Ebene möglich, keine Begrenzungen bei der Ebenenanzahl.
- Trennung von Struktur- und Funktionsebene bei der Konfiguration durch mehrere Tabs in der Arduino-IDE.
- Ansteuerung über mindestens drei Taster/Funktionen (*Auf*, *Ab*, *Enter*).
- Maximal sechs Taster werden unterstützt (*Auf*, *Ab*, *Enter*, *Zurück*, *Links*, *Rechts*).

- Ansteuerung auch über Tastatur, Drehgeber oder andere Eingabemethoden möglich.

- Startbildschirm aktivierbar, er wird nach x Sekunden und/oder zu Beginn angezeigt.

- Für verschiedene Displaygrößen konfigurierbar.

- Scrollbalken aktivierbar, wenn mehr Menüelemente als Zeilen im LCD-Modul vorhanden sind.

Eine ausführliche Beschreibung der `LCDMenuLib` finden Sie im Arduino-Forum unter *goo.gl/z9KFGT*.

Schaltungsaufbau

Das LCD-Modul ist wie in den bisherigen Projekten angeschlossen. Die Taster nutzen sechs zusätzliche digitale Eingänge. Das LCD-Modul ist nur am oberen Steckbrett angeschlossen, das untere liegt lose darunter.

Benötigte Bauteile

- 1 x Nano
- 3 x Steckplatine
- 1 x LCD-Modul
- 1 x 560-Ohm-Widerstand (Grün-Blau-Braun)
- 1 x Potentiometer
- 6 x Taster
- Schaltdraht

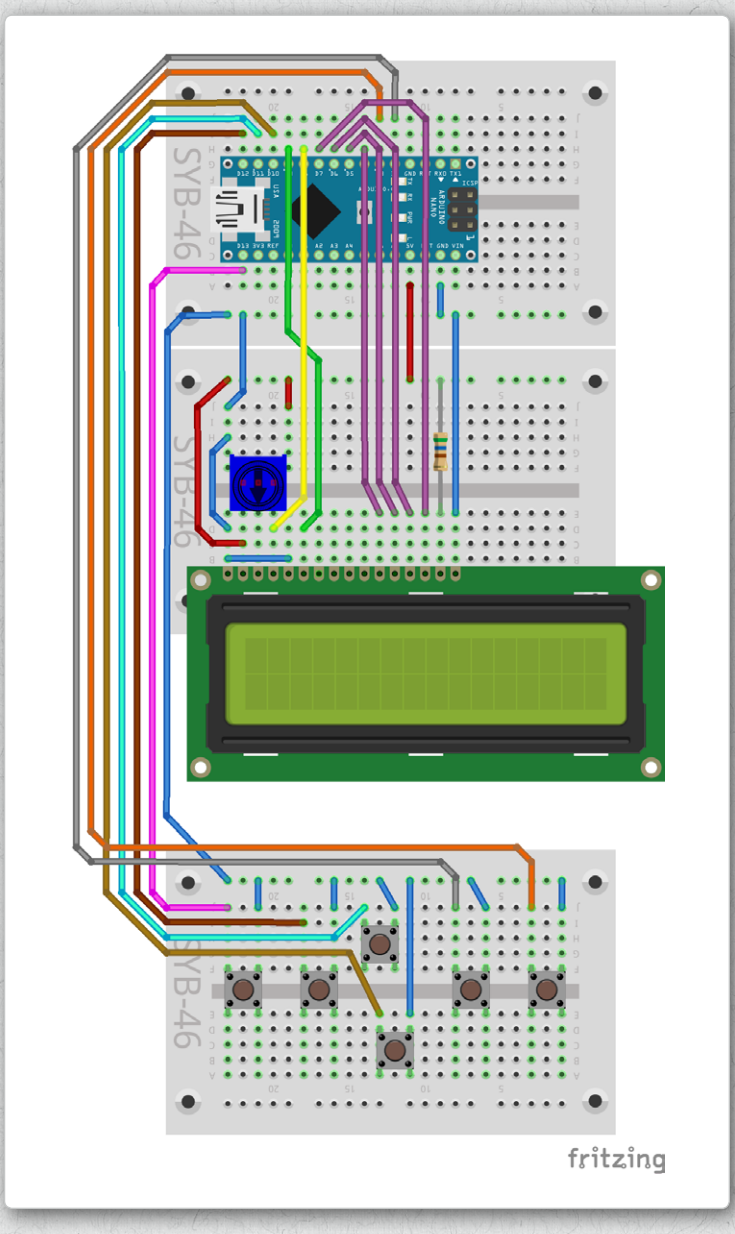


Abb. 2.1: *Schaltschema für das interaktive Menü auf dem LCD-Modul.*

LCDMenuLib2 installieren

Wählen Sie im Menü der Arduino-IDE *Werkzeuge/Bibliotheken verwalten*. Geben Sie im Bibliotheksverwalter oben rechts *LCDMenuLib2* ein, um die Liste zu filtern. Klicken Sie bei der angezeigten Bibliothek auf *Installieren*.

Öffnen Sie in der Arduino-IDE über *Datei/Beispiele/LCDMenuLib2/00_beginners/LCDML_001_liquidCrystal* das Basisprogramm für die Menüsteuerung.

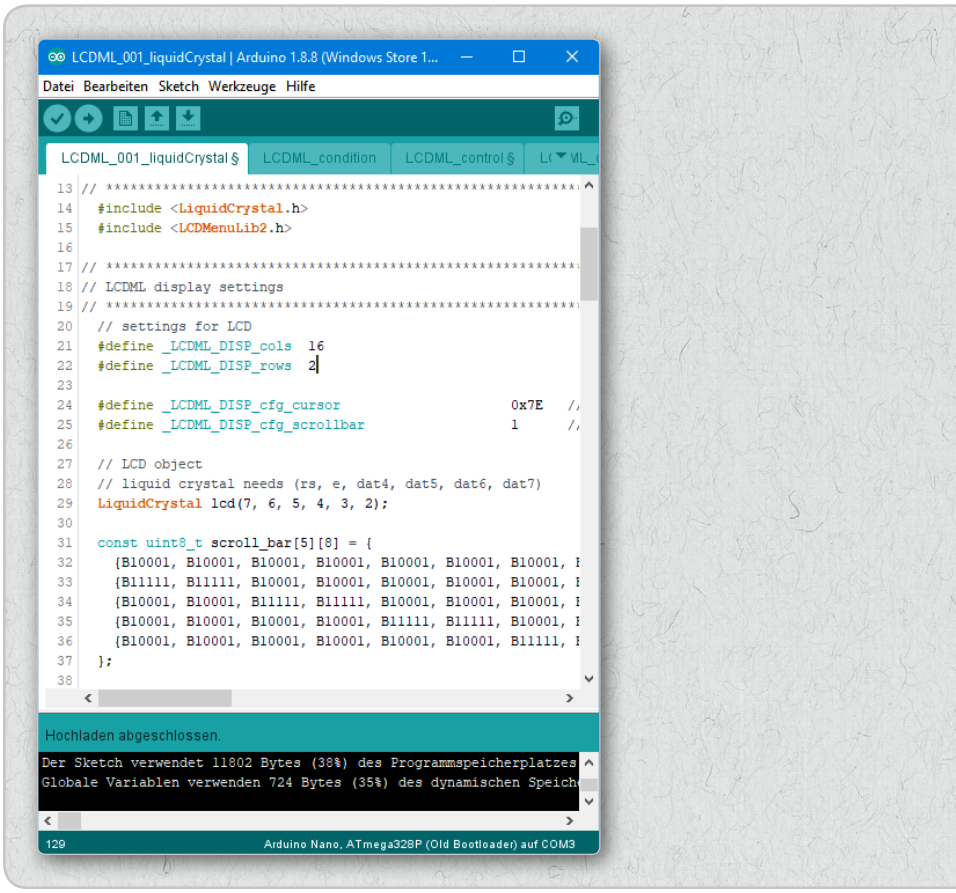


Abb. 2.2: *Das Beispiel öffnet in der Arduino-IDE mehrere Registerkarten.*

Konfiguration für das LCD-Modul anpassen

Auf der Registerkarte *LCDML_001_liquidCrystal* müssen Sie einige Anpassungen für das verwendete LCD-Modul vornehmen. Stellen Sie hier 16 Zeichen und 2 Zeilen ein:

```
// settings for LCD
#define _LCDML_DISP_cols  16
#define _LCDML_DISP_rows  2
```

Außerdem müssen die Anschlüsse für das LCD-Modul angepasst werden. Die folgende Einstellung entspricht der Verdrahtung, die für die anderen Projekte in diesem Maker Kit ebenfalls verwendet wird. Sie können auch einfach die folgende Zeile aus einem der anderen Sketches aus Ihrem Sketchbook kopieren.

```
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
```

Konfiguration für die Steuerungstasten anpassen

`LCDMenuLib2` bietet verschiedene Möglichkeiten, das Menü zu bedienen. Standardmäßig ist die Bedienung mit der PC-Tastatur über den seriellen Monitor voreingestellt. Dabei werden vier Buchstabentasten der Tastatur für die vier Richtungen verwendet, zwei weitere für die Funktionen *Quit* und *Enter*.

Abb. 2.3: *Tasten zur Steuerung.*

Auf der Registerkarte *LCDML_control* sind acht verschiedene Steuerungsmethoden vordefiniert. Um die Taster auf dem Steckbrett zur Steuerung zu verwenden, verwenden Sie Methode 2. Passen Sie dazu die folgende Zeile an:

```
#define _LCDML_CONTROL_cfg      2
```

Die Pinbelegung der Taster am Nano wird weiter unten unter der Überschrift

```
// *********** (2) CONTROL OVER DIGITAL PINS ***********
```

festgelegt. Im abgebildeten Schaltungsaufbau sind keine externen Pull-down-Widerstände eingebaut. Wir verwenden die internen Pull-up-Widerstände und dementsprechend inverse Logik. Diese Einstellung legen Sie in der folgenden Zeile fest:

```
#define _LCDML_CONTROL_digital_low_active      1
```

`LCDMenuLib2` erfordert mindestens drei Taster für die Funktionen *Auf*, *Ab* sowie *Enter*. Tragen Sie die entsprechenden Pins in den folgenden Zeilen ein. Die Tasten für *Quit*, *Links* und *Rechts* sind optional. In unserer Schaltung sind diese Tasten angeschlossen und werden an der gleichen Stelle eingerichtet:

```
#define _LCDML_CONTROL_digital_enter           3
#define _LCDML_CONTROL_digital_up              11
#define _LCDML_CONTROL_digital_down            10
#define _LCDML_CONTROL_digital_quit            13
#define _LCDML_CONTROL_digital_left            12
#define _LCDML_CONTROL_digital_right           2
```

Das Testprogramm

Laden Sie den Sketch, nachdem Sie alle Anpassungen vorgenommen haben, auf den Nano. Die ersten beiden Zeilen des Menüs sind auf dem LCD-Modul zu sehen.

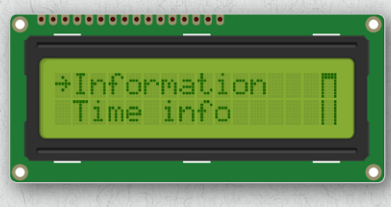


Abb. 2.4: *Der erste Bildschirm des Menüs.*

Mit den Tasten *Ab* und *Auf* bewegen Sie den Pfeilcursor, die Taste *Enter* ruft den entsprechenden Menüpunkt auf. Menüs können Untermenüs enthalten, in denen man sich auch wieder mit den Tasten *Ab* und *Auf* hin- und herbewegen kann. Die Taste *Quit* springt eine Menüebene zurück oder bricht eine dynamische Anzeige im Menü ab.

Der Scrollbalken am rechten Rand der Anzeige zeigt die ungefähre Position innerhalb eines Menüs.

So funktioniert es

Das Menü ist auf der Registerkarte *LCDML_001_liquidCrystal* im Abschnitt

```
// LCDML MENU/DISP
```

definiert und kann dort auch verändert werden. Der Übersicht halber wurden im abgedruckten Programmcode die Kommentare weggelassen.

```
// LCDML_add(id, prev_layer, new_num, lang_char_array,
             callback_function)
LCDML_add (0  , LCDML_0         , 1  , "Information"    ,
           mFunc_information);
LCDML_add (1  , LCDML_0         , 2  , "Time info"      ,
           mFunc_timer_info);
LCDML_add (2  , LCDML_0         , 3  , "Program"        ,
           NULL);
LCDML_add (3  , LCDML_0_3       , 1  , "Program 1"      ,
           NULL);
LCDML_add (4  , LCDML_0_3_1     , 1  , "P1 dummy"       ,
           NULL);
LCDML_add (5  , LCDML_0_3_1     , 2  , "P1 Settings"    ,
           NULL);
LCDML_add (6  , LCDML_0_3_1_2   , 1  , "Warm"           ,
           NULL);
LCDML_add (7  , LCDML_0_3_1_2   , 2  , "Cold"           ,
           NULL);
LCDML_add (8  , LCDML_0_3_1_2   , 3  , "Back"           ,
           mFunc_back);
LCDML_add (9  , LCDML_0_3_1     , 3  , "Back"           ,
           mFunc_back);
LCDML_add (10 , LCDML_0_3       , 2  , "Program 2"      ,
           mFunc_p2);
LCDML_add (11 , LCDML_0_3       , 3  , "Back"           ,
           mFunc_back);
LCDML_add (12 , LCDML_0         , 4  , "Special"        ,
           NULL);
LCDML_add (13 , LCDML_0_4       , 1  , "Go to Root"     ,
           mFunc_goToRootMenu);
LCDML_add (14 , LCDML_0_4       , 2  , "Jump to Time info",
           mFunc_jumpTo_timer_info);
LCDML_add (15 , LCDML_0_4       , 3  , "Back"           ,
           mFunc_back);
```

Jedes Menüelement wird über fünf Parameter definiert:

PARAMETER	BEDEUTUNG
`id`	Fortlaufende Nummer.
`prev_layer`	Ebene, in der das neue Element erzeugt werden soll.
`new_num`	Fortlaufende Nummer innerhalb der aktuellen Menüebene.
`lang_char_array`	Bezeichnung des Menüpunkts. Kann bei einem 16-Spalten-LCD-Modul 14 Zeichen lang sein.
`callback_function`	Die Funktion, die aufgerufen wird, wenn der Menüpunkt mit *Enter* ausgewählt wird. Bei Menüpunkten mit `NULL` sind keine Funktionen hinterlegt. Sie eignen sich dazu, weitere Ebenen im Menü anzulegen oder einfach nur etwas anzuzeigen.

Das Menü kann nach dem vorgegebenen Schema beliebig erweitert und verändert werden. Eine ausführliche Beschreibung der Möglichkeiten finden Sie im Arduino-Forum unter *goo.gl/z9KFGT*.

Stoppuhr mit mBlock 3

3

Der Funktionsumfang von mBlock 3 lässt sich durch vielfältige sogenannte Extensions erweitern. Darunter findet sich auch eine Extension für LCD-Module.

Wählen Sie im Menü *Extensions/Extensions verwalten*. Suchen Sie auf der Seite *Verfügbar* oben im Suchfeld nach *LCD*. Installieren Sie mit einem Klick auf *Download* die Extension *LCD*.

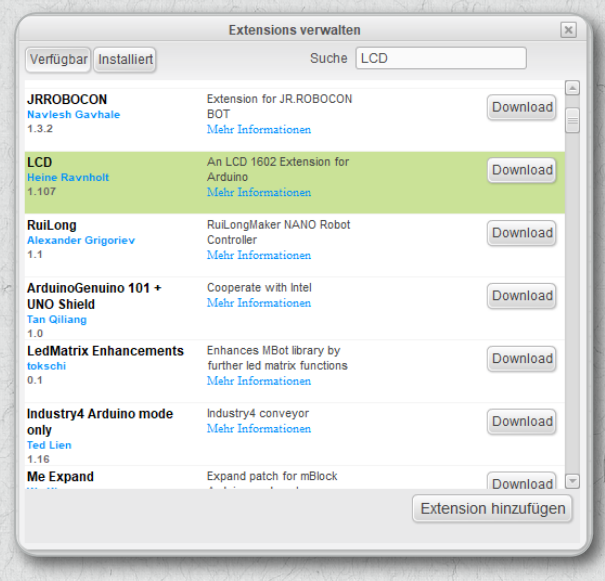


Abb. 3.1: *LCD-Extension für mBlock 3 herunterladen und installieren.*

Nach der Installation erscheinen in der Blockpalette *Roboter* die neuen Blöcke der *LCD*-Extension.

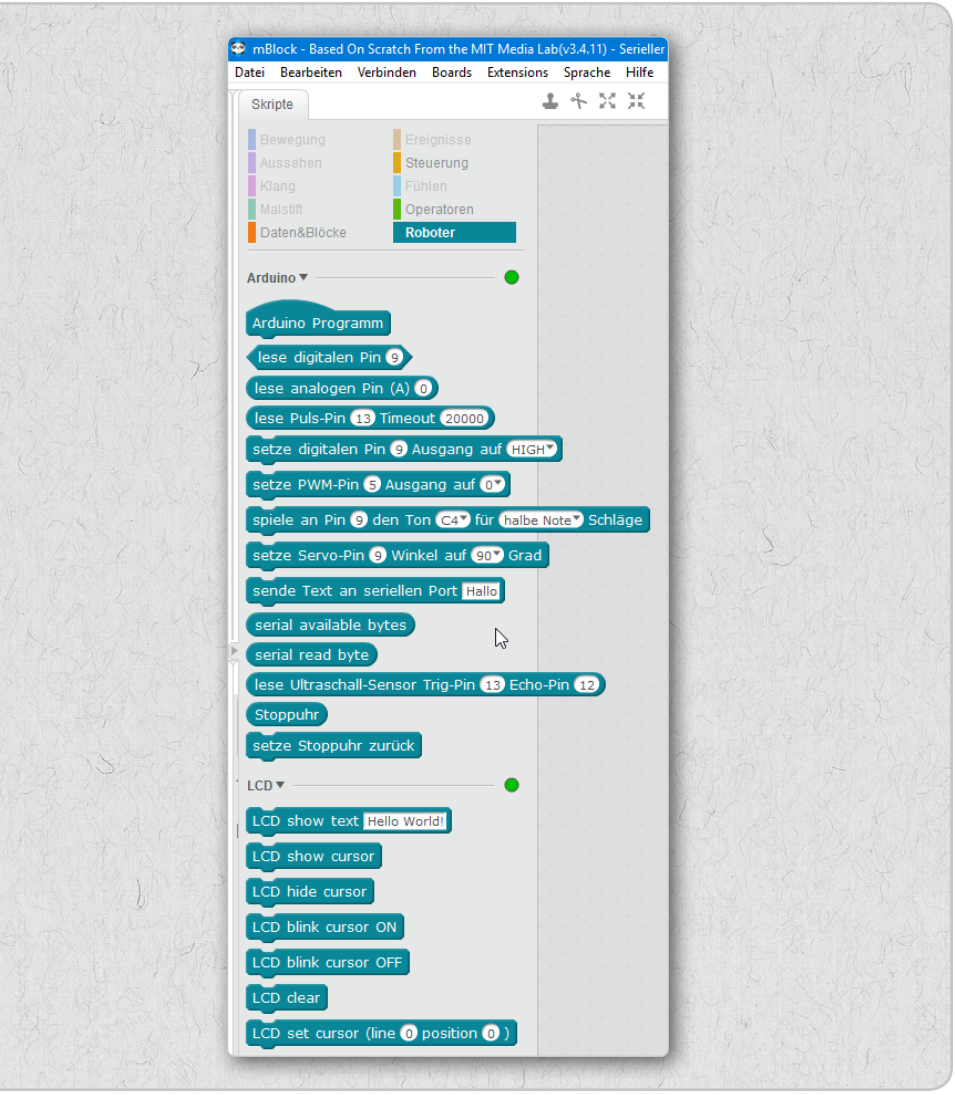


Abb. 3.2: *Die Blöcke der LCD-Extension.*

Stoppuhr mit einer Taste

Arduino-kompatible Platinen haben zwar keine Echtzeituhr, die die wirkliche Zeit anzeigt, aber einen Timer, der ständig läuft und so als Stoppuhr verwendet werden kann.

Die Schaltung nutzt einen einzigen Taster für Start, Stopp und Reset der Stoppuhr. Die untere Zeile des LCD-Moduls zeigt, welche Funktion der nächste Tastendruck auslöst.

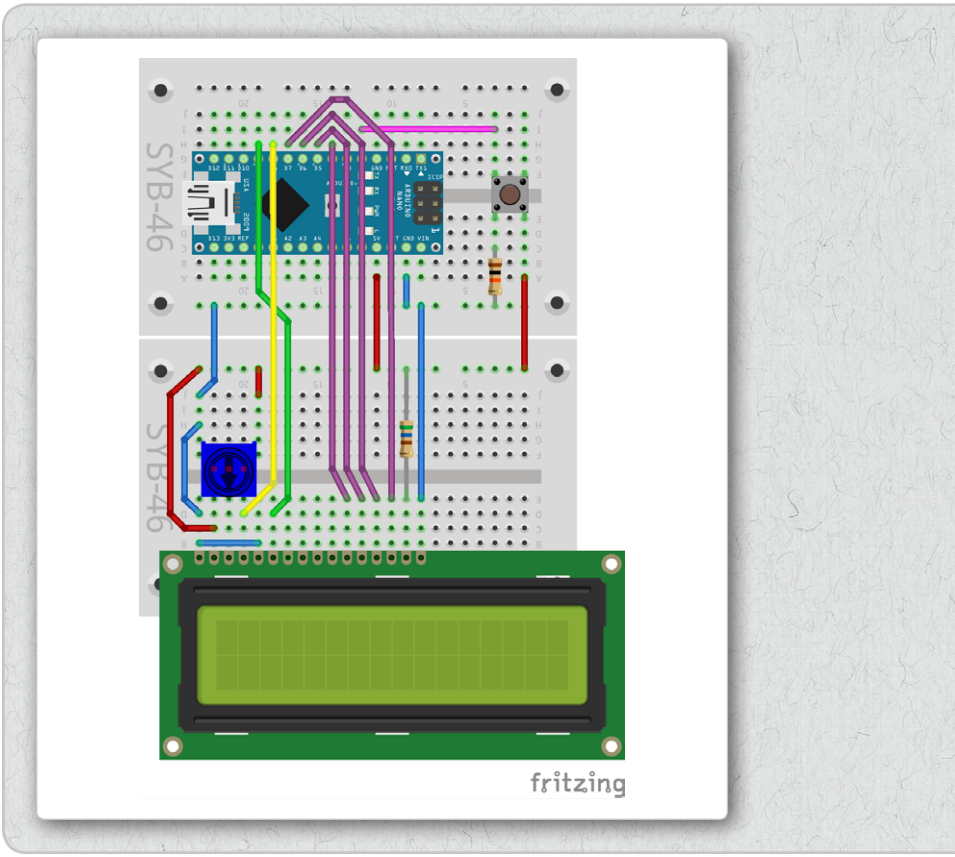


Abb. 3.3: *Stoppuhr mit Taster auf dem LCD-Modul.*

Benötigte Bauteile

- 1 x Nano
- 2 x Steckplatine
- 1 x LCD-Modul
- 1 x 560-Ohm-Widerstand (Grün-Blau-Braun)
- 1 x 10-kOhm-Widerstand (Braun-Schwarz-Orange)
- 1 x Taster
- 1 x Potentiometer
- Schaltdraht

Die *LCD*-Extension für mBlock 3 verwendet standardmäßig immer die in unseren Schaltungen eingesetzte Pinbelegung. Diese lässt sich hier nur durch manuelles Bearbeiten von Konfigurationsdateien und nicht auf der Benutzeroberfläche ändern. Daher nutzen wir diese Pinbelegung für alle Schaltungen in diesem Maker Kit.

Das Programm für die Stoppuhr

Das Programm `stoppuhr01mblock.sb2` aus den Downloads lässt eine Stoppuhr auf dem LCD-Modul laufen. Übertragen Sie das Programm im Arduino-Modus auf den Nano.

So funktioniert es

Das Programm besteht aus einer großen Endlosschleife, in der am Anfang die Stoppuhr und auch die Anzeige auf dem LCD-Modul zurückgesetzt werden. Nachdem der Benutzer die Taste gedrückt hat, läuft die Stoppuhr, bis er sie erneut drückt. Wird die Taste ein drittes Mal gedrückt, startet die Endlosschleife von Neuem.

Abb. 3.4: *Das Programm für die Stoppuhr.*

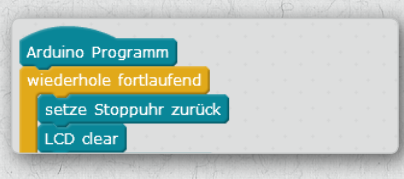


Abb. 3.5: *Stoppuhr und LCD-Modul zurücksetzen.*

Der Block *setze Stoppuhr zurück* aus der *Arduino*-Extension setzt die interne Stoppuhr auf dem Nano zurück. Direkt zu sehen ist davon noch nichts.

Der Block *LCD clear* aus der *LCD*-Extension löscht das LCD-Modul und setzt die Cursorposition für den nächsten Text in die linke obere Ecke.

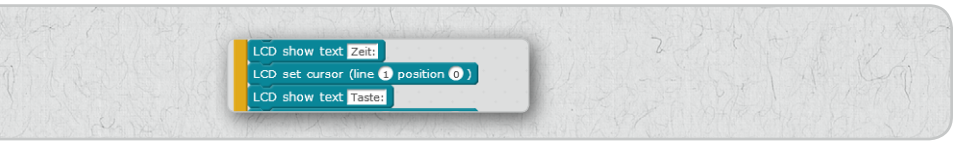


Abb. 3.6: *Text auf dem LCD-Modul ausgeben.*

Der Block *LCD show text ...* aus der *LCD*-Extension zeigt einen Text auf dem LCD-Modul an. Dieser beginnt an der aktuellen Cursorposition. Der erste Block schreibt das Wort `Zeit:` an den Anfang der oberen Zeile.

Der Block *LCD set cursor (line ... position ...)* aus der *LCD*-Extension setzt den Cursor auf die im ersten Feld angegebene Zeile (*0* oder *1*) und die im zweiten Feld angegebene Spalte (*0-15*), in diesem Fall an den Anfang der unteren Zeile. Hier wird das Wort `Taste:` geschrieben.

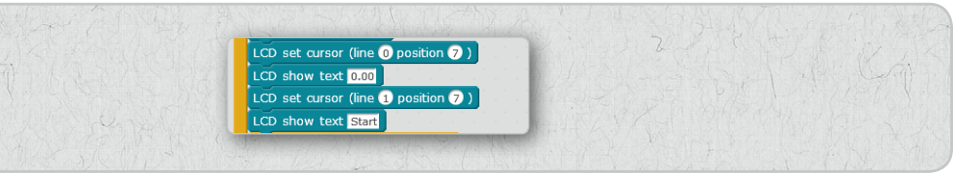


Abb. 3.7: *Weitere Texte auf dem LCD-Modul ausgeben.*

An Position 7 in der oberen Zeile wird der Text `0.00` angezeigt. Hier erscheint später, wenn die Stoppuhr läuft, die Zeit. An der gleichen Position in der unteren Zeile erscheint das Wort `Start`.

Hier wird immer angezeigt, was der nächste Tastendruck bewirkt.

Abb. 3.8: *So sieht die Anzeige beim Programmstart aus.*

Jetzt wartet das Programm, bis der Benutzer die Taste drückt.

Abb. 3.9: *Abfrage der Taste zum Start der Stoppuhr.*

Der Block *warte bis ...* von der Blockpalette *Steuerung* wartet, bis der Logikwert im Feld wahr ist. An dieser Stelle wird der Wert des digitalen Pins 2 ausgelesen, an dem der Taster angeschlossen ist. Damit ein längerer Tastendruck nicht zu Fehlinterpretationen führt, wartet das Programm auch noch, bis der Taster losgelassen wird. Erst dann soll die Stoppuhr starten. Mit dem Block *nicht ...* von der Blockpalette *Operatoren* lässt sich ein Logikwert umkehren. Somit wartet der zweite *warte bis ...*-Block, bis der Taster wieder losgelassen wird.

Danach erscheint in der unteren Zeile anstelle des dort angezeigten `Start` das Wort `Stopp`, da der nächste Tastendruck die Stoppuhr anhält.

Abb. 3.10: *Die Stoppuhr läuft.*

Anschließend startet eine Schleife, in der die Stoppuhr so lange läuft, bis wieder die Taste an Pin 2 gedrückt wird. In jedem Durchlauf wird der Wert des Blocks *Stoppuhr* aus der *Arduino*-Extension an Position 7 in der oberen Zeile des LCD-Moduls angezeigt. Dieser Block kann wie eine Variable auch für Berechnungen eingesetzt werden.

Abb. 3.11: *Die Stoppuhr wird angehalten.*

Drückt der Benutzer die Taste, wird die Schleife beendet. Danach wird wieder gewartet, bis er die Taste loslässt, und anschließend wird in der unteren Zeile das Wort `Reset` angezeigt. Der nächste Tastendruck soll die Stoppuhr zurücksetzen.

Abb. 3.12: *Das Ende der Hauptschleife.*

Drückt der Benutzer die Taste erneut, wird der aktuelle Durchgang der Hauptschleife beendet. Die Schleife springt wieder an den Anfang, wo die Stoppuhr zurückgesetzt wird.

LCD-Statusanzeige für Windows-PCs

4

Besonders in der Casemodder-Szene werden LCD-Module gern zur Anzeige verschiedenster Statusdaten des PCs außen auf dem Gehäuse verwendet. Von der CPU-Temperatur über Geschwindigkeit und Auslastung der Internetverbindung, Anzeige des freien Festplattenspeichers bis hin zu neuen E-Mails oder der aktuellen Musik-Playliste lassen sich allerhand Informationen über ein LCD-Modul auf dem PC-Gehäuse anzeigen.

LCD Smartie

LCD Smartie (*lcdsmartie.sourceforge.net*) ist eine Freeware für Windows, die diverse Statusdaten des PCs ausliest und zur Darstellung auf LCD-Modulen aller Art aufbereitet.

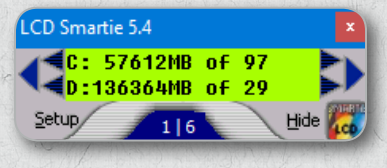


Abb. 4.1: *LCD Smartie simuliert die Anzeige auf dem PC.*

Früher konnten LCD-Module im 8-Bit-Modus direkt an die parallele Druckerschnittstelle des PCs angeschlossen werden. Seit die meisten PCs aber keinen Parallelport mehr haben, fällt diese Möglichkeit weg. Mithilfe eines Arduinos lassen sich LCD-Module

über USB am PC anschließen und zusammen mit LCD Smartie nutzen. Das Programm kann direkt aus dem entpackten Download gestartet werden. Es wird nicht installiert.

Alten Palm als Anzeige verwenden

Wer noch einen alten Palm OS-PDA herumliegen hat, kann auch ihn mit LCD Smartie als Statusanzeige für den PC verwenden. Die Software PalmOrb (*palmorb.sourceforge.net*) stellt die Verbindung zwischen PC und Palm her.

LCD Smartie stellt die Informationen auf mehreren sogenannten Screens dar, die automatisch nacheinander angezeigt werden. Im Windows-Programm kann man durch Anklicken der Pfeile zwischen den Screens wechseln und auch die einzelnen Zeilen vor- und zurückscrollen.

Arduino-Sketch zur Verbindung mit dem PC

Im Internet findet man jede Menge Arduino-Sketches, die die Verbindung zwischen einem PC, auf dem LCD Smartie läuft, und einem Arduino herstellen sollen. Der Sketch `lcd_smartie.ino` aus den Downloads zum Maker Kit basiert auf diversen Freewaresketches, ist übersichtlicher als die meisten Vorlagen und speziell für den Zeichensatz des verwendeten LCD-Moduls angepasst. Die Belegung der Arduino-Pins entspricht den anderen Projekten in diesem Maker Kit.

```
#include <LiquidCrystal.h>
//lcd(RS, E, D4, D5, D6, D7);
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

void setup() {
  Serial.begin(115200);
  lcd.begin(16,2);
```

```
  lcd.clear();
  lcd.print("* LCD  Smartie *");
  lcd.setCursor(0,1);
  lcd.print("* Franzis Nano *");
}

byte serial_getch() {
  int incoming;
  while (Serial.available()==0) {}
  incoming = Serial.read();
  return (byte) (incoming &0xff);
}

void loop() {
  byte rxbyte;
  byte temp;
  byte col;
  byte row;
  uint8_t location;
  uint8_t charMap[7];

  rxbyte = serial_getch();
  if (rxbyte == 254) //Matrix Orbital
  {
    switch (serial_getch())
    {
      case 71:  //Cursorposition setzen
        col = (serial_getch() - 1);
        row = (serial_getch());
        lcd.setCursor(col,row-1);
        break;
      case 72:  //Cursor Home
        lcd.setCursor(0,0);
        break;
      case 78:  //benutzerdefinierte Zeichen
        location = serial_getch();
        for (temp = 0; temp < 8; temp++)
```

```
    {
      charMap[temp] = serial_getch(); //Bytemuster
    }
    lcd.createChar(location, charMap);
    break;
  case 88:  //Display löschen
    lcd.clear();
    lcd.setCursor(0,0);
    break;

  //diese Kommandos werden ignoriert
  case 35: //read serial number
  case 36: //read version number
  case 55: //read module type
  case 59: //exit flow-control mode
  case 65: //auto transmit keypresses
  case 66: //backlight on
  case 96: //auto-repeat mode off (keypad)
  case 67: //auto line-wrap on
  case 68: //auto line-wrap off
  case 70: //backlight off
  case 74: //show underline cursor
  case 75: //underline cursor off
  case 76: //move cursor left
  case 77: //move cursor right
  case 81: //auto scroll on
  case 82: //auto scroll off
  case 83: //show blinking block cursor
  case 84: //block cursor off
  case 86: //GPO OFF
  case 87: //GPO ON
  case 104: //init horiz bar graph
  case 109: //init med size digits
  case 115: //init narrow vert bar graph
  case 118: //init wide vert bar graph
  case 152: //set and remember
  case 153: //set backlight brightness
```

```
        break;
      default:
        //alle anderen Kommandos werden ignoriert, und
das Parameter-Byte wird gelöscht
        temp = serial_getch();
        break;
    }
    return;
  } //ENDE DES KOMMANDO-HANDLERS

  switch (rxbyte)
  {
    //Zeichen, die im Zeichensatz enthalten sind
    case 0xC3: //ASCII "A" Tilde
      rxbyte = 0xAA;
      break;
    case 0xC4: //ASCII "A" Umlaut
      rxbyte = 0x8E;
      break;
    case 0xC5: //ASCII "A" Ring
      rxbyte = 0x8F;
      break;
    case 0xC6: //ASCII "AE" Ligatur
      rxbyte = 0x92;
      break;
    case 0xC7: //ASCII "C" Cedilla
      rxbyte = 0x80;
      break;
    case 0xC9: //ASCII "E" Acute
      rxbyte = 0x90;
      break;
    case 0xD1: //ASCII "N" Tilde
      rxbyte = 0x9C;
      break;
    case 0xD5: //ASCII "O" Tilde
      rxbyte = 0xAC;
      break;
```

```
case 0xD6: //ASCII "O" Umlaut
  rxbyte = 0x99;
  break;
case 0xD8: //ASCII "O" Strich
  rxbyte = 0xAE;
  break;
case 0xDC: //ASCII "U" Umlaut
  rxbyte = 0x9A;
  break;
case 0xDF: //ASCII "s" scharf
  rxbyte = 0xE0;
  break;
case 0xE0: //ASCII "a" Gravis
  rxbyte = 0x85;
  break;
case 0xE1: //ASCII "a" Acute
  rxbyte = 0xA0;
  break;
case 0xE2: //ASCII "a" Zirkumflex
  rxbyte = 0x83;
  break;
case 0xE3: //ASCII "a" Tilde
  rxbyte = 0xAB;
  break;
case 0xE4: //ASCII "a" Umlaut
  rxbyte = 0x84;
  break;
case 0xE5: //ASCII "a" Ring
  rxbyte = 0x86;
  break;
case 0xE6: //ASCII "ae" Ligatur
  rxbyte = 0x91;
  break;
case 0xE7: //ASCII "c" Cedilla
  rxbyte = 0x87;
  break;
case 0xE8: //ASCII "e" Gravis
```

```
  rxbyte = 0x8A;
  break;
case 0xE9: //ASCII "e" Acute
  rxbyte = 0x82;
  break;
case 0xEA: //ASCII "e" Zirkumflex
  rxbyte = 0x88;
  break;
case 0xEB: //ASCII "e" Trema
  rxbyte = 0x89;
  break;
case 0xEC: //ASCII "i" Gravis
  rxbyte = 0x8D;
  break;
case 0xED: //ASCII "i" Acute
  rxbyte = 0xA1;
  break;
case 0xEE: //ASCII "i" Zirkumflex
  rxbyte = 0x8C;
  break;
case 0xEF: //ASCII "i" Trema
  rxbyte = 0x8B;
  break;
case 0xF1: //ASCII "n" Tilde
  rxbyte = 0x9B;
  break;
case 0xF2: //ASCII "o" Gravis
  rxbyte = 0x95;
  break;
case 0xF3: //ASCII "o" Acute
  rxbyte = 0xA2;
  break;
case 0xF4: //ASCII "o" Zirkumflex
  rxbyte = 0x93;
  break;
case 0xF5: //ASCII "o" Tilde
  rxbyte = 0xAD;
  break;
```

```
case 0xF6: //ASCII "o" Umlaut
  rxbyte = 0x94;
  break;
case 0xF8: //ASCII "o" Strich
  rxbyte = 0xAF;
  break;
case 0xF9: //ASCII "u" Gravis
  rxbyte = 0x97;
  break;
case 0xFA: //ASCII "u" Acute
  rxbyte = 0xA3;
  break;
case 0xFB: //ASCII "u" Zirkumflex
  rxbyte = 0x96;
  break;
case 0xFC: //ASCII "u" Umlaut
  rxbyte = 0x81;
  break;

//weitere Symbolumsetzungen
case 0xA3: //Pfund Sterling
  rxbyte = 0xA5;
  break;
case 0xA7: //Paragraf
  rxbyte = 0xD2;
  break;
case 0xB2: //hochgestellte 2
  rxbyte = 0x1E;
  break;
case 0xB3: //hochgestellte 3
  rxbyte = 0x1F;
  break;
case 0xB5: //My
  rxbyte = 0xEA;
  break;
case 0xC0: //"A" Varianten -> "A"
case 0xC1:
```

```
    case 0xC2:
      rxbyte = 0x41;
      break;
    case 0xC8: //"E" Varianten -> "E"
    case 0xCA:
    case 0xCB:
      rxbyte = 0x45;
      break;
    case 0xCC: //"I" Varianten -> "I"
    case 0xCD:
    case 0xCE:
    case 0xCF:
      rxbyte = 0x49;
      break;
    case 0xD2: //"O" Varianten -> "O"
    case 0xD3:
    case 0xD4:
      rxbyte = 0x4F;
      break;
    case 0xD9: //"U" Varianten -> "U"
    case 0xDA:
    case 0xDB:
      rxbyte = 0x55;
      break;
    case 0xDD: //"Y" Acute -> "Y"
      rxbyte = 0x59;
      break;
    case 0xF7: //Divisionssymbol
      rxbyte = 0xB8;
      break;
    default:
      break;
  }

  lcd.write(rxbyte);
}
```

Sollten Sie LCD Smartie bereits auf dem PC laufen haben, beenden Sie es zunächst, um den Sketch auf den Arduino zu laden. Danach erscheint der Startbildschirm auf dem LCD-Modul.

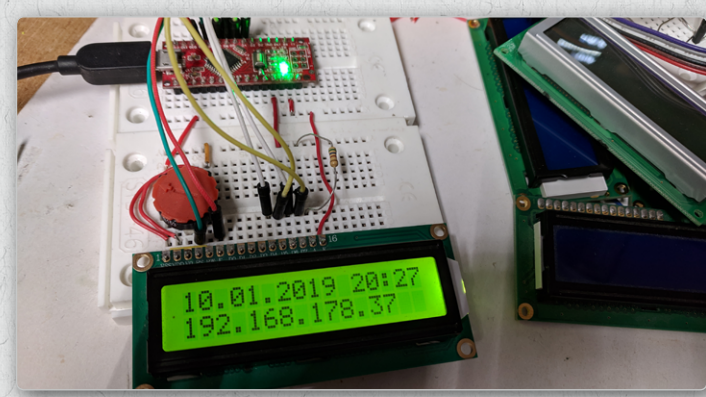


Abb. 4.2: *LCD-Modul mit LCD Smartie am Nano.*

So funktioniert es

Am Anfang wird in der Prozedur `void setup()` die serielle Verbindung mit 115200 Baud eingerichtet, über die LCD Smartie per USB mit dem Arduino kommuniziert. Die bereits bekannten Funktionen richten das LCD-Modul ein.

Eine neu definierte Funktion `serial_getch()` liest die über die serielle Verbindung ankommenden Zeichen und gibt sie als einzelne Bytes an die Hauptprozedur `void loop()` weiter.

LCD Smartie verwendet, wie viele ähnliche Programme, zur Datenausgabe an das LCD-Modul das Datenprotokoll von Matrix Orbital (*www.matrixorbital.com*), einem Hersteller von LCD- und anderen Anzeigemodulen, die über I2C- oder RS32-Schnittstellen an verschiedenste Geräte angeschlossen werden können.

Dieses Datenprotokoll verwendet das Byte `254` als Präfix vor einem Kommando. Ohne dieses Präfix-Byte werden alle Bytes als Zeichen ausgewertet und dargestellt. Alle Zeichen werden im ANSI-Code übertragen. Jedes Sonderzeichen, wie unter anderem Umlaute und andere europäische Buchstaben mit Akzent, sind nur ein Byte lang und nicht zwei wie im Unicode. Daher läuft auch die Ersetzung dieser Zeichen etwas anders als in einem vorherigen Programm.

Wird ein Byte `254` empfangen, wird das nächste Byte eingelesen und über eine ausführliche `switch/case`-Abfrage ausgewertet. Wichtige Kommandos sind hier nur `71`, Cursorposition setzen, `72`, Cursor Home, `78`, benutzerdefinierte Zeichen einlesen und `88`, Display löschen.

```
rxbyte = serial_getch();
if (rxbyte == 254) //Matrix Orbital
{
  switch (serial_getch())
  {
    case 71:  //Cursorposition setzen
      col = (serial_getch() - 1);
      row = (serial_getch());
      lcd.setCursor(col,row-1);
      break;
    case 72:  //Cursor Home
      lcd.setCursor(0,0);
      break;
    case 78:  //benutzerdefinierte Zeichen
      location = serial_getch();
      for (temp = 0; temp < 8; temp++)
      {
        charMap[temp] = serial_getch();
      }
      lcd.createChar(location, charMap);
      break;
```

```
    case 88:  //Display löschen
      lcd.clear();
      lcd.setCursor(0,0);
      break;
```

Viele der weiteren möglichen Kommandos werden ignoriert, da sie nicht benötigt werden. Sie sind anschließend aufgelistet. Alle weiteren Kommandos werden ebenfalls ignoriert, allerdings muss in diesen Fällen noch ein weiteres Parameter-Byte eingelesen und ebenfalls ignoriert werden, damit es nicht als Zeichen angezeigt wird.

```
    case 153: //Helligkeit wird vom LCD-Modul
    nicht unterstützt
      break;
    default:
      //alle anderen Kommandos werden ignoriert, und
      //das Parameter-Byte wird gelöscht
      temp = serial_getch();
      break;
```

War ein eingelesenes Zeichen kein Kommando, wird es als Zeichen dargestellt. Dazu werden in einer weiteren `switch`/`case`-Abfrage die ASCII-Codes der Umlaute und verschiedener internationaler Sonderzeichen abgefragt und durch die entsprechenden Zeichen ersetzt, wenn sie an anderer Stelle im Zeichensatz des LCD-Moduls vorhanden sind.

```
switch (rxbyte)
{
  case 0xC3: //ASCII "A" Tilde
    rxbyte = 0xAA;
    break;
  case 0xC4: //ASCII "A" Umlaut
    rxbyte = 0x8E;
    break;
  case 0xC5: //ASCII "A" Ring
```

```
      rxbyte = 0x8F;
      break;
    case 0xC6: //ASCII "AE" Ligatur
      rxbyte = 0x92;
      break;
```

Ist ein Sonderzeichen nicht vorhanden, wird es durch den entsprechenden Buchstaben ohne Akzent ersetzt, damit keine falschen Zeichen in der Anzeige erscheinen.

```
    case 0xC0: //"A" Varianten -> "A"
    case 0xC1:
    case 0xC2:
      rxbyte = 0x41;
      break;
    case 0xC8: //"E" Varianten -> "E"
    case 0xCA:
    case 0xCB:
      rxbyte = 0x45;
      break;
```

Ganz am Ende wird das Zeichen auf das LCD-Modul geschrieben, und die Prozedur `void loop()` startet neu zur Darstellung des nächsten Zeichens.

```
  lcd.write(rxbyte);
}
```

LCD Smartie einrichten

Nachdem die Verbindung mit dem Nano hergestellt ist, starten Sie LCD Smartie neu. Es wird jetzt noch keine sinnvollen Daten auf dem LCD-Modul anzeigen. Klicken Sie im Displayfenster von LCD Smartie unten links auf *Setup*. Damit öffnet sich das Einrichtungsfenster.

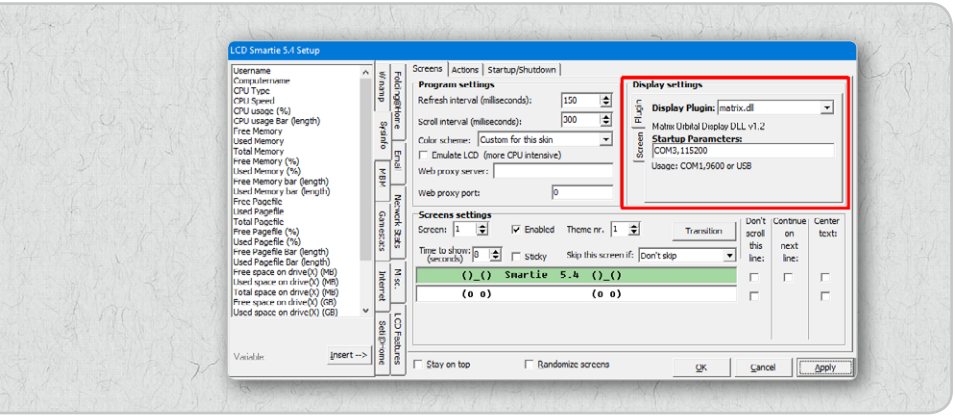


Abb. 4.3: *Displayeinstellungen in LCD Smartie.*

Wählen Sie im Feld *Display settings* unter *Display Plugin* das Plug-in *matrix.dll*. Das Plug-in *HD44780.dll* funktioniert nur mit LCD-Modulen, die am Parallelport angeschlossen sind. Tragen Sie im Feld *Startup Parameters* die serielle Schnittstelle ein, die Sie auch in der Arduino-IDE verwenden, und setzen Sie die Übertragungsrate auf *115200*.

Schalten Sie im Feld *Display settings* auf die Registerkarte *Screen* um und wählen Sie dort bei *LCD size* die Option *2x16*.

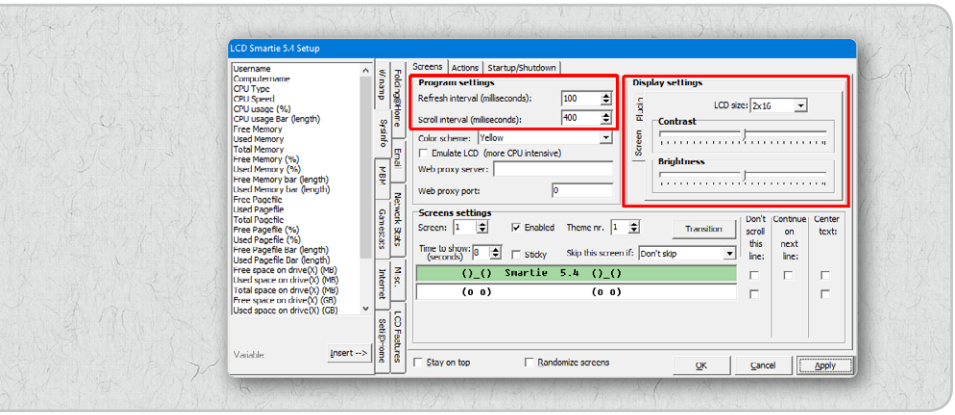


Abb. 4.4: *Displaygröße und Program settings.*

In den *Program settings* können Sie das *Refresh interval* kürzer als die vorgeschlagenen 150 Millisekunden setzen, solange das LCD-Modul noch nicht flackert. Das *Scroll interval* sollten Sie etwas verlängern, damit die Schrift beim Scrollen noch lesbar bleibt. *400* bis *500* sind gute Werte für das verwendete LCD-Modul. Klicken Sie nach einer Änderung auf *Apply* unten rechts, und diese Änderung wird sofort auf dem LCD-Modul sichtbar.

Datenanzeige konfigurieren

LCD Smartie zeigt in der Grundkonfiguration nur einen Startbildschirm mit der Versionsnummer an, Sie können diese Anzeige aber beliebig frei konfigurieren.

Wählen Sie dazu im Feld *Screens settings* den gewünschten Screen. Dann können Sie die beiden Zeilen des LCD-Moduls mit beliebigen Texten füllen.

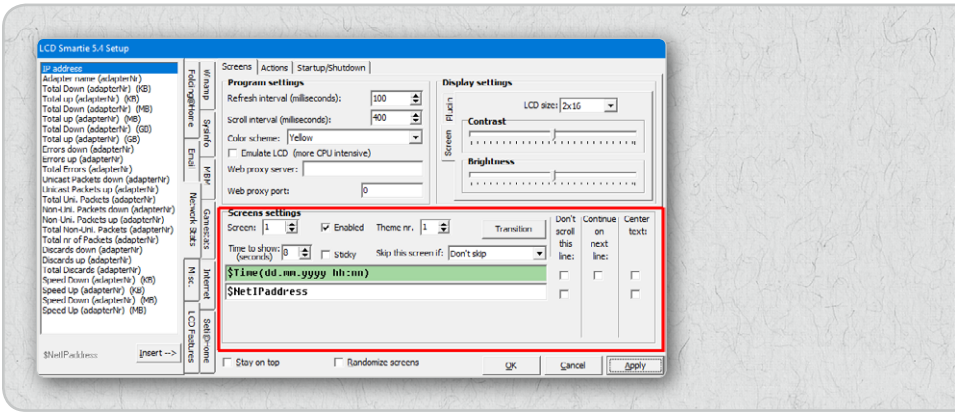


Abb. 4.5: *Screens settings.*

Das abgebildete Beispiel zeigt in der oberen Zeile eine Digitaluhr mit Datumsanzeige und unten die IP-Adresse des PCs. Das Darstellungsformat für das Datum lässt sich in der Zeichenkette anpassen.

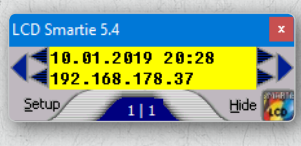


Abb. 4.6: *Digitaluhr und IP-Adresse in LCD Smartie.*

Auf den Registerkarten im linken Teilfenster finden Sie diverse Statusanzeigen, die Sie nun auswählen und mit einem Klick auf *Insert* an der aktuellen Cursorposition in den Displayzeilen einfügen können.

Längere Texte scrollen auf dem LCD-Modul standardmäßig automatisch. Um das zu verhindern, können Sie bei der entsprechenden Zeile den Schalter *Don't scroll this line* aktivieren.

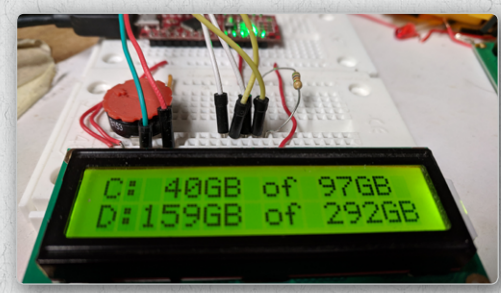


Abb. 4.7: *Anzeige der Festplattenauslastung des PCs mit LCD Smartie.*

Im Feld *Time to show* legen Sie fest, wie lange der jeweilige Screen auf dem LCD-Modul zu sehen sein soll. Solange das Konfigurationsfenster geöffnet ist, wechseln die Screens nie automatisch. Im Feld *Skip this screen if* können Sie einen Screen beim Wechseln automatisch überspringen – wenn z. B. kein WinAmp auf dem PC läuft, brauchen Sie auch dessen Playliste nicht anzuzeigen.

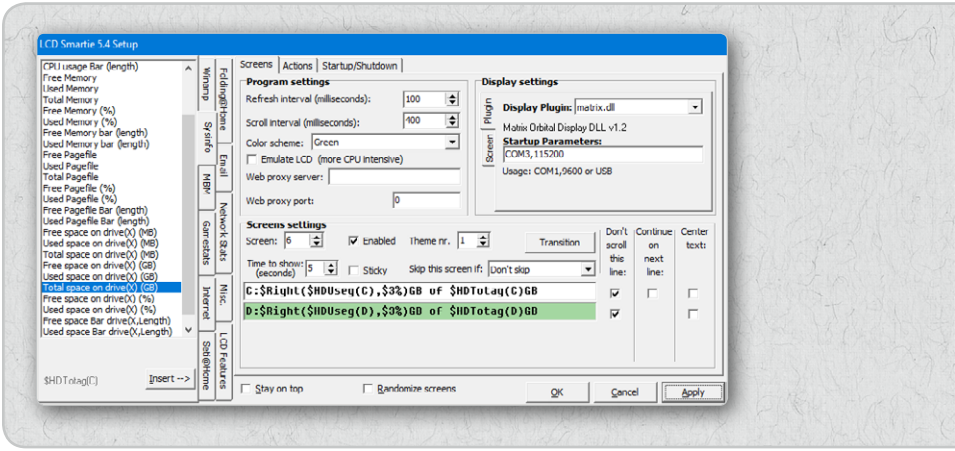


Abb. 4.8: *Parameter für die Festplattenauslastungsanzeige in LCD Smartie.*

Um LCD Smartie sinnvoll als Statusanzeige zu nutzen, starten Sie das Programm auf der Registerkarte *Startup/Shutdown* automatisch mit Windows. Die Arduino-IDE müssen Sie nicht jedes Mal mit starten, da der Nano den zuletzt aufgespielten Sketch automatisch startet, wenn er über den USB-Anschluss Strom vom PC bekommt.

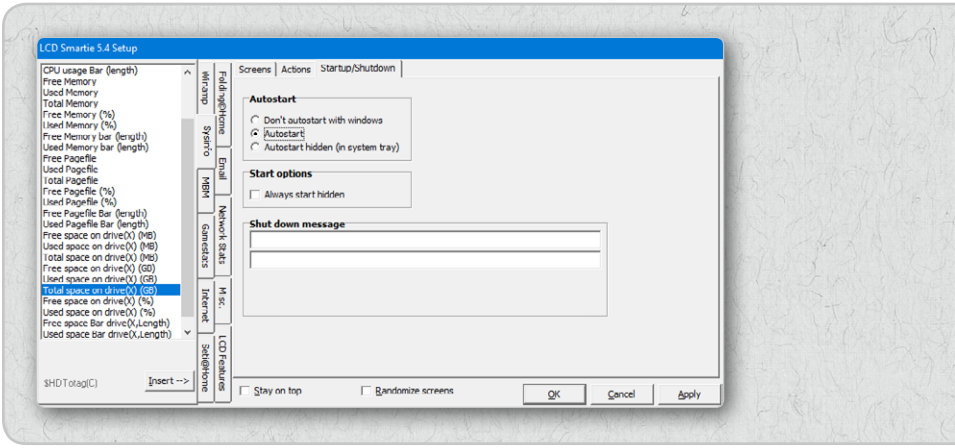


Abb. 4.9: *LCD Smartie automatisch mit Windows starten.*

Umlaute und Sonderzeichen in LCD Smartie

Der Arduino-Sketch setzt Umlaute und Sonderzeichen in den Zeichensatz des LCD-Moduls um. Wenn Sie die Umsetzung von Zeichen überprüfen wollen – besonders wenn Sie den Sketch um weitere Zeichen erweitern –, lassen Sie sich am einfachsten von LCD Smartie eine Textdatei auf dem PC anzeigen. In diese Datei schreiben Sie die gewünschten Zeichen. Nach dem Speichern wird die Anzeige auf dem LCD-Modul sofort aktualisiert. Achten Sie dabei darauf, die Textdatei nur in einem reinen Texteditor zu bearbeiten (nicht in Word) und mit ANSI-Codierung zu speichern. Der Editor aus Windows 10 schlägt als Standard immer die UTF-8-Codierung vor.

Auf dem Konfigurationsbildschirm von LCD Smartie finden Sie auf der Registerkarte *Misc* eine Zeile *$File()*. Diese zeigt die erste Zeile der angegebenen Textdatei auf dem LCD-Modul an.

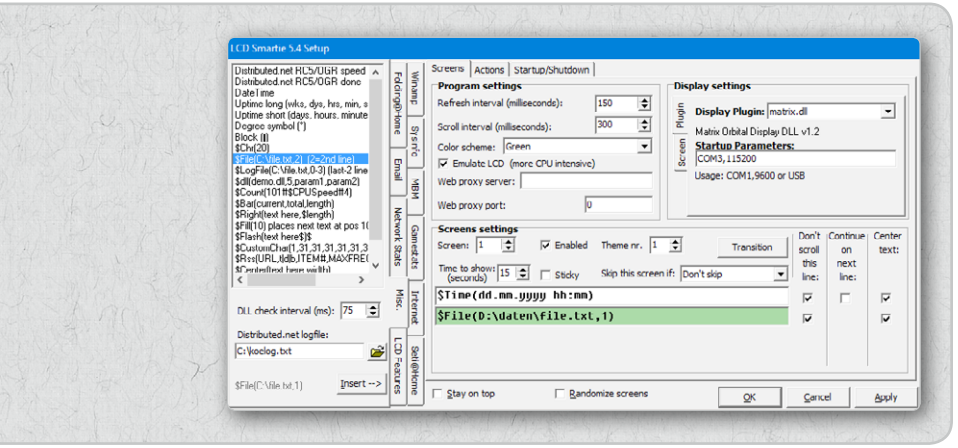


Abb. 4.10: *Textdatei in LCD Smartie anzeigen.*

Das auf dem PC simulierte LCD-Modul zeigt die Umlaute nicht richtig an, da die Umsetzung durch den Sketch auf dem Nano-Board erfolgt.

LCD Smartie mit Plug-ins erweitern

LCD Smartie kann über Plug-ins um zusätzliche Funktionalität erweitert werden. Auf der Webseite *lcdsmartie.sourceforge.net/smartied.htm* finden Sie zahlreiche solcher Plug-ins für verschiedene Zwecke.

Besonders interessant ist das Plug-in *BigNum*, das über einen eigens definierten Zeichensatz Zahlen darstellt, die höher sind als eine Zeile auf dem LCD-Modul.

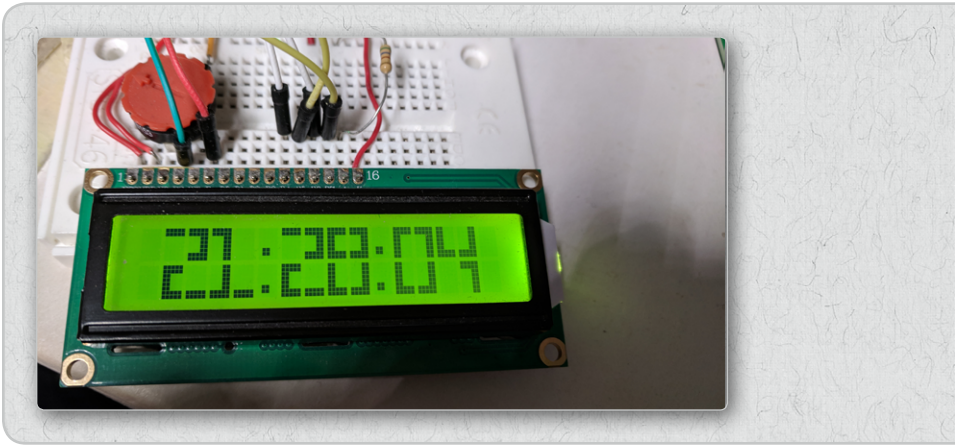


Abb. 4.11: *Uhr mit großen Ziffern.*

Beenden Sie LCD Smartie. Laden Sie sich dann das ZIP-Archiv mit dem Plug-in herunter und entpacken Sie die Datei `bignum.dll` in das Verzeichnis `plugins` der LCD-Smartie-Installation. Sichern Sie sich Ihre Konfigurationsdatei `config.ini` und entpacken Sie die `config-bignum.ini` als `config.ini` in das Hauptverzeichnis der LCD-Smartie-Installation. Starten Sie danach LCD Smartie neu und stellen Sie die Kommunikationsparameter sowie den Typ des LCD-Moduls neu ein.

Die Anzeige mit dem BigNum-Plug-in belegt immer zwei oder auf einem vierzeiligen LCD-Modul vier Zeilen. In beiden Zeilen wird der gleiche Text dargestellt, nur der Parameter für die Zeilennummer ist anders. Die Einstellung, ob der Text scrollen soll oder nicht, muss in beiden Zeilen gleich sein.

```
$dll(bignum,[fontsize],[linenum]#[font],[text])
```

Jede BigNum-Anweisung verwendet vier Parameter:

PARAMETER	BEDEUTUNG
`[fontsize]`	Schriftgröße: 1 = 2 Zeilen hoch, 1 Zeichen breit 2 = 2 Zeilen hoch, 2 Zeichen breit 3 = 4 Zeilen hoch, 3 Zeichen breit (für vierzeilige LCD-Module)
`[linenum]`	Zeilennummer: bei 1 von oben beginnend gezählt
`[font]`	Schriftart (Angabe optional): 0 = selbst definiert 1 = Standard (wird auch verwendet, wenn `[font]`-Parameter fehlt) 2 = fett 3 = Standard für LCD-Module mit durchgängiger unterer Pixelreihe 4 = fett für LCD-Module mit durchgängiger unterer Pixelreihe
`[text]`	Der Text kann nur Ziffern und wenige Sonderzeichen enthalten.

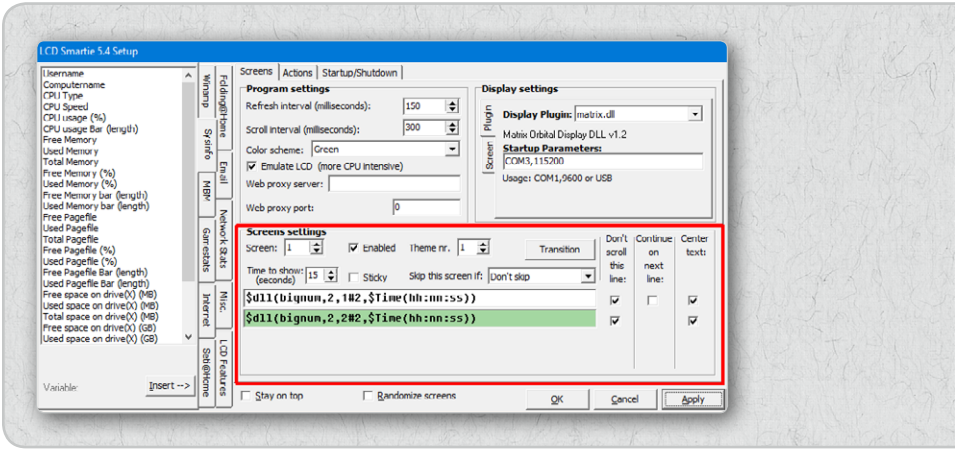


Abb. 4.12: *BigNum-Texte für die oben abgebildete Digitaluhr.*

Texte, die aus Buchstaben bestehen, werden automatisch nur auf einer Zeile dargestellt, da die großen Schriftarten nur Ziffern und wenige Sonderzeichen enthalten.

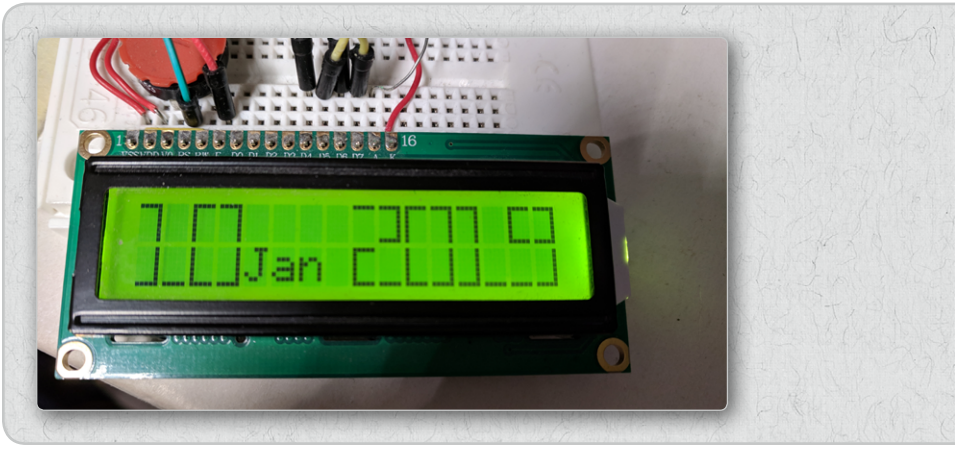


Abb. 4.13: *Kalender mit schlanker Schriftart und einzeiliger Monatsangabe.*

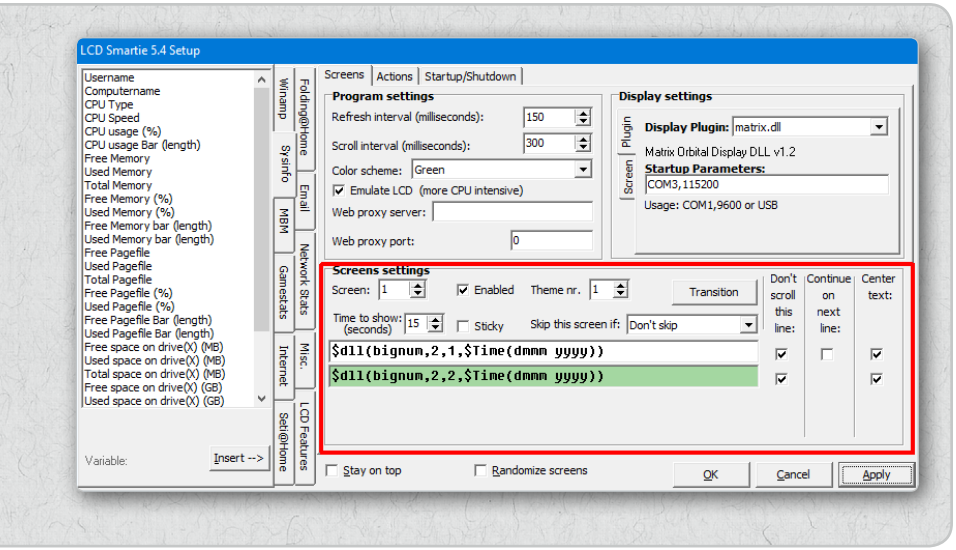


Abb. 4.14: *BigNum-Texte für den abgebildeten Kalender.*

Den Arduino interaktiv steuern

5

Der Nano kann auch dazu verwendet werden, interaktiv ohne Programm LEDs ein- oder auszuschalten oder den Zustand von digitalen und analogen Eingängen anzuzeigen.

Die Schaltung für das Experiment enthält sieben LEDs, die sich interaktiv einschalten lassen, einen Taster und einen Sensorkontakt an digitalen Eingängen sowie ein Potentiometer und einen Sensorkontakt an analogen Eingängen.

Benötigte Bauteile

- 1 x Nano
- 2 x Steckplatine
- 2 x LED rot (mit Vorwiderstand
- 1 x LED orange (mit Vorwiderstand)
- 1 x LED gelb (mit Vorwiderstand)
- 2 x LED grün (mit Vorwiderstand)
- 1 x LED blau (mit Vorwiderstand)
- 1 x Potentiometer
- 1 x Taster
- 1 x 10-kOhm-Widerstand (Braun-Schwarz-Orange)
- 2 x 20-MOhm-Widerstand (Rot-Schwarz-Blau)
- Draht blank
- Schaltdraht

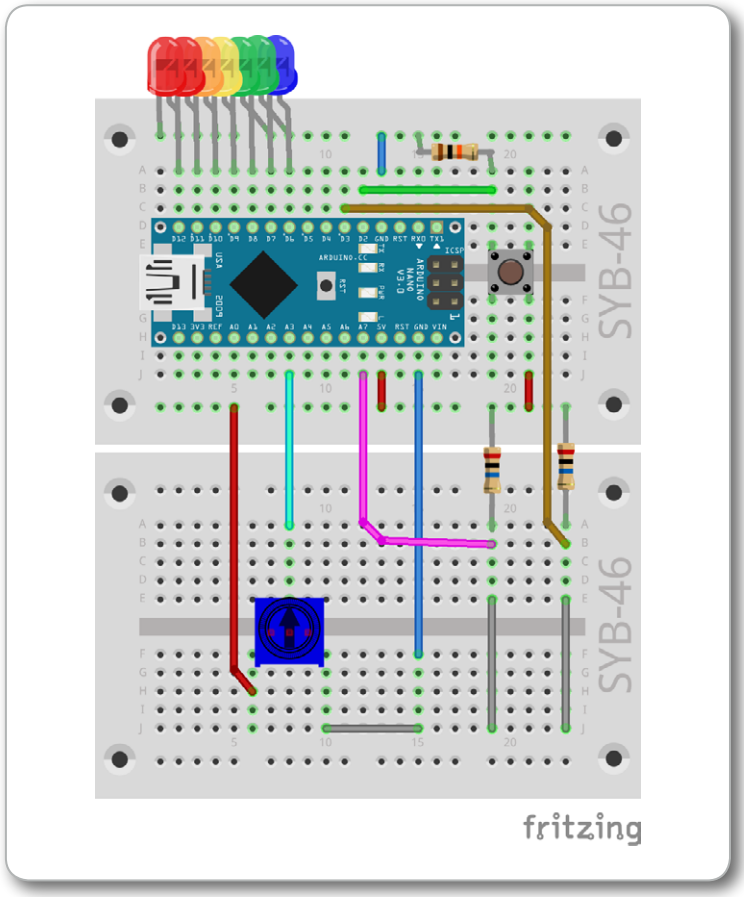


Abb. 5.1: *Testschaltung für die interaktive Steuerung des Nano-Boards.*

Die grauen Leitungen sind blanke Drähte für die beiden Sensorkontakte und den Massekontakt. Die horizontale graue Leitung dient gleichzeitig zur Verbindung des Potentiometers mit Masse.

Die Anschlüsse

Zur Verbindung übertragen Sie den Sketch *StandardFirmata* auf den Nano. Sie finden ihn in der Arduino-IDE unter *Datei/Beispiele/Firmata/StandardFirmata*. Firmata (*www.firmata.org*) ist ein standardisiertes Protokoll, das die interaktive Kommunikation mit dem Arduino über eine USB-Verbindung steuert. Verschiedene Programme nutzen Firmata, um Sensoren des Arduino interaktiv abzufragen oder angeschlossene Geräte vom PC zu schalten.

PINNUMMER	ANSCHLUSS
D2	Taster
D3	Sensorkontakt
D6	LED blau
D7	LED grün
D8	LED grün
D9	LED gelb
D10	LED orange
D11	LED rot
D12	LED rot
A3	Potentiometer
A7	Sensorkontakt

Windows Remote Arduino Experience

Microsoft liefert über den Microsoft Store für Windows 10 eine App *Windows Remote Arduino Experience* zur interaktiven Steuerung von Arduino-kompatiblen Platinen.

Wählen Sie als Erstes im Feld *Connection* die Option *USB*. Ein angeschlossener Arduino, auf dem StandardFirmata installiert ist, wird automatisch erkannt. Klicken Sie auf den Balken *Devices discovered* und danach auf *Connect*, um die Verbindung herzustellen.

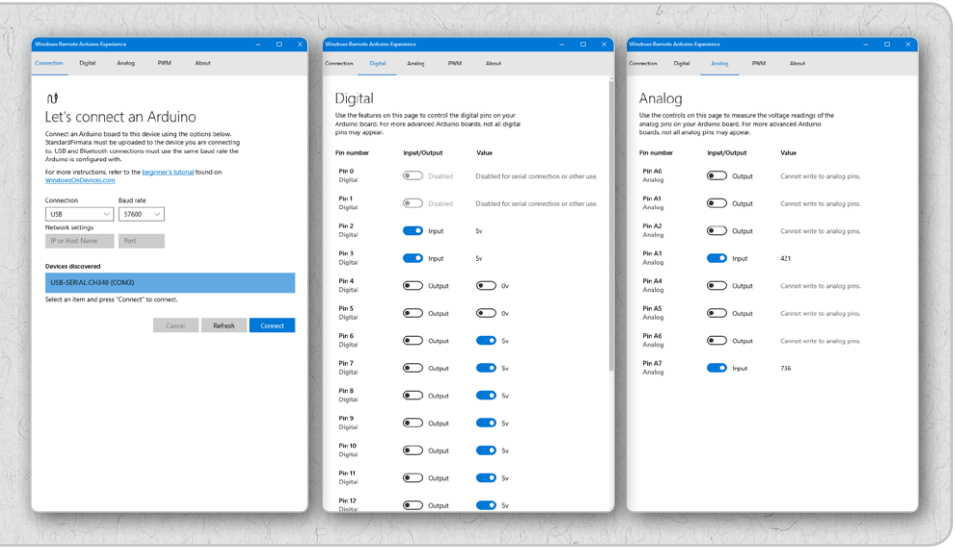


Abb. 5.2: *Interaktive Steuerung mit Windows Remote Arduino Experience.*

Auf der Seite *Digital* können Sie jeden digitalen Pin wahlweise als Eingang oder als Ausgang einrichten. Mit den Schaltern in der Spalte *Value* setzen Sie die Ausgänge auf 0 V (`LOW`) oder 5 V (`HIGH`) und schalten damit die angeschlossenen LEDs ein oder aus.

Setzen Sie die beiden Pins 2 und 3 auf *Input*. Pin 2 springt beim Drücken des Tasters auf 5 V (`HIGH`), Pin 3 mit dem Sensorkontakt steht auf 5 V (`HIGH`) und schaltet bei Berührung auf 0 V (`LOW`) um.

Schalten Sie auf der Seite *Analog* die Pins A3 und A7 auf *Input*. Jetzt sehen Sie dort analoge Werte. Den Wert des Pins A3 stellen

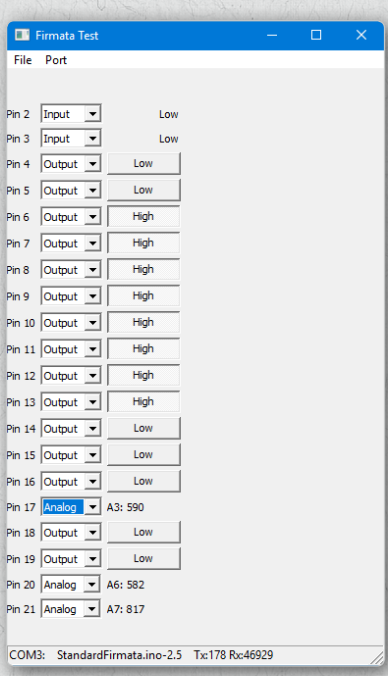
Sie mit dem Potentiometer ein. Der Wert des Pins A7 mit dem Sensorkontakt schwankt innerhalb eines bestimmten Bereichs zufällig, je nach Umgebungseinflüssen. Durch Berührung senken Sie den Wert deutlich, besonders wenn Sie gleichzeitig den Massekontakt berühren.

Diese App läuft auch auf Windows-10-Tablets, die nur einen Micro-USB-Anschluss haben. Die Verbindung zwischen Arduino und Tablet erfolgt dann mit einem USB-Hostadapterkabel, auch als OTG-Kabel bezeichnet. Das Tablet muss dazu den USB-Hostmodus unterstützen, was bei den meisten aktuellen Geräten der Fall ist.

Abb. 5.3: *USB-Hostadapterkabel.*

Firmata Test

Das Programm *Firmata Test* aus den Downloads zu diesem Maker Kit funktioniert prinzipiell gleich. Allerdings müssen Sie im Menü *Port* den verwendeten seriellen Port manuell auswählen, er wird nicht automatisch erkannt. Im Gegensatz zur Microsoft-App läuft dieses Programm auch auf Windows 7. Bei *firmata.org* finden Sie auch eine Linux-Version.



Auch hier können Sie die Pins einzeln auf Input oder Output schalten. Die analogen Pins sind nur etwas anders bezeichnet. Pin 17 im Programm entspricht Pin A3 auf dem Nano, Pin 21 entspricht Pin A7.

Viel Spaß bei weiteren Arduino-Projekten!