Komponenten / Erweiterungsplatinen / Module / Librarys

- DFPlayer Mini SKU Platine
- Display 16 Zeichen
- Display 16 Zeichen mit IC2 Adapter

DFPlayer Mini SKU Platine

Beschreibung

Eine Platine zum Anschluss von Lautsprecher und abspielen von MP3s.
Eine Komplette c Library zum steuern der Platine.

Spezifikation

- supported sampling rates (kHz): 8/11.025/12/16/22.05/24/32/44.1/48
- 24 -bit DAC output, support for dynamic range 90dB , SNR support 85dB
- fully supports FAT16 , FAT32 file system, maximum support 32G of the TF card, support 32G of U disk, 64M bytes NORFLASH
- a variety of control modes, I/O control mode, serial mode, AD button control mode
- advertising sound waiting function, the music can be suspended. when advertising is over in the music continue to play
- audio data sorted by folder, supports up to 100 folders, every folder can hold up to 255 songs
- 30 level adjustable volume, 6 -level EQ adjustable

Betriebsmodus

1. Serial Mode

Hier wird im Serial Baudmode ein hexstring übergeben in fogendem Format

Support for asynchronous serial communication mode via PC serial sending commands
Communication Standard:9600 bps Data bits :1 Checkout :none Flow Control :none

- Instruction Description

DFR0299 instructiao

Es gibt zwei Modi einmal sende und Empfangen.
Über senden werden Commandos wie Pla Pause etc. gesendet und über empfangen des Status.
Wie läuft die MP3 oder den text der Datei.

- Serial Control Cmd

CMD	Function Description	Parameters(16 bit)
0x01	Next	
0x02	Previous	
0x03	Specify tracking(NUM)	0-2999
0x04	Increase volume	
0x05	Decrease volume	
0x06	Specify volume	0-30
0x07	Specify EQ(0/1/2/3/4/5)	Normal/Pop/Rock/Jazz/Classic/Base
0x08	Specify playback mode (0/1/2/3)	Repeat/folder repeat/single repeat/ random
0x09	Specify playback source(0/1/2/3/4)	U/TF/AUX/SLEEP/FLASH
0x0A	Enter into standby – low power loss	
0x0B	Normal working	
0x0C	Reset module	
0x0D	Playback	
0x0E	Pause	
0x0F	Specify folder to playback	1~10(need to set by user)
0x10	Volume adjust set	{DH＝1:Open volume adjust }{DL: set volume gain 0~31}
0x11	Repeat play	{1:start repeat play}{0:stop play}

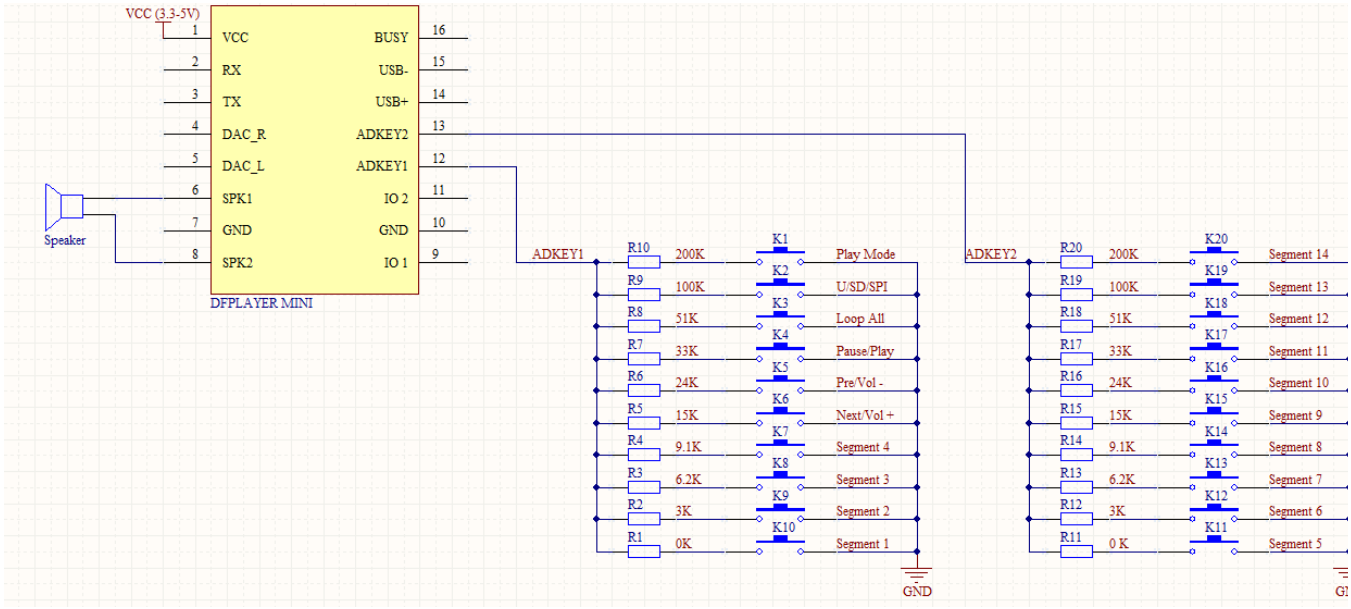
Serial Query Cmd

Commands	Function Description	Parameters(16 bit)
0x3C	STAY	
0x3D	STAY	
0x3E	STAY	
0x3F	Send initialization parameters	0 - 0x0F(each bit represent one device of the low-four bits)
0x40	Returns an error, request retransmission	
0x41	Reply	
0x42	Query the current status	
0x43	Query the current volume	
0x44	Query the current EQ	
0x45	Query the current playback mode	
0x46	Query the current software version	
0x47	Query the total number of TF card files	
0x48	Query the total number of U-disk files	
0x49	Query the total number of flash files	
0x4A	Keep on	
0x4B	Queries the current track of TF card	
0x4C	Queries the current track of U-Disk	
0x4D	Queries the current track of Flash	

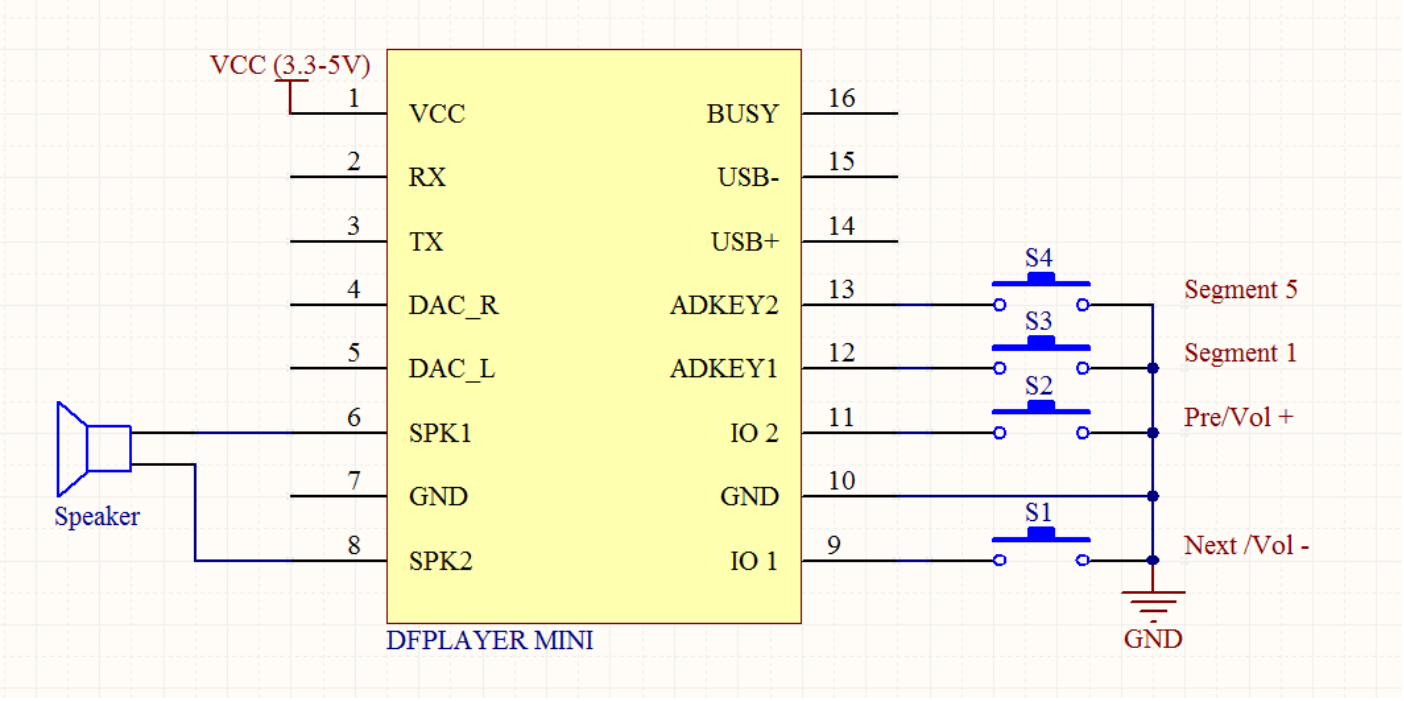
Modi 2, Taster mit Transistoren.

Es gibt auch die Möglichkeit ohne Software das Modul zu betreiben, dort werden die Schalter durch Widerstandswerte dargestellst.

- Refer diagram

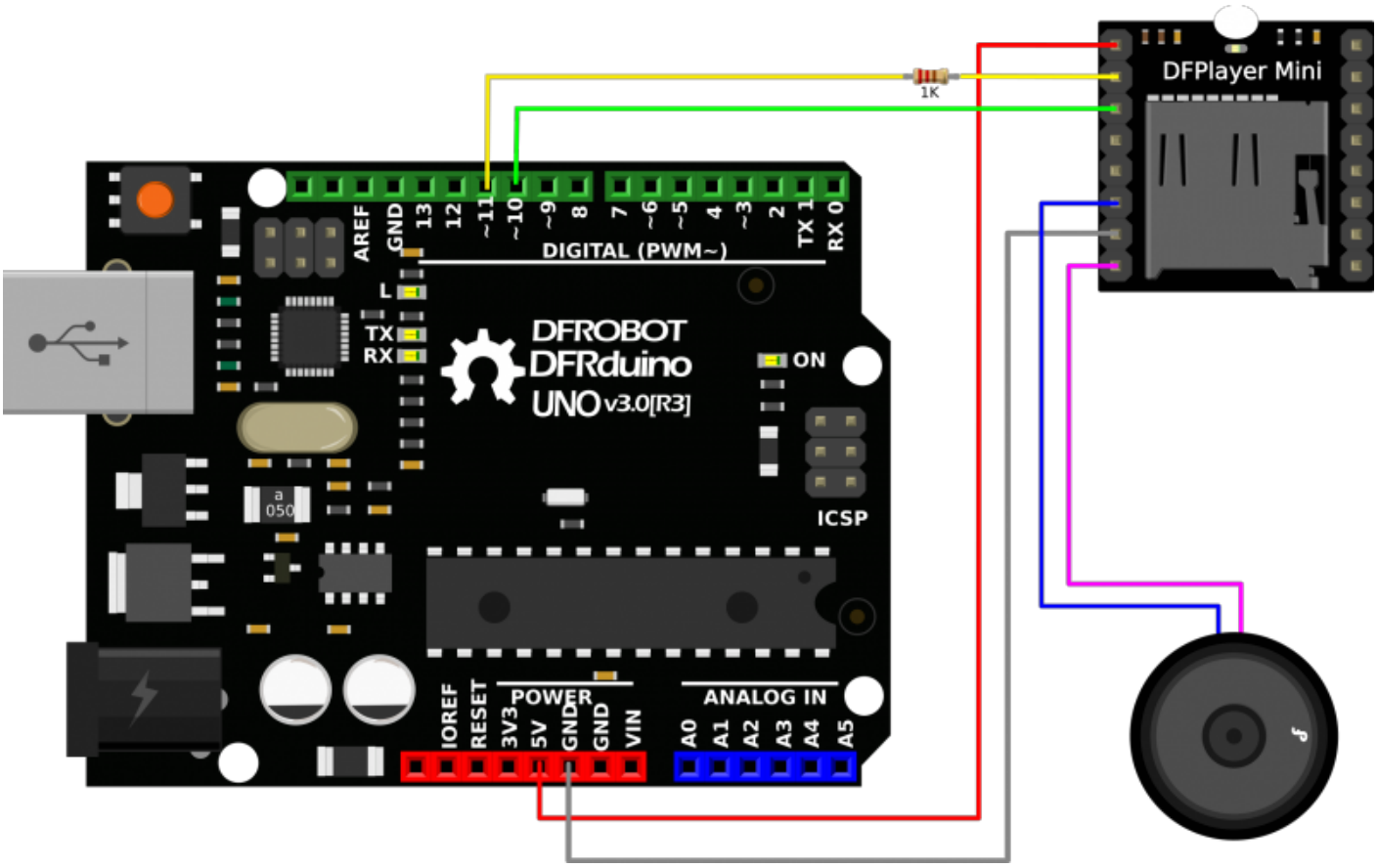


Modi 3. I/O Mode, der Simpleste Modus, ohne Transitoren und Software



Wenn kurz gedrückt wird ist forwärt / rückwärts, wird gedrückt geahlten ist Lauter / Leiser.

Arduinio Anschluss



Software Arduino

Die Software Libray kann unter https://github.com/DFRobot/DFRobotDFPlayerMini/ bezogen werden.
Zur Zeit der Erstllung des Artikels war das die Version von 2018, der Artikel wurde 2022 erstellt. Die Version 1.0.5 .
Damit ist es warscheinlich das es keine neuen Versionen geben wird, da alles funktioniert. Dieser Download ist auch dem Artikel hier angeheftet. Aber es schadet trotzdem nicht zu schauen obs eine neue Version gibt.

Benutzung

Archiv Entpacken und dem Projekt hinzufügen.
Dann noch SoftwareSerial hinzufügen.
Am einfachsten dur PIO-Home und dann library

Beispiel Code:

```
#include <Arduino.h>
#include "SoftwareSerial.h"
#include "DFRobotDFPlayerMini.h"



SoftwareSerial mySoftwareSerial(10, 11); // RX, TX
DFRobotDFPlayerMini myDFPlayer;
void printDetail(uint8_t type, int value);

void setup() {
  // put your setup code here, to run once:
  mySoftwareSerial.begin(9600);
  Serial.begin(115200);

  Serial.println();
  Serial.println(F("DFRobot DFPlayer Mini Demo"));
  Serial.println(F("Initializing DFPlayer ... (May take 3~5 seconds)"));

  if (!myDFPlayer.begin(mySoftwareSerial)) {  //Use softwareSerial to communicate with mp3.
    Serial.println(F("Unable to begin:"));
    Serial.println(F("1.Please recheck the connection!"));
    Serial.println(F("2.Please insert the SD card!"));
    while(true);
  }
  Serial.println(F("DFPlayer Mini online."));

  myDFPlayer.setTimeOut(500); //Set serial communictaion time out 500ms

  //----Set volume----
  myDFPlayer.volume(25);  //Set volume value (0~30).
  myDFPlayer.volumeUp(); //Volume Up
  myDFPlayer.volumeDown(); //Volume Down

  //----Set different EQ----
  myDFPlayer.EQ(DFPLAYER_EQ_NORMAL);
//  myDFPlayer.EQ(DFPLAYER_EQ_POP);
//  myDFPlayer.EQ(DFPLAYER_EQ_ROCK);
//  myDFPlayer.EQ(DFPLAYER_EQ_JAZZ);
//  myDFPlayer.EQ(DFPLAYER_EQ_CLASSIC);
//  myDFPlayer.EQ(DFPLAYER_EQ_BASS);
```

```
  //----Set device we use SD as default----
//  myDFPlayer.outputDevice(DFPLAYER_DEVICE_U_DISK);
  myDFPlayer.outputDevice(DFPLAYER_DEVICE_SD);
//  myDFPlayer.outputDevice(DFPLAYER_DEVICE_AUX);
//  myDFPlayer.outputDevice(DFPLAYER_DEVICE_SLEEP);
//  myDFPlayer.outputDevice(DFPLAYER_DEVICE_FLASH);

  //----Mp3 control----
//  myDFPlayer.sleep();     //sleep
//  myDFPlayer.reset();     //Reset the module
//  myDFPlayer.enableDAC();  //Enable On-chip DAC
//  myDFPlayer.disableDAC();  //Disable On-chip DAC
//  myDFPlayer.outputSetting(true, 15); //output setting, enable the output and set the gain to 15
  myDFPlayer.play(1);  //Play the first mp3
  /*----Mp3 play----
  myDFPlayer.next();  //Play next mp3
  delay(1000);
  myDFPlayer.previous();  //Play previous mp3
  delay(1000);
  myDFPlayer.play(1);  //Play the first mp3
  delay(1000);
  myDFPlayer.loop(1);  //Loop the first mp3
  delay(1000);
  myDFPlayer.pause();  //pause the mp3
  delay(1000);
  myDFPlayer.start();  //start the mp3 from the pause
  delay(1000);
  myDFPlayer.playFolder(15, 4);  //play specific mp3 in SD:/15/004.mp3; Folder Name(1~99); File Name(1~255)
  delay(1000);
  myDFPlayer.enableLoopAll(); //loop all mp3 files.
  delay(1000);
  myDFPlayer.disableLoopAll(); //stop loop all mp3 files.
  delay(1000);
  myDFPlayer.playMp3Folder(4); //play specific mp3 in SD:/MP3/0004.mp3; File Name(0~65535)
  delay(1000);
  myDFPlayer.advertise(3); //advertise specific mp3 in SD:/ADVERT/0003.mp3; File Name(0~65535)
  delay(1000);
  myDFPlayer.stopAdvertise(); //stop advertise
  delay(1000);
```

```
  myDFPlayer.playLargeFolder(2, 999); //play specific mp3 in SD:/02/004.mp3; Folder Name(1~10); File Name(1~1000)
  delay(1000);
  myDFPlayer.loopFolder(5); //loop all mp3 files in folder SD:/05.
  delay(1000);
  myDFPlayer.randomAll(); //Random play all the mp3.
  delay(1000);
  myDFPlayer.enableLoop(); //enable loop.
  delay(1000);
  myDFPlayer.disableLoop(); //disable loop.
  delay(1000);
  */

  //----Read imformation----
  Serial.println(myDFPlayer.readState()); //read mp3 state
  Serial.println(myDFPlayer.readVolume()); //read current volume
  Serial.println(myDFPlayer.readEQ()); //read EQ setting
  Serial.println(myDFPlayer.readFileCounts()); //read all file counts in SD card
  Serial.println(myDFPlayer.readCurrentFileNumber()); //read current play file number
  Serial.println(myDFPlayer.readFileCountsInFolder(3)); //read file counts in folder SD:/03
}

void loop() {
  // put your main code here, to run repeatedly:
   static unsigned long timer = millis();

  //if (millis() - timer > 3000) {
   // timer = millis();
    //myDFPlayer.next();  //Play next mp3 every 3 second.
  //}
 //  myDFPlayer.play(1);
   if (myDFPlayer.available()) {
    printDetail(myDFPlayer.readType(), myDFPlayer.read()); //Print the detail message from DFPlayer to handle different errors and states.
  }

}

void printDetail(uint8_t type, int value){
  switch (type) {
```

```
  case TimeOut:
    Serial.println(F("Time Out!"));
    break;
  case WrongStack:
    Serial.println(F("Stack Wrong!"));
    break;
  case DFPlayerCardInserted:
    Serial.println(F("Card Inserted!"));
    break;
  case DFPlayerCardRemoved:
    Serial.println(F("Card Removed!"));
    break;
  case DFPlayerCardOnline:
    Serial.println(F("Card Online!"));
    break;
  case DFPlayerUSBInserted:
    Serial.println("USB Inserted!");
    break;
  case DFPlayerUSBRemoved:
    Serial.println("USB Removed!");
    break;
  case DFPlayerPlayFinished:
    Serial.print(F("Number:"));
    Serial.print(value);
    Serial.println(F(" Play Finished!"));
    break;
  case DFPlayerError:
    Serial.print(F("DFPlayerError:"));
    switch (value) {
      case Busy:
        Serial.println(F("Card not found"));
        break;
      case Sleeping:
        Serial.println(F("Sleeping"));
        break;
      case SerialWrongStack:
        Serial.println(F("Get Wrong Stack"));
        break;
      case CheckSumNotMatch:
        Serial.println(F("Check Sum Not Match"));
```

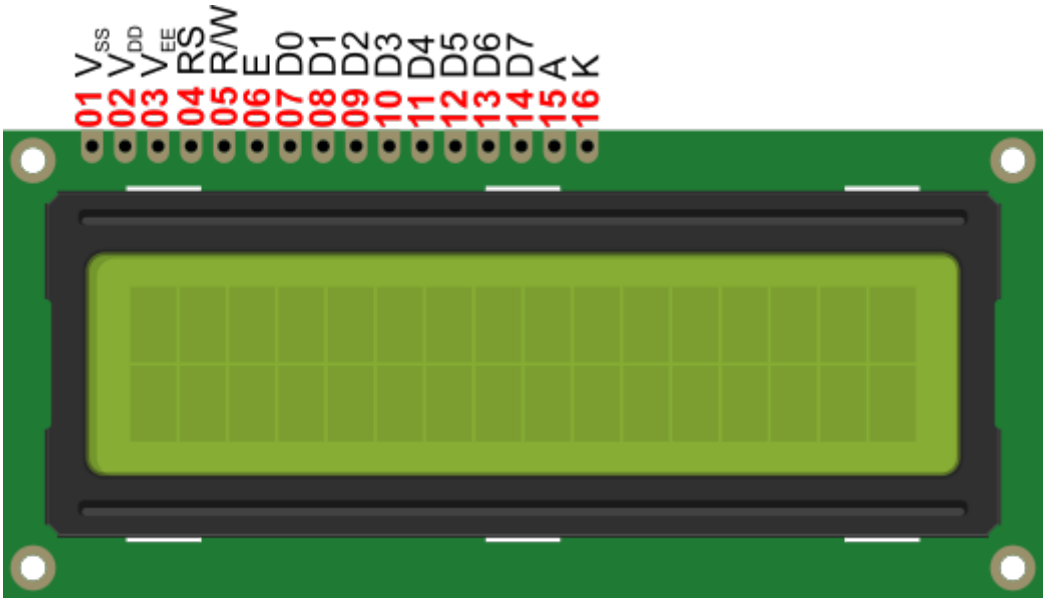
```
        break;
      case FileIndexOut:
        Serial.println(F("File Index Out of Bound"));
        break;
      case FileMismatch:
        Serial.println(F("Cannot Find File"));
        break;
      case Advertise:
        Serial.println(F("In Advertise"));
        break;
      default:
        break;
    }
    break;
  default:
    break;
 }

}
```

Display 16 Zeichen

Beschreibung

Ein Stellen mit zwei Zeilen Display.

Anschluss



Es werden folgende PINS angeschlossen:

- **Pin 1 (V_{SS})** und **Pin 2 (V_{DD})** dienen der Stromversorgung des Displays und der Ansteuerungselektronik. Pin 1 ist dabei auf Masse zu legen, auf Pin 2 sind +5 V Versorgungsspannung zuzuführen.
- **Pin 3 (V_{EE})** ist ein analoger Eingang und dient der Kontrastregelung des Displays. Der Wert muss zwischen 0 V und +5 V liegen.
- **Pin 4 (RS)** ist ein digitaler Eingang und bestimmt, ob die zum Display übermittelten Datenbits als Befehl (LOW) oder Zeichendaten (HIGH) interpretiert werden sollen.
- **Pin 5 (R/W)** ist ein digitaler Eingang, der entscheidet, ob Daten auf dem Display geschrieben (LOW) oder vom Display eingelesen (HIGH) werden sollen. Es ist also tatsächlich möglich, den Inhalt des Displays wieder mit dem Arduino einzulesen. In der Praxis ist das aber eigentlich nie erforderlich. Daher legt man diesen Pin einfach dauerhaft auf Masse (LOW).
- **Pin 6 (E)** ist ein digitaler Eingang, der auf HIGH geschaltet werden muss, damit das Display die an den Datenpins anliegenden Bits ausliest.
- **Pin 7 – Pin 14 (D0 – D7)** sind die 8 Bits des bidirektionalen, paralellen Datenbusses. Da man ungern ganze 8 Ports des Arduinos nur für die Datenübertragung zum Display verbrauchen möchte, nutzt man die Fähigkeit der Ansteuerungselektronik, in den 4-Bit-

Modus zu schalten. In diesem Fall werden nur die hinteren Pins 11 – 14 (D4 – D7) mit dem Arduino verbunden und die 8 Bit in zwei Schritten (jeweils 4 Bit) nacheinander übertragen. Die Pins 7 – 10 lässt man einfach offen.
- **Pin 15 (A)** und **Pin 16 (K)** existieren nur an LCD mit eingebauter Hintergrundbeleuchtung und dienen der Stromversorgung selbiger. An Pin 15 (Anode) kommt die Versorgungsspannung, Pin 16 (Kathode) wird auf Masse gelegt. Je nach LCD muss hier entweder ein Vorwiderstand für die im LCD verbaute LED vorgeschaltet werden oder aber der entsprechende Widerstand befindet sich bereits im LCD. Wenn man sich unsicher ist und kein Hinweis darauf beim LCD zu finden ist, kann man vorsichtshalber einen 220 Ω-Widerstand einbauen.

LCD-Ansteuerung mit analoger Kontrastregelung (Poti)

- Universal-LCD mit Parallelbus (14 oder 16 Pins)
- (Widerstand 220 Ω)
- Trimmpotentiometer 10 kΩ
- Jumperkabel (18×)
- Beim Poti ist Anschluss 1 immer Masse und Anschluss 2 V+ und die Mitte das eigentliche zu Regelnde Gerät LED etc.

Steckbrettansicht

Die relevaten Pins des LCD werden mit dem Arduino verbunden. In die Spannungszuführung der Hintergrundbeleuchtung (so denn überhaupt vorhanden) wurde vorsichtshalber der oben erwähnte Vorwiderstand eingesetzt. Der analoge Eingang der Kontrastregelung (V_{EE}) wird mit dem Schleifkontakt eines Trimmpotentiometers verbunden, welches auf der einen Seite mit +5 V, auf der anderen Seite mit Masse (0 V) verbunden wird. Hierüber lässt sich der Kontrast manuell regeln.

Zur Ansteuerung des LCD wird die LiquidCrystal-Bibliothek von Adafruit genutzt. Der Beispielcode gibt einen Standardtext aus und zählt anschließend die Sekunden seit dem Start des Programms hoch.

LCD-Ansteuerung mit PWM-Kontrastregelung

- Universal-LCD mit Parallelbus (14 oder 16 Pins)
- (Widerstand 220 Ω)
- Jumperkabel (14×)

Steckbrettansicht

Im Normalfall stellt man den Kontrast des LCD einmalig ein und belässt ihn dann in dieser Einstellung. Damit ist das Trimmpotentiometer eigentlich überflüssig und nimmt nur Platz auf dem Breadboard weg. Einen passenden Widerstand (mit festem Wert) zu finden, kann sich aber unter

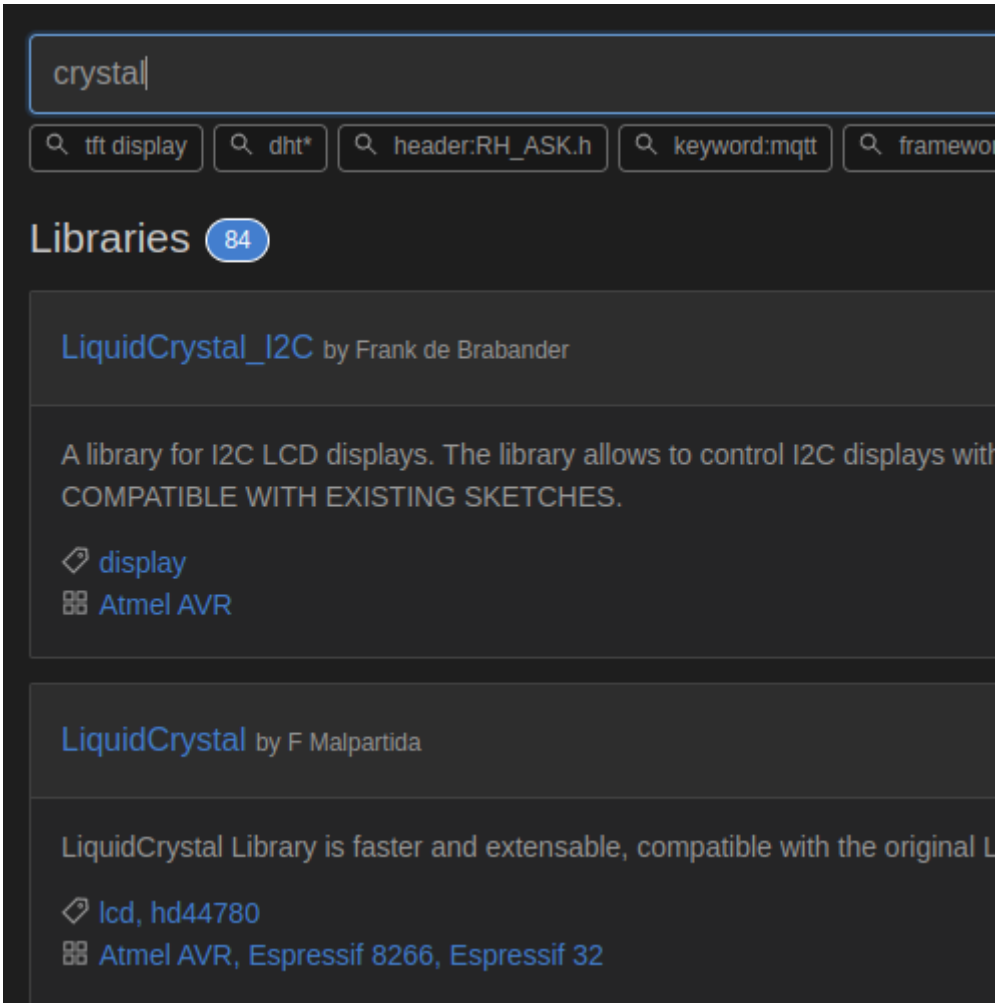
Umständen als schwierig erweisen. Eine Alternative stellt die Kontrastregelung über einen PWM-Ausgang des Arduinos dar. Dazu wird auch der Pin 3 (V_{EE}) an den Arduino angeschlossen und das Trimmpotentiometer kann entfallen. Dafür verliert man natürlich wiederum einen digitalen Ausgang. Man muss von Schaltung zu Schaltung abwägen, was einem lieber ist.

Im Gegensatz zum obigen Beispiel wird bei der Initialisierung mittels der Funktion analogWrite() per PWM der Kontrast des LCD auf einen festen Wert eingestellt. Der optimale Wert muss von Ihnen einmalig auf Ihr LCD angepasst werden.

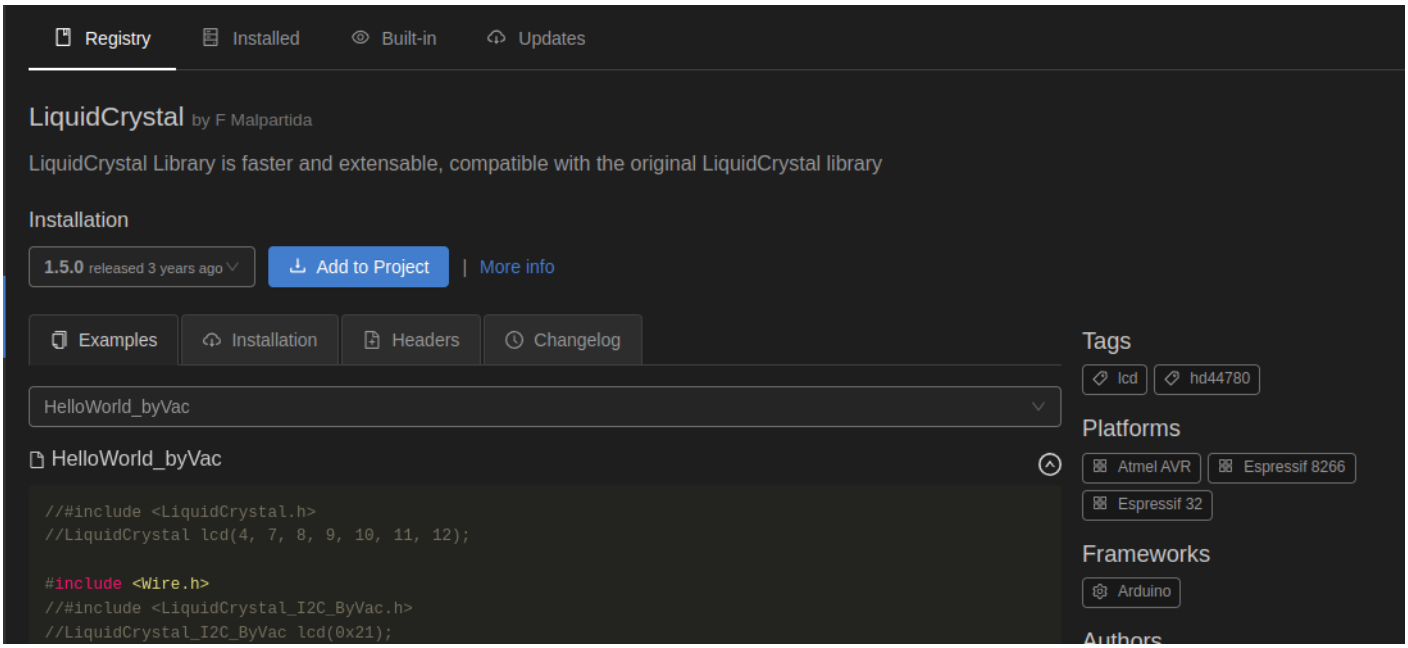
Einbindung 16 Zeichen Display

Bibliotheken installieren

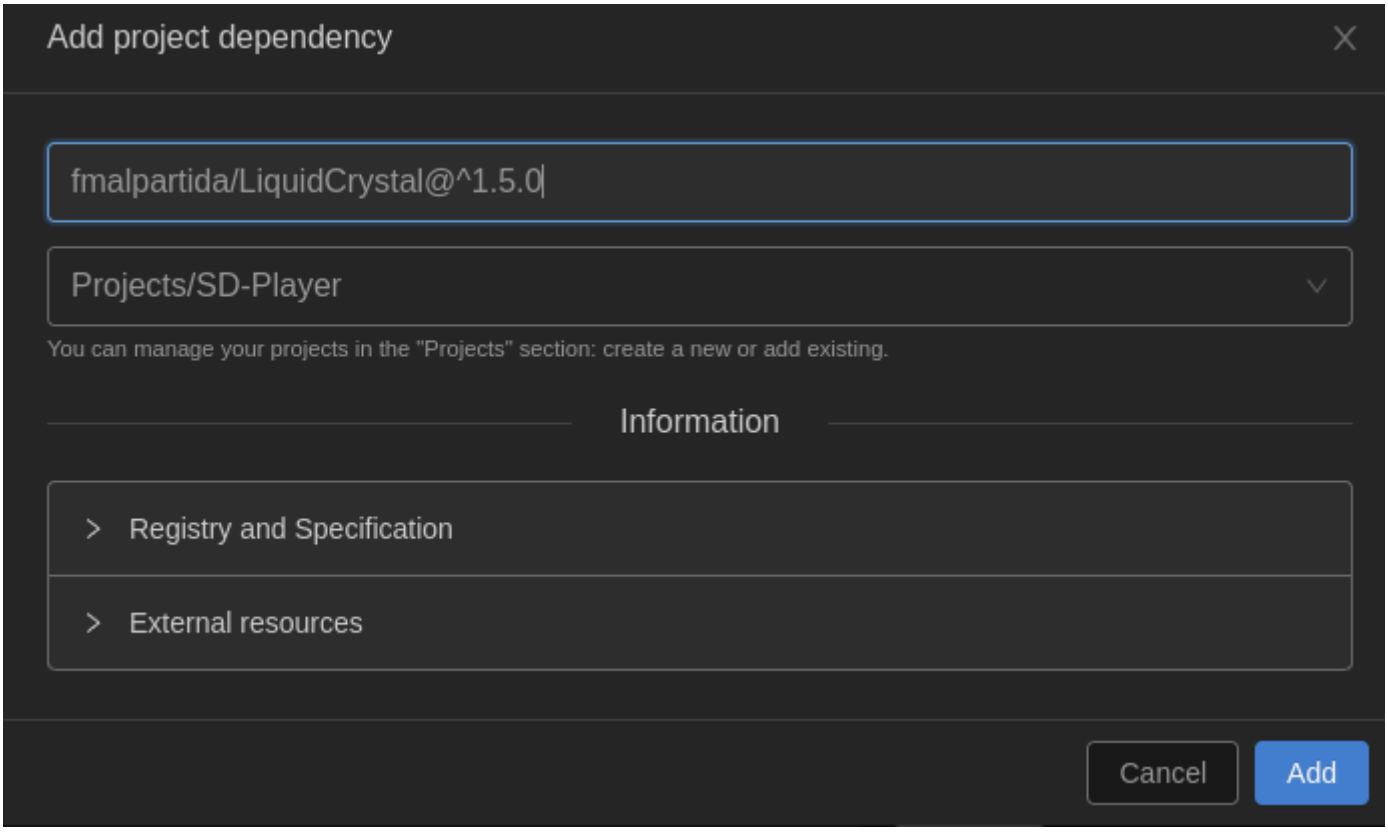
Wir benötigen dazu die LiquidCrystal Library (Diese im PIO-Home Library herunterladen)
Dazu den Suchbegriff crystal eingeben



Dann auf Add to Project klicken



Und nun das Project auswählen



Code

Code Beispiel Direktanschluss 4 Datenpins ohne IC2 BUS:

```
#include <LiquidCrystal.h>
#include <Wire.h>
```

```
#include <SoftwareSerial.h>

#define PIN_LCD_VEE_VO 6                    // Pin für LCD-Pin Vee/Vo (Kontrastwert)
#define PIN_LCD_RS 13                       // Pin für LCD-Pin RS  (Register Select)
#define PIN_LCD_E 12                        // Pin für LCD-Pin E   (Enable)
#define PIN_LCD_D4 2                        // Pin für LCD-Pin D4  (Datenbit 4)
#define PIN_LCD_D5 3                        // Pin für LCD-Pin D5  (Datenbit 5)
#define PIN_LCD_D6 4                        // Pin für LCD-Pin D6  (Datenbit 6)
#define PIN_LCD_D7 5                        // Pin für LCD-Pin D7  (Datenbit 7)

#define LCD_CONTRAST 10                     // Kontrastwert (muss experimentell an das LCD angepasst werden) 0
ganz Dunkel 100 Total Hell, auch nix mehr zu erkennen)
#define LCD_ROWS 2                          // Anzahl der Zeilen des Displays.
#define LCD_COLS 16                         // Anzahl der Spalten des Displays.

LiquidCrystal lcd(PIN_LCD_RS, PIN_LCD_E, PIN_LCD_D4, PIN_LCD_D5, PIN_LCD_D6, PIN_LCD_D7);
int aSeconds = 0; //Als Zähler für die Sekunden, wird fürs Display nichtebenötigt, aber wie bauen hier im Beispiel
einen Zähler
void setup()
{
 analogWrite(PIN_LCD_VEE_VO, LCD_CONTRAST);  // Regele den Kontrast des Display per PWM auf den Wert
LCD_CONTRAST. Wenn ein Poti benützt wird überflüssig
   lcd.begin(LCD_COLS, LCD_ROWS);           // Die Größe des Displays festlegen und das Display intialisieren.
   lcd.setCursor(0, 0);                     // Springe mit dem Cursor in der 1. Zeile an Position 1.
lcd.setCursor(Position,Zeile)
   lcd.print("Sekunden seit");              // Schreibe ab dort den Text "Sekunden seit".
   lcd.setCursor(0, 1);                     // Springe mit dem Cursor in der 2. Zeile an Position 1.
lcd.setCursor(Position,Zeile)
   lcd.print("Start:");                     // Schreibe ab dort den Text "Start:".
}

void loop()
{
 lcd.setCursor(7, 1);                     // Springe mit dem Cursor in der 2. Zeile an Position 8.
   lcd.print(aSeconds);                     // Schreibe ab dort den aktuellen Wert der Variable aSeconds.
   aSeconds++;                              // Erhöhe den Wert der Variablen aSeconds um 1.

   delay(1000);                             // Warte eine Sekunde.
}
```

Quelle

https://rotering-net.de/tut/arduino/lcd-ansteuern.html

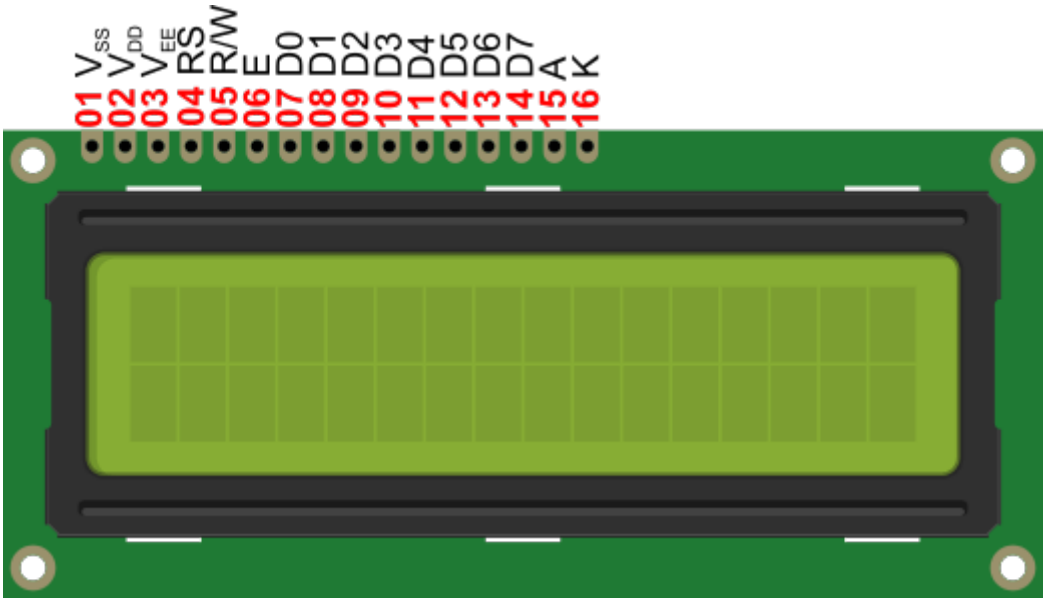
Display 16 Zeichen mit IC2 Adapter

Beschreibung

Das I2C Schnittstellen Modul vereinfacht die Ansteuerung der LCD Displays 1602 und 2004 erheblich.
Normal brauchst du zur Ansteuerung der Displays 6 I/O vom Arduino und eine Leitung, um den Kontrast einzustellen.
Dazu kommt noch die 5V Speisung und GND (Insgesamt also 9 Verbindungen).

Mit dem I2C Schnittstellen Modul brauchst du nur 2 Steuerleitungen, 5V Speisung und GND (Insgesamt nur 4 Verbindungen). Das LCD Schnittstellen Modul wird über den I2C Bus anhand der eingestellten Adresse angesprochen. Standardmässig ist das Modul auf die Adresse 0x27 eingestellt. Die Adresse kann durch löten von Brücken bei A0, A1 und A2

Anschlussbelegung Display



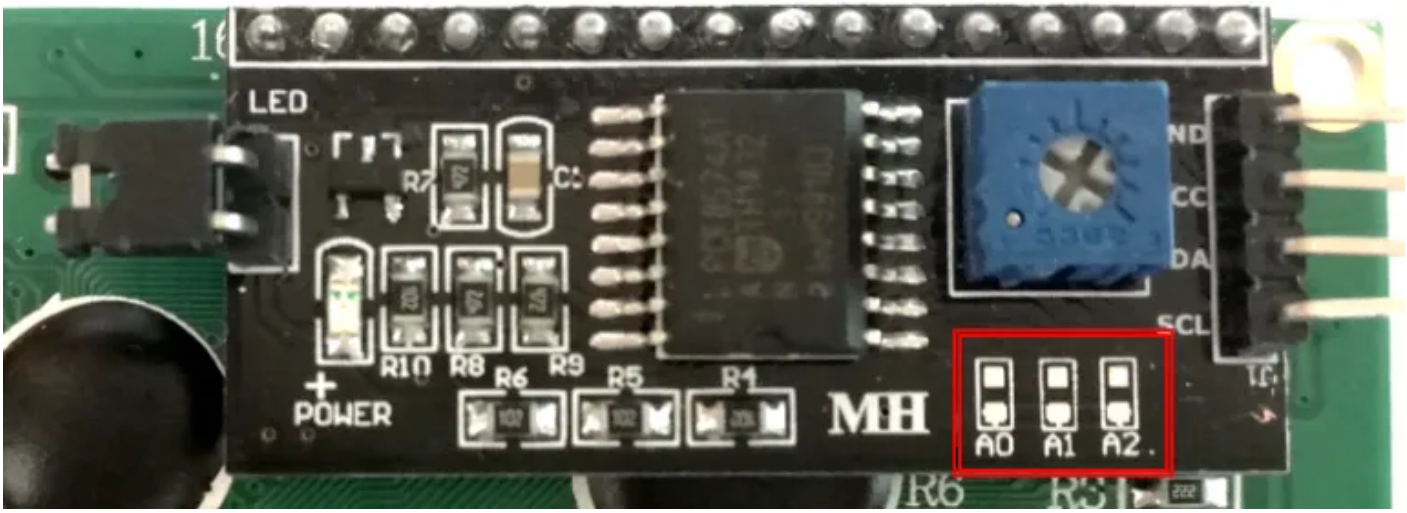
Es werden folgende PINS angeschlossen:

- **Pin 1 (V_{SS})** und **Pin 2 (V_{DD})** dienen der Stromversorgung des Displays und der Ansteuerungselektronik. Pin 1 ist dabei auf Masse zu legen, auf Pin 2 sind +5 V Versorgungsspannung zuzuführen.

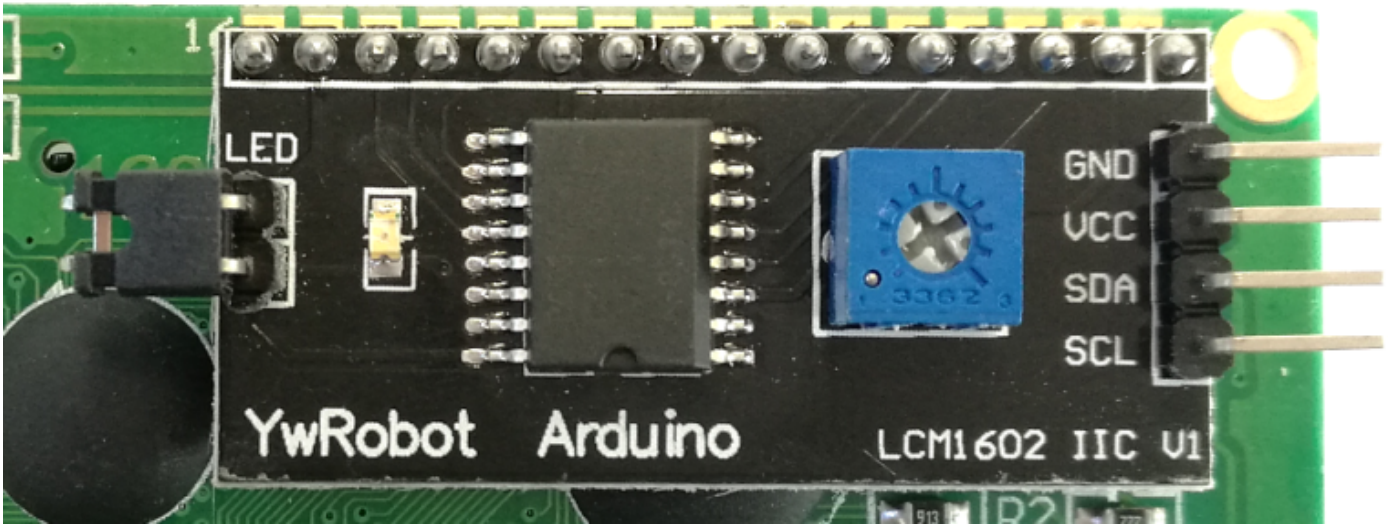
- **Pin 3 (V_{EE})** ist ein analoger Eingang und dient der Kontrastregelung des Displays. Der Wert muss zwischen 0 V und +5 V liegen.
- **Pin 4 (RS)** ist ein digitaler Eingang und bestimmt, ob die zum Display übermittelten Datenbits als Befehl (LOW) oder Zeichendaten (HIGH) interpretiert werden sollen.
- **Pin 5 (R/W)** ist ein digitaler Eingang, der entscheidet, ob Daten auf dem Display geschrieben (LOW) oder vom Display eingelesen (HIGH) werden sollen. Es ist also tatsächlich möglich, den Inhalt des Displays wieder mit dem Arduino einzulesen. In der Praxis ist das aber eigentlich nie erforderlich. Daher legt man diesen Pin einfach dauerhaft auf Masse (LOW).
- **Pin 6 (E)** ist ein digitaler Eingang, der auf HIGH geschaltet werden muss, damit das Display die an den Datenpins anliegenden Bits ausliest.
- **Pin 7 – Pin 14 (D0 – D7)** sind die 8 Bits des bidirektionalen, paralellen Datenbusses. Da man ungern ganze 8 Ports des Arduinos nur für die Datenübertragung zum Display verbrauchen möchte, nutzt man die Fähigkeit der Ansteuerungselektronik, in den 4-Bit-Modus zu schalten. In diesem Fall werden nur die hinteren Pins 11 – 14 (D4 – D7) mit dem Arduino verbunden und die 8 Bit in zwei Schritten (jeweils 4 Bit) nacheinander übertragen. Die Pins 7 – 10 lässt man einfach offen.
- **Pin 15 (A)** und **Pin 16 (K)** existieren nur an LCD mit eingebauter Hintergrundbeleuchtung und dienen der Stromversorgung selbiger. An Pin 15 (Anode) kommt die Versorgungsspannung, Pin 16 (Kathode) wird auf Masse gelegt. Je nach LCD muss hier entweder ein Vorwiderstand für die im LCD verbaute LED vorgeschaltet werden oder aber der entsprechende Widerstand befindet sich bereits im LCD. Wenn man sich unsicher ist und kein Hinweis darauf beim LCD zu finden ist, kann man vorsichtshalber einen 220 Ω-Widerstand einbauen.

Anschluss IC2 Modul an LCD DIsplay

Es gibt zwei Varianten von IC2 LCD Modulen. Einmal mit Lötstellen A0 -A3. Standardmäßig ist hier meist auch 0x27 eingestellt
Über die Lötstellen kann die Adresse eingestellt werden. Bei beiden Varianten ist oben rechts GND für PIN1 vom LCD Display.
Dann die Kontakte einfach 1:1 verbinden.



und einmal ohne, da ist die Adresse fest meist 0x27



I2C Schnittstelle => Arduino (A4 und A5 sind die Seriellen Datenleitungen. GND und 5V dienen der Spannungsversorgung des Moduls

GND => GND

VCC => 5V

SDA => A4

SCL => A5

Ändern der Adresse

Jedes I²C Modul hat eine sogenannte „HEX Adresse“. Über diese Adresse reagiert das I²C-Modul auf die Daten, die vom Arduino auf dem Datenbus an genau diese Adresse gesendet werden. Viele I²C-LCDs haben auch die gleiche HEX-Adresse. Das bedeutet, dass beim Verwenden von zwei Displays beide Displays auf die gesendeten Daten vom Arduino-Board reagieren würden. Man könnte also auf zwei Displays keine unterschiedlichen Daten darstellen.

Die HEX-Adresse kann bei dem Display mit Hilfe der A0, A1 und A2 Lötstellen jedoch verändert werden. Im unveränderten Zustand sind alle drei Lötstellen nicht verbunden. Je nach Kombination, welche der Stellen man mit einer Lötstelle überbrückt, sind also 8 verschiedene Adressen möglich. Abhängig vom Display Typ kann diese Adresse anfangs 0x27 oder 0x3F sein (kann mit dem Adressen „Scanner“ herausgefunden werden, dazu später mehr).

Der Kontrast des LCD Displays kann über das Blaue Potentiometer auf dem Modul eingestellt werden. Die LCD Hintergrundbeleuchtung kann durch entfernen des Jumpers «LED» deaktiviert werden. Für die Ansteuerung gibt es bereits eine fertige Arduino Library, die du im Library Manager unter «LiquidCrystal_I2C» findest.

Tabellen zu HEX Adressen je nach verlöteten Stellen(**I** = verbunden, **:** = nicht verbunden):

A0	A1	A2	HEX Adresse	HEX Adresse
:	:	:	0x27	0x3F
I	:	:	0x26	0x3E
:	I	:	0x25	0x3D
I	I	:	0x24	0x3C
:	:	I	0x23	0x3B
I	:	I	0x22	0x3A
:	I	I	0x21	0x39
I	I	I	0x20	0x38

IC2 Hex Scanner

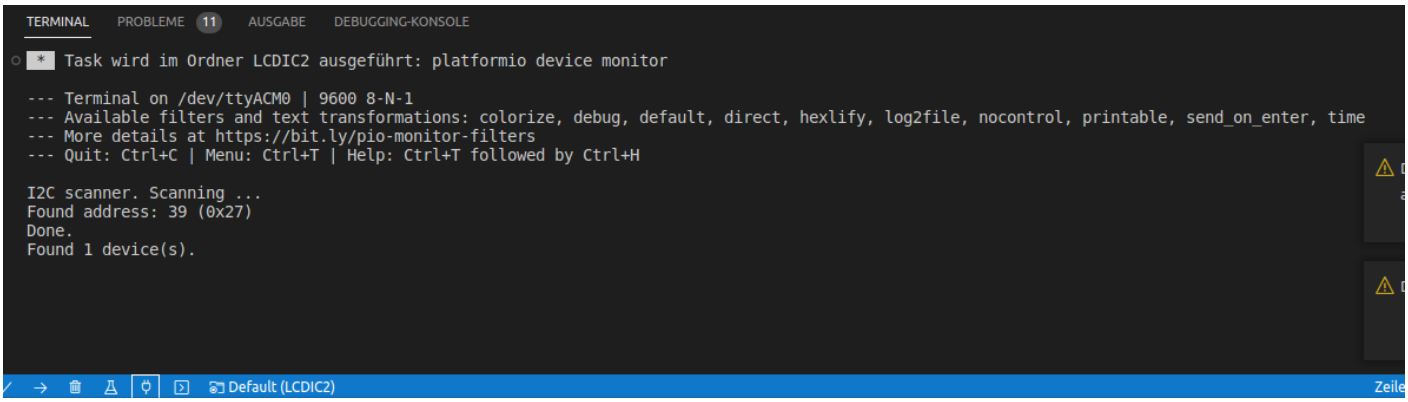
```
// I2C Scanner
// Written by Nick Gammon
// Date: 20th April 2011
#include <Arduino.h>
#include <Wire.h>
void setup() {
Serial.begin (9600);
// Leonardo: wait for serial port to connect
while (!Serial)
{
}
Serial.println ();
Serial.println ("I2C scanner. Scanning ...");
byte count = 0;
```

```
Wire.begin();
for (byte i = 8; i < 120; i++)
{
Wire.beginTransmission (i);
if (Wire.endTransmission () == 0)
{
Serial.print ("Found address: ");
Serial.print (i, DEC);
Serial.print (" (0x");
Serial.print (i, HEX);
Serial.println (")");
count++;
delay (1); // maybe unneeded?
} // end of good response
} // end of for loop
Serial.println ("Done.");
Serial.print ("Found ");
Serial.print (count, DEC);
Serial.println (" device(s).");
} // end of setup
void loop() {}
```

In der Platform.ini folgendes hinzufügen

```
...
; Custom Serial Monitor port
monitor_port = COM4

; Custom Serial Monitor speed (baud rate)
monitor_speed = 9600
...
```

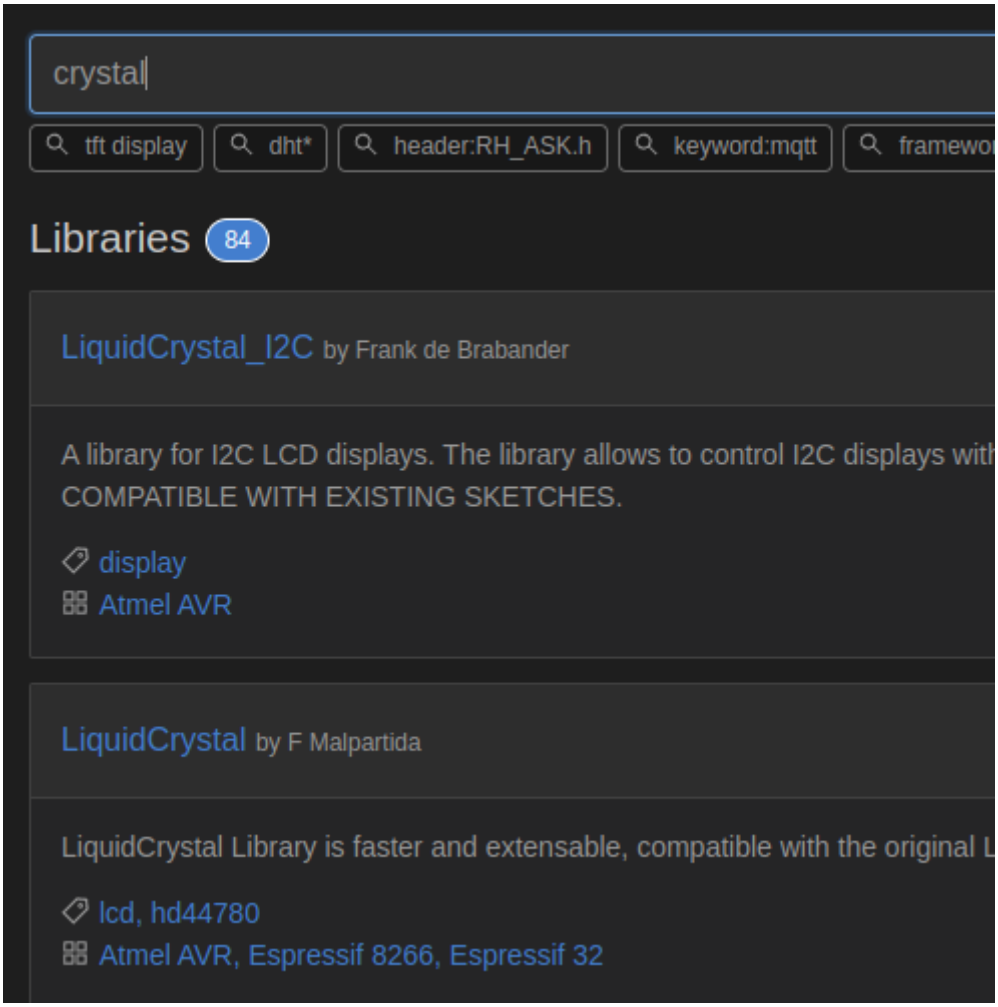
Nun unten in der Leiste auf den Stecker klicken. Dort bekommen wir nun die Serialausgabe. In diesem Fall wissen wir nun das unser Display den Hex Wert 0x27 hat. Siehe Ausgabe. Dank des Hexscanners.



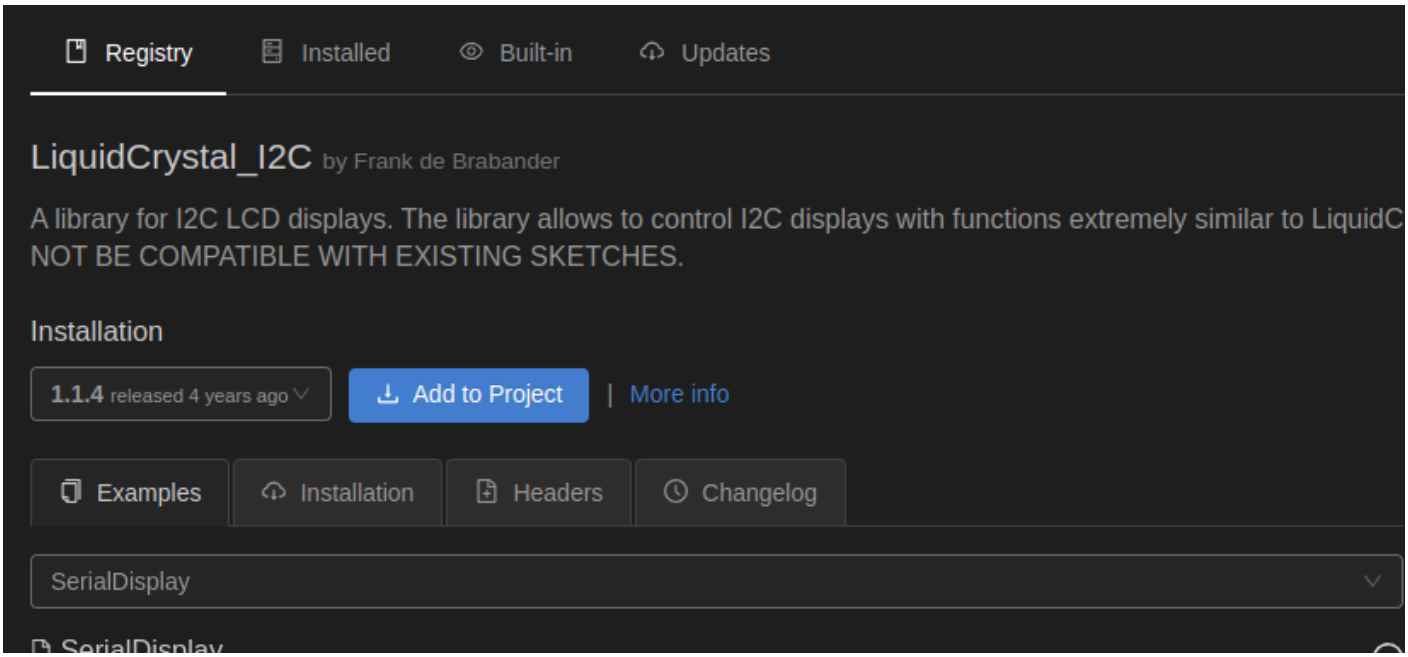
Einbindung 16 Zeichen Display

Bibliotheken installieren

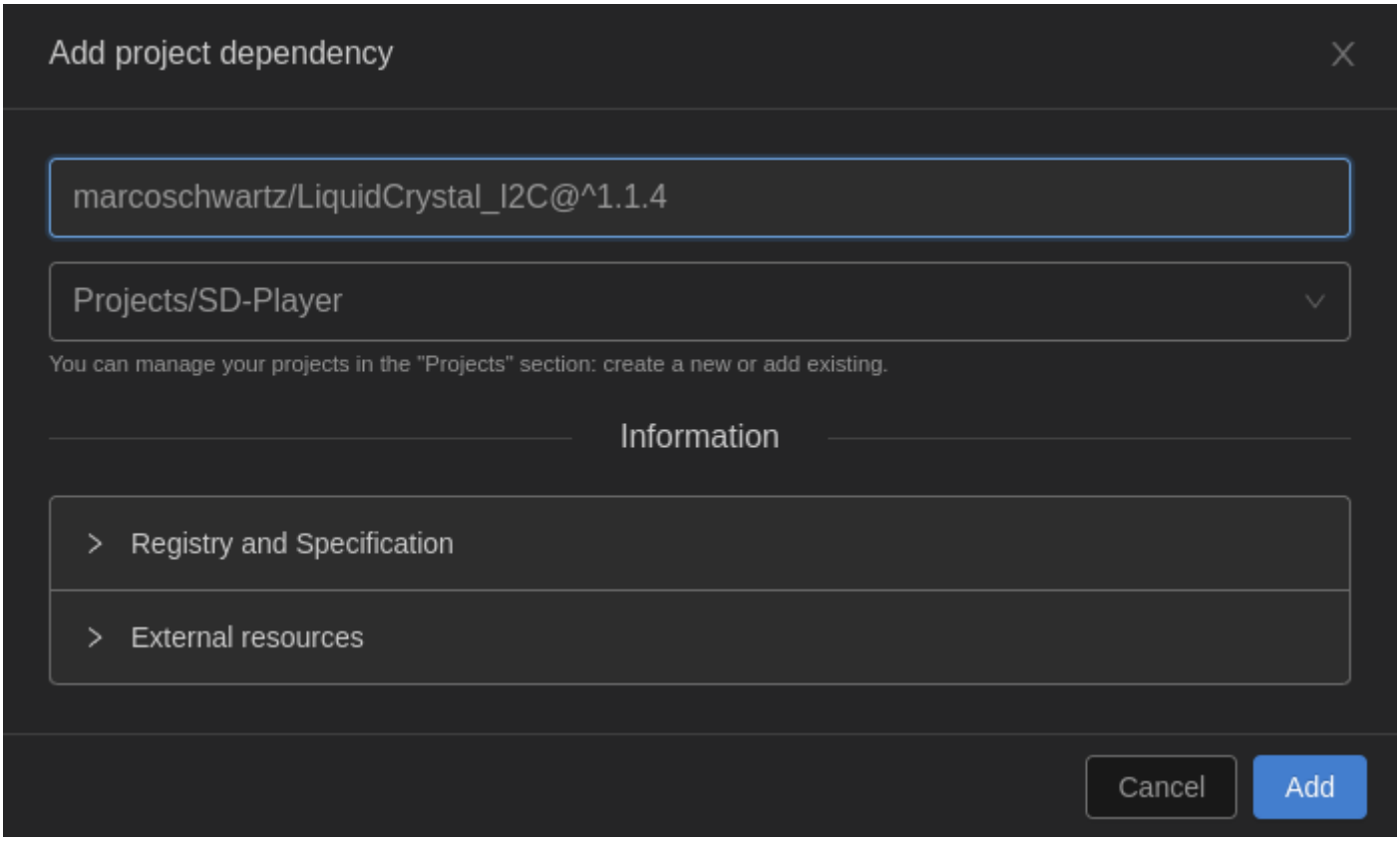
Wir benötigen dazu die LiquidCrystal Library IC2 (Diese im PIO-Home Library herunterladen)
Dazu den Suchbegriff crystal eingeben



Dann auf Add to Project klicken



Und nun das Project auswählen



Code

```
#include <Arduino.h>
#include <Wire.h> // Wire Bibliothek einbinden
#include <LiquidCrystal_I2C.h> // Vorher hinzugefügte LiquidCrystal_I2C Bibliothek einbinden
LiquidCrystal_I2C lcd(0x27, 16, 2); //Hier wird festgelegt um was für einen Display es sich handelt. In diesem Fall eines mit 16 Zeichen in 2 Zeilen und der HEX-Adresse 0x27. Für ein vierzeiliges I2C-LCD verwendet man den
```

```
Code "LiquidCrystal_I2C lcd(0x27, 20, 4)"

void setup()
{
lcd.init(); //Im Setup wird der LCD gestartet
lcd.backlight(); //Hintergrundbeleuchtung einschalten (lcd.noBacklight(); schaltet die Beleuchtung aus).
}

void loop()
{
lcd.setCursor(0, 0);//Hier wird die Position des ersten Zeichens festgelegt. In diesem Fall bedeutet (0,0) das erste Zeichen in der ersten Zeile.
lcd.print("Hacker-Net Telekommuniktion");
lcd.setCursor(0, 1);// In diesem Fall bedeutet (0,1) das erste Zeichen in der zweiten Zeile.
lcd.print("Viel Erfolg!");
}
```

Quelle

https://funduino.de/nr-19-i%C2%B2c-display

https://www.bastelgarage.ch/i2c-schnittstelle-pcf8574-fur-lcd-display

https://funduino.de/nr-06-zwei-i%C2%B2c-displays-gleichzeitig