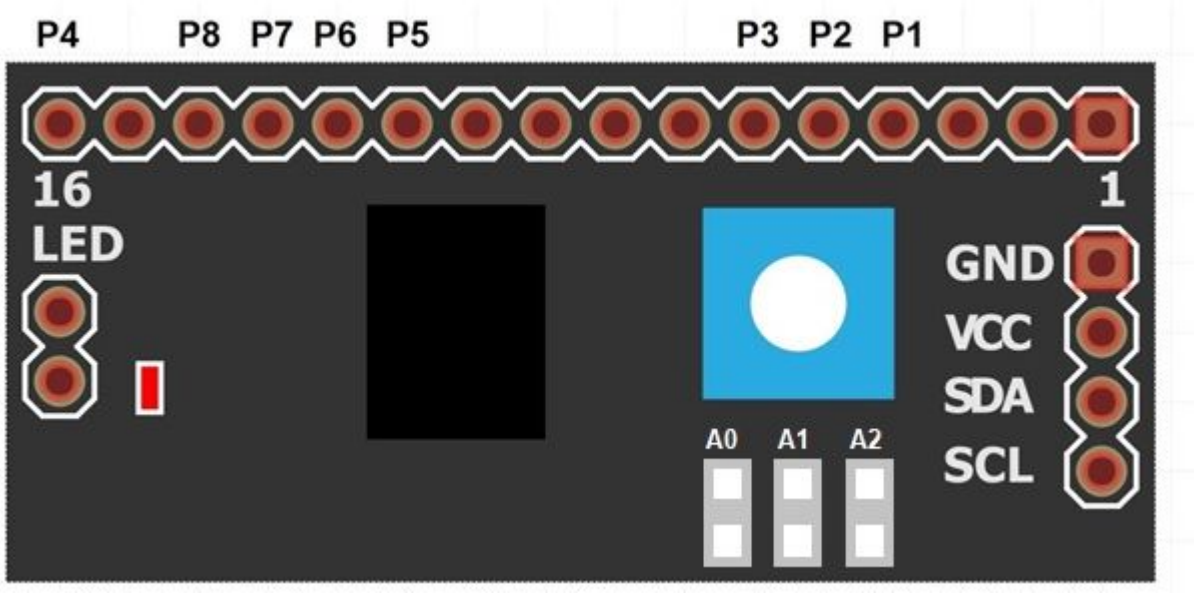
Ansteuerung eines 16 Relais Modules mit dem I2C Bus (Teil 1)

Hallo und willkommen zu einer neuen dreiteiligen Reihe rund um die Erstellung nützlicher Hardware Funktionen und Bibliotheken für eigene Projekte. Wir werden uns in dieser Kurzprojektreihe nicht mit Anleitungen zu fertigen Anwendungen beschäftigen, sondern anhand des Beispiels für die Steuerung unseres 16 fach Relais Modus erklären, wie hardwarenah programmiert werden kann und wie eigene Bibliotheken erstellt werden können. Im dritten Teil der Reihe zeige ich dann wie aus der ersten eigenen Bibliothek eine Bibliothek zum Download auf GitHub wird. Doch zuerst mal zu der Problemstellung und den Basics. Das in unserem Online-Shop angebotene 16-fach Relais Modul besitzt zur Ansteuerung 16 LOW aktive TTL-kompatible Eingänge. Wenn diese direkt an einen Arduino angeschlossen werden und getrenntmit ei angesteuert werden, bleiben für weitere Peripherie nicht mehr sehr viele freie Ports übrig. Ein Ausweg aus dieser Problematik bietet der IC2 Port Expander Baustein PC8574, der pro Baustein 8 parallel Ausgänge für den I2C Bus zur Verfügung stellt. Ein Einsatzgebiet dieses Bausteins ist zum Beispiel der I2C Adapter als Schnittstelle für LCD Displays. Mit zwei von diesen Bausteinen (2x8 bit) kann das 16 Fach Relais Modul angesteuert werden. Wir brauchen für unser heutiges Projekt daher folgende Hardware:

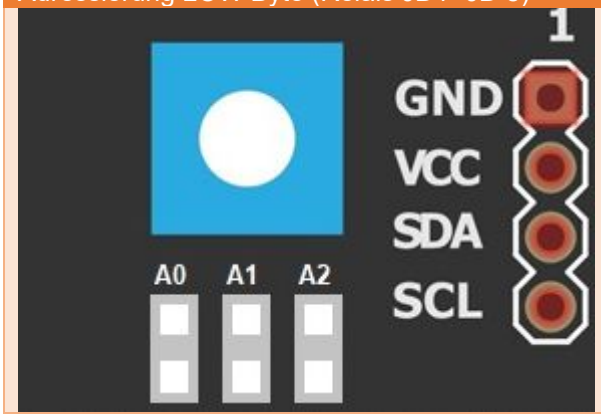
Anzahl	Bezeichnung	Anmerkung
1	Arduino Nano	
2	I2C Adapter als Schnittstelle für LCD Displays.	
1	16-fach Relais Modul	

Damit der I2C Adapter für LCD Displays zur Ansteuerung anderer Hardware genutzt werden kann, muss als nächster Schritt die Belegung mit einem Logiktester getestet werden. Dies habe ich bereits mit folgendem Ergebnis durchgeführt:

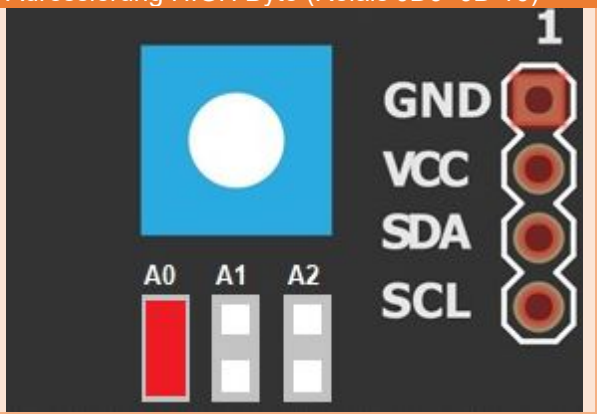


Zum einen fällt auf, dass die Belegung nicht homogen folgend ist, sondern über das Board verteilt ist. Bei Port 4 fällt auf, dass dieser ganz links auf dem Board angeordnet ist und die LED-Hintergrundbeleuchtung bei LCD's steuert. Daraus ergibt sich weiterhin die Besonderheit, das dieser Port invertiert mit einer logischen NICHT Funktion ausgegeben wird. Dieser Umstand ist wichtig für die Ansteuerung des Relais Moduls, da diese Tatsache Einfluss zum einen auf die Verkabelung und auf die Programmierung hat. Da 16 Relais gesteuert werden sollen und wir daher zwei I2C Adapter an einem IC2 Bus benötigen, muss bei dem zweiten Board die IC2 Adresse durch eine Brücke auf dem A0 Pin geändert werden. (Hier in der Abbildung beide Port Module nebeneinandergestellt)

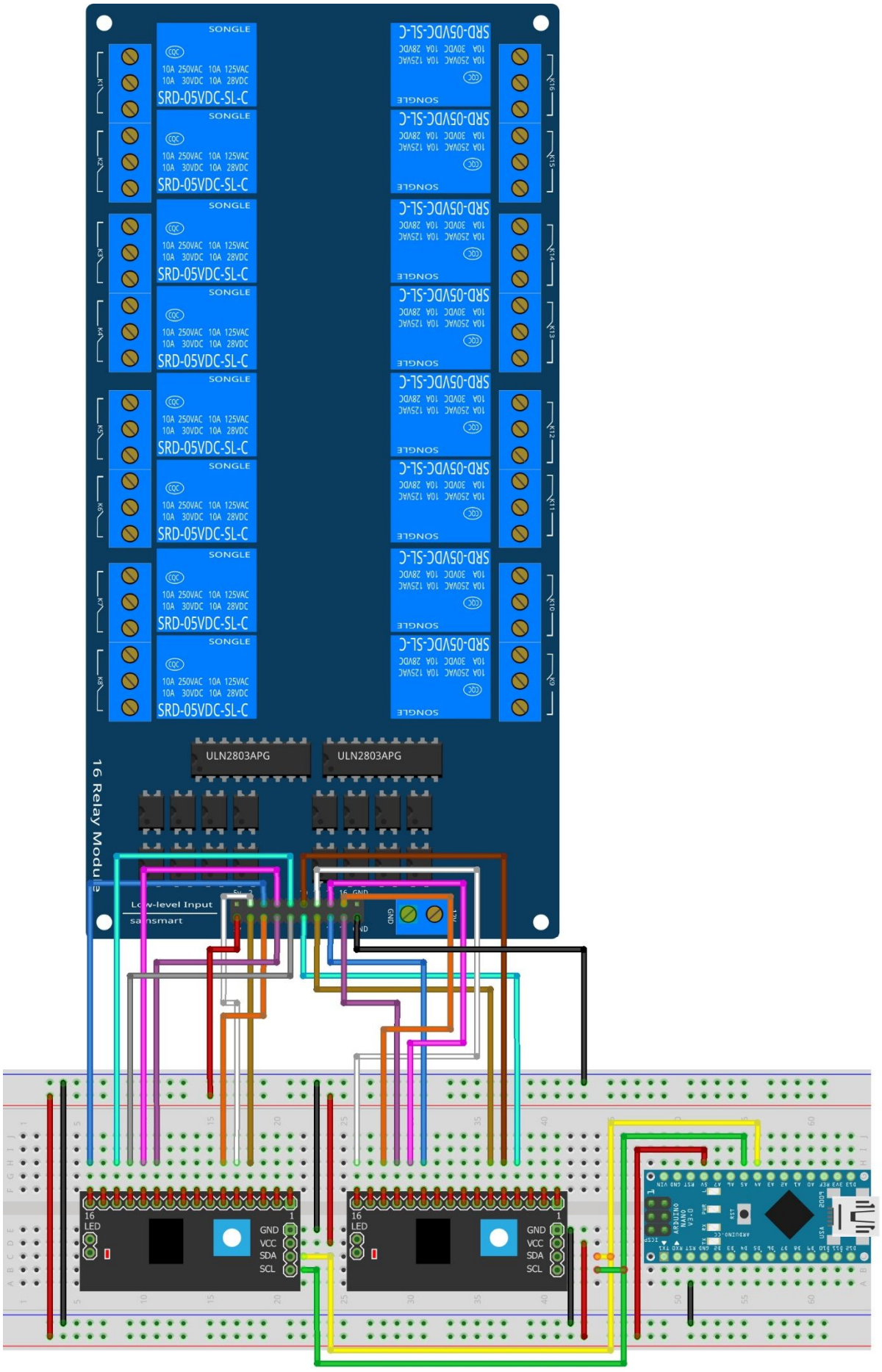
Adressierung LOW Byte (Relais JD1- JD 8)



Adressierung HIGH Byte (Relais JD9- JD 16)



Nachdem den beiden Port Module Ihre I2CAdressen (Standard: 38 und 37) zugewiesen wurden, können diese wie folgt verdrahtet werden:


SONGLE
10A 250VAC 10A 125VAC
10A 30VDC 10A 28VDC
SRD-05VDC-SL-C
ULN2803APG
ULN2803APG
16 Relay Modu
Low-level Input
sainsmart
16
LED
GND
VCC
SDA
SCL
1
fritzing

Nun kann folgendes Programm auf den Arduino hochgeladen werden:

```
/*
 * 16 Port Relay Module Control via two PC8574 LCD Port Expander Module(s)
 * Simple Function to Control 16 Relais Via I2C to save Ports.
 * Notes to Adressing the PC8574 Modules:
 * First  8 Relais (JD1 - JD8) are controlled via a Module without alternate Adresses - (leave A0-A2
open)
 * Second 8 Relais (JD9 - JD16) are controlled via a Module Baseaddress Adress -1  - (A0 is closed)
 *
 * Tobias Kuch 2020 - tobias.kuch@googlemail.com
 * For free use in your own Projects
 *
 * Licensed under GPL 3.0
 *
 */
#include <Wire.h>

#define JD1 0
#define JD2 1
#define JD3 2
#define JD4 3
#define JD5 4
#define JD6 5
#define JD7 6
#define JD8 7

#define JD9 8
#define JD10 9
#define JD11 10
#define JD12 11
#define JD13 12
#define JD14 13
#define JD15 14
#define JD16 15


#define PC8574_LCD_Port_Expander_Base_Addess 39 //PC8574 LCD Port Expander 2 Addess is 38

volatile byte valPort_A = 0;
volatile byte valPort_B = 0;

void setup()
{
  Wire.begin(); // join i2c bus
}

bool SetOutputRelaisStatus(byte RelaisPort, bool Status)
{
// Do some bit-wise Vodoo - Calculations for PC8574 LCD Port Expander  ;)
byte SendCommand = 0;
```

```
byte BitMask = 0;
byte Return = 0;
if (RelaisPort < 8)
 {
 BitMask = ( BitMask | (1  << RelaisPort));
 if (Status)
  {
  valPort_A = valPort_A | BitMask;
  } else
  {
  BitMask = ~BitMask;
  valPort_A = valPort_A & BitMask;
  }
 SendCommand = valPort_A ^ B00001000;                    // Invert Bit 4, leave others untouched
 SendCommand = ~SendCommand;                             // Invert Whole Control Byte für use with
16 Relais Module (Low active)
 Wire.beginTransmission(PC8574_LCD_Port_Expander_Base_Addess);   // transmit to first PC8574
device (Normally at Address #39 DEC)
                                        // device address is specified in datasheet
 Wire.write(SendCommand);                                // sends value byte
 Return = Wire.endTransmission(true);                    // stop transmitting
 if (Return == 0)
  {
  return true;
  } else
  {
  return false;
  }
  }
else if (RelaisPort < 16)                               // High Port Selected
 {
 RelaisPort = RelaisPort - 8;
 BitMask = ( BitMask | (1  << RelaisPort));
 if (Status)
  {
  valPort_B = valPort_B | BitMask;
  } else
  {
  BitMask = ~BitMask;
  valPort_B = valPort_B & BitMask;
  }
 SendCommand = valPort_B ^ B00001000;                    // Invert Bit 4, leave others untouched
 SendCommand = ~SendCommand;                             // Invert Whole Control Byte für use with
16 Relais Module (Low active)
 Wire.beginTransmission(PC8574_LCD_Port_Expander_Base_Addess - 1); // transmit to first
PC8574 device (Normally at Address #39 DEC)
                                        // device address is specified in datasheet
 Wire.write(SendCommand);                                // sends value byte
 Return = Wire.endTransmission(true);                    // stop transmitting
 if (Return == 0)
  {
  return true;
  } else
```

```
   {
   return false;
   }
  }
else
  {
  return false;                                  // Out of Rage
  }
}

void loop()
{
  Serial.begin(9600);
  if (!SetOutputRelaisStatus(JD1, LOW))
  {
    Serial.println("Unsuccessful Port Write to Port JD1");
  }
  SetOutputRelaisStatus(JD2, LOW);
  SetOutputRelaisStatus(JD3, LOW);
  SetOutputRelaisStatus(JD4, LOW);
  SetOutputRelaisStatus(JD5, LOW);
  SetOutputRelaisStatus(JD6, LOW);
  SetOutputRelaisStatus(JD7, LOW);
  SetOutputRelaisStatus(JD8, LOW);
    if (!SetOutputRelaisStatus(JD9, LOW))
  {
    Serial.println("Unsuccessful Port Write to Port JD9");
  }
  SetOutputRelaisStatus(JD10, LOW);
  SetOutputRelaisStatus(JD11, LOW);
  SetOutputRelaisStatus(JD12, LOW);
  SetOutputRelaisStatus(JD13, LOW);
  SetOutputRelaisStatus(JD14, LOW);
  SetOutputRelaisStatus(JD15, LOW);
  SetOutputRelaisStatus(JD16, LOW);
  if (!SetOutputRelaisStatus(16, LOW))
  {
    Serial.println("Unsuccessful Port Write to nonexisting Port"); //
  }

  delay(5000);
  SetOutputRelaisStatus(JD1, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD1, HIGH);
  delay(1000);
  SetOutputRelaisStatus(JD2, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD2, HIGH);
  delay(1000);
  SetOutputRelaisStatus(JD3, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD3, HIGH);
  delay(1000);
```

```
  SetOutputRelaisStatus(JD4, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD4, HIGH);
  delay(1000);
  SetOutputRelaisStatus(JD5, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD5, HIGH);
  delay(1000);
  SetOutputRelaisStatus(JD6, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD6, HIGH);
  delay(1000);
  SetOutputRelaisStatus(JD7, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD7, HIGH);
  delay(1000);
  SetOutputRelaisStatus(JD8, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD8, HIGH);
  delay(1000);
  SetOutputRelaisStatus(JD9, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD9, HIGH);
  delay(1000);
  SetOutputRelaisStatus(JD10, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD10, HIGH);
  delay(1000);
  SetOutputRelaisStatus(JD11, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD11, HIGH);
  delay(1000);
  SetOutputRelaisStatus(JD12, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD12, HIGH);
  delay(1000);
  SetOutputRelaisStatus(JD13, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD13, HIGH);
  delay(1000);
  SetOutputRelaisStatus(JD14, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD14, HIGH);
  delay(1000);
  SetOutputRelaisStatus(JD15, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD15, HIGH);
  delay(1000);
   SetOutputRelaisStatus(JD16, LOW);
  delay(1000);
  SetOutputRelaisStatus(JD16, HIGH);
  delay(1000);
}
```

Interessant und in eigenen Projekten verwendbar ist hier die Funktion

```
SetOutputRelaisStatus(RelaisPort, Status)
```

- RelaisPort (JD1 -JD 16)
- Status (HIGH = ein, LOW = aus)

Im Hauptprogramm kann an jeder Stelle auf die Funktion zurückgegriffen werden. Ich wünsche viel Spaß beim Nachbauen und verwenden in eigenen Projekten ! Im nächsten Teil werde ich zeigen, wie die Funktion in eine Bibliothek ausgelagert werden kann und somit der Port auf andere Projekte flexibler und einfacher wird.