Ansteuerung eines 16 Relais Modules mit dem I2C Bus (Teil 3)

Willkommen zum dritten und letzten Teil der Reihe um die Ansteuerung eines 16 Relais Modules mit dem I2C Bus. Im vorherigen Teil haben wir uns angesehen, wie eine eigene Bibliothek erstellt werden kann und manuell in die Arduino IDE eigebunden wird. Im heutigen Teil der Reihe möchte ich zeigen, wie eine einfach importier bare Bibliothek erstellt werden kann. Diese kann dann zum Beispiel an Bekannte und Freunde weitergeben werden, oder auch auf GitHub veröffentlicht werden. Um dieses Ziel zu erreichen, schauen wir uns zunächst einmal an was eine importier bare ZIP - Bibliothek normalerweise an Dateien enthält. Dies sind im Einzelnen:

1. Header-Datei (.h) - Enthält die Definitionen der Bibliothek
2. Quellcode (.cpp) - Enthält den Code der Bibliothek in OOP Sprache
3. Schlüsselwortdatei (.txt) - Enthält die verwendeten Schlüsselwörter
4. Readme-Datei (.txt) - Enthält weitere Informationen (Version, TODOs usw.)
5. Beispiele (.ino) - Diese sollen zeigen, wie er die Bibliothek bedient wird.

Die Header Datei und die Quell Code Datei (Nummer 1 und Nummer 2 in der Liste) haben wir bereits im Teil 2 erstellt. Diese können für den heutigen Teil weiterverwendet werden. Vollständigkeitshalber habe ich die beiden Dateien am Ende des Blogs abermals angehängt. Neu hinzugekommen ist jedoch u.a. die Schlüsselwortdatei „keywords.txt“, die für das hervorheben zuständig ist. Es gibt zwei verschiedene Arten von Schlüsselwörtern, nämlich „KEYWORD1“ für definierte Klassen und „KEYWORD2“ für in den Klassen definierte Funktionen. Die Schlüsselwortdatei für unsere Bibliothek wird wie folgt gebaut:

Keywords.txt

```
RelayModule KEYWORD1
write KEYWORD2
```

Man erkennt das die Klasse RelayModule mit KEYWORD1 getaggt wird, während die Funktion „write“ mit KEYWORD2 getaggt wird. Die Verwendung des Wortes „write“ wird somit bei Verwendung hervorgehoben. Eine weitere Datei die für unsere Bibiothek benötigt wird, ist die Datei „readme.md“. In dieser Datei wird in Ascii-Schrift eine kurze Zusammenfassung der Funktion der Klasse gegeben. Diese wird beim Einstellen der Bibliothek auf Github den Benutzern angezeigt. Unsere Readme enthält folgenden Text:

Readme.md

```
Ein 16-fach Relais Modul besitzt zur Ansteuerung 16 LOW aktive TTL- kompatible Eingänge. Wenn
diese direkt an einen Arduino angeschlossen werden und getrennt angesteuert werden,
bleiben für weitere Peripherie i.d.R. nicht mehr sehr viele freie Ports übrig. Ein Ausweg aus dieser
Problematik bietet diese Bibliothek in Verbindung mit zwei I2C Port Expander Bausteinen PC8574.
Diese stellen pro Baustein 8 parallel Ausgänge und Eingänge an dem I2C Bus zur Verfügung.
Das Relais Modul kann mithilfe dieser Bibliothek komplett über den I2C Bus unter Belegung von 2
IC2 Adressen angesteuert werden.
Weitere Informationen auf www.github.com/kuchto
```

Zuletzt und nachdem alle oben genannten Dateien erstellt worden sind, können noch ein oder mehrere Beispielsketche erstellt werden, damit Ihre Benutzer, die Ihre selbst genaue Bibliothek nutzen möchten, über mindestens einen Beispielsketch (Beispielprogramm) verfügen. Dieser Beispielsketch zeigt anschaulich die Verwendung der möglichen Funktionen der Bibliothek durch einfache Nutzung. Beispielsketche tragen die Endungen „.ino“ sind im in einer Ordnerstruktur unterhalb vom Hauptordner „examples“ abgelegt. In dem Hauptordner „Examples“ muss für jeden Beispielsketch ein eigener Ordner angelegt werden. Für unsere Bibliothek habe ich nur ein Beispielsketch erstellt, der im Unterordner „I2C_16_fach-Relaismodul“ angelegt habe. Zu beachten ist, das der Name des Unterordnens unter „examples“ mit den Namen der Sketch Datei übereinstimmen muss. Nachfolgend der Inhalt des Beispiel Sketches:

I2C_16_fach-Relaismodul.ino

```
// Beispiel Sketch zur Bibliothek RelayModule
// Tobias Kuch 2020

#include <RelayModule.h>


RelayModule Relay;
void setup()
{
  Serial.begin(9600);
  bool InitOK = Relay.begin();
  if (InitOK)
    {
    Serial.println("Initalisierung OK");
    } else
    {
    Serial.println("Initalisierung nicht OK. Module angeschlossen und richtig adressiert ?");
    }
}

void loop()
{
Relay.write(JD1,HIGH);
delay(1000);
Relay.write(JD2,HIGH);
delay(1000);
Relay.write(JD3,HIGH);
delay(1000);
Relay.write(JD4,HIGH);
delay(1000);
Relay.write(JD5,HIGH);
delay(1000);
Relay.write(JD6,HIGH);
delay(1000);
Relay.write(JD7,HIGH);
delay(1000);
Relay.write(JD8,HIGH);
delay(1000);
Relay.write(JD9,HIGH);
delay(1000);
Relay.write(JD10,HIGH);
delay(1000);
Relay.write(JD11,HIGH);
```

```
delay(1000);
Relay.write(JD12,HIGH);
delay(1000);
Relay.write(JD13,HIGH);
delay(1000);
Relay.write(JD14,HIGH);
delay(1000);
Relay.write(JD15,HIGH);
delay(1000);
Relay.write(JD16,HIGH);
delay(1000);
}
```

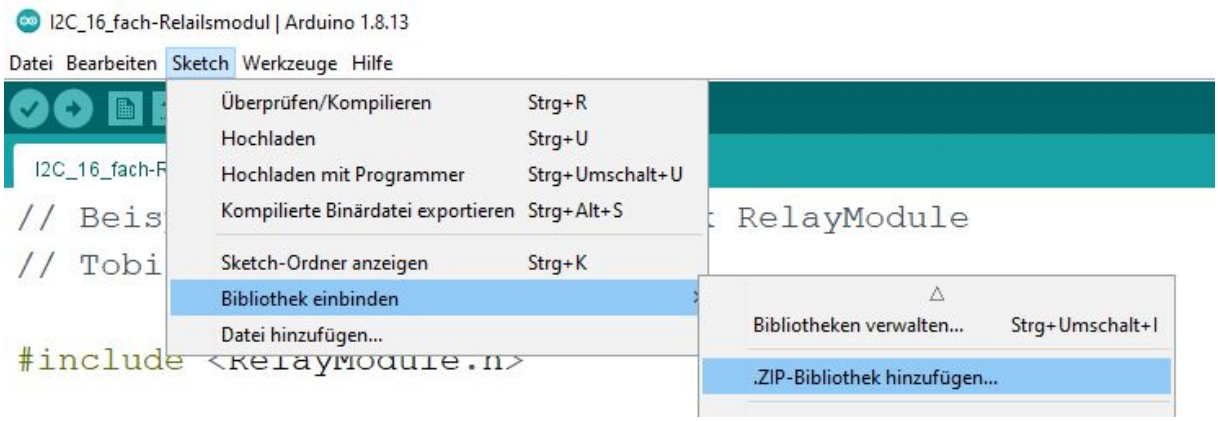
Die Ordnerstruktur der Bibliothek ist also:

```
/root/
Relaymodule.h
Relaymodule.cpp
Keywords.txt
readme.md
/      /examples/
/…../……/I2C_16_fach-Relaismodul/
                                                    /I2C_16_fach-Relaismodul.ino
```

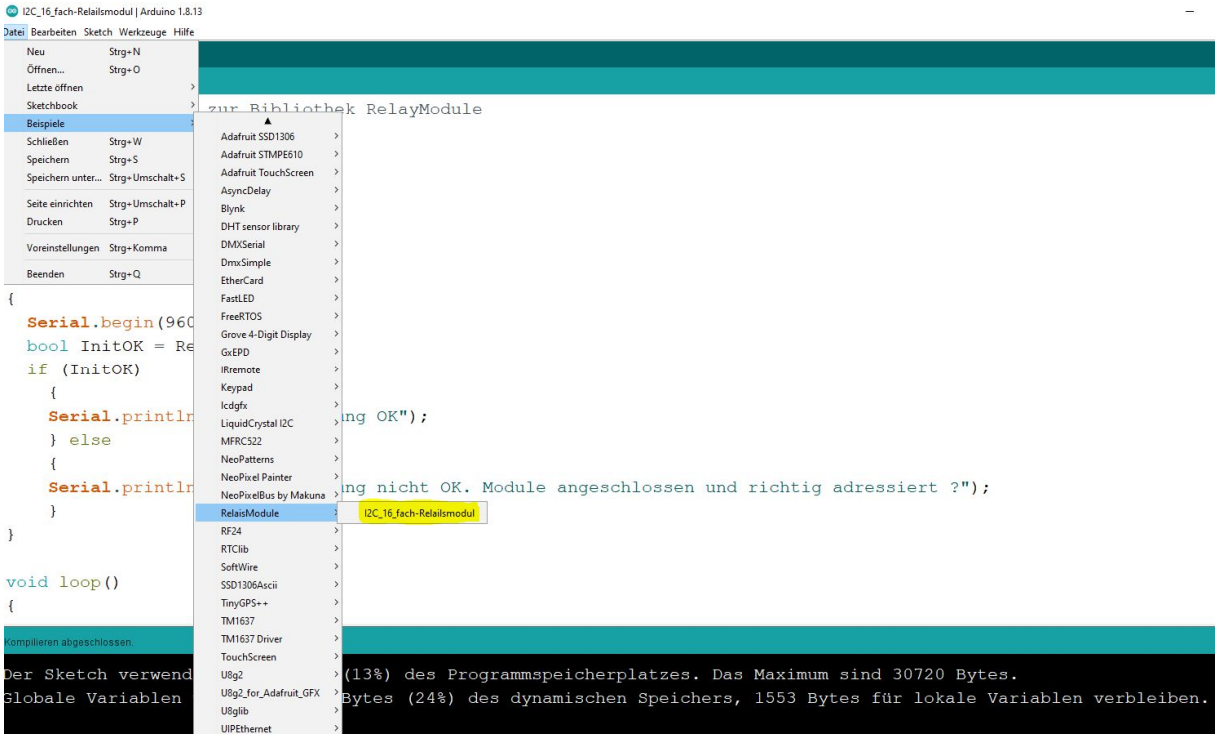
Sobald alle o.g. Dateien erstellt wurden, können Sie mithilfe eines ZIP Packers in ein .ZIP Datei verpackt werden. Wichtig dabei ist, dass dies unter Beibehaltung der Ordnerstruktur geschieht und die daraus resultiere ZIP Datei diese Ordnerstruktur wiederspiegelt. Hier ist die fertige ZIP Archiv Datei zum Herunterladen:

RelaisModule.zip

Die von mir erstellte Zip Bibliothek kann in der Arduino DIE ganz einfach unter dem Menüpunkt Sketch -> Bibliothek einbinden -> .ZIP Bibliothek hinzufügen eigebunden werden



Sobald die ZIP Datei importiert wurde, kann der Beispiel Sketch für die neue Bibliothek unter Datei -> Beispiele->RelaisModule->I2C_16_fach-Relaismodul abgerufen werden. Im nachfolgenden Screenshot ist dies verdeutlicht:



Zur Vervollständigung für alle, die bis jetzt noch nicht alle Teile der Reihe lesen konnten sind die zusätzlich benötigten Inhalte der Dateien „RelayModue.h“ und „relayModule.cpp“ anhängend:

RelayModule.h

```
//Libary  from Tobias Kuch - tobias.kuch@googlemail.com
#ifndef RelayModule_h // this prevents problems if someone accidently #include's your library twice.
#define RelayModule_h

#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

class RelayModule {
  public:
```

```
  RelayModule();
  RelayModule(byte AlternatePort);
  bool begin();
  bool write(byte RelaisPort, bool Status);
  private:
  byte PC8574_LCD_Port_Expander_Base_Addess = 0;
  volatile byte valPort_A = 0;
  volatile byte valPort_B = 0;
};

#endif
```

RelayModule.cpp

```
//Libary from Tobias Kuch - tobias.kuch@googlemail.com
#include "Arduino.h" // gives you access to the standard types and constants of the Arduino language
(this is automatically added to normal sketches, but not to libraries).
#include "RelayModule.h" // and to the definitions in header file
#include <Wire.h>


RelayModule::RelayModule()
{
PC8574_LCD_Port_Expander_Base_Addess = 39;
}

RelayModule::RelayModule(byte AlternatePort)
{
PC8574_LCD_Port_Expander_Base_Addess = AlternatePort;
}


bool RelayModule::begin()
{
byte returna,returnb = 0;
byte SendCommand = 0;
Wire.begin(); // join i2c bus

SendCommand = 0 ^ 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
returna = Wire.endTransmission(true);                     // stop transmitting

SendCommand = 0 ^ 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
returnb = Wire.endTransmission(true);                     // stop transmitting

if ((returnb == 1) or (returna == 1))
   {
   return false;
   } else
   {
```

```
   return true;
   }

}


bool RelayModule::write(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;
```

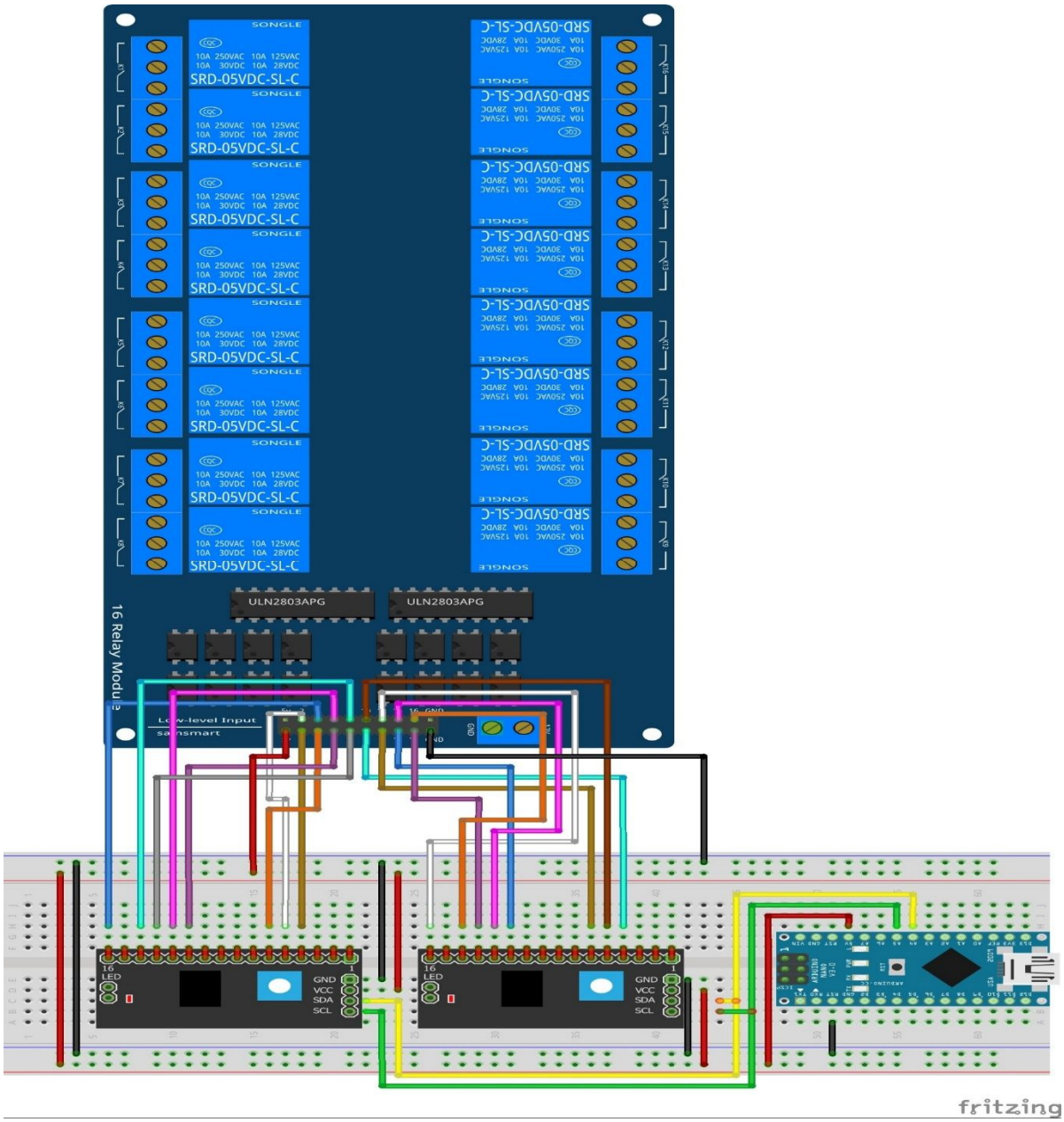
```
    } else
    {
    return false;
    }
  }
else
  {
  return false;                                    // Out of Rage
  }
}
```

I2C_16_fach-Relaismodul.ino (Beispielsketch)

```
// Beispiel Sketch zur Bibliothek RelayModule
// Tobias Kuch 2020

#include <RelayModule.h>


RelayModule Relay;
void setup()
{
  Serial.begin(9600);
  bool InitOK = Relay.begin();
  if (InitOK)
    {
    Serial.println("Initalisierung OK");
    } else
    {
    Serial.println("Initalisierung nicht OK. Module angeschlossen und richtig adressiert ?");
    }
}

void loop()
{
Relay.write(JD1,HIGH);
delay(1000);
Relay.write(JD2,HIGH);
delay(1000);
Relay.write(JD3,HIGH);
delay(1000);
Relay.write(JD4,HIGH);
delay(1000);
Relay.write(JD5,HIGH);
delay(1000);
Relay.write(JD6,HIGH);
delay(1000);
Relay.write(JD7,HIGH);
delay(1000);
Relay.write(JD8,HIGH);
delay(1000);
Relay.write(JD9,HIGH);
delay(1000);
Relay.write(JD10,HIGH);
delay(1000);
Relay.write(JD11,HIGH);
delay(1000);
Relay.write(JD12,HIGH);
delay(1000);
Relay.write(JD13,HIGH);
```

```
delay(1000);
Relay.write(JD14,HIGH);
delay(1000);
Relay.write(JD15,HIGH);
delay(1000);
Relay.write(JD16,HIGH);
delay(1000);
}
```

Verschaltungsschema der Relaisplatine:



Ich wünsche viel Spaß beim Nachbauen und schreiben eigener Bibliotheken.