Ansteuerung eines 16 Relais Modules mit dem I2C Bus (Teil 2)

Haben Sie sich schon einmal gefragt, was hinter den komfortabel einzubindenden Bibliotheken steckt, die Sie in der Arduino IDE über den Bibliotheksverwalter steckt durch die Sie immer neue Funktionen Ihrem Arduino beibringen können? Oder wollten gar selbst mal eine Arduino Bibliothek schreiben? Dann ist dieser und der kommende Teil der Reihe „Ansteuerung eines 16 Relais Modules mit dem I2C Bus“ genau der richtige Blog für Sie! Wir werden im heutigen Teil zeigen, wie manuell eine neue Bibliothek erstellt wird, und wie diese Bibliothek manuell den bereits bestehenden Bibliotheken hinzugefügt werden kann. Leider ist da die bestehende DIE weniger Einsteigerfreundlich, aber ich werde hier jeden Schritt einzeln beschreiben. Zunächst einmal müssen wir uns einen Namen für unsere neue Bibliothek überlegen, der aussagekräftig ist, da dieser an mehreren Stellen benötigt wird. Ich habe mich für die Umschreibung des Sketches aus dem ersten Teil der Reihe für den Bibliotheksnamen „RelaisModule“ entschieden. Für eine funktionierende Bibliothek werden mindestens 2 Dateien mit dem Namensschema „Bibliotheksnamen.h“ als Deklarationsdatei und eine Datei „Bibliotheksnamen,cpp“ als Codedatei benötigt. In der Header Datei werden Konfigurationseinstellungen, Variablen und Konstanten Definitionen, sowie Funktionsdeklarationen (Prototypen) vorgenommen. Der Eigentliche Code für die Funktionen ist dann in der cpp Datei zu finden. Auch ist bei der Erstellung und Programmierung der Bibliotheken objektorientierte Programmierung Grundvoraussetzung. Dazu aber später mehr. Zunächst erzeugen wir in einem beliebigen Temporären Ordner leere Dateien mit den Namen RelayModule.h und RelayModule.cpp. In diese werden folgende Inhalte jeweils hineinkopiert:

Datei: RelayModule.h

```
#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
```

Datei: RelayModule.cpp

```
/*
 * 16 Port Relay Module Control via two PC8574 LCD Port Expander Module(s)
 * Simple Function to Control 16 Relays via I2C to save Ports.
 * Notes to Addressing the PC8574 Modules:
* Tobias Kuch 2020 - tobias.kuch@googlemail.com
 * For free use in your own Projects
 *
 * Licensed under GPL 3.0
 *
 */

#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  ;o)
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
  }
}
```

Nun müssen wir diese beiden Dateien der IDE als Bibliothek in der Bibliotheksorder für eigene Bibliotheken zur Verfügung stellen. Dieser kann sich, je nach Betriebssystem und Konfiguration an unterschiedlichen Orten befinden. Um den Ort herauszufinden, an dem sich Ihre Bibliotheken befinden, können Sie in der Arduino-Entwicklungsumgebung den Reiter Datei/Voreinstellungen aufrufen. Dort ist gespeichert, welcher Pfad als Sketchbook Speicherort benutzt wird. In diesem SketchBook Ordner befindet sich ein Unterordner mit dem Namen „`libraries`“.

In diesem Ordner erzeugen wir einen Unterordner mit dem Namen „Relaismodule“ und kopieren die im ersten Schritt erstellten Dateien RelayModule.h und RelayModule.cpp hinein. Kleiner Tipp: Oft ist der Bibliotheksorder in der Struktur C:\Users\Anmeldename \Documents\Arduino\libraries\ zu finden. Nachdem Sie nun die Dateien erzeugt und in den Bibliotheksordner in einem Unterordner namens „Relaismodule“ kopiert haben, ist es an der Zeit die Arduino IDE neu zu starten. Danach kann die neue Bibliothek genutzt werden, indem diese mir der Compilerdirektive #include <RelayModule.h> zuerst inkludiert und anschließend ein Objekt der Klasse RelayModule instanziiert wird.

Instanziieren einer Klasse: In der objektorientierten Programmierwelt werden Funktionen, Variablen und auch Konstanten in einer Klasse definiert. Durch eine Definition einer Klasse selbst wird noch kein Speicherplatz im RAM und keine CPU Rechenzeit belegt. Erst das instanziieren einer Klasse erzeugt aus der Klasse ein Objekt, das gemäß den definierten Funktionen und Variablen genutzt werden kann. Eine Instanziierung einer Klasse kann mit oder ohne Parameter erfolgen. Der Konstruktor innerhalb einer Klasse muss bei Übergabe von Parametern entsprechend vorbereitet sein. Ein Konstruktor kann auch überladene Funktionen erhalten.

Die Instanziierung der Klasse Relay erfolgt in der Zeile:

```
RelayModule Relay;
```

Der Konstruktor der Klasse Relaismodul ist überladen, d.h. akzeptiert der Konstruktor die o.g Schreibweise oder auch alternativ:

```
RelayModule Relay(I2C Adresse des ersten Bausteins);
```

Falls keine I2C Adresse im Instanziierungsaufruf angegeben wird, wird die Standard Adresse 38ded für den ersten Baustein, und 37dec für den zweiten Baustein angenommen. Andernfalls wird die im Aufruf angegebene I2C Adresse für den ersten Baustein übernommen. Die zweite Adresse für den zweiten Baustein wird nach der Formel Basisadresse erster Baustein – 1 = Adresse zweiter Baustein festgelegt. Nachdem die Instanz so erstellt worden ist, kann die Methode

```
.write (Relaisname,Status)
```

verwendet werden. Akzeptierte Parameter sind:

Relaisname	JD1-JD16
Status	HIGH oder LOW (High Relais zieht an)

Hier ein Beispielcode unter Nutzung unserer Klasse:

```
#include <RelayModule.h>

RelayModule Relay;
void setup()
{
  bool Res =Relay.begin();
}

void loop()
{

Relay.write(JD1,HIGH);
delay(1000);
Relay.write(JD1,LOW);
delay(1000);
Relay.write(JD11,HIGH);
delay(1000);
Relay.write(JD11,LOW);
delay(1000);
Relay.write(JD16,HIGH);
delay(1000);
Relay.write(JD16,LOW);
delay(1000);

}
```

Im vorherigen Teil der Reihe ist der Physische Aufbau der Relais und der beiden I2C Modules beschrieben. Dieser gilt für unseren heutigen Teil ebenso:

Im nächsten und letzten Teil dieser Reihe werde ich zeigen, wie Sie selbst erstellte Bibliotheken einfach in ein Format bringen können, das es Ihnen erlaubt diese für bekannte oder Freunde weiterzugeben, ohne das die hier gezeigten Schritte notwendig sind. Weitere Informationen über das Thema objektorientiere Programmierung und auch weitere Blogs finden sie auf meiner Github Seite unter https://github.com/kuchto. Ich wünsche viel Spaß beim Nachbauen und schreiben eigener Bibliotheken. Bis zum nächsten Mal.