Arduino verstehen und anwenden

Teil 23: Portexpander, Displays und Echtzeituhren am I²C-Bus

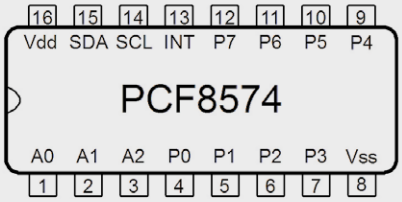

Bild 1: Der Portexpander PCF8574

Portexpander am I²C-Bus

Obwohl der klassische Arduino bereits über eine ganz beträchtliche Anzahl von I/O-Pins verfügt, tritt bei zunehmend komplexeren Projekten immer wieder die Situation auf, dass nicht genügend Pins zur Verfügung stehen. Insbesondere Displays und Anzeigeeinheiten belegen bereits mehrere Pins. Wenn dann noch Sensoren oder Aktuatoren angesteuert werden sollen, dauert es oft nicht lange, bis alle Pins belegt sind. Eine Möglichkeit, diese Limitierung zu umgehen, besteht natürlich im Umstieg auf einen größeren Prozessortyp, etwa den ATmega1280. Dementsprechend könnte also z. B. der Arduino Mega eingesetzt werden. Auch die Verwendung eines Arduino DUE wäre durchaus eine mögliche Alternative.

Diese Boards sind aber vergleichsweise teuer und von den geometrischen Abmessungen her wesentlich größer. Wenn nur einige zusätzliche Pins benötigt werden, so sind sogenannte Portexpander das Mittel der Wahl. Diese Bausteine können über den I²C-Bus angesteuert werden und belegen damit nur die beiden Pins für SDA und SCL.

Ein bekannter Vertreter in der Familie der Portexpander ist der PCF8574 (Bild 1). Dieser bietet acht zusätzliche I/O-Ports pro Baustein. Da bis zu acht Bausteine an einem I²C-Bus betrieben werden können, sollten die mit diesen Bausteinen erzielbaren Portzahlen für die allermeisten Fälle ausreichend sein.

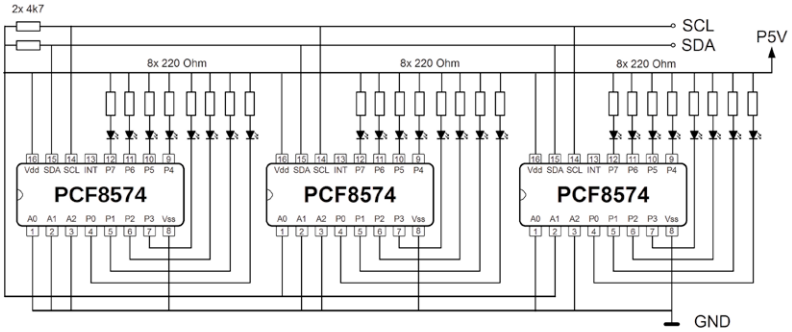

Bild 2: Schaltplan zum Mega-Lauflicht mit drei PCF8574-Portexpandern

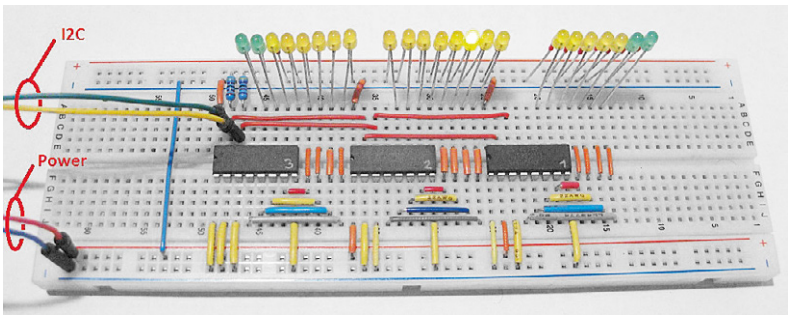

Bild 3: Aufbaubeispiel zum Mega-Lauflicht

Vierundzwanzig zusätzliche I/O-Pins!

Für einen ersten Einstieg in die Technologie der Portexpander soll hier ein Mega-Lauflicht vorgestellt werden. Mit drei PCF8574 sind damit bis zu 24 LEDs ansteuerbar. Die nachfolgenden Bilder zeigen den Schaltplan (Bild 2) und ein Aufbaufoto (Bild 3). Im Aufbaufoto ist zu erkennen, dass die Schaltung mit lediglich vier Leitungen auskommt (Vcc, GND, SCL und SDA).

```
// Mega-LED-chaser using PCF8574(A) Portexpanders

#include <Wire.h>
#define addr0 0b00100000 // I2C adress of 1. PCF8574
// 0b00100001 = adddr0 + 1  I2C adress of 2. PCF8574
// 0b00100010 = adddr0 + 2  I2C adress of 3. PCF8574

void setup()
{ Wire.begin();
}

void loop()
{  for (int k=0; k<=2; k++)
   { for (int i=1; i<=128; i*=2)
     { PCF8574_Write(k, i); delay(100); }
     PCF8574_Write(k, 0);
   }
   for (int k=2; k>=0; k--)
   { for (int i=128; i>=1; i/=2)
     { PCF8574_Write(k, i); delay(100); }
     PCF8574_Write(k, 0);
   }
}

void PCF8574_Write(byte addr, byte data)
{ Wire.beginTransmission(addr0+addr);
  if (data>0) Wire.write(~(128/data));
  else Wire.write(255);
  Wire.endTransmission();
}
```

Im Sketch zum Mega-Lauflicht muss zunächst die I²C-Bibliothek eingebunden werden. Das wird durch *#include <Wire.h>* erledigt.

Anschließend werden die Adressen für die einzelnen Bausteine festgelegt. Da beim ersten PCF8574 alle frei definierbaren Adressbits auf Ground liegen, lautet die Adresse in binärer Schreibweise 0b00100000.

Darin stehen die ersten fünf Bits (00100) für den intern festgelegten Adressteil. Die letzten drei Bits lauten 000. Da bei der Festlegung der Adressen für die anderen beiden ICs binär hochgezählt wurde, lauten die jeweils letzten drei Bits: 001 und 010.

In der Main-Loop werden zwei Schleifen definiert. Die erste Schleife lässt einen Lichtpunkt von links nach rechts laufen, die zweite Schleife sorgt dafür, dass der Punkt wieder zurückläuft.

In den Schleifen werden die einzelnen I²C-Bausteine nacheinander angesprochen. Da die Adressen der Bausteine direkt aufeinanderfolgen, kann das mit einer einfach hochzählenden for-Anweisung erledigt werden. In einer zweiten Schleife wird der Schleifenindex bei jedem Durchlauf verdoppelt. Damit wird eine Folge von Zweierpotenzen erzeugt:

1, 2, 4, 8, 16, 32, 64, 128

oder in Binärschreibweise:

00000001	=	1
00000010	=	2
00000100	=	4
00001000	=	8
00010000	=	16
00100000	=	32
01000000	=	64
10000000	=	128

Dies entspräche einem von links nach rechts laufenden Lichtpunkt. Diese Sequenz wird an die Routine

void PCF8574_Write(byte addr, byte data)

übergeben. Dort wird der Wert 128 durch diese Zahlenfolge dividiert und eine anschließende Invertierung:

Wire.write(~(128/data));

führt dazu, dass der Lichtpunkt mit größer werdenden Zahlen nach rechts läuft. Dies entspricht der intuitiven Vorstellung, dass größere Zahlen weiter rechts auf einem Zahlenstrahl liegen. Natürlich könnte man dieses Verhalten auch auf verschiedenen anderen Wegen erhalten.

Da durch die äußere Schleife alle Bausteine nacheinander angesprochen werden, läuft der Lichtpunkt schließlich durch alle 24 Leuchtdioden.

Im zweiten Schleifenpaar wird entsprechend heruntergezählt bzw. die übergebene Zahl jeweils halbiert. Damit läuft der Lichtpunkt von rechts nach links.

Ansteuerung eines LC-Displays via I²C und PCF8574

Da LC-Displays besonders häufig eingesetzt werden, wird die Möglichkeit zur Einsparung von Prozessorpins hier etwas detaillierter erläutert.

Ein besonderer Vorteil dieser Methode ist, dass damit problemlos auch mehrere Displays an einen Controller angeschlossen werden können. In diesem Fall wird jedes Display an einen eigenen PCF8574 angeschlossen. Wenn die einzelnen Bausteine mit unterschiedlichen Adressen versehen werden, können bis zu 8 LC-Displays an einem einzigen Arduino betrieben werden. Die einzelnen LCDs sind dann einfach über die zugehörigen I²C-Adressen ansprechbar. Hier wird der Vorteil der I²C-Ansteuerung nochmals besonders deutlich: Alle LCDs belegen in dieser Konfiguration nur zwei I/O-Pins. Auch hier ist wieder eine komfortable Bibliothek unter

http://www.xs4all.nl/~hmario/arduino/LiquidCrystal_I2C/LiquidCrystal_I2C.zip

verfügbar.

Aufgrund einer Inkompatibilität kann diese Library nicht mit der IDE ab Version 1.8 kompiliert werden. Bitte verwenden Sie hierfür die Version 1.0.5.

Das zugehörige Programm ist sehr kompakt dank der LiquidCrystal_I2C-Library. Nach dem Einbinden der Bibliotheken mit

#include <Wire.h>
#include <LiquidCrystal_I2C.h>

muss nur noch über

LiquidCrystal_I2C lcd(0x20,16,2);

die I²C-Adresse und das Format des Displays festgelegt werden.

Über den noch freien Pin des PCF8574 kann noch mit

lcd.backlight();

die Hintergrundbeleuchtung des Displays geschaltet werden, soweit diese vorhanden ist. Falls das Backlight höhere Ströme erfordert, sollte noch eine Transistorstufe zwischengeschaltet werden.

```
// I2C_LCD_test

#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// set the I2C address to 0x20 for PCF8574P
// A0=A1=A2=0 => addr = 0b00100000 = 0x20
// for 2 line display wit 16 char per line
LiquidCrystal_I2C lcd(0x20,16,2);

void setup()
{ lcd.init();            // initialize display
}

void loop()
{ lcd.backlight();       // backlight on, if available
  lcd.setCursor(0, 0);
  lcd.print("Hello - ");
  lcd.setCursor(0, 1);
  lcd.print(" I2C ok ");
  while(1);
}
```

Real-Time Clock mit Datumsanzeige

Mit einem Mikrocontrollerboard mit quarzgenauem Takt kann man bereits eine sehr präzise Uhr aufbauen. Da der Arduino über einen 16-MHz-Quarz verfügt, könnte man damit bereits eine recht präzise Uhr aufbauen. In diesem Artikel soll jedoch die Präzision der Zeitmessung noch weiter verbessert werden. Dies ist durch den Einsatz eines Echtzeitbausteins möglich. Diese ICs sind auch unter der Bezeichnung RTCs für **R**eal-**T**ime **C**locks bekannt. Sie beinhalten einen sehr präzisen Timer, der mit einem sogenannten Uhrenquarz betrieben wird. Dieser Quarz läuft mit einer vergleichsweise niedrigen Frequenz von 32,768 kHz. Diese Zahl entspricht einer Zweierpotenz ($2^{15} = 32768$), sodass durch fortlaufende elektronische Frequenzteilung ein genauer Sekundentakt

erzeugt werden kann. Zudem sind Quarze mit dieser Frequenz sehr präzise abgestimmt und erreichen eine gegenüber den klassischen Mikrocontrollerquarzen nochmals deutlich verbesserte Langzeitstabilität.

Neben dem Oszillator enthält der RTC-Baustein auch die komplette Schaltung zur Berechnung und Ausgabe der vollständigen Uhrzeit und des aktuellen Datums. Dadurch können im Controller beträchtliche Ressourcen eingespart und für andere Zwecke verwendet werden. Die Ankopplung des Bausteins erfolgt wieder über den I²C-Bus, sodass auch hier sehr sparsam mit Ressourcen umgegangen wird.

Darüber hinaus ermöglicht der Einsatz eines separaten RTC-Bausteins eine einfache Gangreserve. Mittels eines kleinen Akkus (z. B. 3,6 V, 180 mAh, siehe Bild 4) kann der Baustein problemlos gepuffert werden, sodass er auch bei Ausfall der Hauptspannungsversorgung über eine längere Zeit weiterläuft. Damit muss die Echtzeituhr nur einmal korrekt gestellt werden, dann läuft sie unbeirrt und unaufhaltsam mit hoher Präzision weiter.

Eine Bibliothek zur Ansteuerung des PCF8583 steht unter

https://github.com/edebill/PCF8583

zur Verfügung.

Zum Stellen der Uhr muss lediglich ein String der Form *YYMMddhhmmss;* über den I²C-Bus an den Baustein gesendet werden. Die Zeichen haben dabei die übliche Bedeutung:

YY: Zweistellige Jahresangabe
MM: Zweistellige Monatsangabe
dd: Zweistellige Tagesangabe
hh: Zweistellige Stundenangabe
mm: Zweistellige Minutenangabe
ss: Zweistellige Sekundenangabe

Der nachfolgende Strichpunkt schließt die Zeichenfolge ab. Mit

170901190000;

wird die Uhr also auf den 1. September 2017, 19:00:00 gestellt.

Im Programm muss neben der Bibliothek für den verwendeten RTC-Baustein PCF8583 auch wieder die wire-Library eingebunden werden. Es folgt die Festlegung der benötigten Variablen. Als I²C-Adresse ergibt sich Null, da der Adress-Pin A0 auf GND liegt. Die Variable *yearof = 2000* legt das Jahrtausend fest. Die Werte *of* und *mu* definieren zwei Offsetkonstanten entsprechend dem Datenformat des PCF8583.

Im Set-up wird neben der seriellen Schnittstelle auch ein LCD-Modul initialisiert. Sollen die Zeitdaten auf das LC-Display ausgegeben werden, so muss dieses entsprechend der Standard-Konfiguration nach Bild 5 angeschlossen werden.

Der erste Funktionsblock in der Hauptschleife erfüllt zwei Aufgaben. Zum einen liest er die Zeitdaten vom I²C-Bus und zum anderen schreibt er die aktuelle Zeit und das Datum in den RTC-Baustein, falls diese Werte von der seriellen Schnittstelle gelesen werden.

Der Rest des Programms dient nur noch dazu, die Zeitdaten sowohl an die serielle Schnittstelle als auch an das LC-Display auszugeben. Dabei gibt es zwei Besonderheiten. Zunächst werden die Daten mit dem Formatierungsbefehl „sprintf" in eine gut lesbare Form gebracht. Die Angabe *%02d:%02d:%02d* dient dazu, die Zahlenfolge in die für Uhrzeiten übliche Form zu bringen. Das heißt hier, Dezimalanzeige mit führenden Nullen. Für die Darstellung auf einem PC-Terminal werden die Zeitangaben dann lediglich in numerischer Form gesendet. Die zweite Spezialität besteht darin, dass für das LC-Display die Monatswerte noch in die Monatsnamen konvertiert werden, sodass zum Beispiel statt 01.09.2017 die Anzeige 1. September 2017 erscheint.

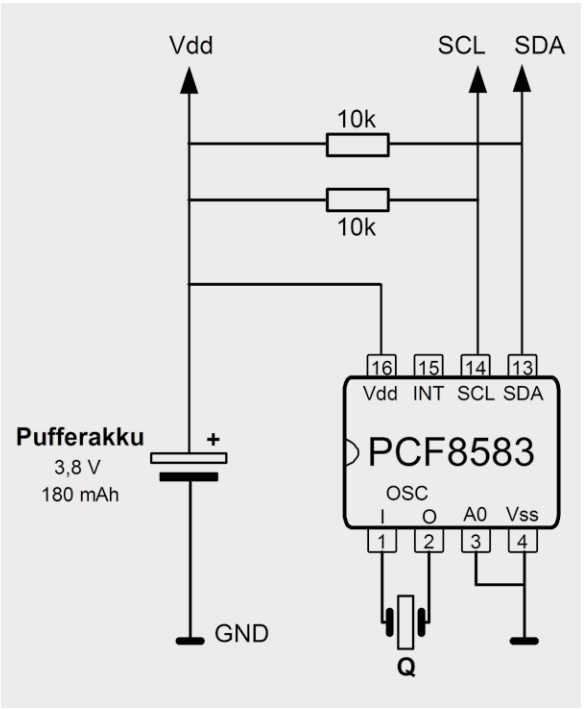

Bild 4: Real-Time-Clock-Baustein PCF 8583 am I²C-Bus

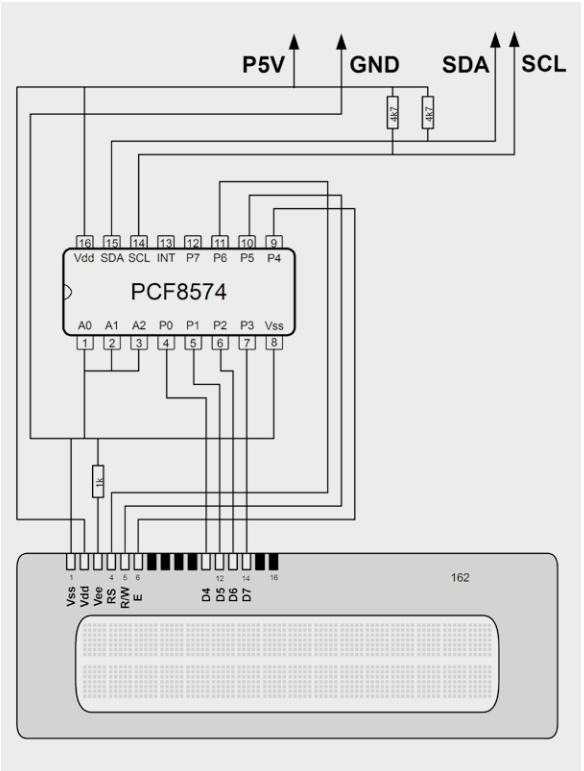

Bild 5: I²C-fähiges LC-Display im Eigenbau

```
// RTC clock using LCD display with 16 columns, 4 lines
// Set time format YYMMddhhmmss;

#include <Wire.h>               // include I2C communication
#include <stdio.h>
#include <PCF8583.h>            // include for RTC unit PCF8583
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  // initialize the LCD interface pins

int  correct_address=0, yearof=2000; byte of=48, mu=10;
PCF8583 p (0xA0);

void setup(void)
{ Serial.begin(9600);
  Serial.println("starting");
  lcd.begin(16, 4);               // set up the LCD' 16 columns , 4 lines:
  lcd.setCursor(3, 0); lcd.print("RTC Clock");
}

void loop(void)
{ if(Serial.available() > 0)
  {  p.year=(byte) ((Serial.read()-of)*mu+(Serial.read()-of))+yearof;
     p.month=(byte) ((Serial.read()-of)*mu+(Serial.read()-of));
     p.day=(byte) ((Serial.read()-of)*mu+(Serial.read()-of));
     p.hour=(byte) ((Serial.read()-of)*mu+(Serial.read()-of));
     p.minute=(byte) ((Serial.read()-of)*mu+(Serial.read()-of));
     p.second=(byte) ((Serial.read()-of)*mu+(Serial.read()-of));
     if(Serial.read()==';') { Serial.println("setting date"); p.set_time();}
  }
  p.get_time();
  char time[50]; char date[50];
  sprintf(time, "%02d:%02d:%02d", p.hour, p.minute, p.second);
  sprintf(date, "%02d/%02d/%02d", p.year, p.month, p.day);
  Serial.print(date); Serial.print(" - "); Serial.println(time);
  lcd.setCursor(3, 1); lcd.print(time);
  lcd.setCursor(19, 0); lcd.print(p.day); lcd.print(". ");
  switch (p.month)
      { case 1 : lcd.print ("Januar    "); break;
        case 2 : lcd.print ("Februar   "); break;
        case 3 : lcd.print ("Maerz     "); break;
        case 4 : lcd.print ("April     "); break;
        case 5 : lcd.print ("Mai       "); break;
        case 6 : lcd.print ("Juni      "); break;
        case 7 : lcd.print ("Juli      "); break;
        case 8 : lcd.print ("August    "); break;
        case 9 : lcd.print ("September "); break;
        case 10 : lcd.print ("Oktober   "); break;
        case 11 : lcd.print ("November  "); break;
        case 12 : lcd.print ("Dezember  "); break;
        }
  lcd.setCursor(21, 1); lcd.print(p.year); delay(100);
}
```

Ausblick

Nachdem im letzten Beitrag die Grundlagen und in diesem Artikel verschiedene Anwendungen des I²C-Busses erläutert wurden, ist dieses Thema nun abgeschlossen.

Im nächsten Teil geht es daher um einen weiteren sehr wichtigen Bus-Typ, das Serial Peripherial Interface, auch als SPI-Bus bekannt. Dieses System zeichnet sich insbesondere durch seine höhere Geschwindigkeit aus.

Für den Arduino besitzt der SPI-Bus sogar eine besondere Bedeutung, da dieses Interface die Hauptprogrammierschnittstelle für den Mikrocontroller des Arduinos darstellt. Darüber hinaus erlaubt die SPI-Schnittstelle den Anschluss von weit verbreiteten Funkmodulen und sogar die bekannten SD- und microSD-Karten werden über SPI mit dem Arduino verbunden. Schließlich können auch noch zwei oder mehrere Arduinos über den SPI-Bus verbunden werden und so Daten austauschen. Auf diese Weise lässt sich ein regelrechter Arduino-Cluster aufbauen, dessen Rechenkapazität lediglich durch die Anzahl der verwendeten Arduinos begrenzt wird.

Den Abschluss des Themas „Schnittstellen und Bussysteme" wird dann der übernächste Artikel bringen. Darin wird neben verschiedenen Spezialsystemen auch eine Übersicht zu allen vorgestellten Datenübertragungsmöglichkeiten enthalten sein. Damit steht der Auswahl des am besten geeigneten Übertragungssystems für eine bestimmte Anwendung nichts mehr im Wege. ELV

Weitere Infos:

- Grundlagen zur elektronischen Schaltungstechnik finden sich in der E-Book-Reihe „Elektronik!" (www.amazon.de/dp/B000XNCB02)
- FRANZIS AVR-Mikrocontroller in C programmieren, Best.-Nr. CO-09 73 52
- Elektor-Praxiskurs AVR-XMEGA-Mikrocontroller, Best.-Nr. CO-12 07 62
- FRANZIS Arduino-Projects-Lernpaket, Best.-Nr. CO-11 51 22
- FRANZIS Physical Computing, Best.-Nr. CO-12 21 81
- FRANZIS Lernpaket Motoren & Sensoren mit Arduino, Best.-Nr. CO-12 74 74

Preisstellung Juni 2017 – aktuelle Preise im ELV Shop

Empfohlenes Produkt	Best.-Nr.	Preis
Arduino UNO	CO-10 29 70	€ 27,95

Alle Arduino-Produkte wie Mikrocontroller-Platinen, Shields, Fachbücher und Zubehör finden Sie unter: www.arduino.elv.de

Download-Paket zum Artikel:
Die Sketche und Beispieldateien zu diesem Artikel können heruntergeladen werden unter www.elv.de: Webcode #10098