



Teil 7: I2C-Kanäle effizient ausnutzen mit dem PCA9516

Geschickt aufgeteilt

Für eine komplexe Raumklima-Messung brauchen Sie viele identische Sensoren.

Der PCA9516 ermöglicht es, diese auf einem I2C-Bus anzusteuern. Martin Mohr

Es gibt spezielle I²C-Bausteine, die nur über eine Adresse verfügen. Was machen Sie aber, wenn Sie mehrere dieser Bausteine, wie den AM2321 , in einem Projekt verwenden möchten? Das ist gar nicht so abwegig, wenn Sie das Raumklima in verschiedenen Räumen eines Hauses überwachen wollen. Die Lösung für dieses Problem stellt ein Baustein wie der PCA9516 dar: Er kann den I²C-Bus multiplexen.

Im Detail

Im Datenblatt  des Moduls findet sich die Angabe, dass es sich hier um einen 5-Channel-Hub handelt. In Bezug auf die reine Lehre ließe sich darüber streiten, ob er wirklich als Verteiler mit fünf Kanälen agiert: Er schaltet den I²C-Master-Bus auf vier unabhängige Slave-Busse durch.

Das bedeutet im Klartext, dass eine Adresse, die auf dem Master-Bus schon belegt ist, auf den Slave-Bussen nicht mehr bereitsteht. Ergo macht der PCA9516 eine Adresse nur je einmal an jedem der vier Slave-Busse des PCA9516 verfügbar – also doch eigentlich nur vier Kanäle.

Um den PCA9516, der in einem SO16-Gehäuse verbaut ist, auf einem Prototyping-Board zu verwenden, löten Sie ihn zuerst auf einen Adapter. Abbildung  1 zeigt das Pinout des Bausteins, das dabei hilft, den Versuch nachzubauen. Abbildung  2 zeigt dagegen den internen Aufbau der Hardware.

Versuch

Die Software im folgenden Beispiel setzt direkt auf die Installation aus Teil 6 dieser Reihe auf . Dieser zeigt, wie Sie

README

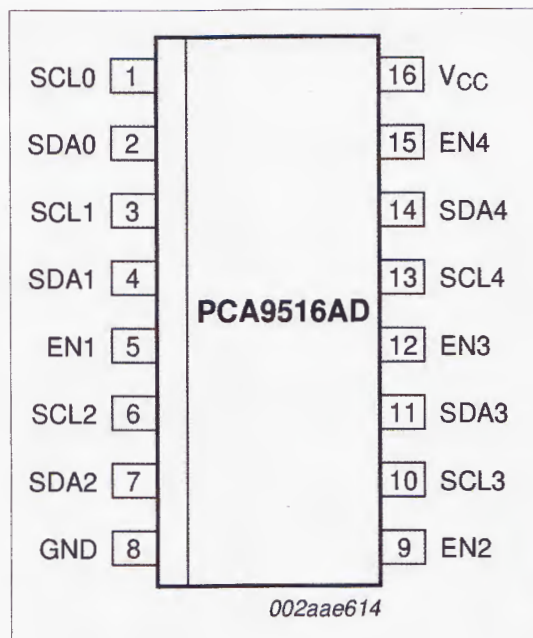
Gehen bei Ihren Projekten die Kanäle auf dem I2C-Bus zur Neige oder überschneiden sich Adressen, hilft Ihnen ein Multiplexer-Baustein wie der PCA9516 weiter.

eine komplette Java-Umgebung auf dem Raspberry Pi aufbauen. Das Testprogramm kommt in Teilen ebenfalls wieder zum Zug **3**.

Das Datenblatt gibt den Hinweis, über den Busmaster (im vorliegenden Fall also den Raspberry Pi) über I/O-Ports denjenigen I²C-Channel auszuwählen, den Sie verwenden möchten. Der Einfachheit halber realisiert das der Beispiel-Code genauso – der RasPi hat genügend dieser I/Os frei. Bei größeren Projekten böte es sich dagegen an, die I/O-Ports zum Steuern des PCA9516 selbst über den I²C-Bus anzusteuern, zum Beispiel über einen PCA9574-Baustein.

Da der Aufbau des Versuchs nicht gerade trivial ausfällt, zeigt uns der Schaltplan **4**, wie Sie alle Bausteine verschalten müssen. Damit Sie sehen, was sich gerade tut, zeigen in den Aufbau integrierte LEDs an, welcher Channel gerade aktiv ist. Diese Dioden können Sie sich sparen, wenn alles einwandfrei arbeitet.

Auf gar keinen Fall weglassen dürfen Sie jedoch die Pullup-Widerstände an den Slave-Bussen. Ohne diese kommen auf den Datenleitungen keine Logikpegel zustande. Der Raspberry Pi integriert solche Widerstände direkt auf der Platine, deshalb läuft uns dieses Problem erst jetzt über den Weg. Selbst ein nicht



1 Das Pinout des in einem SO16-Gehäuse verbauten I²C-Hubs.

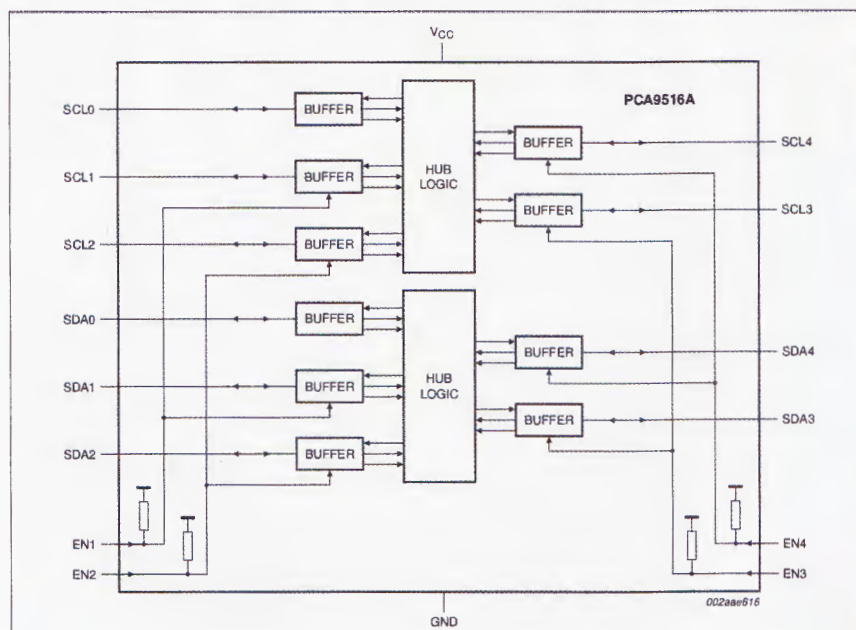
verwendeter Slave-Bus sollte diese Beschaltung bekommen. Falls sie fehlt und Sie den Bus auf den Master schalten, legt er die komplette Kommunikation lahm.

Software

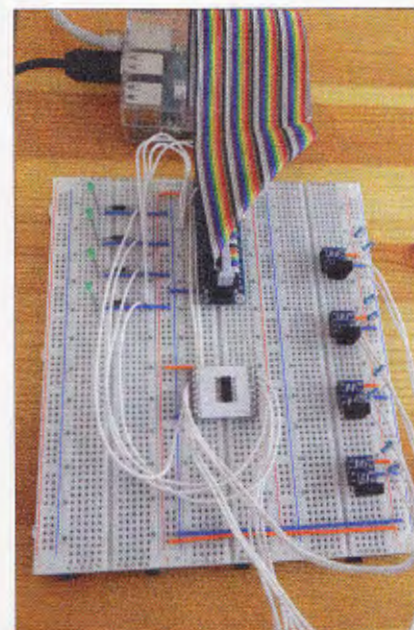
Für das Beispielprogramm kommt Code aus Teil 6 dieser Reihe zum Einsatz. Dabei verschieben Sie nun das Programm aus dem letzten Teil in die Methode `mesen()` (Listing 1). Wie Sie sehen, besteht



Schaltplan, Listings, Beispielcode aus Teil 6
RPG/i2c-7/



2 Der interne Aufbau des PCA9516-Bausteins.



3 Der Versuchsaufbau mit dem PCA9516.

Listing 1

```
import com.pi4j.io.gpio.GpioController;
import com.pi4j.io.gpio.GpioFactory;
import com.pi4j.io.gpio.GpioPinDigitalOutput;

import com.pi4j.io.gpio.PinState;

import com.pi4j.io.gpio.RaspiPin;
import com.pi4j.io.i2c.I2CBus;
import com.pi4j.io.i2c.I2CDevice;
import com.pi4j.io.i2c.I2CFactory;

public class PCA9516 {
    final static GpioController gpio = GpioFactory.getInstance();
    private static final int i2cBus = 1;
    private static final int address = 0x5c;

    public static void main(String[] args) throws InterruptedException {
        GpioPinDigitalOutput[] channel = new GpioPinDigitalOutput[4];
        channel[0] = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_00, "Channel0", PinState.LOW);
        channel[1] = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_01, "Channel1", PinState.LOW);
        channel[2] = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_02, "Channel2", PinState.LOW);
        channel[3] = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_03, "Channel3", PinState.LOW);
        for(int i=0; i<4; i++) {
            System.out.println("Channel: "+i);
            channel[i].toggle();
            messen();
            Thread.sleep(1000);
            channel[i].toggle();
        }
    }

    private static void messen() {
        byte[] wb = new byte[] {(byte)0x03, (byte)0x00, (byte)0x04};
        byte[] rb = new byte[8];
        try {
            I2CBus bus = I2CFactory.getInstance(I2CBus.BUS_1);
            I2CDevice dev = bus.getDevice(address);
            // wecken
            try{dev.read();}catch(Exception e){Thread.sleep(100);};
            // messen
            dev.read(wb, 0, 3, rb, 0, 8);
            int crc = ((rb[6] & 0xff) << 8) | rb[7] & 0xff;
            double hum = (((rb[2] & 0xff) << 8) | rb[3] & 0xff);
            double temp = (((rb[4] & 0xff) << 8) | rb[5] & 0xff);
            hum = hum/10;
            temp = temp/10;
            System.out.println("Luftfeuchtigkeit:"+hum);
            System.out.println("Temperatur:"+temp);
        }
        catch(Exception e) {
            System.out.println(e.toString());
        }
    }

} //class
```


die Methode `main()` aus einer Schleife, die nacheinander die GPIO-Pins für das Freigeben der einzelnen I²C-Kanäle auf logisch 1 setzt.

Wenn ein Kanal aktiv ist, ruft das Programm die Methode `messen()` auf und gibt die Messergebnisse aus. Danach setzt es den Channel wieder auf inaktiv. Mehr macht das kleine Testprogramm nicht. Listing 2 zeigt, wie Sie es mithilfe des Tools Pi4j kompilieren und starten.

Fazit

Interessant an der Ausgabe des Testprogramms ist die Streuung der Messwerte der Sensoren: Obwohl es so aussieht, als ob der Sensor an Kanal 3 falsch misst, liegt der Wert noch in der Toleranz von 3 Prozent für die Luftfeuchtigkeit.

Möchten Sie die Daten der einzelnen Sensoren weiterverarbeiten, empfiehlt es sich, über mehrere Messwerte zu mitteln und diesen Mittelwert für weitere Berechnungen zu verwenden. Beim Testen hatte es den Anschein, als ob der erste Messwert nach einer längeren Pause grundsätzlich sehr ungenau ausfällt. Dazu fand sich aber im Datenblatt keine konkrete Information.

Beim PCA9516 handelt es sich sicher nicht um einen Baustein, den Sie in jedem Feld-, Wald- und Wiesen-Projekt einsetzen. Kommen Sie jedoch einmal in die Verlegenheit, dass die Adressen am I²C-Bus zur Neige gehen oder sich überschneiden, springt der nützliche Hub in die Bresche und hilft relativ einfach aus der Klemme. (agr) ■

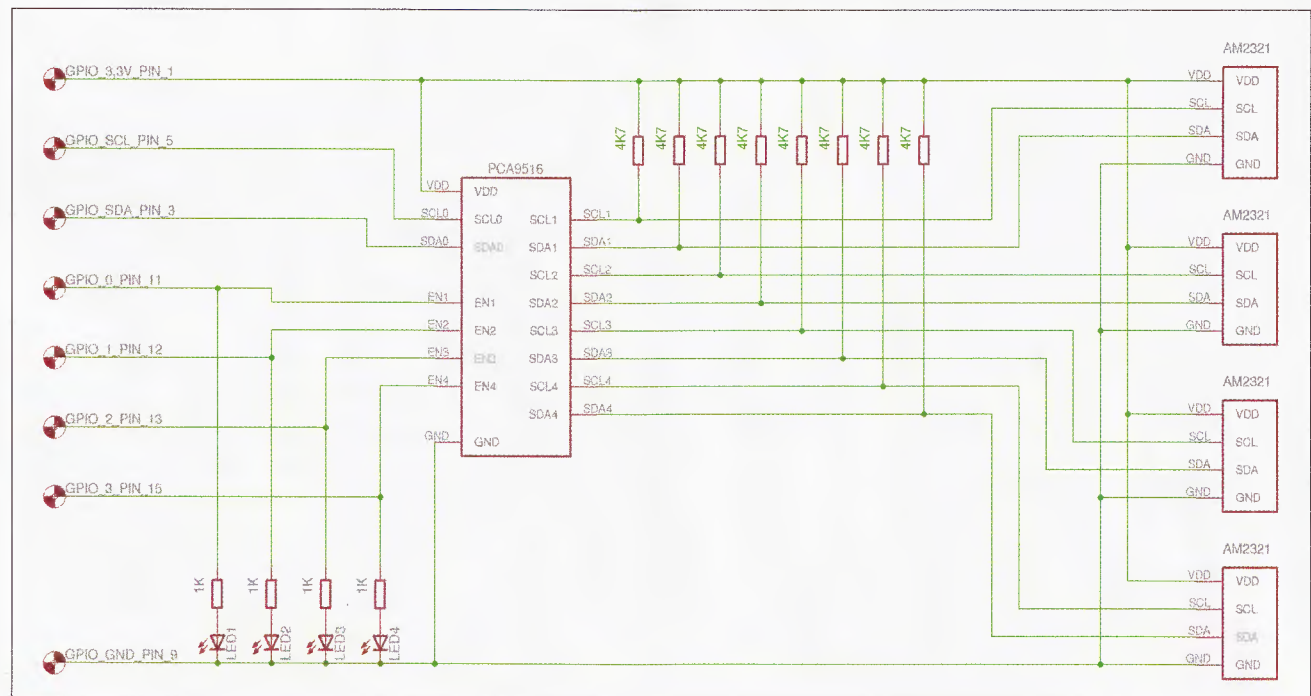


Listing 2

```
$ pi4j -c PCA9516.java
+ javac -classpath '.:classes:*:classes:/opt/pi4j/lib/*' -d . PCA9516.java
$ pi4j -r PCA9516

+ sudo java -classpath '.:classes:*:classes:/opt/pi4j/lib/*' PCA9516

Channel: 0
Luftfeuchtigkeit:37.0
Temperatur:26.4
Channel: 1
Luftfeuchtigkeit:36.0
Temperatur:26.0
Channel: 2
Luftfeuchtigkeit:37.1
Temperatur:26.1
Channel: 3
Luftfeuchtigkeit:42.8
Temperatur:26.3
```



4 Der Schaltplan für den Versuchsaufbau samt Status-LEDs.