Arduino verstehen und anwenden

Teil 25: Der One-Wire-Bus und andere Spezialsysteme

Nachdem in den letzten Artikeln die bekannten Bussysteme I²C und SPI detailliert erläutert wurden, sollen nun die bislang noch nicht erwähnten Busse vorgestellt werden:

- One-Wire
- DHT11 und andere Spezialbusse

Abschließend soll ein Vergleich der Bussysteme die Auswahl einer geeigneten Schnittstelle für eigene Projekte erleichtern.

Der One-Wire-Bus

Der One-Wire-Bus (oft auch „1-Wire") zeichnet sich insbesondere durch seinen einfachen Aufbau und seinen geringen Energieverbrauch aus. Der Bus besitzt stets einen Master, der für die Steuerung der Slaves verantwortlich ist. Eine Besonderheit des Systems ist, dass die Adressen aller Slaves bereits voreingestellt sind. Zudem sind die für den One-Wire-Bus erhältlichen Sensoren herstellerseitig kalibriert und damit sofort einsetzbar.

Aktuell kommt kein anderes allgemein verbreitetes Bussystem mit weniger elektrischer Energie aus als der One-Wire-Bus. Bei einer Versorgungsspannung von 5 V benötigen die Slaves nur wenige Mikroampere Strom. Die geringe Leistungsaufnahme kommt auch der elektromagnetischen Störausstrahlung zugute. Werden die Sensoren über abgeschirmte Leitungen angeschlossen, kann man den One-Wire als praktisch strahlungsfrei betrachten. Der extrem geringe Energieverbrauch ermöglicht sowohl ein kostengünstiges Design als auch eine sehr kompakte Bauform der einzelnen Komponenten. Ein weiterer Vorteil ist, dass bei Temperatursensoren mit niedriger Leistungsaufnahme die Eigenerwärmung kein Problem darstellt.

Auch in Hinblick auf die Anwenderfreundlichkeit verfügt der One-Wire-Bus über einen besonderen Vorzug. Immer mehr Produkte werden mit lokaler Firmware ausgestattet. Ehemals einfache Geräte wie Uhren, Beleuchtungen oder Heizungsregler benötigen heute daher regelmäßige Softwareupdates. In einem modernen sogenannten Smart Home kommen oft hundert oder mehr Geräte und Komponenten mit jeweils updatefähiger Software zum Einsatz. Die Möglichkeit eines Firmwareupdates führt aber immer öfter auch zum Verkauf noch nicht ausgereifter Produkte. Die letzte Stufe der Produktentwicklung erfolgt dann beim Kunden, der mit entsprechenden Softwareupdates zusätzlich belastet wird.

Der One-Wire-Bus dagegen erfordert keine komplexe Software, häufige Updates werden dadurch vermieden. Das Protokoll des Busses ist weitgehend in der Hardware der Komponenten festgelegt. Damit ist es praktisch unveränderbar und auf lange Lebensdauer ausgelegt.

Der Master kontrolliert den Bus

Das One-Wire-System ist, ähnlich wie I²C oder SPI, ein Master-Slave-System. Die dafür verfügbaren Sensoren und Aktoren können jedoch sehr einfach gehalten werden. Beim One-Wire-Bus besteht der Master im Allgemeinen aus einem Mikrocontroller wie beispielsweise dem ATmega328 in einem Arduino UNO. Dieser dient dann zur Steuerung des gesamten Bussystems.

Einsatzbereich für One-Wire

One-Wire eignet sich insbesondere für Sensorik-Anwendungen wie Temperaturmessung, Sammeln von Klimadaten, die Überwachung von Spannungen oder Strömen.

Bild 1: iButton

Häufig kommen hier auch sogenannte iButtons (Bild 1) zum Einsatz. Bei diesem Sensortyp handelt es sich um Offline-Sensoren, die in der Lebensmittelüberwachung und im Logistikbereich als Datenlogger Anwendung finden. Hierbei werden z. B. Luftfeuchte und Temperatur regelmäßig von den iButtons autark, d. h. ohne permanente Verbindung zum Master, protokolliert und später über einen One-Wire-Busmaster ausgelesen.

Auch bei der Kontrolle von Akkus, insbesondere bei Lithium-Ionen-Typen, kann das System eingesetzt werden. Aber auch zur Steuerung und Erfassung von Systemzuständen wie beispielsweise Tasterabfragen, Fensterkontaktüberwachung oder Informationsaustausch zwischen vernetzten Rauchmeldern kommt das One-Wire-System häufig zum Einsatz.

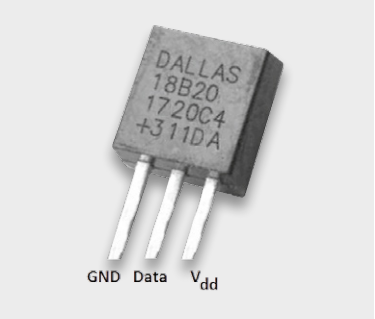

Bild 2: One-Wire-Sensor im Transistorgehäuse

Im Bereich der Gebäudeautomatisierung sind One-Wire-Sensoren wegen des sehr günstigen Preises, der einfachen Verkabelung und des extrem niedrigen Energieverbrauchs beliebt. Neben der Erfassung von Temperatur und Luftfeuchte werden auch der Luftdruck, die Luftqualität oder die Umgebungslichtstärke häufig von One-Wire-Sensoren überwacht (Bild 2).

Die Sensoren selbst übernehmen bereits komplett die Auswertung und Digitalisierung sowie die Steuerung des One-Wire-Interfaces. Bei iButtons oder den Temperatursensoren der Typenreihe DS1820 befindet sich dies alles in einem Standard-Transistorgehäuse (TO-92).

Da die Messdatendigitalisierung bereits im Sensor selbst erfolgt, entfällt jegliche Kalibrierung. Typische Sensoren dieser Kategorie erreichen eine Toleranz von nur ±0,5 °C. Sie sind damit wesentlich präziser als übliche Analogsensoren wie NTCs oder ähnliche Bauelemente.

Meist können One-Wire-Komponenten ihren Strombedarf mittels eines integrierten Kondensators aus der Datenleitung decken (sogenannter Parasitic Power Mode). Damit kommt man mit lediglich zwei Leitungen zum Sensorelement aus. Wenn man die Masseverbindung nicht zählt, hat das System also nur eine einzige aktive Leitung, wodurch sich auch der Name One-Wire erklärt.

Codierung der Slaves

One-Wire-Komponenten gibt es für viele Anwendungen, z. B. für Spannungswandlung, digitale I/O-Ports, Counter, Real Time Clocks (RTC), EEPROM-Speicher und vieles mehr. Dabei gibt es die Komponenten in unterschiedlichen Ausführungen wie z. B. im Transistorgehäuse, in verschiedenen klassischen IC-Bauformen oder eben auch als iButton.

Jedes Slave-Device besitzt eine eigene, weltweit eindeutige, 64 Bit lange Adresse. Diese wird als ROM-Code oder ROM-Identifier bezeichnet. Durch den ROM-Identifier wird gewährleistet, dass sich zwei Slaves in 56 der 64 Bits unterscheiden. Damit stehen insgesamt 2^{56} = 72.057.594.037.927.936, also über 72 Billiarden unterschiedliche Adressen zur Verfügung. Dieser Adressraum sollte auf absehbare Zeit ausreichend sein.

Spannungsversorgung von One-Wire-Komponenten

One-Wire-Chips können auf zwei verschiedene Arten mit Energie versorgt werden. Man unterscheidet zwischen Parasitär-Mode und Normal Mode.

Im parasitären Betriebsmodus sind nur die bereits erwähnten zwei Verbindungen zum jeweiligen Chip erforderlich: ein Datenanschluss und die Masseleitung. Die Datenleitung muss typischerweise über einen 4,7-kΩ-Pull-up-Widerstand mit der Spannungsversorgung des verwendeten Controllers verbunden sein. Wenn diese Leitung im HIGH-Zustand ist, zieht der Chip Strom aus der Leitung, um einen internen Kondensator aufzuladen. Dieser Strom ist in der Regel sehr gering, kann aber bei einer Temperaturumwandlung oder beim Schreiben von EEPROM-Daten auf bis zu 1,5 mA ansteigen. Für eine Temperaturumwandlung des DS18S20-Sensors sind hierfür bis zu 750 ms erforderlich. Der Master ist während dieser Zeit blockiert. Die Ausgabe von Befehlen an andere Busteilnehmer oder das Abrufen von Daten ist innerhalb dieser 750 ms nicht möglich. In der One-Wire-Bibliothek zur Arduino-Programmierung ist diese Wartezeit entsprechend berücksichtigt.

Bei kurzen Leitungslängen funktioniert der Parasitär-Modus im Allgemeinen zufriedenstellend. Falls man allerdings an höchstmöglicher Betriebssicherheit interessiert ist, sollte man den zweiten Betriebsmodus, den sogenannten Normal Mode, in Betracht ziehen.

Der Normalmodus ist die prinzipiell zuverlässigere Betriebsart. Die zusätzliche separate Spannungsversorgung führt zu einem stabileren Gesamtsystem, natürlich auf Kosten eines etwas komplexeren Aufbaus. Der 4,7-kΩ-Pull-up-Widerstand ist hier genauso erforderlich wie beim Parasitär-Mode. Da jetzt aber die Datenleitung frei verfügbar ist, kann der Mikrocontroller den Zustand der Busteilnehmer kontinuierlich abfragen. Auf diese Weise kann eine neue Datenabfrage gestartet werden, sobald die alte beendet ist. Im Gegensatz zum Parasitärmodus ist hier keine Wartezeit von jeweils 750 ms erforderlich.

Hinweis zum Pull-up-Widerstand

Bei größeren Netzwerken können kleinere Pull-up-Widerstände vorteilhaft sein. Eine Messung an der Datenleitung des Systems schafft hier Klarheit. Sind die Signalpegel zu gering, empfiehlt es sich, die Pull-up-Widerstände anzupassen.

Preisgünstig und präzise: der DS18(S/B)20

Eines der bekanntesten und meistgenutzten Bauelemente mit One-Wire-Interface ist der Temperatursensor DS1820. Das Bauteil zeichnet sich durch seinen günstigen Preis und durch seine kleine Bauform aus. Es beinhaltet einen hochpräzisen Temperatursensor sowie eine Auswerteelektronik, die es ermöglicht, den Sensor via One-Wire-Bus direkt an einen Mikrocontroller bzw. den Arduino anzuschließen. Damit wird die digitale Temperaturmessung zum Kinderspiel. Der DS18B20 ist in drei Varianten verfügbar: 8-Pin SO (150 mils), 8-Pin µSOP und 3-Pin TO-92.

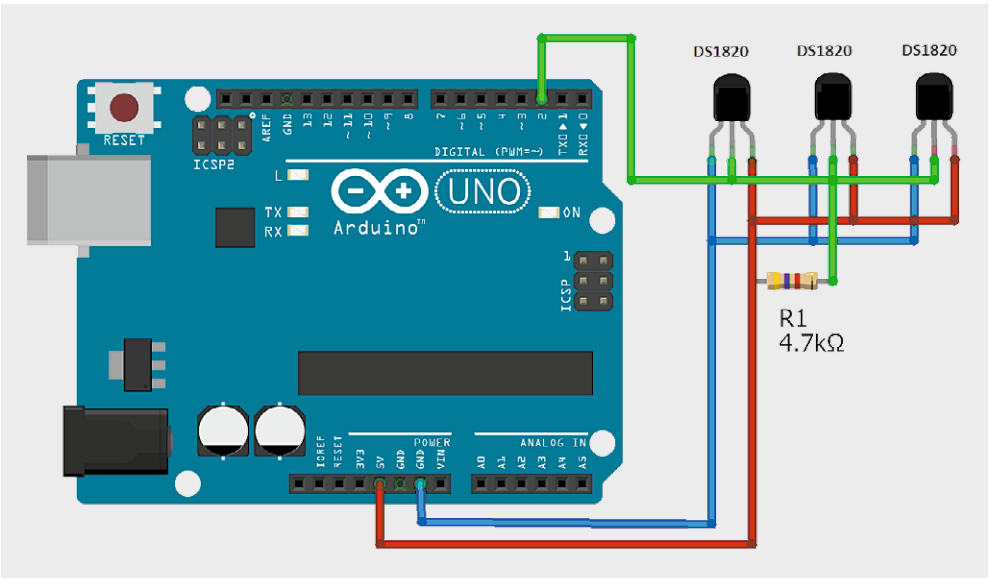

Bild 3: Anschluss von DS1820-Temperatursensoren im Normal Mode

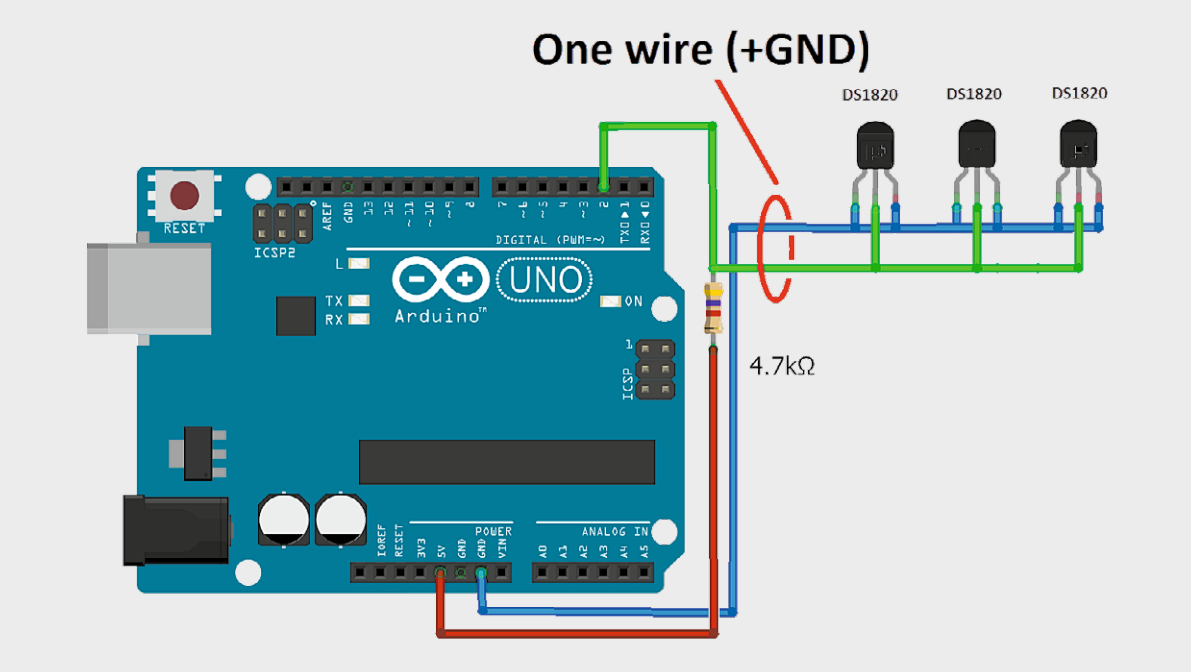

Bild 4: Anschluss von DS1820-Temperatursensoren im Parasitär-Mode

Die klassische Version ist das bekannte Transistorgehäuse TO-92, welches bei Arduino-Anwendungen am häufigsten zum Einsatz kommt. Diese Version ist auch in einem wasserdichten Gehäuse erhältlich. Man kann damit im Außenbereich, einem Aquarium oder im Kühl- bzw. Gefrierschrank problemlos Temperaturen erfassen und überwachen.

Auch zur Temperaturkontrolle in digitalen Schaltnetzen werden die DS1820-Sensoren gerne verwendet. Hier ist es wieder von Vorteil, dass das Signal nicht erst über einen AD-Wandler digitalisiert werden muss, sondern direkt von einem Controller ausgelesen werden kann.

Der Sensor arbeitet mit einer Betriebsspannung zwischen 3 und 5,5 V. Der Messbereich reicht von -55 bis +125 °C. Über diesen gesamten Temperaturbereich wird eine Messgenauigkeit von ±2 °C garantiert. Im eingeschränkten Temperaturintervall von –10 bis +85 °C wird sogar eine Genauigkeit von 0,5 °C erreicht.

Der Sensor arbeitet mit bis zu 12 Bit langen Daten und erreicht so eine Temperaturauflösung von weniger als 0,1 °C. Auch am Arduino kann der DS1820 sowohl mit normaler als auch mit parasitärer Spannungsversorgung betrieben werden. Die beiden Varianten sind in Bild 3 und Bild 4 dargestellt.

Auf der Softwareseite ist unter https://github.com/milesburton/Arduino-Temperature-Control-Library eine hilfreiche Bibliothek verfügbar. Diese gestattet das einfache Auslesen der Sensorwerte, auch wenn mehrere DS1820-Bausteine an einem Bus betrieben werden. Ein Beispielsketch dazu findet sich im Download-Paket zu diesem Artikel.

Bild 5 zeigt die zugehörige Ausgabe der Messwerte auf dem seriellen Monitor. Für jeden Sensor werden der ROM-Code, der Chip-Typ, die empfangenen Daten, die Prüfsumme und der aktuelle Temperaturmesswert in Grad Celsius ausgegeben. Im Bedarfsfall kann das Programm natürlich so abgeändert werden, dass nur noch die Temperaturen der jeweiligen Sensoren angezeigt werden.

Proprietäre Bussysteme

Neben den bekannten und weit verbreiteten Bussen existieren noch eine Reihe weiterer Systeme. Diese sind oftmals den gängigen Bussen sehr ähnlich. Dennoch unterscheiden sie sich in verschiedenen Details, sodass sie mit den anderen Varianten nicht kompatibel sind.

Natürlich können an dieser Stelle nicht alle existierenden Systeme behandelt werden. Stattdessen soll exemplarisch ein spezieller Bus vorgestellt werden, der insbesondere im Arduino-Umfeld weite Verbreitung gefunden hat. Er wird bei den häufig eingesetzten Temperatur- und Luftfeuchte-Sensoren DHT11 bzw. DHT22 verwendet.

Klimamessung mit pulsweitenmodulierten Bussignalen

DHT11-/22-Bausteine verfügen intern über einen kalibrierten Temperatur- und einen Feuchtigkeitssensor sowie einen digitalen Signalausgang. Die Bauelemente sind trotz ihres moderaten Preises für ihre hohe Zuverlässigkeit und gute Langzeitstabilität bekannt. In Verbindung mit einem Arduino kann so ein kostengünstiges Klimaüberwachungssystem realisiert werden. Jeder DHT-Sensor ist herstellerseitig kalibriert, so-

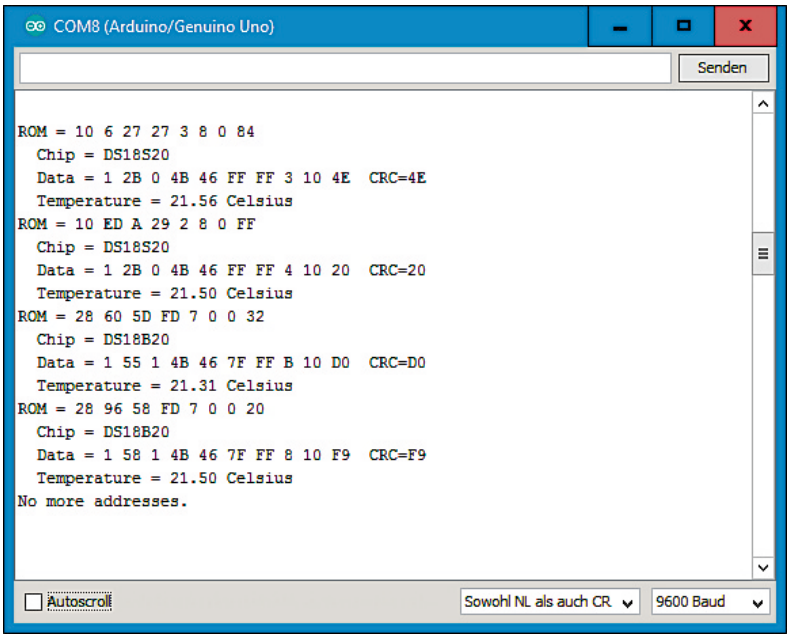

Bild 5: Datenausgabe von vier verschiedenen Sensoren auf dem seriellen Monitor

dass auch bei der Feuchtigkeitsmessung eine große Genauigkeit erreicht wird. Die Kalibrierkoeffizienten werden im internen Speicher des Bausteins abgelegt, sodass der Anwender keine eigene Kalibrierung vornehmen muss. Die serielle Schnittstelle macht die Systemintegration schnell und einfach. Die geringe Baugröße sowie der minimale Stromverbrauch machen die Sensoren zum Mittel der Wahl für viele Smart Home Anwendungen.

Nachdem der Sensor mit der Versorgungsspannung verbunden wurde, sollte innerhalb einer Sekunde keine Anweisung an den Sensor gesendet werden, um zu vermeiden, dass Daten gesendet werden, solange die Sensorelektronik noch keinen stabilen Zustand erreicht hat. Tabelle 1 fasst die technischen Daten der DHT11-/22-Sensoren zusammen.

Tabelle 1

	DHT11	DHT22
Temperaturmessbereich	0 bis 50 °C	-40 bis +125 °C
Temperaturmessgenauigkeit	±2 °C	±0,5 °C
Empfindlichkeitsbereich	20–80 %	0–100 %
Luftfeuchtemessgenauigkeit	±5 %	±3 %
Datenrate	1 Messung/s	2 Messungen/s
Betriebsspannung	3–5 V	3–5 V
Maximale Stromaufnahme	2,5 mA	2,5 mA

Buskommunikation

Für die Kommunikation mit den DHT11-Sensoren wird ein spezielles Datenformat verwendet. Ein Datenübertragungsprozess dauert ca. 4 ms. Die Datenpakete umfassen 40 Bit und werden im folgenden Format übertragen:

- 8 Bit Luftfeuchtedaten, Intergeranteil
- 8 Bit Luftfeuchtedaten, Nachkommastellen
- 8 Bit Temperaturdaten, Intergeranteil
- 8 Bit Temperaturdaten, Nachkommastellen
- 8 Bit Prüfsumme

Die Prüfsumme dient zur Sicherstellung einer korrekten Datenübertragung.

Das Zeitdiagramm in Bild 6 zeigt das Datenübertragungsprotokoll zwischen einem Mikrocontroller und dem DHT11-Sensor.

Man erkennt, dass der Controller die Datenübertragung durch Ausgabe eines „Start-Signals" beginnt. Dazu muss der Controller-Pin als Ausgang konfiguriert werden. Der Controller zieht zuerst die Datenleitung für mindestens 18 ms auf LOW-Potential, dann für die nächsten 20–40 µs auf HIGH. Im Anschluss wird die Leitung freigegeben. Darauf antwortet der Sensor, indem er die Leitung für 80 µs auf LOW zieht, gefolgt von einem logischen HIGH-Pegel, der ebenfalls 80 µs andauert. Der Controller-Pin muss nach dem Start-Signal also als Eingang definiert werden. Der Sensor sendet dann 40 Bits in 5-Byte-Paketen in die Datenleitung. Das höchstwertige Bit wird dabei zuerst gesendet („MSB first").

Um eine logische Null zu senden, zieht der Sensor zuerst die Datenleitung für 50 µs auf LOW. Dann folgt ein HIGH-Signal für 26–28 µs. Für eine logische Eins folgt auf das 50 µs lange LOW-Signal ein 70 µs langer HIGH-Pegel. Hier kommt also ein pulsweitenmoduliertes Signal zum Einsatz, ähnlich wie es auch beim bekannten Zeitzeichensender DCF77 verwendet wird.

Zwei oder mehr Sensoren an einem Controller

Oft sollen nicht nur an einem einzigen Messpunkt Daten gesammelt werden. Vielmehr möchte man gleichzeitig mehrere Sensoren an verschiedenen Orten einsetzen. So kann etwa die Luftfeuchtigkeit sowohl in mehreren Kellerräumen als auch auf dem Dachboden erfasst werden.

Hier kommt der Vorteil eines digitalen Sensors zum Tragen. Während bei einem Standard-Arduino nur sechs Analogeingänge zur Verfügung stehen, kann man auf immerhin 14 digitale I/O-Ports zurückgreifen. Somit könnte man bis zu 14 verschiedene DHT11-Sensoren an einem einzelnen Arduino betreiben. Bei einem MEGA2560 sind es sogar noch weit mehr.

Will man also Messwerte in verschiedenen Räumen gleichzeitig erfassen, muss man sich nur die nötige Anzahl von Sensoren beschaffen – dann steht einer umfangreichen Datenaufnahme nichts mehr im Wege.

Dank der DHT-Bibliothek muss sich der Anwender nicht um das Busprotokoll im Detail kümmern. Die passende Library kann unter

https://github.com/adafruit/DHT-sensor-library

aus dem Internet geladen werden. Der folgende Sketch zeigt, wie mit der Bibliothek zwei DHT11-Sensoren simultan ausgelesen werden können (der vollständige Sketch findet sich auch im Download-Paket zu diesem Artikel).

Dazu muss zunächst die Bibliothek über eine entsprechende Include-Anweisung eingebunden werden. Danach werden die beiden Sensoren initialisiert und die Ports für die Datensignale festgelegt. Im Setup wird die serielle Schnittstelle gestartet. Anschließend erfolgt eine erste Abfrage der Sensoren. Unter Verwendung der Prüfsumme wird festgestellt, ob alle am Arduino angeschlossenen DHT-Sensoren korrekt antworten. In der Hauptschleife werden dann nur noch über die Read-Funktion der Library im Sekundentakt die aktuellen Werte abgefragt. Zum Abschluss werden diese in tabellarischer Form auf dem seriellen Monitor ausgegeben. Selbstverständlich kann der Sketch auch auf drei und mehr Sensoren erweitert werden.

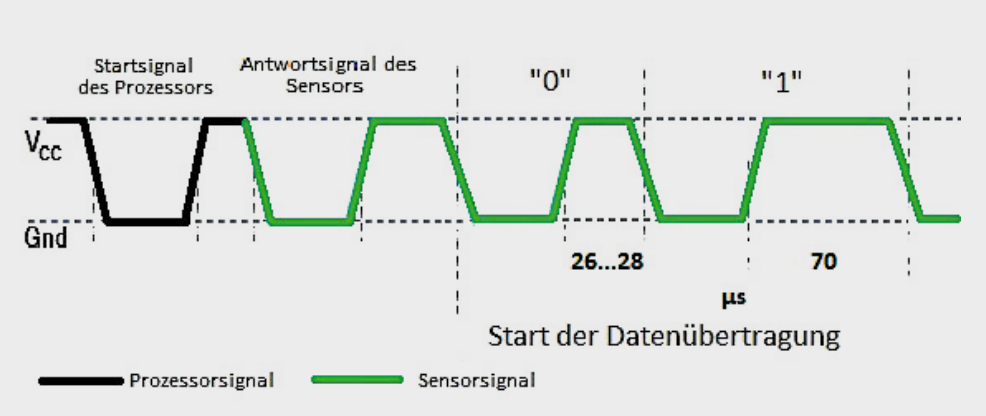

Bild 6: Pegelverlauf auf dem DHT11-Bus

```
// DHT11_multiple_sensors.ino

#include <dht11.h>
DHT11 DHT11_1; DHT11 DHT11_2;

#define DHT11PIN1 2
#define DHT11PIN2 3

int chk1, chk2;

void setup()
{  Serial.begin(9600);
   Serial.print("DHT11_1: ");
     chk1 = DHT11_1.read(DHT11PIN1);
     switch (chk1)
     { case DHTLIB_OK:
       Serial.print("OK,\t");
       break;
       case DHTLIB_ERROR_CHECKSUM:
       Serial.print("Checksum error,\t");
    }
    Serial.print("DHT11_2: ");
     chk2 = DHT11_2.read(DHT11PIN2);
     switch (chk2)
     { case DHTLIB_OK:
       Serial.print("OK,\t");
       break;
       case DHTLIB_ERROR_CHECKSUM:
       Serial.print("Checksum error,\t");
    }
    Serial.println();
}

void loop()
{  int chk1 = DHT11_1.read(DHT11PIN1);
   if (chk1 == DHTLIB_ERROR_CHECKSUM) Serial.print("Checksum error,\t");
   else
   { Serial.print("Temperature 1: "); Serial.print(DHT11_1.temperature);
     Serial.print(" C"); Serial.print("\t");
     Serial.print("Humidity 1: "); Serial.print(DHT11_1.humidity);
     Serial.print(" %");
   }
   Serial.print("\t");
   int chk2 = DHT11_2.read(DHT11PIN2);
   if (chk2 == DHTLIB_ERROR_CHECKSUM) Serial.print("Checksum error,\t");
   else
   { Serial.print("Temperature 2: "); Serial.print(DHT11_2.temperature);
     Serial.print(" C"); Serial.print("\t");
     Serial.print("Humidity 2: "); Serial.print(DHT11_2.humidity);
     Serial.print(" %");
   }
   Serial.println(); delay(1000);
}
```

Bild 7 zeigt den zugehörigen Hardware-Aufbau. Für ein funktionsfähiges System müssen die einzelnen Sensoren lediglich mit der Versorgungsspannung des Arduinos und einem I/O-Pin verbunden werden. Aufgrund der äußerst geringen Leistungsaufnahme ist die Versorgung der DHT11-Bausteine unkritisch. Bei einer größeren Anzahl von Messwertaufnehmern sollten jedoch zur Sicherheit Abblockkondensatoren (z. B. 100 nF) eingefügt werden, insbesondere wenn längere Verbindungsleitungen im Spiel sind. Bild 8 zeigt die Ausgabe der Werte auf dem seriellen Monitor. Dabei zeigt der Sensor 1 Werte in einem leicht erwärmten und mit Wasser gefüllten Glas an. Sensor 2 liefert zum Vergleich die Daten des umgebenden Raums.

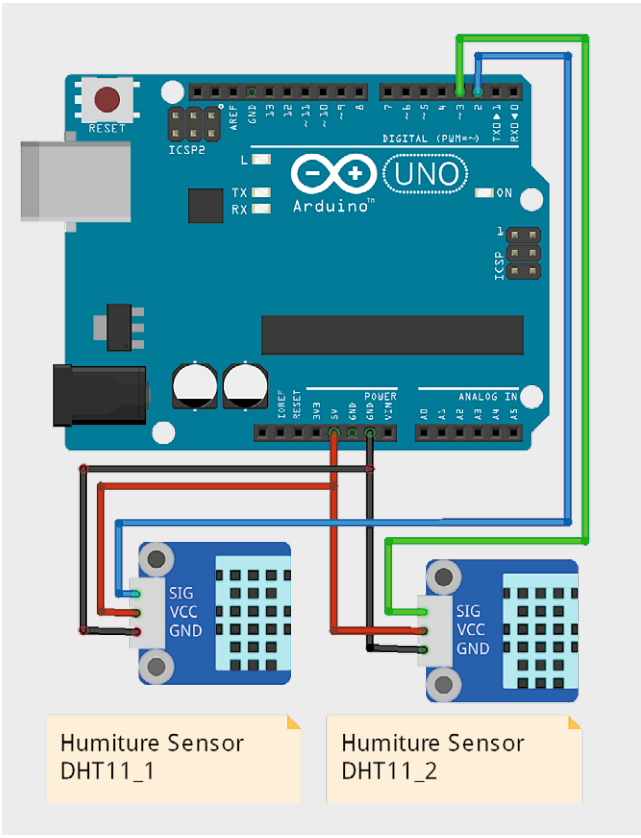

Bild 7: Zwei DHT11-Sensoren am Arduino UNO

Vergleich der Bussysteme

Zum Abschluss der Beiträge zum Thema Mikrocontroller-Busse sollen die einzelnen Systeme im Folgenden miteinander verglichen werden.

Jeder Bus hat seine spezifischen Vor- und Nachteile. Der Anwender steht daher häufig vor dem Problem, welcher Variante er den Vorzug geben soll. Stehen für eine bestimmte Anwendung nur Bauelemente mit einem speziellen Interface zur Verfügung, hat man natürlich nicht die Qual der Wahl, sondern ist auf das betreffende System festgelegt. Für viele Applikationen sind aber mehrere Lösungen verfügbar. In diesem Falle muss man nach den einzelnen Kriterien wie

- erforderlicher Datenrate
- Anzahl der Signalleitungen
- Versorgungsspannungen
- Kosten

und anderen Kriterien eine optionale Auswahl treffen. Die Tabelle 2 kann dabei wertvolle Dienste leisten. Sie zeigt beispielsweise, dass der SPI-Bus häufig die erste Wahl ist, wenn es um schnelle Datenübertragung geht. I²C kommt dagegen zum Einsatz, wenn eine minimale Anzahl von Leitungen wünschenswert ist. Mithilfe der Tabelle sollte die Auswahl eines Bussystems für eine vorgegebene Anwendung kein großes Problem mehr darstellen. Zusammen mit den letzten Artikeln zum Thema Bussysteme sollte der Anwender also in der Lage sein, die gängigsten Probleme zu diesem Thema eigenständig und effizient zu lösen.

Ausblick

Mit diesem Beitrag wird das Thema Bussysteme beendet. Neben den klassischen Systemen wie I²C

Tabelle 2 – Vergleich der Bus-Systeme

	SPI	I²C	One-Wire
Anwendungsbereich	ADCs, DACs, Sensoren	Unterhaltungselektronik, Sensoren, Displays	Sensoren und Messwandler
Vorteile	schnellstes Bussystem Empfangshardware kann aus einfachem Schieberegister bestehen	weit verbreitet viele verschiedene Hersteller	energieeffizient, Mikrowatt-Sensoren möglich, preisgünstig
Nachteile	viele Signalleitungen erforderlich eigene Leitung für jeden Slave notwendig	langsam IC-interne Hardware aufwendig und teuer	langsam
Standardisierung	keine exakte Standardisierung	Philips „THE I2C-Bus SPECIFICATION Version 2.1"	herstellerspezifisch
Hardwareaufwand	4 Signale: MISO, MOSI, SCLK, SS	2 Signale SCL, SDA	„Ein-Draht-Verbindung": Versorgungsspannung kann auch als Datenleitung dienen
Overhead	gering: ein Taktzyklus für jedes übertragende Bit sowie Start und Stopp	hoch: zusätzlich zu den Daten muss auch noch die Adresse übertragen werden	unkritisch
Taktgeschwindigkeit	1 Bit/s bis ca. 5 Megabit/s	1 Bit/s bis 3,4 Megabit/s	Kilobitbereich
Maximale Buslänge	ca. 1 m, abhängig von der Geschwindigkeit und der notwendigen Flankensteilheit	mehrere Meter möglich, Leitungskapazität sollte 400 pF nicht überschreiten	bis zu 100 m möglich
Maximale Anzahl von Busteilnehmern	abhängig von der Anzahl der Chip-Selektleitungen	maximal 1136	theoretisch unbegrenzt
EMV-Probleme	kritisch steile Flanken, hohe HF-Ausstrahlung	gering Signalflanken kontrollierbar	gering, da langsam und nur sehr kleine Signalenergie notwendig
Zukunftsaussichten	einfaches Protokoll kein Ersatz in Sicht	in der Unterhaltungselektronik unverzichtbar	bei hohem Kostendruck unverzichtbar

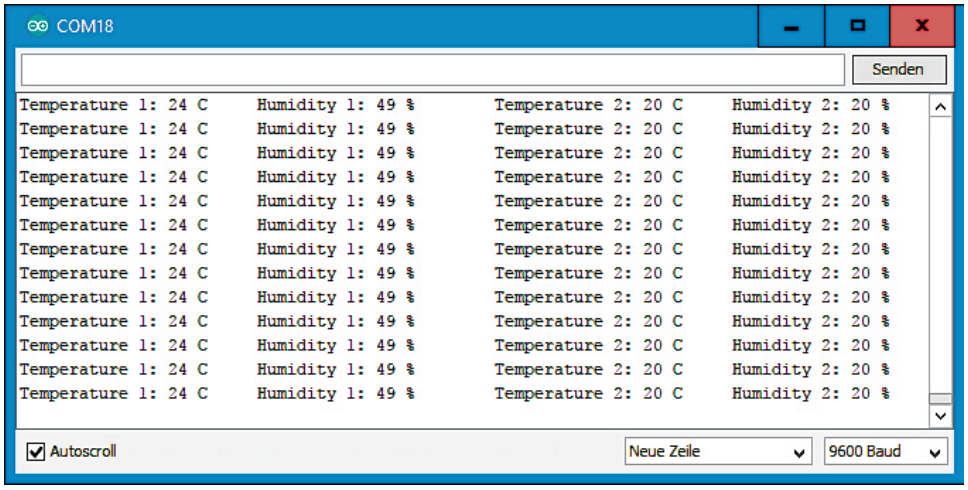

Bild 8: Sensorwerte auf dem seriellen Monitor

und SPI wurden auch weniger bekannte, dafür aber besonders einfache Busse vorgestellt. In weiteren Artikeln wird immer wieder auf diese Informationen zurückgegriffen werden, da Bussysteme zu den zentralen Bestandteilen moderner Elektronikanwendungen zählen.

So wird im nächsten Artikel die drahtlose Signalübertragung im Fokus stehen. Auch die dafür eingesetzten Funkmodule werden häufig z. B. über einen SPI-Bus angesteuert. Signalbezeichnungen wie MISO, MOSI und SCK werden dabei immer wieder auftauchen. Wer diesen und die letzten drei Artikel zu dieser Serie aufmerksam durchgearbeitet hat, sollte damit aber keine Verständnisprobleme mehr haben. 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. CQ-09 73 52
- Elektor-Praxiskurs AVR-XMEGA-Mikrocontroller, Best.-Nr. CQ-12 07 62
- FRANZIS Physical Computing, Best.-Nr. CQ-12 21 81
- FRANZIS Lernpaket Motoren & Sensoren mit Arduino, Best.-Nr. CQ-12 74 74

Download-Paket zum Artikel:

Sketche und Beispieldateien zu diesem Artikel können kostenlos heruntergeladen werden unter www.elv.de: Webcode #10140

Preisstellung Oktober 2017 – aktuelle Preise im ELV Shop

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

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