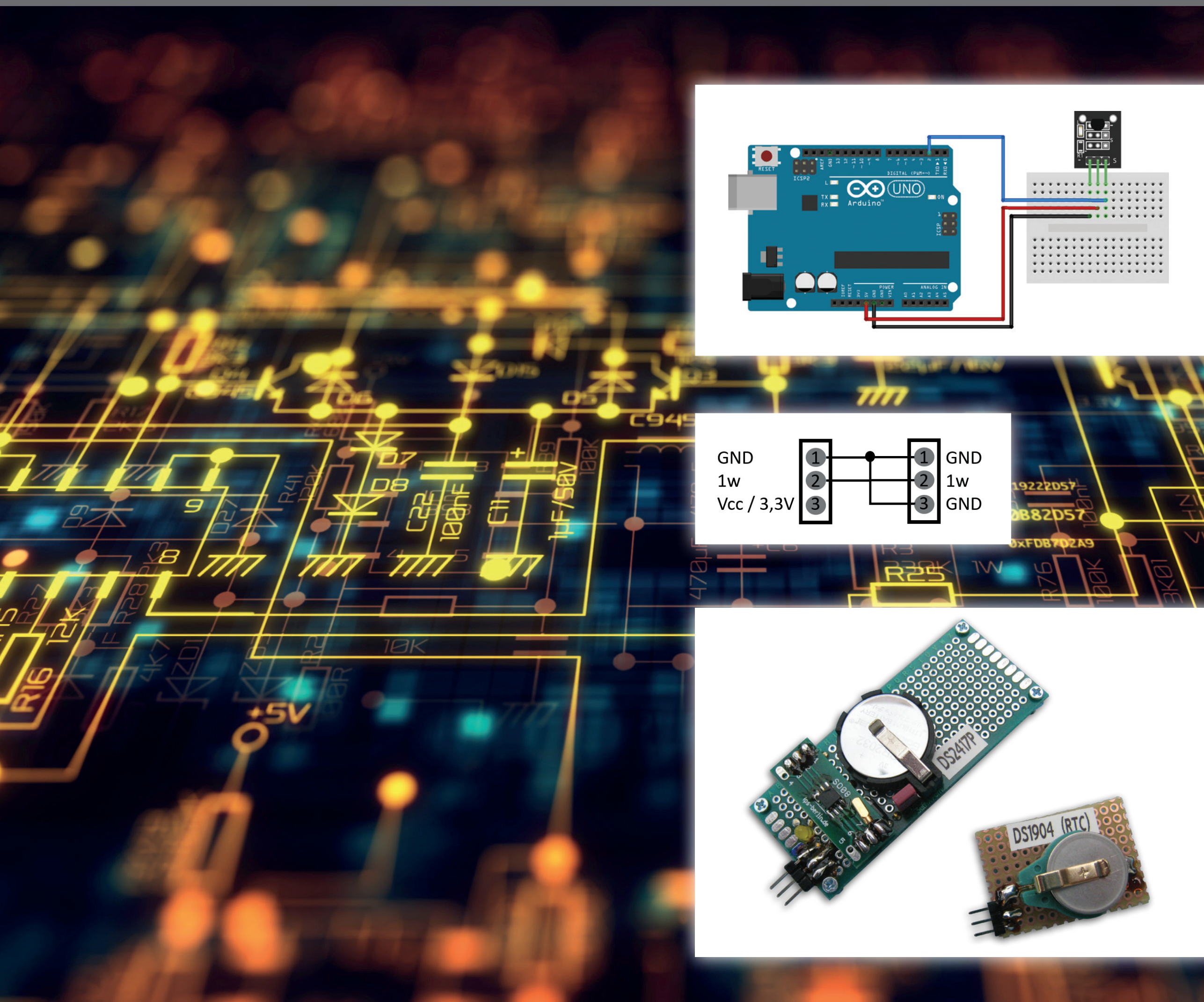
Das 1-Wire-Praxisbuch



Miroslav Cina

elektor

LEARN DESIGN SHARE

Das 1-Wire-Praxisbuch

Miroslav Cina

elektor

LEARN 〉DESIGN〉 SHARE

1. Auflage 2020

Umschlaggestaltung: Elektor, Aachen

Satz und Aufmachung: D-Vision, Julian van den Berg | Oss (NL)
Druck: WILCO, Amersfoort, Niederlande
Printed in the Netherlands

● **ISBN 978-3-89576-350-2**
EISBN: 978-3-89576-368-7
EPUB: 978-3-89576-369-4

Elektor-Verlag GmbH, Aachen
www.elektor.de

Kapitel 1 ● Die OneWire-Welt 13

Kapitel 2 ● OneWire Theorie 16

2.1. OneWire Netzwerktopologie 16

2.1.1. Lineare Topologie 16

2.1.2. Variationen der Linearen Topologie 18

2.1.3. Geschaltete Netzwerke 19

2.2. Protokoll-Konventionen 20

2.2.1. ROM-Code 20

2.2.2. Technische Ebene 21

2.2.3. Kommunikationsschritte 22

2.2.4. Nicht zu vergessen - CRC 23

2.2.5. ROM-Befehle 26

2.2.5.1. Read ROM [33h] 27

2.2.5.2. Match ROM [55h] 27

2.2.5.3. Search ROM [F0h] 27

2.2.5.4. Skip ROM [CCh] 27

2.3. Typische Einsatzgebiete 27

Kapitel 3 ● Die Brücke 29

3.1. DS2482-100 30

3.2. DS2482-800 31

3.3. DS2484 33

Kapitel 4 ● OneWire Praxis 36

4.1. ROM-ID Chips 39

4.1.1. Überblick 40

4.1.2. DS2401 (Familienkode: 01h) 41

4.1.2. DS2411 (Familienkode: 01h) 42

4.1.3. DS1990A (Familienkode: 01h) 43

4.2. Thermometer Chips 43

4.2.1. Überblick 44

4.2.2. Temperaturkodierung und der 2-er-Komplement 46

4.2.3. DS18S20 (Familienkode: 10h) 48

4.2.3.1. Funktionalität 48

4.2.3.2. Temperaturkodierung und Umrechnung 50
4.2.3.3. Konfiguration 53
4.2.3.4. Temperatur messen 53
4.2.3.5. Alarmfunktion 54
4.2.4. DS18B20 (Familienkode: 28h) 55
4.2.4.1. Funktionalität 55
4.2.4.2. Temperaturkodierung und Umrechnung 56
4.2.4.3. Konfiguration 59
4.2.4.4. Temperatur messen 60
4.2.4.5. Alarmfunktion 60
4.3. GPIO 60
4.3.1. Überblick 61
4.3.2. DS2406 (Familienkode: 12h) 62
4.3.2.1. Funktionalität 62
4.3.2.2. PIO Lesen 66
4.3.2.3. PIO Ansteuern 66
4.3.3. DS2413 (Familienkode: 3Ah) 67
4.3.3.1. Funktionalität 67
4.3.3.2. PIO Lesen 69
4.3.3.3. PIO Ansteuern 69
4.3.4. DS2408 (Familienkode: 29h) 70
4.3.4.1. Funktionalität 70
4.3.4.2. Daten aus dem DS2408 Lesen 73
4.3.4.3. Daten in der PIO schreiben 73
4.3.4.4. Activity Latches zurücksetzen 73
4.4. RTC/Timer 73
4.4.1. DS2417 (Familienkode: 27h) 74
4.4.2. DS1904 (Familienkode: 24h) 80
4.5. EEPROM 81
4.5.1. Überblick 81
4.5.2. DS2431 / DS28E07 (Familienkode: 2Dh) 82
4.5.2.1. Speicher-Organisation 82

4.5.3.2. Daten lesen 85
4.5.3.3. Daten schreiben 85
4.5.2. DS24B33 / DS1973 (Familienkode: 23h) 86
4.6. EPROM 86
4.6.1. Überblick 87
4.6.2. DS2502 / DS2502-Exx (Familienkode: 09h / 89h) 88
4.6.2.1. Speicherorganisation 89
4.6.2.2. Daten lesen 90
4.6.2.3. Daten schreiben 90
4.6.3. DS2505 (Familienkode: 0Bh) 91
Kapitel 5 • OneWire Softwarebeispiele 92
5.1. Überblick: Was kann man damit tun? 93
5.2. Hardware des OneWire-DemoBoard2020 94
5.3. OneWire-DemoBoard2020: Hardware der Lite Version 95
5.4. OneWire-Shield für den Arduino UNO 96
5.5. Breakout Boards 101
5.6. Firmware des OneWire-DemoBoard2020 103
5.6.1. Einleitung 103
5.6.2. HW-Implementierung des OneWire Protokolls 105
5.6.2.1. OneWire Reset 107
5.6.2.2. OneWire Write 107
5.6.2.3. OneWire Write mit Strong Pull-Up 108
5.6.2.4. OneWire Write Bit 109
5.6.2.5. OneWire Read 110
5.6.3. Software-Implementierung des OneWire-Protokolls 110
5.6.3.1. Definitionen und Verzögerungsroutinen 111
5.6.3.2. OneWire Reset 112
5.6.3.3. OneWire Write Byte 113
5.6.3.4. OneWire SPU-Write 117
5.6.3.5. OneWire Write Bit 117
5.6.3.6. OneWire Read 117
5.6.4. Grundlagen und Implementierung des I^2C Protokolls 119

5.6.4.1. Logische Ebene 121
5.6.4.2. Technische Grundlagen 125
5.7. OneWire Arduino Sketch 130
5.7.1. Einleitung 131
5.7.2. Initialisierung / Setup 132
5.7.3. OneWire Bibliothek 133
5.7.4. I²C-Bibliothek 134
5.7.5. Die Hauptschleife 135
5.8. DS2401 / DS2411 / DS1990A 136
5.8.1. Hardware 137
5.8.2. Bedienung mit DemoBoards 138
5.8.2.1. Übersicht 138
5.8.2.2. Chip Funktion 1 138
5.8.3. Software 139
5.8.3.1. ROM-ID auslesen 139
5.8.3.2. I²C-EEPROM auslesen 141
5.8.3.3. I²C-EEPROM beschreiben 143
5.9. Temperatursensoren 145
5.9.1. Hardware 146
5.9.2. Bedienung mit DemoBoard 146
5.9.2.1. Übersicht 146
5.9.2.2. Chip-Funktion 1 147
5.9.3. Software 148
5.9.3.1. Temperatur Auslesen 148
5.9.3.2. Temperatur Messen 150
5.10. GPIO - DS2406 151
5.10.1. Hardware 151
5.10.2. Bedienung mit DemoBoard 153
5.10.3. Software 155
5.10.3.1. GPIO lesen 155
5.10.3.2. GPIO schreiben - Option 1 (Byte) 157
5.10.3.3. GPIO schreiben - Option 2 (Bit) 159

5.11. GPIO - DS2408. 161
5.11.1. Hardware 162
5.11.2. Bedienung mit dem DemoBoard. 163
5.11.3. Software. 163
5.11.3.1. Activity Latches zurücksetzen 163
5.11.3.2. GPIO lesen 164
5.11.3.3. Status Register lesen. 166
5.11.3.4. Daten an GPIO schicken. 167
5.12. GPIO - DS2413. 168
5.12.1. Hardware 169
5.12.2. Bedienung mit dem DemoBoard. 169
5.12.3. Software. 170
5.12.3.1. GPIO lesen 170
5.12.3.2. GPIO ansteuern 171
5.13. RTC. 173
5.13.1. Hardware 174
5.13.2. Bedienung mit den DemoBoards 175
5.13.3. Software. 176
5.13.3.1. Chip Oszillator starten 177
5.13.3.2. Zählerinhalt auslesen. 178
5.13.3.3. Zählerinhalt ändern 179
5.13.3.4. Die Zeitberechnung 181
5.14. EEPROM. 182
5.14.1. Hardware 182
5.14.2. Bedienung mit den DemoBoards 183
5.14.2.1. Device-Funktion 1 - Daten auslesen 183
5.14.2.2. Device-Funktion 2 - Daten überschreiben. 184
5.14.3. Software. 185
5.14.3.1. EEPROM lesen. 185
5.14.3.2. EEPROM beschreiben 187
5.14.3.3. Scratchpad beschreiben 188

5.14.3.3. Scratchpad auslesen 190
5.14.3.4. Scratchpad in den EEPROM-Bereich kopieren 192
5.15. EPROM (OTP) 193
5.15.1. Hardware 194
5.15.2. Bedienung mit den DemoBoards 194
5.15.3. Software 196
5.15.3.1. EPROM-Speicher lesen 196
5.15.3.2. Kommentar zum E48/E64 199
5.16. DS2413P als OneWire-zu-I²C-Brücke 202
5.16.1. Hardware 202
5.16.1.1. Die Brücke mit dem DS2413P 203
5.16.1.2. Hardware - LED-Board mit PCF8574N 205
5.16.1.3. Hardware - 7-Segment-Anzeige mit PCF8575C 206
5.16.2. Bedienung mit dem DemoBoard 207
5.16.3. Software 208
5.16.3.1. SCL und SDA mit DS2413P ansteuern 209
5.16.3.2. I²C-Initialisierung 210
5.16.3.3. Startbedingung 210
5.16.3.4. Stoppbedingung 211
5.16.3.5. Ein Byte über I²C verschicken 212
Kapitel 6 • Praxisbeispiele 216
6.1. Elektronisches OneWire-Schloss 216
6.1.1. Die Schaltung 217
6.1.2. Die Bedienung 219
6.1.3. Die Firmware 219
6.1.3.1. I²C-Kommunikation mit EEPROM 219
6.1.3.2. OneWire-Kommunikation 223
6.1.3.3. Die endlose Hauptschleife 224
6.2. Temperatur messen 228
6.2.1. Thermometer mit einer 7-Segment-LED-Anzeige mit gemeinsamer Kathode 229
6.2.1.1. Die Schaltung 230

6.2.1.2. Die Firmware 231
6.2.2. Thermometer mit einer 7-Segment-LED-Anzeige mit gemeinsame Anode 232
6.2.2.1. Die Schaltung 232
6.2.2.2. Die Firmware 232
6.3. OneWire-Suche: Der Algorithmus 233
6.4. OneWire-Suche: Die Implementierung 242
6.4.1. OneWire-Triplet 242
6.4.2. Die Implementierung in Assembler 244
6.5. OneWire-Suche: Die Beispielschaltung 252
6.5.1. Version 1 - ROM-Code-Leser mit PIC16F1828 254
6.5.2. Version 2 - ROM-Code-Leser mit PIC18F13K22 257
6.5.3. Version 3 - ROM-Code-Suche mit PIC18F13K22 257
Kapitel 7 ● Anhang A - Maxim Evaluation Kits 260
7.1. Evaluation Kits 260
7.1.1. DS9090EVK 260
7.1.2. DS9092K 262
7.1.3. DS9481R-3C7 263
7.2. OneWireViewer 264
7.2.1. DS2401 - Maxim-Version 265
7.2.2. DS2433 266
7.2.3. DS2406 267
7.2.4. DS18B20 268
7.2.5.DS2417 269
7.2.6.DS2505 270
7.3. DS28E17K 275
7.3.1. DS28E17 (Familiencode: 19h) 277
7.3.1.1 Übersicht 277
7.3.1.2 Konfigurationsregister 279
7.3.1.2 Status Byte 280
7.3.1.3 Write Byte Status Byte 280
7.3.1.4 CRC16 Berechnung 281

7.3.1.5 Befehle 287
7.3.2. Thermometer DS7505 288
7.3.3. Demosoftware DS28E17 EV Kit 289
7.3.3.1. Beispiel mit DS7505 290
7.3.3.2. Beispiel mit PCF8574 293
7.3.3.3. Beispiel mit PCF8574 - Assembler-Routinen 296
Index 302

Kapitel 1 • Die OneWire-Welt

Was ist OneWire? Und wozu ist es gut? Warum braucht man zwei Leitungen für ein One-Wire-System?

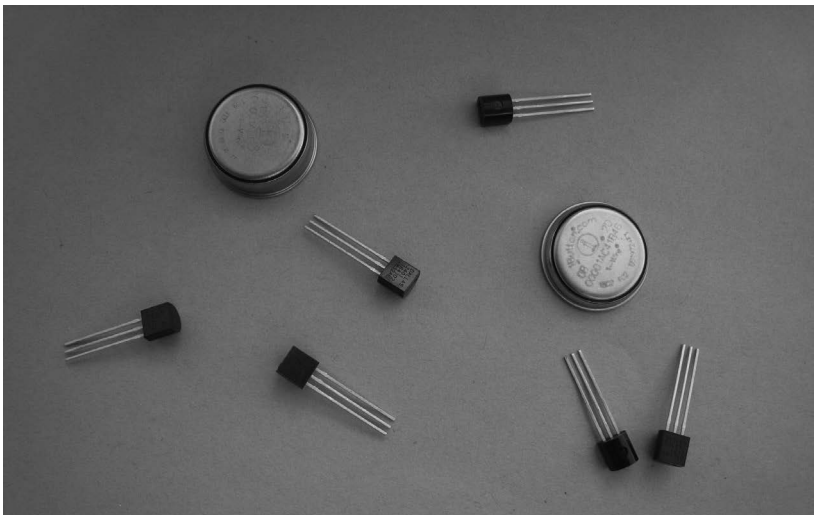
So viele Fragen. Also - um was es geht eigentlich? OneWire ist ein Master-Multi-Slave-Kommunikationsprotokoll, das von der Firma Dallas Semiconductor entworfen worden ist. Dallas Semiconductor ist später (2001) von Maxim Integrated übernommen worden. Das interessanteste an dem Protokoll ist, dass für die Einspeisung von Slave-Devices auf dem Bus keine eigene Leitung notwendig ist. Damit der Master mit einem Slave kommunizieren kann, ist lediglich eine (Daten-) Leitung notwendig. Na ja, eine Leitung - eigentlich zwei, denn GND (Ground) braucht man immer. Das ganze Eindraht-System kann also mithilfe von zwei physischen Leitungen kommunizieren.

Die Grundmerkmale (außer der Anzahl der Leitungen) ist auch die serielle Kommunikation, die lange Distanzen ermöglicht, anderseits aber eine sehr langsame Datenübertragung.

Weil es nur „einen Draht für alles" gibt, wird die Übertragung selber logischerweise nicht synchronisiert - also ist eine exakte Taktung gefragt. Weil die Daten in beide Richtungen übertragen werden, ist auch sehr exakt definiert, wann der Master die Daten übermittelt und wann der Slave. Die Datenübertragung wird immer vom Master initiiert und gegebenenfalls beendet. An einem Bus können mehrere Slaves angeschlossen werden, weswegen auch eine Adressierung eingesetzt wird.

Spezifisch für die OneWire-Adressierung ist, dass jeder einzelne OneWire-Slave-Chip eine eindeutige, 64 Bit lange ID enthält. Wichtig ist zu verstehen, dass wirklich jeder individuelle Chip seine eigene ID besitzt, nicht etwa der Chip-Typ oder die Chip-Familie. Wenn also ein Chip im Gerät ausgetauscht wird, ändert sich auch die zugehörige OneWire-Adresse.

Die bekanntesten beiden Einsatzgebiete des OneWire-Chips sind die Temperaturmessung und die Zutrittskontrolle.



Für die Temperaturmessung werden heutzutage häufig verschiedene abgeleitete Versionen des legendären DS1820-Sensors verwendet.

Für die Zutrittskontrolle (am Hauseingang oder einer Kasse) werden verschiedene „ROM-ID-Chips" á la DS2401/DS1900 eingesetzt.

Die Slave-Chips werden - Stand heute - ausschließlich von Maxim Integrated hergestellt. Der Hersteller garantiert die weltweit eindeutige ROM-ID, die wirklich einzigartig für jeden Chip ist. Als Master werden entweder verschiedene Brücken-Chips mit „OneWire zu einem anderen seriellen Protokoll" oder direkt Mikrokontrollers verwendet.

Das Buch zeigt vor allem, wie man verschiedene verfügbare OneWire-Chips in eigene mikrocontroller-gesteuerte Anwendungen integrieren kann, und zwar inklusive Assembler-Software-Implementierungen für PIC-Mikrokontroller und auch Arduino.

Wichtig ist noch zu sagen, dass das Übertragungsprotokoll an verschiedenen Stellen / Zeitpunkten einen CRC (Cyclic Redundancy Check) verwendet. Was es ist und wie es zu berechnen ist, zeigen wir später. Der Zweck des CRC ist, mögliche Übertragungsfehler zu erkennen und die laufende Kommunikation rechtzeitig abzubrechen und gegebenenfalls zu wiederholen.

Das Protokoll soll eigentlich eine einfache und kostengünstige Kommunikation zwischen Mikrocontroller und Peripherie-Chips ermöglichen - und zwar drahtgebunden. Der Schwerpunkt liegt dabei nicht auf der Geschwindigkeit der Datenübertragung, sondern in der Einfachheit. Genauso wichtig ist es, die peripheren Bauteile so weit wie möglich von dem Mikrocontroller entfernt zu halten - damit man zum Beispiel eine Temperaturmessung an einem fernen Ort durchführen kann. Der „ferne Ort" kann entweder ein Netzteil eines Rechners sein, der von der Hautplatine vielleicht 30 cm entfernt ist, oder aber auch eine Ecke im Garten, die vom Master (Mikrocontroller) ein paar hundert Meter entfernt ist.

Genauso wichtig war es - wie schon kurz erwähnt - mehrere Peripheriebauteile an einen Bus anschließen zu können.

Aus diesen Merkmalen ist auch die Gruppe von verfügbaren OneWire Chips entstanden. Wir finden zum Beispiel keine A/D-Wandler, die sehr schnelle Datenübertragung verlangen, anderseits Temperatursensoren - wo es ausreichend ist, die Temperaturmessung „gelegentlich" durchzuführen, wobei „gelegentlich" ein paar Mal pro Stunde oder jede Sekunde oder alle 200 Millisekunden sein kann. Wir finden Identifikations-Chips, deren Aufgabe es sein kann, eine Tür zu öffnen, oder sogar Uhrenbausteine - wo aber auch die Kommunikation höchstens paar Mal pro Sekunde durchgeführt werden kann. Und es gibt neben den erwähnten noch ein paar andere Arten, die wir alle in den nächsten Kapiteln entdecken.

Im Buch wird meistens der Begriff OneWire verwendet. Im Deutschen ist eher der Begriff 1-Wire üblich. Der besseren Auffindbarkeit bei der Online-Suche wegen wurde 1-Wire im Buchtitel verwendet, während im Buch vorwiegend OneWire verwendet wird. Beide Begriffe sind synonym.

Viel Spaß dabei!

Kapitel 2 ● OneWire Theorie

Eigentlich können wir gleich loslegen. Am Anfang schadet es nicht, sich mit ein wenig Theorie auseinanderzusetzen. Wir werden uns mit der Topologie des Netzwerks beschäftigen, wir erwähnen kurz die typischen Einsatzgebiete, danach machen wir uns mit den Protokollkonventionen vertraut und zeigen auch ein Beispiel, wie man einen CRC-Prüfwert berechnen kann.

Unter „Protokollkonventionen" verstehen wir die Grundlagen der Kommunikation. Dabei klären wir, welche Kommunikationsschritte für die Verbindung zwischen dem Master und den Slaves notwendig sind und wie die einzelnen Bits währen der Kommunikation übertragen werden.

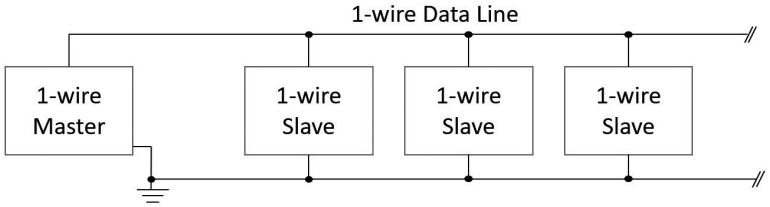
2.1. OneWire Netzwerktopologie

Bei einem OneWire-Netzwerk können entweder ein oder mehrere Slave-Chips an den Bus angeschlossen werden. Deswegen gibt es auf dem Bus eine Adressierung. Jeder Slave-Chip besitzt eine weltweit eindeutige ID, die schon bei der Herstellung als 64-Bit-ID vergeben und im ROM des Chips gespeichert wurde. Das bedeutet, dass jeder OneWire-Slave-Chip ein internes ROM für die ID enthält.

Es kommuniziert immer ein Master mit einem Slave, wobei sich auf dem Bus gleichzeitig mehrere Slaves befinden können. Zu der Datenübertragung wird nur eine Leitung verwendet, wobei keine Synchronisation erfolgt. Das interessanteste ist, dass die Datenleitung auch der Stromversorgung des Slaves dient. Deswegen benötigt man für die Verbindung zwischen Master und Slave(s) wirklich nur GND und eine Leitung.

2.1.1. Lineare Topologie

Die typische OneWire-Topologie ist linear und sieht aus wie folgt:



In anderen Worten: Der 1-Wire- oder auch Eindraht-Bus stellt das System einer seriellen Kommunikation zwischen einem Master und einem oder mehreren Salve-Chips dar, die mit nur zwei Leitungen verbunden sind. Die erste ist die Datenleitung, die oft als „DQ" bezeichnet wird, manchmal aber auch „Data" oder „OWIO" (engl. OneWire Input Output) und die zweite ist GND, für die manchmal auch das Kürzel OWRTN (engl. OneWire ReTurN) verwendet wird. Egal, wie die Benennung auch sein mag – DQ sorgt für den beidseitigen Datenaustausch und stellt außerdem die sogenannte parasitäre Stromversorgung für die Slaves zur Verfügung. GND ist „einfach GND". Die DQ-Leitung ist immer ein Open-Collector-Ausgang. Das heißt, wenn die Leitung „still" ist, liegt sie wegen des externen Pull-up Widerstandes auf einer logischen 1.

Um die Stromversorgung des Slaves kümmert sich prinzipiell das System der parasitären Stromversorgung, das fester Bestandteil jedes Slaves ist. Wenn die DQ Leitung auf „1" liegt, ist dieses System dafür zuständig, genug Energie aus der Leitung „zu klauen", um den Slave für eine Weile lauf- und kommunikationsfähig zu halten.

Manche Chips aber benötigen für aufwändigere Operationen mehr Energie, als die parasitäre Stromversorgung liefern kann, zum Beispiel die Temperaturmessung bei Temperatursensoren oder beim EEPROM-Speicher das Schreiben von Daten vom EEPROM-Puffer ins EEPROM. Bei solchen Operationen muss der Master die Kommunikation auf dem Bus für eine gewisse Zeit unterbrechen und die Leitung „künstlich" auf „1" halten, damit der Bus zur Stromversorgung des Slaves genutzt wird und dieser die Operation erfolgreich beenden kann.

Weil die Datenübertragung nicht mit einem Taktsignal synchronisiert werden kann, ist es äußerst wichtig, die vorgeschriebenen Formate und Zeitspannen bei der Datenübertragung einzuhalten.

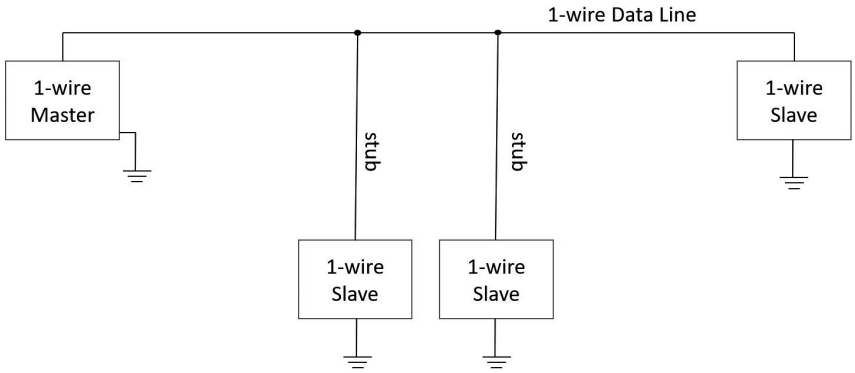
Typische oder besser die am meisten verbreiteten OneWire-Chips sind Temperatursensoren, dann folgen ROM- und EEPROM-Speicher und Echtzeituhren (RTC).

Der größte Vorteil von OneWire-Temperatursensoren gegenüber I^2C-Temperatursensoren ist zweifellos der Anzahl der notwendigen Leitungen. Vor allem, wenn man die Temperatur an verschiedenen, weit voneinander entfernten Stellen messen möchte, ist es ein erheblicher Unterschied, ob man für die Verbindung zu und zwischen den Sensoren vier oder nur zwei Drähte benötigt.

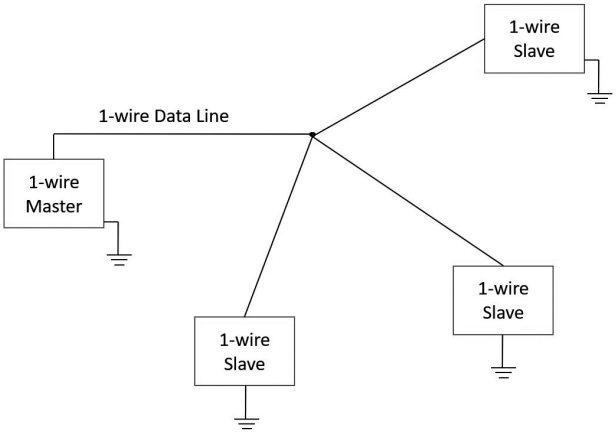
2.1.2. Variationen der Linearen Topologie

Eigentlich könnte man sich verschiedene Variationen der linearen Topologie überlegen - wir werden sogleich zwei davon vorstellen.

Die erste davon ist sogenannte „Stubbed Topology". Prinzipiell geht es darum, dass wir eine Hauptleitung zwischen dem Master und dem am weitest entfernten Slaven haben. Zusätzlich sind weitere Slave-Chips über Stichleitungen (Stubs) von drei oder mehr Metern am Hauptbus angeschlossen.



Die andere Variation ist die „Star Topology" - also Sterntopologie, bei der von einen zentralen Punkt mehrere Leitungen zu den Slave-Chips führen. Diese Topologie könnte man so darstellen:



Laut Hersteller ist die Stern-Topologie die am meisten fehleranffällige und wird deswegen nicht wirklich empfohlen. Genauso nicht empfehlenswert ist es, verschiedene Topologien zu mischen. Mit einer Ausnahme!

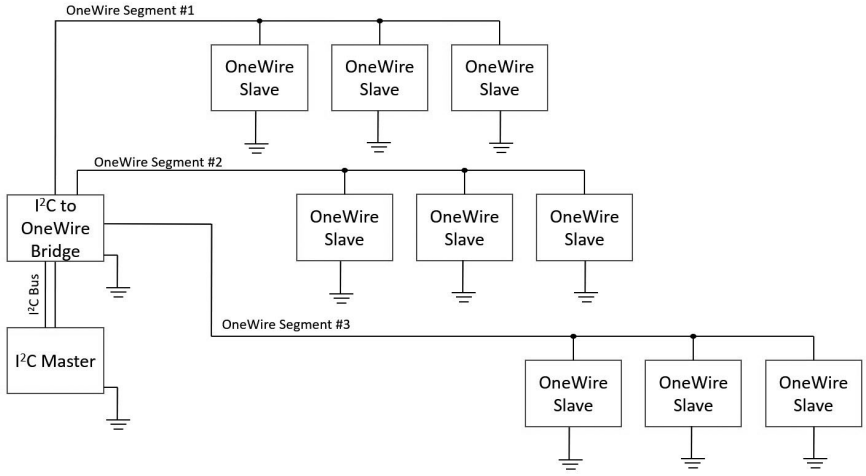
2.1.3. Geschaltete Netzwerke

Um die mögliche Reichweite des Netzwerkes zu verlängern, könnte man eine Mischung von Stern- und Linearer Topologie nutzen, jedoch unter der Vorrausetzung, dass immer nur ein „Strahl" innerhalb des Sternes aktiv ist.

Es handelt sich also eigentlich um eine Lineare Topologie, jedoch können die „Linien" während Laufzeit aus- und eingeschaltet werden. Dies lässt sich beispielsweise mit einfachen Analogschaltern erreichen.

Genauso gut kann man für diesen Zweck eine typische I²C-to-OneWire-Bridge einsetzen. In diesem Fall erhalten wir eine Mischung aus zwei Netzwerktypen. Der Master im Zentrum des Sterns wird an der Bridge über den I²C-Bus angeschlossen, die Zweige der OneWire-Slaves dann über voneinander getrennte OneWire-Busse.

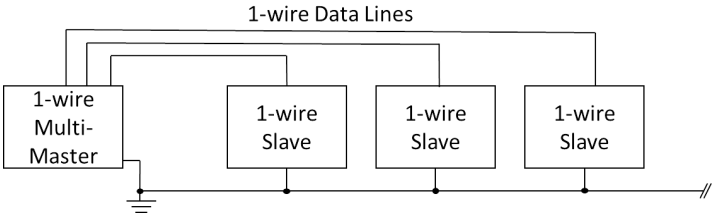
So eine Topologie können wir dann so darstellen:



Der Bridge-Chip dient als Umschalter zwischen den OneWire-Zweigen.

Eine spezielle Abart dieser Topologie wäre, dass an jedem Segment des OneWire-Netzwerkes jeweils nur ein OneWire-Chip angeschlossen ist.

Dies kann man auch so erreichen, dass der Master selber mehrere OneWire Netze direkt (ohne Bridge-Chip) zur Verfügung stellt.



Der Sinn dieser „Single-Salve"-Anordnung ist eine deutliche Vereinfachung der Adressierung. Dazu kommen wir aber später.

2.2. Protokoll-Konventionen

Mit dem Kommunikationsprotokoll werden wir uns detailliert im Kapitel 5 beschäftigen - hier werden wir in aller Kürze nur die Grundlagen behandeln.

Wir schildern, welche Kommunikationsbausteine aus Sicht der technischen Kommunikation wichtig sind und wie der logische Aufbau der Kommunikation aussieht.

Beginnen wir mit einer wichtigen Funktionalität der OneWire-Welt, dem ROM-Code.

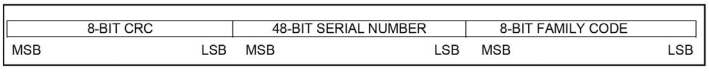
2.2.1. ROM-Code

Wie wir schon wissen, enthält jeder OneWire-Chip eine eigene eindeutige Seriennummer, vergleichbar vielleicht mit den MAC-Adressen in der Welt der Ethernets. Diese Serial Number heißt ROM-Code und ist 64 Bit lang. Der ROM-Code ist dreigeteilt:

- Der Family Code ist ein Byte lang und ist im LSB des ROM-Codes gespeichert. Der Familiencode sagt etwas über die grundsätzliche Funktion des Chips aus (etwa Temperatur messen)

- Die Serial Number ist 6 Bytes lang und liegt „in der Mitte" des ROM-Codes - dies ist die einzigartige ID des Chips

- Der CRC Code (Cyclic Redundance Check) ist ein Byte lang und am MSB des ROM-Codes gespeichert. Der Prüfwert wird aus den ersten 7 Bytes (oder 56 Bits) berechnet und kann zu der Prüfung der Kommunikation dienen.



Der ROM-Code ist deswegen so interessant, als dass es OneWire-Chips gibt, die nur den ROM-Code enthalten und sonst nichts tun. Ein Beispiel dafür ist die „Silicon Serial Number" DS2401. Einen solchen Chip kann man als Schlüssel für verschiedenste Anwendungen einsetzen.

Interessant ist sicherlich zu wissen, dass bei der Datenübertragung von mehreren Bytes immer zuerst das LSB übertragen wird und das MSB als letztes. Bei der ROM-Code-Übertra-

gung erscheint also der Family Code als erstes Byte auf dem Bus, gefolgt von den 6 Bytes der Seriennummer und abschließend der CRC-Prüfwert als MSB.

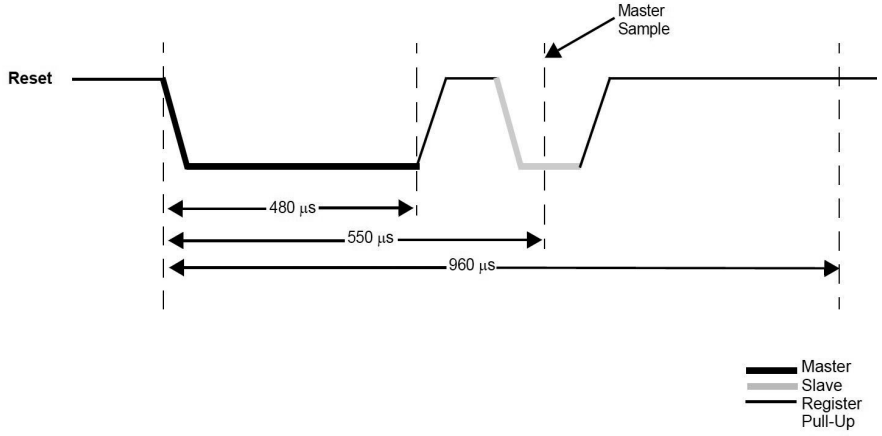
2.2.2. Technische Ebene

Die ganze Kommunikation auf technische Ebene in der OneWire-Welt lässt sich in vier Teile zerlegen, die jetzt kurz beschrieben werden. Aus Sicht des Masters geht es um die folgenden vier Elemente: Reset, eine Null senden, eine Eins senden und ein Bit lesen.

Für die Bitübertragung werden in der OneWire-Welt sogenannte „Slots" mit festem Zeitrahmen und Konventionen verwendet.

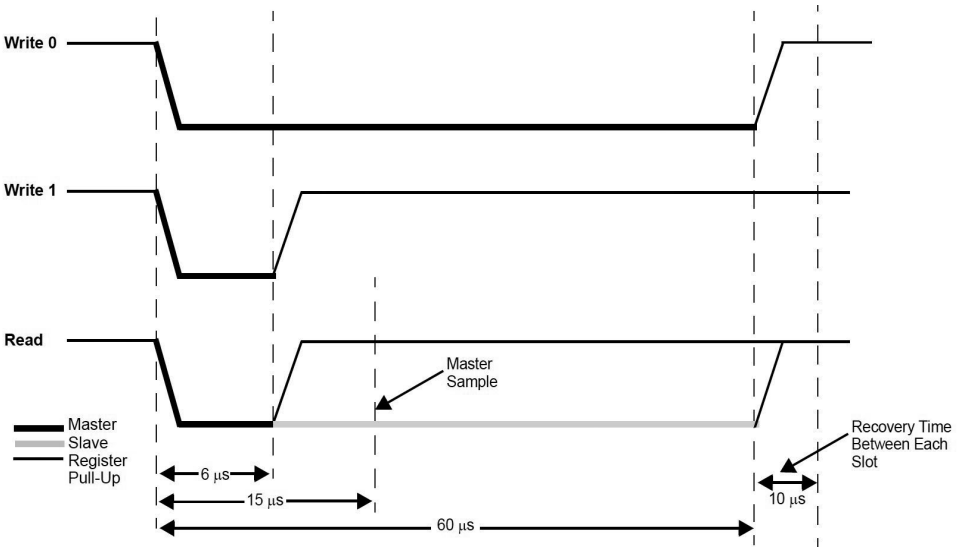
Element 1 - OneWire Reset

Im Bild können wir sehen, dass der Master am Anfang der Initialisierung den Bus auf null setzt, und zwar 480 µs lang. Danach wird der Bus von Master losgelassen, was dazu führt, dass der Pull-Up Widerstand den Bus (wieder) auf den Pegel 1 zieht. Dies ist das Zeichen für den Slave, dass er sich melden soll, was bedeutet, dass er den Bus auf null setzt, und zwar innerhalb der nächsten 70 µs. Zu diesem Zeitpunkt (550 µs ab Beginn der Reset-Prozedur) wird der Master überprüfen, ob der Bus auf null oder eins ist. Eine Null bedeutet, dass sich ein Slave gemeldet hat, eine Eins dagegen, dass der Master den Bus ganz alleine für sich hat...



Element 2 - OneWire Write Null

Das zweite Element ist, eine Null auf dem Bus zu schreiben - sprich, der Master schickt eine Null an den oder die Slaves. Wie auf dem Bild 5.2.2.-02 ganz oben zu sehen, schaltet der Master den Bus-Pegel für die Zeitspanne von 60 µs auf null. Danach wird der Bus vom Master freigegeben. 10 µs muss der Bus nun ruhen, dann kann das nächste Bit übertragen werden.



Element 3 - OneWire Write One
Das dritte Element der Datenübertragung auf dem OneWire-Bus ist es, eine Eins zu versenden. Wenn der Master eine Eins schicken möchte, zieht er den Bus ebenfalls auf null, aber nur für 6 µs, und lässt ihn danach wieder los. Die restlichen 54 µs bleibt der Bus auf eins; plus der vorgeschriebenen Recovery-Zeit von 10 µs zwischen einzelnen Bits.

Element 4 - OneWire Read
Das letzte Element wird benötigt, damit der Master die Daten der Slaves lesen kann. Daten heißt in diesem Kontext - ein Bit. Wie auf dem Bild gezeigt, wird die Kommunikation wieder vom Master initialisiert, und zwar so, dass er den Bus auf Pegel null legt, genau wie beim dritten Element für 6 µs. Danach wird der Bus von Master freigegeben, jetzt kann aber der Slave eine Eins oder eine Null schicken.

Falls der Slave eine Eins senden will, passiert eigentlich gar nichts mehr. Der Bus bleibt auf der Eins und zwar bis Ende dieses Lese-Slots. Wie aus dem Bild 5.2.2.-02 ersichtlich ist, prüft der Master den Buspegel 9 µs nach der Busfreigebe (15 µs von Anfang des Slots gerechnet).

In dem Fall, dass der Slave eine Null sendet, zieht er den Bus für den Rest des Slots auf null.

2.2.3. Kommunikationsschritte

Im OneWire-Land ist es streng vorgeschrieben, wie kommuniziert werden soll. In jeder Kommunikation zwischen Master und Slave gibt es stets drei verpflichtende Schritte:

Schritt 1 - Initialisierung
Schritt 2 - ROM-Befehl
Schritt 3 - Funktionsbefehl

Die ersten beiden Schritte sind immer unabhängig von der Funktion des Slaves. Ob der Master also mit einem Meteorologen (Temperatursensor), einem Archivar (Serial Nummer Chip) oder einem Uhrmacher (RTC) reden möchte, sämtliche Unterhaltung beginnt mit der Initialisierung, gefolgt vom ROM-Befehl. Was danach passiert, hängt aber schon von der Spezialisierung ab.

Schritt 1 - Initialisierung
Die Initialisierung ist ein sehr einfacher (aber auch wichtiger) Schritt, weil sie den Zeitpunkt definiert, an dem ein Dialog beginnen soll. Technisch gesehen heißt es auch „OneWire-Reset". OneWire-Reset heißt wiederum nur, dass der Master die Leitung auf die Null zieht und in diesem Zustand sehr lange hält. Sehr lange in der OneWire-Welt bedeutet 480 µs.

Schritt 2 - ROM-Befehl
Nach der Initialisierung erwarten die Busteilnehmer einen ROM-Befehl (engl. „ROM Function Command"). Dies entspricht der Adressierung. An dieser Stelle gibt der Master Bescheid, mit wem er reden will. Es gibt verschieden Möglichkeiten, diesen Schritt zu realisieren. Wir werden die Möglichkeiten, die verschiedene ROM-Befehle später beleuchten. Jetzt brauchen wir nur zu wissen, dass es einen „Skip ROM"-Befehl gibt. Die Bedeutung dieses Befehls ist, dass wir mit jedem reden wollen, der sich auf dem Bus befindet...

Schritt 3 - Funktionsbefehl
Wenn der ROM-Befehl verarbeitet ist, folgt ein Funktionsbefehl (engl. „Function Command"), der sehr stark vom Chip selber abhängt. Es ist auch nachvollziehbar, dass, will man mit einem Temperatursensor „diskutieren", andere Befehle benötigt werden als in der Kommunikation beispielsweise mit einem GPIO-Chip.

Als Beispiel kann man auf diese Stelle den Befehl „Convert T" (44h) für den Temperatursensor DS18S20 nennen, der eine Temperaturmessung initialisiert. Mehr dazu später.

2.2.4. Nicht zu vergessen - CRC

Wie wir schon wissen, ist das sogenannte „CRC Byte" ein fester Bestandteil des ROM-Codes. Anhand des CRC-Prüfwerts kann der Master prüfen, ob der Datentransfer von Salve zum Master fehlerfrei verlaufen ist. Der CRC-Prüfwert ist im ROM-Code das achte Byte und lässt sich aus den vorherigen 7 Bytes berechnen. Die Idee dabei ist, dass der Master während des Datenempfangs Bit per Bit einen CRC berechnet und diesen am Ende des ROM Code-Transfers mit dem empfangenen CRC Code vergleicht. Falls der berechnete und der empfangene CRC-Code identisch sind, ist die Datenübertragung fehlerfrei verlaufen.

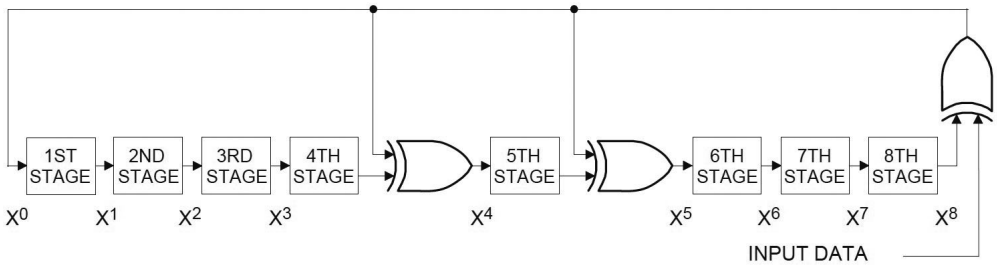
Bei der CRC-Berechnung kann man zwei verschiedene Wege gehen:

- CRC mit einem definierten Algorithmus berechnen
- CRC als „Look-up-Tabelle" implementieren

Die beiden Methoden zeigen wir uns jetzt in kürze.

Noch zu erwähnen ist, dass der CRC nicht nur bei der Übertragung des ROM-Codes stattfindet, sondern oft auch bei einer Datenübertragung (zum Beispiel eines Temperaturwerts). Dabei handelt es sich typischerweise um „Scratchpad"-Operationen. Scratchpad ist eine Art Puffer, den eigentlich fast jeder 1-Wire Chip verwendet - dazu kommen wir aber später. Wie lang dieser Scratchpad-CRC ist, hängt dann von der Größe des Puffers ab.

Der CRC wird in jedem Chip berechnet und als letztes Byte des ROM-Codes ausgesendet. Um die Berechnung kümmert sich der 1-Wire-CRC-Generator, der aus Schieberegistern und XOR-Gates besteht, wie man hier sehen kann:



Eine arithmetische Darstellung der CRC-Berechnung sieht dann so aus:

$$\text{Polynomial} = X^8 + X^5 + X^4 + 1$$

hier bitte **CRC-Polynom** in der Formel schreiben!

Diese Vorgehensweise, um den CRC-Wert zu berechnen und später mit dem empfangenen CRC zu vergleichen, kann man sicherlich auch selber in der Firmware implementieren, aber ich gebe gleich zu, dass es mir relativ kompliziert vorkommt, so was zu implementieren, ganz abgesehen davon, dass ich gar nicht verstehe, was „Polynom" heißt...

Es gibt aber einen Ausweg - eine Möglichkeit, wie man den CRC berechnen kann und sich nicht mit Polynomdarstellungen auseinandersetzen muss, ist die Verwendung von einer „Konstantentabelle" - die sogenannte *CRC Look-up Table*.

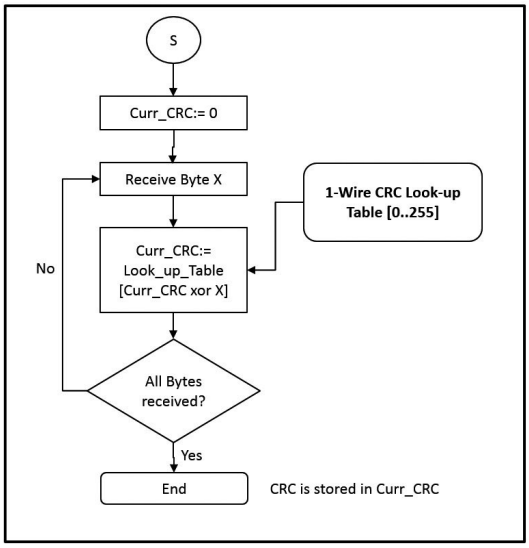
Für Faulpelze, wie ich einer bin, ist die Verwendung einer CRC-Look-Up- oder -Umsetzungstabelle die allerbeste Lösung. Sie ist deutlich einfacher zu verstehen - eigentlich muss man nicht wirklich was verstehen, nur sehr einfach eine Tabelle mit 256 Bytes definieren. Der Algorithmus ist in diesem Fall wirklich sehr einfach. Man braucht nur eine sehr einfache Operation mit dem gerade empfangenen Byte zu machen und das Ergebnis dann als Index in der Umsetzungstabelle einsetzen, also: Sehr_Einfache_Operation (empfangene_Byte) rein - als Index in die Tabelle; Wert auslesen, und da ist schon mein Ergebnis. Dazu muss man seinen Gehirnschmalz nicht strapazieren, nur 256 Words Programmspeicher opfern.

Die Implementierung - oder besser gesagt die Definition der Tabelle (für 8-Bit CRC-Werte wie beim ROM-Code-CRC) kann in Assembler so aussehen:

```
;------------------------------------------------------------------------
;1-wire CRC look-up table
;------------------------------------------------------------------------
ow_crc_lt    da  D'000', D'094', D'188', D'226', D'097', D'063', D'221',
D'131'
         da  D'194', D'156', D'126', D'032', D'163', D'253', D'031', D'065'
         da  D'157', D'195', D'033', D'127', D'252', D'162', D'064', D'030'
         da  D'095', D'001', D'227', D'189', D'062', D'096', D'130', D'220'
         da  D'035', D'125', D'159', D'193', D'066', D'028', D'254', D'160'
         da  D'225', D'191', D'093', D'003', D'128', D'222', D'060', D'098'
         da  D'190', D'224', D'002', D'092', D'223', D'129', D'099', D'061'
         da  D'124', D'034', D'192', D'158', D'029', D'067', D'161', D'255'
         da  D'070', D'024', D'250', D'164', D'039', D'121', D'155', D'197'
         da  D'132', D'218', D'056', D'102', D'229', D'187', D'089', D'007'
         da  D'219', D'133', D'103', D'057', D'186', D'228', D'006', D'088'
         da  D'025', D'071', D'165', D'251', D'120', D'038', D'196', D'154'
         da  D'101', D'059', D'217', D'135', D'004', D'090', D'184', D'230'
         da  D'167', D'249', D'027', D'069', D'198', D'152', D'122', D'036'
         da  D'248', D'166', D'068', D'026', D'153', D'199', D'037', D'123'
         da  D'058', D'100', D'134', D'216', D'091', D'005', D'231', D'185'
         da  D'140', D'210', D'048', D'110', D'237', D'179', D'081', D'015'
         da  D'078', D'016', D'242', D'172', D'047', D'113', D'147', D'205'
         da  D'017', D'079', D'173', D'243', D'112', D'046', D'204', D'146'
         da  D'211', D'141', D'111', D'049', D'178', D'236', D'014', D'080'
         da  D'175', D'241', D'019', D'077', D'206', D'144', D'114', D'044'
         da  D'109', D'051', D'209', D'143', D'012', D'082', D'176', D'238'
         da  D'050', D'108', D'142', D'208', D'083', D'013', D'239', D'177'
         da  D'240', D'174', D'076', D'018', D'145', D'207', D'045', D'115'
         da  D'202', D'148', D'118', D'040', D'171', D'245', D'023', D'073'
         da  D'008', D'086', D'180', D'234', D'105', D'055', D'213', D'139'
         da  D'087', D'009', D'235', D'181', D'054', D'104', D'138', D'212'
         da  D'149', D'203', D'041', D'119', D'244', D'170', D'072', D'022'
         da  D'233', D'183', D'085', D'011', D'136', D'214', D'052', D'106'
         da  D'043', D'117', D'151', D'201', D'074', D'020', D'246', D'168'
         da  D'116', D'042', D'200', D'150', D'021', D'075', D'169', D'247'
         da  D'182', D'232', D'010', D'084', D'215', D'137', D'107', D'053'
;------------------------------------------------------------------------
```

Man muss noch wissen, dass die „Sehr_Einfache_Operation" eine Entweder-Oder-Funktion („Exclusive Or", XOR) ist, und dass man mit einem Startwert von null für den CRC anfangen soll.

Wie gesagt, das Programm muss dann nur noch machen, was im folgenden Bild dargestellt ist:



Ich muss noch zugeben (ich bin nicht ganz sicher ob ich es darf - mache es aber dennoch) - dass ich in meinen Endanwendungen die CRC-Funktionalität nie verwendet habe. Ich habe auch nie Probleme damit gehabt, und manche Anwendungen laufen bei mir seit ein paar Jahren -24/7!

2.2.5. ROM-Befehle

Typischerweise werden immer folgende vier ROM-Befehle von allen OneWire-Chips unterstützt:

Befehl	Name	Beschreibung
F0h	Search ROM	ROM-Suche - alle OneWire-Chips auf dem Bus werden durchgesucht mit dem Ziel, alle individuellen ROM-IDs der Chips zu erfahren
33h	Read ROM	Auslesen des ROM-IDs aus einem OneWire-Chip. Dieser Befehl ist nur bei einem OneWire-Chip auf dem Bus sinnvoll einsetzbar.
55h	Match ROM	Ein konkreter OneWire-Chip wird mit dem Befehl angesprochen.
CCh	Skip ROM	Alle Chips auf dem Bus werden angesprochen.

2.2.5.1. Read ROM [33h]

Der Befehl Read ROM ist eine Aufforderung an einen Slave, seine eigene ID zu übermitteln. Der Befehl kann sinnvoll nur dann eingesetzt werden, wenn sich auf dem OneWire-Bus genau ein Chip befindet. Falls sich mehrere Chips auf dem Bus befinden, werden alle auf einmal ihre IDs abschicken, sodass es zu einer Datenkollision kommt und das Ergebnis ist damit nicht nutzbar.

2.2.5.2. Match ROM [55h]

Nachdem der Master einen Match-ROM-Befehl schickt, folgen noch 64 Bits der ROM-ID. Im weiteren Verlauf wird der Master nur mit dem Slave-Chip kommunizieren, dessen ROM-ID exakt zur ROM-ID passt, die gerade der Master gesendet hat. So kann man auf einem Bus, an den mehrere OneWire-Chips angeschlossen sind, ganz gezielt mit einem individuellen Chip kommunizieren. Um das zu bewerkstelligen, muss der Master die ROM-Codes (ROM-IDs) der Chips kennen. Sie müssen entweder schon bei der Herstellung bekannt sein oder aber zum einen geeigneten Zeitpunkt ermittelt werden. Um diese „Ermittlung" durchzuführen, kann der Befehl Search ROM [F0h] implementiert werden.

2.2.5.3. Search ROM [F0h]

Falls es bei der Anwendung sinnvoll ist, sollte der Master eine „Kennenlernrunde" nach einem wichtigen Ereignis durchführen. Dies dient dazu, dass der Master die OneWire-IDs von allen angeschlossenen OneWire-Chips erfahren kann. Mit diesem Befehl werden wir uns aber nur am Rande beschäftigen, weil wir vermeiden wollen, dass mehrere Chips auf einem Bus angeschlossen sind; und wenn doch - wie man mit der Situation ohne Search-ROM umgehen kann.

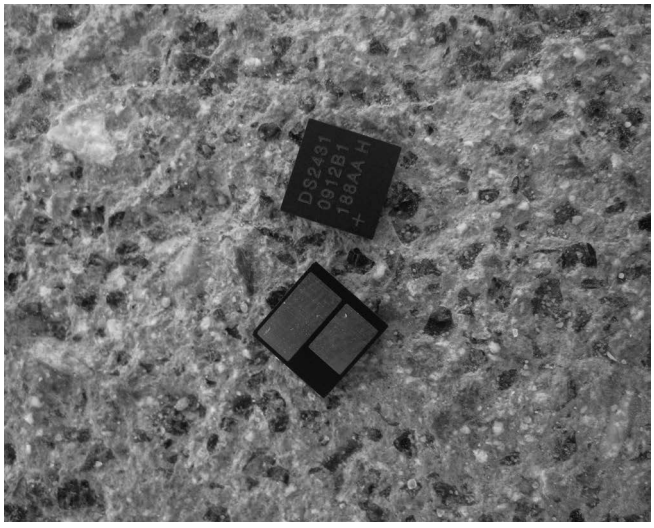
2.2.5.4. Skip ROM [CCh]

Der Befehl Skip ROM definiert, dass der Master alle auf dem Bus angeschlossenen Chips anspricht. Eigentlich hat der Befehl nur dann Sinn, wenn entweder auf dem Bus nur ein Chip angeschlossen ist, oder aber, wenn der Master nur einen Funktionsbefehl sendet, der für alle Slaves von Interesse ist, der Master aber keine Antwort erwartet. Ein typisches Beispiel dafür ist die Temperaturmessung. Falls an einem OneWire-Bus mehrere Temperatursensoren angeschlossen sind und wir möchten, dass all diese die Temperatur messen sollen, können wir erst einmal den ROM-Befehl Skip ROM senden und danach den Funktionsbefehl „Convert Temperature" [44h]. Das bedeutet, dass alle Sensoren ab diesen Zeitpunkt die Temperatur messen. Danach können wir die Messergebnisse mit einem anderen Funktionsbefehl individuell abfragen. Dabei müssen wir aber die Sensoren einen nach dem anderem mit dem ROM-Befehl Match ROM ansprechen.

2.3. Typische Einsatzgebiete

Wahrscheinlich die bekannteste Verwendung von OneWire-ROM-Chips (auch „registration number" genannt) ist die Einlasskontrolle in Bürogebäuden oder Mehrfamilienhäusern. Dabei sind die Chips mechanisch direkt als Schlüssel angelegt oder so gestaltet, dass man sie mit anderen Schlüsseln aufbewahren kann, ohne dass erhöhte Zerstörgefahr besteht. Diese Schlüssel nennt man „iButtons".

EEPROM- oder EPROM-Chips finden zum Beispiel oft in Tonerkartuschen für Laserdrucker Verwendung - als Kopien-Zähler. Wichtig ist auch hier wieder, dass sie mechanisch einfach appliziert werden können.



Kapitel 3 • Die Brücke

Es ist nicht selten der Fall, dass ein Gerät schon mit einem seriellen Bus ausgerüstet ist und diesen für die interne Kommunikation zwischen verschiedenen Komponenten verwendet. Meist handelt es sich nicht um einen OneWire-Bus, sondern RS-232, SPI oder beispielsweise I^2C. Deswegen gibt es von Maxim Integrated auch mehrere „Brücken-Chips", die es ermöglichen, die OneWire-Slave-Chips über andere serielle Schnittstellen zu steuern.

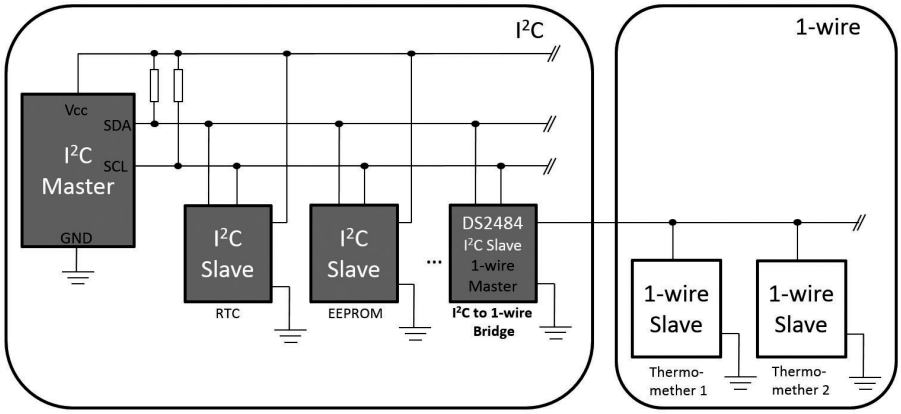
Chip	Schnitt-stelle	Gehäuse	Anzahl der OneWire-Busse	Kommentar / Highlights
DS2480B	RS232	SOIC-8	1	Konverter von RS232 zu OneWire. Ermöglicht die Programmierung von OneWire-EPROMs mit einer externen 12-V-Quelle.
DS2482-100	I^2C	SOIC-8	1	Einfachste Brücke I^2C zu OneWire. Bei Bedarf kann ein interner Pull-up-Widerstand aktiviert werden.
DS2482-101	I^2C	WLP-9	1	Erweiterte Version des DS2482-100 mit Sleep-Mode-Unterstützung
DS2482-800	I^2C	SOIC-16	8	Wie DS2482-100, jedoch mit acht unabhängigen OneWire-Bussen.
DS2483	I^2C	TDFN-8 SOT23-6	1	Vereinfacht gesagt eine flexiblere Version des DS2482-100. Man kann verschiedene Einstellungen für den OneWire-Bus vornehmen
DS2484	I^2C	TDFN-8 SOT23-6	1	Erweiterte Version des DS2484 - u.a wird ein OneWire Bus mit 1,8 V unterstützt (beim DS2483 sind es 3,3 V)

Der große Vorteil solcher „Brücken" ist, dass wir uns gar nicht um das Timing des OneWire-Busses kümmern müssen. Es ist ganz egal, auf welcher Frequenz unser Mikrocontroller tickt und welche Art von I/O-Pins er zu bieten hat. Wir brauchen nur die I^2C-Schnittstelle (oder RS232 beim DS2480B, was wir aber nicht weiter verfolgen), die von sehr vielen PIC-Mikrocontrollern (und auch Mikrocontroller anderer Hersteller) hardwaremäßig unterstützt wird. Deswegen ist es auch ein Kinderspiel, die den OneWire betreffenden Firmware-Abschnitte auf andere Mikrocontroller zu portieren. Normalerweise ist es nur eine Sache von „copy & paste"...

Wir können eine Brücke auch als eine Art „Dolmetscher" bezeichnen - weil wir eigentlich in einer „anderen Sprache" wie I^2C kommunizieren und die Brücke sich um „die Übersetzung" kümmert.

Mit einem solchen Dolmetscher lassen sich die Kommunikationsprinzipien sehr einfach auch auf andere Plattformen wie andere Mikrocontroller oder gar den Raspberry PI (mit Linux oder auch Windows betrieben) oder den Arduino übertragen.

Der Dolmetscher baut auf Basis eines I²C-Busses ganz eigenständig einen neuen 1-Wire-Bus. Gegenüber dem Mikrocontroller stellt er sich als ganz normaler I²C-Slave dar, auf der anderen Seite stellt er die „die Autorität" für seine eigenen 1-Wire Slaves dar.

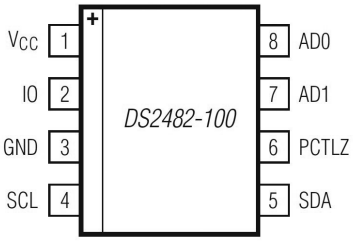


Wir werden uns kurz mit drei verschiedenen OneWire-Brücken-Chips für I²C beschäftigen.

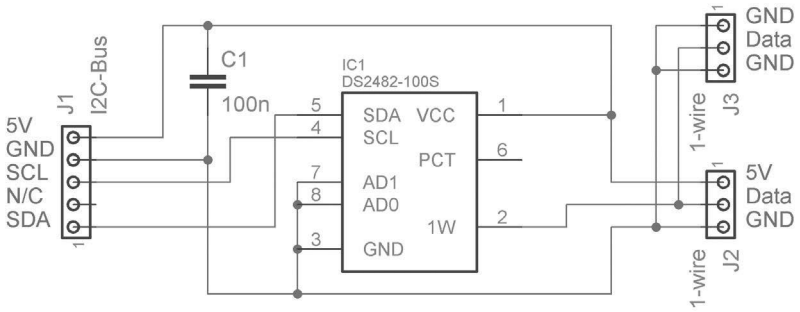
3.1. DS2482-100

Hier handelt sich um das „kleinste" Mitglied der Dolmetscher-Familie. Klein nicht im Sinne von Gehäuse, sondern von seiner Funktionalität (und auch vom Preis). Dieser Chip ist auch im großzügig bemessenen SMD-Gehäuse SOIC-8 erhältlich, das prinzipiell im Labor einfacher handzuhaben ist als der DS2484.

Das SOIC-8 Gehäuse sieht so aus:



Die Applikationsschaltung des DS2482-100 ist ebenfalls sehr überschaubar:



Mit dem Ausgang PCT kann man einen externen MOSFET-Transistor ansteuern und damit die Leistungskraft des 1-Wire Buses stärken.

Anderseits kennt der DS2482-100 die Parametrisierung des 1-Wire Busses, unter anderem des Port Configuration Registers nicht. Für die Experimente mit dem DemoBoard2015 kann man die Chips DS2484, DS2482-100 und den als nächstes beschriebenen DS2482-800 ohne Firmware-Anpassung verwenden. Kurz gesagt, kann man für eigene Anwendungen, falls man keine speziellen Anforderungen hat, den Dolmetscher beliebig wechseln.

Allerdings kann man - anders als beim DS2484 - die I^2C-Adresse des Chips mit den Eingängen AD0 und AD1 beeinflussen. Die Aussage, dass man die Chips ohne Firmware-Anpassung wechseln kann, gilt damit nur unter der Bedingung, dass man AD0 und AD1 an GND anschließt und damit die I^2C-Adresse auf 0011 000 festlegt. Die letzten zwei Bits dieser Adresse werden von AD1 und AD0 festgelegt: 0011 0 AD1 AD0. Theoretisch könnte man also vier verschiedene DS2482-100-Chips auf einem I^2C Bus betreiben.

3.2. DS2482-800

Der letzte Dolmetscher, den ich kurz erwähnen möchte, ist der größere Bruder des DS2482-100. Der wesentliche Unterschied (außer der Preis) ist, dass der DS2482-800 eigentlich acht unabhängige 1-Wire Busse zur Verfügung stellt. Dieser Dolmetscher ist unter anderen in einem SOIC-16-Gehäuse verfügbar. Die Belegung der einzelnen Pins ist im folgenden Bild zu sehen.