# Temperatur Sensor DS18B20

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

**Raspberry Pi: 1-Wire Temperatursensor DS1820**

Dank eines vorhandenen Treiber ist es nicht allzu schwer, sogenannte 1-Wire-Sensoren, -Speicher oder -Chips am Raspberry Pi zu betreiben. Ganz einfach wird einem das bei den bekannten Temperatursensoren gemacht, für die ein Extra-Treiber existiert, der den Sensorzugriff über die Dateischnittstelle ermöglicht.

**Der 1-Wire-Bus**

1-Wire ist ein digitaler, serieller Bus des Herstellers Maxim (ehem. Dallas), der mit einer Datenader und einer Masseleitung auskommt. Die Bezeichnung 1-Wire leitet sich daraus ab, dass zu der ohnehin vorhandenen Masseleitung nur eine weitere Ader für die gesamte Buskommunikation und die Energieversorgung erforderlich ist. Die Form der Energieübertragung bei diesem Bus heißt "parasitic power": Wenn gerade keine Daten übertragen werden, saugt sich der Chip seine Energie aus der aktiven Leitung und speichert sie in einem kleinen Kondensator, der während der Sendeaktivität zur Überbrückung dient. Aber auch eine kontinuierliche Versorgung über einen zweiten Draht ist möglich. 1-Wire wird auch als "MicroLAN" bzw. "Single Wire Serial Interface" bezeichnet. 1-Wire eignet sich insbesondere für Sensorik (Temperaturmessung, Akkuüberwachung, Spannung, Temperatur, Stromfluss), zur Steuerung und Meldung und für Identifikation durch einmalige, eindeutige und nicht veränderbare 64-Bit-ID. Diese ID setzt sich aus einer 48-Bit Seriennummer, einem 8-Bit langen Family-Code und einer 8-Bit-CRC zusammen.

Ursprünglich für die Kommunikation zwischen den Komponenten eines Gerätes bzw. Schaltschrankes entwickelt, wurde der 1-Wire-Bus zu einem einfach handhabbaren Bussystem für Strecken bis zu mehreren hundert Metern erweitert. Dies wurde insbesondere durch verbesserte Hostinterfaces mit Kontrolle der Signalanstiegszeit und aktivem Pull-Up Widerstand sowie durch Rauschfilter und ein optimiertes Timing erreicht. Auch die zusätzliche Versorgung des Bausteins verbessert die Qualität der Übertragung. Jeder Chip besitzt eine eigene Zeitbasis, die die ankommenden Signale nach ihrem logischen Gehalt unterscheiden kann. Die Datenübertragung erfolgt in Blöcken zu 64 Bit und ist bidirektional, seriell, asynchron und Halbduplex, da dieselbe Datenleitung für Senden und Empfangen benutzt wird. Die gesamte Kommunikation wird durch den Busmaster gesteuert.

Hierbei können mehrere Dutzend Slaves an einem Busmaster angeschlossen werden. Die Geschwindigkeit der Datenübertragung reicht von 15,4 kbps (Standard) bis 125 kbps (Overdrive). Hierbei ist "parasitic power" nur bei der Standard-Übertragungsrate möglich, welche jedoch völlig ausreichend ist.

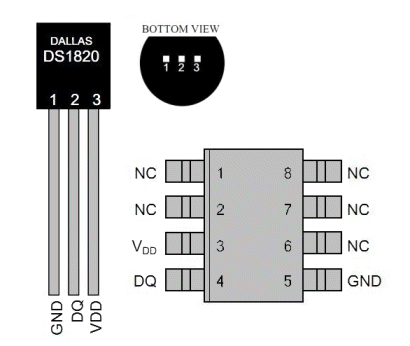
* Um eine logische 1 an den Chip zu senden, wird der Pegel der Datenleitung vom Master für maximal 15 Mikrosekunden auf Masse gezogen.
* Eine logische 0 entspricht einem Low-Pegel von mindestens 60 Mikrosekunden.

Der zeitliche Abstand zwischen beiden Zuständen ist groß genug, um eventuelle Produkttoleranzen zu überbrücken. Der 1-Wire-Bus ist also ein PWM-kodiertes Bussystem.

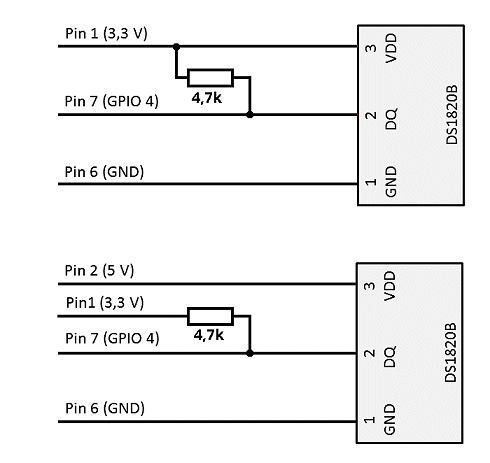
Obwohl die Versorgungsspannung für 1-Wire-Devices normalerweise 5 V beträgt, ist beim Raspberry Pi die verringerte Spannung von 3,3 V nötig, weil dessen GPIO-Ports 3,3 V vertragen und durch höhere Spannungen zerstört werden. Als Alternative kann man den 1-Wire Bus auch mit 5 V betreiben, dann muss aber zwingend das 1-Wire-Datensignal durch einen Spannungsteiler auf 3,3 V oder dergleiche begrenzt werden.

**Temperaturmessung mit dem DS18B20**

Beim [DS18S20](http://www.maximintegrated.com/datasheet/index.mvp/id/2815) und den verwandten Bausteinen [DS18B20](http://www.maximintegrated.com/datasheet/index.mvp/id/2812) sowie [DS1822](http://www.maximintegrated.com/datasheet/index.mvp/id/2795) handelt es sich um integrierte Schaltkreise im TO-92 Gehäuse, die Temperatursensor, Analog-Digital-Wandler und 1-Wire-Interface enthalten. Die Bausteine sind Anschluss- und Softwarekompatibel, sie unterscheiden sich im Wesentlichen in der Messgenauigkeit und im Preis. Den Temperatursensor gibt es auch noch im DIL-Gehäuse, wobei in der Regel die TO92-Variante verwendet wird. Ich verwende die Bezeichung DS18B20 als Synonym für alle Typen. Die Sensoren können direkt am GPIO des RasPi angeschlossen werden.



Der Sensor benutzt dabei die drei Leitungen Masse, 3,3 V und eine Signalleitung. Softwareseitig bringt die Raspbian-Distribution bereits die benötigten 1-Wire Kernel-Module mit, um solche Sensoren auszulesen. Die Module sind nur nicht per Default installiert und aktiv. Der DS18B20 wird an die GPIO-Pins für 3,3 V (Pin 1), Ground (Pin 6) und GPIO 4 (Pin 7) für die Datenleitung angeschlossen. Zwischen die 3,3-V-Leitung und die Datenleitung wird ein 4,7-Kiloohm-Widerstand geschaltet. Bei sehr langen Verbindungsleitungen kann man auch die Versorgung mit 5 V durchführen, wobei der Pullup-Widerstand immer an 3,3 V liegen muss.



Werden mehrere Sensoren benötigt, schließt man alle Sensoren parallel an, wobei der Pullup-Widerstand nur einmal benötigt wird. Er kann in der Nähe der RasPi-Steckerleiste untergebracht werden.

Der parasitäre Modus ist prinzipiell auch möglich. In diesem Fall wird der Vdd-Pin mit dem GND-Pin am Baustein verbunden (1 und 3), die Versorgungsleitung entfällt dann. In diesem Fall muss die Datenleitung auch die Speisung der Sensoren übernehmen, was die Übertragung etwas unsicherer macht. Auch mit den 3,3 V als Versorgung der Datenleitung tun sich viele Sensoren schwerer. Insofern sollte man bei drei Adern bleiben.

Raspbian hat die für den 1-Wire-Bus notwendigen Treiber bereits an Bord, wobei der oben erwähnte GPIO 4 für den 1-Wire-Bus vorgesehen ist. Wie bei allen GPIOs werden auch hier die Daten über virtuelle Dateien im Verzeichnis /sys verarbeitet, genauer in:

/sys/bus/w1/

Die 1-Wire-Unterstützung muss im Kernel als Modul nachgeladen werden werden (als root-User). Es werden zwei Module benötigt, eines für den 1-Wire-Bus allgemein und eines für die Temperaturmessung mit DS1820:

modprobe w1-gpio pullup=1

modprobe w1-therm

Wichtig ist der Parameter pullup=1, der besagt, dass eine parasitäre Speisung über einen Pullup-Widerstand vorhanden ist. Die Module legen im Verzeichnis /sys/bus/w1/devices mehrere symbolische Links auf Unterverzeichnisse an, zum Beispiel (Zeilen umbrochen):

root@raspberrypi:/sys/bus/w1/devices# ls -l

insgesamt 0

lrwxrwxrwx 1 root root 0 Mai 13 07:30 10-000802bf634d

-> ../../../devices/w1\_bus\_master1/10-000802bf634d

lrwxrwxrwx 1 root root 0 Mai 13 07:30 10-000802cfb15d

-> ../../../devices/w1\_bus\_master1/10-000802cfb15d

lrwxrwxrwx 1 root root 0 Mai 13 07:30 w1\_bus\_master1

-> ../../../devices/w1\_bus\_master1

Eigentlich findet alles in dem Verzeichnis statt, das über den symbolischen Link w1\_bus\_master1 erreicht werden kann (wie die Verweise oben zeigen). Der Name der ersten beiden Verzeichnisse setzt sich aus dem Family-Code der Sensoren und deren eindeutigen Identifikationsnummer zusammen. Sensoren vom Typ DS1820 und DS18S20 haben den Family-Code 10, DS18B20 den Code 28 und DS1822 den Code 22.

Diese Befehle laden die erforderlichen Module nur bis zum nächsten Reboot. Nach einem Neustart müssen die Treiber immer wieder neu geladen werden. Damit diese Aktivierung permanent bleibt und bei jedem Bootvorgang die Treiber automatisch geladen werden, müssen diese Aufrufe in die Datei /etc/modules eingetragen werden. Dazu werden einfach die beiden folgenden Zeilen am Ende der Datei hinzugefügt:

# /etc/modules

w1-gpio pullup=1

w1-therm

Bei anderen oder älteren Distribution kann es möglich sein, dass noch das Modul "wire" geladen werden muss; bei mir war dies nicht nötig.

Ab Kernelversion 3.18 wird für das Einsetzen der Module der sogenannte Device Tree verwendet. Ab dann wird die One-Wire-Schnittstelle in der Datei */boot/config.txt* aktiviert, indem dort zwei Zeilen eingetragen werden:

dtoverlay=w1-gpio,gpiopin=4

Damit wird der Pin 4 des GPIO für One Wire reserviert. Oder, falls noch der interne Pullup-Widerstand geschaltet werden soll:

dtoverlay=w1-gpio-pullup,gpiopin=4,extpullup=on

Siehe auch das [Device Tree Kapitel](http://www.netzmafia.de/skripten/hardware/RasPi/RasPi_Install.html#DeviceTree).

Zur Zeit läßt der Treiber bis zu 10 Sensoren am Bus zu. Sie haben jedoch die Möglichkeit, durch einen Eintrag in der Datei /etc/modprobe.d/1-wire.conf diese Zahl zu erhöhen. Falls die Datei noch nicht exisitert, ist sie neu zu erstellen. Dann kann mit der folgenden Zeile die Anzahl der möglichen Sensoren, z. B. auf 15, erhöt werden:

options wire max\_slave\_count=15

Diese Änderung wird aber nur beim Laden des Treibers übernommen, gegebenenfalls erst nach dem nächsten Neustart.

Ich musste aber feststellen, dass mehr als 10 Sensoren am Bus nicht sinnvoll sind, da der Raspberry Pi mit seinem 3.3-V-Eingang eventuell schon bei 8 - 9 Sensoren überfordert scheint. Ich konnte acht Sensoren problemlos beteiben, wobei die Verbindung über recht lange, ungeschrimte Kabel (Telefonleitung) erfolgt; beim neunten Sensor "hing" dann alles. Gegebenenfalls kann man den Pullup-Widerstand auf 2,2 oder sogar 1,2 Kiloohm reduzieren und eventuell noch ein RC-Filter einbauen (siehe Maxim AN148, im Literatur anhang).

Der Zugriff auf die Sensoren erfolgt über das Dateisystem. Sie müssen auch nicht root sein, es darf jeder User lesend darauf zugreifen. Jeder Sensor wird mit seiner ID als Verzeichnis eingebunden. Ist die Hardware angeschlossen und sind die Kernel-Module geladen, können die Temperaturwerte über eine Pseudo-Datei abgerufen werden. Wie gesagt, bekommt jeder Sensor ein eigenes Verzeichnis unter /sys/bus/w1/devices/ (symbolische Links). Innerhalb des Verzeichnis eines Sensors kann die Temperatur aus der (Pseudo-)Datei w1\_slave gelesen werden, beispielsweise durch:

root@raspberrypi: cat /sys/bus/w1/devices/10-000802cfb15d/w1\_slave

33 00 4b 46 ff ff 02 10 f4 : crc=f4 YES

33 00 4b 46 ff ff 02 10 f4 t=25625

Das Ergebnis des Sensors mit der ID 10-000802cfb15d besteht aus zwei Zeilen, die jeweils mit der hart verdrahteten ID des Bausteins und Angaben zur Prüfinfo bzw. dem Ergebniswert enthalten. Die erste Zeile teilt uns mit, ob die Prüfinfo korrekt war ("YES"), in der zweiten Zeile filden wir die Temperaturangabe in Tausenstel Grad (t=25625 → 25,625 °C). Das Datenblatt des Sensors ernüchtert und aber, denn der Sensor misst keineswegs so genau. Vielmehr sind es beim Standard-Typ Abstufungen in größeren Schritten. Aber da das Weiterrechnen einfacher ist, wenn man den Wert nur durch 1000 teilen muss (oder nur an der passenden Stelle ein Komma in den String pfriemeln), wird der Wert halt so übermittelt.

Im Verzeichnis /sys/bus/w1/devices/w1\_bus\_master1 gibt es noch einmal die Wiederholung der Verzeichnisse für die Sensoren und etliche interessante weitere Dateien, von denen einige besprochen werden:

root@raspberrypi:/sys/bus/w1/devices/w1\_bus\_master1# ls -l

insgesamt 0

drwxr-xr-x 3 root root 0 Mai 13 07:29 10-000802bf634d

drwxr-xr-x 3 root root 0 Mai 13 07:29 10-000802cfb15d

lrwxrwxrwx 1 root root 0 Mai 13 07:30 driver

-> ../../bus/w1/drivers/w1\_master\_driver

drwxr-xr-x 2 root root 0 Mai 13 07:30 power

lrwxrwxrwx 1 root root 0 Mai 13 07:29 subsystem -> ../../bus/w1

-rw-r--r-- 1 root root 4096 Mai 13 07:29 uevent

-rw-rw-r-- 1 root root 4096 Mai 13 07:30 w1\_master\_add

-r--r--r-- 1 root root 4096 Mai 13 07:30 w1\_master\_attempts

-r--r--r-- 1 root root 4096 Mai 13 07:30 w1\_master\_max\_slave\_count

-r--r--r-- 1 root root 4096 Mai 13 07:30 w1\_master\_name

-r--r--r-- 1 root root 4096 Mai 13 07:30 w1\_master\_pointer

-rw-rw-r-- 1 root root 4096 Mai 13 07:30 w1\_master\_pullup

-rw-rw-r-- 1 root root 4096 Mai 13 07:30 w1\_master\_remove

-rw-rw-r-- 1 root root 4096 Mai 13 07:30 w1\_master\_search

-r--r--r-- 1 root root 4096 Mai 13 07:30 w1\_master\_slave\_count

-r--r--r-- 1 root root 4096 Mai 13 07:30 w1\_master\_slaves

-r--r--r-- 1 root root 4096 Mai 13 07:30 w1\_master\_timeout

Die Datei w1\_master\_max\_slave\_count enthält die maximal mögliche Zahl der angeschlossenen Sensoren (default: 10). Das Kommando cat w1\_master\_pullup liefert uns die Einstellung des Pullup-Parameters des Treiberaufrufs. Drei weitere Dateien geben Auskunft über Anzahl der aktiven Sensoren ("Slaves"), den Timeout bei der Messung und die IDs dar angeschlossenen Slaves:

cat w1\_master\_slave\_count

2

cat w1\_master\_timeout

10

cat w1\_master\_slaves

10-000802bf634d

10-000802cfb15d

Die Datei w1\_master\_slaves ermöglicht recht einfachen Zugriff auf die Slaves. Für den ersten Test können Sie folgende Schleifen in der Shell eingeben:

for dev in $(cat /sys/bus/w1/devices/w1\_bus\_master1/w1\_master\_slaves)

do

cat /sys/bus/w1/devices/w1\_bus\_master1/$dev/w1\_slave

done

Wenn Sie nur die Temperaturwerte in Grad wollen, können Sie die Schleife modifizieren:

for dev in $(cat /sys/bus/w1/devices/w1\_bus\_master1/w1\_master\_slaves)

do

tail -1 /sys/bus/w1/devices/w1\_bus\_master1/$dev/w1\_slave |\

awk -F 't=' '{print $2/1000}'

done

Als nächstes sollen die Daten per Python-Script abgefragt werden. Durch Erweiterung des folgenden Programms lassen sich die Daten auch in eine Datei schreiben oder in einer Datenbank speichern. In beiden Fällen ist aber die SD-Karte nicht das geeignete Speichermedium. Immer wenn auf der Platte geschrieben oder gelesen werden muss, ist ein externes Speichermedium angebracht, z. B. ein USB-Stick oder eine USB-Festplatte (die aber dann mit einem aktiven Hub zur Stromversorgung). Siehe auch [RasPi\_Laufwerke](http://www.netzmafia.de/skripten/hardware/RasPi/RasPi_Laufwerke.html). Das Script sollte sich durch die Kommentare selbst erklären:

#!/usr/bin/python

# -\*- coding: utf-8 -\*-

# 1-Wire Slave-Liste lesen

file = open('/sys/devices/w1\_bus\_master1/w1\_master\_slaves')

w1\_slaves = file.readlines()

file.close()

# Fuer jeden 1-Wire Slave aktuelle Temperatur ausgeben

for line in w1\_slaves:

# 1-wire Slave extrahieren

w1\_slave = line.split("\n")[0]

# 1-wire Slave Datei lesen

file = open('/sys/bus/w1/devices/' + str(w1\_slave) + '/w1\_slave')

filecontent = file.read()

file.close()

# Temperaturwerte auslesen und konvertieren

stringvalue = filecontent.split("\n")[1].split(" ")[9]

temperature = float(stringvalue[2:]) / 1000

# Temperatur ausgeben

print(str(w1\_slave) + ': %6.2f °C' % temperature)

Die Ausgabe dazu sieht bei mir so aus:

10-000802bf634d: 24.25 °C

10-000802cfb15d: 24.38 °C

**Programmierung in Perl**

Da ich ja mehr der Shell-Perl-C-Fraktion angehöre, schiebe ich auch gleich noch ein Perl-Script nach. Das Programm hat auch gegenüber der Python-Version noch einige Erweiterungen:

Es wird geprüft, ob die beiden notwendigen Module schon geladen sind und falls nicht, werden sie nachgeladen. Dazu wird die Pseudo-Datei /proc/modules gelsen, die im Prinzip dieselbe Info enthält, wie die Ausgabe des Kommandos lsmod, nur nicht schön formatiert.

Danach wird, genau wie beim Python-Programm, die Liste der aktiven Sensor-IDs gelesen und anschließend fragt das Programm die Sensoren einzeln ab. Für das Testen erzeugt das Programm auf Wunsch zusätzliche Debug-Ausgaben, die sich mit dem Kommandozeilenschalter "-v" aktivieren lassen.

#!/usr/bin/perl

use strict;

use warnings;

# fuer Verbosity auf 1 setzen, default 0

my $DEBUG = 0;

# Dateipfad zu die Pseudo-Dateien der Temperaturmessung

my $PATH = '/sys/bus/w1/devices/w1\_bus\_master1/';

# hier stehen die Device-Ids drin

my $SLAVES = $PATH . 'w1\_master\_slaves';

my %deviceIDs; # IDs der Temperatursensoren

my %temperatures; # aktuelle Temperaturen

my $arg =''; # Kommandozeilenargument

my ($key, $ID, $value);

# Kommandozeilenparameter (-v fuer verbose)

if ($#ARGV >= 0) { $arg = $ARGV[0]; }

if ($arg eq '-v') { $DEBUG = 1; }

# Nachsehen, ob die Module geladen sind

# gegebenenfalls laden

check\_modules();

# Ermitteln der Device-IDs

get\_device\_IDs();

for $key (sort(keys %deviceIDs))

{

$ID = $deviceIDs{$key};

$value = read\_device($ID);

if ($value ne 'U')

{

$temperatures{$key} = $value;

printf ("%2d: ID=%s %6.2f\n", $key, $ID, $value);

}

}

exit 0;

sub check\_modules

{

# Nachsehen, ob die Module geladen sind

# gegebenenfalls nachladen

my $mods;

# alle geladenen Module stehen in der Datei /proc/modules

open (DATEI, '<', '/proc/modules') or

die "Unable to open /proc/modules: $!";

$mods = join(' ',<DATEI>);

close (DATEI);

if ($mods =~ /w1\_gpio/ && $mods =~ /w1\_therm/)

{

# alles O.K., Module sind da

if ($DEBUG) { print "w1 modules already loaded \n"; }

}

else

{

# Nein, das Nachladen versuchen

if ($DEBUG) { print "loading w1 modules \n"; }

system("sudo modprobe w1-gpio");

system("sudo modprobe w1-therm");

}

}

sub get\_device\_IDs

{

# Einlesen der IDs aller am Bus vorhandenen Sensoren

# und Speichern in einem Hash

my $index = 0;

open(INFILE, '<', $SLAVES) or die("Unable to open $SLAVES: $!");

while(<INFILE>)

{

chomp;

$deviceIDs{$index} = $\_;

if ($DEBUG) { print "$index: $\_ \n"; }

$index++;

}

close(INFILE);

}

sub read\_device #(ID)

{

# Liest die Temperaturwerte eines Sensors aus. Die ID des

# Sensors wird als Parameter uebergeben, Rueckgabewert ist

# die gemessene Temperatur.

# Liefert das Ergebnis eine falsch Pruefinfo, wie 'U' (fuer

# 'undefined') zurueckgegeben.

my $deviceID = shift;

my $filename = $PATH . $deviceID . '/w1\_slave';

my $sensordata;

open (DATEI, '<', $filename) or die "Unable to open $filename: $!";

$sensordata = join(' ',<DATEI>);

close (DATEI);

if($sensordata =~ m/YES/)

{

#fix for negative temps from http://habrahabr.ru/post/163575/

$sensordata =~ /t=(\D\*\d+)/i;

$sensordata = ($1/1000);

return ($sensordata);

}

else

{

if ($DEBUG) { print ("CRC Invalid for $deviceID.\n$sensordata\n"); }

return ('U');

}

}

Die Ausgabe des Programms ist ebenso knapp wie bei Python:

0: ID=10-000802bf634d 17.81

1: ID=10-000802cfb15d 18.00

Eigentlich soll der Raspberry Pi keine Datenbanken, ständig wachsende Dateien, Grafikausgabe oder einen Webserver beherbergen. Das alles soll ein zentraler Server erledigen, der sowieso schon die genannten Systeme laufen hat. Daher wird das Perl-Programm zu einem kleinen Server erweitert. Sobald eine Anfrage auf dem (frei gewählten) Port 2000 eintrifft, übergibt der RasPi bereitwillig die Daten. Das Programm zur Datenabfrage ist genauso einfach. Näheres können Sie im [Skript "Netzwerk-Programmierung"](http://www.netzmafia.de/skripten/inetprog/inetprog2.html) nachlesen. Dort werden auch die Hintergründe erläutert. Insgesamt ist die Erweiterung mit nur wenigen Zeilen erledigt:

#!/usr/bin/perl

use strict;

use warnings;

use IO::Socket;

use strict;

# fuer Verbosity auf 1 setzen, default 0

my $DEBUG = 0;

# Portnummer fuer den Server

use constant MYPORT => 2000;

# Dateipfad zu die Pseudo-Dateien der Temperaturmessung

my $PATH = '/sys/bus/w1/devices/w1\_bus\_master1/';

my $SLAVES = $PATH . 'w1\_master\_slaves';

my %deviceIDs; # IDs der Temperatursensoren

my %temperatures; # aktuelle Temperaturen

my $arg =''; # Kommandozeilenargument

my $sock = ''; # fuer den Server

my $client = 0;

my ($key, $ID, $value, $sendestring);

# Kommandozeilenparameter (-v fuer verbose)

if ($#ARGV >= 0) { $arg = $ARGV[0]; }

if ($arg eq '-v') { $DEBUG = 1; }

# Nachsehen, ob die Module geladen sind

# gegebenenfalls nachladen

check\_modules();

# Ermitteln der Device-IDs

get\_device\_IDs();

# Server starten ...

$sock = new IO::Socket::INET(LocalPort => MYPORT,

Reuse => 1,

Listen => 5)

or die "can't create local socket: $@\n";

if ($DEBUG) { print "Accepting connections on Port ", MYPORT, "...\n"; }

# ... und auf den client warten

while ($client = $sock->accept())

{

# Eine Verbindung ist eingetroffen.

if ($DEBUG) { print "Accepted connection from ",

$client->peerhost(), ":", $client->peerport(), "\n"; }

# Daten senden

for $key (sort(keys %deviceIDs))

{

$ID = $deviceIDs{$key};

$value = read\_device($ID);

if ($value ne 'U')

{

$temperatures{$key} = $value;

$sendestring = sprintf ("%2d: ID=%s %6.2f\n", $key, $ID, $value);

# Daten zum Client schicken

print $client $sendestring;

}

}

# Verbindung beenden

$client->close() if defined $client;

}

exit 0;

sub check\_modules

{

# Nachsehen, ob die Module geladen sind

# gegebenenfalls laden

my $mods;

open (DATEI, '<', '/proc/modules') or die "Unable to open /proc/modules: $!";

$mods = join(' ',<DATEI>);

close (DATEI);

if ($mods =~ /w1\_gpio/ && $mods =~ /w1\_therm/)

{

if ($DEBUG) { print "w1 modules already loaded \n"; }

}

else

{

if ($DEBUG) { print "loading w1 modules \n"; }

system("sudo modprobe w1-gpio");

system("sudo modprobe w1-therm");

}

}

sub get\_device\_IDs

{

# Einlesen der IDs aller am Bus vorhandenen Sensoren

# und Speichern in einem Hash

my $index = 0;

open(INFILE, '<', $SLAVES) or die("Unable to open $SLAVES: $!");

while(<INFILE>)

{

chomp;

$deviceIDs{$index} = $\_;

if ($DEBUG) { print "$index: $\_ \n"; }

$index++;

}

close(INFILE);

}

sub read\_device

{

# Liest die Temperaturwerte eines Sensors aus. Die ID des

# Sensors wird als Parameter uebergeben, Rueckgabewert ist

# die gemessene Temperatur.

# Liefert das Ergebnis eine falsch Pruefinfo, wie 'U' (fuer

# 'undefined') zurueckgegeben.

my $deviceID = shift;

my $filename = $PATH . $deviceID . '/w1\_slave';

my $sensordata;

open (DATEI, '<', $filename) or die "Unable to open $filename: $!";

$sensordata = join(' ',<DATEI>);

close (DATEI);

if($sensordata =~ m/YES/)

{

#fix for negative temps from http://habrahabr.ru/post/163575/

$sensordata =~ /t=(\D\*\d+)/i;

$sensordata = ($1/1000);

return ($sensordata);

}

else

{

if ($DEBUG) { print ("CRC Invalid for $deviceID.\n$sensordata\n"); }

return ('U');

}

}

Zum Testen können Sie einfach in einer zweiten Shell mit dem Kommando telnet localhost 2000 die Daten abrufen. Auf Serverseite kann man das verfolgen, wenn der Schalter "-v" mitgegeben wurde:

pi@raspberrypi ~ $ perl tempserver.pl -v

w1 modules already loaded

0: 10-000802bf634d

1: 10-000802cfb15d

Accepting connections on Port 2000...

Accepted connection from 10.10.10.91:51690

Accepted connection from 10.10.10.91:51691

...

Mit etwas "Vorspann" vor den Daten, genauer gesagt einem ordentlichen HTTP-Header und Einstellung auf Port 80 (Port 80 darf aber nur root) wären die Daten auch per Browser abrufbar. So läßt sich relativ preiswert ein verteiltes System von Sensorknoten aufbauen, die dann über das LAN ihre Daten an einen zentralen Server liefern. Der führt dann die Daten aller Stationen zusammen und könnte dann auch für Webgrafiken oder Ähnliches sorgen.

Noch zwei Anregungen zum Schluß:

* Man könnte noch im Perl-Programm einen Hash einrichten, der die Zuordnung der Sensor-ID zu einer Mess-Stelle erlaubt. Man kann so im Programm und bei der Ausgabe mit symbolischen Angaben arbeiten, zum Beispiel für die Überwachung von 19"-Server-Racks:
* my %sensors = (
* "10-000802bf634d" => "R1O", # Rack 1 oben
* "10-000802cfb15d" => "R1M", # Rack 1 mitte
* "10-0008009e8646" => "R1U", # Rack 1 unten
* "10-000800bc80e0" => "R2O", # Rack 2 oben
* "10-0008009e6889" => "R2M", # Rack 2 mitte
* "10-000800aacfe7" => "R2U", # Rack 2 unten
* "10-000800aa974f" => "Raum", # Raumtemperatur
* "10-0008009e700a" => "Klima",# Klimaanlage Geblaese
* );
* Wenn Sie im Perl-Programm noch einen Hash einrichten, der für jede Sensor-ID einen Korrekturwert enthält, dann können Sie die Sensoren noch genauer kalibrieren. Durch exakte Vergleichsmessung wird der Korrekturwert ermittelt und in dem Hash gespeichert. Bei der Messung wird dann der jeweilige Korrekturwert zum Messwert addiert.
* Der Sensor DS18x20 kann problemlos direkt mit einem längeren Kabel verlötet werden. Isolieren Sie die drei Anschlüsse mit dünnen Wärmeschrumpfschlauch und dann die gesamte Verbindung vom Kabelansatz bis über den Baustein hinaus mit einem dickern Schrumpschlauch. Dann ist der Sensor gegen Umwelteinflüsse geschützt und auch die Lötstelle knicksicher gemacht worden.



**Mehr Sensoren?!**

Eine Möglichkeit, die Anzahl der angeschlossenen Sensoren zu erhöhen, wäre das Ausweichen auf andere Schnittstellen, z. B. seriell oder I2C. Dazu bräuchte man dann einen sogenannten Hostadapter wie er z. B. von Sheepwalk Electronics angeboten wird (<http://www.sheepwalkelectronics.co.uk/product_info.php?products_id=30> oder <http://www.sheepwalkelectronics.co.uk/product_info.php?products_id=33>) Es können dann mehrere Hostadapter angeschlossen werden, die ihrerseits jeweils eine bestimmte Anzahl Sensoren verwalten.

Eine weitere Idee wäre das Gruppieren der Sensoren, wie es unter "Switched Networks" in den "Guidelines for Reliable Long Line 1-Wire Networks" (AN148, siehe Literatur) beschrieben ist. Für die Umschaltung könnten digitale Multiplexer oder auch Analogschalter verwendet werden. Damit ist im Grunde keine Grenze mehr nach oben, man kann halt immer nur 10 Sensoren gleichzeitig abfragen und muss in der Software beim Umschalten wohl auch auf das Timing im Treiber achten (Delay oder schlimmstenfalls Treiber entladen und wieder laden).

Eventuell kann man auch beide Ideen kombinieren. Man nimmt den Seriell-zu-1-Wire-Hostadapter DS2480B für die Anbindung einer Sensor- Gruppe und multiplext dann TXD/RXD der seriellen Verbindung (sind ja TTL-Pegel).

**Die gefühlte Temperatur**

Jeder hat schon in einem Wetterbericht den Begriff "Windchill" oder "gefühlte Temperatur" gehört. Damit soll berücksichtigt werden, dass bei starkem Wind die von einem Menschen gefühlte Temperatur niedriger sein kann als die tatsächliche Temperatur.

Die Lufttemperatur, die wir empfinden, wird als "gefühlte Temperatur" bezeichnet. Diese gefühlte Lufttemperatur ist von vielen Faktoren abhängig, beispielsweise

* von der Luftfeuchtigkeit,
* von der Windgeschwindigkeit,
* davon, ob die Sonne scheint oder nicht,
* von der Art unserer Bekleidung,
* davon, ob wir uns körperlich bewegen oder in Ruhe sind.

Die "gefühlte Temperatur" ist damit von Mensch zu Mensch sehr unterschiedlich. Es gibt, neben den oben genannten, weitere Parameter, die hier eine Rolle spielen. Sogar eine Hautcreme kann das Temperaturempfinden verändern. Dann gibt es eine genetisch verankerte Widerstandsfähigkeit gegenüber tiefen Temperaturen, und auch der geschlechtsspezifische Unterschied ist unübersehbar. Frauen frieren meist schneller als Männer. So gibt es nicht nur eine Definitionen für die "gefühlte Temperatur".

Die Angabe der "gefühlten Temperatur" im Wetterbericht bezieht sich auf einen Modell-Menschen. Der Deutsche Wetterdienst verwendet dazu das Klima-Michel-Modell. Dies ist ein computersimulierter Durchschnittsmensch mit folgenden Eigenschaften: Es ist ein 35 Jahre alter und 1,75 Meter großer Mann mit einem Körpergewicht von 75 Kilogramm, der sich mit ca. 5 km/h im Freien fortbewegt und der Witterung entsprechend gekleidet ist.

Man kann die gefühlte Temperatur nach der folgenden, relativ einfachen Formel ausrechnen, in die nur die tatsächliche Temperatur (t) in Grad Celsius und die Windgeschwindigkeit (v) in km/h eingehen. Der Windchill (W) in Grad Celsius ergibt sich aus:

W = 13,12 + 0,6215 \* t – 11,37 \* v0,16 + 0,3965 \* t \* v0,16

Ein Beispiel dazu: Bei einer Temperatur von –5 Grad Celsius und einer Windgeschwindigkeit von 15 Kilometer pro Stunde beträgt die gefühlte Temperatur rund –11 Grad Celsius.

**Literatur und Links**

* [Guidelines for Reliable Long Line 1-Wire Networks](http://pdfserv.maximintegrated.com/en/an/AN148.pdf)
* [1-Wire Communication Through Software](http://pdfserv.maximintegrated.com/en/an/AN126.pdf)
* [Using the DS2480B Serial 1-Wire Line Driver](http://pdfserv.maximintegrated.com/en/an/AN192.pdf)
* [1-Wire Search Algorithm](http://pdfserv.maximintegrated.com/en/an/AN187.pdf)
* [Python-Programm DS1820.py](http://www.netzmafia.de/skripten/hardware/RasPi/Projekt-Onewire/DS1820.py)
* [Perl-Programm DS1820.pl](http://www.netzmafia.de/skripten/hardware/RasPi/Projekt-Onewire/DS1820.pl)
* [Perl-Client-Server-Programm DS1820\_1.pl](http://www.netzmafia.de/skripten/hardware/RasPi/Projekt-Onewire/DS1820_1.pl)

***Copyright © Hochschule München, FK 04, Prof. Jürgen Plate*  
Letzte Aktualisierung: 12/26/2017 10:37:07**