

Tischmultimeter M9803R

Auslesen der Schnittstelle mittels Perl-Programm

Jürgen Plate, 12. Oktober 2011

Dieser Artikel will anhand einer Aufgabenstellung aus der Praxis nicht nur zeigen, wie Sie mit einem Perl-Programm die Lösung realisieren, sondern auch, welche Hürden und Fallstricke oft bei scheinbar einfach zu lösenden Problemen lauern.

Es gibt im Handel diverse Multimeter mit einer seriellen Schnittstelle, darunter einige Hand- und Tischmultimeter – also digital anzeigende Messinstrumente für Strom, Spannung, Widerstand usw. Ein Tischmultimeter, das Modell M9803R (Bild 1), gibt es bei verschiedenen Anbietern zu Preisen zwischen 80 und 120 Euro (z. B. bei Pollin für ca. 80 Euro, Bestell-Nr. 830 256), was für ein Gerät mit den beschriebenen Eigenschaften schon recht günstig ist. Das Display verfügt auch noch über eine Balkenanzeige, und neben Strom, Spannung und Widerstand sind auch Kapazitäts- und Frequenzmessungen möglich. Die Stromversorgung kann stationär über das Netz oder mit zwei verschiedenen Batteriesätzen erfolgen. Bei <http://www.pollin.de> kann man sich auch vorab die Bedienungsanleitung herunterladen.



Bild 1: Ansicht des M9803R

Dem Gerät liegt eine CD mit Windows-Software bei, aber der Ehrgeiz bestand ja darin, die Daten mit einem eigenen Programm einzulesen. Eine Suche im Web förderte leider nur wenige Informationen zutage, was möglicherweise auch daran liegt, dass die Schnittmenge zwischen Computer- und Elektronikfreaks immer kleiner wird. Immerhin fand ich das Datenprotokoll unter http://www.multimeterwarehouse.com/Ms344_345_9803R_dataprotocol.txt. Als Übertragungsparameter sind dort sieben Datenbits, kein Paritätsbit, zwei Stoppbits (7N2) angegeben.

Leider sind die dort angegebenen Informationen nicht ganz richtig, was zu einer längeren Forschungsnacht führte. Mit einem einfachen Programm, das lediglich die Daten einliest und hexadezimal auf dem Bildschirm ausgibt, begannen die ersten Versuche. Es kamen auch Daten, die aber keineswegs dem beschriebenen Protokoll entsprachen. Zudem gab es zwischendurch immer mal Fehlermeldungen (Framing Error), die darauf hinwiesen, dass 7N2 nicht das richtige Übertragungsformat sein konnte. Auch Versuche mit gerader und ungerader Parität führten nicht zum Ziel. Das Format war sowieso recht verdächtig, denn seit dem Aussterben mechanischer Fernschreiber verwendet eigentlich niemand mehr zwei Stoppbits. Beim nächsten Versuch mit dem Allerweltsformat acht Datenbits, ohne Parität und mit einem Stoppbit (8N1) kamen zumindest keine Fehlermeldungen mehr, auch wenn die Daten immer noch seltsam aussahen.

Die Funktion des höchstwertigen Bits war aber immer noch unklar. Es handelte sich jedenfalls nicht um ein Paritätsbit, sondern es wird anscheinend unabhängig vom Inhalt nur bei bestimmten Bytes gesetzt. Es ist nur bei den Bytes 0, 5, 6, 7 und 8 auf 1 gesetzt und bei den anderen Bytes auf 0. Möglicherweise will der Hersteller auf diese Weise das Auslesen des eigentlichen Zahlenwertes und das Erkennen des Endes eines Datensatzes erleichtern. Beim Messbereich (Byte 5) kann nämlich auch der Wert $0 \times 0a$ (Newline) auftreten, der auch als letztes Zeichen eines Datensatzes kommt – eine Erkennung des Datensatzendes mit Test auf $0 \times 0a$ ist daher nicht möglich. Im Programm verwende ich daher 8N1 als Datenformat und werfe das höchstwertige Bit erst einmal weg.

Bei der Beschreibung des Protokolls gibt es noch einen weiteren Fehler auf der oben angegebenen Webseite: Byte Nummer 1 enthält die niederwertigste Stelle des Messwerts (also die rechte Stelle) und Byte Nummer 4 die höchstwertige (linke) Stelle, nicht umgekehrt. Außerdem liegen die vier

Ziffern des Messwerts nicht als ASCII-Zeichen, sondern als BCD-Zahlen, also binär vor. Damit stellt sich das Protokoll folgendermaßen dar:

```

Byte 0:  0 0 0 VZ BAT 0 OR
          VZ: 0 for positive, 1 for negative
          BAT: 0 for normal, 1 for low battery
          OR: 0 for normal, 1 for Over Range

Byte 1:  Anzeigeziffer (Digit 4) binaer
Byte 2:  Anzeigeziffer (Digit 3) binaer
Byte 3:  Anzeigeziffer (Digit 2) binaer
Byte 4:  Anzeigeziffer (Digit 1) binaer

Byte 5:  Unit / Messbereich
          0000000 DC V
          0000001 AC V
          0000010 DC mA
          0000011 AC mA
          0000100 Ohm
          0000101 Buzzer
          0000110 Diode
          0000111 ADP
          0001000 DC A
          0001001 AC A
          0001010 Hz
          0001011 ?
          0001100 CAP

Byte 6:  Range / Messbereich
          0000000 400 mV 4 mA 400 Ohm 10 kHz 4 nF
          0000001 4 V 40 mA 4 kOhm 100 kHz 40 nF
          0000010 40 V 400 mA 40 kOhm 1000 kHz 400 nF
          0000011 400 V 4 A 400 kOhm 4000 nF
          0000100 4000 V 4000 kOhm 40000 nF
          0000101 40000 kOhm 100 Hz
          0000110 1000 Hz
          Diode: 4 V

          Buzzer: 400 Ohm

          DC A and AC A: 40 A

Byte 7:  Special Function 1
          0 0 0 max min rel hold
          0 for normal, 1 for the function

Byte 8:  Special Function 2
          0 0 0 mem auto manl apof
          0 for normal, 1 for the function

Byte 9:  CR (Carriage Return, 0x0d)
Byte 10: LF (Line Feed, 0x0a)

```

Bevor dieses Problem gelöst wurde, war aber noch ein Kampf mit der Hardware auszufechten, denn die serielle Schnittstelle zeigte zunächst ein seltsames Verhalten auf der elektrischen Seite: Auf dem Oszilloskop zeigte sich, dass keineswegs die vorgeschriebenen Pegel eingehalten wurden (glücklicherweise kommen die meisten seriellen Schnittstellen oft auch mit solchen Signalen zurecht – sonst hätte ich vermutlich schon im Vorfeld aufgegeben). Ich erinnerte mich aber an ein vor Jahren eingesetztes Board zur Erfassung von Analogdaten, das ähnliche Eigenschaften aufwies. Aufschrauben des M9803R brachte Klarheit. Logischerweise musste eine strikte elektrische Trennung zwischen Messgerät und Schnittstelle erfolgen, da ja Spannungen bis zu 4000 Volt gemessen werden können.

Bild 2 zeigt den „Optokoppler“, der hier nicht als integrierte Schaltung vorliegt, sondern aus einer normalen IR-LED als Sender und einer Photodiode als Empfänger besteht. Da nun die Schnittstelle vom Rest des Messgeräts entkoppelt sein soll und sich der Hersteller wohl ein zweites Netzteil nur für die Schnittstelle gespart hat, muss diese vom Computer aus versorgt werden. Da nur gesendet wird, stehen alle anderen Leitungen der Schnittstelle zur Verfügung, um die RS232-Schnittstelle mit positiver und negativer Spannung zu versorgen. Wenn Ihnen jetzt die Leitungen DTR und RTS einfallen, ist das eigentlich genau richtig. Man setzt beispielsweise DTR auf 1 (+12 V) und RTS auf 0 (-12 V) und hat, was man benötigt.

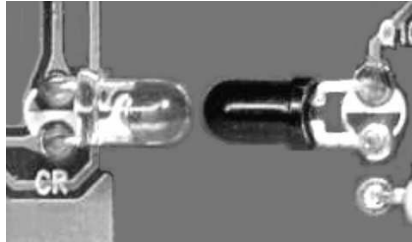


Bild 2: Optische Übertragung der seriellen Daten (links LED, rechts Photodiode)

Schon wieder mit Zitronen gehandelt! Die fernöstlichen Entwickler, ausgestattet mit einem ganz besonderen Humor, verwenden zwar DTR für die positive Spannung, aber für die negative Spannung muss TXD herhalten. Da bei RXD und TXD die Zuordnung der Spannungen zu 0 und 1 gerade andersherum ist als bei den Steuerleitungen, erhöht das den Spaß beim Programmieren. Für diejenigen, die sich etwas mit Elektronik auskennen, habe ich versucht, die Interface-Schaltung herauszufinden. Bild 3 gibt die Schaltung ohne Gewähr wieder.

Damit war aber auch das letzte Geheimnis des Geräts gelüftet, und es konnte endlich ans Programmieren gehen. Das ganze „Reverse Engineering“ bis dahin hat fünfmal so lange gedauert, wie das Schreiben des folgenden Basisprogramms. Hätte der Hersteller sein Handbuch um eine zweiseitige Schnittstellenbeschreibung erweitert, wäre mir die Arbeit erspart geblieben.

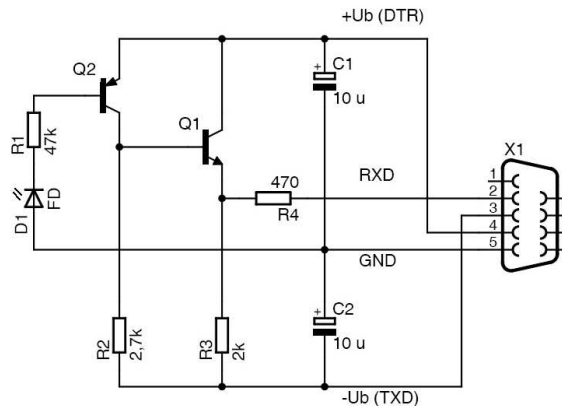


Bild 3: Schaltung der seriellen Schnittstelle des M9803R

Neben den beschriebenen Problemen mit der Schnittstelle sind mir einige Unsauberkeiten in den Funktionen aufgefallen: Bei der Frequenzmessung ist die Übertragung mitunter unregelmäßig. Mal kommen die Daten wesentlich schneller als bei den anderen Messbereichen, mal gibt es längere Pausen. Manchmal werden bei kurzgeschlossenen Eingängen auch noch Restspannungen angezeigt, was wohl am internen Wandlungsverfahren liegt. Andererseits bekommt man doch relativ viel fürs Geld und kann Langzeitmessungen per Computer durchführen.

Ich vertrete bei Langzeiterfassung von Messwerten die Devise, die Darstellung der sich ergebenden Zeitreihe nicht dem Messprogramm zu überlassen, sondern getrennt davon durchzuführen. Sind die Daten mal erfasst, kann man sich nämlich noch alle möglichen Formen der Weiterverarbeitung und Darstellung überlegen. Für die Darstellung von Zeitreihen lassen sich z. B. Programme wie MRTG oder RRD-Tool verwenden. Aber auch eine Speicherung in einer Datenbank wäre möglich. Darum lasse ich (auch aus Platzgründen) die Weiterverarbeitung im folgenden Programm bewußt offen.

Nach der üblichen Initialisierung werden die Leitungen DTR und TXD entsprechend den Anforderungen für die Stromversorgung gesetzt. Danach setze ich für die Beendigung einen Signalhandler für Strg-C, der nichts weiter macht, als die Schnittstelle zu schließen. Im Hauptprogramm werden dann immer 11 Datenbytes eingelesen, das achte Bit auf Null gesetzt und die Daten in einem Array abgelegt. Dieses Array wird dann mit der Funktion `analyze()` weiterbearbeitet. In diesem Teil des Programms wäre noch etwas Arbeit nötig: Wenn aus irgendeinem Grund die Daten mal asynchron laufen (also Byte 0 vom Messgerät nicht in Byte 0 des Arrays landen), müsste man wieder aufsynchronisieren. Dazu könnten das achte Bit der Datenworte neben dem Datensatzende aus CR und LF herangezogen werden. Ich habe auf jeden Fall mal die Debug-Ausgaben im Programm gelassen.

Der Messwert wird immer als vierstellige Zahl ohne Dezimalpunkt und ohne Vorzeichen geliefert. Das Vorzeichen muss man sich aus den Datenbyte 0 holen. Die Hashes am Programmfang definieren die Teilfaktoren, durch die der Messwert geteilt werden muss, um den realen Wert zu ergeben. Wenn beispielsweise der Messbereich „40 V“ eingestellt ist und der Messwert „1020“ beträgt, muss er durch 100 geteilt werden, um die korrekte Gleitkommazahl 10,20 zu erhalten. In der Funktion `analyze()` werden abhängig vom Messbereich (Datenbyte 5) und Range (Datenbyte 6) die entsprechenden Hashwerte gewählt. Hier greife ich auch auf das Modul `Switch` zurück, um eine übersichtliche `switch-case`-Struktur im Programm zu erzielen. Neben dem Messwert und der Dimension werden auch die Datenbytes 0, 7 und 8 ausgewertet und ins Ergebnis übernommen. Daher liefert die Funktion auch eine Liste aus drei Werten: den Messwert einmal als reine Gleitkommazahl und einmal als Ausgabestring mit Dimensionsangabe sowie die Spezialtastenfunktionen als String.

```

use strict;
use warnings;
use Win32::SerialPort;
use Term::ReadKey;
use Switch;

# Software fuer Messgeraet M9803R

# fuer Debugging auf 1 setzen
my $DEBUG = 1;

# Definition von Dimension und Teilfaktoren fuer die
# einzelnen Messbereiche
# Messbereiche (Byte 5)
my %bereich = ( 0 => 'V DC', 1 => 'V AC',
               2 => 'mA DC', 3 => 'mA AC',
               4 => 'kOhm', 5 => 'Buzzer',
               6 => 'Diode', 7 => 'ADP',
               8 => 'A DC', 9 => 'A AC',
               10 => 'kHz', 11 => '?',
               12 => 'nF',
               );

# Teilfaktoren (Byte 6)
# Teilfaktor Spannung (Bereich 0, 1)
my %u_range = ( 0 => 10000.0, 1 => 1000.0,
                2 => 100.0, 3 => 10.0,
                4 => 1.0,
                );

# Teilfaktor Strom (Bereich 2, 3)
my %i_range = ( 0 => 1000.0, 1 => 100.0,
                2 => 10.0, 3 => 1.0,
                );

# Teilfaktor Widerstand (Bereich 4, 5)
my %r_range = ( 0 => 10000.0, 1 => 1000.0,
                2 => 100.0, 3 => 10.0,
                4 => 1.0, 5 => 0.1,
                );

# Teilfaktor Frequenz kHz (Bereich 10)
my %f1_range = ( 0 => 1000.0, 1 => 100.0,
                 2 => 10.0,
                 );

# Teilfaktor Frequenz Hz (Bereich 10)
my %f2_range = ( 5 => 100.0, 6 => 10.0,
                 );

# Teilfaktor Kapazitaet (Bereich 12)
my %c_range = ( 0 => 1000.0, 1 => 100.0,
                2 => 10.0, 3 => 1.0,
                4 => 0.1,
                );

# Teilfaktor Diode (Bereich 6)
my %d_range = ( 0 => 1000.0 );

# Teilfaktor Strom (Bereich 8, 9)

```

```

my %I_range = ( 0 => 100.0 );

# Signalhandler fuer Strg-C
$SIG{INT} = \&finish;

my $port = Win32::SerialPort->new('COM1')
    or die "Oops!\n";

$| = 1;
# Fehlermeldungen einschalten
$port->error_msg(1);
$port->user_msg(1);

# defined, weil "0" ein legaler Rueckgabewert ist
defined $port->parity_enable(1) || die "parity_enable geht nicht\n";

# Parameter setzen
$port->baudrate(9600)    || die "baudrate geht nicht\n";
$port->parity('none')   || die "parity geht nicht\n";
$port->databits(8)      || die "databits geht nicht\n";
$port->stopbits(1)     || die "stopbits geht nicht\n";
$port->handshake('none') || die "handshake geht nicht\n";

# Das Interface des Geraets wird vom Computer aus mit
# Strom versorgt. Empfang erfolgt normal ueber RXD,
# positive Spannung ueber DTR (muss daher auf 1 gesetzt werden)
# negative Spannung ueber TXD (muss daher auf 1 gesetzt werden)
$port->dtr_active(1)    || die "DTR geht nicht\n";
$port->break_active(1)  || die "TXD geht nicht\n";

# Wartezeit fuer Lesen eines Zeichens
$port->read_char_time(10);
# Wartezeit fuer Lese-Overhead
$port->read_const_time(100);

# Devicecontrolblock schreiben
$port->write_settings    || die "write_settings geht nicht\n";

my @data = ();
my $n = 0;

while(1)
{
    # Zeichen von der seriellen Schnittstelle lesen
    my $byte = $port->read(1);
    if ($byte)
    {
        $byte = (ord($byte) & 127);
        $data[$n] = $byte;
        # Datensatz (hoffentlich) komplett
        if ($n == 10)
        {
            if ($DEBUG)
            {
                print "Debug: ";
                printf "%02x ", $_ foreach (@data);
                print "\n";
            }
            # Datensatz scheint o.k. zu sein
            if ($byte == 0x0a)
            {
                # Datensatz auswerten
                my ($raw, $disp, $spec) = analyze(@data);
                print "$disp $spec\n";
                # hier koennte man die $raw-Daten in einem Array
                # fuer eine automatische Zeitreihen-Auswertung
                # (zusammen mit einem Zeitstempel) abspeichern
                # oder grafisch darstellen oder, oder ...
            }
            else
            {
                die "Out of Sync!\n";
            }
        }
        $n = ($n + 1) % 11;
    }
}

```

```

    }
}

sub finish
{
# Programm ordentlich beenden
$port -> close();
undef $port;
print "\nBye\n";
exit 0;
}

sub analyze # (@data)
{
# Datensatz auswerten
my @data = @_;
my $display = '';
my ($dimension, $faktor);
# Fehleranzeigen zuerst
return (0, 'Overflow', '') if ($data[0] & 0x01);
return (0, 'Battery low', '') if ($data[0] & 0x04);
# Messwert-Anzeige

# Vorzeichen
$display = '-' if ($data[0] & 0x08);
# 48 ist der Wert des ASCII-Zeichens "0"
$display .= chr(48 + $data[4]);
$display .= chr(48 + $data[3]);
$display .= chr(48 + $data[2]);
$display .= chr(48 + $data[1]);
my $value = 1.0 * $display;
print "Debug: $display - $value\n" if ($DEBUG);

# Messbereich und Teilfaktor bestimmen
$dimension = $bereich{$data[5]};
switch ($data[5])
{
case [0, 1] { $faktor = $u_range{$data[6]}; }
case [2, 3] { $faktor = $i_range{$data[6]}; }
case [4, 5] { $faktor = $r_range{$data[6]}; }
case [6]    { $faktor = $d_range{$data[6]}; }
case [8, 9] { $faktor = $I_range{$data[6]}; }
case [10]   { if ($data[6] < 3)
              {
                $faktor = $f1_range{$data[6]};
              }
              else
              {
                $faktor = $f2_range{$data[6]};
                $dimension = 'Hz';
              }
            }
case [12]   { $faktor = $c_range{$data[6]}; }
}
$value = $value/$faktor;
$display = sprintf("%4.3f %s", $value, $dimension);

# Special Functions
my $special = '';
$special .= 'Hold,' if ($data[7] & 0x01);
$special .= 'Rel,'  if ($data[7] & 0x02);
$special .= 'Min,'  if ($data[7] & 0x04);
$special .= 'Max,'  if ($data[7] & 0x08);
$special .= 'Apof,' if ($data[8] & 0x01);
$special .= 'Manl,' if ($data[8] & 0x02);
$special .= 'Auto,' if ($data[8] & 0x04);
$special .= 'Mem,'  if ($data[8] & 0x08);
$special =~ s/,,$//; # Komma am Ende weg

printf("Debug: %4.3f %s / %s\n", $value,
        $dimension, $special) if ($DEBUG);
return ($value, $display, $special);
}

```

Nun brauchen Sie nur noch das Gerät anzuschließen, das Programm zu starten und dann am Instrument die RS232-Taste zu drücken. Wie ich schon weiter oben angedeutet habe, bietet das Programm die Basis für viele weitere Ideen, etwa den Einbau in ein Perl-TK-Fenster, die grafische Anzeige des Messwerts als Zahl oder als Balken mithilfe des GD-Pakets, die Sammlung der Daten in einer Datenbank, das Darstellen von Messreihen per Webinterface und vieles mehr.