Monitorprogramm für den SH20

Der ETA SH20 bietet über eine RS232 Schnittstelle die Möglichkeit zum Austausch von Daten. Über die Schnittstelle können einige Werte ausgelesen und damit ein komfortables Monitoring ermöglicht werden, es können aber auch ein paar (wenige) Werte geschrieben und damit manipulativ auf den Kessel zugegriffen werden. Grund genug, ein kleines Programm zu schreiben, um mit dem Fox Board die Kesselkontrolle zu übernehmen.

Selber Schuld...

Sollte jemand, der das hier liest, auf die Idee des Nachbaus kommen und sollte diesem jemand dabei der Kessel, der Keller und/oder die ganze Hütte abrauchen, übernehme ich dafür keine Verantwortung. Dieser Bericht ist als Anregung zu verstehen, wie man einen Zugriff auf einen ETA-Kessel vornehmen kann, der Bericht versteht sich nur als vorschlagend. Sollte jemand das unten stehende Programm mit einem anderen ETA-Teil benutzen, das kein SH-Kessel ist, wird es Probleme geben. Die Beschreibungen hier beziehen sich NUR AUF ETA-SH Scheitholzkessel!

Weiterhin sei an dieser Stelle ausdrücklich betont, dass alle nachfolgenden Programmbeispiele für UNIX- und hier insbesondere für Linux-Rechner entwickelt wurden. Obwohl Portierungen auf andere OSes sicher möglich sind, bezieht sich diese Beschreibung nur auf das Betriebssystem Linux.

Bei uns im Keller steht ein ETA SH20 TWIN, ein Holzvergaser mit angeschlossenem Pelletbrenner. Da die Daten des Pelletkessels nicht relevant sind, habe ich auf Abfragen und Zugriffe auf diesen verzichtet und behandle den Kessel wie einen einfachen SH-Holzvergaserkessel.

Weitere benutzte Hardware

Wie oben schon erwähnt ist die Software für ein Fox Board entwickelt worden. Sie läuft zwar auch ohne Änderungen sofort auf anderen Linux-Rechnern, dennoch möchte ich an dieser Stelle nochmal erwähnen, warum ich mich für das Fox Board als Steuerrechner entschieden habe: Das Datalogging soll dauerhaft geschehen, der Rechner muss Tag und Nacht laufen. Ein normaler PC verbraucht dabei aber zu viel Strom, ist also zu teuer.

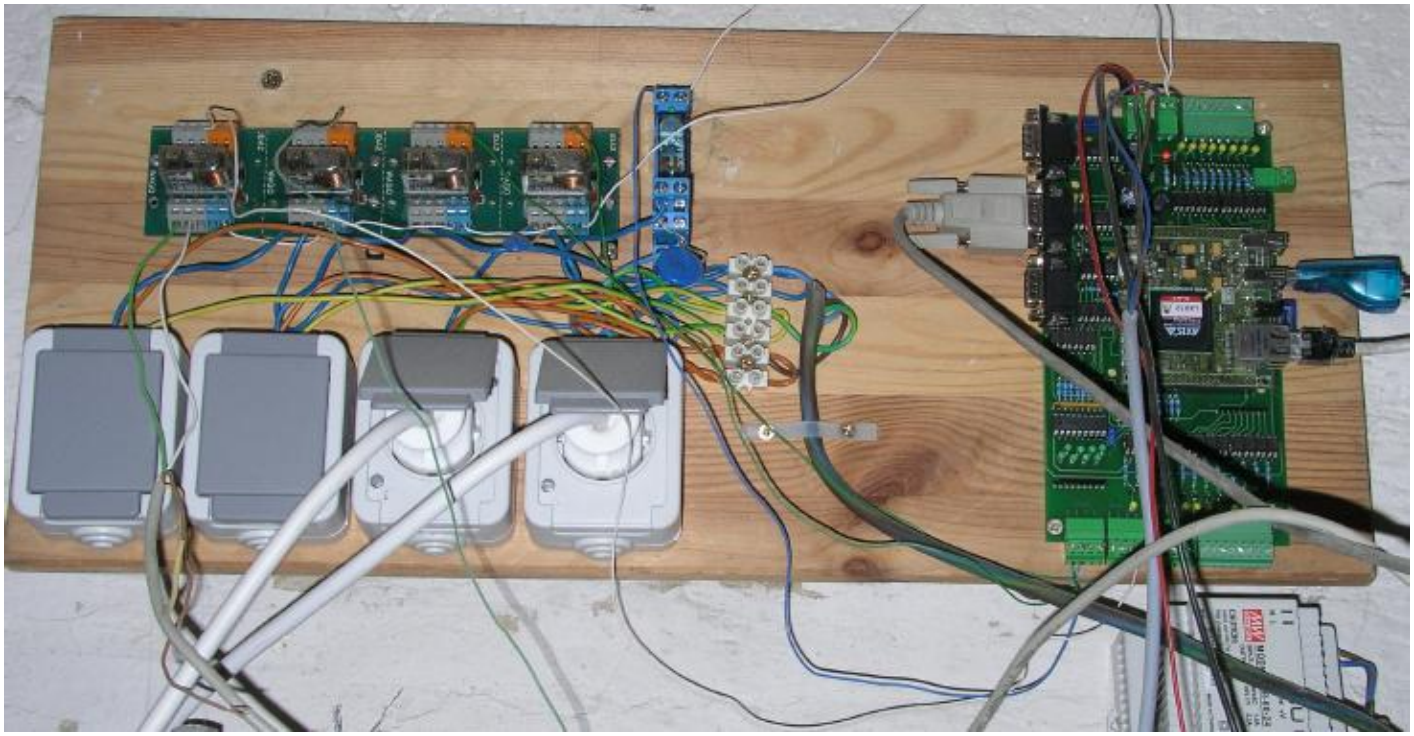

Bild 1. Das Fox Board mit Carrierbord als Steuerrechner im Keller

Das Fox Board, ein vollwertiger Linux-Rechner mit Web-Server und einem Stromverbrauch von einem Watt ist prädestiniert als Dauerrechner. Neben den Temperaturdaten des ETA-Kessels erfasst das Fox Board auch noch über ein 1-wire-System weitere Temperaturdaten, etwa die Temperaturen des Boilers in unterschiedlichen Höhen. Außerdem arbeitet das Fox Board als Steuerrechner und übernimmt einige Regelungsaufgaben, etwa die Strombereitstellung für den Hausstaubsauger oder den Badezimmerventilator. Und viel kosten tut das Fox Board auch nicht, für unter 150 Eu kann man es bekommen. Dazu kommen noch die Bauteile für die RS232, insbesondere der MAX3232 von Maxim, maximal noch mal ein Zehner. Und ein Carrierboart, wie im Bild 1. zu sehen, braucht man erst, wenn das Fox Board noch andere Aufgaben übernehmen soll.

Ziel der Programmierung - grundlegende Gedanken

Als ich anfing, mir über die Erstellung eines Programmes zum Zugriff auf den ETA-SH20 Gedanken zu machen, wollte ich etwas haben, was mir die aktuellen Temperaturwerte ins Wohnzimmer liefert. Damit ich also aus dem Sessel heraus sehen kann, wann Holz eingefüllt werden muss, bevor es mir die kälter werdenden Füße sagen. Aber nicht nur die Anzeige der Daten war mir wichtig, ich wollte auch in das "Heizgeschehen" eingreifen können.

Wir haben im Keller einen 150 Liter Boiler zur Bereitstellung von Warmwasser stehen. In den meisten Haushalten, in denen so ein Teil steht, wird das Wasser darin permanent auf einer festen Temperatur gehalten, rund um die Uhr, auch nachts oder wenn die Hausbewohner im Urlaub sind. Aus Zeiten, in denen wir noch mit Gas heizten, wussten wir, dass die Rundumdieuhrwarmwasserbereitstellung bei knapp 50 Grad mit täglichem Duschen weit über 20 kWh (mindestens 2 m^3 Gas) kostete. Weiterhin haben wir aber schon damals gewusst, dass nach Mitternacht nur noch selten jemand duscht und auch niemand mehr zum Geschirrspülen Lust hat. Warmwasser wird eigentlich nur in bestimmten Zeiten gebraucht, bei uns vormittags und abends. Und dann muss das Wasser in diesen Zeiten nicht mit 50 Grad zur Verfügung stehen. Zum Duschen reichen 40 Grad aus, 40 Grad warmes Wasser ist schon ordentlich heiss auf der Haut und muss sogar schon durch Beimischung von Kaltwasser gekühlt werden. Und zum Zähneputzen und spülen reicht Wasser mit einer Temperatur von 30 Grad auch aus.

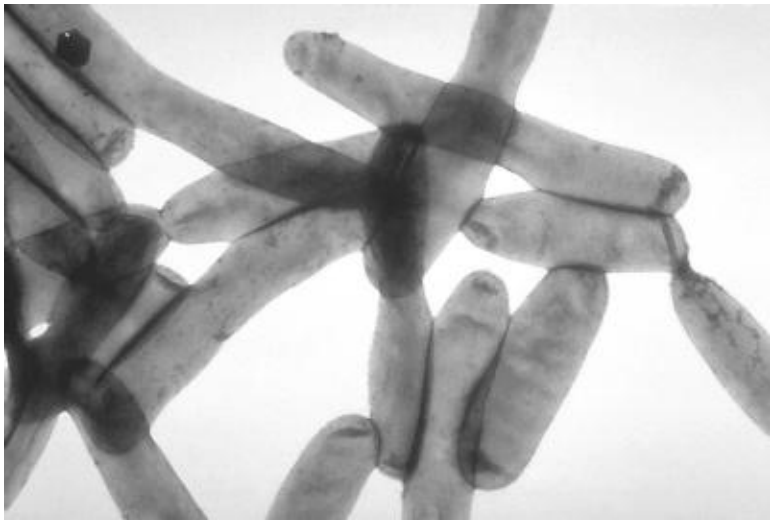

Bild 2. Legionella pneumophila beim Familientreffen

Legionellen, diese kleinen stäbchenförmigen Bakterien aus der Familie der Legionellaceae, wollte ich nicht im Wasser haben. Sie fühlen sich bei 25 bis 50 Grad Wassertemperatur am wohlsten und sterben ab 60 Grad ab. Aber deshalb rund um die Uhr das Wasser auf 60 Grad halten? 150 Liter warmes Wasser werden in einem 4-Personen Haushalt mindestens einmal am Tag komplett ausgetauscht. Bei einem 1000 Liter Boiler würde ich das einsehen, nicht aber bei 150 Litern. Und sollte sich doch mal eine Legionella pneumophila (siehe Bild 2.) oder ein Kollege zuviel in unserem Boiler tummeln, selbst im kältesten Winter hat die Solaranlage das Wasser schon öfters auf 60 Grad und mehr erwärmt.

Im Gegensatz zu meiner alten Gastherme bietet der ETA-SH eine Steuerung, die es erlaubt, für unterschiedliche Zeiten verschiedene Temperaturen vorzugeben. Aber, wie obiger Text zeigt, ist das auch noch nicht ausreichend. Es wird nicht zu festen Zeiten geduscht, es muss nicht jeden Morgen und jeden Abend Wasser mit 40 Grad zur Verfügung stehen.

Ich wünschte mir einen Schalter neben der Badezimmertür, auf den ich 10 Minuten vor dem Duschen drücken kann und der dann das Aufheizen des Wassers initiiert. Das Wasser, was vormittags und abends mit etwas über 30 Grad zur Verfügung steht und den Rest des Tages solar geheizt wird, wenn entsprechende Sonne scheint. Das Wasser soll auch nur vor dem Duschen aufgeheizt werden und nicht auch wieder nach dem Duschen.

Ähnliche Gedanken kann man sich zu anderen Gewerken auch machen, etwa der Warmwasserzirkulatiuon, die auch nicht rund um die Uhr laufen muss, sondern nur, wenn man sie braucht. Nach der Realisierung vieler kleiner solcher Schritte haben wir gesehen, wieviel Energie sich insgesamt einsparen lässt. Und das mit recht wenig komfortverzicht.

Das ETA-Protokoll

Die Firma ETA stellt ein einfaches Protokoll zur Verfügung, mit dem über eine RS232 Schnittstelle auf den Kessel zugegriffen werden kann. Ganz leicht lassen sich damit Temperaturwerte auslesen. Die Parameter für die RS232 sind in Tabelle 1. aufgezeigt.

Geschwindigkeit	19200 Baud
Startbits	1
Stoppbits	1
Datenbitsbits	8
Parität	keine

Tab 1. RS232-Parameter

Der Aufbau eines Telegramms geschieht nach dem Protokoll wie folgt: Eingeschlossen in Mengenklammern "{" und "}" wird das Telegramm verpackt, das versandt werden soll. "{" ist also das STX (Starttext) und "}" das ETX (Endtext) in der ETA-Welt. Direkt hinter dem STX steht ein Funktionscode aus zwei Zeichen (zwei Byte), dahinter folgt eine Angabe über die Länge der Nutzdaten (Nachricht) als viertes Byte. An Platz fünf steht die Prüfsumme der Nutzdaten. Hinter der Prüfsumme folgen die Nutzdaten, sofern welche geschickt werden. Direkt hinter den Nutzdaten folgt das ETX-Zeichen, also die schließende Mengenklammer "}".

Eine kürzeste Nachricht sieht wie folgt aus:

{	Funktionscode1	Funktionscode2	Länge der Nachricht	Prüfziffer	}

Tab 2. Der Aufbau des Telegramms

oder, um es einmal mit konkretem Protokollcode zu füllen:

1.	"{"	"M"	"E"	0	0	"}"
2.	0x7B	0x4D	0x45	0x00	0x00	0x7D

Tab 3. Ein Beispieltelegramm.

In obigem Beispiel (Tabelle 3.) ist die Länge der Nachricht gleich Null, das Telegramm selber ist die Nachricht. Da das Telegramm keine Nachricht trägt, ist auch der Wert der Prüfziffer der Nachricht Null. Das obige Telegramm "{ME00}" ist übrigens das Telegramm zum Abbestellen abonnierter Daten. Dringend zu beachten ist, dass die zweite Zeile der Tabelle 3. die hexadezimalen Werte trägt, die tatsächlich übertragen werden müssen.

Die Prüfziffer bildet sich durch Addition der Bytes der Nachricht und Verundung mit 0xFF, oder einfacher gesagt: nach der Addition der Nachrichtenbytes muss der errechnete Wert Modulo 256 gerechnet werden.

Die Entwickler bei ETA gehen davon aus, dass der normale Mensch an einer permanenten Versorgung mit Temperaturdaten interessiert ist, deshalb ist ein einfaches und einmaliges Anfordern eines Temperaturdatums gar nicht so einfach möglich. Es wäre nur über den Umweg eines Abonnements möglich, dass sofort nach der ersten Datenlieferung (mit obigem "{ME00}") gekündigt werden müsste. Also passen wir uns diesem Denken an und bestellen ein Abonnement: Eine Abonnementsbestellung ist ein Telegramm mit einem festen Funktionscode "MC", einer Angabe der gewünschten Daten und einer Zeitangabe (in Sekunden) die bestimmt, in welchem Intervall die Daten gesendet werden sollen. Der Aufbau ist wie folgt:

"{"	"M"	"C"	Länge	Prüfziffer	Zeitintervall	Nutzdatum	...	Nutzdatum	"}"

Tab 4. Aufbau des Telegramms zur Bestellung eines Abonnements.

Daten, die man bestellen kann, gibt es einige. Sie unterscheiden sich bei den einzelnen ETA-Geräten auch in Anzahl und zu benutzendem Referenzwert, es sind beim SH-Kessel andere als beim Hackschnitzel-Kessel oder beim Pelletbrenner. Die hier aufgezeigten stellen einen Auszug aus den möglichen des SH-Kessels dar.

Um Daten zu bestellen muss man jeweils zwei Bytes angeben, die ein Datum identifizieren. Mich haben nicht alle interessiert, die für mich wichtigsten, die ich im unten nachfolgenden Programm benutzt habe, werden in Tabelle 5. nochmal zusammengefasst.

Datum	**Byte 1**	**Byte 2**

Temperatur Puffer oben	0x00	0x0c
Temperatur Puffer mitte	0x00	0x0b
Temperatur Puffer unten	0x00	0x0a
Temperatur Boiler	0x00	0x0d
Aussentemperatur	0x00	70
Pufferladezustand (in %)	0x00	75
Abgastemperatur	0x00	15
Kesseltemperatur	0x00	8
Rücklauftemperatur	0x00	9
Temperatur Vorlauf MK1	0x00	68

Tab 5. Bestellbare Daten

Wenn man ein Datum bestellen möchte, muss man noch angeben, wo man es bestellen möchte. Der Bestellort SH-Kessel hat dabei den Wert 0x08. Soll beispielsweise die Wassertemperatur im Boiler alle zehn Sekunden vom SH-Kessel geliefert werden, dann sieht das Telegramm wie folgt aus:

STX	FC1	FC2	Länge	Prüfziffer	Intervall	Bestellort	Datum1	Datum2	ETX
"{"	"M"	"C"	04	31	0x0a	0x08	0x00	0x0d	"}"

Tab 6. Beispieltelegramm zur Wassertemperatur im Boiler

Sollte die RS232-Verbindung vom Rechner zum Kessel stehen und alles korrekt eingerichtet sein und sollte der Boiler beispielsweise 42,2 Grad haben, dann sähe das Antworttelegramm wie folgt aus:

"{"	"M"	"D"	05	188	0x08	0x00	0x0d	0x01	0xa6	"}"

Tab 7. Antworttelegramm zur Wassertemperatur im Boiler

Die Datenbytes der Nachricht (im Beispiel 0x00 und 0x0d) werden gespiegelt und dahinter der aktuell gemessene Wert angegeben, der noch berechnet werden muss. Dazu wird das erste Byte mit 256 multipliziert und zum zweiten addiert. Danach muss dieses Ergebnis noch durch 10 geteilt werden. Im Beispiel also

$$(1*256+0xa6)/10 = (256+166)/10 = 422/10 = 42,2$$

Eine Konkatenation der Abfragewerte ist möglich, sollte zum Beispiel die Boiler- und die Aussentemperatur abgefragt werden sollen, dann sieht das entsprechende Telegramm wie folgt aus:

STX	FC1	FC2	Länge	Prüfziffer	Intervall	Bestellort1	Datum1.1	Datum1.2	Bestellort2	Datum2.1	Datum2.2	ETX
"{"	"M"	"C"	07	109	0x0a	0x08	0x00	0x0d	0x08	0x00	70	"}"

Tab 8. Konkatenierte Abfrage

Auch die Antwort ist dann eine Konkatenation mehrerer Werte entsprechend obiger Tabelle 7.

Als letztes Schmankerl noch das Telegramm zur Sonderladung des Boilers, das mit dem Funktionscode "IH" beginnt und dann funktioniert, wenn im Menüpunkt MK1 des SH-Kessels der SMS-Betrieb aktiviert wurde:

STX	FC1	FC2	Länge	Prüfziffer	Datum1.1	Datum1.2	ETX
"{"	"I"	"H"	2	0x40	0x40	0x00	"}"

Tab 9. Telegramm: Boiler aufheizen

Aufbau und Benutzung des Programmes

Das unten folgende C-Programm habe ich als Server aufgebaut, der dauerhaft auf einen Port (im Moment Port 5123) lauscht, Befehle entgegennimmt und ausführt. In einem weiteren Thread läßt sich das Programm alle 20 Sekunden vom ETA-Kessel die Werte der obigen Tabelle 5. ausliefern. Mögliche Befehle sind die Anforderung der aktuellen Temperaturen, das Sonderwärmen des Boilerwassers, sowie das Runterfahren des Servers. Weitere Funktionalität wird zukünftig sicher noch hinzu kommen.

Beim Start des Programmes wird als Parameter angegeben, über welche serielle Schnittstelle der Kessel erreicht werden kann. Bei mir sieht der Start des Programmes wie folgt aus:

```
ETADat /dev/ttyS2
```

Weitere Parameter, wie zum Beispiel die Angabe eines variablen Ports, der zur Abfrage zur Verfügung stehen soll, sind geplante Kleinigkeiten.

Nach dem Start des Programmes kann auf das Programm via telnet (oder ähnlichem) über den Port 5123 zugegriffen werden, etwa über

```
telnet localhost 5123
```

Hier stellt sich dann eine Konsole zur Verfügung, die Befehle entgegennimmt. Das nachfolgende Programm, das nun Port 5123 bedient, kennt auch den Befehl

```
help.
```

Für den batch-Betrieb ist der Befehl

```
statclose
```

aufgenommen worden, der die abonnierten aktuellen Temperaturdaten ausgibt. Eine solche Ausgabe kann dann wie folgt aussehen:

```
[reineke@fox]9447# telnet localhost 5123

Hilfe durch Eingabe von help

# statclose

Puffer Oben    : 65.60
Puffer Mitte   : 37.40
Puffer Unten   : 37.40
Boiler         : 17.60
Aussen         : -3.00
Puffer Ladezst.: 33.00%
Abgas          : 34.60
Kessel         : 41.60
Ruecklauf      : 35.20
```

```
Vorlauf MK1    : 42.80
Connection closed by foreign host.
[reineke@fox]9447#
```

Anstelle des Befehls

`statclose`

kann auch der Befehl

`status`

gefolgt vom Befehl

`close`

eingegeben werden. Zur Anwärmung des Boilerwassers dienen die Befehle

`sonderwaermen`

und

`sonderwaerclose.`

Der Befehl

`shutdown`

terminiert das Programm.

Der Aufbau des Programmes als Server verdoppelt zwar den Programmcode, bietet aber einen viel größeren Komfort in der Abfrage und in der Steuerungsmöglichkeit. Die erste Version des Programmes hat sich noch für jede Anfrage wieder neu beim Kessel angemeldet, das ist zwar auch möglich, erschwert aber die Implementierung weitergehender Funktionalität, wie etwa das Sonderwärmen des Boilerwassers.

Das ganze auf dem Fox Board

Um das Programm auf dem Fox Board laufen zu lassen, habe ich es in die `/etc/inittab` als `respawn` für den Runlevel 3 eingetragen, es wird automatisch gestartet, sobald der Runlevel 3 erreicht wird. Sollte es dann abstürzen oder mit dem Befehl `shutdown` terminiert werden, wird es sofort vom `init` Prozess wieder gestartet. Ist der Socket (Port 5123) dann noch gebunden, beendet sich das Programm nach 5 Sekunden und wird von `init` sofort wieder gestartet. Spätestens nach einer Minute ist der Socket dann freigegeben und das Programm läuft wieder.

Der Eintrag in der `/etc/inittab` sieht dafür wie folgt aus:

```
eta:3:respawn:/mnt/flash/reineke/ETADat /dev/ttyS2
```

und kann unten in die Datei eingetragen werden.

Da auf dem Fox Board ein Web-Server läuft, kann aus dem Datenverzeichnis des Web-Servers ein `link` in das Arbeitsverzeichnis des Rechners auf eine Datei `temperaturen` gemacht und ein Programm `/mnt/flash/reineke`

/tempDat erstellt werden, dass den Inhalt der Datei temperaturen erzeugt. Die Datei /mnt/flash/reineke/tempDat kann wie folgt aussehen:

```
#!/bin/sh
DATEI=/tmp/$$_temperaturen
/bin/echo Datum"          : "`/bin/date "+%Y_%m_%d %H:%M"`> $DATEI
/usr/bin/nc localhost 5123 > $DATEI.$$ << EOF
statclose
EOF
/bin/cat $DATEI.$$ | /bin/grep \: >> $DATEI
/bin/rm $DATEI.$$
```

Bei mir werden zusätzlich noch die 1-wire Daten gesammelt, insgesamt könnte der Output von /mnt/flash /reineke/tempDat wie folgt aussehen:

```
Datum          : 2009_01_08 14:05
Puffer Oben    : 47.40
Puffer Mitte   : 37.20
Puffer Unten   : 37.60
Boiler         : 20.90
Aussen         : -2.20
Puffer Ladezst.: 21.00%
Abgas          : 34.00
Kessel         : 40.60
Ruecklauf      : 33.60
Vorlauf MK1    : 40.70
Aussen (1w)    : -1.75
Boiler oben    : 23.25
Boiler mitte   : 20.56
Boiler unten   : 17.45
```

Und damit das nun alles läuft, wird noch im cron der Aufruf des Programmes /mnt/flash/reineke/tempDat derart eingetragen, dass das Programm seinen Output alle fünf Minuten in die richtige Datei /mnt/flash/reineke /temperaturen schreibt:

```
*/5 * * * * /mnt/flash/reineke/tempDat > /mnt/flash/reineke/temperaturen
```

Fertig, damit steht eine Datei mit Temperaturwerten auf dem Web-Server zur Verfügung, die einem graphischen Visualisierungstool übergeben oder von einer Software via net abgeholt werden kann. Aber das kann an einer anderen Stelle beschrieben werden.

...und wie funktioniert nun der Duschschalter?

Bild 3. Meldung des SH nach Betätigung des Duschschalters

Da war doch noch was: Der Duschschalter! Das Vorhaben, Wärmeenergie nur dann abzugeben, wenn sie wirklich gebraucht wird. Also der Schalter neben dem Badezimmer, auf den man dann ein paar Minuten vor dem Duschen drücken muss, um warmes Wasser zu haben.

Ist eigentlich ganz einfach. Das Sonderwärmen des Boilerwassers ist schon beschrieben und das nachfolgende Programm hat es auch schon implementiert. Nur am Kessel sind zwei Einstellungen vorzunehmen und ein kleines weiteres Programm auf dem Fox Board ist zu schreiben.

Am Kessel muss die SMS-Steuerung aktiviert und unter dem Menüpunkt "Boiler" muss eine Temperatur eingestellt werden, auf die das Wasser erwärmt werden soll. Und das in der Zeit von 00:00 bis 00:00 Uhr (siehe Bild 4.), also einer Zeitspanne, die so kurz ist, dass der Kessel im Automatikbetrieb nie aktiv wird. Der ETA holt sich aus allen eingetragenen Werten den höchsten raus und heizt beim Sonderheizen (siehe Bild 3.)auf diesen Wert auf.

Bild 4. 40 Grad werden nur mit "Sonderladen" erreicht

Das Fox Board bekommt ein Kabel auf IOPort B22. Wird das Kabel auf Masse gelegt (geschaltet, Schalter neben der Badezimmertür), erkennt das nachfolgende shell-Script dies und initiiert das Aufheizen, sofern das Wasser nicht schon warm ist. Bewusst habe ich hier wieder ein shell-Script für die Doku eingesetzt, um die Fähigkeiten und die einfache Programmierung des Fox Boards nochmal in den Vordergrund zu stellen. Die Funktionalität des shell-Scriptes neu in ein C-Programm zu gießen, ist eine kleine Aufgabe.

Die Kontrolle über die Wassertemperatur macht das Programm (shell-Script) dadurch, dass es die obig beschriebene Datei `/mnt/flash/reineke/temperaturen` ausliest.

```
#!/bin/sh
```

```
while true ; do

  B22=`/bin/readbits -p g -b 22`

  if [ "$B22" -eq 0 ] ; then

    A=""

    while [ -z $A ] ; do
      A=`cat /mnt/flash/reineke/temperaturen|\
         grep "Boiler  "|cut -d":" -f2|cut -d. -f1`
      echo $A
      sleep 1
    done

    if [ $A -le 38 ] ; then
      echo sonderladen
      /usr/bin/nc localhost 5123  << EOF
sonderwaerclose
EOF
    else
      echo nicht laden
    fi
  fi
done
```

Natürlich kann man sich alle `echo`-Ausgaben sparen, da man sie eh nicht sieht; in den wenigsten Fällen hängt ja ein Monitor am Fox Board...

Das C-Programm, Version vom 8. Januar 2009

Das nachfolgende C-Programm, dass zum Beispiel unter dem Namen ETADat.c gespeichert werden kann, wird mit dem GNU C Compiler `gcc` einfach kompiliert. Da es allerdings die `pthread`-Library benutzt, muss zum komplieren

```
gcc ETADat.c -lpthread -o ETADat
```

eingegeben werden. Übrigens habe ich mir für das Fox Board inzwischen angewöhnt, meine C-Programme via Browser in Italien direkt beim Hersteller kompilieren zu lassen; dadurch kann ich auf verschiedensten Rechnern arbeiten, ohne auf jedem eine eigene Entwicklungsumgebung installieren zu müssen.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <termios.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdarg.h>
#include <signal.h>
#include <unistd.h>
#include "sys/ioctl.h"
```

```
#include "fcntl.h"
#include <pthread.h>

int tty_fd;
pthread_t leseTTY;
int server_socket;
volatile int client_socket=-1;

unsigned char start[3]          = {'{','M','C'};
unsigned char datas[10][3]={{0x08,00,0x0c},{0x08,00,0x0b},{0x08,00,0x0a}
                           ,{0x08,00,0x0d},{0x08,00,70},{0x08,00,75}
                           ,{0x08,00,15},{0x08,00,8},{0x08,00,9}
                           ,{0x08,00,68}};
unsigned char *namen[10]  ={"Puffer Oben    ","Puffer Mitte   ","Puffer Unten   "
                            ,"Boiler         ","Aussen         ","Puffer Ladezst."
                            ,"Abgas          ","Kessel         ","Ruecklauf      "
                            ,"Vorlauf MK1    "};
unsigned char ende[1]     = {'}'};
unsigned char stopp[6]    = {'{','M','E',0,0,'}'};
unsigned char boilerAn[8]    = {'{','I','H',2,0x40,0x40,0,'}'};
float werte[10];
const int anzahlWerte = 10;
volatile unsigned char ausgabe[500];
unsigned char ausgabeText[500];
volatile int sonderwaermflag=0;


int tty_open(char* tty_dev) {
  struct termios new_attributes;
  tty_fd = open(tty_dev,O_RDWR| O_NOCTTY | O_NONBLOCK);
  bzero(&new_attributes,sizeof(new_attributes));
  new_attributes.c_cflag &= ~PARENB;
  new_attributes.c_cflag &= ~CSTOPB;
  new_attributes.c_cflag &= ~CSIZE;
  new_attributes.c_cflag |= CS8;
  new_attributes.c_cflag |= B19200;
  int zahl = tcsetattr(tty_fd, TCSANOW, &new_attributes);
  if (zahl < 0) {
    printf ("Fehler %d\n",zahl);
  }
  return tty_fd;
}

void erzeugeUndSendeOutString() {
  unsigned char tmp[123];
  int i,pruef=0,lang = 0;
  memcpy(tmp+lang,start,3);
  lang += 6;
  for ( i = 0 ; i < 10 ; i++) {
    memcpy(tmp+lang,datas[i],3);
    lang += 3;
  }
  tmp[5] = 20;           // Zeit
  for (i=0;i<lang;i++) {
    if (i>=5) {
      pruef += tmp[i];
    }
  }
  tmp[3] = lang-5;
  tmp[4] = pruef%256;
  tmp[lang]= ende[0];
  lang++;
  write(tty_fd,tmp,lang);
}
```

```
void setzeAusgabe(char *was) {
  strcpy(ausgabeText,was);
}


void leseLeer() {
  unsigned char tmp[30];
  unsigned wort[500];
  int lang;
  int a,i;
  int anzahlBytes = -1;
  ausgabe[0]=0;
  lang=0;
  a = read (tty_fd,tmp,1);
  while ((a != -1)) {
    if (a>0) {
      if (lang == 3) {
        anzahlBytes = tmp[0];
      }
      wort[lang]=tmp[0];
      lang++;
      if (tmp[0] == '}') {
        if (lang >= anzahlBytes && anzahlBytes > 0) {
          break;
        }
      }
    }
    a = read (tty_fd,tmp,1);
  }
}

void leseundInterpretiereInString() {
  unsigned char tmp[30];
  unsigned wort[500];
  int lang;
  int a,i;
  float wert;
  int anzahlBytes = -1;
  while (1) {
    ausgabe[0]=0;
    lang=0;
    a = read (tty_fd,tmp,1);
    while ((a != -1)) {
      if (a>0) {
        if (lang == 3) {
          anzahlBytes = tmp[0];
        }
        wort[lang]=tmp[0];
        lang++;
        if (tmp[0] == '}') {
          if (lang >= anzahlBytes && anzahlBytes > 0) {
            break;
          }
        }
      }
      a = read (tty_fd,tmp,1);
    }
    for (i=0 ; i < 10 ; i++) {
      for (a=5;a<lang;a+=5) {
        if (wort[a]==datas[i][0] && wort[a+1]==datas[i][1] && wort[a+2]==datas[i][2]) {
          wert=(float)wort[a+3]*256.0+(float)wort[a+4];
          if (wert > 65000) wert-= 65536;

          if (wert < -1000 || wert > 10000) {
            wert = werte[i];
```

```
            wert=0;
          }

          if (!strstr(namen[i],"Puffer Ladezst.")) {
            wert /=10;
            sprintf(tmp," %2.2f",wert);
          }
          else {
            sprintf(tmp," %2.2f%%",wert);
          }
          werte[i] = wert;
          sprintf (ausgabe,"%s%s:%s\n",ausgabe,namen[i],tmp);
        }
      }
    }
    setzeAusgabe(ausgabe);
    if (sonderwaermflag) {
      write(tty_fd,boilerAn,8);
//      leseLeer();
      sonderwaermflag=0;
    }
  }
}

char *parse (char *was) {
  char *tmp;
  char *retVal = malloc(strlen(was));
  strcpy(retVal,was);
  tmp = strstr(retVal,"\n");
  if (tmp != NULL) {
    tmp[0]=0;
  }
  while ((tmp = strstr(retVal,"\t"))) {
    tmp[0] = (char)' ';
  }
  while ((tmp = strstr(retVal,"  "))) {
    memmove(tmp,tmp+1,strlen(tmp));
  }
  return retVal;
}

int serverStart(int port) {
  int anzahl, laenge;
  struct sockaddr_in serverinfo, clientinfo;
  char zeichen[200];
  char empfangen[1000];
  int execCode;

  if (port == -1) port = 5100;

  server_socket = socket(AF_INET, SOCK_STREAM, 0);

  serverinfo.sin_family = AF_INET;
  serverinfo.sin_addr.s_addr = htonl(INADDR_ANY);
  serverinfo.sin_port = htons(port);
  laenge = sizeof(serverinfo);

  execCode = bind(server_socket, (struct sockaddr *)&serverinfo, laenge);
  if (execCode == -1) {
    printf ("Funktion bind konnte nicht ausgefuehrt werden, das Programm wird\n");
    printf ("in 5 Sekunden beendet\n");
    sleep(5);
    return 3;
  }
  execCode = listen(server_socket, 3);
```

```
  if (execCode == -1) {
    printf ("Funktion listen konnte nicht ausgefuehrt werden, das Programm wird\n");
    printf ("in 5 Sekunden beendet\n");
    sleep(5);
    return 1;
  }
  while (1) {
    client_socket = -1;
    printf("\n Der Server wartet...");
    fflush(stdout);
    client_socket = accept(server_socket, (struct sockaddr *)&clientinfo, &laenge);
    printf("\n Verbindung mit %s\n",inet_ntoa(clientinfo.sin_addr));
    sprintf(zeichen,"\nHilfe durch Eingabe von help\n\n");
    write(client_socket,zeichen,strlen(zeichen));
    while (1) {
      write(client_socket,"# ",2);
      anzahl = read(client_socket,empfangen,sizeof(empfangen));
      empfangen[anzahl]=0;
      char *tmp=parse(empfangen);
      strcpy(empfangen,tmp);
      free(tmp);
      printf ("%s\n",empfangen);
      if (strstr(empfangen,"close")==empfangen) {
        break;
      }
      else if (strstr(empfangen,"shutdown")==empfangen) {
        write(client_socket,"bye\n",4);
        close(client_socket);
        close(server_socket);
        return 0;
      }
      else if (strstr(empfangen,"help")==empfangen) {
        sprintf (zeichen,"Hilfe:\n");
        sprintf (zeichen,"%s  close           - Verbindung beenden\n",zeichen);
        sprintf (zeichen,"%s  shutdown        - Server runterfahren\n",zeichen);
        sprintf (zeichen,"%s  sonderwaerclose - Heizt den Boiler auf und terminiert\n",zeichen);
        sprintf (zeichen,"%s  sonderwaermen   - Heizt den Boiler auf\n",zeichen);
        sprintf (zeichen,"%s  statclose       - Zeigt die aktuellen Temperaturen an ",zeichen);
        sprintf (zeichen,"%sund terminiert\n",zeichen);
        sprintf (zeichen,"%s  status          - Zeigt die aktuellen Temperaturen an\n",zeichen);
        write(client_socket,zeichen,strlen(zeichen));
      }
      else if (strstr(empfangen,"status")==empfangen) {
        sprintf (zeichen,"%s",ausgabeText);
        write(client_socket,zeichen,strlen(zeichen));
      }
      else if (strstr(empfangen,"statclose")==empfangen) {
        sprintf (zeichen,"\n%s",ausgabeText);
        write(client_socket,zeichen,strlen(zeichen));
        break;
      }
      else if (strstr(empfangen,"sonderwaermen")==empfangen) {
        sonderwaermflag=1;
      }
      else if (strstr(empfangen,"sonderwaerclose")==empfangen) {
        sonderwaermflag=1;
        break;
      }
    }
    close(client_socket);
  }
  return 0;
}
```

```
int main(int argc, char *argv[]) {
  int port = 5123;
  if (tty_open(argv[1])<0) {
    fprintf (stderr,"tty open error %s\n", strerror(errno));
    exit(EXIT_FAILURE);
  }
  erzeugeUndSendeOutString();
  pthread_create(&leseTTY,NULL,(void *)&leseundInterpretiereInString,NULL);
  serverStart(port);
  write(tty_fd,stopp,6);
  close(tty_fd);
  exit(0);
}
```

Zur Hauptseite

e-mail: *Ulrich Franzke* *<sinus@ulrich-franzke .de>*