Heizungssteuerung

Wenn ein Heizkessel keine Steuerung für Heizkreise hat oder wenn eine neue Steuerung für eine ältere Anlage Energieeinsparungen verspricht, kann auch Eigeninitiative eine Lösung sein. Wer sich die Entwicklung einer preiswerten Heizungssteuerung zutraut, wird in dem folgenden Artikel die nötigen Anregungen finden.

von Dieter Holzhäuser
(http://www.system-maker.de)

Heizungsanlagen und die entsprechenden Steuerungen sind zu vielfältig, um im Rahmen dieses Artikels ganz allgemein beschrieben zu werden. Daher wird ein konkretes Projekt vorgestellt, das seit Februar 2007 in Betrieb ist. Hard- und Software sind jedoch so modular aufgebaut, dass Anpassungen an anders gestaltete Anlagen, leicht möglich sind. Aus den Darstellungen darf nicht geschlossen werden, dass die beschriebenen Einzelheiten vollständig dem Stand der Technik oder industriellen Produkten entsprechen. Es handelt sich eben „nur“ um ein ganz bestimmtes privates Projekt.

Technologisches Schema

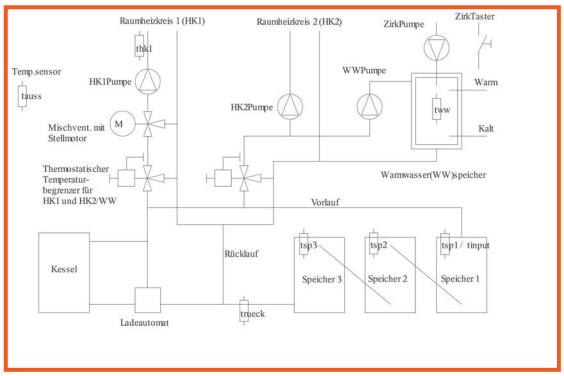

Bild 1.

Die Wärme, die der Heizkessel für Stückholz erzeugt, wird in drei hintereinander geschalteten Wasserbehältern (je 1000 l) gespeichert. Für diesen Vorgang sorgt der Ladeautomat.

An den Kesselvorlauf sind sowohl Speicher 1 als auch die 3 Heizkreise angeschlossen (der Oberbegriff Heizkreis wird hier auch für die Warmwasserbereitung verwendet).

Diese Schaltung wurde vom Kesselhersteller empfohlen. Sie ist aber problematisch, da der Ladevorgang des Speichers die Heizkreise beeinflusst. Vielleicht will der Hersteller damit nicht die Möglichkeit aufgeben, bei leeren Speichern nur mit dem Kessel zu heizen.

Alle Heizkreise haben in ihrem Vorlauf eine Umwälzpumpe. Je ein thermostatischer Temperaturbegrenzer mit Rücklaufbeimischung ist für Heizkreis 1 und gemeinsam für die beiden anderen Heizkreise vorgesehen, denn Speicher- und Kessel-Vorlauftemperaturen von 90°C und mehr sind keine Seltenheit.

Raumheizkreis 1 hat ein 3-Wege-Mischventil mit Stellmotor. Er versorgt eine Fußbodenheizung, deren Heizschleifen keine eigenen Thermostatventile besitzen. Raumheizkreis 2 ist für eine noch zu installierende Heizkörperheizung mit Thermostatventilen vorgesehen.

Heizkreis 3 lädt den Warmwasserspeicher (200 l), dessen Zirkulationskreis durch eine Pumpe betrieben wird. Zum Start der Pumpe dient ein Taster.

Das technologische Schema enthält auch die installierten Temperatursensoren, die mit den Namen bezeichnet sind, wie sie im Quellcode für die Temperaturen verwendet werden.

Aufgaben der Steuerung

Generell stellt die Steuerung fest, ob aufgrund der Außentemperatur zu heizen ist oder nicht.

Falls ja, ist für Heizkreis 1 die Vorlauftemperatur zu berechnen, mit der die gewünschte Raumtemperatur erreicht wird. Grundlage dafür ist die Heizkurve des Gebäudes. Das Mischventil stellt die berechnete Vorlauftemperatur her. Außerdem soll der Raumtemperaturwunsch zeitabhängig zu verändern sein (Nachtabsenkung, Urlaub).

Heizkreis 2 heizt abhängig von der Außentemperatur, genauso wie Heizkreis 1. Eine weitergehende Temperatursteuerung ist zurzeit nicht vorgesehen.

Heizkreis 3 stellt eine Warmwassertemperatur mit zeitabhängigem Sollwert her (Zeitspannen täglich und nach Datum). Da Wärme aus Speichern bezogen wird, ist genügend Heizleistung vorhanden, um auf einen Vorrang der Warmwasserbereitung verzichten zu können.

Die Zirkulationspumpe ist mit einem Taster in der Wohnung für eine bestimmte Zeit einzuschalten.

Der Kessel bringt seine Steuerung mit. Daher werden die Speicher als unerschöpfliche Wärmequelle angesehen, deren Versiegen eine Störung bedeutet.

Hard- und Softwarekonzept

Neben den eigentlichen Steuerungsvorgängen ergibt sich aus den Aufgaben ein ziemlich umfangreicher Anwenderdialog. Die Steuerung muss die Wünsche des Betreibers und auch die Parametereinstellungen vom Experten entgegennehmen. In umgekehrter Richtung muss sie Informationen über den Betrieb der Heizung und insbesondere Störungsmeldungen ausgeben.

Wenn sich der Autor des Artikels „Der vielarmige R8C“ in Elektor 11/2006 ein Heizungsprojekt vornimmt, dann mit dem R8C von Renesas und natürlich mit dem dort beschriebenen Multitasking (mt). Nur reichen der Speicherplatz und die I/Os eines R8C für ein solches Projekt nicht aus. Daher sind in der Heizungssteuerung ein R8C für den Anwenderdialog und ein zweiter für das operative Geschäft (für die eigentliche Steuerung) zuständig. Die beiden R8C sind über ihre seriellen Schnittstellen 0 (Pin 10/11) verbunden und bilden ein (Mini-)Netz, dessen geringe Geschwindigkeit aber ausreicht, weil nur

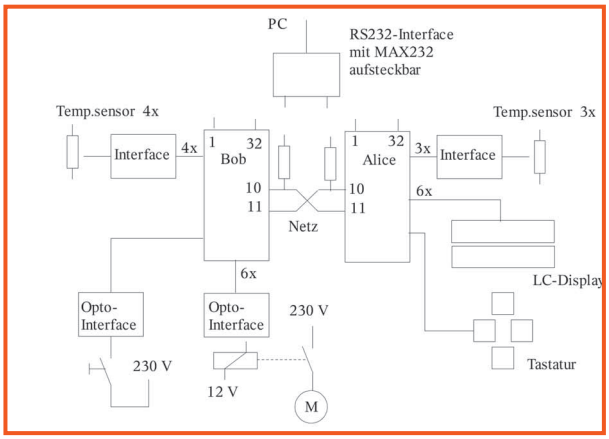

Bild 2.

Parameter und Betriebsdaten zu kommunizieren sind.

Der Einfachheit halber werden die in der Kryptografie üblichen Namen für zwei Kommunikanten übernommen, auch wenn in diesem Artikel Kommunikation ein Thema am Rande ist. Also heißt der MC für den Anwenderdialog Alice und der für die Steuerung Bob. Ganz allgemein wird ein Netzwerkteilnehmer MC oder mc genannt.

Alices Peripherie besteht aus einem LC-Display mit 2 x 16 Zeichen, einer Tastatur mit 4 Tasten, die nur einen analogen Eingang benötigt, und Interfaces zur Messung von 3 Temperaturen, die nur informativ von Bedeutung sind.

Bob muss 4 Temperaturen messen, die in die Steuerung eingehen. Für die Ansteuerung der 4 Umwälzpumpen und des Stellmotors ist ein Optokoppler-Interface mit 6 Relais vorgesehen. Der Taster zum Start der Zirkulationspumpe wirkt über ein 230-V-Input-Interface, natürlich auch mit Optokoppler, auf die Steuerung ein.

Schaltpläne der meisten Interfaces, ihre Beschreibung und ein Bild der ausgeführten Steuerung sind auf der Heft-CD zu finden.

RTOS mt im Schnelldurchgang

Nicht nur das erwähnte Multitasking (mt), sondern auch die Vernetzung und der Anwenderdialog sind unabhängig von der Steuerungsaufgabe in Modulen programmiert und können daher die Grundlage zur Lösung auch anderer Aufgaben sein. Im Gegensatz zu Bibliotheken, die Funktionsaufrufe bereitstellen, haben Module aufwendigere Programmierschnittstellen. Zusammen mit mtsys.c, dem Multitasking-Kernel bilden die Module so etwas wie ein Betriebssystem, das RTOS mt:

Die Task für die Vernetzung mehrerer R8C wird von Modul net.c bereitgestellt. Genauso wie mtsys.c ist auch net.c bei jedem Netzwerkteilnehmer einzubinden.

Die Task für den Anwenderdialog im Modul lcd.c muss nur auf dem MC laufen, an den das Dialoggerät angeschlossen ist. lcd.c greift seinerseits auf die Bibliothek convert.c zurück, die Daten in anzeigbare Texte und umgekehrt konvertiert.

Eine ausführliche Beschreibung aller Module des RTOS mt ist auf der Heft-CD zu finden. Die Einzelheiten zur Anwendung entnimmt man am besten aus den jeweiligen Headerdateien.

Mit den Funktionen der Bibliothek flash.c können Daten dauerhaft gespeichert und wieder ausgelesen werden. Diese Funktionen kümmern sich auch um die Verwaltung des Flash-Datenspeichers.

Da die Module von RTOS mt nach Bedarf eingesetzt werden, wächst die Programmgröße nicht mit der Anzahl der Module, die es insgesamt gibt. Also sind der Erweiterung von RTOS mt kaum Grenzen gesetzt.

Der folgende Schnelldurchgang hilft, die Strukturierung von Bobs und Alices Software besser zu verstehen.

mt-Tasks

mt-Tasks bestehen aus mehreren (Task-)Funktionen, die von einem endlos laufenden Scheduler aufgerufen werden und die auch dorthin zurückkehren, wenn sie nicht weiterlaufen können, weil sie auf ein mt-Ereignis warten müssen. mt-Ereignisse sind Zeitabläufe, steigende und fallende Bitflanken sowie Semaphoren (vergleichbar mit Eisenbahnsignalen, die von Tasks (Stellwerk) auf freie Fahrt gesetzt werden und von anderen Tasks (Zug) passiert werden).

Das Warten wird beim Scheduler durch Aufruf einer mt-Funktion angemeldet, bevor sich die Taskfunktion beendet. Als Argument, wird auch die Taskfunktion (natürlich nur eine aus der eigenen Task) angegeben, die der Scheduler bei Eintritt des Ereignisses aufrufen soll. Damit ist der Kreis geschlossen. Tasks laufen daher genauso endlos wie der Scheduler. Nur sind sie durch Wartezeiten unterbrochen, in denen der Scheduler andere Tasks laufen lässt.

Es gibt Sonderformen: Tasks mit nur einer Taskfunktion, Taskfunktionen, die keine mt-Funktion aufrufen (endliche Tasks) oder die zum Scheduler zurückkehren, ohne auf etwas zu warten. Schema einer mt-Task, bestehend aus den beiden Taskfunktionen demo1 () und demo2 () :

```
void demo1 ();
void demo2 () {
.   //Teilaufgabe von Task DEMO
.
mtwait ( DEMOSEMA , demo1);
//Taskfunktion demo2 möchte auf
  Semaphore
//DEMOSEMA warten und dann mit
  demo1 fortfahren
}
void demo1 () {
.   //Teilaufgabe von Task DEMO
.
mtdelay ( x , demo2);
    //Taskfunktion demo1 möchte
      x ms warten
    //und dann soll demo2
      aufgerufen werden
}
```

Im Hauptprogramm müssen bestimmte Definitionen, Deklarationen, Initialisierungen und Aufrufe gemacht werden. Die Einzelheiten sind in den Kommentaren von mtsys.h zu finden.

Nur wenn der Scheduler nicht immer laufbereite Tasks vorfindet und auch Zeit in seiner Leerlaufschleife verbringen kann, hat der MC die Leistungsreserven, um echtzeitfähig zu arbeiten.

mt-Netz

Das mt-Netz stellt für Tasks auf mehreren MC die Intertaskkommunikation bereit, sonst nichts. Tasks kommunizieren ganz einfach über globale Variablen und Semaphoren. Wenn die Tasks so auf die MC aufgeteilt werden, dass nur zeitunkritisch kommuniziert wird, ist die „langsame“ serielle Ringverbindung der am Netz beteiligten MC kein Problem. Dem Vorteil, dass es keine Konkurrenz gleichzeitig sendender Teilnehmer und keine aufwendig zu verwaltenden Datenpuffer gibt, steht der Nachteil gegenüber, dass empfangene Daten, die für einen anderen MC bestimmt sind, weitergeleitet werden müssen.

Was für die Tasks auf ein und demselben MC eine globale Variable ist, das ist für Tasks, die auf mehrere MC verteilt sind, die netzglobale Variable. Das ist eine zusammenfassende Bezeichnung für eine konkrete Ganzzahl-Definition (max. 4 Byte) und ein Element in der Netzliste an gleicher Position bei jedem MC, der an der netzglobalen Variablen ein Interesse hat.

Die konstante Netzliste ist der Kern der Netz-Programmierschnittstelle. Sie enthält konstante Daten für jede zu kommunizierende Variable und Semaphore.

Mit der Funktion net (..PUT..) verbreitet ein MC den produzierten Wert einer netzglobalen Variablen. Auch kann dieser MC, wenn nötig, von einem anderen mit net (..GET..) dazu aufgefordert werden, den gerade aktuellen Wert zu senden.

Ganz ähnlich kann ein MC in einem Element der Netzliste Semaphoren zum Setzen und Zurücknehmen anbieten, wozu andere MC net (..SET..) und net (..CLEAR..) benutzen. (Diese Fähigkeit des Netzwerks kommt bei der Heizungssteuerung allerdings nicht vor.)

Beispiel einer konstanten Netzliste von Alice

```
#define BOB   2
#define ALICE 3     //selbst
#define EVE   4
long a;     //lok. Definition
              einer netzglobalen
              Variablen
const unsigned char mc = ALICE
            //eigene MC-Nummer
const netlist1type netlist1[]= {
      (int) & a, NLONG,EVE,
            //netzglobale
              Variable (ggf. bei
              EVE anzufordern)
      TESTSEM,   NSEM,    ALICE,
            //eigene Semaphore
              zur Beeinflussung
              anbieten
      DUMMYSEM,  NSEM,    BOB,
            //Semaphore
              beeinflussen, die
              BOB anbietet
      }
```

Neben der Netzliste gehören bestimmte Deklarationen, Definitio-

nen und Aufrufe zur Programmierschnittstelle.

Jeder MC kann das Netz zyklisch prüfen lassen, indem er eine Funktion bereitstellt, die bei kommender und gehender Netzstörung aufgerufen wird, um die nötigen Dinge zu tun. Eine unvermeidliche Netzstörung ist der Reset eines oder aller Netzwerkteilnehmer.

Anwenderdialog

Nur die Task LCD aus dem Modul lcd.c kommuniziert mit dem Anwender und nur mit Daten, deren nähere Angaben in der konstanten Anzeigeliste eingetragen sind. Die aufwendige Verwaltung von möglichen Zugriffen aller Tasks auf die begrenzte Ressource „Dialoggerät“ entfällt. Als Dialoggerät dient ein LC-Display mit zwei Zeilen inklusive einer Tastatur mit 4 Richtungstasten, die an einem Analogeingang angeschlossen sind. Die Kommunikationsmöglichkeiten sind auf die Anzeigeliste, das Dialoggerät sowie auf Steuerungen zugeschnitten und folgen den gleichen Regeln.

Das LCD zeigt zur selben Zeit nur eine Date (hier Singular von Daten) an, und zwar in der ersten Zeile einen konstanten Text zur Erklärung und in der zweiten den Zahlenwert oder den Text der Date. Der Zahlenwert ist ein Ganzzahltyp, der in einen Text für Datum, Uhrzeit, „Pseudo“-Dezimalzahl oder andere Formen konvertiert ist. Die Anzeige der Date wird im Zyklus von etwa 0,5 s aktualisiert.

Der Zahlenwert in seiner konvertierten Form kann editierbar sein. Texte sind nicht editierbar, sie bieten aber die Möglichkeit, eine Aktion auszulösen.

Als Ergänzung zur Anzeigeliste kann eine Prüffunktion definiert werden, die nach jedem Editier- oder Aktionsvorgang aufgerufen wird und beispielsweise die Eingabe prüft oder die Aktion ausführt.

Außerhalb des Editiermodus kann mit der Tastatur durch die Daten in der Anzeigeliste navigiert werden. Die Regeln für das Editieren und Navigieren sind in der Headerdatei lcd.h zu finden.

Für das Modul lcd.c sind die konstante Anzeigeliste und die Prüffunktion die Hauptbestandteile der Programmierschnittstelle.

Beispiel einer konstanten Anzeigeliste mit Prüffunktion

```
long t, z, ta;
const  lcdlisttype  lcdlist[] = {
   “Uhr”,      (int) & t,      TIME,
        0,DISP, // t als Uhrzeit anzeigen
   “Zahl”,     (int) & z,      LONG,
        0,EDIT, // z als long-Typ
                    editieren
   “Speichern”, (int) “taussen”,
   STRING, 0,EDIT, // Aktion in der
                        Prüffunktion
                        auslösen
};

void ftest (char a, long l) {
   //Prüffunktion
   switch (a) {
        case 1: if (l < 1000) z = l ; break;
        case 2: ta = taussen; break;
   }
}
const void ( * testfunc) (char, long) =
   ftest;
```

Flash-Datenspeicher

Zwei Blöcke zu je 2 KByte bilden den Flash-Datenspeicher des R8C, in den neue Daten nur aufeinander folgend geschrieben werden. Vorhandene Daten können nicht überschrieben werden, weil sie nur Blockweise löschbar sind. Deswegen werden die beiden Blöcke von Modul flash.c als Ringpuffer verwaltet, was auch das Löschen des vollen Puffers mit den ältesten Daten umfasst, die dann ohne Warnung verloren gehen.

Man kann mit der Funktion fwrite () beliebige Strukturen speichern, die durch eine Kennzahl zu unterscheiden sind. Die Funktion fread () liest eine Struktur mit einer bestimmten Kennzahl, und zwar den ältesten, den jüngsten oder den nächsten Eintrag, ausgehend vom letzten Zugriff. Bei einem gespeicherten Parameter, der z.B. vom Anwender eingegeben wurde, ist es nur sinnvoll den jüngsten Eintrag auszulesen, bei Log-Daten können alle Einträge von Bedeutung sein.

Bobs Software

Wie bei vielen Steuerungen nimmt die Lösung für den Kern des Problems relativ wenig Raum ein gegenüber den Randaufgaben, zu denen insbesondere die Versorgung des Problemkerns mit Daten und Parametern gehört. Bei der Heizungssteuerung erledigt das zum großen Teil Alice, die daher keine Berührung mit operativen Heizungsaufgaben hat. Deshalb ist die Beschreibung von Alices Software auf der Heft-CD zu finden. Aber auch Bob ist mit der Gewinnung von Daten befasst, und zwar misst er zyklisch die Temperaturen, die in der Steuerung verwendet werden müssen.

Der Kern von Bobs Software besteht aus den Heizkreistasks HK1 (ergänzt durch HK1CTRL), HK2 und WW (ergänzt durch ZK). Die Beschreibung von Bobs Software ist eine Beschreibung des Quelltextes heizbob.c auf der Heft-CD, der bei der Lektüre dieses Abschnitts einbezogen werden sollte. Aus Platzgründen werden hier nur die wesentlichen Teile dargestellt.

Heizkreise können sich in verschiedenen Zuständen befinden, mit denen dann unterschiedliche Verhaltensweisen verbunden sind. Zum Beispiel muss sich ein Raumheizkreis bei sommerlichen Temperaturen anders verhalten (anders gesteuert werden) als im Winter oder bei Ausschaltung.

Daher sind Heizkreiszustände in Form von Zahlen definiert, von denen der Einfachheit halber alle Heizkreise Gebrauch machen. (Es spielt keine Rolle, wenn die Verhaltensweise eines Heizkreises die Bezeichnung des Zustandes nicht genau trifft.)

```
//Zustände von Heizkreisen
#define AUS   0
#define PAS   1
#define WRT   2  //Warten
#define AKT   3
#define ERR   4
```

Daneben gibt es die Bezeichnung der Zustände in Textform, die an Alice übertragen und im LC-Display angezeigt wird.

```
typedef struct { char  st
    [4] ;}  styp ;
const  styp  stattxt [] = {
    "AUS",  "PAS",  "WRT",
    "AKT",  "ERR" } ;
```

Raumheizkreise

Die Task HK2, also die für die Heizkörperheizung, ist eine „Light-Version“ von Task HK1, die die Fußbodenheizung steuert. Deshalb muss hier nicht weiter auf HK2 eingegangen werden. Das Grundkonzept der Tasks für die Raumheizung ist auch bei Task WW für die Warmwasserbereitung zu finden.

Task HK1 besteht aus nur einer Taskfunktion hk11, die im Zyklus von 1 s läuft, da sie mit dem Aufruf von mtdelay (1000,hk11); endet.

Zunächst prüft die Taskfunktion, ob ein Fehler vorliegt, der die Arbeit der Task unmöglich macht. Falls ja, wird der Zustand ERR gesetzt, unabhängig vom aktuellen Zustand. Der Heizkreiszustand wird in hk1status gespeichert.

Nach dieser Prüfung wird mit einer switch-Anweisung zu dem Code gesprungen, mit dem der bestehende Zustand behandelt wird. Es wird geprüft, ob die Voraussetzungen vorliegen, diesen Zustand zu verlassen. Falls ja, wird der neue Zustand gesetzt, falls nein, werden die zum aktuellen Zustand gehörenden Aktionen ausgeführt. Auch Aktionen von der Art „ Pumpe ein“, werden bei jedem Zyklus wiederholt, und zwar aus Sicherheitsgründen.

Unter case ERR: wird geprüft, ob die bestehenden Fehler verschwunden sind, denn nur dann wird in einen anderen Zustand gewechselt. An dieser Stelle hat Zustand AUS Priorität. Daher wird hk1onoff abgefragt, ein Anwenderparameter, der das Ein/Ausschalten des Heizkreises speichert. Auch bei den meisten anderen Zuständen beginnt die Prüfung des Zustandswechsels mit der Abfrage der Ein/Ausschaltung.

Da Heizkreis 1 die Fußbodenheizung versorgt, ist in seine Steuerung die Regelung der Vorlauf-Isttemperatur eingebettet. Geregelt wird aber nur im Zustand AKT. Im Normalbetrieb wechseln sich die Zustände AKT und PAS ab, was „heizen“ und „nicht heizen“ bedeutet.

Erst nach einer Vorbereitungszeit kann in den Zustand AKT gewechselt werden. Der Zwischenzustand WRT lässt diese Zeit ablaufen, indem er den Zeitzähler hk1time dekrementiert. Gesetzt wird der Zeitzähler mit dem Anwenderparameter wrttime, der mit 30 (s) voreingestellt ist. In der Vorbereitungszeit läuft die Umwälzpumpe, wodurch die betroffenen Temperatursensoren in den stationären Zustand gelangen.

Die Entscheidung, ob zu heizen ist oder nicht, wird von Task HZAKTIV getroffen, die das Ergebnis mit hzaktiv bereitstellt. Die Abfrage von hzaktiv kommt in Task HK1 bei

Fehlerverwaltung

//Fehlerspeicher
char error[9] = {,-‘,‘-‘,‘-‘,‘1‘,‘1‘,‘1‘,‘1‘,‘-‘,0} ;

Jedem Fehler ist ein Element im Vektor error[] zugeordnet, das den Fehler mit Textzeichen ‚1‘ oder Varianten des Fehlers mit ‚2‘ usw. speichert. Das Zeichen ‚-‘ zeigt an, dass der Fehler nicht vorhanden ist. Zum einfachen Zugriff auf den Fehlerspeicher sind für die Indices Fehlernamen definiert.

Mit Zugriff durch & error[0] und & error[4] wird der Fehlerspeicher zur Anzeige im LC-Display an Alice übertragen.

Zur Fehlerdarstellung gehört auch die Fehlersammelmeldung in Form einer erhöhten Blinkfrequenz der Lebenszeichen-LED, die von Task ALIVE betrieben wird.

Heizen oder nicht heizen?

Auf den ersten Blick scheint eine Task, die diese Frage behandelt, unnötiger Aufwand zu sein, wo man doch einfach die Außentemperatur entscheiden lassen könnte. Es ist aber nicht nötig, in einer kühlen Nacht auf Heizbetrieb zu gehen, wenn ein warmer Tag zu erwarten ist.

Da keine einfache Möglichkeit besteht, die Wettervorhersage in die Heizungssteuerung einzubeziehen, nimmt Task HZAKTIV die Vergangenheit her. Insbesondere bei Gebäuden mit massiven Wänden ist soviel von der Wärme eines freundlichen Tages gespeichert, dass man auch aus diesem Grund auf Heizbetrieb in der kühlen Nacht verzichten kann.

Daher wird mit hzaktiv gleich ON (heizen) restriktiver umgegangen als mit OFF (nicht heizen):

hzaktiv wird im Zustand OFF auf ON gesetzt, wenn die Außentemperatur kleiner als der Parameter (talim-0.5°) ist und auch die mittlere Außentemperatur der letzten 24 Stunden kleiner als der Parameter ta24lim ist.

Im Zustand ON wird hzaktiv auf OFF gesetzt, wenn die Außentemperatur (talim+0.5°) überschreitet.

Die Hysterese der Umschaltung beträgt mindestes 1°.

Und nur wegen der etwas aufwändigen Berechnung der mittleren Außentemperatur gibt es Task HZAKTIV.

Da für das Problem „heizen oder nicht heizen“ keine perfekte Lösung zu existieren scheint, bietet Task HZAKTIV ein weites Experimentierfeld für Energiesparer.

```
//Task HK1
int hk1time = 0;

void hk11 () {
  long t;
  //Störung abfragen
  if ( (error[THK1]   != ,-') ||
       (error[TAUSS]  != ,-') ||
       (error[TINPUT] != ,-' ) ||
       (error[HAND]   != ,-' ) )
                hk1status = ERR  ;
  switch (hk1status) {
    case AUS:  if (hk1onoff == ON)  {
                 //eingeschaltet
                 if (hzaktiv == OFF)
                   //nicht heizen
                   hk1status = PAS;
                 else {
                   //Heizen vorbereiten
                   hk1time = wrttime;
                   hk1status = WRT;
                 }
               }
               else HK1PUAUS
               break;

    case WRT:  hk1time--;
               if (hk1time == 0) {
                 //Vorbereitung beendet
                 hk1time  = ctrlcyc;
                 hk1status = AKT;
               }
               else HK1PUEIN
               break;

    case PAS:  if (hk1onoff == OFF)
                 //ausgeschaltet
                 hk1status = AUS;
               else {
                 if (hzaktiv == ON)  {
                   //Heizen vorbereiten
                   hk1time = wrttime;
                   hk1status = WRT;
                 }
                 else HK1PUAUS
               }
               break;

    case AKT:  if (hk1onoff == OFF) {
                 //ausgeschaltet
                 errclear ();
                 hk1status = AUS;
               }
               else if (hzaktiv == OFF ) {
                      //nicht Heizen
                      errclear ();
                      hk1status = PAS;
                    }
                    else  {
                      hk1time--;
                      if (hk1time == 0) {
                        //regeln
                        hk1time = ctrlcyc;
                        signal (HK1SEM);
                      }
                      HK1PUEIN
                    }
                    break;

    default:  hk1status = ERR;
    //default sollte nicht vorkommen

    case ERR: if ( (error[THK1]   == ,-' ) &&
                   (error[TAUSS]  == '-' ) &&
                   (error[TINPUT] == '-' ) &&
                   (error[HAND]   == '-' ) ){
                //Störung gegangen
                if (hk1onoff == OFF)
                  hk1status = AUS;
                else {
                  if (hzaktiv == OFF) {
                    hk1status = PAS;
                  }
                  else {
                    hk1time = wrttime;
                    hk1status = WRT;
                  }
                }
              }
              else  HK1PUAUS
              break;
  }
  hk1stat = stattxt[hk1status];
  net (11, PUT);
  // für Task HK1CTRL weg höh. Auffrischfrequ
  if ( error[HK1DOWN] != ,-') ts = NOKCYC;
  mtdelay (1000, hk11);
}
```

allen Zuständen, außer WRT und AKT vor.

Zustand AKT führt den Regelvorgang in einem bestimmten Zyklus aus. Dafür wird der Zeitzähler hk1time dekrementiert, der bei Verlassen des Zustands WRT mit dem Anwenderparameter ctrlcyc gesetzt wird (voreingestellt 15 s). Dass der Regelvorgang auszuführen ist, signalisiert die Semaphore HK1SEM, auf die Task HK1CTRL wartet. Die Regelaufgabe ist so verschieden von dem, was Task HK1 zu tun hat, dass eine besondere Regel-Task gerechtfertigt ist.

Task HZ1CTRL

Die Funktion hz1ctrl2 der Task HZ1CTRL startet den Stellmotor des Mischventils mit der Laufzeit und der Laufrichtung, die sich aus der Regelabweichung ergeben, und zwar nachdem die Vorlauf-Solltemperatur errechnet und ein möglicher Fehler durch ein zu niedriges Temperaturangebot behandelt wurde.

Die Formel für die Vorlauf-Solltemperatur, die von der Außentemperatur abhängig ist, ist die Geradengleichung der Heizkurve. Sie ist durch einen oberen Heizkurvenpunkt mit den Koordinaten tvo und tao und einen unteren mit tvu und tau festgelegt.

Mit tad ungleich 0 wird die Heizkurve parallel verschoben. Diesen Wert errechnet Alice aus dem zeitabhängigen Raumtemperaturwunsch des Anwenders, z.B. Nachtabsenkung.

taud ungleich 0 ändert die Steigung der Heizkurve. Mit diesem Wert kann der Anwender, natürlich mit Hilfe von Alice, die Heizkurve korrigieren, wenn die Raumtemperatur bei niedrigen Außentemperaturen unpassend ist.

Zu beachten ist, dass nur mit Ganzzahlen gerechnet wird. Dezimalzahlen, z.B. für Temperaturen werden mit dem 1000fachen Wert als Ganzzahl gespeichert und dennoch als Dezimalzahl am LC-Display

angezeigt (in RTOS mt PFLOAT-Typ genannt). Nur wenn in der Formel die Multiplikation vor der Division ausgeführt wird, führt der Divisionsrest zu vernachlässigbaren Ungenauigkeiten.

Zum Thema Heizkurve gibt es eine besondere Beschreibung auf der Heft-CD, in der auch die Formeln abgeleitet werden, die Alice verwendet. Insbesondere ist dort die etwas geheimnisvolle Beziehung zwischen der (nicht gemessenen) Raumtemperatur und der Heizkurve erklärt.

Da die Temperatur tinput, die der Wärmespeicher liefert, im Laufe der Zeit absinkt, kann es vorkommen, dass sie nicht mehr ausreicht, den Heizkreises zu versorgen. Das kann auch für die manuell einstellbare thermostatische Begrenzung der Temperatur zutreffen. Beides führt zum Fehler des zu geringen Temperaturangebots. Die Einstellung des Temperaturbegrenzers muss vom Anwender als Ersatz für eine automatische Erfassung in den Parameter thk1max eingetragen werden, der mit 55°C voreingestellt ist.

Wie bei den meisten Fehlerbehandlungen muss geprüft werden, ob der Fehler verschwunden ist, wenn er vorliegt, oder ob er gekommen ist, wenn er nicht vorgelegen hat. Zum Speichern wird auch hier ein Element des Vektors error[] (s.o.) benutzt. Die Prüfung sieht eine Hysterese von 1°C vor.

Wenn die Task HK1 aus irgendeinem Grund den Zustand AKT verlässt und der Fehler vorliegt, wird er gegenstandslos und die Meldung wird mit der Funktion errclear () zurückgesetzt. Das zu geringe Temperaturangebot ist daher „nur“ ein Fehler innerhalb des Zustands AKT, der erhalten bleibt, um auf die Beseitigung zu warten, z.B. durch Aufheizen der Speicher.

Wenn kein Fehler vorliegt, setzt die Regelung ein. Zunächst wird der absolute Wert der Regelabweichung (thk1 – tvsoll) als Laufzeit des Stellmotors in ms übernommen, aber nur bei Werten größer als 0,3°C. Die Laufzeit ist auf 5000 ms begrenzt. Mit dem Parameter lzfaktor (PFLOAT) kann der Anwender das Proportionalverhalten beeinflussen.

Das Vorzeichen der Regelabweichung bestimmt, ob der Stellmotor das Ventil auf- oder zufährt. Nach der Laufzeit, die mit mtdelay () hergestellt wird, schaltet Taskfunktion hk1ctrl den Stellmotor aus und erwartet den nächsten Regelvorgang.

Auch mit der Regelmethode sollte man experimentieren, um das bestmögliche Ergebnis bei der im Einzelfall vorhandenen Regelstrecke zu erzielen.

```
//Task HK1CTRL
void errclear () {
  //falls Fehler durch zu niedr. Temp.angebot
  //besteht, Fehlermeldung rücksetzen
  if ( error[HK1DOWN] == ,1') {
    error[HK1DOWN] = ,-';
    net (0, PUT);
    ts = OKCYC;
  }
}

void hk1ctrl1 ();
void hk1ctrl2 () {
  long lz = 0;
  long t;
  //Vorlauf-Solltemperatur errechnen
  tvsoll = (tvo  - tvu) *
           (tauss - tad - (tau + taud) ) /
           (tao - (tau + taud) ) +  tvu;
  net (9, PUT);
  //Temperaturangebot feststellen
  if (tinput < thk1max)
            t = tinput; else t =  thk1max;
  //Fehler durch zu niedr. Temp.angebot bearb.
  if ( error[HK1DOWN] != '-') {
    if (t > (tvsoll + 500)  ) {
      errclear ();        //Fehler gegangen
    }
    else  {
      mtcoop (hk1ctrl1 ); //Fehler noch vorh.
      return;
    }
  }
  else if (t < (tvsoll - 500)  )  {
        error[HK1DOWN] = '1';  //Fehler gekom
        net (0, PUT);
        mtcoop (hk1ctrl1 );
        return;
      }
  //Kein Fehler, regeln
  lz = thk1 - tvsoll;
  if (lz < 0) lz = tvsoll - thk1;
  if (lz < 300 ) {
    mtcoop (hk1ctrl1 );
    return;
  }
  //Regelabweichung zu Motorlaufzeit
  //und Laufrichtung verarbeiten
  if (lz > 5000) lz = 5000;
  lz = lz * lzfaktor;
  lz = lz / 1000;
  if ( (thk1 - tvsoll) < 0 )
              VENTILAUF  else VENTILZU
  mtdelay (lz, hk1ctrl1);
}

void hk1ctrl1 () {
  VENTILAUS
  mtwait ( HK1SEM, hk1ctrl2);
}
```

Warmwasserbereitung

Die Zustände AUS und ERR werden von der Taskfunktion ww1 von Task WW genauso behandelt wie bei den Tasks HK1 und HK2.

Im Zustand PAS der Warmwasserbereitung findet zyklisch die Prüfung statt, ob die Isttemperatur tww die Solltemperatur twwsoll, abzüglich der halben Hysterese unterschreitet. Falls ja, ist Aufheizen angesagt und es muss in den Zustand AKT gewechselt werden.

Zustand AKT benötigt den Zeitzähler wwtime, mit dem ein Timeout realisiert ist, d.h. der Aufheizvorgang sollte, bevor die Zeit time2 (voreingestellt 2400s) abgelaufen ist, beendet sein. Das ist der Fall, wenn die Isttemperatur tww größer als die Solltemperatur, zuzüglich der halben Hysterese ist. Es wird wieder in den Zustand PAS gewechselt und der normale Ablauf beginnt von neuem.

Nicht nur der Timeout ist ein Fehler innerhalb von Zustand AKT. Wenn der Zeitzähler die Zeit time1 (voreingestellt 300s) erreicht hat, findet eine einmalige Prüfung des Temperaturangebots statt. (Das mögliche zu niedrige Temperaturangebot ist bei den Raumheizkreisen ausführlicher beschrieben.) Bei der Warmwasserbereitung muss das Angebot t mindestens um die Temperatur td höher sein als die Warmwassertemperatur, die erreicht werden soll, denn nur

```
// Task WW
int wwtime = 0;
void wwclear () {
   error[WWDOWN] = ,-';
   net (0, PUT);
   ts = OKCYC;
}

void ww1 () {
char c; long t;
  if ((error[TWW]    != ,-')  ||
    (error[TINPUT]   != ,-')  ||
    (error[HAND]     != '-') )
                  //Störung vorhanden
                  wwstatus = ERR;
  switch (wwstatus) {
    case AUS: if (wwonoff == ON)
                wwstatus = PAS;
              else WWPUAUS
              break;

    case PAS: if (wwonoff == OFF)
                wwstatus = AUS;
              else if (tww <
                      (twwsoll-(twwhyst/2))){
                     wwstatus = AKT;
                     wwtime = 0;
                   }
                   else WWPUAUS
              break;

    case AKT: if (wwonoff == OFF)
                wwstatus = AUS;
              else
if (tww >= (twwsoll+(twwhyst/2)))
  wwstatus = PAS;
else {
  wwtime++;
  if (wwtime == time1) {
    if (tinput<twwmax) t = tinput;
    else t = twwmax;
    if (t < (twwsoll + (twwhyst / 2) + td )){
      //Fehler Temp.angebot zu niedrig
      error[WWDOWN] = '2';
      goto l1;
    }
    else WWPUEIN
  }
  else if ( wwtime > time2) {
        // timeout
        error[WWDOWN] = ,1';
     l1: net (0, PUT);
        wwstatus = WRT;
        wwtime = 0;
       }
       else WWPUEIN
}
              break;

    case WRT: //Warten nach Temp. Angeb. Fehler
              if (wwonoff == O){
                wwstatus = AUS;
                wwclear ();
              }
              else {
                wwtime++;
                if (wwtime >= time3) {
                  wwstatus = PAS;
                  wwclear ();
                }
              }
              WWPUAUS
              break;

    default : wwstatus = ERR;
    case ERR: wwclear ();
              if ((error[TWW] == ,-') &&
                  (error[TINPUT] == ,-') &&
                  (error[HAND]   == ,-') )
                              wwstatus = PAS;
              else WWPUAUS
  }    //switch end

  wwstat = stattxt[wwstatus];
  net ( 8, PUT);
  //Auffrischung
  if ( error[WWDOWN] != '-') ts = NOKCYC;
  mtdelay (1000, ww1);
}
```

dann ist ein ausreichend schneller Wärmeübergang an das Warmwasser möglich.

Anders als bei Task HK1, wird hier bei einem Fehler, der im Zustand AKT festgestellt wird, in den Zustand WRT gewechselt. Dort verbringt die Task WW die Wartezeit time3 (voreingestellt 1800s). Mit einer Aus/Einschaltung kann man diese Wartephase auch vorzeitig beenden. Läuft die Wartezeit ab, geht es mit gelöschter Fehlermeldung (wwclear ()) in den Zustand PAS und sehr wahrscheinlich auch in den Zustand AKT, in dem wohl wieder ein Fehler festgestellt wird usw. Das wiederholt sich bis kein Fehler mehr auftritt.

Wozu das Ganze? Wenn das Temperaturangebot nicht ausreicht, soll die Warmwasserbereitung nur in größeren Zeitabständen (time3) und nur für wenige Minuten (time1) testen, ob Wärme zur Verfügung steht. Der Timeout kann z.B. wegen eines Defektes auftreten. Auch dann wird nur nach Ablauf der Zeit time3 ein neuer Aufheizversuch vorgenommen. Sollte der Timeout entstehen, weil während der Aufheizung eine größere Wassermenge entnommen wurde, ist die Wartezeit eher nachteilig, wird aber in Kauf genommen. Der Gedanke dahinter: Die Warmwasserbereitung soll, soweit möglich, selbst bemerken, dass Unregelmäßigkeiten verschwunden sind, also nicht auf eine Anwenderquittung angewiesen sein, und sie soll die Fehlerphase energiesparend überstehen (kurze Pumpenlaufzeit).

Task ZK

Sie steuert die Zirkulationspumpe und gehört im erweiterten Sinn auch zur Warmwasserbereitung.

Mit einer gesteuerten Zirkulation lässt sich viel Energie sparen. Eine Möglichkeit ist, die Zirkulationspumpe nur zu bestimmten Tageszeiten einzuschalten, eine andere besteht aus einem Taster in der Wohnung, der bei Bedarf gedrückt wird, und die Pumpe für eine kurze Zeit laufen lässt.

Task ZK macht von der zweiten Möglichkeit Gebrauch und erwartet den Tastendruck mit mtbitdown (..). Sie ist sehr einfach aufgebaut und muss hier nicht näher dargestellt werden.

Temperaturmessung

```
//tauss   (12 k)
#define T01  34000   //Kalibriertemperaturen
#define T02  19500
#define T03  0
#define T04  -18000
#define M01  386     //Messwerte bei T1 bis T4
#define M02  515
#define M03  716
#define M04  855
const long c0 =  (T02 - T01) / (M02 - M01);
const long d0 =  (T03 - T02) / (M03 - M02);
const long e0 =  (T04 - T03) / (M04 - M03);
long tauss;            //Aussentemperatur

void temp2 () {   //Aussentemp. tauss   (8)
  int mw ;
  mw = adc (8);
  if (iserror(mw, TAUSS) == 0) {
    if (error[CAL] != ,-')
      tauss = (long) mw * 1000;
    else {
      if (mw < M02)
        tauss = (mw - M01)* c0 + T01;
      else if (mw < M03)
            tauss = (mw - M02) * d0 + T02;
          else tauss = (mw - M03) * e0 + T03;
    }
    net (2, PUT);
  }
  mtcoop (temp3);
}
```

Mit NTC's (temperaturabhängiger Widerstand mit negativem Koeffizient) von 10 k bei 25°C werden von Bob über ein einfaches Interface (siehe Artikel Hardware auf der Heft-CD) die steuerungstechnisch verwendeten Temperaturen gemessen:

Die Funktion adc (), die aus convert. c stammt, veranlasst die jeweilige AD-Wandlung, die so schnell ist, dass darauf gewartet werden kann, und gibt den ADC-Wert im Bereich von 0 bis 1023 zurück. Da bei NTC's der Zusammenhang zwischen Widerstand und Temperatur nichtlinear ist, wird die Temperatur, die dem ADC-Wert entspricht über eine Linearisierungsfunktion ermittelt, die durch 4 Punkte festgelegt ist. Die Koordinaten dieser Punkte werden durch Kalibirierung bestimmt und sollten nur den interessierenden Messbereich umfassen. Dadurch kann die Linearisierungsfunktion zwischen zwei Punkten als linear angesehen und eine gemessene Temperatur mit ausreichender Genauigkeit berechnet werden.

Es ist vorteilhaft, wenn die Mitte des interessierenden Messbereichs etwa bei einem ADC-Messwert von 500 liegt, also auch etwa in der Mitte der ADC-Scale. Man erreicht das in gewissen Grenzen durch den Widerstand, mit dem der NTC einen Spannungsteiler bildet (siehe Schaltplan). Bei der Heizungssteuerung beträgt dieser Widerstand 12k für die Außentemperatur und 5k6 für alle anderen Temperaturen. Deshalb unterscheiden sich auch die Kalibrierdaten.

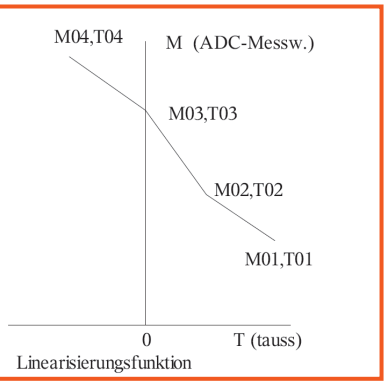

Bild 3.

Task TEMP führt die 4 Messungen im Zyklus von 1 s aus. Sie verwendet für jede Temperatur eine eigene Taskfunktion und zeigt sich dadurch im Hinblick auf das Multitasking kooperativ. Es genügt, nur die Taskfunktion zur Messung der Aussentemperatur (temp2 ()) und deren Kalibirierungsdaten (T01,... M01,...) darzustellen und zu beschreiben.

Temperatur	Name	Analogkanal
Außentemp.	tauss	8
Vorlauftemp. HK1	thk1	9
Warmwassertemp.	tww	10
Temp. Speicher 1	tinput	11

Nachdem mit adc (8); der ADC-Messwert festgestellt wurde, wird er von der Funktion iserror () überprüft. Ein Fehler liegt vor, wenn der ADC-Messwert in der Nähe seiner Ober- oder Untergrenze liegt. Dann kann auf einen Kurzschluss oder eine Unterbrechung der Messleitung geschlossen werden, vorausgesetzt das Interface ist vorhanden. Die Funktion iserror () zeigt nicht nur an, ob ein Fehler besteht, sondern führt auch alles nötige aus, wenn ein Fehler gekommen oder gegangen ist.

Wenn kein Kalibirierungszustand vorliegt, entscheidet der ADC-Messwert, in welchem der drei Bereiche der Linearisierungsfunktion die Berechnung der Außentemperatur ausgeführt werden muss. Das geschieht wie bei der

Kalibirierung

Wenn nach Reset der Eingang für den Zirkulationstaster (Doppelfunktion spart einen Eingang) auf 0 gehalten wird, entsteht der Kalibirierungszustand (siehe auch: Runlevel, Handbetrieb und Netz). Er wird als Fehler gemeldet, und die operativen Tasks starten nicht. Dieser Zustand kann nur durch erneuten Reset verlassen werden.

Task TEMP misst Temperaturen wie gewöhnlich, setzt aber, ohne die Linearisierungsfunktion zu benutzen, die Temperaturvariablen mit dem 1000fachen des ADC-Messwertes, der dann im LC-Display von Alice erscheint.

Zum Kalibrieren muss man die über den Messbereich verteilten 4 Temperaturen herstellen (z.B. Kühltruhe, Eiswasser, Wohnraum u.a.) und mit einem guten Thermometer sowie dem NTC messen. Die Temperatur und ihr ADC-Messwert bilden das Wertepaar M..,T.., das in den Programmtext eingetragen wird.

Kalibrieren ist bei Bobs Programm Sache des Programmierers, nicht des Anwenders.

Heizkurve durch Anwendung der Zweipunktformel aus der analytischen Geometrie.

Dabei ergibt sich z.B. der Ausdruck (T02– T01) / (M02– M01), der nur Konstanten enthält und schon vom Compiler berechnet und als Konstante c0 gespeichert werden kann.

Auch bei dieser Berechnung werden die schon erwähnten mt-PFLOAT-Typen für Temperaturen verwendet, und nur wenn Multiplikationen vor Divisionen ausgeführt werden, sind die Ergebnisse verwertbar.

Die ersten Erfahrungen haben gezeigt, dass die Schwankungen aufeinander folgender Messungen (etwa +/-0,2°C) sich nicht ungünstig auswirken. Sie könnten, falls nötig, z.B. mit der Bildung eines gleitenden Durchschnitts, minimiert werden.

Runlevel, Handbetrieb und Netz

Bobs Heizkreistasks werden von der Task RL über Taskfunktion rl1 () gestartet. Es sind drei runlevel definiert.

Nach Reset herrscht runlevel 1, und der bleibt auch bestehen, wenn der Kalibrierungszustand festgestellt wird. In diesem Fall beendet sich Task RL. Daher kann dieser Zustand nur nach erneutem Reset verlassen werden.

Wenn nach Reset nicht kalibriert werden soll, wird sofort runlevel 2 gesetzt. Der bleibt solange erhalten, bis aktuelle Daten über das Netz zur Verfügung stehen, ohne die es sinnlos wäre, die Steuerung zu betreiben.

Wenn das Netz steht (netstatus = ON), wird in den endgültigen runlevel 3 geschaltet. Nach 1 s wird davon ausgegangen, dass aktuelle Daten über das Netz eingetroffen sind, und Taskfunktion rl2 () startet die Heizkreistasks.

Jetzt hat Task RL in Gestalt von Taskfunktion rl3 () nur noch die Aufgabe, zyklisch den Eingang für die Automatik-/Handumschaltung abzufragen.

Handbetrieb ist für die Steuerung nichts weiter als ein Fehler, bei dem alle operativen Tasks ihre Arbeit einstellen. So können im Notfall starkstromseitig die Pumpen von Hand ein- und ausgeschaltet werden, und auch das Mischventil ist von Hand zu bedienen.

Bei Reset von Alice im laufenden Betrieb ist das Netz für kurze Zeit nicht in Betrieb. Bobs Task RL bleibt im runlevel 3 und alle Tasks arbeiten weiter, da die Daten genügend aktuell sind. Umgekehrt verhält sich Alice bei Reset von Bob genauso.

In der konstanten Netzliste, dem wichtigsten Teil der net.c-Programmierschnittstelle, sind Daten von netzglobalen Variablen enthalten. Netzbetrieb ist nur möglich, wenn der Bezug auf eine bestimmte netzglobale Variable in Alices Netzliste die gleiche Position einnimmt wie in Bobs Liste und wenn die Typen übereinstimmen. Es dient der Übersichtlichkeit, wenn auch die Namen gleich sind, was aber nicht zwingend erforderlich ist. Die Netzwerkteilnehmer verständigen sich über das Netz nur mit Hilfe der Indices, die auch in der Funktion net () verwendet werden. (Wer möchte, kann für die Indices auch Namen definieren.)

Die Funktion fnet (), ein optionaler Teil der Programmierschnittstelle, wird vom Netzwerkmodul aufgerufen, wenn das Netz kommt oder geht. Netzwerkteilnehmer nutzen diese Funktion, um bei kommendem Netz die Daten zu verbreiten, die nicht zyklisch oder in großen Zeitabständen entstehen. Kommt das Netz, weil ein Teilnehmer selbst einen Reset ausführt, könnten Daten nicht aktuell sein, weil Tasks, die sie erzeugen, noch nicht gestartet sind. Daher muss eine Task nach ihrem Start, die betreffenden Daten auch selbst verbreiten, sobald sie aktuell sind. Das ist zum Beispiel der Fall bei ta24, der mittleren Außentemperatur der letzten 24 Stunden mit Netzlistenindex 14.

◆

```
char netstatus = OFF;
//Task RL
char runlevel = 1;
// 1 nach Reset u. beim Kalibrieren
// 2 nach 1 (Netz erwarten)
// 3 nach 2 (Auto- oder Handbetrieb)
void rl3 () {
  if (!AUTO) {
    if (error [HAND] == ,-') {
      error  [HAND] = ,1';
      net ( 0, PUT);
    }
    ts = NOKCYC;
  }
  else {
    if (error [HAND] != ,-') {
      error [HAND] = ,-';
      ts = OKCYC;
      net ( 0, PUT);
    }
  }
  mtdelay ( 1000, rl3 );
}

void rl2 () {
  start (HZAKTIV, 1, hzaktiv1);
  start (HK1,1, hk11);
  start (HK1CTRL,1, hk1ctrl1);
  start (HK2, 1, hk21);
  start (WW, 1, ww1);
  start (ZK, 1, zk1);
  mtcoop (rl3);
}

void rl1 () {
  switch (runlevel) {
    case 1 :  if (NOCAL) runlevel = 2;
                else {
                error [CAL] = ,1';
                ts = NOKCYC;  //Task RL beenden
                return;
              }
              break;

    case 2 :  if (netstatus == ON ) {
                runlevel = 3;
                mtdelay ( 1000, rl2  );
                return;
              }
              break;
  }
  mtdelay (200, rl1);
}
```