LIAB ApS
Østre Allé 6
DK-9530 Støvring
Tlf: +45 98 37 06 44
http://www.liab.dk

Energinet.dk
Tonne Kjærsvej 65
DK-7000 Fredericia
Att: Steen Kramer

Journal nr. 2011-03-08/1 Støvring den 8. marts 2011

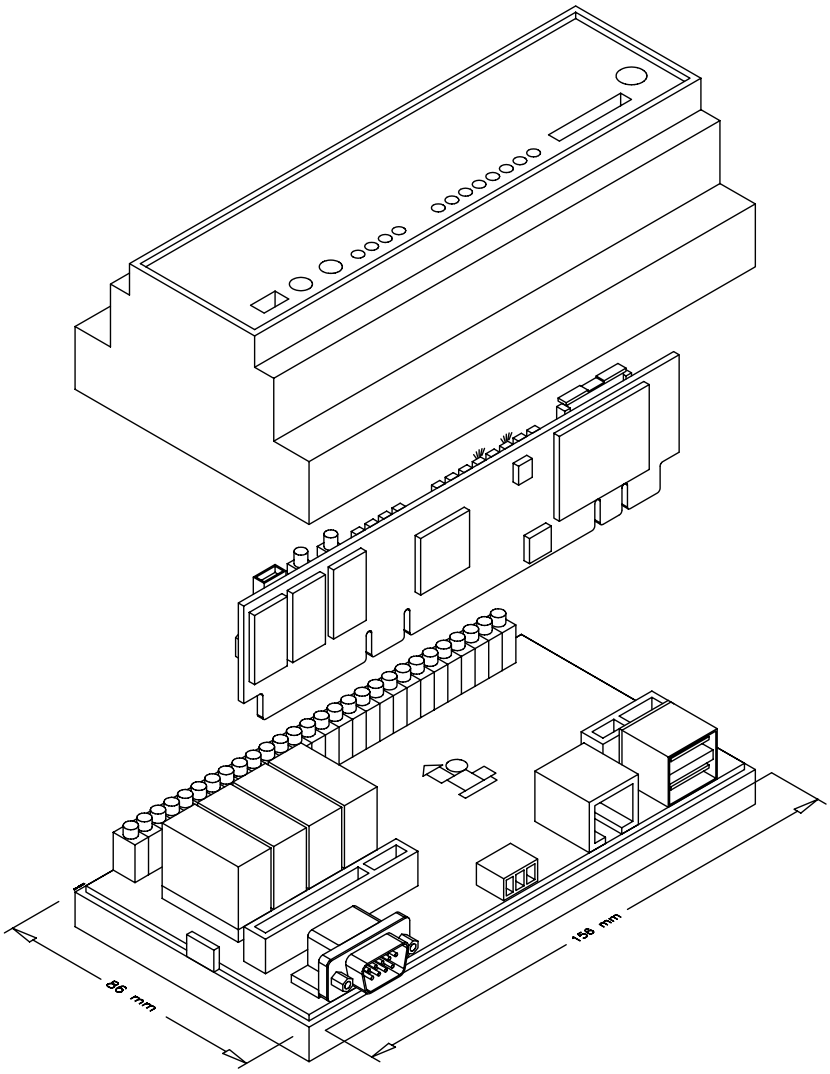
Software Development Kit for LIABSG computer

I forbindelse med demonstrationprojektet vedrørende dataopsamling og styring af varmepumper i private hjem, er der blevet opbygget styringscomputer, der monteres ved siden af varmepumpen. Denne computer indeholder en række analoge og digitale indgange, samt tre relæer. Fem temperatursensorer, to flowmålere og en elmåler er tilsluttet indgangene. En eller flere relæer kan benyttes til styre den tilkoblede varmepumpe, typisk ved enten at tvangsstoppe den eller få den til at skifte mellem to instillinger. Til computeren er der således skrevet en mængde software til at håndtere disse ind- og udgange. Det primære program til styringen kaldes kontroldæmonen (programmet er navngivet "`contdaem`" i Linux-systemet, kort for "Control Daemon"). Som navnet angiver er kontroldæmonen et program, der kører i baggrunden og til stadighed udfører en række opgaver med at aflæse indgangene for derefter at regulere på udgangene efter de styringsalgoritmer, der er indkodet i kontroldæmonen. Det er en relativt nem sag at udvide kontroldæmonen, idet den er modulært opbygget. Hvilke moduler en given styring er opbygget af, beskrives i en konfigurationsfil, skrevet i XML. Ny moduler skrives i programmeringssproget C.

Det primære formål med denne rapport er at give en introduktion til konfigurering af kontroldæmonen og udvikling af nye moduler til den. I det følgende vil der dog først blive givet et overblik over LIABSG computeren og den tilhørende styrebox, der er udviklet til demonstrationsprojektet. Ligeledes vil der blive givet en beskrivelse af hvordan man installerer den nødvendige udviklingsoftware på en PC med Ubuntu Linux.

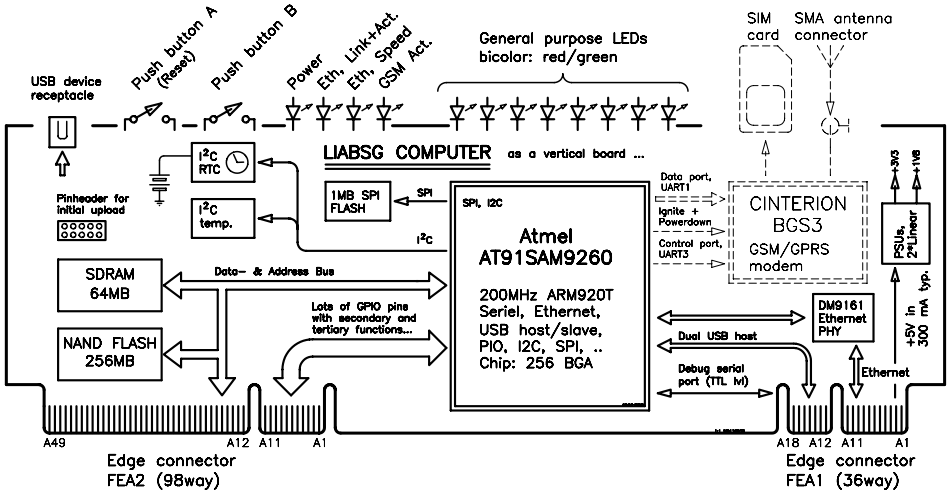
1 LIABSG computeren

Som intelligent enhed i styringsboxen benyttes en af LIAB udviklet computer: LIABSG. "SG" står for "Smart Grid", det almindelige udtryk for systemer til automatisk styring af elforbruget hos den enkelte elkunde. På Fig. 1 ses en eksploderet tegning af LIABSG computeren. Den egentlige computer, LIABSG CPU, ses som det lodrette print, der kan indskydes i kantkonnektorerne på det vandrette bæreprint nederst på tegningen. Ud over konnektorer og stik er der på bæreprintet elektronik (PT1000 signalfortærkere, A/D-konvertere, relæer, mm.) der er tilpasset varmepumpestyringen i demonstrationsprojektet. De to print: LIABSG CPU og bæreprintet er designet således, at de kan monteres i den standard DIN-skinne box, som LIAB traditionelt benytter til styringscomputere. Boxen en 9 enheder (9U) bred, svarende til 156 mm. Kassen opfylder normen DIN 43880, hvilket gør at den nemt kan monteres i almindelige elskabe og tavler.



Figur 1: *LIABSG computeren: den lodrette LIABSG CPU og et bæreprint*

Det lodrette print: LIABSG CPU udgør en komplet Linux computer med processor, RAM, FLASH og Ethernet. En blokdiagram af computeren er vist på Fig. 2. Tilslutninger mellem computeren og omgivelserne designes ind på bæreprintet, idet konnektorer og stik er placeret her. Computeren kommer i to versioner: med og uden GSM modem. Computere som bruges i demonstrationsprojektet er uden modem.

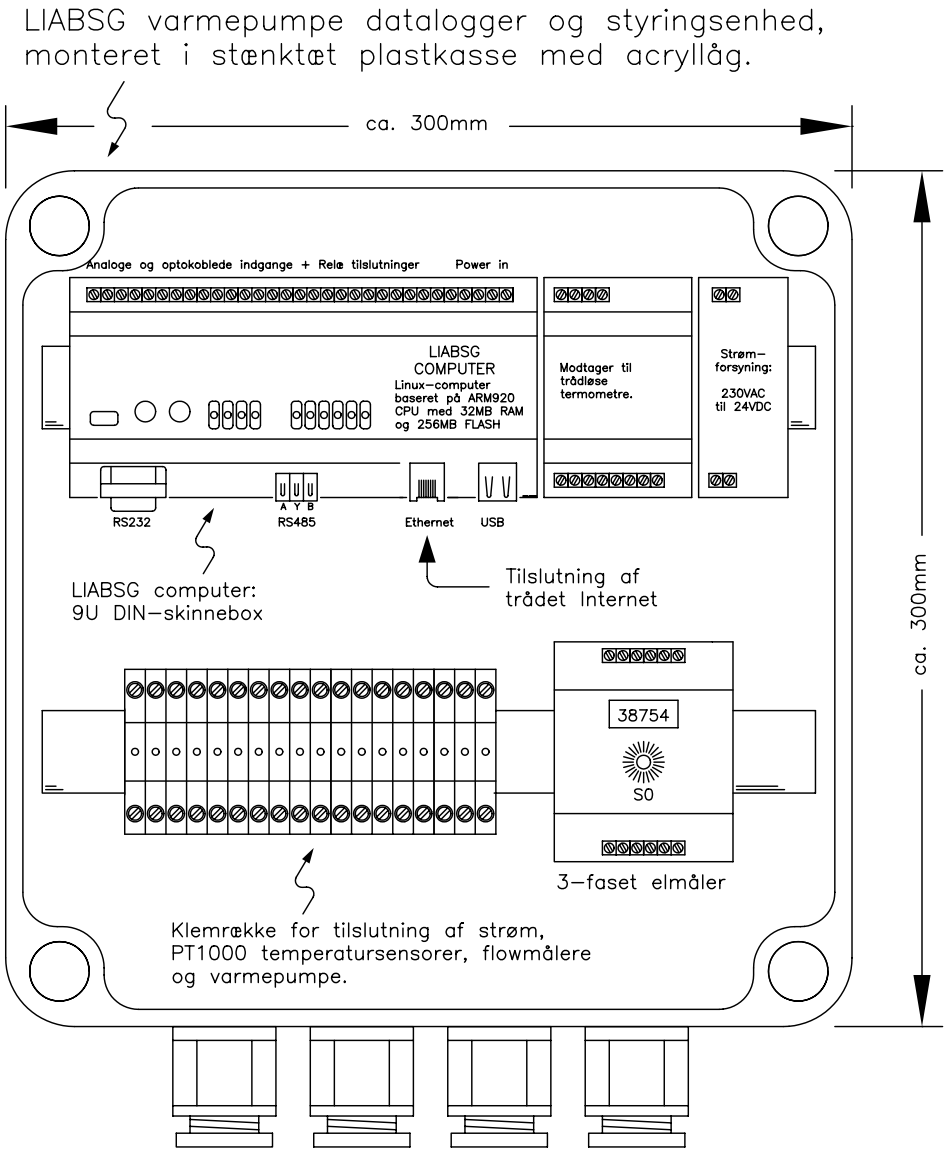


Figur 2: *LIABSG CPU: en Linux-computer til styring og dataopsamling.*

Operativsystemet Linux benyttes til LIABSG computeren, hvilket giver en række fordele: Linux er et velkendt operativsystem med stor udbredelse og er ikke mindst kendt for sin stabilitet. Samtidig indeholder Linuxsystemer faciliteter såsom multitasking, filsystemer, netværk, USB support, helt på linie med hvad MS Windows kan tilbyde. En lang række standardapplikationer fås til Linux, f.eks. web-servere, databaser, XML-applikationer, mm. Ovenpå Linux programmeres herefter selve styrings- og dataopsamlingsapplikationen, specielt kontroldæmonen. Til demonstrationsprojektet placeres LIABSG computeren i en monteringsbox med acryllåg sammen med en række andre komponenter: en energimåler, en modtager til trådløse termometre og en strømforsyning. Der er ligeledes monteret en klemrække, der benyttes når elektrikeren skal tilslutte boxen til varmepumpen og sensorerne. Den færdige monteringsbox ses på Fig. 3.

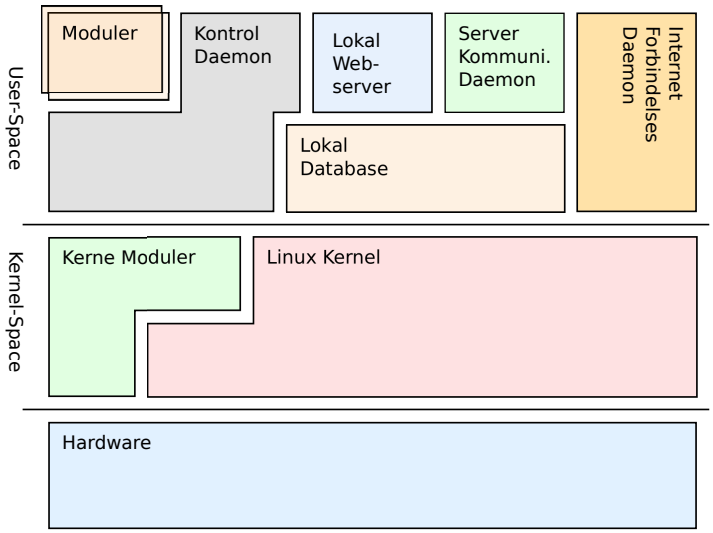
2 Software til LIABSG computeren

På LIABSG computeren benyttes en relativ ny version af Linux styresystemet, en Linux kerne version 2.6.29 eller nyere. LIAB standard embeddede Linux-distribution lægges i systemets ramdisk.



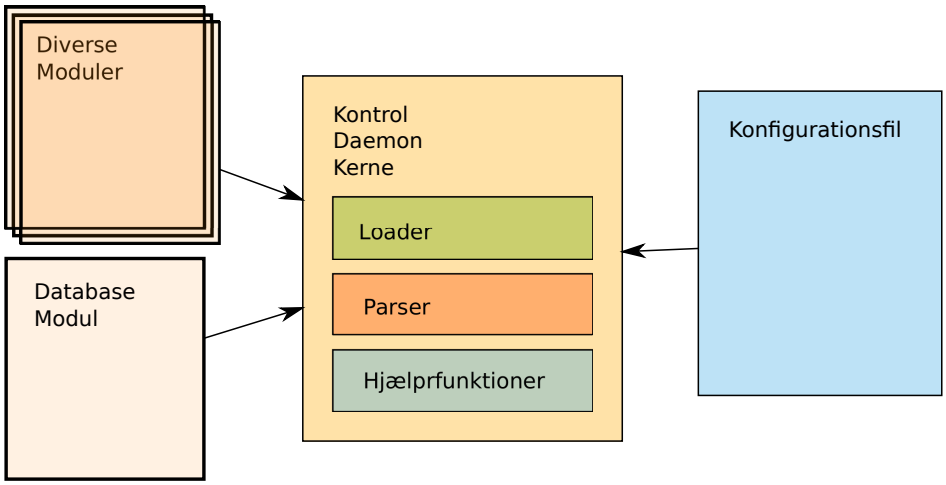
Figur 3: *Styrebox til demonstrationsprojektet.*

Under opstart, som tager ca. 30 sekunder, startes en række services såsom netværk, web-server, mm. Til slut startes LIAB's styrings- og dataloggerapplikationer, ikke mindst kontroldæmonen. På Fig. 4 ses en blokdiagram over applikationerne Nederst haves hardware snitfladerne i LIABSG'en, inklusiv A/D-konvertere, optokoblede indgange og relæer som beskrevet i sidste afsnit Til alle hardware elementer er skrevet drivere, ofte i form a Linux kernemoduler, som kører i kernel space. Kontroldæmonen er derimod et user space program som kører ved siden af en lang række andre programmer. Ved hjælp af den lokale web-server er det muligt at vise øjebliksværdier af variable i kontroldæmonen på en lokal hjemmeside. Sidst giver Server Kommunikations Dæmonen og Internet Forbindelses Dæmonen mulighed for at indsamlede data kan sendes videre via Internettet til de til demonstrationsprojektet tilknyttede centrale serverne.



Figur 4: *Lagdelt model af softwaren i LIABSG styreboxen.*

På Fig. 5 ses en figur af kontroldæmonen med de tilknyttede ressourcer. Dæmonens opførsel styres af en konfigurationsfil, der specificeres i XML-formatet. Til dæmonen er tilknyttet en række moduler. Nye moduler skrives i sproget C og kompileres på en host-PC med Linux. De kompilerede modulerne i form af såkaldte "shared objects" (`.so`-filer) indlægges nu i et bestemt direktorie på LIABSG computeren. Hvilke moduler der skal loades specificeres i konfigurationsfilen. Ændringer i moduler og konfigurationsfil får effekt når kontroldaemonen genstartes.



Figur 5: *Kontroldæmonen og dens ressourcer*

3 Udviklingsmiljø til en Linux Host PC

Før man kan gå igang med at skrive sine egne moduler til kontroldæmonen skal man have de nødvendige værktøjer installeret på en host PC, som benytter operativsystemet Linux. En PC'er med en frisk installation af Ubuntu 10.04 eller 10.10 vil være et godt udgangspunkt. Følgende yderligere pakker skal installeres på host PC'en:

```
..$ sudo apt-get install build-essential
..$ sudo apt-get install automake
..$ sudo apt-get install autoconf
..$ sudo apt-get install libtool
```

Til kontroldæmonen er der fra LIAB's side udgivet en DVD der dels indeholder den komplette distribution til LIABSG computeren, dels den nødvendige software til at udvikle programmer til kontroldæmonen. Den nævnte distribution indeholder også alle dele af softwaren til demonstrationsprojektet vedrørende varmepumper. På DVD'en findes følgende to tar-arkiver:

```
LIABSG-heatpump-feb-2011.tar.bz2  og
contdaem-src-feb2011.tgz
```

Før det er muligt at udvikle programmer til LIABSG'en skal de to ovennævnte arkiv-filer installeres på PC'eren, f.eks. i brugerens hjemmedirektorie. Det er vigtigt at udpakningen foregår med `root`-rettigheder, f.eks. på følgende måde:

```
 ..$ cd
 ..$ sudo tar -xvjf /media/LIABSGheatpumpFeb2010...
                   .../LIABSG-heatpump-feb-2011.tar.bz2
[sudo] password for xxx: <enter your password>
 ..$ sudo tar -xvzf /media/LIABSGheatpumpFeb2010...
                   .../contdaem-src-feb2011.tgz
```

Nu haves to direktorierne: `LIABSG-heatpump-feb-2011` og `contdaem-src-feb2011`. For at kunne arbejde på filerne i disse direktorier som almidelige bruger, skal alle filer med userid `998` (og kun disse) have ændret ejerskab. Dette gøres nemmest med `find`-kommandoen:

```
 ..$ sudo find LIABSG-heatpump-feb-2011 -user 998 ...
                ... -exec chown <usr>:<grp> {} \;
 ..$ sudo find contdaem-src-feb2011 -user 998 ...
                ... -exec chown <usr>:<grp> {} \;
 ..$ sudo chown <usr>:<grp> ...
                ... LIABSG-heatpump-feb-2011 contdaem-src-feb2011
```

hvor <usr>:<grp> angiver brugerens userid og group på host PC'en. Den sidste linie ændre ejerskabet af direktorierne selv.

Da vi på LIABSG computeren har en anden processorarkitektur (ARM) på target end på host PC'en (x86), skal man have installeret en krydskompiler, som kan findes i LIABSG-heatpump-feb-2011 direktoriet:

```
..$ sudo cd LIABSG-heatpump-feb-2011/software/crosscompiler
..$ sudo cp -a opt /
```

Man kan nu benytte krydscompileren ved at bruge den fulde sti:

```
..$ /opt/crosstool/gcc-4.0.2-glibc-2.3.6/arm-unknown-linux-gnu/ ..
    ..  bin/arm-softfloat-linux-gnu-gcc
```

men indsættes følgende linie nederst i filen .bash_profile i brugerens hjemmedirektorie:

```
PATH=$PATH:/opt/crosstool/gcc-4.0.2-glibc-2.3.6/ ...
   arm-unknown-linux-gnu/bin
export PATH
```

kan man kalde krydscompileren således:

```
..$ arm-unknown-linux-gnu-gcc hello.c
```

4 Kommunikation med LIABSG

Når en styrebox med en LIABSG opsættes ved siden af en varmepumpe for derefter at blive tilsluttet til internettet, vil al opstart og kommunination med de centrale servere foregå automatisk. Ønsker man at ændre på kontroldæmonens opførsel eller indføre ny moduler, må man have mulighed for at kommunikere med den direkte. Da LIABSG computeren benytter DHCP til at opnå en (lokal) IP-adresse, må man først idetificere denne adresse. Dette kan gøres på følgende måde:

```
..$ sudo nmap -sP <ip0>.<ip1>.<ip2>.* | grep -A1 Liab
    f.eks:
..$ sudo nmap -sP 192.168.1.* | grep -A1 Liab
AC Address: 00:15:8C:00:18:3A (Liab ApS)
Nmap scan report for 192.168.1.223
```

hvor <ip0>.<ip1>.<ip2> er de først tre bytes i det IP-range ens udstyr sidder på. Resultatet af nmap-kommandoen finder kun en enhed med MAC-nummer indenfor LIAB's range: 192.168.1.223.

Det bør nu være muligt at logge ind på LIABSG'en ved hjælp af telnet:

```
..$ telnet 192.168.1.223
Trying 192.168.1.223...
Connected to 192.168.1.223.
Escape character is '^]'.

Linux 2.6.29.4 (liab.liab.dk) (ttyp0)
liab login:
```

Man kan da prøve at logge ind med userid og password hhv. root og skov9240 og se hvilke processer der kører:

```
liab login: root
Password:
root@liab# ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:09 init [3]
    2 ?        S<     0:00 [kthreadd]
    3 ?        S<     0:00 [ksoftirqd/0]
  ...
  ...
  981 ?        S      0:00 /usr/bin/contdaem
  989 ?        S      1:39 /usr/bin/contdaem
  990 ?        S      0:00 /usr/bin/contdaem < mange af dem!>
  ...
  1054 ?       SLs    0:00 /usr/sbin/watchdog
  ...
root@liab#
```

Skriv nu en passende version af det klassiske hello world program:

```
#include <stdio.h>

int main(int narg, char *argp[])
{
    printf("Hello from LIABSG!\n");
}
```

lagt i filen hello.c. Kompilér det med den installerede krydscompiler:

```
..$ arm-unknown-linux-gnu-gcc hello.c -o hello
```

Gå tilbage til dit login på LIABSG'en og udfør følgende kommando:

```
root@liab# cat > .rhosts
<host-PC'erens IP-nummer> <udviklerens userid>
<ctrl-D>
root@liab#
```

På denne måde er det nu muligt at benytte Linux-kommandoen "`rcp`" (remote copy) til at overføre filer fra host-PC'eren til LIABSG'en via netværk. Ønskes filen `hello` overført gøres dette på følgende måde fra host PC'en:

```
..$ rcp hello root@192.168.1.223:.
```

hvorefter filen ligger i `root`'s hjemmedirektorie: `/root`.

Fra `telnet`-sessionen til LIABSG computeren kan man nu starte `hello` programmet:

```
root@liab# ./hello
Hello from LIABSG!
root@liab#
```

Kan man få ovenstående resultat frem, kan man være rimeligt sikker på, at man har fået sat udviklingsmiljøet ordentligt op!

5 Kompilering og dokumentation

Nu hvor udviklingsmiljøet er korrekt sat op, bør det være muligt at kompilere sin egen version af kontroldæmonen:

```
..$ cd contdaemsrc
..$ make all
 list=''; for subdir in $list; do \
  make  $subdir-lib  ; done
 list='logdb xmlparser aeeprom contdaem cmddb db-...
  make  $subdir-mk  ; done
 make[1]: Entering directory '/home/midi/liabcpgm...
 cd logdb ; ./autogen.sh
 libtoolize: putting auxiliary files in '.'.
 libtoolize: copying file './config.guess'
 libtoolize: copying file './config.sub'
 ...
```

```
  <kompilering i 3 til 10 minutter afhængig af host PC'en >
  ...
  usr/lib/libxmlpar.so.0.0.0
  usr/lib/libaeeprom.so
  ls -l  contdaem.tar.gz
  -rw-r--r-- 1 midi midi 335010 2011-03-15 20:18 contdaem.tar.gz
 ..$
```

Ovenstående angiver at kompileringen gik godt!

I direktoriet `contdaemsrc` findes alle kildetekster til kontroldæmonen og ud over en `Makefile` findes følgende underdirektorier:

```
aeeprom     cmddb       deviceroot  modbusd
armlib      contdaem    doxdoc      templatemod
buildroot   db-module   logdb       xmlparser
```

I Tabel 1 ses en beskrivelse af indholdet i de enkelte underdirektorier. Online-dokumentationen er skevet som doxygen-kommentarer i de relevante kildetekster. For at generere og læse i dokumentationen skrives:

```
..$ make dox
..$ firefox doxdoc/html/index.html
```

I Tabel 2 på side 20 haves en liste over udviklede moduler pr. 21. marts 2011.

6 Konfigurationsfiler

Før man går igang med at kode et nyt modul i programmeringssproget C til kontroldæmonen, er det værd at checke om man ikke kan klare sig med moduler, som allerede ER skrevet. Nedenfor er vist en simpelt konfigurationsfi, `simpleconf.xml`, der sammenknytter tryk på tryk på den højre tast på fronten af LIABSG computeren med relæ no. 1!

```
<!--  This is the first and very simple module for the
      LIABSG control daemon! Mikael Dich, Feb 2011
-->
<modules>
     <module type="keyinput" name="midisinput" >
       <input device="/dev/input/event0"/>
       <trigger text="Test" unit="" name="button" key_code="0x106" />
     </module>
     <module type="relay" name="midisrelay" verbose="0">
       <output device="/sys/class/leds/relay_1/brightness"/>
```

```
      <listen event="midisinput.button" />
    </module>
</modules>
```

Direktorier:	Indhold:
Online dokumentation:	
`doxdoc`	Doxygen dokumetation i bla. HTML og LaTeX
Biblioteker som kontroldæmonen anvender:	

`armlib`
`xmlparser`
`logdb`
`aeeprom`
	Generelt bibliotek med expat XM parser og sqlite db.
	Overbygning til expat XML parseren.
	Faciliteter til datalogning.
	Faciliteter til LIABSG baseboard EEPROM.

Direktorietræer:	
`buildroot` `deviceroot`	Direktorietræ hvori contdaem opbygges. Ekstrakt af `buildroot` som indeholder alle filer, der skal kopieres til `/` på LIABSG'eren.
Programmer og konfigurationsfiler:	
`contdaem`	Kildeteksten til kontroldæmon-programmet! I direktoriet `contdaem/xmlfiles` findes et antal konfigurationsfiler. I direktoriet `contdaem/src/modules` findes kildetekster til et antal moduler.
Kildetekst til moduler:	

`cmddb`
`db-module`
`modbusd`
`templatemod`
	Modul til kommandointerfacet.
	Modul til datalogning.
	Modul til MODBUS (til trådløse termometre).
	Modul-template til det videre arbejde!

Tabel 1: *Beskrivelse af direktorier under* `contdaemsrc`

Følgende karakteristika ses for en konfigurationsfil:

- En konfigurationsfil skal skrives efter XML-standarden og øverst i filen ses hvordan man laver en kommentar. Herefter kommer en gruppe afgrænset af markeringerne `<modules>` og `\verb</modules>`. Indenfor denne gruppe kan man have en eller flere instanser af forskellige moduler.
- En instans af et modul angives med markeringerne `<module ...>` og indtil den matchende `</module>`. I første markering angives hvilken modultype det drejer sig om (`keyinput`) og hvad den given instans af modulet skal hedde (`midisinput`). Herefter kommer et antal tags, der er specifikke for de enkelte moduler. Modultypen skal matche navnet på et modul

i direktoriet `/usr/share/control-daemon/plugins`. For at de viste, simple modul skal kunne fungere, skal man kunne finde modul-filerne `keyinput.so` og `relay.so` i `plugin`-direktoriet.

- En essentiel mekanisme til kommunikation mellem moduler er `events`. En instans af et modul kan sende et event, annonceret gennem et `trigger`-tag: I `keyinput` instansen annonceres en event med navnet `button`. `keyinput`-moduler er programmeret således at både key-press og key-release trigger et event. Der overføres yderligere data med hvert event genner en struct af typen `uni_data`.
- Instansen af `relay`-modulet lytter netop på events fra `keyinput`-modulet. Output sættes efter de med eventen overførte data: `event->data`.

For at prøve ovenstående modul udføres først følgende kommandoer på LIABSG'en:

```
root@liab# killall -9 watchdog
root@liab# killall -9 contdaem
```

Herefter går man til direktorier `contdaemsrc`, hvor man har bygget den nyeste version af kontroldæmonen. Nu kopieres

```
..$ rcp -r deviceroot/* root@192.168.1.223:/
..$ rcp xmlfiles/simpleconf.xml root@192.168.1.223:
```

Første kommando opdaterer hele kontroldæmon-systemet og anden linie lægger den simple konfigurationsfil i `/root`. På LIABSG'eren udføres nu følgende kommando:

```
root@liab# contdaem -h
contdaem: Daemon for logging data and controling peripherals ...
 -m <conf>    : Set module configuration file (default /etc/c...
 -P <dir>     : Set plugin directory (default /usr/share/cont...
 -p <dir>     : Set extra plugin directory
 -f           : Run contdaem in the foreground.
 -h           : Help (this text)
LIAB ApS, Christian Rostgaard, February 2010
root@liab# contdaem -f -m ./simpleconf.xml
pid in file is  1107
Loading types from /usr/share/control-daemon/plugins/dbmodule.so
Loading types from /usr/share/control-daemon/plugins/threshold.so
...
Types loaded:
Module type dblogger
Module type threshold
```

```
...
<ctrl-C>
deleting modules
unloading module: midisrelay
unloading module: midisrelay
unloading module: midisinput
root@liab#
```

Efter start af kontroldæmonen med kommandoen: `contdaem -f -m ...` kan man prøve at trykke på højre knap på LIABSG computeren. Man skulle nu gerne høre relæet klikke inde i den!

7 Programmering af modul til kontroldæmon

Efterhånder er der udviklet adskillige moduler til kontroldæmonen. I mange tilfælde kan man løse en given opgave ved at kombinere fukntionaliteten af et par moduler. Har man specieller ønsker kan man dog komme i en situation, hvor det er nødvendigt at selv at skrive et modul. Udvikling af moduler sker nemmest i direktoriet, hvor de forskellige templates til modulerne ligger: `contdaem_src/templatemod/src`. Den simpleste template findes i `verysimplemod.c` og er listet nedenfor. For at kompilere dette modul udføres følgende kommandoer (vi antager at vi starter i direktorier `contdaem_src`, ligesom da vi byggede selve kontroldæmonen):

```
..$ pwd
......./contdaem_src
..$ cd templatemod/src
..$ cp verysimplemod.c templatemod.c
..$ cd ../..
..$ make templatemod-mk

..$ rcp xmlfiles/verysimplemod_conf.xml root@192.168.1.223:
```

Det kompilerede modul ligger nu i direktoriet
`contdaem_src/buildroot/usr/share/control-daemon/plugins`
i form af et "shared object" (`*.so`-fil), som kontroldæmonen kan loade under opstart. Denne fil skal nu kopieres over til direktoriet:
`/usr/share/control-daemon/plugins`
på target, dvs. på LIABSG computeren:

```
..$ rcp buildroot/usr/share/control-daemon/plugins/...
     ...templatemod.so root@192.168.1.223:/usr/...
     ...share/control-daemon/plugins
```

(en lang linie, knækket af hensyn til dokumentet). Med følgende konfigurationsfil (`verysimplemod_conf.xml`) oprettes en instans af vores meget simple modul. Det har ikke megen praktisk gavn, idet det eneste der sker i modulet er, at det hvert sekund udskriver en tællervariabel til `stdout`:

```
<!--  This is the first and very simple module for the
      LIABSG control daemon! Mikael Dich, Feb 2011
-->
<modules>
     <module type="keyinput" name="midisinput" >
       <input device="/dev/input/event0"/>
       <trigger text="Test" unit="" name="button" key_code="0x106" />
     </module>
     <module type="templatemod" name="midistemp" >
     </module>
     <module type="relay" name="midisrelay" verbose="0">
       <output device="/sys/class/leds/relay_1/brightness"/>
       <listen event="midisinput.button" />
     </module>
</modules>
```

Hele kildeteksten til vores simple modul: `verysimplemod.c` er listet nedenfor. De essentielle elementer gennemgås efter kildeteksten.

```
/*
** contdaem-module: Eksempel på et simpelt modul, skrevet i
** programmeringssproget C, til LIABSG's kontroldaemon.
** LIAB ApS, marts 2011.
*/
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "module_base.h"

/*
** Datastruktur til at opbevare alle private data for vores
** modul. Ligeledes haves en wrapperfunktion, som returnerer
** en pointer til instansen af TemplateObject
*/
typedef struct sTemplateObject
{
    struct module_base base;
    int count;
} TemplateObject;
```

```
TemplateObject* module_get_struct(struct module_base *module)
{
    return (TemplateObject*)module;
}

/*
** Init-funktion: blive kaldt en gang når contdaem starter.
*/
int module_init(struct module_base *base, const char **attr)
{
    TemplateObject *this;
    this = module_get_struct(base);
    this->count = 0;
    return 0;
}

/*
** loop-funktion: blive kaldt når contdaem er færdig med sine
** init-funktioner. Loop-funktionen skal loope så længe base->run er
** forskellige fra 0.
*/
void *module_loop(void *parm)
{

    struct module_base *base;
    TemplateObject *this;

    base = (struct module_base *)parm;
    this = module_get_struct(base);

    while(base->run)
    {
        printf("%2d ", this->count); fflush(stdout);
        this->count++;
        sleep(1);
    }
}

/*
** Foskellige instanser af datastrukturer, som selve contdaem har brug
** for til at holde styr på instanser af modulet.
*/

/* Liste af XML tags, der er definerer for dette modul: */
struct xml_tag module_tags[] =
{
    {
        .el     = NULL,
```

```
        .parent = NULL,
        .start  = NULL,
    }
};

/* Liste af event handlers, der er definerer for dette modul: */
struct event_handler handlers[] =
{
    {
        .name = ""
    }
};

/* Central datastruktur for modulet: */
struct module_type module_type =
{
    .name             = "templatemod",
    .xml_tags         = module_tags,
    .handlers         = handlers,
    .type_struct_size = sizeof(TemplateObject),
};

struct module_type *module_get_type()
{
    return &module_type;
}
```

- Kildeteksten til et modul er blot en almindelig fil med C-kode, som starter med at inkludere relevante header-filer. Den centrale header-fil for kontroldæmonen er `module_base.h`.

- For at kunne gemme data mellem forskellige funktioner i modulkoden behøves en passende datastruktur. I det viste eksempel kaldes denne for `TemplateObject` og det er helt op til brugeren at definere relevante elementer i denne. Hver gang der oprettes en instans af modulet bliver der allokeret plads til en instans af denne struktur.

- Funktionen `module_get_struct` skal defineres og kan være så simpel som angivet i ovenstående kildetekst.

- Funktionen `module_init` kaldes når kontroldæmonen er startet og den har loadet alle sine moduler. Funktionen kaldes en gang for hver instans, der oprettes i konfigurationsfilen.

- Funktionen `module_loop` blive kaldt når kontroldæmonen er færdig med alle init-funktioner. Loop-funktionen forventes at loope så længe base->run forskellige fra 0.

- Til slut haves forskellige datastrukturer, som indeholder information om XML tags og event-handlere. Instanser af disse strukturer samles sammen i den overordnede datastruktur: `module_type`. En instant af denne datastruktur med navnet `module_type` skal eksistere i slutningen af hver kildetekst for et modul. Endelig skal man som angivet definere funktionen `module_get_type`.

Som sagt gør dette første modul ikke meget gavn ud over at skrive tællervariablen `count` ud på `stdout`. For at få et første modul, der rent faktisk kan bruge til noget praktisk, indfører vi i loop-funktionen en funktion, der kan sende et event til et andet modul. Med faste mellemrum sender modulet et event, som andre moduler kan lytte på. Intervallet mellem to events bestemmes fra konfigurationsfilen. Kildeteksten for dette modul findes i filen `simplesendeventmod.c`.

Først indfører vi to elementer i vores datastruktur `TemplateObject`. Det ene, nye element benyttes til at gemme information om eventet, det andet til at gemme interval-tiden:

```
typedef struct sTemplateObject
{
    struct module_base base;
    struct event_type *TikTakEvent;
    int interval;
    int count;
} TemplateObject;
```

I slutningen af kildeteksten udvides arrayet `module_tags` af typen `xml_tag` med nedenstående element, således at man kan referere til vores event-sender med XML-tag'et `<tiktak ... \>`: Funktionen `TikTakEventSenderStart` kaldes når XML-parseren møder dette tag.

```
/* Liste af XML tags, der er definerer for dette modul: */
struct xml_tag module_tags[] =
{
    {
        .el     = "tiktak",
        .parent = "module",
        .start  = TikTakEventSenderStart,
    },
    {
        .el     = NULL,
    },

};
```

Funktionen `TikTakEventSenderStart` indeholder følgende kode, der dels aflæser

en eventul parameter med navnet `interval`, dels knytter et event til eventuelle instanser af vores modul:

```
#define MINIMUMINTERVAL (100)
#define DEFAULTINTERVAL (1000)
#define MAXIMUMINTERVAL (10000)

int TikTakEventSenderStart(XML_START_PAR)
{
    struct modules *modules;
    struct module_base *base;
    TemplateObject *this;

    modules = (struct modules*)data;
    base    = ele->parent->data;
    this    = module_get_struct(base);

    this->interval = get_attr_int(attr, "interval", DEFAULTINTERVAL);
    if (this->interval < MINIMUMINTERVAL)
        this->interval = MINIMUMINTERVAL;
    if (this->interval > MAXIMUMINTERVAL)
        this->interval = MAXIMUMINTERVAL;

    this->TikTakEvent = event_type_create_attr(base, NULL, attr);
    base->event_types = event_type_add(base->event_types,
                                       this->TikTakEvent);
    return 0;
}
```

Sidst indføres følgende kode i loop-funktionen, således at der genereres et event med det foreskrevne interval:

```
  ...
    while(base->run)
    {
        printf("%2d ", this->count); fflush(stdout);
        this->count++;

        if(this->TikTakEvent != NULL)
        {
            data  = uni_data_create_int(this->count % 2);
            event = module_event_create(this->TikTakEvent, data, NULL);
            module_base_send_event(event);
        }
        usleep(1000*this->interval);
    }
  ...
```

Som tidligere omtalt følger der med hvert event en instans af typen `uni_data`. Med funktionskaldet `uni_data_create_int` oprettes en instans af nævnte type og som parres med eventet i næste linie. Dernæst at blive afsendt til moduler, der lytter på dette event. Heltalt-værdien der afsendes skifter mellem 0 og 1 som funktion af beregningen `(this->count % 2)`.

For at bygge det nye modul udføres nu følgende kommandoer på host-PC'en:

```
..$ pwd
......./contdaem_src
..$ cd templatemod/src
..$ cp simplesendeventmod.c templatemod.c
..$ cd ../..
..$ make templatemod-mk
..$ rcp buildroot/usr/share/contdaem osv.
```

Dernæst overføres følgende konfigurationsfil (`simplesendeventmod_conf.xml`):

```
<!--  This is a simple config file with an event sender for the
      LIABSG control daemon! Mikael Dich, Feb 2011
-->
<modules>
     <module type="templatemod" name="midistemp" >
       <tiktak text="First on, then off!" name="tt" interval="1000" />
     </module>

     <module type="relay" name="midisrelay" verbose="0">
       <output device="/sys/class/leds/relay_1/brightness"/>
       <listen event="midistemp.tt" />
     </module>
</modules>
```

til target med følgende kommando:

```
..$ rcp xmlfiles/simplesendeventmod_conf.xml root@192.168.1.223:
```

Starter vi nu kontroldæmonen på LIABSG'eren med følgende kommando:

```
root@liab# contdaem -f -m simplesendeventmod_conf.xml
```

skulle man gerne kunne høre relæet klikke en gang per sekund...

Modulnavn:	Funktion:
`accumulate`	Midling og summering af data fra events over et givet tidsinterval.
`dbmodule`	Logning af data til lokal database.
`cmddb`	Modtagelse af kommandoer.
`keyinput`	Håndtering af key inputs, f.eks. fra `/dev/input/event0`
`ledpanel`	Kontrol af lysdioder på LIABSG fronten.
`readfile`	Læsning fra A/D-konvertere mm. via `/dev`-files.
`relay`	Styring af de tre relæer i LIABSG computeren.
`sysinf`	Indhentning af data for Linux-systemet. (memory forbrug, forsyningsspændinger, mm.)
`templatemod`	Template til modul-programmering.
`threshold`	Databehandling med hysterese og tærskelværdier.
`waterpower`	Energiberegning på basis af temperaturforskel og vandflow.

Tabel 2: *Eksisterende moduler for kontroldæmonen (pr. 21. marts 2011)*