Univerza v Ljubljani
Fakulteta
za računalništvo
in informatiko

Seminarska naloga

Temperaturni senzor na sistemu ARM FRI-SMS

Avtorji:
Miha Zidar
Matic Potočnik
Anže Pečar
Jaka Sivec

Mentor:
dr. Rajko Mahkovic
Aleksander Lukič

19. maj 2011

Kazalo

1 Uvod **2**

2 Opis sistema **2**
2.1 Oprema . . . 2
2.1.1 I^2C . . . 2
2.1.2 LM75 . . . 3
2.2 Server . . . 3
2.3 Client . . . 3

3 Celoten postopek **4**
3.1 Priprava okolja . . . 4
3.2 Priprava jedra . . . 4
3.3 Priprava jedernega modula . . . 5
3.4 Priprava TFTP streznika . . . 5
3.5 priprava FRI-SMS ploščice . . . 6
3.6 Server in Client . . . 7
3.7 Vsi programi . . . 7

4 Izvorna koda **8**
4.1 Server . . . 8
4.2 Client . . . 11
4.3 Kernel module . . . 14
4.4 Syscall Patch file . . . 21
4.5 Asembler testni program . . . 22

1 Uvod

Seminarska naloga predstavlja skupek vseh vaj ki smo jih imeli pri predmetu *Operacijski sistemi 1.* V tej seminarski nalogi smo naredili jedrni modul - gonilnik za temperaturni senzor - za opeacijski sistem Linux ki teče na ARM9 procesorju FRI-SMS razvojne ploščice. Ustvarili smo nov sistemski klic, ki s pomočjo našega gonilnika, preko I^2C vodila, prebere temperaturo z digitalnega senzorja *LM75*. Za prikaz temperature, pa smo naredili tudi strežniški program, ki vsako sekundo prebere temperaturo in jo shrani v krožni pomnilnik. Zraven pa spada še grafični odjemalec, ki s strežnika dobi zadnjo, ali pa zadnjih nekaj prebranih temperatur in jih prikaže končnemu uporabniku.

Kot dodatek smo naredili še testno okolje, s katerim smo testirali delovanje senzorja, neodvisno od operacijskega sistema Linux. To smo naredili z asemblerskim programom, ki naredi dve branji po en bajt iz senzorja v eno-sekundnih intervalih, ter nato podatke posreduje serijskim vratom. Program smo pognali s programom winIDEA, na računalniku pa smo imeli odprt serijski terminal na katerem smo opazovali podatke, ki smo jih brali s senzorja.

2 Opis sistema

2.1 Oprema

Strežniški del seminarske naloge se izvaja na razvojnem sistemu *FRI-SMS* z *ARM9* procesorjem. Na tem sistemu je naložen operacijski sistem Linux z jedrom verzije 2.6.27. V to jedro smo dodali nov sistemski klic z kodo *361*, ki skupaj z našim modulom *ac.ko*, skrbita za branje temperature s senzorja.

2.1.1 I^2C

I^2C je vodilo, ki ga je razvil *Phillips* in se uporablja za priključitev nizko-hitrostnih perifernih naprav na matične plošče ali vgrajene (*embedded*) sisteme. Sestavljata ga dve vodili *Serial Data Line* in *Serial Clock*. I^2C vsebuje 7 bitni pomnilniški prostor, ki omogoča priključitev 112 vozlišč, ki komunicirajo preko istega vodila (ostalih 16 naslovov je rezerviranih). Vsako vozlišče ima lahko dve vlogi:

- *master* - vozlišče, ki oddaja uro in naslove *slave* vozlišč
- *slave* - vozlišče, ki prejema uro in naslove

Obstajajo štirje načini delovanja:

- *master transmit* - master vozlišče pošilja podatke slave vozlišču
- *master receive* - master vozlišče prejema podatke slave vozlišča
- *slave transmit* - slave vozlišče pošilja podatke master vozlišču
- *slave receive* - slave vozlišče prejema podatke master vozlišča

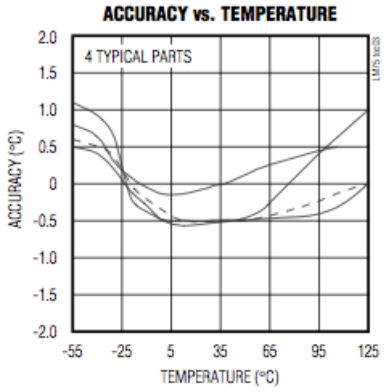
Definirani so tudi trije tipi sporočil, vsak izmed njih se začne s *START* in konča s *STOP*:

- Enojno sporočilo s katerim master piše podatke sužnju;
- Enojno sporočilo s katerim master bere podatke s sužnja
- Kombinirano sporočilo - master izvede vsaj dva branja ali pisanja na enega ali več sužnjev.

Preko I^2C vodila smo priključili naš temeraturni senzor *LM75* na ARM ploščico.

2.1.2 LM75

LM75 je senzor temperature in digitalni detektor prekoračenja temperature z I^2C vmesnikom, ki se zaradi majhne energijske zahtevnosti in vmesnika za I^2C pogosto uporablja za temperaturno upravljanje in varovanje pred napakami zaradi pregrevanja. Gostitelj lahko senzor vpraša za temperaturo kadarkoli, izhod za pregrevanje (OS) pa se vključi, ko zaznana temperature preseže neko vnaprej določeno vrednost. OS lahko deluje v prekinitvenem ali primerjalnem načinu in omogoča programatsko nastavljanje temperaturnega praga.



Slika 1: Natančnost LM75 senzorja v odvisnosti od temperature.

Senzor deluje pri temperaturah med $-55°C$ in $+125°C$, vendar temperatura vpliva na njegovo natančnost kot je razvidno iz Slike 1. Natančnost se v grobem giblje med -0.5 in +1 največja pa je pri $35°C$.

LM75 temperaturo sporoča v dvojiškem komplementu 9-ih bitov, ki se prenašajo v zgornjem in spodnjem bajtu. Biti D15-D7 predstavljajo podatek o temperaturi, kjer najmanj uteženi bit predstavlja $0.5°C$, najbolj uteženi bit pa predznak. Prvi se prenese najbolj uteženi bit, zadnjih 7 bitov spodnjega bajta pa je zanemarljivih.

Senzor nima lastne ure, saj deluje v podrejenem načinu in signal *clk* dobi preko I^2C vodila. V našem primeru smo za naslavljanje senzorja uporabili fiksen naslov *1001111*.

2.2 Server

Server oziroma temperaturni strežnik, deluje kot demon (proces ki se izvaja v ozadju). Vsako sekundo prebere temperaturo s senzorja in jo shrani v krožni pomnilnik. Med tem pa ena nit, ki jo strežnik ustvari ob zagonu, čaka na vhodne zahteve odjemalca na privzetih vratih 20000. Odjemalec lahko od strežnika zahteva eno izmed naslednjih treh stvari:

- `get_single_temp` - vzame temperaturo ki je na začetku krožnega pomnilnika.
- `get_last_temp` - vrne zadnjo prebrano temperaturo in ne vpliva na krožni pomnilnik.
- `get_temp_all` - vrne vse temperature in hkrati tudi izprazni krožni pomnilnik.

2.3 Client

Client je program s katerim uporabnik lahko prenaša trenutno temperaturo ali zgodovino temperatur in si jih ogleda. Za prenos prebranih temperaturnih meritev, mora uporabnik najprej vpisati IP naslov FRI-SMS sistema, na katerem je strežnik. V primeru, da strežnik ne uporablja privzetih vrat, mora uporabnik izpolniti tudi to polje, sicer ga lahko pusti praznega in se bodo uporabila privzeta vrata 20000. Ko uporabnik vpiše pravilne podatke strežnika, lahko nato izbere enega izmed zgoraj naštetih ukazov, ki jih podpira strežnik.

3 Celoten postopek

Da smo sistem usposobili, je bilo potrebno urediti precej stvari. V nadaljevanju je opisan postopek, po katerem smo prišli do delujočega sistema. Celotni postopek se lahko od uporabnika do uporabnika razlikuje, saj je odvisno kako ima pripravljeno delovno okolje.

3.1 Priprava okolja

Preden bomo lahko pripravili jedro, moramo na svojem računalniku pripraviti celotno okolje in namestiti vse potrebne programe:

- Namestiti moramo programe in knjižnice na primer *build-essential, svn, ncurses* in podobno. Če nam kak paket manjka nam bo javilo napako, ki jo nato odpravimo s pomočjo *apt-get* ukaza ali *Googla*, če nismo prepričani kateri paket nam manjka.
- Za make okolje potrebujemo *bash* in ne *dash*, ki je privzeta lupina na Ubuntu-ju. To lahko preverimo z *ls -l /bin/sh* in popravimo z ukazom *sudo ln -sf /bin/bash /bin/sh*
- Moramo imeti tudi celotno buildroot okolje, v katerem se nahaja *arm-linux-gcc*. To okolje dobimo na spletni strani *FRI-SMS*

  Buildroot okolje za prevajanje programja

  in ker imajo nakatere stvari absolutno pot vpisano, se mora ta mapa nahajati v */home/arm/* direktoriju.
- Pomaga pa tudi ukaz *sudo apt-get install good-luck* :)

3.2 Priprava jedra

1. S svn stežnika predmeta prenesemo linux-2.6.27-fri jedro

   ```
   1 cd /home/arm/
   2 svn checkout http://212.235.189.172/svn/miha.zidar/linux-2.6.27-fri
       linux-2.6.27-fri --username student
   3 cd linux-2.6.27-fri
   ```

2. V jedro dodamo popravek iz vaje7, ki naredi potrebne spremembe v linux jedru.

   ```
   1 patch -p0 < vaja7.diff
   ```

3. Posamezne module lahko vklopimo ali izklopimo z pomočjo menuja, kar je lako koristno če pri prevajanju dobimo kakšno napako.

   ```
   1 make menuconfig
   ```

4. Nato jedro zgradimo in naredimo sliko - če vam manjka kakšen paket, vas bo prevajalnik opozoril. Ko manjkajoče pakete naložite, še enkrat poženite isti ukaz.

   ```
   1 make CROSS_COMPILE=arm-linux- ARCH=arm uImage
   ```

Slika uImage se nahaja v *arch/arm/boot/uImage*.

3.3 Priprava jedernega modula

1. Prestavimo se v mapo kjer imamo ac.c in naredimo Makefile (ac.c smo predelali iz datoteke *predloga.c*)

```
1 echo "obj-m := ac.o" > Makefile
```

2. Zgradimo jedrni modul - pri tem se nahajamo v mapi */home/arm/gon* in imamo jedro v mapi */home/arm/linux-2.6.27-fri*

```
1 make CROSS_COMPILE=arm-linux- -C /home/arm/linux-2.6.27-fri M=/home/arm/
    gon modules
```

Sedaj imamo v tej mapi na voljo jedrni modul *ac.ko*.

3.4 Priprava TFTP streznika

1. Spodaj opisani postopek smo povzeli z naslova
http://www.unix.com/ubuntu/127020-configuring-ubuntu-9-04-tftp-server.html

Priprava TFTP streznika

2. Namestimo TFTP

```
1 sudo apt-get install xinetd tftpd tftp
```

3. Ustvarimo datoteko */etc/xinetd.d/tftp* in vanjo vpisemo

```
1  service tftp
2  {
3  protocol = udp
4  port = 69
5  socket_type = dgram
6  wait = yes
7  user = nobody
8  server = /usr/sbin/in.tftpd
9  server_args = /tftpboot
10 disable = no
11 }
```

4. Ustvarimo mapo /tftpboot

```
1 sudo mkdir /tftpboot
2 sudo chmod -R 777 /tftpboot
3 sudo chown -R nobody /tftpboot
```

5. Poženemo TFTP strežnik

```
1 sudo /etc/init.d/xinetd start
```

6. nato damo v mapo strežnika uImage datoteko.

```
1 sudo cp /home/arm/linux-2.6.27-fri/arch/arm/boot/uImage /tftpboot
2 sudo chmod -R 777 /tftpboot
3 sudo chown -R nobody /tftpboot
```

3.5 priprava FRI-SMS ploščice

1. FRI-SMS ploščico povežemo na računalnik preko serijskega porta s poljubnim programom

```
    naprava: /dev/ttyS0
    hitrost: 115200
    brez paritete
    8 bitov
    1 stop bit
    brez flow control
```

2. Vklopimo FRI-SMS ploščico, in prekinemo boot postopek (v treh sekundah od vklopa pritisnemo katerokoli tipko) in tako pridemo v uBoot meni.

3. V uBoot okolju nastavimo IP naslov ploščicne in strežnika ter MAC naslov naprave, kar potrebujemo za to, da bomo lahko s TFTP prenesli sliko novega jedra - za dodatne možnosti si oglejte *http://www.embedian.com/index.php?main_page=ubootev*

uBoot okolje

```
setenv ipaddr 192.168.1.2
setenv serverip 192.168.1.4
ethaddr 12:12:12:12:12:12
```

4. Ko smo prižgali ploščico, so se nam izpisali naslovi določenih stvari. Izpis zgleda približno tako:

```
Area 0: C0000000 to C00041FF (RO) Bootstrap
Area 1: C0004200 to V00083FF      Enviroment
Area 2: C0008400 to C0041FFF (RO) U-Boot
Area 3: C0042000 to C0251FFF      Kernel
Area 4: C0252000 to C041FFFF      FS
```

5. V tem primeru se jedro(kernel) nahaja na naslovu od C0042000 do C0251FFF. Sedaj lahko pobrišemo staro jedro. Pred naslov je potrebno dati 0x, saj gre za šestnajstiški zapis.

```
erase 0xC0042000 0xC0251FFF
```

6. Prenesemo uImage (novo linux jedro) s TFTP strežnika.

```
tftp 0x21000000 uImage
```

7. Sedaj pa prenesemo uImage iz RAMa v flash pomilnik. Tukaj predstavlja *0xabc*, velikost jedra, ki jo lahko preberemo iz izpisa ob koncu TFTP prenosa jedra na ploščico.

```
cp.b 0x21000000 0xC0042000 0xabc
```

8. Sedaj lahko zaženemo sistem z novim jedrom

```
boot
```

9. Prijavimo se v sistem (po možnosti kot root), prenesemo ac.ko na ploščico (na poljuben način) in sedaj le še dodamo jedrni modul ac.ko, da ga bo jedro lahko uporabilo.

```
ssh 192.168.1.4:ac.ko .
insmod ac.ko
```

   Če je bil modul uspešno naložen, se mora pokazati naslednja vrstica

```
ac 0-004f: hwmon0: sensor 'lm75'
```

Sedaj naj bi FRI-SMS ploščica znala pravilno servirati sistemski klic 361 (oz get_temp).

3.6 Server in Client

Za prevajanje Server-ja in Client-a nimamo zapletenega postopka, saj je vse že v *Makefile* datoteki. Edino kar je potrebno narediti pred ukazom make, je navesti, kje se nahaja arm-linux-gcc prevajalnik.

```
1 export PATH=/home/arm/buildroot-2009.11-fri/output/staging/usr/bin/:\$PATH
```

3.7 Vsi programi

Vsi programi, ki jih lahko ustvarimo z zgoraj opisanim postopkom, so že prevedeni v prilogi seminarski nalogi, v mapi *compiled*. Dodatno pa smo dodali še prevedeni strežnik ki ne uporablja sistemskega klica, ampak vrača kar naključna števila in je namenjen za testiranje. Strežnik pa je preveden tudi za Ubuntu Linux na 32-bitni arhitekturi, ter za ARM procesor.
Vsi prevedeni programi v prilogi so:

- *ac.ko*
- *uImage*
- *client-x86*
- *server-arm-syscall*
- *server-arm-random*
- *server-x86-syscall*
- *server-x86-random*

4 Izvorna koda

4.1 Server

Spodaj je izvorna koda za temperaturni strežnik, in Makefile za ARM arhitekturo. Za prevajanje *server.c* datoteke so potrebne še *defs.h, buf.h, buf.c* datoteke, ki so priložene seminarski nalogi.

Listing 1: Makefile

```
#export PATH=/home/arm/buildroot-2009.11-fri/output/staging/usr/bin/:$PATH
CC=arm-linux-gcc
#CFLAGS=-pthread -I. -DBUF_ELEMENT="struct cmd_t"
CFLAGS=-pthread -I. -DBUF_ELEMENT="struct cmd_t" --sysroot=/home/arm/
    buildroot-2009.11-fri/output/staging
SERVER_OBJ= server.o buf.o
#BUFFER_OBJ= KrozniVmesnik.o buf.o

%.o: %.c
        $(CC) -c -o $@ $< $(CFLAGS)

#buf: $(BUFFER_OBJ)
#       $(CC) $^ -o $@ $(CFLAGS)

server: $(SERVER_OBJ)
        $(CC) -o $@ $^ $(CFLAGS)

all: server

.PHONY: clean

clean:
        rm -f *.o *~ server
```

Listing 2: server.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <pthread.h>
#include "defs.h"
#include "buf.h"

void *server_listen();
static int cmd_exec(int fd, struct cmd_t cmd);
int s;
int lastTemp;

int get_temp(){
    //lastTemp = syscall(361);
    lastTemp = (int)(random()%120)+160;

    //printf("temp: %d \n",lastTemp);

    return lastTemp;
```

```
25 }
26
27 static int cmd_exec(int fd, struct cmd_t cmd){
28     FILE *f;
29     int temp;
30     int read_status;
31     f = fdopen(fd, "w");
32     if (f == NULL) return 1;
33
34     if(strcmp( cmd.name , "get_single_temp" ) == 0){
35         read_status = buf_get(&temp);
36         if (read_status == 1) fprintf(f,"napaka pri branju bufferja\n");
37         if (read_status == 2) fprintf(f,"buf je prazen\n");
38         if (!read_status) fprintf(f,"%d\n",temp);
39
40     }else if(strcmp( cmd.name , "get_last_temp" ) == 0){
41         fprintf(f,"%d\n",lastTemp); // last temp bi mogla biti
               sinhronizirana
42
43     }else if(strcmp( cmd.name , "get_temp_all" ) == 0){
44         read_status = buf_get(&temp);
45         if (read_status == 2) fprintf(f,"buf je prazen\n");
46         while (!read_status){
47             fprintf(f,"%d\n", temp);
48             read_status = buf_get(&temp);
49         }
50         if (read_status == 1) fprintf(f,"napaka pri branju bufferja\n");
51     }
52
53     fclose(f);
54     return 0;
55 }
56
57 static int setup_listen(short port){
58         int s, e, val;
59         struct sockaddr_in server_addr;
60
61         s = socket(PF_INET, SOCK_STREAM, 0);
62         if (s < 0) return -1;
63
64         val = 1;
65         e = setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (void *) &val, sizeof(
               val));
66         if (e) return -1;
67
68         memset((void *) &server_addr, 0, sizeof(server_addr));
69         server_addr.sin_family = AF_INET;
70         server_addr.sin_port = htons(port);
71         server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
72
73         e = bind(s, (struct sockaddr *) &server_addr, sizeof(server_addr));
74         if (e) return -1;
75
76         e = listen(s, 128);
77         if (e) return -1;
78
79         return s;
80 }
81
82 static void serve(int client){
83         int n;
```

```
84          struct cmd_t cmd;
85
86          n = read(client, (void *) &cmd, sizeof(cmd));
87          if (n != sizeof(cmd)) return;
88
89          cmd_exec(client, cmd);
90  }
91
92  static void run_server(int server){
93          int s, cl_len;
94          struct sockaddr_in client_addr;
95          cl_len = sizeof(client_addr);
96          s = accept(server, (struct sockaddr *) &client_addr, &cl_len);
97          if (s < 0) return;
98          serve(s);
99          close(s);
100 }
101
102
103 int main(int argc, char **argv)
104 {
105   int pth1;
106
107   pthread_t thread_serv_listen;
108   pthread_attr_t attr; // set of attributes for the thread
109   pthread_attr_init(&attr); // get the default attributes
110
111   s = setup_listen(SERVER_PORT);
112   if (s < 0) {
113     printf("error␣opening␣server\n");
114     exit(1);
115   }else{
116     printf("server␣started\n");
117   }
118
119   if( (pth1 = pthread_create( &thread_serv_listen, &attr, server_listen,
        NULL)) ){
120     printf("Server␣listen␣thread␣creation␣failed:␣%d\n", pth1);
121   }else{
122     printf("Server␣thread␣created\n");
123   }
124
125
126   while(1){
127 //    printf("while1 read temp\n");
128     usleep(1000000); // cakamo eno sekundo
129     buf_put((element_t)get_temp());
130   }
131   close(s);
132
133   return 0;
134 }
135
136 void *server_listen(){
137     while(1){
138         run_server(s);
139     }
140 }
```

4.2 Client

Tako kot za strežnik, tudi tukaj niso prikazane vse datoteke, ampak le tiste, ki so pomembne za razumevanje programa, ostale datoteke, ki so pomemebne za prevajanje pa se nahajajo v prilogi.

Listing 3: client.cpp

```
1  #include <sys/types.h>
2  #include <sys/socket.h>
3  #include <netinet/in.h> /* struct sockaddr_in, ... */
4  #include <arpa/inet.h>
5  #include <string.h>
6  #include <stdlib.h>
7  #include "defs.h"
8  #include "client.h"
9
10 Client::Client(QString addr, QString port, QString path, QString name,
       QString arg)
11 {
12     this->addr = addr;
13     this->port = port.toInt();
14     this->path = path;
15     this->name = name;
16     this->arg = arg;
17 }
18
19 int Client::server_connect(const char *ip, short port)
20 {
21     int s, e;
22     struct sockaddr_in server_addr;
23
24     s = socket(PF_INET, SOCK_STREAM, 0);
25     if (s < 0) return -1;
26
27     memset((void *) &server_addr, 0, sizeof(server_addr));
28
29     server_addr.sin_family = AF_INET;
30     server_addr.sin_port = htons(port);
31     server_addr.sin_addr.s_addr = inet_addr(ip);
32
33     e = ::connect(s, (struct sockaddr *) &server_addr, sizeof(server_addr));
34     if (e) return -1;
35
36     return s;
37 }
38
39 void Client::send_cmd(int s, cmd_t cmd)
40 {
41     char buf[81];
42     int n;
43
44     write(s, (void *) &cmd, sizeof(cmd));
45     while ((n = read(s, (void *) buf, sizeof(buf)-1)) > 0) {
46         buf[n] = '\0';
47         emit dataReceived(buf);
48         //qDebug(buf);
49     }
50 }
51
52 void Client::run()
53 {
54     //TODO: custom port
```

```
    int s;
    struct cmd_t cmd;
    int connPort = SERVER_PORT;
    if (port > 0){
        connPort = port;
    }

    strcpy(cmd.path, path.toLatin1().data());
    strcpy(cmd.name, name.toLatin1().data());
    strcpy(cmd.arg, arg.toLatin1().data());

    s = server_connect(addr.toLatin1().data(), connPort);
    if (s < 0) {
        emit connectError();
        return;
    }

    send_cmd(s, cmd);

    ::close(s);
}
```

Listing 4: mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "client.h"
#include <QMessageBox>
#include <QPainter>
#include <cstdlib>
#include <qbrush.h>


static int graphData[DATA_LENGTH];
static int sleepInterval = 1;
int myTimerID = -1;
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui(new Ui::
    MainWindow){
    ui->setupUi(this);
}

MainWindow::~MainWindow(){
    delete ui;
}

void MainWindow::DataReceived(QString text){
    int temp = text.toInt();
    if (myTimerID != -1){
        int bottomRange = 150;
        int topRange = 200; //razpon temperature od 15C do 35C
        int i;
        for (i=0 ; i < DATA_LENGTH-1 ; i++){
            graphData[i] = graphData[i+1];
        }
        graphData[DATA_LENGTH-1] = (int)(((float)(temp-bottomRange)/topRange
            )*ui->widget_5->height());
        if (graphData[DATA_LENGTH-1] < 0 ){
            graphData[DATA_LENGTH-1] = 0;
        }
        //TODO izpisi temperaturo .. k ne vem ce sm dobr normalizeru qDebug
            ("\n\n");
```

```
35      }
36      QStringList textsplit = text.split("\n");
37      foreach(QString str, textsplit) {
38          if(str.compare("")) {
39              ui->textEdit->setPlainText( QString::number(str.toFloat()*0.1) +
                    "\n" + ui->textEdit->toPlainText());
40          //ui->textEdit->setPlainText( str + "\n" + ui->textEdit->toPlainText
                ());
41          }
42      }
43
44  }
45
46  void MainWindow::ConnectError(){
47      QMessageBox::warning(this, "Napaka", "Napaka v povezavi.");
48      killTimer(myTimerID);
49      ui->getTempButton->setText("Draw Graph");
50      myTimerID = -1;
51  }
52
53
54  void MainWindow::on_getTempAllButton_clicked(){
55      //ui->textEdit->clear();
56      Client *client = new Client(ui->lineEditAddr->text(), ui->lineEditPort->
            text(), "" ,"get_temp_all", "" );
57      connect(client, SIGNAL(dataReceived(QString)), this, SLOT(DataReceived(
            QString)));
58      connect(client, SIGNAL(connectError()), this, SLOT(ConnectError()));
59      client->start();
60  }
61
62  void MainWindow::on_getTempButton_clicked(){
63      //TODO: make a trehad that calls this in intervals:
64      if (myTimerID == -1){
65          myTimerID=startTimer(1000*sleepInterval);
66          ui->getTempButton->setText("Stop Graph");
67      }else{
68          killTimer(myTimerID);
69          ui->getTempButton->setText("Draw Graph");
70          myTimerID = -1;
71      }
72  }
73
74  void MainWindow::on_getLastTempButton_clicked(){
75      //ui->textEdit->clear();
76      Client *client = new Client(ui->lineEditAddr->text(), ui->lineEditPort->
            text(), "" ,"get_last_temp", "" );
77      connect(client, SIGNAL(dataReceived(QString)), this, SLOT(DataReceived(
            QString)));
78      connect(client, SIGNAL(connectError()), this, SLOT(ConnectError()));
79      client->start();
80  }
81
82  void MainWindow::paintEvent(QPaintEvent*){
83    QPainter p(this);
84    QPainterPath pp;
85
86    int i;
87    int x = ui->widget_5->x()+ui->widget_4->x()+ui->widget_2->x();
88    int y = ui->widget_5->y()+ui->widget_4->y()+ui->widget_2->y()+15;
89    int width = ui->widget_5->width();
```

```
90    int height = ui->widget_5->height();
91    p.setRenderHint(QPainter::Antialiasing,true);
92    p.drawLine(x,y,x+width,y);
93    p.drawLine(x,y+height,x+width,y+height);
94    p.drawLine(x,y,x,y+height);
95    p.drawLine(x+width,y,x+width,y+height);
96
97    pp.moveTo(x,y+height);
98    pp.lineTo(x,y+height-graphData[0]);
99    for (i=1 ; i < DATA_LENGTH ; i++){
100       pp.lineTo(x+ (int)(((float)width/((float)DATA_LENGTH-1))*i),y+height-
              graphData[i]);
101   }
102   pp.lineTo(x+width,y+height-graphData[i]);
103   pp.lineTo(x+width,y+height);
104   p.fillPath(pp,QBrush("#c56c00"));
105 }
106
107 void MainWindow::on_clearButton_clicked(){
108     ui->textEdit->clear();
109 }
110
111 void MainWindow::timerEvent(QTimerEvent *event){
112     if (event->timerId()==myTimerID && myTimerID != -1){
113         this->on_getLastTempButton_clicked();
114         this->update();
115     }
116 }
```

4.3 Kernel module

Listing 5: Makefile

```
1 obj-m := ac.o
```

Listing 6: ac.c

```
1  /*
2      lm75.c - Part of lm_sensors, Linux kernel modules for hardware
3               monitoring
4      Copyright (c) 1998, 1999  Frodo Looijaard <frodol@dds.nl>
5
6      This program is free software; you can redistribute it and/or modify
7      it under the terms of the GNU General Public License as published by
8      the Free Software Foundation; either version 2 of the License, or
9      (at your option) any later version.
10
11     This program is distributed in the hope that it will be useful,
12     but WITHOUT ANY WARRANTY; without even the implied warranty of
13     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14     GNU General Public License for more details.
15
16     You should have received a copy of the GNU General Public License
17     along with this program; if not, write to the Free Software
18     Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
19 */
20
21 #include <linux/module.h>
22 #include <linux/init.h>
23 #include <linux/slab.h>
```

```
24 #include <linux/jiffies.h>
25 #include <linux/i2c.h>
26 #include <linux/hwmon.h>
27 #include <linux/hwmon-sysfs.h>
28 #include <linux/err.h>
29 #include <linux/mutex.h>
30 #include "ac.h"
31
32 extern int (*ac_gettemp)(void);
33
34 /*
35  * This driver handles the LM75 and compatible digital temperature sensors.
36  * Only types which are _not_ listed in I2C_CLIENT_INSMOD_*() need to be
37  * listed here.  We start at 9 since I2C_CLIENT_INSMOD_*() currently allow
38  * definition of up to 8 chip types (plus zero).
39  */
40
41 struct i2c_client *ac_client;
42
43 enum lm75_type {                /* keep sorted in alphabetical order */
44         ds1775 = 9,
45         ds75,
46         /* lm75 -- in I2C_CLIENT_INSMOD_1() */
47         lm75a,
48         max6625,
49         max6626,
50         mcp980x,
51         stds75,
52         tcn75,
53         tmp100,
54         tmp101,
55         tmp175,
56         tmp275,
57         tmp75,
58 };
59
60 /* Addresses scanned */
61 static const unsigned short normal_i2c[] = { 0x48, 0x49, 0x4a, 0x4b, 0x4c, 0
    x4d, 0x4e, 0x4f, I2C_CLIENT_END };
62
63 /* Insmod parameters */
64 I2C_CLIENT_INSMOD_1(lm75);
65
66
67 /* The LM75 registers */
68 #define LM75_REG_CONF           0x01
69 static const u8 LM75_REG_TEMP[3] = {
70         0x00,           /* input */
71         0x03,           /* max */
72         0x02,           /* hyst */
73 };
74
75 /* Each client has this additional data */
76 struct lm75_data {
77         struct device           *hwmon_dev;
78         struct mutex            update_lock;
79         u8                      orig_conf;
80         char                    valid;          /* !=0 if registers are
            valid */
81         unsigned long           last_updated;   /* In jiffies */
82         u16                     temp[3];        /* Register values,
```

```
                                                0 = input
                                                1 = max
                                                2 = hyst */
};

static int lm75_read_value(struct i2c_client *client, u8 reg);
static int lm75_write_value(struct i2c_client *client, u8 reg, u16 value);
static struct lm75_data *lm75_update_device(struct device *dev);


/*-----------------------------------------------------------------------*/

/* sysfs attributes for hwmon */

static ssize_t show_temp(struct device *dev, struct device_attribute *da,
                         char *buf)
{
        struct sensor_device_attribute *attr = to_sensor_dev_attr(da);
        struct lm75_data *data = lm75_update_device(dev);
        return sprintf(buf, "%d\n",
                       LM75_TEMP_FROM_REG(data->temp[attr->index]));
}

static ssize_t set_temp(struct device *dev, struct device_attribute *da,
                        const char *buf, size_t count)
{
        struct sensor_device_attribute *attr = to_sensor_dev_attr(da);
        struct i2c_client *client = to_i2c_client(dev);
        struct lm75_data *data = i2c_get_clientdata(client);
        int nr = attr->index;
        long temp = simple_strtol(buf, NULL, 10);

        mutex_lock(&data->update_lock);
        data->temp[nr] = LM75_TEMP_TO_REG(temp);
        lm75_write_value(client, LM75_REG_TEMP[nr], data->temp[nr]);
        mutex_unlock(&data->update_lock);
        return count;
}

static SENSOR_DEVICE_ATTR(temp1_max, S_IWUSR | S_IRUGO,
                          show_temp, set_temp, 1);
static SENSOR_DEVICE_ATTR(temp1_max_hyst, S_IWUSR | S_IRUGO,
                          show_temp, set_temp, 2);
static SENSOR_DEVICE_ATTR(temp1_input, S_IRUGO, show_temp, NULL, 0);

static struct attribute *lm75_attributes[] = {
        &sensor_dev_attr_temp1_input.dev_attr.attr,
        &sensor_dev_attr_temp1_max.dev_attr.attr,
        &sensor_dev_attr_temp1_max_hyst.dev_attr.attr,

        NULL
};

static const struct attribute_group lm75_group = {
        .attrs = lm75_attributes,
};

/*-----------------------------------------------------------------------*/

/* device probe and removal */

```

```
144 static int
145 lm75_probe(struct i2c_client *client, const struct i2c_device_id *id)
146 {
147         struct lm75_data *data;
148         int status;
149         u8 set_mask, clr_mask;
150         int new;
151 
152         if (!i2c_check_functionality(client->adapter,
153                         I2C_FUNC_SMBUS_BYTE_DATA | I2C_FUNC_SMBUS_WORD_DATA)
                            )
154                 return -EIO;
155 
156         data = kzalloc(sizeof(struct lm75_data), GFP_KERNEL);
157         if (!data)
158                 return -ENOMEM;
159 
160         i2c_set_clientdata(client, data);
161         mutex_init(&data->update_lock);
162 
163         /* Set to LM75 resolution (9 bits, 1/2 degree C) and range.
164          * Then tweak to be more precise when appropriate.
165          */
166         set_mask = 0;
167         clr_mask = (1 << 0)                     /* continuous conversions */
168                 | (1 << 6) | (1 << 5);          /* 9-bit mode */
169 
170         /* configure as specified */
171         status = lm75_read_value(client, LM75_REG_CONF);
172         if (status < 0) {
173                 dev_dbg(&client->dev, "Can't read config? %d\n", status);
174                 goto exit_free;
175         }
176         data->orig_conf = status;
177         new = status & ~clr_mask;
178         new |= set_mask;
179         if (status != new)
180                 lm75_write_value(client, LM75_REG_CONF, new);
181         dev_dbg(&client->dev, "Config %02x\n", new);
182 
183         /* Register sysfs hooks */
184         status = sysfs_create_group(&client->dev.kobj, &lm75_group);
185         if (status)
186                 goto exit_free;
187 
188         data->hwmon_dev = hwmon_device_register(&client->dev);
189         if (IS_ERR(data->hwmon_dev)) {
190                 status = PTR_ERR(data->hwmon_dev);
191                 goto exit_remove;
192         }
193 
194         dev_info(&client->dev, "%s: sensor '%s'\n",
195                 data->hwmon_dev->bus_id, client->name);
196 
197     ac_client = client;
198 
199         return 0;
200 
201 exit_remove:
202         sysfs_remove_group(&client->dev.kobj, &lm75_group);
203 exit_free:
```

```
204         i2c_set_clientdata(client, NULL);
205         kfree(data);
206         return status;
207 }
208
209 static int lm75_remove(struct i2c_client *client)
210 {
211         struct lm75_data *data = i2c_get_clientdata(client);
212
213         hwmon_device_unregister(data->hwmon_dev);
214         sysfs_remove_group(&client->dev.kobj, &lm75_group);
215         lm75_write_value(client, LM75_REG_CONF, data->orig_conf);
216         i2c_set_clientdata(client, NULL);
217         kfree(data);
218         return 0;
219 }
220
221 static const struct i2c_device_id lm75_ids[] = {
222         { "ds1775", ds1775, },
223         { "ds75", ds75, },
224         { "lm75", lm75, },
225         { "lm75a", lm75a, },
226         { "max6625", max6625, },
227         { "max6626", max6626, },
228         { "mcp980x", mcp980x, },
229         { "stds75", stds75, },
230         { "tcn75", tcn75, },
231         { "tmp100", tmp100, },
232         { "tmp101", tmp101, },
233         { "tmp175", tmp175, },
234         { "tmp275", tmp275, },
235         { "tmp75", tmp75, },
236         { /* LIST END */ }
237 };
238 MODULE_DEVICE_TABLE(i2c, lm75_ids);
239
240 /* Return 0 if detection is successful, -ENODEV otherwise */
241 static int lm75_detect(struct i2c_client *new_client, int kind,
242                        struct i2c_board_info *info)
243 {
244         struct i2c_adapter *adapter = new_client->adapter;
245         int i;
246
247         if (!i2c_check_functionality(adapter, I2C_FUNC_SMBUS_BYTE_DATA |
248                                      I2C_FUNC_SMBUS_WORD_DATA))
249                 return -ENODEV;
250
251         /* Now, we do the remaining detection. There is no identification-
252            dedicated register so we have to rely on several tricks:
253            unused bits, registers cycling over 8-address boundaries,
254            addresses 0x04-0x07 returning the last read value.
255            The cycling+unused addresses combination is not tested,
256            since it would significantly slow the detection down and would
257            hardly add any value. */
258         if (kind < 0) {
259                 int cur, conf, hyst, os;
260
261                 /* Unused addresses */
262                 cur = i2c_smbus_read_word_data(new_client, 0);
263                 conf = i2c_smbus_read_byte_data(new_client, 1);
264                 hyst = i2c_smbus_read_word_data(new_client, 2);
```

```
265                 if (i2c_smbus_read_word_data(new_client, 4) != hyst
266                  || i2c_smbus_read_word_data(new_client, 5) != hyst
267                  || i2c_smbus_read_word_data(new_client, 6) != hyst
268                  || i2c_smbus_read_word_data(new_client, 7) != hyst)
269                         return -ENODEV;
270                 os = i2c_smbus_read_word_data(new_client, 3);
271                 if (i2c_smbus_read_word_data(new_client, 4) != os
272                  || i2c_smbus_read_word_data(new_client, 5) != os
273                  || i2c_smbus_read_word_data(new_client, 6) != os
274                  || i2c_smbus_read_word_data(new_client, 7) != os)
275                         return -ENODEV;
276
277                 /* Unused bits */
278                 if (conf & 0xe0)
279                         return -ENODEV;
280
281                 /* Addresses cycling */
282                 for (i = 8; i < 0xff; i += 8)
283                         if (i2c_smbus_read_byte_data(new_client, i + 1) !=
                                conf
284                          || i2c_smbus_read_word_data(new_client, i + 2) !=
                                hyst
285                          || i2c_smbus_read_word_data(new_client, i + 3) !=
                                os)
286                                 return -ENODEV;
287         }
288
289         /* NOTE: we treat "force=..." and "force_lm75=..." the same.
290          * Only new-style driver binding distinguishes chip types.
291          */
292         strlcpy(info->type, "lm75", I2C_NAME_SIZE);
293
294         return 0;
295 }
296
297 static struct i2c_driver lm75_driver = {
298         .class          = I2C_CLASS_HWMON,
299         .driver = {
300                 .name   = "lm75",
301         },
302         .probe          = lm75_probe,
303         .remove         = lm75_remove,
304         .id_table       = lm75_ids,
305         .detect         = lm75_detect,
306         .address_data   = &addr_data,
307 };
308
309 /*-----------------------------------------------------------------------*/
310
311 /* register access */
312
313 /* All registers are word-sized, except for the configuration register.
314    LM75 uses a high-byte first convention, which is exactly opposite to
315    the SMBus standard. */
316 static int lm75_read_value(struct i2c_client *client, u8 reg)
317 {
318         int value;
319
320         if (reg == LM75_REG_CONF)
321                 return i2c_smbus_read_byte_data(client, reg);
322
```

```
323         value = i2c_smbus_read_word_data(client, reg);
324         return (value < 0) ? value : swab16(value);
325 }
326
327 static int lm75_write_value(struct i2c_client *client, u8 reg, u16 value)
328 {
329         if (reg == LM75_REG_CONF)
330                 return i2c_smbus_write_byte_data(client, reg, value);
331         else
332                 return i2c_smbus_write_word_data(client, reg, swab16(value))
                    ;
333 }
334
335 static struct lm75_data *lm75_update_device(struct device *dev)
336 {
337         struct i2c_client *client = to_i2c_client(dev);
338         struct lm75_data *data = i2c_get_clientdata(client);
339
340         mutex_lock(&data->update_lock);
341
342         if (time_after(jiffies, data->last_updated + HZ + HZ / 2)
343             || !data->valid) {
344                 int i;
345                 dev_dbg(&client->dev, "Starting␣lm75␣update\n");
346
347                 for (i = 0; i < ARRAY_SIZE(data->temp); i++) {
348                         int status;
349
350                         status = lm75_read_value(client, LM75_REG_TEMP[i]);
351                         if (status < 0)
352                                 dev_dbg(&client->dev, "reg␣%d,␣err␣%d\n",
353                                                 LM75_REG_TEMP[i], status);
354                         else
355                                 data->temp[i] = status;
356                 }
357                 data->last_updated = jiffies;
358                 data->valid = 1;
359         }
360
361         mutex_unlock(&data->update_lock);
362
363         return data;
364 }
365
366 /*-----------------------------------------------------------------------*/
367
368 /* module glue */
369
370
371 static int gettemp(void){
372   int value;
373
374   if (ac_client != NULL){
375     value = i2c_smbus_read_word_data(ac_client, 0x0); //ta funkcija je v
        jedru in naredi dostop do registra (ta register ima steviko 0) v
        katerem je trenutna temperatura
376     return (swab16(value) /128 * 5); //čaritmetini sift levo za 7 bitov ... 5
         je čnatannost za pol stopnije k smo tko rekl
377     //return 12;//(swab16(value) /128 * 5); //čaritmetini sift levo za 7
        bitov ... 5 je čnatannost za pol stopnije k smo tko rekl
378   }else{
```

```
379     return -1;
380   }
381 }
382
383 static int ac_init(void){
384   ac_gettemp = gettemp;
385   return i2c_add_driver(&lm75_driver); //ta vrstica registrira nas gonilnik
        znotraj i2c infrastrukture
386 }
387
388 static void ac_exit(void){
389   ac_gettemp = NULL;
390   i2c_del_driver(&lm75_driver);
391 }
392
393
394 MODULE_AUTHOR("Zidar Miha");
395 MODULE_DESCRIPTION("LM75 driver");
396 MODULE_LICENSE("GPL");
397
398 module_init(ac_init)
399 module_exit(ac_exit)
```

4.4 Syscall Patch file

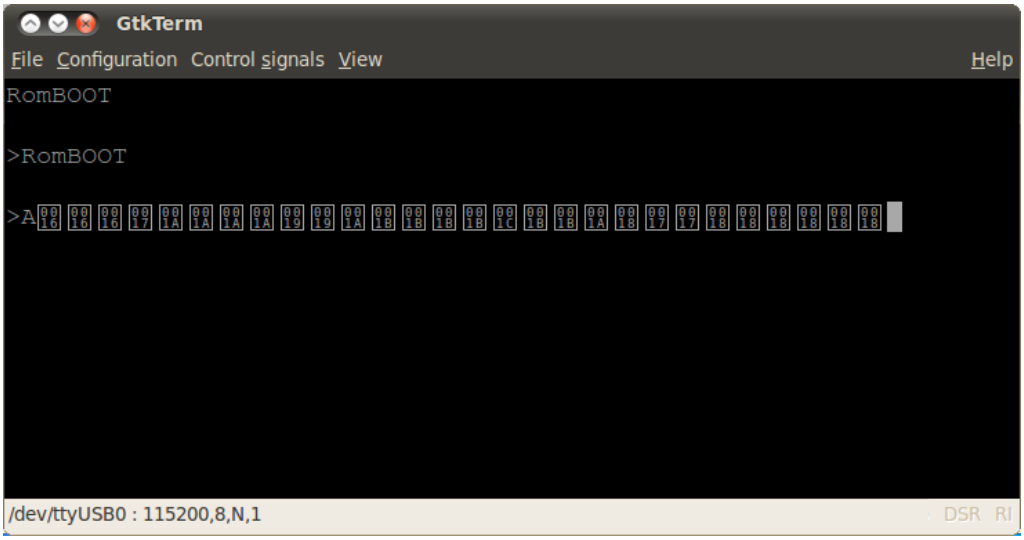
Listing 7: SysPatch.diff

```
1  Index: Makefile
2  ===================================================================
3  --- Makefile    (revision 2804)
4  +++ Makefile    (working copy)
5  @@ -620,7 +620,7 @@
6
7
8   ifeq ($(KBUILD_EXTMOD),)
9  -core-y          += kernel/ mm/ fs/ ipc/ security/ crypto/ block/
10 +core-y          += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ ac/
11
12  vmlinux-dirs    := $(patsubst %/,%,$(filter %/, $(init-y) $(init-m) \
13                       $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
14 Index: arch/arm/include/asm/unistd.h
15 ===================================================================
16 --- arch/arm/include/asm/unistd.h       (revision 2804)
17 +++ arch/arm/include/asm/unistd.h       (working copy)
18 @@ -388,6 +388,8 @@
19  #define __NR_pipe2                      (__NR_SYSCALL_BASE+359)
20  #define __NR_inotify_init1              (__NR_SYSCALL_BASE+360)
21
22 +#define __NR_gettemp                     (__NR_SYSCALL_BASE+361)
23 +
24  /*
25   * The following SWIs are ARM private.
26   */
27 Index: arch/arm/kernel/calls.S
28 ===================================================================
29 --- arch/arm/kernel/calls.S     (revision 2804)
30 +++ arch/arm/kernel/calls.S     (working copy)
31 @@ -370,6 +370,7 @@
32                 CALL(sys_dup3)
33                 CALL(sys_pipe2)
```

```
 /* 360 */      CALL(sys_inotify_init1)
+               CALL(sys_gettemp)
 #ifndef syscalls_counted
 .equ syscalls_padding, ((NR_syscalls + 3) & ~3) - NR_syscalls
 #define syscalls_counted
Index: ac/ac.c
===================================================================
--- ac/ac.c     (revision 0)
+++ ac/ac.c     (revision 0)
@@ -0,0 +1,12 @@
+#include <linux/linkage.h>
+#include <linux/module.h>
+
+int (*ac_gettemp)(void);
+
+EXPORT_SYMBOL(ac_gettemp);
+
+asmlinkage int sys_gettemp(void)
+{
+  return ac_gettemp();
+}
+
Index: ac/Makefile
===================================================================
--- ac/Makefile (revision 0)
+++ ac/Makefile (revision 0)
@@ -0,0 +1 @@
+obj-y := ac.o
```

4.5 Asembler testni program

Program ki inicializira registre za komunikacijo s serijskimi vrati in za vodilo I^2C. Podatki ki se prikažejo v Com terminalu so surovi podatki, ki ji dobimo z senzorja $LM75$ in so oblike $t_{15}t_{14}\cdots t_8t_7n_6\cdots n_0$. Biti t_w predstavljajo prebrano temperaturo v formatu ki jo poda $LM75$, biti n_z pa predstavljajo naslov naprave.



Slika 2: Primer izhoda testnega programa.

Listing 8: lm75.asm

```
.text
.code 32

.global _start
_start:

/* select irq mode */
  mrs r0, cpsr
  bic r0, r0, #0x1F   /* clear mode flags */
  orr r0, r0, #0xD2   /* set irq mode + DISABLE IRQ, FIQ*/
  msr cpsr, r0

/* init irq stack */
  ldr sp,_Lirqstack_end

/* select System mode
  CPSR[4:0]     Mode
  --------------
   10000          User
   10001          FIQ
   10010          IRQ
   10011          SVC
   10111          Abort
   11011          Undef
   11111          System
*/

  mrs r0, cpsr
  bic r0, r0, #0x1F   /* clear mode flags */
```

```
30    orr r0, r0, #0xDF   /* set system mode + DISABLE IRQ, FIQ*/
31    msr cpsr, r0
32
33    /* init stack */
34    ldr sp,_Lstack_end
35
36    /* setup system clocks */
37    bl clk_init
38
39    /* enable I cache */
40    bl enable_I_cache
41
42  .global _main
43  /* main program */
44  _main:
45  /* user code */
46
47  .equ TWI_BASE, 0xFFFAC000
48  .equ PMC_BASE, 0xFFFFFC00  /* (PMC) Base Address */
49
50  .equ PMC_PCER, 0x10
51  .equ TWI_CR, 0x00
52  .equ TWI_MMR, 0x04
53  .equ TWI_RHR, 0x30
54  .equ TWI_SR, 0x20
55  .equ TWI_CWGR, 0x10
56
57  .equ PIOA_BASE, 0xFFFFF400
58  .equ PIO_PDR, 0x04
59  .equ PIO_MDER, 0x50
60  .equ PIO_ASR, 0x70
61
62
63  .equ PIOC_BASE, 0xFFFFF800
64  .equ PIO_SODR, 0x30
65  .equ PIO_CODR, 0x34
66
67  .equ DBGU_BASE, 0xFFFFF200
68  .equ DBGU_CR,   0x0000
69  .equ DBGU_MR,   0x0004
70  .equ DBGU_SR,   0x0014
71  .equ DBGU_RHR,  0x0018
72  .equ DBGU_THR,  0x001C
73  .equ DBGU_BRGR, 0x0020
74
75
76    ldr r0, =DBGU_BASE
77    mov r1, #0b1010000            /*mov r1, 0x101011100*/
78    str r1, [r0, #DBGU_CR]
79    mov r1, #0x800
80    str r1, [r0, #DBGU_MR]
81    mov r1, #0x1A
82    str r1, [r0, #DBGU_BRGR]
83
84
85    mov r4, #0x41
86    bl debug_transmit
87
88
89    ldr r0, =PMC_BASE
90    mov r1,#1 << 11
```

```
91    str r1,[r0,#PMC_PCER]
92
93    ldr r0, =PIOA_BASE
94    mov r1,#0b11 << 23
95    str r1,[r0,#PIO_PDR]
96    str r1,[r0,#PIO_MDER]
97    str r1,[r0,#PIO_ASR]
98
99    ldr r0,=TWI_BASE
100   ldr r1,=0b11000000010000000
101   str r1,[r0,#TWI_CWGR]
102   mov r1,#0b100100
103   str r1,[r0,#TWI_CR]
104
105   mov r1,#0b1001111 << 16
106   orr r1,r1,#1 << 12
107   str r1,[r0,#TWI_MMR]
108
109 rep:
110   mov r1,#1
111   str r1,[r0,#TWI_CR]
112
113 wait1:
114   ldr r1,[r0,#TWI_SR]
115   ands r1,r1,#2
116   beq wait1
117
118   ldr r1,[r0,#TWI_RHR]
119   add r4,r1,#0
120   bl debug_transmit
121   mov r4, #0x20
122   bl debug_transmit
123   strb r1,hi
124
125
126   mov r1,#2
127   str r1,[r0,#TWI_CR]
128
129 wait2:
130   ldr r1,[r0,#TWI_SR]
131   ands r1,r1,#2
132   beq wait2
133
134   ldr r1,[r0,#TWI_RHR]
135   add r4,r1,#0
136   bl debug_transmit
137   mov r4, #0x20
138   bl debug_transmit
139   strb r1,lo
140
141
142 wait3:
143   ldr r1,[r0,#TWI_SR]
144   ands r1,r1,#1
145   beq wait3
146
147   ldr r1,=100000000
148 dly:
149   subs r1,r1,#1
150   bne dly
151
```

```
  mov r4, #0x20
  bl debug_transmit
  mov r4, #0x20
  bl debug_transmit
  mov r4, #0x20
  bl debug_transmit
  mov r4, #0x20
  bl debug_transmit

  b rep

  mov r0,#0xA3
  bl  POSLJI
  b _wait_for_ever

debug_transmit:
  stmfd sp!, {r1 - r3 , lr}
  ldr r1, =DBGU_BASE
zanka_debug_transmit:
  ldr r2, [r1, #DBGU_SR]
  ands r2, r2, #2
  beq zanka_debug_transmit
  str r4, [r1, #DBGU_THR]
  ldmfd sp!, {r1 - r3 , pc}


POSLJI: stmfd   sp!,{r1-r3,lr}
                ldr             r1,=PIOC_BASE
                mov             r2,#8
                mov             r3,#1
ZANKA:  str             r3,[r1,#PIO_SODR]
                bl              DELAY
                str             r3,[r1,#PIO_CODR]
                bl              DELAY
                tst             r0, #0x80
                blne            DELAY
                mov             r0,r0,lsl #1
                subs            r2,r2,#1
                bne             ZANKA
                ldmfd   sp!,{r1-r3,pc}

DELAY:  stmfd sp!,{lr}
        ldmfd sp!,{pc}

/* end user code */

_wait_for_ever:
  b _wait_for_ever


/* variables here */

hi:   .space 1
lo:   .space 1
TABELA: .word 1,2,3,4,5,6,7,8,9,10
MIN: .space 4
MAX: .space 4

/* end variables */

  .align 2
```

```
213 _Lstack_end:
214   .long __STACK_END__- 2*13*4  @ space for 26 registers on IRQ stack
215 _Lirqstack_end:
216   .long __STACK_END__
217
218 .end
```