Jabber Client:	Points:	____ / 3
XMPP handling:	Points:	____ / 3
VS1011e Driver:	Points:	____ / 3
Volume control:	Points:	____ / 2
PS/2 Interface:	Points:	____ / 2
Ping + Port unreachable:	Points:	____ / 1
TinyOS programming:	Points:	____ / 1
Theory Tasks:	Points:	____ / 6
Overall:	Points:	____ / 21

Microcontroller VU

Application Protocol

Danny Milosavljevic, MatrNr. 0826039
e0826039@student.tuwien.ac.at

Tutor: Michael Spiegel

June 26, 2014

Declaration of Academic Honesty

I hereby declare that this protocol (text and code) is my own original work and that I have completed this work using only the sources cited in the text.

(Date) (Signature of Student)

Admission to Publish

□ I explicitly **allow** the publication of my solution (protocol and sourcecode) on the course webpage.

□ I **do not allow** the publication of my solution (default if nothing is checked).

(Signature of Student)

Contents

1 Overview

1.1 Connections, External Pullups/Pulldowns

See http://ti.tuwien.ac.at/ecs/teaching/courses/mclu/exercises/spec-2.pdf/at_download/file.

In addition: P5 ADC input to PF2.

1.2 Installation

make JABBER_USERNAME=xxx bigAVR6_1280 install

1.3 Design Decisions

- Keyboard mapping is not configurable.
- Subscription response is assumed to be reliable. If our response to a subscription request gets lost, too bad.
- 4 buddies visible max (when a new buddy sends a message, it drops the buddy which hadn't sent/received a message for the longest time to make room for the new buddy, then inserts the new buddy).
- 80 Byte received message max
- 31 Byte jid max
- 20 slot keyboard buffer (one slot is 16 bit)
- 31 Byte max entry length
- Touchscreen calibrated to (0, -10).
- Jabber Username is determined at compile time and then hardcoded into the executable (alternative would be: leave ”from” off the message and just send it like that).
- When you send a message to someone, it acknowledges the receiver's subscription request, if any.

1.4 Specialities

Does your solution have something special (positive or negative)?

Line Editing with Cursor and Insert and Delete keys.

2 Main Application

The Application is a Jabber client.

After boot, it logs into the Jabber server at 10.60.0.1 port udp/5222 (see Application/udp_config.h). Then it updates the (Jabber) status to ”available”. When the server sends the roster, it is shown in the active buddies list at the top of the screen (you may need to restart the proxy beforehand for the server to actually send the roster).

Once it receives a message, it makes sure the sender is in the active buddies list at the top of the screen (if the list is full, it replaces the buddy which was sending or receiving messages the least). If the sender is not selected, it displays the symbol M in front of the buddy in order to signal to you that this buddy sent a message.

When you click on the buddy in the active buddies list at the top of the screen, it shows the (last) message it received from this buddy, and, if applicable, whether the buddy wants to subscribe to your presence.

If the buddy is not available, the entry is striken out.

The 2 line LCD and a PS/2 keyboard can be used to send a message to the currently selected buddy. Type the message and press RETURN. The message is immediately sent to the buddy.

3 JabberClientC

This is the main module. It contains: Text entry box including caret handling and input handling, active buddies list handling including TouchScreen. When a message is received, it plays a sound. If the buddy that sent the message isn’t currently selected, it marks it with a M icon. When the buddy is selected, the message received from him is displayed on the graphical LCD.

4 HplPS2C

Handles the keyboard input via PS/2 at PORTD. Only provides an async event receivedCode(code) for the raw data received.

Used http://www.computer-engineering.org/ps2keyboard/ as reference.

5 KeyboardC

High-level keyboard handler. Provides key to ASCII mapping and buffering. Will actually signal events with the ASCII code, if possible.

6 VolumeAdcC

Handles P5 potentiometer.

7 HplVS1011eC

Low-level driver for the VS1011e (MP3). Only can process 32 Byte at once and access the registers.

8 MessageManagerC

Handles Jabber messages and provides the active buddies list.

9 VS1011eC

High-level sample-based MP3 handler. Can only play samples from PROGMEM. Can set the volume - with automatically linearized loudness.

9.1 Controlling Loudness

The linear Loudness scale for the music volume is approximated like this:

$$\begin{aligned} N &= 1 - (1 - a)^4 \\ 0 \leq a \leq 1 \Rightarrow 0 &\leq N \leq 1 \end{aligned}$$

In order to get all tempoerary results to fit into 16 bits (as requested), I first made the following choices:

The new $\tilde{a}$ is supposed to be in range [0, 255], limited to integers only. The new $\tilde{N}$ is supposed to be in range [0, 255], limited to integers only.

So I set

$$\begin{aligned} \tilde{a} &= 255a \\ \tilde{N} &= 255N \end{aligned}$$

And substitute into the equation to get:

$$\begin{aligned}
N &= 1 - (1-a)^4 \\
\tilde{N} &= 255 - 255(1 - \frac{\tilde{a}}{255})^4 \\
255 &= \sqrt[4]{255}^4 \\
\tilde{N} &= 255 - \left(\sqrt[4]{255} - \sqrt[4]{255}\frac{\tilde{a}}{255}\right)^4 \\
\sqrt[4]{255} &\approx 4 \\
\frac{255}{\sqrt[4]{255}} &\approx 64 \\
\tilde{N} &\approx 255 - \left(4 - \frac{\tilde{a}}{64}\right)^4 \\
\tilde{N} &\approx 255 - \left(\frac{256 - \tilde{a}}{64}\right)^4 \\
\tilde{N} &\approx 255 - \left(\frac{255 - \tilde{a}}{64}\right)^4 \\
\tilde{N} &\approx 255 - \left(\frac{(255 - \tilde{a})^2}{64^2}\right)^2 \\
0 \leq \tilde{a} \leq 255 \Rightarrow 0 &\leq (255 - \tilde{a}) \leq 255 \\
(255 - \tilde{a}) \leq 255 \Rightarrow (255 - \tilde{a})^2 &\leq 255^2 < 2^{16} \\
64^2 = (2^6)^2 &= 2^{12} < 2^{16}
\end{aligned}$$

In order to only use integer arithmetic, I use instead

$$\tilde{N} \approx 255 - \left\lfloor \frac{(255 - \tilde{a})^2}{64^2} \right\rfloor^2$$

which consistently overshoots:

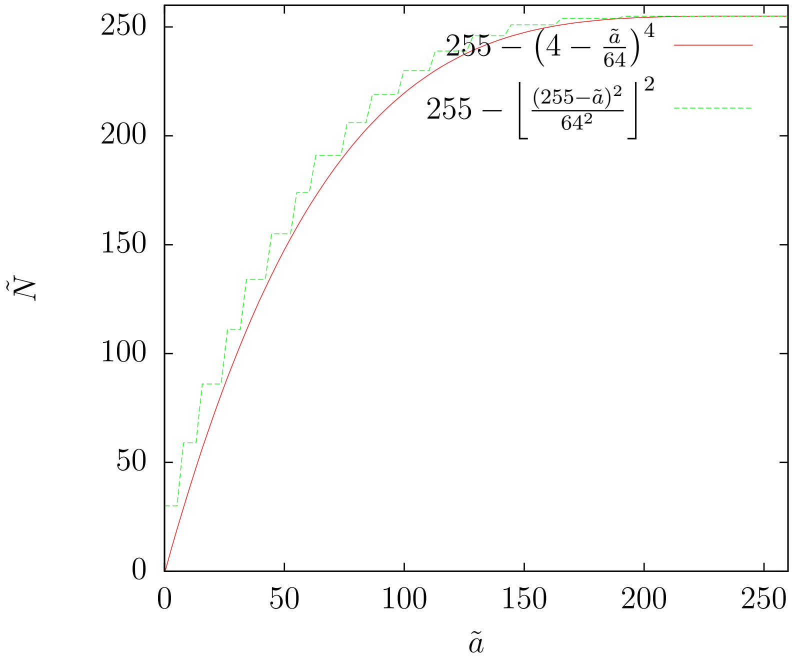
Therefore, I use a manual correction at the lower end in order to be able to reach $\tilde{N} = 0$.

10 Problems

No problems.

11 Work

Estimate the work you put into solving the Application.



reading manuals, datasheets	20 h
program design	10 h
programming	20 h
debugging	15 h
questions, protocol	15 h
Total	80 h

12 Theory Tasks

Please work out the theory tasks very carefully, as there are very limited points to gain!

Assume that you are part of a group of n people. Every few weeks you and your peers try to organize a get-together for the group. Assume that everyone has a different schedule and thus their own unique desired date and time. Sadly, the communication medium you use is unreliable. Every time you send a message to the group it is possibly received only by a subset of the group, in the worst case the message is lost and thus received by no one. Luckily, everyone has ample free-time, thus as soon as someone receives a message with an unknown date, it gets marked in his/her calendar. Everyone in the group visits the local pub on every date marked in his/her calendar.

The goal of this task is to find the point in time where enough system-wide communication occurred to guarantee that at least at one date all peers of the group are present.

12.1 Communication protocol

To simplify the task, communication is divided into several subsequent rounds. Starting with round $r = 1$, in each round every peer $p_i,\ i \in \{1, .., n\}$, has the ability to send a message to the group. The round r message sending happens in zero time and is synchronous, i.e., all peers send and receive messages at the exact same time. Assume that the communication starts early enough, such that at least n^2 communication rounds take place before the earliest date proposed by any one of the peers. Recall that even though messages are sent via a broadcast, each message may be received only by a (possible empty) subset of the group. After each round, every peer p_i marks the received dates $d_j,\ j \in \{1, .., n\}$, on his/her calendar c_i.

To counteract the unreliable communication system, every peer sends all dates marked on his/her calendar (every peer starts out with one unique date prior to round 1) to the group members. Moreover, for every message sent by p_i and received by p_j ($i \neq j$) in some round r, the sender p_i reliably receives, in the same round, a response message from p_j with all dates know to p_j up to and including round $r - 1$.

Furthermore, despite the unreliable nature of the protocol, it is guaranteed that for every round r, the graph G^r, that results from representing every peer as a node and every message sent from p_i to p_j in round r as an edge $(p_i \rightarrow p_j)$, is connected.

1. [3 Points] Task 1:
Proof rigorously that after n rounds there exists a date at which all peers of the group are present.

Every node, by using of the path (which is then guaranteed to exist between any nodes), can ”slowly” transmit its calendar to any other node, until every node has the same calendar.

Say node A manages to transfer its calendar to its neighbour node B. Then node B will reliably answer with its calendar so far, which node A will incorporate into its calendar.

If node A manages to transfer its calendar to its neighbour node C, then node C will reliably answer with its calendar so far, which node A will incorporate into its calendar.

In this way node A will end up with the accumulated calendar of node B and node C so far (although node B and node C are still behind).

This took one round.

Each of the n nodes does this at the same time, increasing their calendar size (or at least keeping it the same) and progressing its dates through the network (”slowly”) forward. (Note: because of the connectivity, that also means that A must have received a broadcast as well, by definition. So in this sense, A is also ”forward”)

Specifically, look at one node A. In one round, it will have incorporated the ($r-1$-round) calendars of all its forward neighbours. It might still miss entries of farther successors.

However, each farhter successor also incorporated the ($r-1$-round) calendars of all its forward neighbours. This might still miss ($r-1$-round) entries of farther successors. etcetc.

In the end, one farthest successor is the node A, by definition of connectivity. Its (r − 1-round) entries will also be incorporated into its ”back” neighbour - and it will also receive the new entries of the second-farthest successor.

For each node after each round, either it already knew all dates or it got at least one new date (from another node, which maybe got it from another node etcetc).

After n rounds, it thus got n-1 new (unique) dates (and one already-known date). These were all elements of the initial set. Therefore, it is the initial set.

2. [3 Points] Task 2:
Assume that the guaranteed response messages system fails (this is a frequent problem in unreliable communication scenarios, such as wireless sensor networks) and instead the communication graph is weakly connected with exactly one root R^r. A root R^r of round r is a node that does not receive any message in round r. Proof rigorously that after n^2 rounds, also in this scenario, it is guaranteed that there exists a date at which all peers of the group are present.

Note: The proof whether this holds (or not) in the case of less than n^2 rounds is still an open research question.

There are n different dates. There are n nodes.

In each round, there is either progress or all nodes already know a common date. Proof:

Per round, there is only no global progress when all successors already know each date. Otherwise, if one node receives the date, it will pass it on until all others know the date - actively tells it forward if possible. In the next round it passes it backward, if asked.

Because the graph is weakly connected, each node is asked (for $r - 1$-round dates) or tells forward or both.

Thus after one round, another node B knows this node’s dates.

In the next round, node B is either asked or tells.

So another node C knows node B’s dates.

Each time there’s a forward arrow in a path, another (end) node gets to know this (beginning) node’s calendar of round r and tells the beginning node its calendar, as in Task 1. Each time there’s a backward arrow in a path, another (beginning) node gets to know this (end) node’s calendar of round r and tells the end node its calendar, as in Task 1.

A Listings

Include EVERY source file of your Application (including headers)!!! And EVERY file you have modified!

A.1 Main

Listing 1: Application/prepare

```
#!/bin/sh
set −e
p=”‘pwd‘”
cd /homes/m0826039/Sources/tinyos/tinyos_ss14
source tinyOSExports
cd ”${p}”
```

Listing 2: Application/compile

```
#!/bin/bash
exec make ”$@” bigAVR6_1280 install
```

Listing 3: Application/Makefile

```
COMPONENT=JabberClientAppC

PFLAGS += −I${TOSDIR}/chips_ecs/atm1280/spi −I${TOSDIR}/chips_ecs/enc28j60

include $(MAKERULES)

ifndef JABBER_USERNAME
JABBER_USERNAME = $(shell id −un)
endif

PFLAGS += −DJABBER_USERNAME=’”$(JABBER_USERNAME)”’
#CFLAGS += −DDEBUG
```

Listing 4: Application/udp_config.h

```
#ifndef UDP_CONFIG_H
#define UDP_CONFIG_H

// The local udp jabber port to use
#define UDP_LOCAL_PORT  5222UL
// The udp port of the jabber gateway
#define UDP_REMOTE_PORT 5222UL

// note the ’,’ (instead of the usual ’.’) between numbers
#define IP_JABBER_GW     10,60,0,1

// If this macro is defined custom settings will be applied
#define CUSTOM_IP_SETTINGS

// the following settings are only applied if CUSTOM_IP_SETTINGS is defined
// note the ’,’ (instead of the usual ’.’) between numbers
#define IP_LOCAL 10,60,0,10
#define NETMASK 255,255,255,0
#define GATEWAY 10,60,0,1

#ifndef JABBER_USERNAME /* see Makefile */
#define JABBER_USERNAME ”m0826039”
#endif

#endif
```

Listing 5: Application/JabberClientAppC.nc

```
#include <Atm128Uart.h>
#include "udp_config.h"
#include "debug.h"
configuration JabberClientAppC
{
}
#define MAX_KEYBOARD_BUFFER 20
implementation
{
  components StdoDebugC;
  components MainC, JabberClientC;
  components LedsC;
  components BufferedLcdC, HplKS0108C, TouchScreenC;
  components Atm1280SpiC;
  components KeyboardC, HplPS2C;
  components HplAtm128GeneralIOC, HplAtm128InterruptC;
  components HplVS1011eC, VS1011eC;
  components MessageManagerC, LlcTransceiverC, IpTransceiverC, /*PingC, */new UdpC(UDP_LOCAL_PORT);
#ifndef WLAN
  components Enc28j60C as EthernetC;
#else
  components Mrf24wC as EthernetC;
#endif
  components new TimerMilliC() as TouchTimer;
  components new TimerMilliC() as CaretBlinkTimer;
  components PlatformSerialC;
  components VolumeAdcC;
  components new AdcReadClientC();
  components BusyWaitMicroC;
  JabberClientC.VolumeReading -> VolumeAdcC;
  VolumeAdcC.Read2 -> AdcReadClientC;
  /* Read<uint16_t> = AdcReadClientC; */
  AdcReadClientC.Atm128AdcConfig -> VolumeAdcC; /* used by AdcReadClientC */
  AdcReadClientC.ResourceConfigure -> VolumeAdcC; /* used by AdcReadClientC */

  /* MAIN */

  JabberClientC.TouchTimer -> TouchTimer;
  JabberClientC.CaretBlinkTimer -> CaretBlinkTimer;
  JabberClientC.Leds -> LedsC;
  JabberClientC.LCD2 -> BufferedLcdC;
  JabberClientC.GLCD -> TouchScreenC.Glcd;
  JabberClientC.Keyboard -> KeyboardC;
  JabberClientC.TouchScreen -> TouchScreenC.TouchScreen;
  JabberClientC.MessageManager -> MessageManagerC;
  JabberClientC.MP3 -> VS1011eC;
  JabberClientC.Boot -> MainC.Boot;

  /* KEYBOARD */

  components new AsyncQueueC(uint16_t, MAX_KEYBOARD_BUFFER) as KeyboardBuffer;
  KeyboardC.Buffer -> KeyboardBuffer;
  KeyboardC.HplPS2 -> HplPS2C;
  HplPS2C.DataPort -> HplAtm128GeneralIOC.PortD4;
  HplPS2C.ClockPort -> HplAtm128GeneralIOC.PortD1;
  HplPS2C.ClockInterrupt -> HplAtm128InterruptC.Int1; /* PinD1 */
  MainC.SoftwareInit -> HplPS2C.Init;
  MainC.SoftwareInit -> KeyboardC.Init;

  /* MP3 */

  HplVS1011eC.RSTPort -> HplAtm128GeneralIOC.PortF7; /* OK */
  HplVS1011eC.CSPort -> HplAtm128GeneralIOC.PortF6; /* OK */
  HplVS1011eC.DREQPort -> HplAtm128GeneralIOC.PortD2;  /* OK */
  HplVS1011eC.BSYNCPort -> HplAtm128GeneralIOC.PortF3; /* OK */
  HplVS1011eC.DREQInterrupt -> HplAtm128InterruptC.Int2; /* OK, PortD2 */
  HplVS1011eC.DataPort -> Atm1280SpiC.SpiByte;
  HplVS1011eC.DataControl -> Atm1280SpiC.SpiControl;
  HplVS1011eC.DataPacket -> Atm1280SpiC.SpiPacket;
  HplVS1011eC.Resource -> Atm1280SpiC.Resource[unique("Atm128SpiC.Resource")];
  HplVS1011eC.ResetWaiter -> BusyWaitMicroC;
  VS1011eC.HplVS1011e -> HplVS1011eC;
  MainC.SoftwareInit -> HplVS1011eC.Init;

  /* UDP */

  MessageManagerC.LCD2 -> BufferedLcdC;
  MessageManagerC.UdpSend -> UdpC;
  MessageManagerC.UdpReceive -> UdpC;
  MessageManagerC.Control -> EthernetC;
  LlcTransceiverC.Mac -> EthernetC;
  MessageManagerC.IpControl -> IpTransceiverC;
#ifdef WLAN
  MessageManagerC.WlanControl -> EthernetC;
#endif
  MessageManagerC.Boot -> MainC.Boot;

}
```

Listing 6: Application/JabberClientC.nc

```
#include <avr/pgmspace.h>
#include <printf.h>
#include "Timer.h"
#include "Keyboard.h"
#include "buddy.h"
#include "sounds.h"
#include "debug.h"

#define MAX_2LINE_DISPLAY_WIDTH 16
#define MAX_ENTRY_SIZE 32
#define ICON_WIDTH 8

module JabberClientC @safe() {
        uses interface Timer<TMilli> as TouchTimer;
        uses interface Timer<TMilli> as CaretBlinkTimer;
        uses interface Leds;
        uses interface Boot;
        uses interface BufferedLcd as LCD2;
        uses interface Glcd as GLCD;
        uses interface Keyboard as Keyboard;
        uses interface TouchScreen;
        uses interface MessageManager;
        uses interface Read<uint16_t> as VolumeReading;
        uses interface MP3;
}
implementation {
        uint8_t entryLine, entryCol, entryPos;
        char entry[MAX_ENTRY_SIZE];
        bool entryCaretVisible = TRUE; /* at this instant */

        uint8_t buddyID; /* selected */

        ts_coordinates_t glcdCoo;

        event void TouchTimer.fired() {
                /*debug("JabberClientC", "Timer 0 fired @ %s.\n", sim_time_string());*/
                call TouchScreen.getCoordinates(&glcdCoo);
                call VolumeReading.read();
        }
        void undrawCaret() {
                if(entryCaretVisible) {
                        char cc[2] = {(entry[entryPos] != 0 ? entry[entryPos] : '␣'),0};
                        call LCD2.goTo(entryLine, entryCol);
                        call LCD2.write(cc);
                        entryCaretVisible = FALSE;
                }
        }
        /* both args in PROGMEM */
```

```
void playSound(PGM_VOID_P src, PGM_VOID_P xlen /* 16 bit target */) {
        call MP3.sendData(src, pgm_read_word(xlen));
}
void drawCaret() {
        if(!entryCaretVisible) {
                char cc[2] = {'|',0};
                call LCD2.goTo(entryLine, entryCol);
                call LCD2.write(cc);
                entryCaretVisible = TRUE;
        }
}
event void CaretBlinkTimer.fired() {
        if(entryCaretVisible)
                undrawCaret();
        else
                drawCaret();
        call LCD2.forceRefresh();
}
void moveCaret(int8_t offset) {
        undrawCaret();
        if(offset < 0) {
                if(entryPos > 0) {
                        --entryPos;
                        if(entryCol > 0)
                                --entryCol;
                        else {
                                entryCol = MAX_2LINE_DISPLAY_WIDTH - 1;
                                if(entryLine > 0)
                                        --entryLine;
                        }
                }
        } else if(offset > 0) {
                if(entry[entryPos] != 0) {
                        ++entryPos;
                        ++entryCol;
                        if(entryCol >= MAX_2LINE_DISPLAY_WIDTH) {
                                entryCol = 0;
                                ++entryLine;
                        }
                }
        }
        drawCaret();
}
event void Keyboard.receivedChar(uint8_t chr) {
        if(strlen(entry) < MAX_ENTRY_SIZE - 1) {
                memmove(&entry[entryPos + 1], &entry[entryPos], MAX_ENTRY_SIZE - entryPos);
                entry[entryPos] = chr;
                call LCD2.goTo(entryLine, entryCol);
                call LCD2.write(&entry[entryPos]);
                moveCaret(1);
        }
}
void deleteEntryChar() {
        undrawCaret();
        memmove(&entry[entryPos], &entry[entryPos + 1], MAX_ENTRY_SIZE - (entryPos + 1));
        call LCD2.goTo(entryLine, entryCol);
        call LCD2.write(&entry[entryPos]);
        call LCD2.write(" ");
}
void backspaceEntryChar() {
        if(entryPos == 0)
                return;
        moveCaret(-1);
        deleteEntryChar();
}
void clearEntry() {
        undrawCaret();
        entryPos = entryLine = entryCol = 0;
        entry[0] = 0;
        call LCD2.clear();
        }
#define XBUDDY(buddyID) (((buddyID) == 0 || (buddyID) == 2) ? 0 : 64)
#define YBUDDY(buddyID) (((buddyID) >= 2) ? 20 : 10)
        uint8_t getHitBuddy(uint8_t x, uint8_t y) {
                if(y < 20) {
                        if(y < 10)
                                return x < 64 ? 0 : 1;
                        else
                                return x < 64 ? 2 : 3;
                } else
                        return INVALID_BUDDY_ID;
        }
        void updateGUI() {
                uint8_t i;
                call GLCD.fill(0x00);
                /*debug("UPDATING GUI");*/
                for(i = 0; i < MAX_BUDDIES; ++i) {
                        buddy_t* buddy = call MessageManager.getBuddy(i);
                        if(buddy == NULL)
                                continue;
                        switch(buddy->status) {
                        case BUDDY_STATUS_EMPTY:
                                break;
                        case BUDDY_STATUS_UNAVAILABLE:
                                call GLCD.drawLine(XBUDDY(i), YBUDDY(i) - 4, XBUDDY(i) + 63, YBUDDY(i)
                                     - 4);
                                break;
                        case BUDDY_STATUS_AVAILABLE:
                                break;
                        case BUDDY_STATUS_SUBSCRIBED: /* WTF FIXME */
                                break;
                        }
                        call GLCD.drawText(buddy->jid, XBUDDY(i) + 2 + ICON_WIDTH, YBUDDY(i));
                        if(buddyID == i) { /* currently selected buddy */
                                call GLCD.drawRect(XBUDDY(i), YBUDDY(i) - 7 - 2, XBUDDY(i) + 63,
                                    YBUDDY(i) + 1);
                                if(buddy->gotNewMessage)
                                        call MessageManager.markMessageSeen(buddy); /* will reset flag
                                             */
                                if(buddy->message[0] != 0)
                                        call GLCD.drawText(buddy->message, 0 + 2, 32);
                                else if(buddy->requestsPresenceSubscription)
                                        call GLCD.drawText("<Requests␣to␣subscribe␣to␣your␣presence.␣
                                            Answer␣to␣subscribe>", 0 + 2, 32);
                        }
                        if(buddy->gotNewMessage)
                                call GLCD.drawText("M", XBUDDY(i) + 2, YBUDDY(i));
                }
                call GLCD.drawLine(0, 23, 127, 23);
        }
        void sendCurrentMessage() {
                call MessageManager.sendMessage(buddyID, entry);
                updateGUI(); /* sendMessage() could have updated the "subscription requested" flag */
        }
        event void Keyboard.keyPressed(enum Key key, uint8_t shiftState) {
                switch(key) {
                case VK_LEFT:
                        moveCaret(-1);
                        break;
                case VK_RIGHT:
                        moveCaret(1);
                        break;
                case VK_RETURN:
                case VK_KP_ENTER:
                        sendCurrentMessage();
                        clearEntry();
                        break;
                case VK_BACKSPACE:
                        backspaceEntryChar();
                        break;
        case VK_DELETE:
                deleteEntryChar();
                break;
        case VK_ESCAPE:
                clearEntry();
                break;
        case VK_END:
                while(entry[entryPos] != 0)
                        moveCaret(1);
                break;
        case VK_HOME:
                while(entryPos > 0)
                        moveCaret(-1);
                break;
        default:
                break;
        }
}
event void TouchScreen.coordinatesReady() {
        uint8_t xbuddyID = getHitBuddy(glcdCoo.x, glcdCoo.y);
        if(xbuddyID != buddyID && xbuddyID != INVALID_BUDDY_ID) {
                buddyID = xbuddyID;
                /* don't allow selecting EMPTY buddies */
                {
                        buddy_t* buddy = call MessageManager.getBuddy(buddyID);
                        if(buddy == NULL)
                                buddyID = INVALID_BUDDY_ID;
                }
                updateGUI();
        }
}
event void Boot.booted() {
        entryLine = entryCol = entryPos = 0;
        buddyID = INVALID_BUDDY_ID;
        entry[0] = 0;
        call CaretBlinkTimer.startPeriodic(500);

        call TouchScreen.calibrate(0,-10);
        call TouchTimer.startPeriodic(100);
        call LCD2.autoRefresh(60); /* ms */
        call LCD2.clear();
        call LCD2.write(”|”); /* cursor */
        call GLCD.fill(0x00);
        /*call MP3.sineTest(TRUE);*/
        /*playSound(login_mp3, &login_mp3_len);*/
        updateGUI();
}
event void MessageManager.presenceUpdated(uint8_t xbuddyID, buddy_t* buddy) {
        switch(buddy->status) {
        case BUDDY_STATUS_UNAVAILABLE:
                playSound(logout_mp3, &logout_mp3_len);
                break;
        case BUDDY_STATUS_AVAILABLE:
                playSound(login_mp3, &login_mp3_len);
                break;
        case BUDDY_STATUS_EMPTY: /* cannot happen */
        case BUDDY_STATUS_SUBSCRIBED: /* TODO what is this? */
        }
        updateGUI();
}
event uint8_t MessageManager.subscriptionRequest(char* jid) {
        updateGUI();
        return 0; /* will be manually allowed later (on sendMessage). */
}
event void MessageManager.messageReceived(uint8_t xbuddyID, buddy_t* buddy) {
        playSound(noti_mp3, &noti_mp3_len);
        updateGUI();
}
event void VolumeReading.readDone(error_t result, uint16_t val) {
        if(result == SUCCESS) {
```

```
                                call MP3.setVolume(val >> 2);
                        }
                }
}
```

A.2 Keyboard

Listing 7: Application/Keyboard.nc

```
#include <stdint.h>
#include "Keyboard.h"
interface Keyboard {
        /**
         * Fired when a character has been entered on the keyboard
         *
         * @param chr ASCII value of the character entered
         */
        event void receivedChar(uint8_t chr);

        /** Mostly useful for cursor keys etc. Code is the scan code, shiftState the shift state flags
            . */
        event void keyPressed(enum Key code, uint8_t shiftState);

}
```

Listing 8: Application/Keyboard.h

```
#ifndef __KEYBOARD_H
#define __KEYBOARD_H

enum Key {
        VK_INSERT = 0xE070,
        VK_HOME = 0xE06C,
        VK_PAGE_UP = 0xE07D,
        VK_DELETE = 0xE071,
        VK_END = 0xE069,
        VK_PAGE_DOWN = 0xE07A,
        VK_UP = 0xE075,
        VK_LEFT = 0xE06B,
        VK_DOWN = 0xE072,
        VK_RIGHT = 0xE074,
        VK_KP_ENTER = 0xE05A,
        VK_RETURN = 0x5A,
        VK_F1 = 0x5,
        VK_F2 = 0x6,
        VK_F3 = 0x4,
        VK_F4 = 0xC,
        VK_F5 = 0x3,
        VK_F6 = 0xB,
        VK_F7 = 0x83,
        VK_F8 = 0x0A,
        VK_F9 = 0x1,
        VK_F10 = 0x9,
        VK_F11 = 0x78,
        VK_F12 = 0x7,
        VK_ESCAPE = 0x76,
        VK_CAPS = 0x58,
        VK_LGUI = 0xE01F,
        VK_RGUI = 0xE027,
        VK_APPS = 0xE02F,
        VK_TAB = 0x0D,
        VK_BACKSPACE = 0x66,
};
enum {
        /* shift state */
        SS_LSHIFT = 1 << 0,
        SS_RSHIFT = 1 << 1,
        SS_LCTRL = 1 << 2,
        SS_RCTRL = 1 << 3,
        SS_LALT = 1 << 4,
        SS_RALT = 1 << 5,
};
#endif /* ndef __KEYBOARD_H */
```

Listing 9: Application/KeyboardC.nc

```
#include <ctype.h>
module KeyboardC {
        uses interface HplPS2;
        provides interface Keyboard;
        provides interface Init;
        uses interface AsyncQueue<uint16_t> as Buffer;
}
implementation {
        enum {
                /* states */
                S_INITIAL,
                S_E0,
                S_F0,
                S_E0_F0,
        };
#define SHIFTED_MAP { \
                [0x1C] = 'A',\
                [0x32] = 'B',\
                [0x21] = 'C',\
                [0x23] = 'D',\
                [0x24] = 'E',\
                [0x2B] = 'F',\
                [0x34] = 'G',\
                [0x33] = 'H',\
                [0x43] = 'I',\
                [0x3B] = 'J',\
                [0x42] = 'K',\
                [0x4B] = 'L',\
                [0x3A] = 'M',\
                [0x31] = 'N',\
                [0x44] = 'O',\
                [0x4D] = 'P',\
                [0x15] = 'Q',\
                [0x2D] = 'R',\
                [0x1B] = 'S',\
                [0x2C] = 'T',\
                [0x3C] = 'U',\
                [0x2A] = 'V',\
                [0x1D] = 'W',\
                [0x22] = 'X',\
                [0x35] = 'Y',\
                [0x1A] = 'Z',\
                [0x45] = ')',\
                [0x16] = '!',\
                [0x1E] = '@',\
                [0x26] = '#',\
                [0x25] = '$',\
                [0x2E] = '%',\
                [0x36] = '^',\
                [0x3D] = '&',\
                [0x3E] = '*',\
                [0x46] = '(',\
                [0x0E] = '~',\
                [0x4E] = '_',\
                [0x5D] = '|',\
                [0x29] = ' ',\
                [0x54] = '{',\
                [0x55] = '+',\
                [0x5B] = '}',\
                [0x4C] = ':',\
                [0x52] = '"',\
                [0x41] = '<',\
        [0x49] = '>',\
        [0x4A] = '?',\
        [0x61] = '>', /* German keyboard */}
static uint8_t PROGMEM asciiFromScan[4][256] = {
        [0] = {
                [0x1C] = 'a',
                [0x32] = 'b',
                [0x21] = 'c',
                [0x23] = 'd',
                [0x24] = 'e',
                [0x2B] = 'f',
                [0x34] = 'g',
                [0x33] = 'h',
                [0x43] = 'i',
                [0x3B] = 'j',
                [0x42] = 'k',
                [0x4B] = 'l',
                [0x3A] = 'm',
                [0x31] = 'n',
                [0x44] = 'o',
                [0x4D] = 'p',
                [0x15] = 'q',
                [0x2D] = 'r',
                [0x1B] = 's',
                [0x2C] = 't',
                [0x3C] = 'u',
                [0x2A] = 'v',
                [0x1D] = 'w',
                [0x22] = 'x',
                [0x35] = 'y',
                [0x1A] = 'z',
                [0x45] = '0',
                [0x16] = '1',
                [0x1E] = '2',
                [0x26] = '3',
                [0x25] = '4',
                [0x2E] = '5',
                [0x36] = '6',
                [0x3D] = '7',
                [0x3E] = '8',
                [0x46] = '9',
                [0x0E] = '`',
                [0x4E] = '-',
                [0x5D] = '\\',
                [0x29] = '␣',
                [0x54] = '[',
                [0x55] = '=',
                [0x5B] = ']',
                [0x4C] = ';',
                [0x52] = '\'',
                [0x41] = ',',
                [0x49] = '.',
                [0x4A] = '/',
                [0x61] = '<', /* German keyboard */
        },
        [SS_LSHIFT] = SHIFTED_MAP,
        [SS_RSHIFT] = SHIFTED_MAP,
        [SS_LSHIFT|SS_RSHIFT] = SHIFTED_MAP};
volatile uint8_t state; /* for decoding */
volatile uint8_t shiftState;
command error_t Init.init() {
        atomic {
                state = S_INITIAL;
        }
        return SUCCESS;
}
uint8_t updateShiftState(uint16_t code) {
        switch(code) {
        case 0x11:
                shiftState |= SS_LALT;
                break;
        case 0x12:
                shiftState |= SS_LSHIFT;
                break;
        case 0x59:
                shiftState |= SS_RSHIFT;
                break;
        case 0x14:
                shiftState |= SS_LCTRL;
                break;
        case 0xE011:
                shiftState |= SS_RALT;
                break;
        case 0xE014:
                shiftState |= SS_RCTRL;
                break;
        case 0x0F11:
                shiftState &=~ SS_LALT;
                break;
        case 0x0F12:
                shiftState &=~ SS_LSHIFT;
                break;
        case 0x0F59:
                shiftState &=~ SS_RSHIFT;
                break;
        case 0x0F14:
                shiftState &=~ SS_LCTRL;
                break;
        case 0xF011:
                shiftState &=~ SS_RALT;
                break;
        case 0xF014:
                shiftState &=~ SS_RCTRL;
                break;
        }
        return shiftState;
}
bool breakCodeP(uint16_t code) {
        return (code & 0xF000) == 0xF000 || (code & 0xFF00) == 0x0F00;
}
task void enqueued() {
        uint16_t receivedData;
        uint8_t xshiftState;
        atomic {
                if(!call Buffer.empty()) {
                        receivedData = call Buffer.dequeue();
                        xshiftState = updateShiftState(receivedData);
                } else
                        return;
        }
        {
                /* TODO if there are more maps for different shift states, just remove &3 and
                    increase asciiFromScan size to 64 */
                uint8_t asciiCode = (receivedData < 256) ? pgm_read_byte(&asciiFromScan[
                    xshiftState&3][receivedData]) :
                                    (receivedData >= 0xE000 && receivedData < 0xE100) ? 0/*
                                        asciiFromScan[xshiftState&3][0x100 | (receivedData & 0
                                        xFF)]*/ :
                                    0;
                if(!breakCodeP(receivedData)) {
                        signal Keyboard.keyPressed(receivedData, xshiftState);
                        if(asciiCode != 0)
                                signal Keyboard.receivedChar(asciiCode);
                }
        }
}
async event void HplPS2.receivedCode(uint8_t code) {
        atomic {
                switch(state) {
                case S_INITIAL:
```

```
                                if(code == 0xE0)
                                        state = S_E0;
                                else if(code == 0xF0)
                                        state = S_F0;
                                else {
                                        call Buffer.enqueue(code);
                                        post enqueued();
                                        state = S_INITIAL;
                                }
                                break;
                        case S_E0:
                                if(code == 0xF0) { /* break */
                                        state = S_E0_F0;
                                } else { /* extended key make code */
                                        call Buffer.enqueue(0xE000 | code);
                                        post enqueued();
                                        state = S_INITIAL;
                                }
                                break;
                        case S_F0:
                                call Buffer.enqueue(0x0F00 | code);
                                post enqueued();
                                state = S_INITIAL;
                                break;
                        case S_E0_F0:
                                /* extended break code */
                                call Buffer.enqueue(0xF000 | code);
                                post enqueued();
                                state = S_INITIAL;
                                break;
                        }
                }
        }
}
```

Listing 10: Application/HplPS2.nc

```
interface HplPS2 {
        /**
         * Fired when a a scan code part is received from the keyboard.
         *
         * @param chr scan code part received
         */
        async event void receivedCode(uint8_t code);

}
```

Listing 11: Application/HplPS2C.nc

```
module HplPS2C {
        uses interface GeneralIO as DataPort;
        uses interface GeneralIO as ClockPort;

        uses interface HplAtm128Interrupt as ClockInterrupt;
        provides interface HplPS2;
        provides interface Init;
}
implementation {
        /* PS/2 data arrives LSB first */
        enum {
                B_START = 0,
                B_PARITY = 9,
                B_STOP = 10,
        };
        volatile uint16_t receivedData;
        volatile uint8_t bitIndex;
        command error_t Init.init() {
                atomic {
                        bitIndex = 0;
                        receivedData = 0;
                }
                call DataPort.set();
                call DataPort.makeInput();
                call ClockPort.set();
                call ClockPort.makeInput();
                call ClockInterrupt.clear();
                call ClockInterrupt.edge(FALSE); /* falling edge */
                call ClockInterrupt.enable();
                return SUCCESS;
        }
        async event void ClockInterrupt.fired() {
                /*call ClockInterrupt.disable();*/
                atomic {
                        uint8_t value = call DataPort.get();
                        if(bitIndex == 0) {
                                if (value != 0) /* skip junk until the start bit */
                                        return;
                                receivedData = 0;
                        }
                        receivedData |= value << bitIndex;
                        ++bitIndex;
                        if(bitIndex == 11) {
                                uint8_t parity = 1, i;
                                for(i = 1; i <= 8; ++i)
                                        if((receivedData & (1 << i)) != 0)
                                                parity ^= 1;
                                bitIndex = 0;
                                // receivedData&1 start bit
                                // 1<<1 .. 1<<8 data bits
                                // 1<<9 parity bit (odd)
                                // 1<<10 stop bit. always 1
                                if((receivedData & (1 << B_STOP)) != 0 && ((receivedData & (1 <<
                                    B_PARITY)) >> B_PARITY) == parity) /* stop bit is set */ {
                                        receivedData = (receivedData >> 1) & 0xFF; /* just the payload
                                             */
                                        signal HplPS2.receivedCode(receivedData);
                                }
                        }
                }
        }
}
/*
 Steps the host must follow to send data to a PS/2 device:

    1)   Bring the Clock line low for at least 100 microseconds.
    2)   Bring the Data line low.
    3)   Release the Clock line.
    4)   Wait for the device to bring the Clock line low.
    5)   Set/reset the Data line to send the first data bit
    6)   Wait for the device to bring Clock high.
    7)   Wait for the device to bring Clock low.
    8)   Repeat steps 5-7 for the other seven data bits and the parity bit
    9)   Release the Data line.
    10) Wait for the device to bring Data low.
    11) Wait for the device to bring Clock  low.
    12) Wait for the device to release Data and Clock

 All data is transmitted one byte at a time and each byte is sent in a frame consisting of 11-12 bits.
       These bits are:

    1 start bit.  This is always 0.
    8 data bits, least significant bit first.
    1 parity bit (odd parity).
    1 stop bit.  This is always 1.
    1 acknowledge bit (host-to-device communication only)

*/
```

A.3 MP3

Listing 12: Application/sounds.h

```
#ifndef __SOUNDS_H__
#define __SOUNDS_H__

const uint8_t PROGMEM login_mp3[] = {
  0x49, 0x44, 0x33, 0x04, 0x00, 0x40, 0x00, 0x00, 0x00, 0x2b, 0x00, 0x00,
  0x00, 0x0c, 0x01, 0x20, 0x05, 0x01, 0x72, 0x62, 0x16, 0x2d, 0x54, 0x58,
  0x58, 0x58, 0x00, 0x00, 0x00, 0x15, 0x00, 0x00, 0x00, 0x53, 0x6f, 0x66,
  0x74, 0x77, 0x61, 0x72, 0x65, 0x00, 0x4c, 0x61, 0x76, 0x66, 0x35, 0x33,
  0x2e, 0x32, 0x30, 0x2e, 0x30, 0xff, 0xf3, 0x10, 0xc4, 0x00, 0x01, 0xe8,
  0x05, 0xed, 0xa0, 0x00, 0x00, 0x00, 0xcc, 0xb1, 0x40, 0x0a, 0xa3, 0xa9,
  0xdc, 0xa2, 0xca, 0xf4, 0xff, 0xff, 0xff, 0xff, 0xf3, 0x12, 0xc4, 0x05,
  0x02, 0x80, 0x05, 0xfc, 0x00, 0x00, 0x44, 0x00, 0xa5, 0xb3, 0x19, 0x22,
  0x08, 0x02, 0x06, 0x08, 0x0e, 0xdd, 0x3d, 0x7f, 0xff, 0xff, 0xff, 0xf3,
  0x10, 0xc4, 0x09, 0x02, 0xb8, 0x06, 0x0e, 0x21, 0x40, 0x10, 0x00, 0xfe,
  0x2b, 0xff, 0xff, 0x83, 0xf2, 0x81, 0xbf, 0xea, 0x77, 0xff, 0x53, 0x9c,
  0xff, 0xf3, 0x10, 0xc4, 0x0b, 0x05, 0x59, 0x4e, 0xd8, 0x01, 0x81, 0x10,
  0x00, 0x47, 0xff, 0x38, 0x86, 0x7f, 0xff, 0xc8, 0x46, 0x40, 0x01, 0x0a,
  0xff, 0xff, 0xff, 0xf3, 0x10, 0xc4, 0x02, 0x02, 0x60, 0x56, 0xf8, 0x01,
  0xc7, 0x10, 0x00, 0xe8, 0x1c, 0x7b, 0xff, 0xff, 0xff, 0xda, 0x9a, 0x30,
  0x3f, 0xff, 0xab, 0xa8, 0xff, 0xf3, 0x10, 0xc4, 0x05, 0x03, 0x01, 0x46,
  0xe0, 0x10, 0x68, 0x04, 0xd1, 0x1f, 0x9f, 0x97, 0xff, 0xfe, 0x15, 0xe7,
  0x14, 0x0a, 0x22, 0x03, 0x0c, 0x98, 0xff, 0xf3, 0x10, 0xc4, 0x06, 0x04,
  0x21, 0x46, 0xfc, 0x78, 0x01, 0x44, 0xa3, 0xaf, 0xff, 0xfe, 0xff, 0xff,
  0xfc, 0xdf, 0xff, 0xf5, 0x1e, 0x7a, 0xff, 0xfe, 0xff, 0xf3, 0x10, 0xc4,
  0x02, 0x02, 0x31, 0x46, 0xd4, 0x00, 0x68, 0x04, 0xd4, 0xae, 0xa0, 0xa7,
  0x57, 0xfc, 0x1a, 0x26, 0xff, 0xfe, 0xbe, 0xb0, 0x94, 0x5f, 0xff, 0xf3,
  0x10, 0xc4, 0x06, 0x02, 0x49, 0x46, 0xcc, 0x00, 0x68, 0x04, 0xd5, 0xfc,
  0x18, 0xd2, 0xd5, 0xff, 0xff, 0xe8, 0x08, 0x7f, 0xfa, 0x8c, 0x8a, 0x5d,
  0xff, 0xf3, 0x12, 0xc4, 0x09, 0x01, 0xd9, 0x4a, 0xcc, 0x00, 0x50, 0x04,
  0xd4, 0x00, 0x01, 0xbf, 0xff, 0xff, 0xfe, 0x32, 0x26, 0x03, 0xff, 0xff,
  0xff, 0xfe, 0x21, 0xff, 0xf3, 0x10, 0xc4, 0x0f, 0x01, 0xf0, 0x5a, 0xcc,
  0x00, 0x08, 0x0e, 0x10, 0x21, 0x25, 0x40, 0x47, 0xff, 0xff, 0xff, 0xc8,
  0xa1, 0xd8, 0x29, 0xb3, 0xff, 0xff, 0xf3, 0x10, 0xc4, 0x14, 0x01, 0xf8,
  0x56, 0xd4, 0x08, 0x00, 0x0e, 0x12, 0xff, 0xff, 0xd4, 0x3d, 0x26, 0x15,
  0xff, 0xff, 0xff, 0xf8, 0xf5, 0x80, 0x41, 0xff, 0xf3, 0x10, 0xc4, 0x19,
  0x01, 0xf0, 0x56, 0xd0, 0x08, 0x00, 0x0e, 0x10, 0x84, 0xff, 0xff, 0xff,
  0xfe, 0xb1, 0xea, 0x5f, 0xb1, 0x1f, 0xff, 0xff, 0xff, 0xff, 0xf3, 0x10,
  0xc4, 0x1e, 0x01, 0xf0, 0x5a, 0xc8, 0x00, 0x00, 0x16, 0x10, 0x6a, 0x6a,
  0x5b, 0xc1, 0xf0, 0xa7, 0xff, 0xff, 0x7f, 0xf9, 0x24, 0xd5, 0x1b, 0xff,
  0xf3, 0x10, 0xc4, 0x23, 0x01, 0xa0, 0x5a, 0xd4, 0x08, 0x00, 0x0a, 0x10,
  0xe7, 0x6a, 0xc1, 0x81, 0xff, 0xff, 0xff, 0xff, 0x93, 0x49, 0x04, 0xfa,
  0x39, 0xff, 0xf3, 0x10, 0xc4, 0x29, 0x02, 0x18, 0x5a, 0xdc, 0x30, 0x00,
  0x0a, 0x10, 0x40, 0x36, 0xff, 0xfa, 0xbf, 0xff, 0xeb, 0x1e, 0x60, 0x0e,
  0x3f, 0x02, 0x81, 0xff, 0xf3, 0x10, 0xc4, 0x2d, 0x01, 0xd0, 0x56, 0xc8,
  0x00, 0x08, 0x0c, 0x10, 0x2f, 0xff, 0xff, 0x6f, 0xff, 0xff, 0x7f, 0xff,
  0xf6, 0x1d, 0x6a, 0x0c, 0xed, 0xff, 0xf3, 0x12, 0xc4, 0x32, 0x02, 0x48,
  0x56, 0xb0, 0x00, 0x08, 0x14, 0x10, 0x20, 0x75, 0xdf, 0xff, 0xff, 0xfe,
  0x44, 0x95, 0x0a, 0x17, 0x14, 0x02, 0x5f, 0xff, 0xff, 0xf3, 0x10, 0xc4,
  0x36, 0x02, 0x80, 0x5a, 0xa8, 0x00, 0x10, 0x16, 0x0c, 0xff, 0x18, 0xff,
  0xf3, 0xca, 0x14, 0x60, 0x00, 0xa0, 0x53, 0xa9, 0xff, 0xd7, 0xff, 0xf3,
  0x10, 0xc4, 0x39, 0x02, 0xf8, 0x5a, 0xa8, 0x28, 0x10, 0x0e, 0x0c, 0x4c,
  0x41, 0x4d, 0x45, 0x33, 0x2e, 0x39, 0x39, 0x2e, 0x35, 0x55, 0x55, 0x55,
  0xff, 0xf3, 0x10, 0xc4, 0x3a, 0x03, 0xd1, 0x4a, 0xd4, 0x78, 0x01, 0x44,
  0xa0, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55,
  0x55, 0x55, 0xff, 0xf3, 0x10, 0xc4, 0x37, 0x02, 0x68, 0x56, 0xa4, 0x00,
  0x28, 0x12, 0x10, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55,
  0x55, 0x55, 0x55, 0x55, 0xff, 0xf3, 0x10, 0xc4, 0x3a, 0x02, 0x98, 0x5a,
  0xa0, 0x00, 0x08, 0x0c, 0x10, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55,
  0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0xff, 0xf3, 0x10, 0xc4, 0x3c, 0x01,
  0xf8, 0x02, 0x66, 0x40, 0x08, 0x00, 0x02, 0x55, 0x55, 0x55, 0x55, 0x55,
  0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55
};
const uint16_t PROGMEM login_mp3_len = 680;

const uint8_t PROGMEM logout_mp3[] = {
  0x49, 0x44, 0x33, 0x04, 0x00, 0x40, 0x00, 0x00, 0x00, 0x4c, 0x00, 0x00,
0x00, 0x0c, 0x01, 0x20, 0x05, 0x05, 0x17, 0x38, 0x20, 0x55, 0x54, 0x50,
0x45, 0x31, 0x00, 0x00, 0x00, 0x0e, 0x00, 0x00, 0x00, 0x65, 0x6c, 0x65,
0x76, 0x61, 0x74, 0x6f, 0x72, 0x20, 0x64, 0x69, 0x6e, 0x67, 0x54, 0x43,
0x4f, 0x4e, 0x00, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00, 0x42, 0x6c, 0x75,
0x65, 0x73, 0x54, 0x49, 0x54, 0x32, 0x00, 0x00, 0x00, 0x0e, 0x00, 0x00,
0x00, 0x65, 0x6c, 0x65, 0x76, 0x61, 0x74, 0x6f, 0x72, 0x20, 0x64, 0x69,
0x6e, 0x67, 0xff, 0xf3, 0x14, 0x64, 0x00, 0x00, 0x00, 0x01, 0x06, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x40, 0x00, 0x00, 0x00, 0x00, 0x90,
0xff, 0x78, 0xff, 0xf3, 0x14, 0x64, 0x03, 0x00, 0x00, 0x01, 0x16, 0x00,
0x00, 0x23, 0x00, 0x00, 0x00, 0x02, 0x60, 0x00, 0x00, 0x46, 0x00, 0x01,
0xff, 0xe7, 0xff, 0xf3, 0x14, 0x64, 0x06, 0x00, 0x00, 0x01, 0x18, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x60, 0x00, 0x00, 0x00, 0x00, 0x04,
0x3a, 0xfe, 0xff, 0xf3, 0x14, 0x64, 0x09, 0x00, 0x20, 0x03, 0x1e, 0x00,
0xa0, 0x00, 0x00, 0x00, 0x00, 0x02, 0x64, 0x01, 0x40, 0x00, 0x00, 0x0b,
0xfc, 0x6a, 0xff, 0xf3, 0x14, 0x64, 0x0b, 0x00, 0x68, 0x19, 0x92, 0x00,
0xc1, 0x00, 0x00, 0x00, 0xd8, 0x32, 0xec, 0x01, 0x82, 0x10, 0x00, 0x88,
0x30, 0xe2, 0xff, 0xf3, 0x14, 0x64, 0x07, 0x00, 0x38, 0x11, 0x98, 0x00,
0xe1, 0x00, 0x00, 0x00, 0x70, 0x27, 0x04, 0x01, 0xc2, 0x00, 0x00, 0x55,
0x85, 0x3d, 0xff, 0xf3, 0x14, 0x64, 0x06, 0x00, 0x28, 0x11, 0x8e, 0x00,
0x00, 0x07, 0x00, 0x00, 0x78, 0x2a, 0xe4, 0x00, 0x00, 0x14, 0x00, 0x03,
0x88, 0x88, 0xff, 0xf3, 0x14, 0x64, 0x05, 0x00, 0x34, 0x13, 0x8e, 0x00,
0x00, 0x05, 0x00, 0x00, 0x90, 0x2a, 0xe0, 0x00, 0x00, 0x14, 0x00, 0x1e,
0x80, 0xc4, 0xff, 0xf3, 0x14, 0x64, 0x04, 0x00, 0x38, 0x13, 0x8c, 0x00,
0x00, 0x07, 0x00, 0x00, 0x90, 0x2a, 0xe8, 0x00, 0x00, 0x0e, 0x00, 0x85,
0x3d, 0x03, 0xff, 0xf3, 0x14, 0x64, 0x03, 0x00, 0x34, 0x13, 0x90, 0x00,
0x00, 0x05, 0x00, 0x00, 0x68, 0x26, 0xf0, 0x00, 0x00, 0x0a, 0x00, 0x2a,
0xe4, 0x68, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x03, 0x98, 0x00,
0x00, 0x02, 0x00, 0x00, 0x58, 0x26, 0xf8, 0x00, 0x00, 0x04, 0x00, 0x35,
0xe4, 0x79, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x03, 0x98, 0x00,
0x00, 0x00, 0x00, 0x00, 0x50, 0x07, 0x00, 0x00, 0x00, 0x04, 0x00, 0x05,
0xe4, 0x79, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x03, 0x98, 0x00,
0x00, 0x00, 0x00, 0x00, 0x50, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05,
0xe4, 0x79, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x03, 0x98, 0x00,
0x00, 0x00, 0x00, 0x00, 0x50, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05,
0xe9, 0xe4, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x20, 0x03, 0x94, 0x00,
0x00, 0x02, 0x00, 0x00, 0x50, 0x06, 0xfc, 0x00, 0x00, 0x04, 0x00, 0x15,
0xc3, 0x61, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x13, 0x8a, 0x00,
0x00, 0x05, 0x00, 0x00, 0x48, 0x22, 0xe0, 0x00, 0x00, 0x0a, 0x00, 0x4a,
0xc3, 0x60, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x13, 0x86, 0x00,
0x00, 0x05, 0x00, 0x00, 0x40, 0x22, 0xec, 0x00, 0x00, 0x04, 0x00, 0xaa,
0xe9, 0xe4, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x20, 0x03, 0x92, 0x00,
0x00, 0x02, 0x00, 0x00, 0x50, 0x06, 0xf4, 0x00, 0x00, 0x04, 0x00, 0x6a,
0xe4, 0xf9, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x03, 0x92, 0x00,
0x00, 0x00, 0x00, 0x00, 0x50, 0x06, 0xf4, 0x00, 0x00, 0x00, 0x00, 0x1a,
0xe4, 0xf9, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x03, 0x92, 0x00,
0x00, 0x00, 0x00, 0x00, 0x50, 0x06, 0xf4, 0x00, 0x00, 0x00, 0x00, 0x1a,
0xe4, 0xf9, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x03, 0x92, 0x00,
0x00, 0x00, 0x00, 0x00, 0x50, 0x06, 0xf4, 0x00, 0x00, 0x00, 0x00, 0x1a,
0xe4, 0xf9, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x03, 0x8a, 0x00,
0x00, 0x00, 0x00, 0x00, 0x50, 0x06, 0xf0, 0x00, 0x00, 0x04, 0x00, 0x1a,
0x88, 0x70, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x11, 0x7c, 0x00,
0x00, 0x07, 0x00, 0x00, 0x50, 0x26, 0xd8, 0x00, 0x00, 0x0a, 0x00, 0xda,
0x83, 0xc3, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x20, 0x11, 0x84, 0x00,
0x00, 0x02, 0x00, 0x00, 0x50, 0x26, 0xd8, 0x00, 0x00, 0x0a, 0x00, 0x6a,
0xe4, 0xba, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x03, 0x88, 0x00,
0x00, 0x02, 0x00, 0x00, 0x40, 0x06, 0xe8, 0x00, 0x00, 0x04, 0x00, 0x6a,
0xe4, 0xb9, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x03, 0x88, 0x00,
0x00, 0x00, 0x00, 0x00, 0x50, 0x06, 0xe8, 0x00, 0x00, 0x00, 0x00, 0x3a,
0xe4, 0xb9, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x03, 0x88, 0x00,
0x00, 0x00, 0x00, 0x00, 0x50, 0x06, 0xe8, 0x00, 0x00, 0x00, 0x00, 0x3a,
0xe4, 0xb9, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x03, 0x88, 0x00,
0x00, 0x00, 0x00, 0x00, 0x50, 0x06, 0xe8, 0x00, 0x00, 0x00, 0x00, 0x3a,
0xe4, 0xb9, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x28, 0x03, 0x84, 0x00,
0x00, 0x00, 0x00, 0x00, 0x50, 0x06, 0xdc, 0x00, 0x00, 0x04, 0x00, 0x3a,
0xc6, 0x83, 0xff, 0xf3, 0x14, 0x64, 0x02, 0x00, 0x20, 0x13, 0x7e, 0x00,
0x00, 0x02, 0x00, 0x00, 0x40, 0x22, 0xd0, 0x00, 0x00, 0x04, 0x00, 0xc3,
0x21, 0x75, 0xff, 0xf3, 0x14, 0x64, 0x03, 0x00, 0x28, 0x13, 0x7a, 0x00,
0x00, 0x05, 0x00, 0x00, 0x48, 0x22, 0xc4, 0x00, 0x00, 0x0a, 0x00, 0xc6,
0xe4, 0x95, 0xff, 0xf3, 0x14, 0x64, 0x03, 0x00, 0x20, 0x13, 0x80, 0x00,
0x00, 0x02, 0x00, 0x00, 0x50, 0x06, 0xd8, 0x00, 0x00, 0x04, 0x00, 0xe4,
  0x39, 0x25, 0xff, 0xf3, 0x14, 0x64, 0x03, 0x00, 0x28, 0x03, 0x80, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x50, 0x06, 0xd8, 0x00, 0x00, 0x00, 0x00, 0xe4,
  0x39, 0x25, 0xff, 0xf3, 0x14, 0x64, 0x03, 0x00, 0x28, 0x03, 0x80, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x50, 0x06, 0xd8, 0x00, 0x00, 0x00, 0x00, 0xe4,
  0x39, 0x25, 0xff, 0xf3, 0x14, 0x64, 0x03, 0x00, 0x28, 0x03, 0x80, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x50, 0x06, 0xd8, 0x00, 0x00, 0x00, 0x00, 0xe4,
  0x39, 0x25, 0xff, 0xf3, 0x14, 0x64, 0x03, 0x00, 0x28, 0x03, 0x7c, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x50, 0x06, 0xd0, 0x00, 0x00, 0x00, 0x00, 0x82,
  0xc6, 0x88, 0xff, 0xf3, 0x14, 0x64, 0x03, 0x00, 0x20, 0x11, 0x74, 0x00,
  0x00, 0x02, 0x00, 0x00, 0x40, 0x26, 0xc8, 0x00, 0x00, 0x04, 0x00, 0x30,
  0xc5, 0xe4, 0xff, 0xf3, 0x14, 0x64, 0x04, 0x00, 0x28, 0x11, 0x6c, 0x00,
  0x00, 0x07, 0x00, 0x00, 0x50, 0x26, 0xb8, 0x00, 0x00, 0x0a, 0x00, 0x71,
  0x95, 0xe4, 0xff, 0xf3, 0x14, 0x64, 0x04, 0x00, 0x28, 0x03, 0x78, 0x00,
  0x00, 0x02, 0x00, 0x00, 0x40, 0x26, 0xc4, 0x00, 0x00, 0x04, 0x00, 0x79,
  0x05, 0xe4, 0xff, 0xf3, 0x14, 0x64, 0x04, 0x00, 0x28, 0x03, 0x78, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x50, 0x06, 0xc4, 0x00, 0x00, 0x00, 0x00, 0x79,
  0x05, 0xe4, 0xff, 0xf3, 0x14, 0x64, 0x04, 0x00, 0x28, 0x03, 0x78, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x50, 0x06, 0xc4, 0x00, 0x00, 0x00, 0x00, 0x79,
  0x05, 0xfc, 0xff, 0xf3, 0x14, 0x64, 0x04, 0x00, 0x28, 0x03, 0x78, 0x00,
  0xa0, 0x00, 0x00, 0x00, 0x50, 0x06, 0xb8, 0x01, 0x40, 0x00, 0x00, 0x13,
  0x8f, 0xfc, 0xff, 0xf3, 0x14, 0x64, 0x04, 0x00, 0x60, 0x11, 0x70, 0x00,
  0xc1, 0x00, 0x00, 0x80, 0xd0, 0x2a, 0xb8, 0x01, 0x82, 0x00, 0x00, 0xe8,
  0x15, 0x95, 0xff, 0xf3, 0x14, 0x64, 0x00, 0x00, 0x28, 0x01, 0x12, 0x00,
  0xe0, 0x00, 0x00, 0x00, 0x00, 0x01, 0xf4, 0x01, 0xc0, 0x00, 0x00, 0x87,
  0x55, 0x4c
};
const uint16_t PROGMEM logout_mp3_len = 1142;

const uint8_t PROGMEM noti_mp3[] = {
  0x49, 0x44, 0x33, 0x04, 0x00, 0x40, 0x00, 0x00, 0x00, 0x2b, 0x00, 0x00,
  0x00, 0x0c, 0x01, 0x20, 0x05, 0x01, 0x72, 0x62, 0x16, 0x2d, 0x54, 0x58,
  0x58, 0x58, 0x00, 0x00, 0x00, 0x15, 0x00, 0x00, 0x00, 0x53, 0x6f, 0x66,
  0x74, 0x77, 0x61, 0x72, 0x65, 0x00, 0x4c, 0x61, 0x76, 0x66, 0x35, 0x33,
  0x2e, 0x32, 0x30, 0x2e, 0x30, 0xff, 0xf3, 0x10, 0xc4, 0x00, 0x02, 0x80,
  0x06, 0x11, 0x50, 0x00, 0x44, 0x00, 0x03, 0x09, 0xc3, 0xdf, 0xfe, 0xef,
  0xff, 0x31, 0xef, 0xe8, 0x01, 0x89, 0xff, 0xff, 0xf3, 0x12, 0xc4, 0x03,
  0x01, 0xf0, 0x02, 0xb0, 0x61, 0x40, 0x18, 0x02, 0xf5, 0xbf, 0xf9, 0x40,
  0x41, 0xfa, 0x79, 0x03, 0xf8, 0xb7, 0xfc, 0xe0, 0x62, 0xf8, 0xff, 0xf3,
  0x10, 0xc4, 0x09, 0x04, 0x10, 0x6e, 0xe0, 0x01, 0x82, 0x10, 0x00, 0xc7,
  0x72, 0x87, 0x0e, 0x79, 0x47, 0x06, 0x15, 0x06, 0x0a, 0x0a, 0x80, 0x28,
  0xff, 0xf3, 0x10, 0xc4, 0x05, 0x04, 0x08, 0x8e, 0xf4, 0x01, 0x82, 0x50,
  0x00, 0xf7, 0xbf, 0xff, 0xea, 0x1f, 0x9c, 0x2f, 0x22, 0x9a, 0x1a, 0x4b,
  0x2a, 0x3c, 0xff, 0xf3, 0x10, 0xc4, 0x01, 0x02, 0xd8, 0x82, 0xe9, 0x81,
  0xc1, 0x48, 0x01, 0x00, 0x33, 0xe4, 0xff, 0xff, 0xff, 0xd4, 0x80, 0xd9,
  0x2a, 0x95, 0xfa, 0x18, 0xff, 0xf3, 0x10, 0xc4, 0x02, 0x02, 0x88, 0xbe,
  0xec, 0x00, 0x08, 0x0e, 0x51, 0xf9, 0xe0, 0x24, 0x9f, 0xff, 0xa8, 0xd1,
  0x87, 0xaa, 0x72, 0xb6, 0xa3, 0x1b, 0xff, 0xf3, 0x10, 0xc4, 0x04, 0x03,
  0x38, 0xc2, 0xf4, 0x00, 0x10, 0x0e, 0x6c, 0x56, 0xd0, 0x28, 0x77, 0xff,
  0x63, 0x4e, 0x06, 0x1b, 0x5f, 0x11, 0x05, 0xf6, 0xff, 0xf3, 0x10, 0xc4,
  0x04, 0x03, 0x88, 0xbe, 0xc4, 0x00, 0x28, 0x26, 0x50, 0x10, 0x38, 0xde,
  0x7f, 0xfc, 0x94, 0x40, 0x50, 0xe1, 0x58, 0xaa, 0x00, 0x09, 0xff, 0xf3,
  0x10, 0xc4, 0x02, 0x02, 0x58, 0xbe, 0xe8, 0x80, 0x01, 0xda, 0xe4, 0x0e,
  0x10, 0xea, 0x9f, 0xea, 0x95, 0x8a, 0xd5, 0x7e, 0xa1, 0xd0, 0x5b, 0xff,
  0xff, 0xf3, 0x12, 0xc4, 0x05, 0x02, 0x68, 0xbe, 0xcc, 0x00, 0x08, 0x20,
  0x18, 0xe7, 0x18, 0x4a, 0xc2, 0xaa, 0x1f, 0xd4, 0x31, 0xc8, 0x7f, 0xf3,
  0x26, 0x0e, 0x18, 0xff, 0xf3, 0x10, 0xc4, 0x09, 0x02, 0x90, 0xbe, 0xc0,
  0x00, 0x10, 0x24, 0x18, 0x3c, 0x6a, 0x0c, 0x1f, 0xc8, 0xc7, 0x3f, 0xfa,
  0x42, 0xa8, 0x43, 0x23, 0xf5, 0xff, 0xf3, 0x10, 0xc4, 0x0b, 0x02, 0x38,
  0xbe, 0xd4, 0x40, 0x08, 0x14, 0x18, 0x09, 0xeb, 0xff, 0xd2, 0x24, 0x84,
  0x2a, 0x01, 0xf3, 0x80, 0x03, 0x30, 0x32, 0xff, 0xf3, 0x10, 0xc4, 0x0f,
  0x02, 0x18, 0xbe, 0xc8, 0x20, 0x08, 0x14, 0x18, 0x2f, 0x77, 0xff, 0xf1,
  0x84, 0x55, 0xf2, 0x17, 0xc0, 0xe3, 0x7d, 0xdb, 0xf2, 0xff, 0xf3, 0x10,
  0xc4, 0x13, 0x02, 0xc8, 0xbe, 0xdc, 0x79, 0x40, 0x38, 0x00, 0x8f, 0x0e,
  0x47, 0xd5, 0x21, 0x00, 0x51, 0x7e, 0xbc, 0xce, 0x47, 0x15, 0xfb, 0xff,
  0xf3, 0x10, 0xc4, 0x14, 0x05, 0x48, 0xc2, 0xe8, 0x01, 0x85, 0x28, 0x00,
  0x0c, 0xaa, 0x36, 0x52, 0x85, 0x1b, 0xff, 0xff, 0xf3, 0x80, 0xd1, 0x6f,
  0xe4, 0xff, 0xf3, 0x10, 0xc4, 0x0b, 0x02, 0xd0, 0x86, 0xe8, 0x01, 0xc2,
  0x50, 0x00, 0x2a, 0xf9, 0xaf, 0x44, 0x13, 0xf2, 0x0b, 0xff, 0xf9, 0x53,
  0x55, 0x7a, 0x10, 0xff, 0xf3, 0x10, 0xc4, 0x0c, 0x02, 0x48, 0xbe, 0xd8,
  0x00, 0x08, 0x1a, 0x50, 0xba, 0xcd, 0xc1, 0x64, 0x91, 0xa2, 0x1a, 0x7f,
0x91, 0xaf, 0xf0, 0x7a, 0x5f, 0xff, 0xf3, 0x12, 0xc4, 0x0f, 0x05, 0x78,
0xbe, 0xec, 0xc8, 0x30, 0x4e, 0x4d, 0xff, 0xba, 0x8a, 0x42, 0x74, 0x53,
0x36, 0x3c, 0x00, 0x04, 0x1e, 0x81, 0x50, 0x0d, 0xff, 0xf3, 0x10, 0xc4,
0x07, 0x03, 0xa0, 0xbe, 0xf5, 0xa0, 0x38, 0x0e, 0x52, 0x67, 0xe2, 0x07,
0xa7, 0xfd, 0x62, 0x80, 0xfa, 0x0e, 0x00, 0x00, 0x34, 0x7c, 0xff, 0xf3,
0x10, 0xc4, 0x05, 0x03, 0x60, 0xbe, 0xf2, 0x40, 0x08, 0x0e, 0x52, 0xc3,
0xf1, 0x92, 0xd6, 0x7f, 0xfb, 0x82, 0x82, 0xea, 0x7e, 0xb1, 0x3c, 0xa3,
0xff, 0xf3, 0x10, 0xc4, 0x04, 0x02, 0x38, 0xbe, 0xcc, 0x00, 0x08, 0x20,
0x18, 0xff, 0xe7, 0xc2, 0x10, 0xad, 0x1f, 0xe4, 0x81, 0x6f, 0xff, 0x51,
0xb0, 0x25, 0xff, 0xf3, 0x10, 0xc4, 0x08, 0x02, 0x68, 0xc2, 0xc4, 0x00,
0x10, 0x1a, 0x18, 0x8f, 0x3a, 0x1f, 0xd4, 0x3b, 0x8a, 0x5f, 0xfe, 0x52,
0x0e, 0xc5, 0xf7, 0x1f, 0xff, 0xf3, 0x10, 0xc4, 0x0b, 0x02, 0x80, 0xc2,
0xb4, 0x00, 0x10, 0x1a, 0x18, 0xd4, 0x20, 0x86, 0x9f, 0xfd, 0x67, 0xc1,
0x28, 0x2f, 0xd5, 0x23, 0xf9, 0x0c, 0xff, 0xf3, 0x10, 0xc4, 0x0e, 0x02,
0x88, 0xbe, 0xb8, 0x00, 0x10, 0x1a, 0x18, 0xb3, 0xff, 0xdc, 0x30, 0x1b,
0x2a, 0x7e, 0xb1, 0xa5, 0xff, 0xfd, 0x30, 0x88, 0xff, 0xf3, 0x10, 0xc4,
0x10, 0x02, 0x08, 0xbe, 0xd0, 0x20, 0x08, 0x1a, 0x1a, 0x34, 0x7e, 0xa1,
0x2c, 0x2d, 0xff, 0xe7, 0x0f, 0x82, 0x88, 0x50, 0x7e, 0xa1, 0xff, 0xf3,
0x12, 0xc4, 0x14, 0x02, 0x00, 0xbe, 0xd0, 0x00, 0x08, 0x1a, 0x18, 0x26,
0x34, 0xff, 0xe7, 0x1c, 0x77, 0x06, 0xa7, 0x7e, 0xa1, 0x3c, 0x97, 0xff,
0xa2, 0xff, 0xf3, 0x10, 0xc4, 0x1a, 0x02, 0x80, 0xbe, 0xc4, 0x00, 0x08,
0x1a, 0x18, 0x09, 0x61, 0x0d, 0x1f, 0xdc, 0x4c, 0x48, 0xed, 0xff, 0xa9,
0x00, 0x89, 0x14, 0xff, 0xf3, 0x10, 0xc4, 0x1d, 0x02, 0x80, 0xc2, 0xc4,
0x00, 0x08, 0x1a, 0x18, 0x55, 0x1f, 0xd4, 0x0c, 0x49, 0x7f, 0xfb, 0x06,
0x12, 0x0a, 0x0d, 0xf6, 0x02, 0xff, 0xf3, 0x10, 0xc4, 0x20, 0x02, 0x28,
0xbe, 0xc8, 0x00, 0x08, 0x14, 0x18, 0xf0, 0xf9, 0xff, 0xfb, 0x01, 0x92,
0x5a, 0x33, 0x0c, 0x01, 0xf4, 0xfd, 0x40, 0xff, 0xf3, 0x10, 0xc4, 0x24,
0x02, 0x90, 0xbe, 0xb8, 0x00, 0x10, 0x1a, 0x18, 0x74, 0xff, 0xd7, 0xec,
0x2f, 0x77, 0xea, 0x0f, 0xf5, 0x03, 0xa0, 0x6e, 0x6f, 0xff, 0xf3, 0x10,
0xc4, 0x26, 0x02, 0x28, 0xbe, 0xb4, 0x00, 0x10, 0x1a, 0x18, 0x57, 0xec,
0x2c, 0xcf, 0x7f, 0xbf, 0xf5, 0xd5, 0x0f, 0x7e, 0xa0, 0x9b, 0x14, 0xff,
0xf3, 0x10, 0xc4, 0x2a, 0x02, 0x60, 0xbe, 0xac, 0x00, 0x28, 0x14, 0x18,
0x3f, 0xfd, 0x00, 0xfe, 0x1a, 0xec, 0x0f, 0xf6, 0xf5, 0x88, 0x39, 0x47,
0xff, 0xff, 0xf3, 0x10, 0xc4, 0x2d, 0x03, 0x40, 0xba, 0xb4, 0x50, 0x10,
0x14, 0x14, 0xd0, 0x15, 0x4a, 0x7f, 0xfd, 0x0a, 0x0f, 0xf5, 0x0b, 0x91,
0xdf, 0xeb, 0xd5, 0xff, 0xf3, 0x12, 0xc4, 0x2d, 0x03, 0x48, 0xc2, 0xa0,
0x00, 0x38, 0x1a, 0x18, 0x82, 0x68, 0x6f, 0xd7, 0x96, 0x26, 0x42, 0x00,
0xd5, 0xe9, 0xe8, 0x1e, 0x1d, 0xed, 0xff, 0xf3, 0x10, 0xc4, 0x2d, 0x02,
0xc0, 0xba, 0xa0, 0x00, 0x38, 0x1a, 0x18, 0xfa, 0xc8, 0x43, 0xbf, 0xff,
0xa2, 0x23, 0x03, 0x7e, 0x0f, 0x8b, 0x7f, 0xf5, 0xff, 0xf3, 0x10, 0xc4,
0x2f, 0x03, 0x18, 0xbe, 0xa0, 0x00, 0x38, 0x1a, 0x14, 0x8a, 0x03, 0xff,
0x85, 0x7d, 0xda, 0x6a, 0x06, 0xfa, 0xaf, 0xff, 0xd4, 0x5e, 0xff, 0xf3,
0x10, 0xc4, 0x2f, 0x03, 0x00, 0xbe, 0x9c, 0x00, 0x38, 0x14, 0x18, 0xdf,
0xfd, 0x60, 0xc0, 0x9d, 0xfd, 0x0e, 0xfc, 0x3b, 0x19, 0xff, 0xfa, 0x9d,
0xff, 0xf3, 0x10, 0xc4, 0x30, 0x03, 0xb0, 0xbe, 0xa4, 0x50, 0x28, 0x14,
0x14, 0xff, 0xf8, 0xd5, 0xff, 0xfe, 0x20, 0x2f, 0xd7, 0xff, 0xf8, 0x45,
0x1f, 0xd4, 0xff, 0xf3, 0x10, 0xc4, 0x2e, 0x03, 0x50, 0xc2, 0xa4, 0x28,
0x28, 0x0e, 0x1a, 0x13, 0xff, 0xe8, 0x10, 0x7f, 0xff, 0x84, 0x9f, 0xf8,
0x99, 0xbf, 0x87, 0x15, 0xff, 0xf3, 0x10, 0xc4, 0x2d, 0x03, 0xb8, 0xc2,
0x94, 0x00, 0x38, 0x14, 0x08, 0x2a, 0x1f, 0xf6, 0xfe, 0x59, 0xdf, 0x91,
0xfe, 0x77, 0xf5, 0xaa, 0x67, 0xf1, 0xff, 0xf3, 0x10, 0xc4, 0x2b, 0x03,
0xa1, 0x36, 0x98, 0x00, 0x38, 0x05, 0x40, 0x17, 0xf0, 0xd7, 0xea, 0x7f,
0xe2, 0x2f, 0xe1, 0xaa, 0x10, 0x00, 0xe0, 0x00, 0xff, 0xf3, 0x12, 0xc4,
0x29, 0x03, 0x90, 0xba, 0x90, 0x00, 0x10, 0x0a, 0x08, 0x57, 0xf4, 0x7f,
0xff, 0xd3, 0x00, 0x01, 0x30, 0x14, 0x7f, 0x56, 0xdf, 0xd2, 0x30, 0xff,
0xf3, 0x10, 0xc4, 0x28, 0x02, 0x88, 0x06, 0xa4, 0x28, 0x08, 0x44, 0x00,
0x3f, 0xff, 0xff, 0xff, 0x47, 0x5d, 0x30, 0x0d, 0xdf, 0xff, 0xff, 0xff,
0xd1, 0xff, 0xf3, 0x10, 0xc4, 0x2a, 0x02, 0xa0, 0x02, 0x88, 0x08, 0x00,
0x00, 0x02, 0x26, 0xfb, 0xd3, 0xa7, 0xa2, 0x9f, 0xff, 0xfe, 0xd0, 0x10,
0x62, 0x5a, 0x27, 0xff, 0xf3, 0x10, 0xc4, 0x2c, 0x02, 0x38, 0x02, 0x15,
0xf8, 0x00, 0x00, 0x00, 0xff, 0xd8, 0xbf, 0xff, 0xff, 0xe8, 0x4d, 0x00,
0x71, 0xf8, 0xfc, 0x7d, 0x9a, 0xff, 0xf3, 0x10, 0xc4, 0x30, 0x01, 0xf8,
0x06, 0x1e, 0x40, 0x00, 0x44, 0x00, 0x9d, 0xfb, 0xf3, 0x2a, 0xff, 0xff,
0xff, 0xff, 0xd4, 0x2f, 0xa9, 0x15, 0x09, 0xff, 0xf3, 0x10, 0xc4, 0x35,
0x01, 0xa8, 0x02, 0x14, 0x10, 0x00, 0x00, 0x00, 0xd0, 0x05, 0x12, 0xad,
0xcf, 0xff, 0xe9, 0xff, 0xff, 0xa5, 0x4c, 0x41, 0x4d, 0xff, 0xf3, 0x10,
0xc4, 0x3b, 0x01, 0xf8, 0x06, 0x0d, 0x40, 0x00, 0x00, 0x00, 0x45, 0x33,
0x2e, 0x39, 0x39, 0x2e, 0x35, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0xff,
0xf3, 0x10, 0xc4, 0x40, 0x02, 0xb0, 0x05, 0xf8, 0x00, 0x00, 0x00, 0x00,
0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55,
  0x55, 0xff, 0xf3, 0x10, 0xc4, 0x42, 0x01, 0xe8, 0x06, 0x08, 0x20, 0x00,
  0x00, 0x02, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55,
  0x55, 0x55, 0x55, 0xff, 0xf3, 0x12, 0xc4, 0x47, 0x02, 0x58, 0x02, 0x6a,
  0x40, 0x00, 0x44, 0x02, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55,
  0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0xff, 0xf3, 0x10, 0xc4, 0x4b, 0x01,
  0xd8, 0x02, 0x04, 0x00, 0x08, 0x00, 0x00, 0x55, 0x55, 0x55, 0x55, 0x55,
  0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0xff, 0xf3, 0x10, 0xc4,
  0x50, 0x02, 0xb0, 0x05, 0xfa, 0x00, 0x00, 0x00, 0x00, 0x55, 0x55, 0x55,
  0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55
};
const uint16_t PROGMEM noti_mp3_len = 1438;

#endif
```

Listing 13: Application/MP3.nc

```
#include <avr/pgmspace.h>
interface MP3
{
        /**
         * Start and stop sine test
         * @param on TRUE to start - FALSE to stop
         * @return SUCCESS if command was successfully sent over SPI
         */
        command error_t sineTest(bool on);

        /**
         * Set volume
         * @param volume Volume to be set
         * @return SUCCESS if command was successfully sent over SPI
         */
        command error_t setVolume(uint8_t volume);

        /**
         * Set Stream Mode
         * @return SUCCESS if command was successfully sent over SPI
         */
        command error_t setStreamMode(void);

        /**
         * Send data
         * @param data A pointer to a data buffer where the data is stored (IN PROGMEM)
         * @param len Length of the message to be sent
         * @return SUCCESS if request was granted and sending the data started
         */
        command error_t sendData(PGM_VOID_P data, uint16_t len);

        /**
         * Check if VS1011e is ready to accept new data
         * @return FALSE if VS1011e is busy or sending of data is in progress
         *      - otherwise TRUE
         */
        command bool isBusy(void);
}
```

Listing 14: Application/VS1011eC.nc

```
#include <ctype.h>
module VS1011eC {
        uses interface HplVS1011e;
        provides interface MP3;
}
#define MAX_PACKET_SIZE 32 /* Byte */
implementation {
        const uint8_t* dataBeginning; /* in PROGMEM */
        uint16_t dataLen;
        uint8_t packet[MAX_PACKET_SIZE];
        command error_t MP3.sineTest(bool on) {
                call HplVS1011e.reset();
                if(on) {
```

```
			static uint8_t magic[8] = {0x53,0xef,0x6e,0xcc,0x00,0x00,0x00,0x00};
			call HplVS1011e.writeRegister(MODE, (1<<11)|(1<<5)|(1<<6));
			return call HplVS1011e.sendData(magic, sizeof(magic));
			/*return call MP3.sendData(magic, sizeof(magic));*/
		}
		//return call HplVS1011e.writeRegister(MODE, (1 << 11)|(1<<6)); /* native mode */
		return SUCCESS;
	}
	uint8_t linearLoudnessFromSlider2(uint8_t value) {
		/* return 255 (1 - (1 - x)^4) where x = value/255 */
#if 0
		double x = value/255.0;
		return 255.0*(1 - (1 - x)*(1 - x)*(1 - x)*(1 - x));
#endif
		uint16_t tmp, result;
		uint16_t numerator = (255 - value);
		numerator *= numerator;
		tmp = numerator / (64*64);
		result = 255 - tmp*tmp;
		if(value < 5)
			result = 0; /* clamp because of rounding error */
		return result;
	}
	uint8_t linearLoudnessFromSlider(uint8_t value) {
		return linearLoudnessFromSlider2(value >> 3);
	}
	command error_t MP3.setVolume(uint8_t volume) {
		uint8_t temp = linearLoudnessFromSlider(0xff - volume);
		return call HplVS1011e.writeRegister(VOL, temp | (temp << 8));
	}
	command error_t MP3.setStreamMode(void) {
		return call HplVS1011e.writeRegister(MODE, (1 << 11) | (1 << 6));
	}
	task void sendMore() {
		atomic {
			if(dataLen > 0) {
				uint16_t packetLen = dataLen < MAX_PACKET_SIZE ? dataLen :
				    MAX_PACKET_SIZE;
				memcpy_P(packet, dataBeginning, packetLen);
				call HplVS1011e.sendData(packet, packetLen);
			}
		}
	}
	/* data is a PROGMEM address! */
	command error_t MP3.sendData(PGM_VOID_P data, uint16_t len) {
		atomic {
			dataBeginning = data;
			dataLen = len;
		}
		return post sendMore();
	}
	command bool MP3.isBusy(void) {
		return call HplVS1011e.isBusy();
	}
	async event void HplVS1011e.sendDone(error_t error) {
		if(error == SUCCESS) {
			dataBeginning += MAX_PACKET_SIZE;
			if(dataLen > MAX_PACKET_SIZE)
				dataLen -= MAX_PACKET_SIZE;
			else
				dataLen = 0;
		}
		post sendMore();
	}
}
```

Listing 15: Application/HplVS1011e.nc

```
#include "HplVS1011e.h"
```

```
interface HplVS1011e
{
        /**
         * Resets the VS1011e
         */
        command void reset(void);

        /**
         * Set a VS1011e register to a given value
         *
         * @param mp3Register Register to be written
         * @param mp3Cmd Command to be written
         * @return SUCCESS if command was successfully sent over SPI
         */
        command error_t writeRegister(mp3_reg_t mp3Register, uint16_t mp3Cmd);

        /**
         * Read a VS1011e register (only if MISO is attached and Ethernet disconnected)
         *
         * @param mp3Register Register to be read
         * @param value A pointer to data buffer where register content will be stored
         * @return SUCCESS if command was successfully sent over SPI
         */
        command error_t readRegister(mp3_reg_t mp3Register, uint16_t *value);

        /**
         * Send data to the VS1011e
         *
         * @param data A pointer to data buffer
         * @param len Length of message#
         * @return SUCCESS if request was granted and sending data started
         */
        command error_t sendData(uint8_t *data, uint16_t len);

        /**
         * Notification that sending data completed
         *
         * @param error SUCCESS if sending completed successfully
         */
        async event void sendDone(error_t error);

        /**
         * Test if VS1011e is ready to accept new data
         *
         * @return FALSE if ready to accept data
         */
        command bool isBusy(void);
}
```

Listing 16: Application/HplVS1011e.h

```
#ifndef __HPLVS1011E_H__
#define __HPLVS1011E_H__

typedef enum {
        OP_WRITE= 0x02,
        OP_READ = 0x03,
} opcode_t;

typedef enum {
        CLOCKF  = 0x03,
        MODE    = 0x00,
        VOL     = 0x0b,
} mp3_reg_t;

#endif
```

Listing 17: Application/HplVS1011eC.nc

```
module HplVS1011eC {
        uses interface SpiByte as DataPort;
        uses interface SpiControl as DataControl;
        uses interface SpiPacket as DataPacket;
        uses interface GeneralIO as RSTPort;
        uses interface GeneralIO as CSPort;
        uses interface GeneralIO as DREQPort; /* high: MP3 module can receive 32 Byte data */
        uses interface GeneralIO as BSYNCPort;
        uses interface HplAtm128Interrupt as DREQInterrupt;
        uses interface Resource;
        uses interface BusyWait<TMicro,uint16_t> as ResetWaiter;
        provides interface HplVS1011e;
        provides interface Init;
}
implementation {
        uint8_t* packetData;
        uint16_t packetLen;
        command bool HplVS1011e.isBusy() {
                return !call DREQPort.get();
        }
        event void Resource.granted() {
                //static uint8_t dummy[32] = {0};
                //call DataControl.setClock(SPI_SPEED_128);
                /* TODO when board is busy, release ? */
                call BSYNCPort.clr();
                call DataPacket.send(packetData, NULL/*dummy*/, packetLen);
                /* next see DataPacket.sendDone() */
        }
        async event void DataPacket.sendDone(uint8_t* txBuf, uint8_t* rxBuf, uint16_t len, error_t
            error) {
                call BSYNCPort.set();
                call Resource.release();
                signal HplVS1011e.sendDone(error);
        }
        async event void DREQInterrupt.fired() {
                if(call DREQPort.get()) { /* not busy */ /* this is clear from the configuration of
                    the interrupt */
                        call Resource.request();
                        call DREQInterrupt.disable();
                }
        }
        command error_t HplVS1011e.sendData(uint8_t *data, uint16_t len) {
                packetData = data;
                packetLen = len;
                if(!call DREQPort.get()) { /* busy */
                        call DREQInterrupt.enable();
                        return SUCCESS;
                } else {
                        return call Resource.request();
                }
        }
        command error_t HplVS1011e.writeRegister(mp3_reg_t addr, uint16_t data) {
                while(call HplVS1011e.isBusy())
                        ;
                if(call Resource.immediateRequest() == SUCCESS) {
                        //call DataControl.setClock(SPI_SPEED_128);
                        call CSPort.clr();
                        call DataPort.write(OP_WRITE);
                        call DataPort.write(addr);
                        call DataPort.write((uint8_t)((data>>8) & 0xff));
                        call DataPort.write((uint8_t)((data>>0) & 0xff));
                        call CSPort.set();
                        call Resource.release();
                        return SUCCESS;
                } else
                        return FAIL;
        }
        command error_t HplVS1011e.readRegister(mp3_reg_t addr, uint16_t *value) {
                while(call HplVS1011e.isBusy())
                        ;
                if(call Resource.immediateRequest() == SUCCESS) {
                        //call DataControl.setClock(SPI_SPEED_128);
                        call CSPort.clr();
                        call DataPort.write(OP_READ);
                        call DataPort.write(addr);
                        *value = call DataPort.write(0xFF) << 8;
                        *value |= call DataPort.write(0xFF);
                        call CSPort.set();
                        return SUCCESS;
                } else
                        return FAIL;
        }
        command error_t Init.init(void) {
                call CSPort.makeOutput();
                call CSPort.set();
                call RSTPort.makeOutput();
                call BSYNCPort.makeOutput();
                call BSYNCPort.set();
                call DREQPort.makeInput();
                call HplVS1011e.reset();
                return SUCCESS;
        }
        command void HplVS1011e.reset(void) {
                /* Hardware reset */
                call RSTPort.clr();
                /* wait 1 ms then set again */
                call ResetWaiter.wait(1000);
                call RSTPort.set();
                call DREQInterrupt.clear();
                call DREQInterrupt.edge(TRUE); /* rising edge */
                call HplVS1011e.writeRegister(CLOCKF, 12500);
                call HplVS1011e.writeRegister(MODE, (1 << 11)); /* native mode */
                call HplVS1011e.writeRegister(VOL, 0x6000); /* takes long? */
        }
}
```

A.4 ADC P5

Listing 18: Application/VolumeAdcC.nc

```
#include <stdint.h>
#include "buddy.h"
#include "Atm128Adc.h"

/* PF2 */

module VolumeAdcC {
        provides interface Read<uint16_t> as Read1;
        uses interface Read<uint16_t> as Read2;
        provides interface Atm128AdcConfig; /* used by AdcReadClientC */
        provides interface ResourceConfigure; /* used by AdcReadClientC */
}
implementation {
        async command uint8_t Atm128AdcConfig.getChannel() {
                return ATM128_ADC_SNGL_ADC2;
        }
        async command uint8_t Atm128AdcConfig.getRefVoltage() {
                return ATM128_ADC_VREF_AVCC;
        }
        async command uint8_t Atm128AdcConfig.getPrescaler() {
                return ATM128_ADC_PRESCALE_128;
        }
        async command void ResourceConfigure.configure() {
        }
        async command void ResourceConfigure.unconfigure() {
        }
        event void Read2.readDone(error_t result, uint16_t val) {
                signal Read1.readDone(result, val);
        }
        command error_t Read1.read() {
                return call Read2.read();
        }
}
```

A.5 MessageManager

Listing 19: Application/MessageManager.nc

```
/**
 * @brief Message interface used to exchange and manage chat messages
 */

#include "buddy.h"

interface MessageManager
{

        /**
         * @brief Signals an updated presence field
         * @details On receiving a presence stanza the destined
         * buddy will be updated and this event will be triggered.
         * @param buddyID A unique identifier
         *      between 0 and MAX_BUDDIES-1
         * @param A pointer to the updated buddy
         */
        event void presenceUpdated(uint8_t buddyID, buddy_t* buddy);

        /**
         * @brief Signals a subscription request from the given JID
         * @details The subscription request has to return the
         *      result immediately.
         *      A non-zero value indicates a granted request.
         * @param jid The jid of the requesting entity
         * @returns The result of the request, zero on denied.
         */
        event uint8_t subscriptionRequest(char* jid);

        /**
         * @brief Signals the reception of a new message
         * @param buddyID A unique identifier
         *      between 0 and MAX_BUDDIES-1
         * @param A pointer to the updated buddy
         */
        event void messageReceived(uint8_t buddyID, buddy_t* buddy);

        /**
         * @brief Sends the given message
         * @param If the transmitter is currently busy the request
         *      will be discarded and an appropriate value
         *      will be returned.
         * @param buddyID The unique buddy identifier,
         *      lower than MAX_BUDDIES
         * @param message A valid pointer to the message to send
         * @return The status of the operation
         */
        command error_t sendMessage(uint8_t buddyID, char* message);

        /**
         * @brief Returns the buddy from the list of known buddies
         * @details If the buddy ID is invalid NULL will be returned
         * @return The corresponding buddy
         */
        command buddy_t* getBuddy(uint8_t buddyID);

        /**
         * @brief Sends an acknowledgement for a presence subscription
         * @param destination_jid receiver of the acknowledgement
         * @return status of the operation
         */
        command error_t sendPresenceSubscriptionAck(const char* destination_jid);

        /**
         * @brief Marks the last message from buddy as seen.
         * @param buddy
         */
        command void markMessageSeen(buddy_t* buddy);
}
```

Listing 20: Application/buddy.h

```
/**
 * @file buddy.h
 * @brief The header file contains the buddy data type definition.
 */

#ifndef _BUDDY_H
#define _BUDDY_H

/** @brief The maximum length of JIDs */
#define MAX_JID_LENGTH (32)
/** @brief The number of buddies to store */
#define MAX_BUDDIES (4)
/**
 * @brief The length of each text message including the terminating
 * zero character
 */
#define MAX_MESSAGE_LENGTH (81)

#define INVALID_BUDDY_ID MAX_BUDDIES


/**
 * @brief The current status of the buddy
 * @details The buddy status empty indicates an empty buddy slot.
 * Empty buddies usually won't be propagated.
 */
typedef enum {
        BUDDY_STATUS_EMPTY = 0, BUDDY_STATUS_UNAVAILABLE,
        BUDDY_STATUS_AVAILABLE, BUDDY_STATUS_SUBSCRIBED
} buddy_status_t;

/**
 * @brief Defines a struct containing all relevant information to
 * manage a buddy
 */
typedef struct
{
        char jid[MAX_JID_LENGTH];
        /** The last message received */
        char message[MAX_MESSAGE_LENGTH];
        buddy_status_t status;
        uint8_t activity; /* 0 nonactive. higher: more active */
        bool requestsPresenceSubscription; /* whether the buddy wants to subscribe to our presence */
        bool hasPresenceSubscription; /* whether the buddy is subscribed to our presence */
        bool gotNewMessage; /* whether we got a message that the user (presumably) hasn't seen yet */
} buddy_t;


#endif
```

Listing 21: Application/MessageManagerC.nc

```
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>
```

```
#include "ip.h"
#include "printf.h"
#include "buddy.h"
#include "debug.h"
#include "udp_config.h"
#ifdef WLAN
#include "wlan.h"
#endif
#define MAX_DATA 700 /* Byte. Because of (possible) escaping, this can be at least six times larger
    than you would expect ("&apos;"). */

module MessageManagerC {
        provides interface MessageManager;
        uses interface Boot;
        uses interface UdpSend;
        uses interface UdpReceive;
        uses interface SplitControl as Control;
        uses interface IpControl;
        uses interface BufferedLcd as LCD2;
#ifdef WLAN
        uses interface WlanControl;
#endif
}
implementation {
        buddy_t buddies[MAX_BUDDIES]; /* by ID */
        buddy_status_t presenceStatus = BUDDY_STATUS_EMPTY; /* my own */
        int buddiesCount = 0;
        bool inited = FALSE;
        event void Boot.booted() {
                in_addr_t *ip;
#ifdef CUSTOM_IP_SETTINGS
                in_addr_t cip = { .bytes {IP_LOCAL}};
                in_addr_t cnm = { .bytes {NETMASK}};
                in_addr_t cgw = { .bytes {GATEWAY}};
                call IpControl.setIp(&cip);
                call IpControl.setNetmask(&cnm);
                call IpControl.setGateway(&cgw);
#endif
                ip = call IpControl.getIp();
#ifdef WLAN
                call WlanControl.setSSID(SSID);
                call WlanControl.setSecurityType(SECURITY_TYPE_WPA);
                call WlanControl.setPassphrase(WPA2_PASSPHRASE);
                call WlanControl.setWirelessMode(WIRELESS_MODE_INFRA);
#endif
                call Control.start();
        }

        event void Control.stopDone(error_t error) {

        }


        event void UdpSend.sendDone(error_t error) {
                debug("sendDone:␣%d", error);
        }

        const uint8_t* datachr(const uint8_t* s, uint8_t c) {
                for(; *s != 0; ++s)
                        if(*s == c)
                                return s;
                return NULL;
        }
        int datancmp(const uint8_t* s1, const char* s2, int len) {
                return strncmp((const char*) s1, s2, len);
        }
        int datalen(const uint8_t* s) {
                int result = 0;
                for(; *s != 0; ++s, ++result)
                        ;
        return result;
}
/* like strcpy */
void datacpy(uint8_t* dest, const char* src) {
        for(; *src != 0; ++src, ++dest)
                *dest = (uint8_t) *src;
}

/** finds the given XML Tag in haystack. Returns a pointer to the text right behind the tag
    name.
    Example result:
      <hello world="bar">
             ^
*/
const uint8_t* findXMLTag(const char* tag, const uint8_t* haystack) {
        int taglen = strlen(tag);
        if(haystack != NULL) while(haystack[0] != 0) {
                while(*haystack != 0 && *haystack != '<')
                        ++haystack;
                if(*haystack == 0)
                        break;
                ++haystack;
                if(datancmp(haystack, tag, taglen) == 0 && (isspace(haystack[taglen]) ||
                    haystack[taglen] == '>'))
                        return &haystack[taglen];
                else { /* false positive */
                        ++haystack;
                }
        }
        return NULL;
}
/** skips a quoted string.
    Example result:
      "foo"
           ^ */
const uint8_t* skipXMLQuotedString(const uint8_t* haystack) {
        uint8_t c = haystack[0];
        if(c == '"' || c == '\'') {
                ++haystack;
                while(haystack[0] != 0 && haystack[0] != c)
                        ++haystack;
                if(haystack[0] == c)
                        return ++haystack;
                else
                        return NULL;
        } else
                return NULL;
}
/** skips an attribute. If it encounters a '>', stops there.
    Example result:
     foo="bar"
              ^
 */
const uint8_t* skipXMLAttribute(const uint8_t* haystack) {
        while(isspace(*haystack))
                ++haystack;
        while(*haystack != 0 && *haystack != '=' && *haystack != '>')
                ++haystack;
        if(*haystack == '>')
                return haystack;
        if(*haystack == '=')
                ++haystack;
        return skipXMLQuotedString(haystack);
}
/** finds the given XML Attribute in haystack. Returns a pointer to the text right behind the
    equals sign after the attribute name.
    Assumes that haystack was retrieved by findXMLTag.
    Example result:
      bar="baz" baz="quoox"
                    ^
```

```
         */
        const uint8_t* getXMLAttribute(const char* name, const uint8_t* haystack) {
                int namelen = strlen(name);
                while(haystack != NULL && haystack[0] != 0 && haystack[0] != '>') {
                        while(isspace(haystack[0]))
                                ++haystack;
                        if(datancmp(haystack, name, namelen) == 0 && haystack[namelen] == '=')
                                return &haystack[namelen] + strlen("=");
                        else
                                haystack = skipXMLAttribute(haystack);
                }
                return NULL;
        }
        /** Gets the xml node's children.
            Assumes that haystack was retrieved by findXMLTag.
            Example result:
               bar="baz" baz="quoox">hello world
                                     ^
         */
        const uint8_t* getXMLChildren(const uint8_t* haystack) {
                if(haystack == NULL)
                        return NULL;
                while(*haystack != '>' && *haystack != 0)
                        haystack = skipXMLAttribute(haystack);
                /*for(; *haystack != 0 && *haystack != '>'; ++haystack)
                        ;*/
                if(*haystack != 0)
                        ++haystack;
                return haystack;
        }
        /** Given an XML Attribute Value
              "foo"
            checks whether it's equals to a given value. Note that value is just supposed to be
                something like
              foo
          */
        bool isXMLAttributeValueEqual(const char* value, const uint8_t* haystack) {
                if(haystack[0] == '"' || haystack[0] == '\'') {
                        int valuelen = strlen(value);
                        char c = haystack[0];
                        ++haystack;
                        return datancmp(haystack, value, valuelen) == 0 && haystack[valuelen] == c;
                } else
                        return FALSE;
        }
        /** copies text from src to destination, unescaping xml as we go. srclen does not include 0
            terminator.
            destination must have at least (srclen + 1) space. */
        void unescape_strpcpy(char* destination, const uint8_t* src, int srclen) {
#if 0
                memcpy(destination, src, srclen);
                destination[srclen] = 0;
#endif
                uint8_t c;
                for(; *src != 0 && srclen > 0; --srclen, ++destination) {
                        c = *src;
                        if(c == '&') {
                                ++src;
                                if(datancmp(src, "amp;", strlen("amp;")) == 0)
                                        c = (uint8_t) '&';
                                else if(datancmp(src, "lt;", strlen("lt;")) == 0)
                                        c = (uint8_t) '<';
                                else if(datancmp(src, "gt;", strlen("gt;")) == 0)
                                        c = (uint8_t) '>';
                                else if(datancmp(src, "apos;", strlen("apos;")) == 0)
                                        c = (uint8_t) '\'';
                                else if(datancmp(src, "quot;", strlen("quot;")) == 0)
                                        c = (uint8_t) '"';
                                else
                                        break;
                while(*src != (uint8_t) ';' && *src != 0)
                    ++src;
                if(*src != 0) /* skip ';' */
                    ++src;
            } else
                ++src;
            *destination = (char) c;
        }
        *destination = 0;
    }
    /** Extracts text node's text. destinationAllocation is the size of the destination space
      including the 0 byte.
      Example result (in destination):
        blah
     */
    error_t copyXMLText(const uint8_t* haystack, char* destination, size_t destinationAllocation)
      {
        const uint8_t* text = haystack;
        const uint8_t* textend;
        if(text == NULL)
            return FAIL;
        for(textend = text; *textend != (uint8_t) '<' && *textend != 0; ++textend)
            ;
        if(*textend != 0) {
            int textlen = textend - text;
            if(textlen <= destinationAllocation - 1) {
                unescape_strpcpy(destination, text, textlen);
                destination[textlen] = 0;
                return SUCCESS;
            }
        }
        return FAIL;
    }
    /** Extracts attribute's value. destinationAllocation is the size of the destination space
      including the 0 byte.
      Example result:
        blah
     */
    error_t copyXMLAttributeValue(const uint8_t* haystack, char* destination, size_t
      destinationAllocation) {
        const uint8_t* text = haystack;
        const uint8_t* textend;
        char c;
        if(text == NULL)
            return FAIL;
        c = text[0];
        if(c == '"' || c == '\'') {
            ++text;
            textend = datachr(text, c);
            if(textend != NULL) {
                int textlen = textend - text;
                if(textlen <= destinationAllocation - 1) {
                    unescape_strpcpy(destination, text, textlen);
                    return SUCCESS;
                }
            }
        }
        return FAIL;
    }

#ifdef WLAN
    event void WlanControl.lostConnection() {
        debug("lost␣connection");
    }
#endif

    uint8_t getBuddyByName(const char* name) {
        int i;
        int nameEndpos;
        if(name == NULL)
```

```
                return INVALID_BUDDY_ID;
        nameEndpos = strlen(name);
        if(nameEndpos > MAX_JID_LENGTH - 1)
                return INVALID_BUDDY_ID;
        for(i = 0; i < MAX_BUDDIES; ++i) {
                buddy_t* buddy = &buddies[i];
                if(buddy->status != BUDDY_STATUS_EMPTY) {
                        if(strcmp(name, buddy->jid) == 0)
                                return i;
                }
        }
        return INVALID_BUDDY_ID;
}
/* create a buddy. Must not exist beforehand. nameEtc is the part of the xml text where the
    buddy name starts.
   If there's no space left, drops oldest buddy. */
uint8_t createBuddy(const char* name) {
        int i;
        uint8_t min_activity = 0;
        uint8_t min_activity_i = 0;
        const char* nameEnd = strchr(name, '"');
        int nameEndpos = (nameEnd != NULL) ? nameEnd - name : strlen(name);
        if(nameEndpos > MAX_JID_LENGTH - 1)
                return INVALID_BUDDY_ID;
        for(i = 0; i < MAX_BUDDIES; ++i) {
                buddy_t* buddy = &buddies[i];
                if(buddy->status == BUDDY_STATUS_EMPTY) {
                        buddy->requestsPresenceSubscription = FALSE;
                        buddy->hasPresenceSubscription = FALSE;
                        buddy->gotNewMessage = FALSE;
                        buddy->activity = MAX_BUDDIES - 1; /* for now has maximum activity */
                        buddy->status = BUDDY_STATUS_UNAVAILABLE;
                        memcpy(buddy->jid, name, nameEndpos);
                        buddy->jid[nameEndpos] = 0;
                        memset(buddy->message, 0, MAX_MESSAGE_LENGTH);
                        return i;
                }
        }
        /* Note: here, everything is full */
        /* find the least active buddy */
        for(i = 0; i < MAX_BUDDIES; ++i) {
                buddy_t* buddy = &buddies[i];
                if(buddy->status != BUDDY_STATUS_EMPTY) {
                        if(buddy->activity < min_activity) {
                                min_activity = buddy->activity;
                                min_activity_i = i;
                        }
                }
        }
        /* Note: min_activity should have ended up being 0 */
        for(i = 0; i < MAX_BUDDIES; ++i) {
                buddy_t* buddy = &buddies[i];
                /* (activity - 1) modulus MAX_BUDDIES */
                if(i != min_activity_i)
                        --buddy->activity; /* scroll all the existing entries since now the "
                            most inactive" slot got free */
        }
        if(min_activity_i != -1) { /* should be */
                buddy_t* buddy = &buddies[min_activity_i];
                buddy->status = BUDDY_STATUS_UNAVAILABLE;
                buddy->requestsPresenceSubscription = FALSE;
                buddy->hasPresenceSubscription = FALSE;
                buddy->gotNewMessage = FALSE;
                buddy->activity = MAX_BUDDIES - 1; /* for now has maximum activity */
                memcpy(buddy->jid, name, nameEndpos);
                buddy->jid[nameEndpos] = 0;
                memset(buddy->message, 0, MAX_MESSAGE_LENGTH);
                return min_activity_i;
        } else
                return INVALID_BUDDY_ID;
```

```
}

/* ensures that the given buddy exists. */
uint8_t ensureBuddyExists(const char* name) {
        uint8_t buddyID = getBuddyByName(name);
        if(buddyID == INVALID_BUDDY_ID)
                buddyID = createBuddy(name);
        return buddyID;
}

command buddy_t* MessageManager.getBuddy(uint8_t buddyID) {
        return (buddyID >= MAX_BUDDIES) ? NULL : (buddies[buddyID].status !=
            BUDDY_STATUS_EMPTY) ? &buddies[buddyID] : NULL;
}
error_t sendXML(uint8_t* data) {
        static in_addr_t destination = { .bytes {IP_JABBER_GW}};
        return call UdpSend.send(&destination, UDP_REMOTE_PORT, data, datalen(data));
}
/* special printf version that does escaping.
   It supports the format specifiers:
        %as  for text inside attributes. The attribute is assumed to be delimited by single
            quotes, i.e. foo=’%as’.
        %ms  for text nodes. Example: <bar>textnode</bar>
   result_alloc is the size of the output buffer (including 0 terminator).
*/
int xvsnprintf(uint8_t* result, size_t result_alloc, const char* format, va_list args) {
        size_t result_sz = 0;
        uint8_t f;
        do {
                f = *format;
                if(f == ’%’) {
                        bool isXMLContent = FALSE;
                        bool isAttributeContent = FALSE;
                        ++format;
                        if(*format == ’a’) {
                                isXMLContent = TRUE;
                                isAttributeContent = TRUE;
                                ++format;
                        } else if(*format == ’m’) {
                                isXMLContent = TRUE;
                                ++format;
                        } else if(*format == ’%’) {
                                if(result_sz < result_alloc) {
                                        *result = f;
                                        ++result;
                                        ++result_sz;
                                }
                                ++format;
                                continue;
                        }

                        if(*format == ’s’) {
                                const char* arg = va_arg(args, const char*);
                                ++format;
                                if(isXMLContent) {
                                        for(; *arg != 0; ++arg) {
                                                uint8_t a = (uint8_t) *arg;
                                                switch(a) {
                                                case ’<’:
                                                        if(result_sz + strlen(”&lt;”) <
                                                            result_alloc) {
                                                                datacpy(result, ”&lt;”);
                                                                result += strlen(”&lt;”);
                                                                result_sz += strlen(”&lt;”);
                                                        }
                                                        break;
                                                case ’>’:
                                                        if(result_sz + strlen(”&gt;”) <
                                                            result_alloc) {
                                                                datacpy(result, ”&gt;”);
                                                result += strlen("&gt;");
                                                result_sz += strlen("&gt;");
                                        }
                                        break;
                                case '&':
                                        if(result_sz + strlen("&amp;") <
                                            result_alloc) {
                                                datacpy(result, "&amp;");
                                                result += strlen("&amp;");
                                                result_sz += strlen("&amp;");
                                        }
                                        break;
                                case '"':
                                        if(result_sz + strlen("&quot;") <
                                            result_alloc) {
                                                datacpy(result, "&quot;");
                                                result += strlen("&quot;");
                                                result_sz += strlen("&quot;");
                                        }
                                        break;
                                case '\'':
                                        if(result_sz + strlen("&apos;") <
                                            result_alloc) {
                                                datacpy(result, "&apos;");
                                                result += strlen("&apos;");
                                                result_sz += strlen("&apos;");
                                        }
                                        break;
                                default:
                                        /*if(isAttributeContent && a == '\'')
                                            {
                                                if(result_sz + strlen("&apos
                                                    ;") < result_alloc) {
                                                        datacpy(result, "&apos
                                                            ;");
                                                        result += strlen("&
                                                            apos;");
                                                        result_sz += strlen("&
                                                            apos;");
                                                }
                                                break;
                                        } else*/ if(result_sz < result_alloc)
                                            {
                                                *result = a;
                                                ++result;
                                                ++result_sz;
                                        }
                                }
                        }
                } else {
                        for(; *arg != 0; ++arg) {
                                if(result_sz < result_alloc) {
                                        *result = *arg;
                                        ++result;
                                        ++result_sz;
                                }
                        }
                }
        } else { /* unknown format specifier */
                errno = EINVAL;
                return -1;
        }
} else {
        if(result_sz < result_alloc) {
                *result = f;
                ++result;
                ++result_sz;
        }
        ++format;
}
```

```
        } while(f != 0);
        return result_sz;
}
int xsnprintf(uint8_t* result, size_t result_alloc, const char* format, ...) {
        int status;
        va_list args;
        va_start(args, format);
        status = xvsnprintf(result, result_alloc, format, args);
        va_end(args);
        return status;
}
command error_t MessageManager.sendPresenceSubscriptionAck(const char* destination_jid) {
        static uint8_t data[MAX_DATA];
        int status;
        if(destination_jid == NULL)
                return FAIL;
        debug("SENDING ACK");
        status = xsnprintf(data, MAX_DATA, "<presence from='%as@jmcvl.tilab.tuwien.ac.at' to
            ='%as@jmcvl.tilab.tuwien.ac.at' type='subscribed'></presence>", JABBER_USERNAME,
            destination_jid);
        if(status == -1)
                return FAIL;
        if(status >= MAX_DATA)
                return ESIZE;
        { /* Note: this assumes that the sending of the Ack succeeds */
                uint8_t buddy_id = getBuddyByName(destination_jid);
                buddy_t* buddy = call MessageManager.getBuddy(buddy_id);
                if(buddy != NULL) {
                        debug("RESETTING requestsPresenceSubscription");
                        buddy->requestsPresenceSubscription = FALSE;
                        buddy->hasPresenceSubscription = TRUE;
                }
        }
        return sendXML(data);
}
error_t login() {
        uint8_t data[MAX_DATA];
        int status = xsnprintf(data, MAX_DATA, "<presence></presence>");
        if(status == -1)
                return FAIL;
        if(status >= MAX_DATA)
                return ESIZE;
        return sendXML(data);
}
event void Control.startDone(error_t error) { /* Ethernet started */
        if(!inited) {
                inited = TRUE;
                login();
        }
}
command error_t MessageManager.sendMessage(uint8_t buddyID, char* text) {
        buddy_t* buddy = call MessageManager.getBuddy(buddyID);
        static uint8_t data[MAX_DATA];
        int status;
        if(buddy == NULL)
                return FAIL;
        /* if there is a presence subscription request from the contact, acknowledge it as
            soon as we send a message */
        if(buddy->requestsPresenceSubscription) {
                debug("requestsPresenceSubscription was set");
                call MessageManager.sendPresenceSubscriptionAck(buddy->jid);
        }
        status = xsnprintf(data, MAX_DATA, "<message from='%as@jmcvl.tilab.tuwien.ac.at' to='%
            as@jmcvl.tilab.tuwien.ac.at' type='chat'><body>%ms</body></message>",
            JABBER_USERNAME, buddy->jid, text);
        if(status == -1)
                return FAIL;
        if(status >= MAX_DATA)
                return ESIZE;
        return sendXML(data);
```

```
}
/*void debugL(char* data) {
        call LCD2.goTo(0,1);
        if(data == NULL)
                data = "?";
        else
                data[31] = 0;
        call LCD2.write(data);
}*/

/* "foo@bar" -> "foo" */
static void cutDomain(char* destination) {
        char* x = strchr(destination, '@');
        if(x != NULL)
                *x = 0;
}
event void UdpReceive.received(in_addr_t *srcIp, uint16_t srcPort, uint8_t *data, uint16_t len
    ) {
        const uint8_t* presence;
        const uint8_t* message;
        char sender_jid[MAX_JID_LENGTH + 30]; /* and domain */
        char receiver_jid[MAX_JID_LENGTH + 30]; /* and domain */
        uint8_t sender_id;
        buddy_t* sender_buddy;
        data[len] = 0;
        /*printf("%s", data);*/
        presence = findXMLTag("presence", data);
        message = findXMLTag("message", data);
        if(presence != NULL) {
                const uint8_t* type = getXMLAttribute("type", presence);
                if(copyXMLAttributeValue(getXMLAttribute("from", presence), sender_jid, sizeof
                    (sender_jid)) != SUCCESS)
                        return;
                if(copyXMLAttributeValue(getXMLAttribute("to", presence), receiver_jid, sizeof
                    (receiver_jid)) != SUCCESS)
                        return;
                if(strcmp(sender_jid, receiver_jid) != 0) { /* don't add myself */
                        cutDomain(sender_jid);
                        cutDomain(receiver_jid);
                        sender_id = ensureBuddyExists(sender_jid);
                        sender_buddy = call MessageManager.getBuddy(sender_id);
                        if(sender_buddy != NULL && strcmp(receiver_jid, JABBER_USERNAME) == 0)
                            {
                                if(type != NULL && isXMLAttributeValueEqual("subscribe", type)
                                    ) { /* someone else want's to subscribe to my status */
                                        uint8_t rstatus;
                                        debug("SETTING␣requestsPresenceSubscription");
                                        if(!sender_buddy->hasPresenceSubscription) {
                                                sender_buddy->requestsPresenceSubscription =
                                                    TRUE;
                                                sender_buddy->gotNewMessage = TRUE; /* to make
                                                     user look at "message" from system */
                                                rstatus = signal MessageManager.
                                                    subscriptionRequest(sender_buddy->jid);
                                        } else
                                                rstatus = 1;
                                        if(rstatus != 0) { /* wants to allow */
                                                call MessageManager.
                                                    sendPresenceSubscriptionAck(sender_buddy->
                                                    jid);
                                        }
                                } else if(type != NULL && isXMLAttributeValueEqual("
                                    unsubscribe", type)) {
                                        sender_buddy->hasPresenceSubscription = FALSE;
                                } else {
                                        if(type == NULL)
                                                sender_buddy->status = BUDDY_STATUS_AVAILABLE;
                                        else if(isXMLAttributeValueEqual("available", type))
                                                sender_buddy->status = BUDDY_STATUS_AVAILABLE;
                                        else if(isXMLAttributeValueEqual("unavailable", type))
                            sender_buddy->status =
                              BUDDY_STATUS_UNAVAILABLE;
                        signal MessageManager.presenceUpdated(sender_id,
                          sender_buddy);
                    }
                }
            }
        }
        if(message != NULL) {
            const uint8_t* body = findXMLTag("body", message);
            if(copyXMLAttributeValue(getXMLAttribute("from", message), sender_jid, sizeof(
              sender_jid)) != SUCCESS)
                return;
            if(copyXMLAttributeValue(getXMLAttribute("to", message), receiver_jid, sizeof(
              receiver_jid)) != SUCCESS)
                return;
            cutDomain(sender_jid);
            cutDomain(receiver_jid);
            sender_id = ensureBuddyExists(sender_jid);
            sender_buddy = call MessageManager.getBuddy(sender_id);
            if(body != NULL && sender_buddy != NULL && strcmp(receiver_jid,
              JABBER_USERNAME) == 0) {
                const uint8_t* text = getXMLChildren(body);
                if(copyXMLText(text, sender_buddy->message, MAX_MESSAGE_LENGTH) ==
                  SUCCESS) {
                    sender_buddy->gotNewMessage = TRUE;
                    signal MessageManager.messageReceived(sender_id, sender_buddy)
                      ;
                }
            }
        }
    }
    command void MessageManager.markMessageSeen(buddy_t* buddy) {
        buddy->gotNewMessage = FALSE;
    }
}
```

A.6 GLCD

Listing 22: Application/GlcdP.nc

```
/**
 * High level implementation for KS0108 Glcd
 * @author:     Markus Hartmann e988811@student.tuwien.ac.at
 * @date:       01.02.2012
 * Based on an implementation of Andreas Hagmann
 */

/*

   Pages:
   01234567
  |--------|
  |        |
  | Ctrl0  |  X-pos
  |        |   |
  |--------|   |
  |        |   |
  | Ctrl1  |   |-----Y-page
  |        |
  |--------|

*/

#include "KS0108.h"
#include "Standard5x7.h"

/*************** INTERNAL DEFINITIONS ***************/
```

```
#define KS0108_SET_PAGE              (0xB8)    // 10111XXX: set lcd page (X) address
#define KS0108_SET_ADDR              (0x40)    // 01YYYYYY: set lcd Y address
#define KS0108_XPIXELS               (128)
#define KS0108_CONTROLLER_XPIXELS    (64)
#define KS0108_YPAGES                (8)

/** Number of spaces that tabs (\t) is replaced with. */
#define TAB_WIDTH               (2)

typedef struct xy_point_signed_t{
  int16_t x, y;
} xy_point_signed;

module GlcdP
{
  provides interface Glcd;
  provides interface Init;
  uses interface HplKS0108 as Hpl;
}

implementation {
  /************* PROTOTYPES **********/
  void setAddress(const uint8_t x_pos, const uint8_t y_page);
  void plot4points(const xy_point c, const uint8_t x, const uint8_t y);
  void drawChar(const char c, const xy_point p, const font* f);

  command error_t Init.init()
  {
    error_t ret = call Hpl.init();
    return ret;
  }

  command error_t Glcd.setPixel(const uint8_t x, const uint8_t y)
  {
    uint8_t data;
    uint8_t x_pos = x;
    /* y_page = y/8 */
    uint8_t y_page = (y>>3);
    uint8_t controller = ((x >> 6) & 0x01);
    controller ^= 1;

    setAddress(x_pos, y_page);
    data = call Hpl.dataRead(controller);
    data |= (1 << (y & 7));
    setAddress(x_pos, y_page);
    call Hpl.dataWrite(controller, data);

    return SUCCESS;
  } /* END setPixel */

  command error_t Glcd.clearPixel(const uint8_t x, const uint8_t y)
  {
    uint8_t data;
    uint8_t x_pos = x;
    /* y_page = y/8 */
    uint8_t y_page = (y>>3);
    uint8_t controller = ((x >> 6) & 0x01);
    controller ^= 1;

    setAddress(x_pos, y_page);
    data = call Hpl.dataRead(controller);
    data &= ~(1 << (y & 7));
    setAddress(x_pos, y_page);
    call Hpl.dataWrite(controller, data);

    return SUCCESS;
  } /* END clearPixel */

  command error_t Glcd.invertPixel(const uint8_t x, const uint8_t y)
  {
```

```
  uint8_t data;
  uint8_t x_pos = x;
  /* y_page = y/8 */
  uint8_t y_page = (y>>3);
  uint8_t controller = ((x >> 6) & 0x01);
  controller ^= 1;

  setAddress(x_pos, y_page);
  data = call Hpl.dataRead(controller);
  data ^= (1 << (y & 7));
  setAddress(x_pos, y_page);
  call Hpl.dataWrite(controller, data);

  return SUCCESS;
} /* END invertPixel */

command error_t Glcd.fill(uint8_t pattern)
{
  uint8_t x_pos;
  uint8_t y_page;

  for (y_page = 0; y_page < KS0108_YPAGES; y_page++){
    setAddress(0, y_page);
    setAddress(KS0108_CONTROLLER_XPIXELS, y_page);
    for (x_pos = 0; x_pos < KS0108_XPIXELS; x_pos++){
      call Hpl.dataWrite(0, pattern);
      call Hpl.dataWrite(1, pattern);
    }
  }
  return SUCCESS;
} /* END fill */

command error_t Glcd.drawLine(const uint8_t x1,const uint8_t y1,const uint8_t x2,const uint8_t y2)
{
  xy_point p1;
  xy_point p2;

  int16_t err, e2;
  xy_point_signed d, s;
  xy_point px;

  p1.x = x1;
  p1.y = y1;
  p2.x = x2;
  p2.y = y2;

  px.x = p1.x; px.y = p1.y;
  d.x = p2.x - p1.x;
  d.y = p2.y - p1.y;
  if(d.x < 0)
    d.x = -d.x;
  if(d.y < 0)
    d.y = -d.y;

  s.x = -1;
  if(p1.x < p2.x)
    s.x = 1;

  s.y = -1;
  if(p1.y < p2.y)
    s.y = 1;

  err = d.x - d.y;

  while(1){
    call Glcd.setPixel(px.x, px.y);

    if((px.x == p2.x) && (px.y == p2.y))
      break;
    e2 = err<<1;

    if(e2 > -d.y){
      err -= d.y;
      px.x += s.x;
    }
    if(e2 < d.x){
      err += d.x;
      px.y += s.y;
    }
  }
  return SUCCESS;
} /* END drawLine */

command error_t Glcd.drawRect(const uint8_t x1,const uint8_t y1,const uint8_t x2,const uint8_t y2)
{
  uint8_t dx, dy, x;
  xy_point p1;
  xy_point p2;

  p1.x = x1;
  p1.y = y1;
  p2.x = x2;
  p2.y = y2;

  dx = (p2.x > p1.x ? 1 : -1);
  dy = (p2.y > p1.y ? 1 : -1);

  for(x = p1.y + dy; x != p2.y; x += dy){
    call Glcd.setPixel(p1.x, x);
    call Glcd.setPixel(p2.x, x);
  }

  x = p1.x;

  while(1){
    call Glcd.setPixel(x, p1.y);
    call Glcd.setPixel(x, p2.y);

    if(x == p2.x)
      break;

    x += dx;
  }
  return SUCCESS;
} /* END drawRect */

command error_t Glcd.drawEllipse(const uint8_t x, const uint8_t y, const uint8_t radius_h, const
    uint8_t radius_v)
{
  int16_t sqRx, sqRy;
  xy_point p;
  xy_point c;
  int16_t  chgX, chgY;
  int16_t stopX, stopY, error;

  c.x = x;
  c.y = y;

  sqRx = radius_h * radius_h;
  sqRy = radius_v * radius_v;
  chgX = sqRy;
  chgY = sqRx;

  sqRx <<= 1;
  sqRy <<= 1;

  chgX *= (1 - (radius_h<<1));

  stopX = sqRy * radius_h;
  stopY = 0;

  p.x = radius_h;  p.y = 0;
  error = 0;

  while(stopX >= stopY){
    plot4points(c, p.x, p.y);

    ++p.y;
    stopY += sqRx;
    error += chgY;
    chgY  += sqRx;

    if(((error << 1) + chgX) > 0){
      --p.x;
      stopX -= sqRy;
      error += chgX;
      chgX  += sqRy;
    }
  }

  chgX = (sqRy>>1);
  chgY = (sqRx>>1) * (1 - (radius_v<<1));
  stopX = 0;
  stopY = sqRx * radius_v;

  p.x = 0;     p.y = radius_v;
  error = 0;

  while(stopX <= stopY){
    plot4points(c, p.x, p.y);

    ++p.x;
    stopX += sqRy;
    error += chgX;
    chgX  += sqRy;

    if(((error << 1) + chgY) > 0){
      --p.y;
      stopY -= sqRx;
      error += chgY;
      chgY  += sqRx;
    }
  }
  return SUCCESS;
} /* END drawEllipse */

command void Glcd.drawText(const char *text, const uint8_t x, const uint8_t y)
{
  const font* f = &Standard5x7;
  xy_point p;
  xy_point px;
  p.x = x;
  p.y = y;
  px = p;

  for(; *text != 0; ++text){
    if((*text == '\n') || (*text == '\r')){
      px.x = p.x;
      px.y += f->lineSpacing;
    } else if(*text == '\t'){
      px.x += TAB_WIDTH * f->width;
    } else {
      if(px.x + f->charSpacing > KS0108_XPIXELS) { /* doesn't fit. wrap. */
        px.x = p.x;
        px.y += f->lineSpacing;
      }
      if(px.y >= KS0108_YPAGES*8) { /* doesn't fit. */
        break;
      }
```

```
        drawChar(*text, px, f);
        px.x += f->charSpacing;
      }
    }
  } /* END drawText */

  /************ PRIVATE ************/

  void drawChar(const char c, const xy_point p, const font* f)
  {
    const uint8_t *cpointer = (f->font) + ((c - f->startChar) * (f->width));
    uint8_t cp, cv, i;
    xy_point px = p;

    if((c < f->startChar) || (c > f->endChar))
      return;

    for(cp = 0; cp < 5; ++cp){
      px.y = p.y - f->height;

      cv = pgm_read_byte(cpointer);

//      while(cv > 0){
        for(i=0; i<7;++i){
        if((cv & 1) != 0){
          call Glcd.setPixel(px.x, px.y);
        } else {
          call Glcd.clearPixel(px.x, px.y);
        }

        cv >>= 1;
        ++px.y;
      }

      ++px.x;
      ++cpointer;
    }
  } /* END drawChar */

  void plot4points(const xy_point c, const uint8_t x, const uint8_t y)
  {
    call Glcd.setPixel(c.x + x, c.y + y);
    if(x != 0)
      call Glcd.setPixel(c.x - x, c.y + y);
    if(y != 0)
      call Glcd.setPixel(c.x + x, c.y - y);

    call Glcd.setPixel(c.x - x, c.y - y);
  } /* END plot4points */

  void setAddress(const uint8_t x_pos, const uint8_t y_page)
  {
    /* x < 64 write to controller 1 - else 0 */
    uint8_t controller = (x_pos >> 6);
    controller ^= 1;
    call Hpl.controlWrite(controller, KS0108_SET_ADDR|(x_pos & 0x3F));
    call Hpl.controlWrite(controller, KS0108_SET_PAGE|(y_page & 0x07));
  } /* END setAddress */

}
```

A.7 TCP/IP Stack

Listing 23: Application/PingP.nc

```
/**
 * @author:       Andreas Hagmann <ahagmann@ecs.tuwien.ac.at>
 * @date:         12.12.2011
 *
 * based on an implementation of Harald Glanzer, 0727156 TU Wien
 */

module PingP {
        uses interface IcmpReceive;
        uses interface IcmpSend;
}

implementation {
        uint8_t srcData[522];
        uint16_t srcDataLen;
        in_addr_t srcIp;
        bool sendingPong = FALSE;
        task void sendPong() {
                call IcmpSend.send(&srcIp, 0, 0, srcData, srcDataLen); // packet->header.len - sizeof(
                    icmp_header_t));
        }
        event void IcmpReceive.received(in_addr_t *xsrcIp, uint8_t code, uint8_t *data, uint16_t len)
            { /* received an ICMP echo request */
                if(!sendingPong) {
                        /* send an ICMP echo reply */
                        sendingPong = TRUE;
                        memcpy(srcData, data, len);
                        srcDataLen = len;
                        memcpy(&srcIp, xsrcIp, sizeof(*xsrcIp));
                        post sendPong();
                }
        }
        event void IcmpSend.sendDone(error_t error) {
                sendingPong = FALSE;
        }
}
```

Listing 24: Application/UdpTransceiverP.nc

```
/**
 * @author:      Andreas Hagmann <ahagmann@ecs.tuwien.ac.at>
 * @date:        12.12.2011
 *
 * based on an implementation of Harald Glanzer, 0727156 TU Wien
 */

#include ”udp.h”

module UdpTransceiverP {
        provides interface PacketSender<udp_queue_item_t>;
        provides interface UdpReceive[uint16_t port];
        uses interface IpSend;
        uses interface IpReceive;
        uses interface IcmpSend;
        uses interface IpPacket;
}

implementation {
        udp_packet_t packet;
        /* for unreachable */
        in_addr_t rsrcIp;
        uint16_t rsrcPort;
        uint8_t rdata[8 + sizeof(ip_header_t) + 4] = {0}; /* RFC 792 */
        bool sendingResponse = FALSE;

        command error_t PacketSender.send(udp_queue_item_t *item) {
                // create udp packet

                packet.header.srcPort = item->srcPort;
                packet.header.dstPort = item->dstPort;
                packet.header.len = item->dataLen + sizeof(udp_header_t);
                memcpy(&(packet.data), item->data, item->dataLen);
                return call IpSend.send(&(item->dstIp), (uint8_t*)&(packet), packet.header.len);
        }

        task void sendUnreachable() {
                call IcmpSend.send(&rsrcIp, 3, 3, rdata, sizeof(rdata));
        }
        default event void UdpReceive.received[uint16_t port](in_addr_t *srcIp, uint16_t srcPort,
            uint8_t *data, uint16_t len) { /* default event handler if we do not know what to do with
            this UDP packet */
                ip_packet_t* rpacket;
                if(!sendingResponse) {
                        memcpy(&rsrcIp, srcIp, sizeof(*srcIp));
                        rsrcPort = srcPort;
                        rpacket = call IpPacket.getPacket();
                        if(rpacket != NULL) {
                                memcpy(rdata + 4, rpacket, sizeof(rdata) - 4);
                                sendingResponse = TRUE;
                                post sendUnreachable();
                        }
                }
        }

        event void IcmpSend.sendDone(error_t error) {
                sendingResponse = FALSE;
        }

        event void IpReceive.received(in_addr_t *srcIp, uint8_t *data, uint16_t len) {
                udp_packet_t *p = (udp_packet_t*)data;

                signal UdpReceive.received[p->header.dstPort](srcIp, p->header.srcPort, (uint8_t*)&(p
                    ->data), p->header.len - sizeof(udp_header_t));
        }

        event void IpSend.sendDone(error_t error) {
                signal PacketSender.sendDone(error);
        }
}
```