```
/*
 * XRF Firmware Uploader 1.0 - a tool to upload new firmware on XRF
 *                             modules from Ciseco (www.ciseco.co.uk)
 *
 * Copyright (C) 2012  Marc Eberhard (eberhardma@googlemail.com)
 * Tweaked by Matt Lloyd <dps.lwk at gmail.com> to work on OSX
 *
 * For all XRF related question visit the forum at http://
openmicros.org
 *
 * This program is free software: you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/
>.
 *
 */

#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <termios.h>
#include <iostream>

const char* default_device_filename = "/dev/ttyUSB0";
const int   default_baud_rate       = 9600; // baud
const long  default_timeout         =   10; // seconds

struct firmware
{
    char* line;
    firmware* next;
};

void print_usage(char* binary_name)
{
    printf("Usage: %s <options>\n"
```

```
           "          -b <baud rate>         (default %d baud)\n"
           "          -d <device filename>   (default %s)\n"
           "          -f <firmware filename> (required)\n"
           "          -h                     (print this help message)
\n"
           "          -t <timeout>           (default %ld seconds)\n"
           "          -p                     (bypass AT mode, used for
device stuck in bootloader mode)\n",
     binary_name, default_baud_rate, default_device_filename,
default_timeout);
}

void send_bytes(int fd, const char* bytes)
{
    int bytes_out = write(fd, bytes, strlen(bytes));

    if (bytes_out < 0)
    {
        int e = errno;
        printf("Could not write to device: %s\n", strerror(e));

        exit(1);
    }
    else if (bytes_out < (int) strlen(bytes))
    {
        printf("Write returned %d bytes instead of %d.\n", bytes_out,
(int) strlen(bytes));

        exit(1);
    }

}

bool is_data_available(int fd, long timeout)
{
    struct timeval tv;
    tv.tv_sec  = timeout;
    tv.tv_usec = 0;
    fd_set rfds;
    FD_ZERO(&rfds);
    FD_SET(fd, &rfds);

    int retval = select(fd + 1, &rfds, NULL, NULL, &tv);

    if (retval < 0)
    {
        int e = errno;
        printf("Could not read from device: %s\n", strerror(e));

        exit(1);
```

```
    }
    else if (retval > 0)
    {
        return true;
    }

    return false;
}

char receive_byte(int fd, long timeout)
{

    if (!is_data_available(fd, timeout))
    {
        printf("Timeout, no data received within %ld seconds\n",
timeout);

        exit(1);
    }

    char c_in;
    int bytes_in = read(fd, &c_in, 1);

    if (bytes_in > 1)
    {
        printf("Device returned %d bytes instead of 1.\n", bytes_in);

        exit(1);
    }
    else if (bytes_in == 0)
    {
        printf("No data available from device\n");

        exit(1);
    }
    else if (bytes_in < 0)
    {
        int e = errno;
        printf("Could not read from device: %s\n", strerror(e));

        exit(1);
    }

    return c_in;
}

void flush_device(int fd, long timeout)
{

    while (is_data_available(fd, 0))
```

```
    {
        receive_byte(fd, timeout);
    }

}

void receive_response(int fd, char* response, int len, long timeout)
{

    for (int i = 0; i < len; i++)
    {
       response[i] = 0;
    }

    for (int i = 0; i < len; i++)
    {
        char c_in = receive_byte(fd, timeout);

        if (c_in == '\r')
        {
            return;
        }

        response[i] = c_in;
    }

}

void enter_command_modus(int fd, long timeout)
{
    printf("<> Entering command modus\n");
    send_bytes(fd, "+++");
    char response[3];
    receive_response(fd, response, 3, timeout);
    printf("<- %s\n", response);

    if (strncmp(response, "OK", 2) != 0)
    {
        printf("Invalid response received\n");

        exit(1);
    }

}

void execute_command(int fd, const char* command, const char*
expected_response, long timeout)
{
    const int len = strlen(expected_response);
    printf("-> %s\n", command);
```

```
    send_bytes(fd, command);
    send_bytes(fd, "\r");
    char* response = new char[len + 1];
    receive_response(fd, response, len + 1, timeout);
    printf("<- %s\n", response);

    if (strncmp(response, expected_response, len) != 0)
    {
        printf("Invalid response received\n");

        exit(1);
    }

    delete[] response;
}

void info_command(int fd, const char* command, int lines, long
timeout)
{
    printf("-> %s\n", command);
    send_bytes(fd, command);
    send_bytes(fd, "\r");

    for (int i = 0; i < lines; i++)
    {
        char response[128];
        receive_response(fd, response, 128, timeout);
        printf("<- %s\n", response);
    }

}

int main(int argc, char** argv)
{
    int         baud_rate         = default_baud_rate;
    const char* device_filename   = default_device_filename;
    const char* firmware_filename = 0;
    long        timeout           = default_timeout;
    int         c;
    bool bypass = false;

    opterr = 0;

    while ((c = getopt (argc, argv, "b:d:f:ht:p")) != -1)
    {

        switch (c)
        {

            case 'b':
```

```
            {
                baud_rate = atoi(optarg);

                if (baud_rate != 9600)
                {
                    printf("Reset device to 9600 baud for
reprogramming!\n");

                    return 1;
                }

                break;
            }

            case 'd':
            {
                device_filename = optarg;

                break;
            }

            case 'f':
            {
                firmware_filename = optarg;

                break;
            }

            case 'h':
            {
                print_usage(argv[0]);

                return 1;
            }

            case 't':
            {
                timeout = atol(optarg);

                break;
            }

            case 'p':
            {
                 bypass = true;

                break;
            }

            default:
```

```
            {
                print_usage(argv[0]);

                return 1;
            }

        }

    }

    if (!firmware_filename)
    {
        printf("Required argument firmware filename missing\n\n");
        print_usage(argv[0]);

        return 1;
    }

    if (optind < argc)
    {
        printf("Too many arguments given\n\n");
        print_usage(argv[0]);

        return 1;
    }

    printf("Writing new firmware file %s to device %s with baud rate
%d...\n",
     firmware_filename, device_filename, baud_rate);
    printf("Reading firmware file...\n");
    FILE* fd_firmware = fopen(firmware_filename, "r");

    if (!fd_firmware)
    {
        int e = errno;
        printf("Could not open firmware file %s for reading: %s\n",
firmware_filename, strerror(e));

        return 1;
    }

    int firmware_lines = 0;
    firmware* firmware_data = 0;
    firmware* last_data = 0;
    char new_line[80];

    while (fscanf(fd_firmware, "%s[^\n]", new_line) > 0)
    {
        int len = strlen(new_line);
```

```
        if (len != 67)
        {
            printf("Length with invalid length of %d in firmware
file\n", len);

            return 1;
        }

        firmware_lines++;
        firmware* new_data = new firmware;
        new_data->line = new char[len + 1];
        strcpy(new_data->line, new_line);
        new_data->next = 0;

        if (!firmware_data)
        {
            firmware_data = new_data;
        }

        if (last_data)
        {
            last_data->next = new_data;
        }

        last_data = new_data;
    }

    fclose(fd_firmware);
    printf("Read %d lines from firmware file\n", firmware_lines);
    printf("Opening device...\n");
    int fd_device = open(device_filename, O_RDWR);

    if (fd_device == -1)
    {
        int e = errno;
        printf("Could not open device %s for communication: %s\n",
device_filename, strerror(e));

        return 1;
    }

    printf("Setting serial parameters...\n");
    struct termios new_termios;
    tcgetattr(fd_device, &new_termios);

    switch (baud_rate)
    {

        case 300:
        {
```

```
    cfsetispeed(&new_termios,B300);
                cfsetospeed(&new_termios,B300);
    break;
}

case 600:
{
    cfsetispeed(&new_termios,B600);
                cfsetospeed(&new_termios,B600);
    break;
}

case 1200:
{
    cfsetispeed(&new_termios,B1200);
                cfsetospeed(&new_termios,B1200);
    break;
}

case 2400:
{
    cfsetispeed(&new_termios,B2400);
                cfsetospeed(&new_termios,B2400);
    break;
}

case 4800:
{
    cfsetispeed(&new_termios,B4800);
                cfsetospeed(&new_termios,B4800);
    break;
}

case 9600:
{
    cfsetispeed(&new_termios,B9600);
                cfsetospeed(&new_termios,B9600);
    break;
}

case 19200:
{
    cfsetispeed(&new_termios,B19200);
                cfsetospeed(&new_termios,B19200);
    break;
}

case 38400:
{
    cfsetispeed(&new_termios,B38400);
```

```
                    cfsetospeed(&new_termios,B38400);
        break;
    }

    case 57600:
    {
        cfsetispeed(&new_termios,B57600);
                    cfsetospeed(&new_termios,B57600);
        break;
    }

    case 115200:
    {
        cfsetispeed(&new_termios,B115200);
                    cfsetospeed(&new_termios,B115200);
        break;
    }

    case 230400:
    {
        cfsetispeed(&new_termios,B230400);
                    cfsetospeed(&new_termios,B230400);
        break;
    }

    default:
    {
        printf("Unsurported baud rate %d\n", baud_rate);

        return 1;
    }

}

cfmakeraw(&new_termios);
new_termios.c_oflag &= ~(ONLCR | OCRNL | ONLRET | ONOCR);
tcsetattr(fd_device, TCSANOW, &new_termios);
struct termios cmp_termios;
tcgetattr(fd_device, &cmp_termios);

if (new_termios.c_cflag != cmp_termios.c_cflag)
{
    printf("Could not set serial communication parameters\n");

    return 1;
}

printf("Waiting for device to settle...\n\n");
sleep(2);
flush_device(fd_device, timeout);
```

```
if (!bypass) {
     enter_command_modus(fd_device, timeout);
     info_command(fd_device, "ATVR", 2, timeout);
     execute_command(fd_device, "ATPG", "OK", timeout);
} else {
     printf("Bypassing AT mode");
}
usleep(100000);
send_bytes(fd_device, "~Y");
char c_in = receive_byte(fd_device, timeout);

if (c_in != '3')
{
    printf("Invalid response received\n");

    exit(1);
}

send_bytes(fd_device, "W");
c_in = receive_byte(fd_device, timeout);

if (c_in != 'W')
{
    printf("Invalid response received\n");

    exit(1);
}

firmware* data = firmware_data;
int lines = 0;

while (data != 0)
{
    c_in = receive_byte(fd_device, timeout);

    if (c_in != 'R')
    {
        printf("Invalid response received\n");

        exit(1);
    }

    send_bytes(fd_device, data->line);
    c_in = receive_byte(fd_device, timeout);

    if (c_in != 'A')
    {
        printf("Invalid response received\n");

        exit(1);
```

```
        }

        data = data->next;
        lines++;
        printf("<> Sent %d of %d lines...\r", lines, firmware_lines);
        fflush(stdout);
    }

    printf("\n");
    c_in = receive_byte(fd_device, timeout);

    if (c_in != 'R')
    {
        printf("Invalid response received\n");

        exit(1);
    }

    c_in = receive_byte(fd_device, timeout);

    if (c_in != 'y')
    {
        printf("Invalid response received\n");

        exit(1);
    }

    send_bytes(fd_device, "X");
    printf("\nAll OK, XRF successfully reprogrammed!\n\n");
    printf("Waiting for device to settle...\n\n");
    sleep(2);
    flush_device(fd_device, timeout);
    enter_command_modus(fd_device, timeout);
    info_command(fd_device, "ATVR", 2, timeout);
    close(fd_device);

    return 0;
}
```