VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

BEZDRÁTOVÉ OVLÁDÁNÍ POČÍTAČE PC

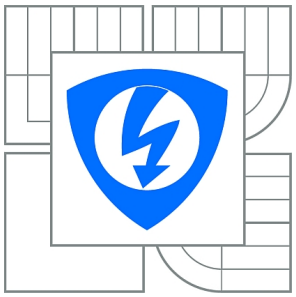
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. JIŘÍ MACHÁČEK
AUTHOR

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

BEZDRÁTOVÉ OVLÁDÁNÍ POČÍTAČE PC

WIRELESS PC CONTROLL

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

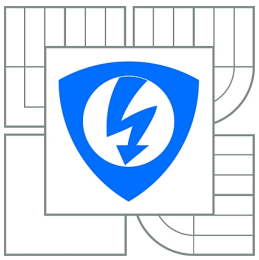
AUTOR PRÁCE — Bc. JIŘÍ MACHÁČEK

AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ MACHO, Ph.D.

SUPERVISOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student:	Bc. Jiří Macháček	***ID:***	106608
Ročník:	2	***Akademický rok:***	2011/2012

NÁZEV TÉMATU:

Bezdrátové ovládání počítače PC

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s problematikou vzdáleného ovládání počítače PC prostřednictvím bezdrátové komunikace.
2. Navrhněte koncepci vzdáleného ovládání počítače PC prostřednictvím bezdrátové sítě. Nakreslete blokové schéma systému.
3. Navrhněte obvodové řešení bezdrátového ovládání, nakreslete schéma zapojení a vypočtěte hodnoty jednotlivých součástek.
4. Navrhněte desku plošných spojů a vytvořte výrobní dokumentaci.
5. Vytvořte potřebné programové vybavení a odlaďte je.

DOPORUČENÁ LITERATURA:

[1] VÁŇA, Vladimír. ARM pro začátečníky. Praha : BEN - technická literatura, 2009. 195 s. ISBN 978-807-3002-466.
[2] MATOUŠEK, David. Práce s inteligentními displeji LCD : [znakové a grafické displeje, přípravky a programy]. 1 vyd. Praha : BEN - technická literatura, 2006. 222 s. ISBN 80-730-0121-7.

Termín zadání:	6.2.2012	***Termín odevzdání:***	21.5.2012

Vedoucí práce: Ing. Tomáš Macho, Ph.D.
Konzultanti diplomové práce:

doc. Ing. Václav Jirsík, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Předložená práce se zabývá návrhem a vývojem bezdrátového dálkového ovládání počítače PC, které je schopné ovládat počítač s operačním systémem Windows a Linux prostřednictvím Wifi. V rámci práce byl vytvořen řídící software pro oba operační systémy a zároveň bylo implementováno šifrování pro zabezpečení komunikace.

KLÍČOVÁ SLOVA

mikrokontrolér, dotykový displej, Wifi, Telnet, SSL, OpenCV, vzdálená plocha

ABSTRACT

This work deals with design of PC remote control, which is able to control computer with operating system Windows and Linux using Wifi connection. There was also created necessary support software for both operating systems. Communication was secured by ciphering algorithm.

KEYWORDS

microcontroler, touch screen, Wifi, Telnet, SSL, OpenCV, remote desktop

MACHÁČEK, Jiří *Bezdrátové ovládání počítače PC*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2012. 86 s. Vedoucí práce byl Ing. Tomáš Macho, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Bezdrátové ovládání počítače PC“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych na tomto místě poděkoval vedoucímu diplomové práce panu Ing. Tomáši Machovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

1 ÚVOD

Předložená diplomová práce se zabývá návrhem a vývojem zařízení, jehož úkolem je umožnit užvateli ovládat počítač PC bezdrátově i pomocí kabelu.

Motivací k tvorbě tohoto výrobku je snaha umožnit uživateli pracovat na výkonném PC umístěném mimo jeho aktuální pracoviště a dovolit mu provádět na něm nejběžnější činnosti jako procházení dokumentů, nastavování jeho chování, případně vyhledávání v databázích nebo na internetu. Pomocí takto navrženého zařízení by mohla být usnadněna práce například ve skladech při inventurách, během veletrhů pro ovládání reklamních tabulí apod.

Zařízení bude podporovat práci s operačními systémy Windows a Linux, kdy ve Windows bude možné plnohodnotně sledovat dění na obrazovce počítače a přímo na ně reagovat pohyby kurzoru získanými pomocí emulovaného touchpadu a tlačítek, případně odesílat text pomocí integrované klávesnice zařízení. V operačním systému Linux nebude implementováno grafické ovládání, protože se systémy na bázi Unix lze mnohem efektivněji pracovat pomocí příkazové řádky. Z tohoto důvodu bude software zařízení obsahovat emulátor terminálu, pomocí kterého bude možné odesílat příkazy vzdálené stanici a následně na ně reagovat pomocí vestavěné klávesnice. Pro zlepšení uživatelského komfortu při práci v Linuxu bude implementován terminálový profil (např. vt100, xterm) pro jednoduché grafické formátování výstupu.

V průběhu tvorby zařízení budou nejprve zvoleny klíčové součástky jako procesor, displej, paměť apod., poté bude navrženo blokové schéma, z něhož vychází obvodové schéma výrobku. Na základě obvodového schématu bude realizována deska plošného spoje, která bude osazena a oživena.

Pro navržený hardware bude navržen firmware, který umožní ovládání samotného výrobku a následně i komunikaci se stranou počítače. Z předchozího textu vyplývá, že bude nutné vytvořit dva typy software pro PC s operačním systémem Windows a Linux.

Během návrhu přístroje bude diskutována možnost zabezpečení komunikace, protože při použití bezdrátového rozhraní může snadno dojít k odposlouchávání třetí osobou, a tedy i k úniku citlivých informací jako jsou hesla apod.

2 ZÁKLADNÍ KONCEPCE A BLOKOVÉ SCHÉMA

Aby bylo možné navrhnout obvodové schéma, je nejprve nezbytné stanovit požadavky na zařízení. Mezi tyto požadavky patří nejen způsob komunikace zařízení s uživatelem, ale také volba komunikačních rozhraní atd. Tyto a další problémy budou diskutovány v následujících podkapitolách.

2.1 Koncepce výrobku

Při návrhu základní koncepce zařízení je vycházeno z předpokladu, že se bude jednat o zařízení s dotykovým displejem, na němž bude zobrazena pracovní plocha vzdáleného počítače (případně jeho textový výstup), prostřednictvím tohoto displeje bude zařízení také ovládáno. Komunikace s počítačem bude probíhat pomocí standardního rozhraní dostupného v běžném PC. Výrobek bude obsahovat akumulátor, aby bylo možné používat jej kdekoli bez nutnosti použít napájecí kabel.

Při provedeném průzkumu trhu na začátku tvorby diplomové práce bylo zjištěno, že se na trhu jednoúčelová zařízení tohoto typu prakticky nevyskytují. Patrně je to způsobeno tím, že na trh nastupují zařízení Tablet PC, do nichž je možné doinstalovat aplikaci, jež nahradí danou funkci PC, případně umožní komunikaci s tímto PC na požadované úrovni.

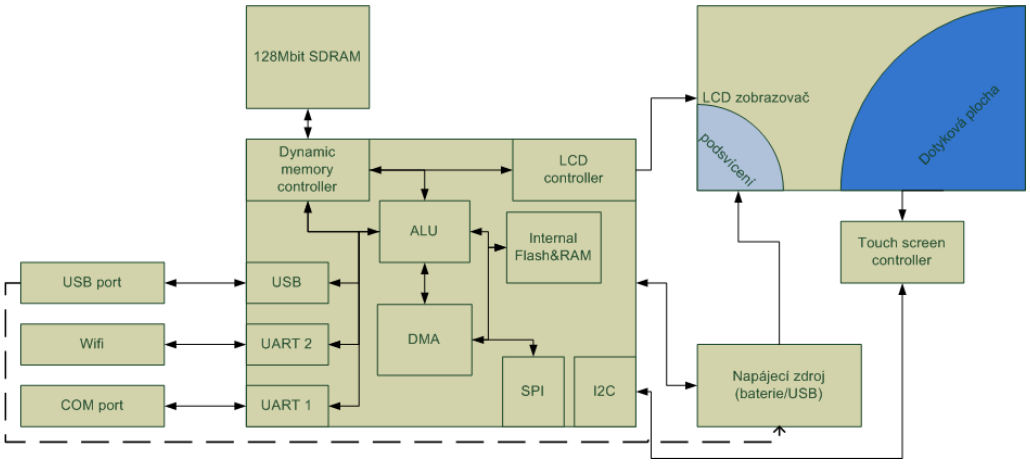
Zařízení by mohlo být realizováno několika způsoby, přičemž nejjednodušší je samozřejmě použití již zmíněného hotového výrobku (tabletu) s operačním systémem Android apod., pro nějž již existuje aplikace typu VNC Viewer, což je program umožňující ovládání PC z jiného stroje. Nevýhodou takového řešení je však jeho vysoká cena. Tablet je totiž poměrně drahé zařízení, které je často (zejména v případě levných tabletů) přes vysoký teoretický výpočetní výkon zahlceno operačním systémem, který obsluhuje. V rámci této práce bude vytvořeno jednodušší jednoúčelové zařízení, a tedy i levnější na výrobu, pokud bude vyráběno ve velkých sériích.

2.2 Blokové schéma

Navržené blokové schéma je uvedeno na obrázku 2.1.

2.2.1 Zobrazovací a procesorová jednotka

Jádrem zařízení bude mikrokontrolér, který bude řídit veškeré periferie, včetně displeje apod. Požadavky na tento mikrokontrolér vyplynou z dalšího textu, důležité je, aby poskytoval dostatečný výpočetní výkon pro řízení periferií, také by měl být schopen přepnutí do úsporného režimu v případě, že zařízení není používáno.



Obr. 2.1: Blokové schéma zařízení

Jednou z hlavních periferií výrobku bude displej. Standardně jsou u podobných zařízení používány displeje s úhlopříčkou 4"-10", rozlišení těchto displejů se běžně pohybuje v rozmezí od 320x240 pixelů do 1280x800. Navrhovaný přístroj bude spíše menších rozměrů, jako optimální rozlišení displeje je uvažován rozměr 480x272 pixelů, praktickým pokusem bylo ověřeno, že postačuje barevná hloubka 12 bitů, aby obraz z počítače nebyl barevně znatelně zkreslený.

Pro ovládání zařízení, je požadována dotyková plocha na povrchu displeje pro získání příkazů od uživatele. Pro pohodlné ovládání postačí rezistivní verze této plochy.

Standadní displej tohoto typu v sobě obsahuje řadič typu HX8257, jehož katalogový list je k nalezení v [11], nebo podobný. Tento řadič na rozdíl od jiných (používaných v menších displejích) neobsahuje žádnou interní paměť pro obrazová data, proto je nezbytné jej stabilně překreslovat pomocí řídícího procesoru. Aby bylo překreslování stabilní a displej neblikal, musí procesor poskytovat patřičné řídící signály, přičemž synchronizační signál musí mít frekvenci alespoň 9 MHz.

Tato frekvence již není zanedbatelná a při softwarovém generování řídících signálů dochází již tímto algoritmem k zahlcení procesoru. Aby nedocházelo k výkonnostním problémům, je třeba zvolit mikrokontrolér s vhodnou taktovací frekvencí, případně s periferií, která obstará řízení displeje bez účasti řídícího softwaru. Také je možné použít externí čip, jenž obstará veškeré řízení.

Po zvážení všech možných alternativ řízení displeje (CPLD, externí řídící čip apod.) byl zvolen mikrokontrolér s periferií pro řízení TFT displeje. Toto řešení je nejjednodušší z konstrukčního hlediska, lze také předpokládat, že bude nejlevnější.

V rámci této kapitoly je nutné vyřešit i paměťové nároky softwaru výrobku. Z volby displeje vyplývá, že bude nezbytné v paměti mikrokontroléru udržovat data

obrazu displeje, procesor tedy musí obsahovat dostatečně velký blok paměti pro tento účel. Z parametrů displeje vyplývá, že paměťový prostor pro uložení obrazu musí mít následující velikost:

$$n = 2 * h * l = 2 * 480 * 272 = 261120B = 255kB \tag{2.1}$$

Kde n je objem dat, h a l jsou geometrické rozměry displeje. Velikost musí být násobena dvěma, protože při zvolené barevné hloubce každý pixel zabere dva bajty.

Pro uložení dat displeje tedy nepostačuje interní paměť RAM procesoru (procesor obsahuje 64kB paměti pro programová data a několik dalších bufferů určených pro specifické periferie, celkově 98kB), musí zde tedy být připojena externí paměť, jejíž velikost bude diskutována v dalším textu. Je ale jasnou podmínkou, že paměť musí mít velikost pro uložení minimálně jednoho obrazového bufferu.

Aby bylo možné uložit do zařízení například předlohy obrazových prvků, je vhodné připojit také externí paměť Flash. Tato paměť standardně může komunikovat pomocí paralelního rozhraní nebo pomocí SPI, pro navrhovaný výrobek je vhodná alternativa sériová (je potřeba méně vodičů pro komunikaci s MCU), přestože komunikace s ní je pomalejší. Data z ní však lze během startu procesoru uložit do paměti RAM a dále s nimi pracovat zde.

Aby bylo možné snímat informace z dotykové plochy, musí být použit externí obvod pro snímání, s nímž bude procesor komunikovat po jedné z komunikačních periferií, pravděpodobně přes rozhraní I^2C. Výhodou externího obvodu je, že v procesoru není třeba řešit řídící signály pro dotykovou plochu, lze tak strojový čas procesoru využít jiným způsobem.

2.2.2 Komunikační rozhraní

V zadání práce je specifikováno, že zařízení má být schopno bezdrátové komunikace. Jako vhodná rozhraní byly uvažovány standardní periferie kancelářského PC nebo notebooku, jimiž jsou Bluetooth a Wifi. Aby bylo možné učinit rozhodnutí, musí být stanoveny požadavky na toto rozhraní.

Nároky na rychlost komunikace vycházejí z předpokládaných objemů dat, jež budou putovat oběma směry. Odchozí data není potřeba v tomto okamžiku řešit, protože jejich objem je zanedbatelný v porovnání s příchozími obrazovými daty. V první fázi vývoje výrobku uvažujeme, že obrazová data nebudou žádným způsobem komprimována, pouze budou při malých změnách obrazu odesílány pouze takto vzniklé rozdíly. Důvodem je snaha o zjednodušení softwaru, který se tak bude lépe odlaďovat a snáze se budou hledat chyby a problémy v jednotlivých jeho částech.

Pokud budeme předpokládat plný snímek při barevné hloubce 12 bitů, jeden snímek tak zabírá dle rovnice 2.1 255kB. Takto plný snímek však bude odesílán

spíše výjimečně (nepředpokládá se, že by byl výrobek užit ke sledování filmů či hraní 3D her), proto lze uvažovat spíše velké množství malých obrázků, jejichž objem dat bude podstatně menší.

Předpokládejme tedy plný snímek každých 5s a průměrně 5 snímků za sekundu, jež pokrývají 10% plochy, dospějeme výpočtem k závěru, že rozhraní musí být schopno přenést:

$$k = \frac{n}{5} + 5 * 0.1 * n = 178,5kB/s \tag{2.2}$$

Nyní je zřejmé, že komunikační rozhraní musí být schopné přenášet data alespoň rychlostí

$$s = k * 8 = 1428kBit/s = 1,4MBit/s \tag{2.3}$$

S touto rychlostí může pracovat Bluetooth i Wifi, ovšem nevýhodou Bluetooth je malý dosah v porovnání s Wifi. Pokud výrobek bude obsahovat Wifi modul, bude moci komunikovat s hostitelským počítačem z prakticky neomezené vzdálenosti, jedinou podmínkou bude existence Access Pointu (AP) v okolí výrobku a dostupnost počítače ze sítě tohoto AP.

Nevýhodou vybraného řešení je nutnost použít šifrování dat, jinak by hrozilo odposlouchávání komunikace třetí osobou.

Nad rámec zadání diplomové práce bude výrobek obsahovat také USB rozhraní a port RS-232, které bude možné použít ke stejnému účelu jako Wifi. Kvůli omezené rychlosti RS-232 však přes toto rozhraní nebude možné přijímat obrazová data, zařízení tak bude možné použít pouze jako touchpad nebo klávesnici.

Z předchozích informací vyplývá, že procesor musí obsahovat odpovídající komunikační rozhraní (UART a USB), pro komunikaci s Wifi modulem je nejvhodnější rozhraní UART.

3 OBVODOVÝ NÁVRH ZAŘÍZENÍ

Na základě navrženého blokového schématu bude nyní řešeno obvodové schéma výrobku a na jeho základě bude navržena deska plošného spoje zařízení.

3.1 Obvodové schéma zařízení

Obvodové schéma bude řešeno v logických blocích na základě blokového schématu z předchozí kapitoly.

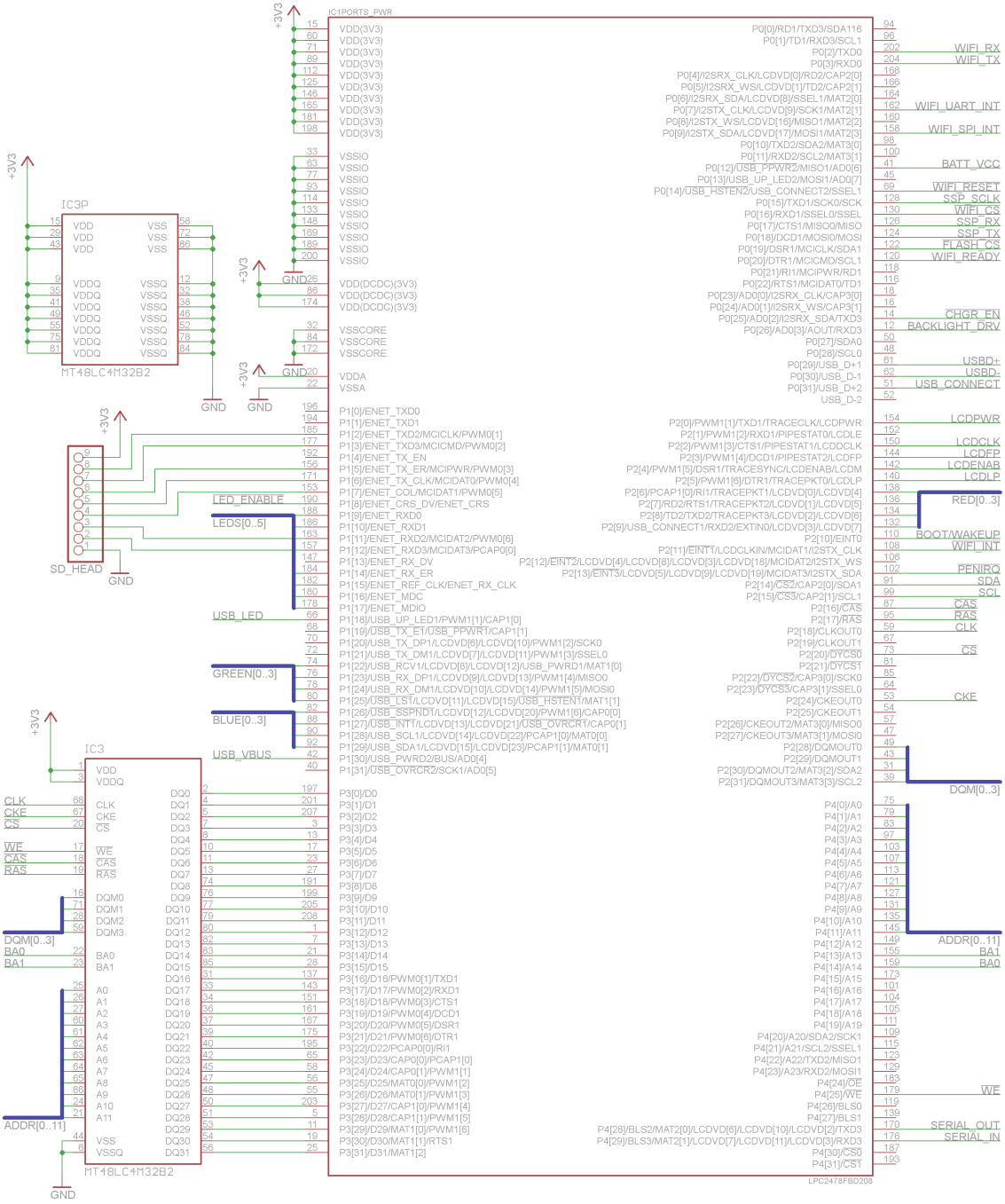
3.1.1 Mikroprocesorová jednotka

Na základě předchozí kapitoly je nyní nutné zvolit vhodný konkrétní typ mikrokontroléru. Poté, co byly vyloučeny mikrokontroléry v pouzdrech řady BGA, protože jejich osazování a oživování je v amatérských podmínkách extrémně složité, byl zvolen mikrokontrolér LPC2478 společnosti NXP, jehož katalogový list je dostupný z [1]. Alternativou k tomuto procesoru byl typ LH75411, ten však neobsahuje flash paměť, opět by se tak komplikovala tvorba desky, protože by bylo nezbytné použít externí paměťový čip.

Výhodou zvoleného mikrokontroléru je kromě řadiče displeje také externí paměťová sběrnice s řadičem paměti SDRAM, USB kontrolér, 512kB paměti flash, tři UART jednotky, hodiny reálného času apod. Mikrokontrolér umožňuje teoretickou maximální rychlost 100 MHz, nicméně předpokládá se, že bude taktován na doporučených 72MHz. Lze očekávat, že tento výkon bude pro zařízení dostačující, pokud budou vhodně zvoleny pracovní algoritmy.

Při popisu jeho zapojení budeme vycházet ze dvou základních dokumentů. Prvním dokumentem je zmíněný standardní katalogový list kontroléru, který lze nalézt na [1]. Druhý je tzv. uživatelský manuál čipu, jenž lze nalézt na [2]. V těchto dokumentech jsou uvedeny všechny nezbytné pasivní prvky, bez kterých MCU nemůže pracovat. Na obrázku 3.1 je uvedeno zapojení periferních a napájecích pinů procesoru, na obrázku 3.2 jsou dále nezbytné podpůrné obvody procesoru a jejich zapojení.

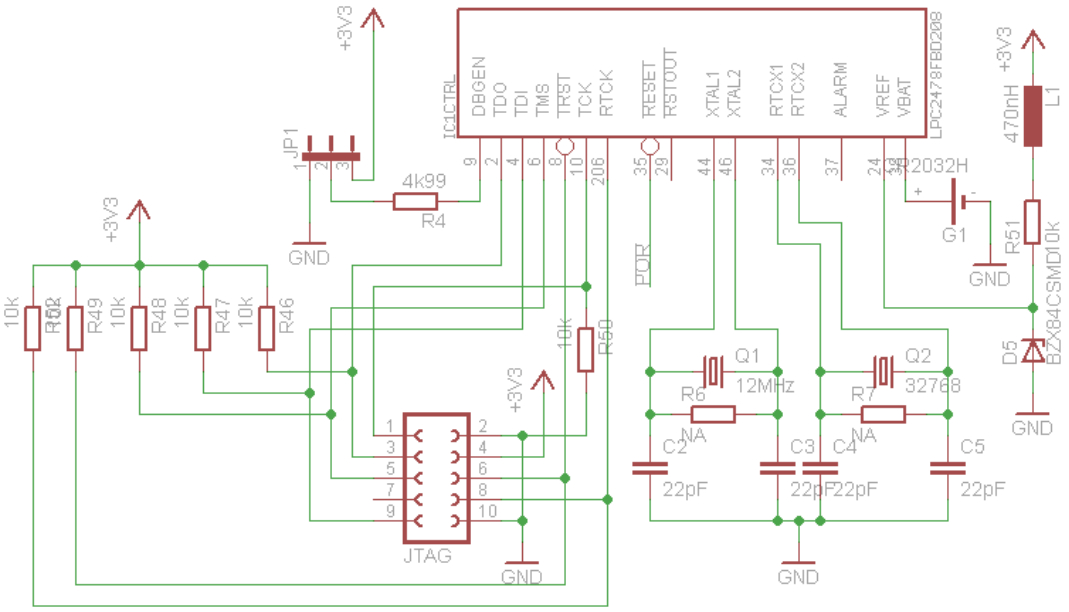
Tento mikrokontrolér obsahuje 14 párů napájecích pinů, které je nezbytné připojit k napájecímu napětí 3,3V. Toto napětí je dodáváno stabilizovaným zdrojem, jehož konstrukce bude řešena dále. Každý pár napájecích pinů je doplněn blokovacím kondenzátorem 100nF, které zde pro přehlednost nejsou zobrazeny.



Obr. 3.1: Mikroprocesorová jednotka zařízení

Podpůrné obvody mikrokontroléru

Procesor obsahuje dva oscilátory, jež pro svou funkci potřebují externí krystal. Volba hodnoty krystalu hlavního oscilátoru vychází z doporučení výrobce. Podle manuálu [2] by tato hodnota měla být v intervalu $f \in <1; 24> MHz$. Dále musíme vycházet z taktování jádra procesoru. Procesor bude taktován na 72MHz, předpokládáme také



Obr. 3.2: Mikroprocesorová jednotka zařízení - podpůrné obvody

použití USB rozhraní, které vyžaduje co nejpřesnější frekvenci 48MHz. Proto bude použit krystal 12MHz.

Druhý oscilátor je určen pro řízení hodin reálného času. Krystal tohoto oscilátoru má zvolenu frekvenci $f_{rtc} = 32768Hz$, což je hodnota, s jejíž pomocí lze nastavit předděličky periferie tak, aby hodiny generovaly signál o frekvenci 1Hz.

Oba krystaly jsou doplněny kondenzátory $C_2 - C_5$, vzniklo tak standardní zapojení oscilátoru používaného u MCU. Hodnoty jsou zvoleny podle doporučení katalogového listu MCU.

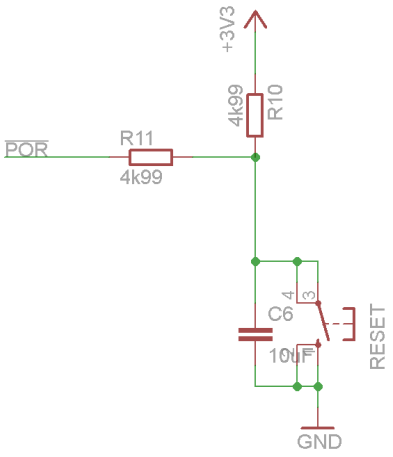
Jako doplnění hodin reálného času je připojena k MCU knoflíková baterie s napětím 3,3V, jež slouží jako napájecí zdroj pro druhý oscilátor v případě výpadku napájení. Toto je velice výhodná vlastnost MCU, protože uživatel může zařízení vypnout vypínačem a jeho základní nastavení, čas apod. zůstane zachováno (MCU obsahuje 2kB paměti SRAM pro data, která mají být uchována po výpadku napájení).

Reset obvodu je řešen dvěma způsoby. První z nich je reset pomocí signálu *nTRST*, jenž je řízen pomocí JTAG adaptéru. Tento reset uvádí do výchozího stavu

všechny obvody vyjma obvodů JTAG periferie. Druhým signálem je reset $nPOR$[1], který resetuje celý procesor. Externí obvod k řízení tohoto pinu se skládá z tlačítka RESET, rezistorů $R_{10} = R_{11} = 4,99k\Omega$ a kondenzátoru $C_6 = 10uF$. Časová konstanta obvodu (délka resetu obvodu po uvolnění tlačítka) je vypočtena dle vztahu:

$$\tau_{Reset} = R_{10}.C_6 = 4990.10^{-5} = 49,9ms \tag{3.1}$$

Vzhledem k tomu, že pro splnění podmínek pro vyvolání události resetu je potřeba jen několik hodinových cyklů jádra, je tato doba více než dostačující. Rezistor R_{11} je v obvodu umístěn, aby nebyl zbytečně zatěžován pin procesoru při nabíjení kondenzátoru. Schéma tohoto obvodu je na obrázku 3.3.



Obr. 3.3: Resetovací obvod

Dále je k MCU připojena reference napájecího napětí, pro vyhlazení tohoto napětí je v obvodu připojena sériově cívka $L_1 = 470nH$, hodnota napětí je stabilizována na hodnotu 3,3V pomocí Zenerovy diody D_5 a rezistoru R_{51}.

Jako další z funkčních pinů, které je třeba ošetřit je pin DBGEN, který slouží pro výběr režimu JTAG konektoru. Při log. 1 je procesor v režimu EmbeddedICE a je povolena možnost debuggování. V log. 0 je procesor v režimu Boundary scanu.

S možností debuggování přímo souvisí konektor JTAG, jehož zapojení odpovídá (z praktických důvodů) zapojení konektoru debuggeru AVR Dragon. Tento konektor je totiž desetipinový, zatímco standardní JTAG konektor například prostředku J-link má pinů 20. Jelikož nepředpokládám použití tohoto debuggeru pro ladění, není tato změna na škodu.

[1]Power-on reset

Externí paměť

Je poměrně běžné, že většina mikrokontrolérů obsahuje na čipu určité malé množství paměti RAM, dále také paměť Flash, zřídka také EEPROM paměť nebo tzv. Once programable memory. Námi zvolené MCU obsahuje pouze paměti Flash a SRAM. Paměť flash má kapacitu 512kB, paměť SRAM 98kB. Množství paměti Flash je pro naši aplikaci dostačující, množství paměti SRAM však nikoli. Abychom mohli provozovat LCD displej, bude totiž potřeba v RAM paměti MCU udržovat data obrazu tohoto displeje. Problém lze snadno vyřešit připojením externí paralelní paměti k paměťové sběrnici. Tato sběrnice podporuje připojení pamětí SRAM o šířce 8, 16 a 32 bitů, dále SDRAM o šířce 16 a 32 bitů.

Nejprve bylo nezbytné stanovit nároky na tuto paměť. Měla by být dostatečně rychlá. Vzhledem k tomu, že MCU lze taktovat až na 100MHz (přestože je plánováno provozovat jej na 72MHz), měla by být přístupová doba do této paměti dostačující i v případě maximální frekvence. Přístupovou dobu do paměti lze vypočíst jako:

$$T_p = \frac{1}{f} = \frac{1}{1.10^8} = 10ns \tag{3.2}$$

Dále by měla umožnit uložení dostatečného množství dat. Toto množství by mělo odpovídat alespoň trojnásobku velikosti datového bufferu potřebného pro uložení jednoho snímku (dva snímky v paměti a odkládací prostor pro ostatní data). Minimální velikost tak odpovídá:

$$V_{min} = 3.h.l.k = 3.480.272.2 = 765kB \tag{3.3}$$

kde h a l jsou rozměry displeje v bodech a k je počet bajtů pro uložení jednoho obrazového bodu[2].

Také je vhodné, aby paměť disponovala alespoň 16-ti bitovou datovou sběrnicí pro vyšší rychlost přenosu dat (32 bitové statické paměti se běžně nevyrábějí, bylo by tedy nutné použít 2 čipy). Na jiném vývojovém kitu s procesorem AT91SAM7SE256 bylo odzkoušeno, že i s takovouto pamětí je běh celého programu významě zpomalen přibližně na polovinu oproti běhu s daty v interní paměti. Bylo tedy rozhodnuto s výhodou použít místo statického řadiče procesoru dynamický řadič, jenž by měl umožňovat čtení po shlucích (burstech).

Výhodou dynamické paměti je také její lepší poměr cena/výkon[3]. Byla tedy zvolena paměť *MT48LC4M32B2*, jejíž výhodou je navíc 32-bit široká datová sběrnice. Bližší informace o paměti lze nalézt v jejím katalogovém listu [3], odtud také vychází její zapojení. Pro svůj provoz vyžaduje stabilizované napájení 3,3V, jež musí

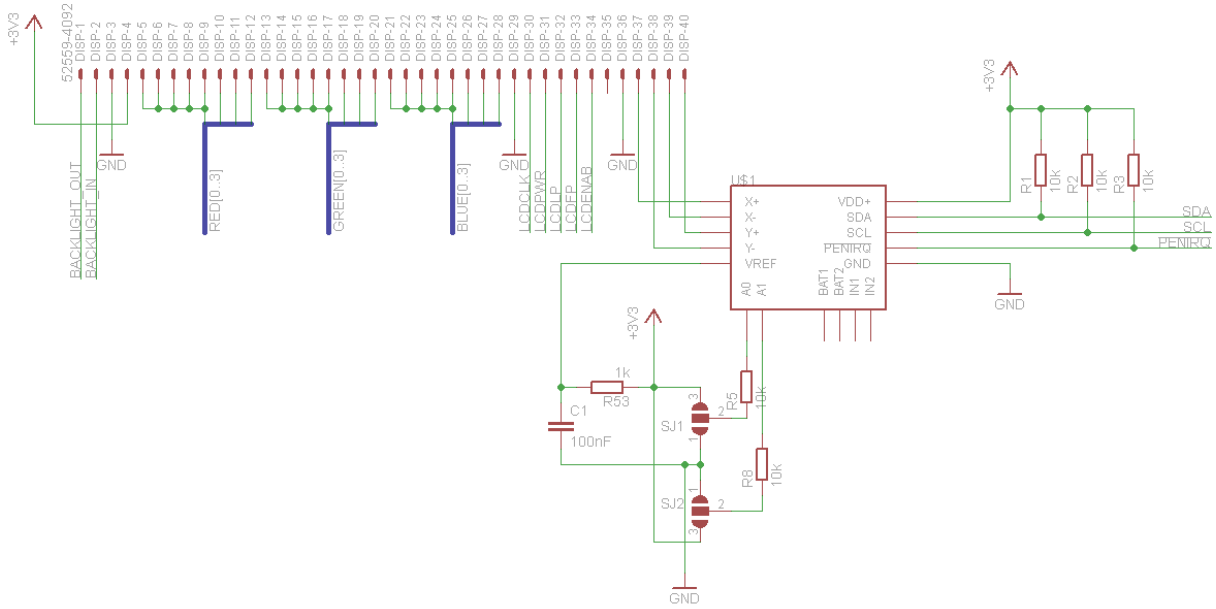
[2]Zvolili jsme 12-bit barevnou hloubku

[3]cena SRAM 2MB/10ns je 980,- s DPH, cena SDRAM 128 MB/133MHz je 390,-

být přivedeno na všechny páry napájecích pinů. Každý pár je opět doplněn blokovacím kondenzátorem s kapacitou 100nF, kondenzátory opět nejsou pro přehlednost zakresleny.

3.1.2 Zobrazovací modul

Jako nejvhodnější displej splňující požadované parametry byl zvolen typ PT0434827t-a401 společnosti Palm Technology, jenž je řízen již zmíněným řadičem HX8257 [11]. Disponuje rozlišením 480x272 pixelů s barevnou hloubkou 24 bitů, použita však bude pouze zmíněná 12-ti bitová hloubka.

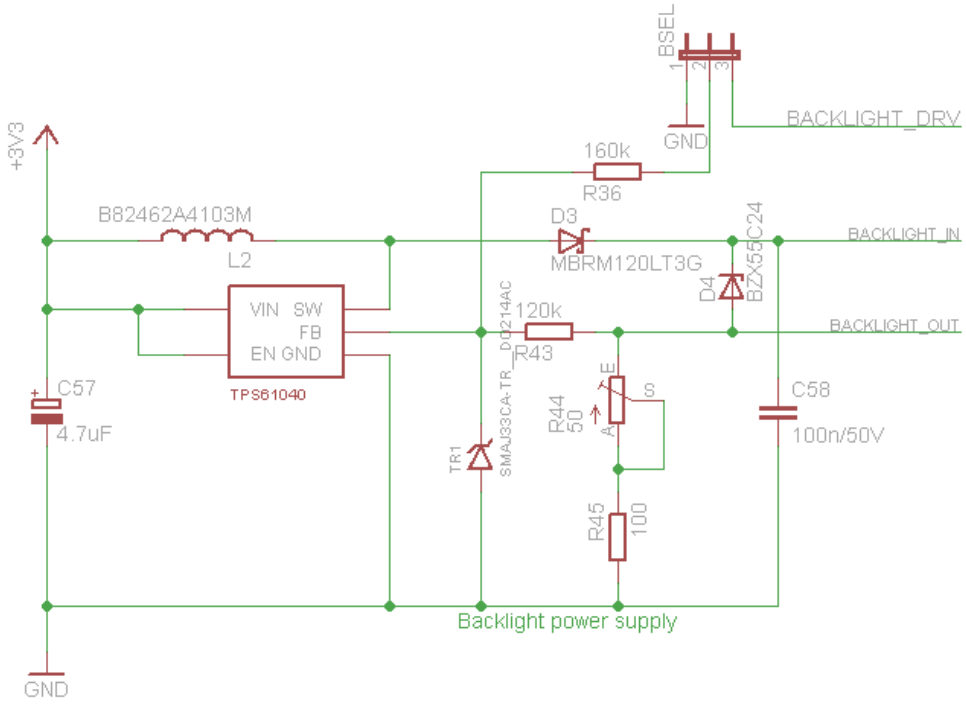


Obr. 3.4: Konektor pro připojení LCD displeje

Připojení displeje je uvedeno na obrázku 3.4. Pro fyzické připojení displeje je užit ZIFF konektor se 40 piny. Je zvolen přímý typ, který umožňuje připojení linek z obou stran. Toto řešení je výhodnější z hlediska realizace DPS. Zapojení vývodů konektoru je řešeno na základě informací z katalogového listu LCD displeje [4] a podle informací z manuálu procesoru [2].

Jsou zde zřejmé tři datové sběrnice, pro každou z barev jedna. Dále jsou zde připojeny řídící vodiče pro řízení řadiče, vertikální a horizontální synchronizace, hodinový signál apod. Logická část displeje je napájena z hlavní napájecí větve 3,3V.

Pro napájení podsvětlení bylo nutné použít externí obvod s DC/DC měničem, jehož schéma zapojení je na obrázku 3.5. Požadavkem na tento obvod byla schopnost



Obr. 3.5: Obvod pro napájení podsvětlení LCD

dodávat trvale proud alespoň 20mA při napětí 19V s napájením ze společného zdroje 3,3V. Dále byla důležitá možnost analogového řízení jasu. Hlavním prvkem tohoto obvodu je součástka tps61040, jejíž katalogový list [5] je ke stažení na stránkách výrobce. Odtud je také převzato zapojení měniče pro režim práce při konstantním proudu a řešení analogového řízení, včetně hodnot součástek. Analogový signál pro řízení intenzity podsvětlení je získán z výstupu D/A převodníku procesoru, tento signál je označen *BACKLIGHT_DRV*. Změnou nastavení propojky *BSEL* lze podsvětlení trvale zapnout.

Pro funkci měniče je nezbytná cívka L_2, jejíž konkrétní typ je doporučován výrobcem, stejně jako typ diody D_3[4]. Proud diodami podsvětlení je definován sériovou kombinací rezistorů R_{44} a R_{45}, přičemž princip regulace spočívá ve "snaze"obvodu udržet na referenčním pinu FB napětí $U_{FB} = 1,233V$. V regulačním obvodu je dále zařazen dělič tvořený rezistory R_{43} a R_{36}, na jehož výstupu je měřena zmíněná regulační odchylka. Změnou potenciálu na pinu rezistoru R_{36} se mění vlastnosti děliče a je tak umožněna změna požadované hodnoty proudu diodami.

Aby nedošlo k poškození měniče při odpojení LCD displeje, je v obvodu zařazena

[4]Výpočet vlastností součástek je poměrně komplexní záležitost rozebraná v katalogovém listu. Výpočet však nebylo potřeba provádět, protože katalogový list obsahuje ukázkové zapojení obvodu s hodnotami odpovídajícími našim potřebám.

Zenerova dioda D_4 s nominálním napětím $U_z = 24V$, která omezuje maximální výstupní napětí obvodu na definovanou mez. Pokud by dioda v obvodu nebyla, po odpojení displeje by došlo k zvyšování napětí až na cca 38V, rozkmitání, přehřátí měniče a jeho zničení[5].

Řízení dotykové plochy je řešeno pomocí externího čipu TSC2003 [7] společnosti Texas Instruments. Jedná se o kompletní řešení pro řízení rezistivních dotykových displejů, získaná data jsou do mikrokontroléru přenášena pomocí I^2C sběrnice, pro včasnou obsluhu mikrokontrolérem je připojen signál *nPENIRQ* k pinu externího přerušení procesoru. Propojky $SJ1$ a $SJ2$ slouží k nastavení adresy čipu na sběrnici I^2C. Proud tekoucí propojkami je omezen na minimální hodnotu rezistory $R_5 = R_8 = 10k$. Jejich hodnota není kritická, protože vstupy obvodu nejsou ovlivněny interními pull-updown rezistory.

K definování klidové úrovně signálů *SDA*, *SCL* (standardních signálů sběrnice I^2C) a *nPENIRQ* slouží rezistory $R_1 = R_2 = R_3 = 10k$. Jejich hodnota byla zvolena na základě zkušenosti.

3.1.3 Wifi modul

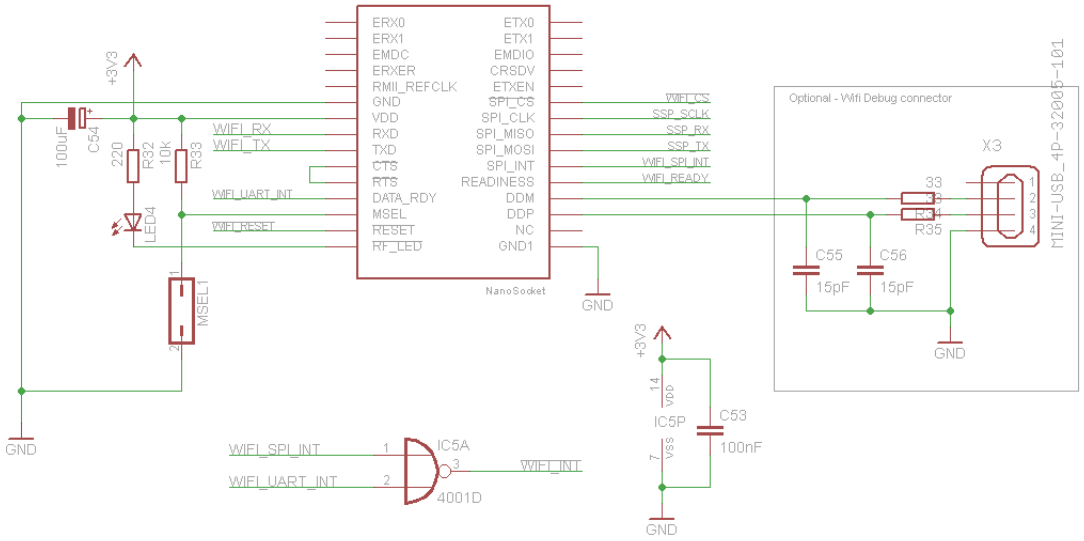
Pro bezdrátovou komunikaci mezi konzolí a počítačem PC byl zvolen modul společnosti Connect One s názvem NanoSocket iWifi. Byl zvolen pro své velmi dobré firmwarové vybavení, obsahuje totiž kompletní TCP/IP stack včetně podpory šifrování. Mezi jeho další výhody patří komunikační rychlost až 3MBit/s přes UART a až 12MBit/s přes rozhraní SPI. Původně byl sice vybrán modul společnosti Roving Networks RN-171, jehož předností byla i nízká cena[6], společnost Roving Networks ovšem poskytovala znatelně horší technickou podporu a dokumentace jejich výrobků byla plná nejasností a chyb.

Zapojení modulu je uvedeno na obrázku 3.6, celé zapojení je vytvořeno na základě informací z katalogového listu modulu NanoSocket [10].

Celý wifi modul je napájen z hlavní napájecí větve +3V3. S hlavním mikroprocesorem je propojen pomocí UART sběrnice i pomocí SPI rozhraní. Tato koncepce byla zvolena, protože v době návrhu desky nebylo jasné, jak přesně se bude chovat firmware zařízení. Procesor totiž obsahuje DMA řadič, ten umožňuje přesuny dat pomocí UART rozhraní i SPI rozhraní, případně mezi dvěma lokacemi paměti. SPI rozhraní je sice cca. 4x rychlejší, nicméně jeho obsluha je o něco složitější. Očekávám, že po odzkoušení chování prototypu bude zachováno pouze jedno komunikační rozhraní.

[5] Výrobce uvádí, že měnič má tepelnou ochranu, ta však pracuje pouze pokud k přehřátí dojde za standardního provozu se zátěží.

[6] NanoSocket: 1700,-Kč, RN-171: 1300,-



Obr. 3.6: Schéma zapojení wifi modulu

Pro řízení komunikace jsou k procesoru přivedeny také příslušné signalizační linky *nWIFI_RESET* pro reset modulu, *WIFI_UART_INT* pro vyvolání přerušení procesoru po přijetí dat modulem, *WIFI_READY* pro zjištění, jestli modul dokončil interní boot sekvenci, *WIFI_SPI_INT* pro vyvolání přerušení od SPI rozhraní a *nWIFI_CS* pro výběr modulu při komunikaci přes SPI. INT signály jsou spojeny pomocí hradla NOR označeného jako IC_5 do jediného signálu *nWIFI_INT*, jenž je připojen k pinu externího přerušení procesoru. Toto řešení bylo zvoleno kvůli nedostatku pinů externích přerušení, kdy bylo potřeba obsloužit dva signály přerušení pomocí jednoho pinu procesoru.

Pro signalizaci uživateli je k modulu připojena LED dioda, její rezistor je zvolen na základě následující rovnice:

$$R_{32} = \frac{U - U_{LED}}{I_{LED}} = \frac{3,3 - 1,5}{0,01} = 180\Omega \tag{3.4}$$

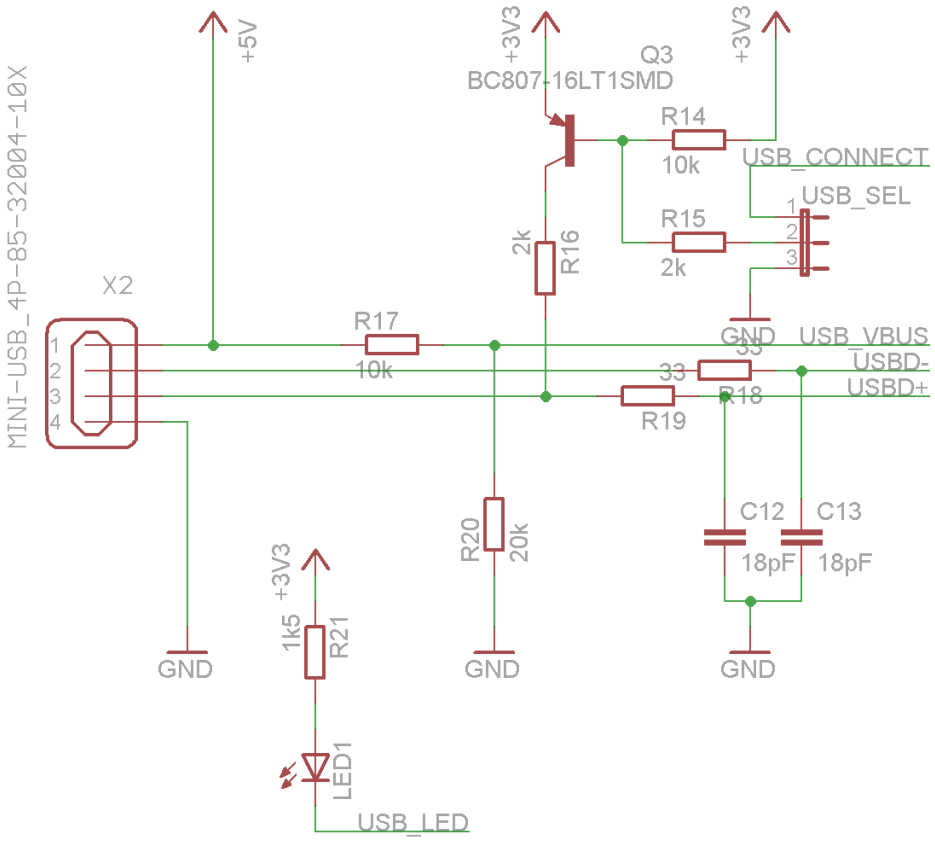
K modulu wifi je také připojena propojka $MSEL_1$, jež umožňuje přivést log. 0 na pin MSEL a uvést wifi modul do režimu bootloaderu. Standardně je pin připojen k log. 1 pomocí pull-up rezistoru R_{33}.

3.1.4 Komunikace prostřednictvím kabelu

Tento způsob komunikace je řešen nad rámec práce, jelikož podle zadání má výrobek být schopen komunikovat pouze bezdrátově. Je implementováno USB rozhraní a standardní sériový port RS232.

USB

Vzhledem k tomu, že procesor obsahuje jako jednu z periferií USB kontrolér, postačí vyřešit zapojení konektoru. Přes tento konektor by také mělo probíhat nabíjení interního akumulátoru.



Obr. 3.7: Schéma zapojení konektoru USB

Schéma zapojení je uvedeno na obrázku 3.7. Toto zapojení vychází z doporučení z katalogového listu mikrokontroléru, je však upraveno na základě schématu vývojového kitu LPC2478-STK, které je ke stažení na stránkách společnosti OLIMEX [15].

Připojení vychází z následujícího principu detekce spojení na obou stranách (*Host* a *Device*):

1. Linky USB portu PC jsou v neaktivním stavu, aktivní je pouze napájení +5V na konektoru.
2. Je připojeno zařízení typu Device.
3. Zařízení *Device* rozpozná napájecí napětí na svém konektoru.

4. Zařízení *Device* aktivuje pull-up rezistory na datových linkách.
5. Zařízení *Host* rozpozná, že bylo připojeno zařízení *Device*.

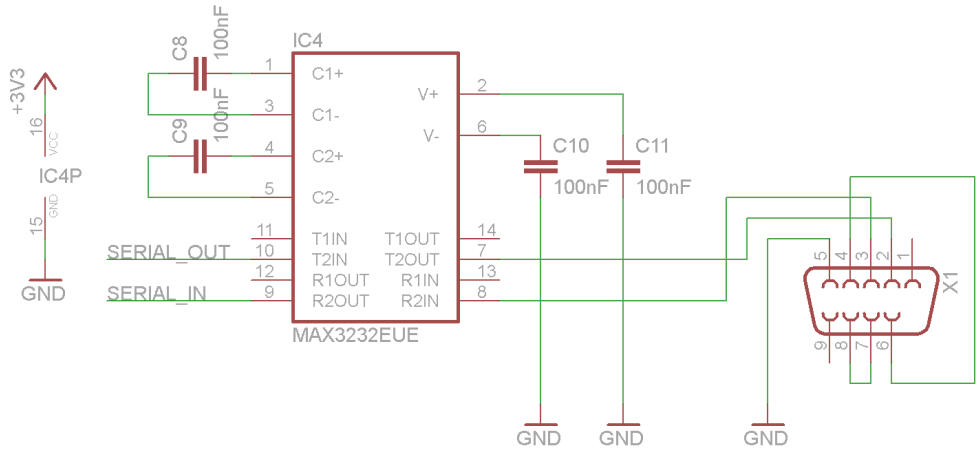
Pojmy *Host* a *Device* jsou obecně zavedené pojmy, kdy *Host* řídí komunikaci a *Device* je poskytovatel služeb jako myš, externí disk apod.

Datové linky *USBD* jsou připojeny k pinům USB portu USB1. Připojení k Hostitelskému zařízení je detekováno pomocí signálu *VBUS*, jehož napětí je definováno výstupem napěťového děliče tvořeného rezistory $R_{17} = 10k\Omega$ a $R_{20} = 20k\Omega$. Napětí z USB linky je tak zredukováno z 5V na 3,3V. Signál *USB_CONNECT*, který slouží k povolení rozhraní USB, je propojen s MCU přes tranzistor $Q3$ typu BC807. Tímto tranzistorem je připojen pull-up rezistor $R_{16} = 2k\Omega$, spínáním tranzistoru je tak možné povolit nebo zakázat detekci zařízení hostitelským zařízením. Signál *USB_CONNECT* je možné trvale nastavit do log. 0 přepínačem *USB_SEL* a povolit tak trvale rozhraní USB.

Rezistory $R_{19} = R_{18} = 33\Omega$ a kondenzátory $C_{12} = C_{13} = 18pF$ jsou zde pro zajištění impedance rozhraní. Jejich hodnota je opět stanovena v manuálu procesoru.

K indikaci úspěšného připojení k hostitelskému zařízení doplňuje schéma dioda LED_1. Proud diodou je definován rezistorem $R_{21} = 1k5$.

RS232



Obr. 3.8: Schéma zapojení konektoru RS232

Schéma zapojení konektoru RS232 je uvedeno na obrázku 3.8. Jelikož sériové rozhraní mikrokontroléru používá jiné logické úrovně než rozhraní RS232, je nezbytné použít konvertor těchto úrovní. Rozhodl jsem se využít známý převodník MAX3232, jehož výhodou je jednoduché zapojení a snadná dostupnost na trhu. Převodník je

zapojen standardním způsobem, zapojení je sestaveno na základě informací z katalogového listu [8]. Na straně procesoru je připojen k portu UART3, napájení obvodu je přivedeno z hlavní napájecí větve 3,3V. Základní zapojení doplňují 4 keramické kondenzátory C_8 - C_{11} s kapacitou 100nF, které jsou potřeba pro správnou funkci vnitřní nábojové pumpy obvodu. Samozřejmostí je opět blokovací kondenzátor, který pro přehlednost není ve schématu zobrazen. Na vnější straně zařízení je umístěn standardní konektor DE-9, který známe z běžného PC pod názvem COM port nebo také sériový port.

3.1.5 Indikační obvod

Aby měl uživatel lepší přehled o aktuálním stavu zařízení, obsahuje výrobek diody pro indikaci různých událostí. Těmito událostmi jsou:

- Aktivita UART - přijímání a odesílání dat
- Komunikace s Wifi modulem - čekání na odpověď
- Indikace normálního stavu

Pro indikaci stavů zařízení je ve schématu zakreslen obvod, jehož zapojení je uvedeno na obrázku 3.9. Jsou to v podstatě 3 dvoubarevné diody, jež jsou spínány přes budič sběrnice 74HC366D s negovanými výstupy. Ačkoli by měl být procesor schopen budit diody přímo, jedná se o "čistější řešení", jelikož diody lze signálem *nOE* odpojit a šetřit tak energii v případě, že je zařízení v úsporném režimu. Diodami také může téct větší proud, což je výhodné, pokud žádáme větší svítivost LED. Z praktických důvodů byly vybrány diody se společnou anodou, protože jsou spínány log. 0, která umožňuje spínání většího proudu než log. 1.

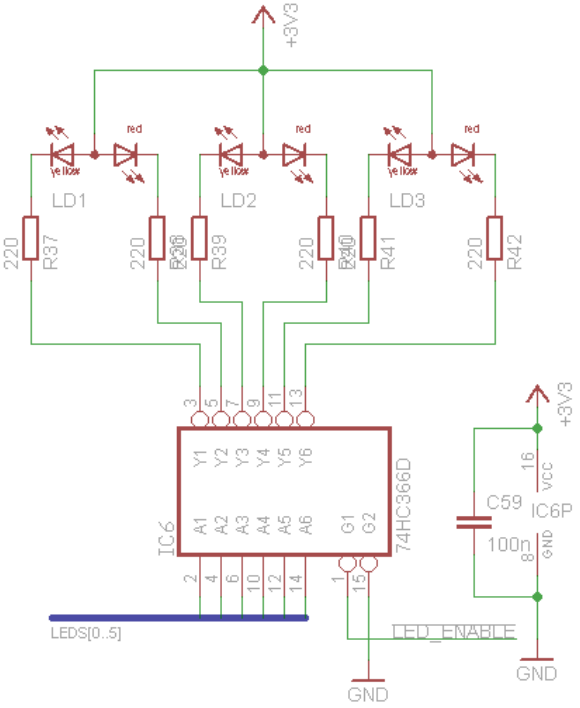
Proud diodami je stanoven rezistory $R_{37} - R_{42}$ na hodnotu přibližně 8mA, výpočet je již uveden v předchozím textu dle rovnice 3.4.

Pokud uvažujeme, že při indikaci aktivity UART rozhraní bude dioda při vysílání svítit červeně a při přijímání zeleně, indikace normálního chodu bude označena zeleným blikáním druhé diody a červený svit této diody bude indikovat čekání na odpověď wifi modulu, dospějeme k závěru, že jedna dioda (dva stavy) nemají přiřazenou žádnou událost. Je to z důvodu snadného přidání další události k indikaci během případného rozšiřování softwaru.

3.1.6 Další periferní obvody

Schéma periferních obvodů je uvedeno na obrázku 3.10. Těmito obvody jsou:

- Wake-up obvod
- Flash paměť
- Konektor pro ladění



Obr. 3.9: Indikační obvod

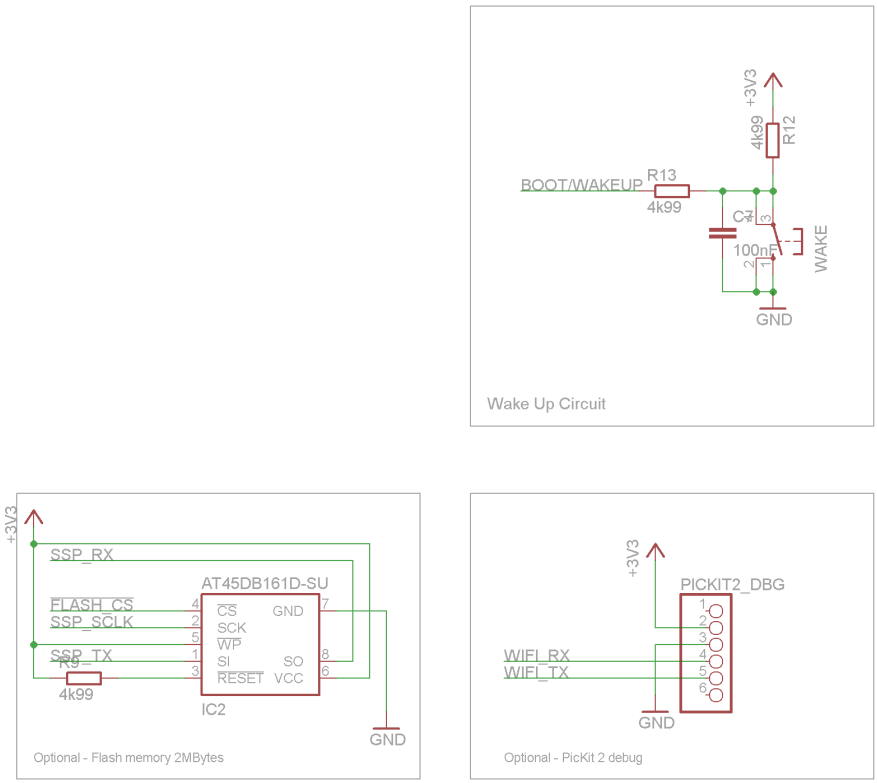
Wake-up obvod

U zařízení se předpokládá, že bude napájeno baterií Li-Ion. Protože baterie má omezenou kapacitu, bude procesor přepínán do úsporného režimu, který zajistí, že spotřeba výrobku klesne na minimum. Probuzení z takového stavu je však možné pouze specifickými signály, mezi něž mimo jiné patří změna log. úrovně na vstupu externího přerušení.

Princip funkce obvodu je prakticky stejný jako princip zapojení resetovacího tlačítka, pouze je zde užit jiný kondenzátor (100x menší pro menší časovou konstantu). Kondenzátor zde má funkci akumulátoru energie, který omezuje zákmity a šum vznikající na kontaktech tlačítka.

Flash paměť

Pro uložení pomocných dat (ne nezbytných pro start procesoru) slouží sériová paměť flash AT45DB161D s kapacitou 2Mb. Bližší informace o tomto obvodu lze nalézt v katalogovém listu [13]. Paměť je napájena ze standardní napájecí větve 3,3V. Tok dat mezi pamětí a procesorem je zajištěn pomocí sběrnice SPI, pro tento účel jsou přivedeny signály *SSP_TX*, *SSP_RX*, *SSP_SCLK*. Pro výběr zařízení na sběrnici je přiveden signál *nFLASH_CS*. Nepředpokládáme, že paměť bude nezbytné za běhu



Obr. 3.10: Schémata zapojení jednotlivých periferních obvodů

zařízení resetovat, proto je signál *nRESET* paměti připojen přes pull-up odpor k napájecímu napětí. Paměť bude stále přístupná pro čtení i zápis, proto je signál *nWP* připojen k napájecímu napětí.

Konektor pro ladění

Pro pohodlnější ladění softwaru zařízení je osazen konektor *PICKIT2_DBG*. Tento konektor je připojen na komunikační linku mezi MCU a Wifi modulem. Je tak možné simulovat Wifi modul z PC. Zapojení konektoru odpovídá zapojení zařízení PicKit 2, což je primárně ladicí nástroj pro mikrokontroléry společnosti Microchip. Jeho druhotnou funkcí je USB-UART most mezi PC a zařízením.

3.1.7 Napájecí zdroj

Pro napájení mikrokontroléru a všech dalších částí desky je třeba správně navrhnout napájecí zdroj. Jak již bylo definováno v předchozích kapitolách, napájecí zdroj by měl umožnit napájení z baterie typu Li-Ion a dobíjení této baterie z portu USB.

Dále by mělo být možné zakázat pomocí software zařízení nabíjení baterie a měla by být zajištěna teplotní ochrana baterie.

Nejprve bylo nezbytné stanovit napájecí nároky celého zařízení. Všechny prvky obvodu jsou schopny pracovat při napájecím napětí 3,3V, proto bude pro napájení zvolen DC/DC měnič s tímto výstupním napětím. Jelikož v tomto bodě jsou již známé veškeré prvky obvodu, lze s jistotou říci, jaký bude běžný proudový odběr a odběr ve špičkách. Vzhledem k tomu, že největšími "konzumenty"energie jsou MCU, paměť, LCD, Wifi modul a indikátory stavu, bylo třeba určit proudový odběr těchto periferií. Odběr byl určen experimentálně.

- MCU - 130 mA
- Paměť - 100 mA
- LCD - 30 mA (logika) + 100mA (podsvětlení)
- Wifi - 250 mA
- Indikátory - 60 mA

Pokud sečteme předcházející hodnoty, dojdeme k závěru, že odběr zařízení ve "špičkách"je přibližně 570 mA. Tato hodnota je však maximální možná, průměrný odběr bude o něco menší (nebudou svítit některé diody, wifi nevysílá apod.).

Napájecí obvod

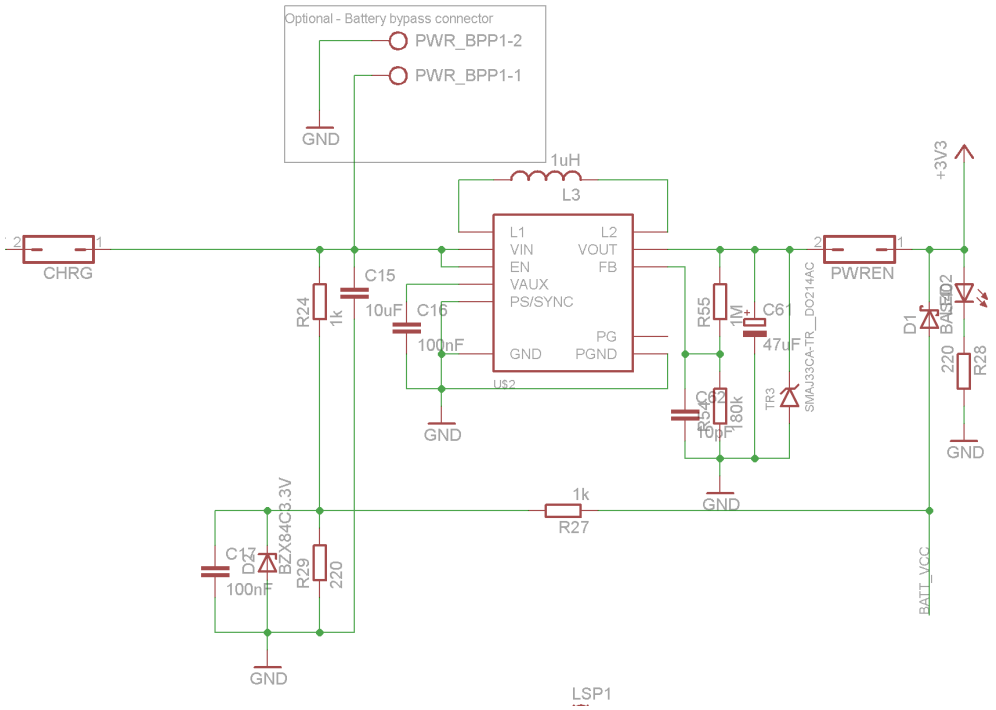
Schéma napájecího obvodu je uvedeno na obrázku 3.11. Hlavní částí napájecího obvodu je tzv. Buck-boost converter typu TPS63060 společnosti Texas Instruments. Veškeré informace k převodníku byly čerpány z jeho katalogového listu [6]. Jeho výhodou je, jak již z názvu vyplývá, schopnost pracovat v režimu zvyšujícím napětí i snižujícím napětí. Tato skutečnost je velice výhodná, pokud chceme napájet zařízení z baterie, jež má nominální napětí blízké napětí napájeného obvodu. Výhodou obvodu je také jeho vysoká účinnost (výrobce udává až 93%). Jsou tak do značné míry omezeny tepelné ztráty v napájecím zdroji.

Vstupní napětí obvodu je vyhlazováno kondenzátorem C_{15}, výstupní napětí je vyhlazováno kondenzátorem C_{61}. Jako činný prvek převodníku je použita cívka $L_3 = 1uH$, její hodnota je doporučena katalogovým listem výrobce. Vzhledem k tomu, že se jedná o regulovatelný obvod, je nezbytné připojit zpětnou vazbu pro regulaci výstupního napětí. Hodnotu rezistorů zpětné vazby zjistíme z rovnice:

$$R_{55} = R_{54}.\frac{V_{OUT}}{V_{FB}} - 1 \tag{3.5}$$

Na základě katalogového listu konvertoru byla zvolena hodnota rezistoru $R_{55} = 1M\Omega$. Dosazením do rovnice získáme hodnotu $R_{54} = 180k$. Na doporučení výrobce je pro lepší vlastnosti zpětné vazby doplněn filtrační kondenzátor $C_{62} = 10pF$.

Pro ochranu napájeného obvodu před napěťovými špičkami, které by mohly vzniknout při přechodových dějích na zdroji, je také přidán transil TR_3 s nominálním napětím 3,3V.



Obr. 3.11: Schéma zapojení napájecího obvodu

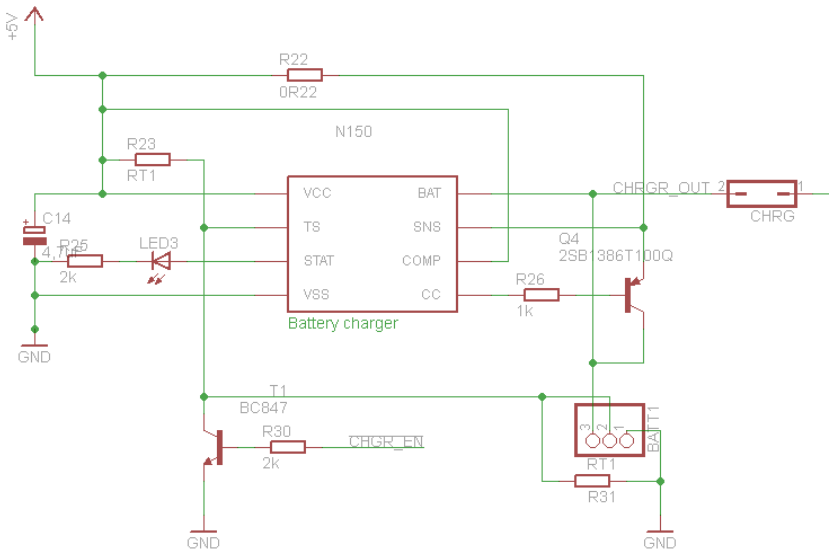
Od nabíjecího obvodu je napájecí zdroj oddělen propojkou *CHRGR*. Tato propojka je zde pro účely testování a umožňuje oddělit nabíjecí část od zbytku desky. Druhá propojka *PWREN* umožňuje vypnout napájení zbytku desky. Po umístění do plastového obalu by propojka měla sloužit jako konektor pro vypínač.

Aby mělo zařízení možnost měřit napětí na baterii, je ze zdrojové části vyveden signál *BATT_VCC*. Vzhledem k tomu, že napětí na baterii je větší než napětí na referenci A/D převodníku procesoru, bylo nezbytné užít napěťový dělič pro poměrné snížení napětí přivedeného na procesor. Napětí měřené procesorem je v poměru přibližně 1:6, ke zjištění přesného poměru bude potřeba softwarová kalibrace.

Maximální výstupní signál je omezen dvěma diodami. Zenerova dioda omezuje vrcholovou hodnotu napětí na 3,3V, dioda D_1 zajišťuje omezení napětí na aktuální hodnotu napájecího napětí procesoru. Rezistor R_{27} slouží k omezení proudu diodou D_1 v případě, že je vypnuto napájení procesorové části.

Nabíjení akumulátoru

Jako bateriový zdroj zařízení se předpokládá použití Li-Ion baterie s nominální hodnotou výstupního napětí 3,7V. Pro tuto baterii se jako nejvhodnější způsob dobíjení hodí obvod společnosti Texas Instruments, typ BQ2057. Schéma zapojení nabíječky je uvedeno na obrázku 3.12.



Obr. 3.12: Schéma zapojení nabíjecího obvodu

Integrovaný obvod je zapojen podle katalogového listu výrobce [9]. Odtud vycházejí prakticky všechny použité součástky. Rezistor $R_{22} = 0R22$ slouží k měření nabíjecího proudu, tranzistor Q_4 slouží jako výkonový prvek pro regulaci nabíjecího proudu. Pro kontrolu stavu baterie slouží rezistory R_T. Jejich osazení je volitelné, pokud nejsou osazeny, měl by být pin *SENS* připojen přes konektor *BATT1* k měřicímu pinu baterie. Jako vhodná baterie se jeví standardní baterie z mobilního telefonu s napětím 3,7V.

Nabíjení je možné vypnout pomocí mikrokontroléru signálem *nCHRGR_EN*. Tímto pinem je spínán tranzistor T_1, jehož prostřednictvím je výstup *SENS* spojen se zemí.

3.2 Realizace základní desky zařízení

Deska plošného spoje je navržena na základě zapojení popsaného v předchozí kapitole. Při tvorbě desky byl kladen důraz na dodržení základních pravidel, kterými jsou:

- Vzdálenosti prvků
- Minimální rozměry
- Užití blokovacích kondenzátorů
- Rozlitý polygon signálu *GND*

Některé součástky a jejich předlohy nejsou součástí standardních knihoven programu, proto bylo nezbytné pro chybějící součástky vytvořit jejich elektronické předlohy (rozložení vývodů, schematické značky atd.).

Pro usnadnění rozvodu napájecích cest byl rozlit navíc polygon se signálem *+3V3* pod tělem procesoru. Je také kladen důraz na to, aby napájecí linky, které přenášejí větší proud, byly dostatečně široké.

Zvláštní přístup byl aplikován při tvorbě návrhu napájecího zdroje. Protože bylo riziko, že se bude při zátěži zahřívat, byl umístěn do rohu desky a navíc oddělen od hlavního polygonu tenkou mezerou, aby nedocházelo k rozvodu tepla do zbytku desky.

Obrázek desky plošného spoje je z praktických důvodů vložen do přílohy. Vzhledem k tomu, že celé schéma je kresleno v programu Eagle, je i rozložení součástek a vodivých cest tvořeno v tomto softwaru.

S tvorbou desky přímo souvisí i volba konkrétních typů součástek včetně dodavatelů. Proto byl vyhotoven souhrnný seznam součástek. Při objednávání součástek od dodavatele může být často výhodné objednat více součástek stejné nominální hodnoty. Často se také stává, že součástka je dostupná v minimálním množství pěti kusů apod. (například kondenzátory, rezistory...) Proto byla provedena optimalizace součástek tak, aby bylo možné snížit počet hodnot rezistorů apod. Příkladem může být situace, kdy výpočtem je zjištěna hodnota rezitoru 2k, avšak z charakteru obvodu je zřejmé, že postačí hodnota 2k2.

Z uvedeného samozřejmě vyplývá, že těmito úpravami nesmí dojít k narušení funkce obvodu.

4 SOFTWAROVÉ VYBAVENÍ

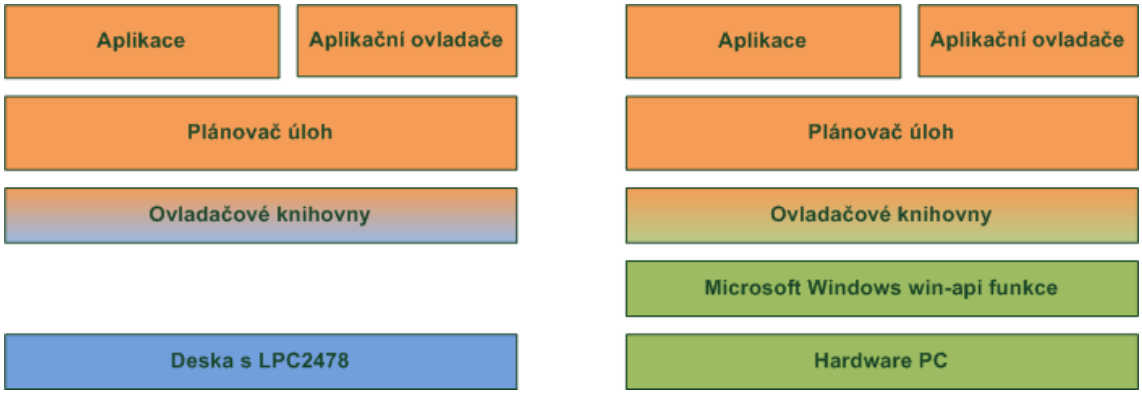
Z pohledu návrhu software se jedná o poměrně komplexní úlohu, kterou je potřeba rozdělit do několika logických celků. Těmito celky jsou:

- Firmware zařízení
- Ovládací software pro OS Windows
- Ovládací software pro OS Linux

Tato kapitola bude tedy logicky rozdělena, aby bylo možné přehledně popsat jednotlivé části softwaru.

4.1 Firmware zařízení

Ihned na počátku této kapitoly je nezbytné uvést, jak byla navržena struktura softwaru. Struktura je zobrazena na obrázku 4.1.



Obr. 4.1: Blokové schéma firmwaru zařízení

Jsou zde zřejmé dvě konfigurace. Levá konfigurace je standardní firmware uložený do procesoru, zatímco pravá konfigurace odpovídá alternativnímu debuggovacímu prostředí v MS Windows, která je použita pro ladění softwaru bez dostupnosti hardwaru. Obarvení bloků symbolizuje, že oranžová část softwaru je neměnná (alespoň ve významném měřítku), zatímco modrá a zelená část je odlišná pro každé vývojové prostředí. V zájmu možnosti snadného přechodu mezi oběma prostředími bylo nezbytné co nejstriktněji oddělit funkce závislé na hardwaru od funkcí, které jsou čistě algoritmické.

Vzhledem k volbě procesoru (tím je LPC2478) bylo nejprve nezbytné vybrat vývojové prostředí. Volba vývojového prostředí je také ovlivněna i volbou progra-

mátoru, který bude použit (jedná se o programátor SB JTAG[1]). Mezi možné volby patřilo například prostředí Keil uVision, dále například Eclipse s pomocí rozhraní OpenOCD, případně CrossStudio for ARM. Po několika testech bylo zvoleno prostředí CrossStudio for ARM díky možnostem ladění, které nabízí, také díky technické podpoře a možnosti neomezené velikosti generovaného softwaru.

Výhodou zvoleného prostředí je plná podpora procesoru i z hlediska dostupných knihoven. Aby bylo možné procesor plnohodnotně používat, je nezbytné při jeho startu provést inicializaci. Tato je provedena s pomocí balíčku *Embedded Artists LPC2478 OEM Board Support Package*, který lze zdarma do prostředí nainstalovat prostřednictvím průvodce obsaženého v prostředí.

Jako alternativní prostředí bylo vybráno Microsoft Visual Studio 2010.

Software je psán v programovacím jazyce C++, bohužel však při programování embedded zařízení je nutné vyvarovat se některých specifik tohoto jazyka a určité oblasti software psát na úrovni běžného jazyka C. Důvodem je jednak omezená podpora jazyka Embedded C++, jenž jednoduše neobsahuje určité vlastnosti klasického C++. Mezi ně patří například výjimky, šablony apod.

Bohužel však to, že kompilátor podporuje některou vlastnost, nemusí znamenat, že vlastnost bude správně pracovat v mikrokontroléru. Během vývoje firmwaru jsem narazil na nečekané problémy s globálními konstruktory tříd. Ty pracovaly naprosto správně v prostředí Visual Studio, nicméně v prostředí Cross Studio konstruktory nebyly po startu volány a instance tříd tedy v paměti sice existovaly, ale nebyly inicializované. Bylo tedy nutností konstruktory nahradit standardními metodami Init(), jež třídu zinicializovaly "ručně".

Jak je vidět na obrázku, je software určitým způsobem logicky členěn. Toto členění bude shrnuto v následujícím seznamu:

- Operační systém
- Ovladačové knihovny
- Aplikační ovladače
- Aplikace
- Komunikace s PC s OS Windows
- Komunikace s PC s OS Linux
- Simulátor

4.1.1 Operační systém

Pro správnou funkci zařízení a efektivní běh softwaru a jednodušší ladění bylo na počátku tvorby firmwaru nezbytné rozhodnutí, jak koncipovat a realizovat vlastní

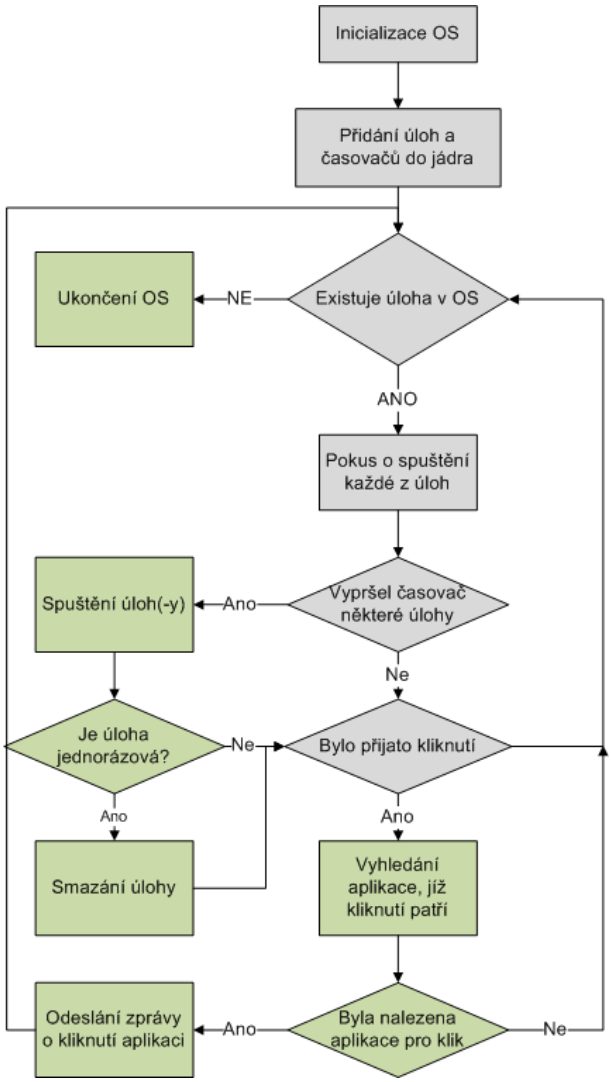
[1]Autorem je Jiří Bezstarosti, bohužel stránky, ze kterých je programátor převzat, jsou již nedostupné.

operační systém zařízení. Existovala možnost použít jeden z dostupných systémů reálného času, například FreeRTOS, případně použít operační systém na bázi Linuxu.

Toto řešení se zpočátku jevilo jako zbytečné. Proto jsem se rozhodl vytvořit jednoduchý plánovač úloh realizovaný knihovnou pro správu procesů, která se později ukázala jako zcela dostačující. Navržený plánovač umožňuje následující akce:

- Cyklické spouštění úloh
- Časování úloh
- Správa úloh a časovačů
- Nastavování priorit úloh
- Zasílání zpráv mezi procesy
- Správa události kliknutí na displej

Pracovní cyklus plánovače úloh je uveden na obrázku 4.2



Obr. 4.2: Cyklus plánovače úloh

Z obrázku je evidentní, že systém běží v nekonečné smyčce, která končí pouze v případě, že v systému neexistují žádné úlohy. Barva bloků značí, které akce se provádí v každém cyklu OS (šedá) a které akce se provádí pouze v případě, že nastala určitá událost (zelené).

Základem plánovače úloh jsou dvě pole, pole procesů a pole časovačů. Proces je tvořen strukturou *TASK*, časovaná událost je uložena ve struktuře typu *TIMER*.

Definice vlastností struktury TASK je následující:

```
typedef struct TASK
{
    void (*task)(void);                 //Funkce prirazena uloze
    TASK_NAME name;                     //ID ulohy
    PRIORITY priority;                  //Priorita ulohy
    unsigned char remaining;            //Zbyvajici pocet cyklu do spusteni
    STATE state;                        //Stav ulohy
    char *internalState;                //Odkaz na interni stav aplikace
    TASK_TYPE type;                     //Typ ulohy
    RESULT(*msgBox)(MESSAGE *msg);      //Schranka pro zpravy prirazena uloze
    TRect *LCD_area;                    //Oblast displeje
} TASK;
```

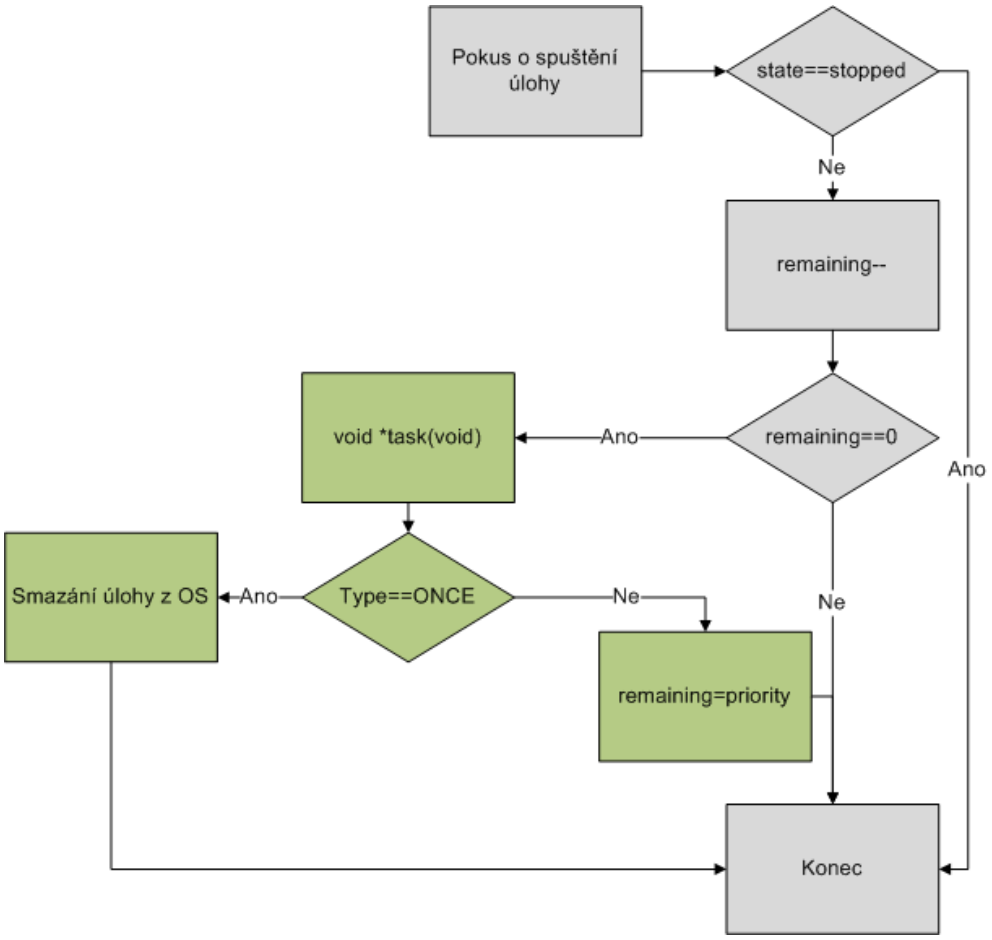
Každá úloha musí mít přiřazenou funkci, která je jejím základem. Dále musí mít úloha přiřazené ID, kterým je identifikována. Toto ID by mělo být pro správnou funkci jednoznačné. Priorita úlohy je celé číslo v rozsahu $< 0; n >$, kde n je libovolné číslo. Vyšší číslo značí nižší prioritu. Toto číslo je po spuštění úlohy (nebo po vložení úlohy do systému) kopírováno do proměnné *remaining*. Při každém pokusu o spuštění úlohy je tato proměnná dekrementována, ovšem ke spuštění funkce **task* dojde až v případě, že je toto číslo nulové.

Pro ilustraci, jak je úloha v systému spouštěna, slouží vývojový diagram 4.3.

Proměnná *state* definuje, jestli je spuštění úlohy povoleno, proměnná *type* definuje, jestli je úloha jednorázová nebo je spuštěna cyklicky. Zasílání zpráv je provedeno voláním funkce **msgBox* s patřičným parametrem **msg*, který je ukazatelem na zprávu.

Zpráva pro aplikaci/proces je tvořena následující strukturou:

```
typedef struct MESSAGE
{
   MESSAGE_CORE msg;    //zprava
   void* data;          //ukazatel na data
}MESSAGE;
```



Obr. 4.3: Životní cyklus úlohy v systému

Jádro sdělení je proměnná definovaná jako enum (tedy celé číslo). Touto proměnnou je definován typ dalších dat, na něž ukazuje ukazatel **data*. Velmi často se stává, že ke zprávě není třeba připojovat žádná data (například zpráva RUN, STOP, případně REFRESH, INIT atd.), pak je ukazatel inicializován na nulu. Opačným příkladem je zpráva APP_CLICK, jejímž obsahem je pozice kliknutí.

Aby mohla aplikace přijímat kliknutí, musí mít v operačním systému uloženu informaci o rezervované oblasti na displeji (proměnná *LCD_area*). Systém po přijetí kliknutí vyhodnotí, které aplikaci je třeba doručit toto kliknutí, a pošle jí souřadnice události.

Časované události (tzv. časovače) pracují velmi podobným způsobem, jsou však definovány strukturou *TIMER*, jejíž popis je uveden níže.

```
typedef struct TIMER
{
    void(*timer)(void);       //callback casovace
```

```
    TIMER_NAME name;          //ID casovace
    int interval;             //Interval v ms
    int remaining;            //zbyvajici cas pro spusteni
    TIMER_STATE state;        //stav
    TIMER_TYPE type;          //typ
}TIMER;
```

V knihovně časovače je vytvořeno pole těchto struktur. Ty jsou při každém vyvolání přerušení všechny kontrolovány a je dekrementována jejich členská proměnná *remaining*. Plánovač v každé své smyčce toto pole kontroluje a struktury, jejichž *remaining <= 0*, mají tuto proměnnou re-inicializovánu na hodnotu *interval*. Poté je zavolána jejich funkce **timer(void)*. Každý časovač má opět nadefinovanou proměnnou *state*, jež definuje, jestli časovač běží. Dále je definován typ časovače *type*, který značí, jestli je časovač periodický nebo jednorázový. V případě jednorázového časovače je tento po spuštění odebrán z jádra plánovače.

Aby bylo možné jednoduchým způsobem měřit čas, který uplynul od určitých událostí, je v systému s každým vyvolaným přerušením od časovače inkrementována proměnná *timeStamp*. Uživatelská aplikace může zavoláním funkce *GetTimeStamp* získat aktuální hodnotu proměnné a následně při volání funkce *GetElapsedTime()* zjistit, kolik násobků základního časového intervalu uplynulo. Předpokladem je inkrementace proměnné po 1 ms.

Nevýhodou tohoto plánovače úloh je nemožnost dodržení nároků na systém reálného času, protože aplikacím nelze odebrat zdroje po jejich spuštění. Může se tedy stát, že některá z aplikací tu a tam zabere čas procesoru i na velmi dlouhou dobu (například při vykreslování obrazovky nebo zpracování obrazových dat z PC). Tím může nastat situace, že systém nemůže zavolat callback patřičných časovačů včas a ten proběhne se zpožděním (například o 1-2ms později). Tento problém však není v našem zařízení nijak kritický, a tudíž je možné jej ignorovat.

Knihovny plánovače úloh nejsou žádným způsobem závislé na hardwaru, lze je tedy bez problémů využít v simulátoru zařízení i na reálném hardware.

4.1.2 Knihovny ovladačů

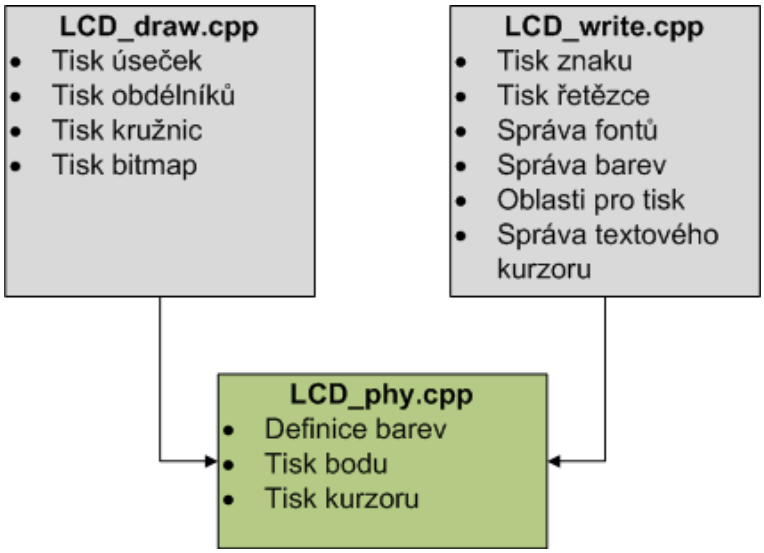
Tato část softwaru přímo ovládá hardwarové prostředky systému, dovoluje také aplikacím ovládat periferie procesoru a další části desky. Knihovny se vyznačují tím, že pracují přímo s hardwarem. Patří mezi ně:

- Ovladač grafického rozhraní
- Ovladač UARTu
- Ovladač snímače dotykové plochy

- Ovladač podsvětlení
- Ovladač LED
- Ovladač měření baterie

Ovladač grafického rozhraní

Ovladač grafického rozhraní je patrně jedním z nejsložitějších ovladačů vytvořených v rámci této práce. Byl částečně převzat z bakalářské práce [12] a podstatným způsobem upraven a rozšířen. Tato knihovna je navržena tak, aby byla minimálně závislá na hardwaru a umožňovala tak s minimálními změnami použití jiného displeje, případně jiného zobrazovacího zařízení (okno v OS Windows). Struktura knihovny a podporované funkce jsou uvedeny na obrázku 4.4.



Obr. 4.4: Struktura grafické knihovny

Všechny funkce z vyšších knihoven (GLCD_draw.cpp a GLCD_write.cpp) jsou z hlediska hardwaru závislé pouze na funkcích z knihovny GLCD_phy.cpp, jež obsahuje funkce pro práci s jednotlivými pixely (zabarvení pixelu na pozici, ztmavení pixelu), případně práci s celým obrazovým zásobníkem dat (uložení zásobníku, obnovení zásobníku).

Obrazový zásobník je vlastně pole bajtů o velikosti definované šířkou, výškou a barevnou hloubkou. Z tohoto pole bajtů jsou data při každém obnovení displeje zkopírována na obrazovou plochu displeje (tedy přibližně 60x za sekundu). Kreslení probíhá následujícím způsobem:

Nejprve je vypočtena pozice v poli pro kreslení:

$$P = x + (HCPL + 1) * y \tag{4.1}$$

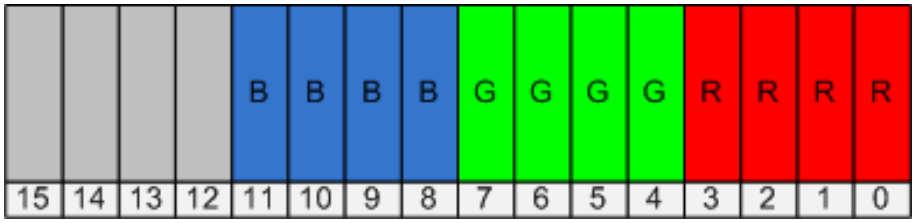
kde *HCPL* je teoretická šířka displeje a x a y jsou souřadnice požadovaného bodu. Poté je pixel o požadované barvě uložen do pole. Vzhledem k internímu řešení procesoru není teoretická šířka displeje stejná jako skutečná, proto je *HCPL* definováno jako počet hodinových pulzů, které řadič potřebuje pro přejetí jedné řádky, tedy 525. Zde se vychází z katalogového listu procesoru [2] a katalogového listu řadiče displeje [11].

Kreslení různých geometrických útvarů probíhá dle matematické definice pozic jejich bodů, za zmínku stojí pouze problém, který nastane při kreslení přímek na bodový displej v celočíselné aritmetice. Pokud totiž v celočíselné aritmetice tiskneme přímku se směrnicí $k < 1$, vznikne zaokrouhlovací chyba, která způsobí, že vzniklý útvar není spojitý. Je tedy třeba přímku kreslit inverzně, tedy vlastně otočenou o 90°. Problém je hlouběji rozpracován opět ve zmíněné bakalářské práci [12].

Bitmapy, které dokáže knihovna tisknout, nesmí být žádným způsobem komprimovány a musí být navíc v následujícím formátu:

```
const unsigned short bitmapa[] = {sirka, vyska, p1, p2, p3, ...};
```

Ke generování obrázků v tomto formátu byla vytvořena jednoduchá funkce v Matlabu[2], která provede načtení standardní bitmapy, poté provede zápis hlavičky, šířky a výšky a následně jednotlivé body obrázku převede do patřičného formátu a zapíše je do souboru. Tento formát je při 12-ti bitové hloubce následující:



Obr. 4.5: Rozložení barevných složek pixelu v typu short

Vzniklý soubor lze vložit do projektu v jazyce C a pomocí funkcí knihovny vykreslit obrázek[3].

Při popisu způsobu tisku znaků je nejprve nezbytné zmínit, že veškeré fonty použité pro tuto práci (je myšlen firmware zařízení) jsou generovány na základě předloh ze systému Windows (jedná se o písma *Terminal*, velikost 6, tučné a *Courier*, velikost 15, tučné). Písma byla zpracována pomocí konvertoru společnosti Fujitsu *Font Converter Tool*, jenž lze stáhnout z [16]. Tyto fonty byly vybrány pro svou jednoduchost. Obsahují totiž spojité široké linie bez významných jasových změn.

[2]Ukázkový skript je umístěn v elektronických přílohách práce.
[3]Funkce LCD_draw_bitmap a LCD_draw_bitmap_cond

Znaky ze zadaného fontu jsou uloženy do hlavičkového souboru, kde jsou reprezentovány sadou čísel. Hlavičkový soubor obsahuje definici dvojrozměrného pole:

```
const unsigned int Courier12[2][19] =
{
        {0x00000000,0x00000000,0x00000000,0x7F000000,...},
        {0x00000000,0x00000000,0x00000000,0x1E000000,...}
};
```

První index pole (zde 2) značí, že byly vygenerovány a jsou dostupné 2 znaky. Druhý index značí, že každý znak má 18 řádků (poslední pozice je nulová, je přidána automaticky). Každá pozice řádku pole definuje jeden řádek znaku, přičemž každý bit čísla značí jeden pixel (je-li bit 1, pixel je aktivní). Jelikož je použitá šířka znaků maximálně 16, lze oříznout poslední čtyři nuly z každé definiční pozice, protože jsou vždy nulové. Poté je možné změnit definici pole na const unsigned short. Tento typ má poloviční nároky na paměť, a tudíž i výsledný program je méně paměťově náročný.

Každý ze znaků lze tisknout na libovolnou pozici v obraze, lze použít libovolnou barvu písmene a pozadí. Typ písma je definován následující strukturou:

```
typedef struct FONT
{
    RGB_COLORS color;
    FONT_SIZE size;
    RGB_COLORS bg_color;
}FONT;
```

Proměnné *color* a *bg_color* definují barvu písmene a pozadí, proměnná *size* definuje velikost písma. Aktuálně jsou implementovány dvě velikosti, *FONT_MINIMAL*, což je alias pro písmo *Terminal*, a *FONT_SMALL* pro písmo *Courier*. Teoreticky je knihovna připravena celkově pro pět fontů, nicméně jejich šablony nejsou nadefinovány.

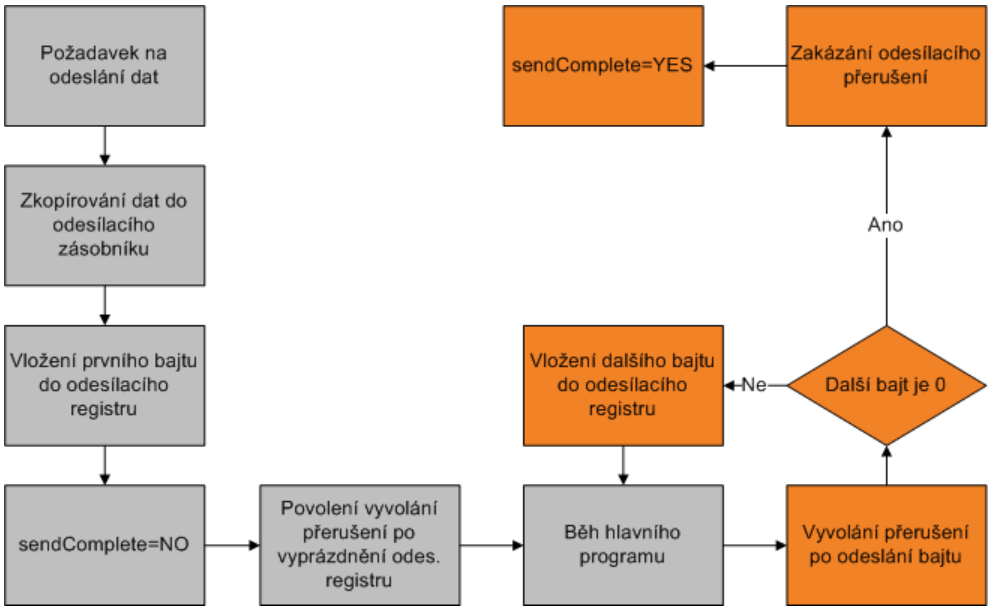
Tisk řetězců probíhá do takzvaného ROI [4], což je tvarem vlastně obdélník. Ten je třeba nastavit před tiskem textu. Tato oblast zájmu značí, kam je dovoleno tisknout znaky, tudíž z jaké oblasti by se neměl kurzor dostat ven. Pokud kurzor přeteče přes okraj (pravý, případně dolní), vrací se na začátek oblasti (začátek dalšího řádku nebo levý horní roh ROI). Tento algoritmus usnadňuje volbu pozic pro tisk, není totiž nezbytné, aby vyšší vrstvy programu kontrolovaly, jestli nedošlo k přetečení apod.

[4] Region of Interest, resp. oblast zájmu.

Knihovna podporuje několik způsobů tisku řetězců. Prvním z nich je neformátovaný tisk znaků (znak po znaku, jež vyplňují ROI)[5]. Dále je možné zavolat funkci, která nejprve počítá, kolik znaků má každé slovo, a teprve pokud se toto slovo vejde před konec ROI, vytiskne jej, jinak jej tiskne na nový řádek[6]. Tuto funkci je vhodné použít pokud pracujeme s běžným textem, který se nevejde na jeden řádek. Velmi specifická je funkce pro tisk konzolového textu[7]. Ta totiž dovoluje formátovat každý znak v textu zvlášť, nicméně vyžaduje zvláštní vstupní parametry. Popisu této funkce bude věnován prostor v dalším textu.

Ovladač UARTu

Ovladač UARTu je navržen tak, aby co nejméně zabíral procesorový čas během přerušení. Hlavní program je tedy přerušen pouze na dobu nezbytně nutnou k vykonání nejnezbytnějších operací. Z hlediska obsluhy sériového rozhraní rozlišujeme dvě základní situace, příjem a odesílání dat. Postup pro odeslání dat je uveden na obrázku 4.6.



Obr. 4.6: Stavový diagram odesílání dat ze zařízení

Nejprve jsou zkopírována data do odesílacího zásobníku, tím je zaručeno, že zůstanou po celou dobu odesílání neměnná. Poté je povoleno přerušení, které je vyvoláno při vyprázdnění odesílacího registru. Následně je nastavena proměnná *sendComplete* do stavu *NO*. Odesílání je zahájeno vložením prvního bajtu do odesílacího

[5]Funkce LCD_printf
[6]Funkce LCD_cprintf
[7]Funkce LCD_console_printf

registru. Když je bajt odeslán, je vyvoláno přerušení, v němž je vložen do registru další bajt. Pokud jsou všechny bajty odeslány, je zakázáno odesílací přerušení a proměnná *sendComplete* je nastavena do stavu *YES*. V diagramu značí oranžová barva operace v přerušení a šedá ostatní operace.

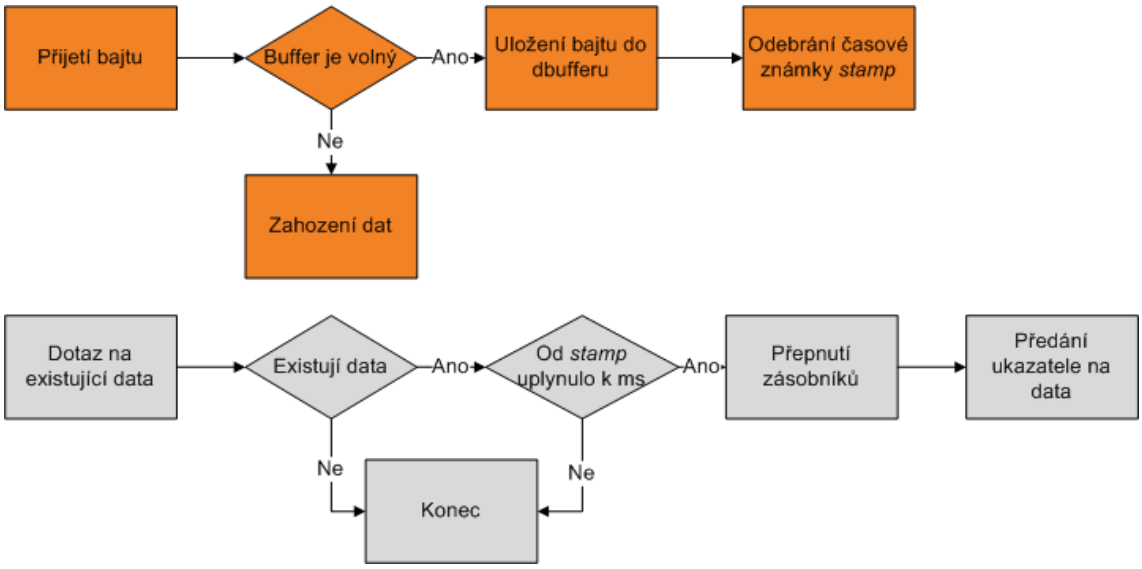
Příjem dat je složitější, protože je nutné rozpoznat, jestli přijímání spojitého bloku dat skončilo. Pro příjem dat byla vytvořena zvláštní třída s názvem *dbuffer*. Její definice je následující:

```
class dbuffer
{
public:
    unsigned char A[DBUFFER_MAX_LENGTH];    //Zasobnik s daty
    int top;                                //Vrchol zasobniku
    ANSWER Lock;                            //Zamek zasobniku
    ANSWER DataRdy;                         //Data pripravena
    dbuffer(void);                          //Konstruktor
    ~dbuffer(void);                         //Destruktor
    TIME_STAMP stamp;                       //Casova znamka poslednich dat
    int GetData(unsigned char * out);       //Vycteni dat
    void Clear(void);                       //Vycisteni zasobniku
    int GetCount(void){return top;}         //Vraci pocet bajtu v zasobniku
    ANSWER AddData(char in);                //Vlozi data do zasobniku
};
```

Třída má veškeré své proměnné veřejně přístupné, kdyby se jednalo o program pro PC, nebylo by toto zcela korektní řešení. Vzhledem k tomu, že tento software musí být co nejrychlejší vzhledem k omezenému výkonu jednočipu, je třeba tento nedostatek ignorovat.

V paměti jsou umístěny dva identické zásobníky, přičemž vždy jeden slouží k příjmu dat a druhý ke čtení přijatých dat a jejich zpracování. Existuje předpoklad, že data budou rychleji zpracována, než přijímána. To sice není zaručeno na straně zařízení, nicméně je to zaručeno ze strany řídícího softwaru v PC (bude popsáno dále v této práci). Stavový diagram je uveden na obrázku 4.7

Existuje snaha, aby přijímaná data byla pokud možno logicky segmentována po jednotlivých balíčcích. Pokud by tomu tak nebylo, nastávaly by komplikace s rozdělováním dat na logické úseky. Proto třída *dbuffer* obsahuje jako jednu ze svých proměnných proměnnou *stamp*, jež obsahuje systémový čas okamžiku, kdy byla odebrána. Při kontrole, jestli byl přijat blok dat, je kontrolována i tato časová známka a je kontrolováno, jestli uplynul dostatečný časový interval od posledních přijatých dat. Pokud ano, může si nadřazená aplikace převzít tato data a dále s nimi pracovat.



Obr. 4.7: Stavový diagram příjmu dat zařízením

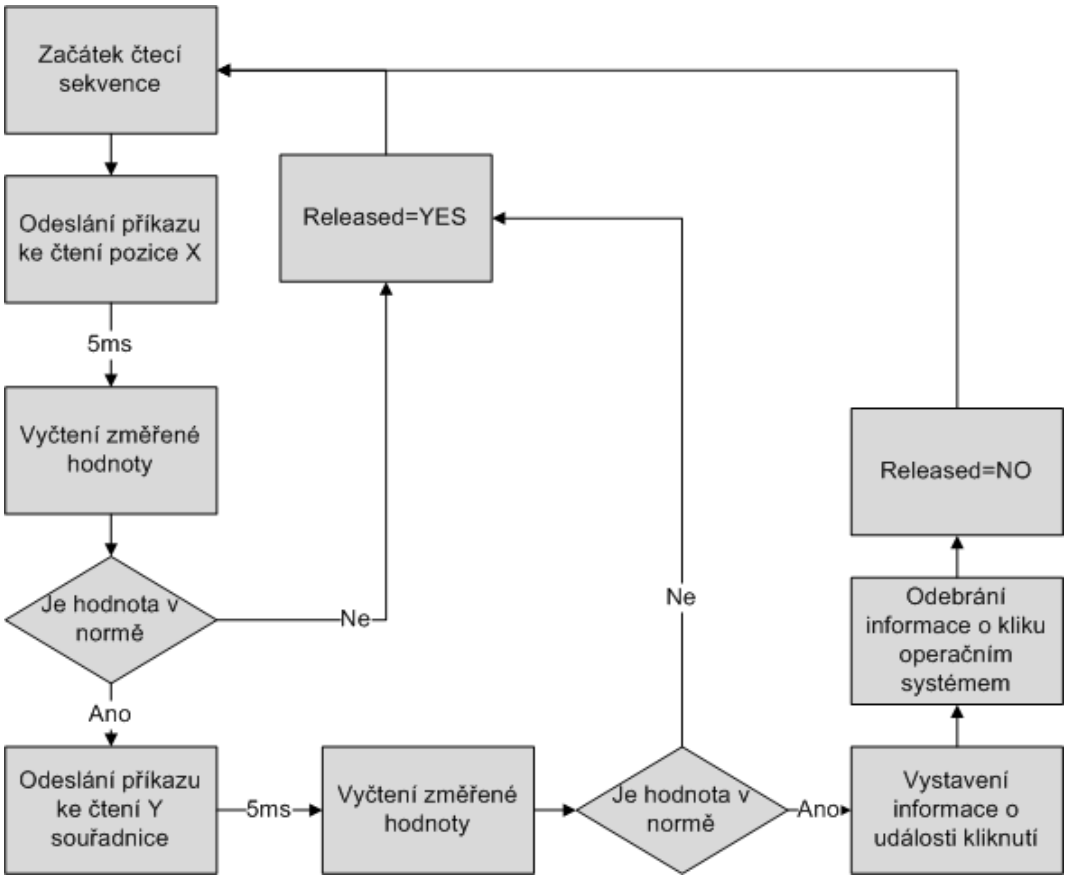
Ovladač snímače dotykové plochy

Snímání dotykové plochy probíhá pomocí extermího čipu, jenž je s mikrokontrolérem propojen pomocí sběrnice I^2C. Tato sběrnice je výhodná, pokud není nutná velká komunikační rychlost a je požadována jednoduchost řízení. Tento text nebude zaměřen na popis rozhraní I^2C a jeho funkci, protože popis lze nalézt v množství jiných textů. Je zde popsán pouze algoritmus, kterým probíhá zadávání příkazů externímu čipu a následně vyčítání dat.

Vyčítání z čipu je navázáno na plánovač úloh, resp. na systém časovačů. Jedním z časovačů je totiž funkce, jež obsahuje stavový automat popsaný na obrázku 4.8. Vyčítání z čipu probíhá cyklicky, čas mezi jednotlivými kroky je přibližně 5ms. V základním provedení knihovna podporuje jednoduchá kliknutí. Pro lepší orientaci v chování uživatele k displeji je zde zavedena proměnná *wasReleased*, v diagramu značená jako *Released*, jež dává informaci, jestli uživatel na ploše drží prst (případně stylus), nebo je plocha volná. Této vlastnosti lze s výhodou využít při simulaci Touchpadu, jejíž popis je součástí dalších podkapitol.

Ovladač podsvětlení

Ovladač podsvětlení pracuje jako jednoduchý čítač (navázaný opět na čas systému). Aplikace, ovladače či plánovač úloh informují pomocí volání funkce *BacklightRequest*, že nastala událost, která vyžaduje, aby bylo aktivní podsvětlení displeje. Z několika důvodů není podsvětlení rozsvěceno nárazově, ale postupně po rampě, která je definována v konfiguraci podsvětlení. Důvodem je zejména snaha omezit proudové



Obr. 4.8: Stavový diagram vyčítání polohy kliknutí na displej

nárazy při spuštění podsvětlení (podsvětlení při plném výkonu odebírá ze zdroje přibližně 100mA). Druhý důvod je spíše estetického rázu, zapnutí a vypnutí po rampě jednoduše lépe vypadá.

Konfigurace podsvětlení je uložena v následující struktuře:

```
typedef struct BACKLIGHT_SETUP_STRUCT
{
    short RampUp;           //strmost zapinaci rampy
    short RampDown;         //strmost vypinaci rampy
    int TimeOut;            //cas podsviceni
    int HardTimeOut;        //cas vypnuti displeje
    short DACMinimum;       //Minimalni proud diodami
    short DACMaximum;       //Maximalni proud diodami
}BACKLIGHT_SETUP_STRUCT;
```

Proměnné *RampUp* a *RampDown* definují diferenci mezi dvěma kroky úpravy jasu. Dále jsou definovány proměnné *TimeOut*, jež definuje maximální čas, po který

má displej svítit po posledním podnětu, a *HardTimeOut*, která definuje, jak dlouho má být zapnutý řadič displeje po posledním podnětu. Funkce vypínání displeje však není v tomto firmwaru definována z důvodu nedostatku času.

Ovladač LED

LED diody slouží v zařízení k indikaci stavů výrobku uživateli. Jejich obsluha je velice jednoduchá, protože jsou přímo připojeny ke GPIO portu. Jejich ovládání tedy spočívá pouze v nastavení příslušných pinů do log. 1 resp. log. 0. Není zde tedy třeba popisovat žádné vývojové diagramy.

Ovladač pro měření stavu baterie

Měření baterie probíhá pomocí plovoucího průměru z několika měření. Úloha měření je opět navázána na časovač, který v definovaném intervalu (aktuálně 5s) spouští plánovač úloh. Časovaná úloha aktivuje AD převodník a zadá periferii pokyn k převodu. Převod trvá několik milisekund, proto není vhodné čekat na dokončení úlohy. Zdroje jsou tedy předány ostatním aplikacím. Když je převod dokončen, je vyvoláno přerušení a v tomto přerušení je vložena změřená hodnota do pole posledních pěti hodnot. Pokud aplikace potřebuje zjistit hodnotu napětí na baterii, vrací jí knihovna průměrnou změřenou hodnotu. Tu si aplikace poté musí přepočítat na napětí.

Přepočet na napětí probíhá podle následující rovnice:

$$U_{bat} = k * \frac{U_{ref} * H_{ADC}}{1024} \tag{4.2}$$

U_{bat} je hodnota napětí na baterii, U_{ref} je hodnota referenčního napětí, což je napětí, kterým je napájen procesor, H_{ADC} je hodnota získaná z AD převodníku a k je převodní konstanta vstupního děliče napětí.

4.1.3 Aplikační ovladače

Jedná se o jednoduché aplikace, které běží v systému a mají za úkol zprostředkovat vyšším aplikacím spojení s nižšími knihovnami. Hlavním úkolem těchto knihoven bylo propojení funkcí, které pracují na principu jednorázového volání, jako je zadání dat UARTu, s knihovnami, jež pracují na principu stavového automatu. Výhodou těchto knihoven je, že jsou integrovány do jádra systému jako procesy, tudíž je možné jim zasílat zprávy, také jim lze nastavovat priority apod.

Mezi zmíněné knihovny patří zejména tyto:

- Ovladač sériového rozhraní
- Ovladač Wifi
- Inicializátor

Ovladač sériového rozhraní

Tato aplikace slouží pro správu dat, která mají být odeslána přes sériovou linku. Je v softwaru zařazena z toho důvodu, že čas od času vznikají nároky na odeslání většího počtu balíčků dat. Aby mohla být data bezprostředně odeslána, musí být ovladač UARTu volný. Pokud tedy právě probíhá odesílání, není možné začít odesílat další data.

Navíc je třeba vyřešit adresování dat na dva UARTy, z nichž každý ústí na jedno komunikační rozhraní. Jeden UART spojuje procesor s Wifi modulem, druhý umožňuje komunikaci přes sériový port.

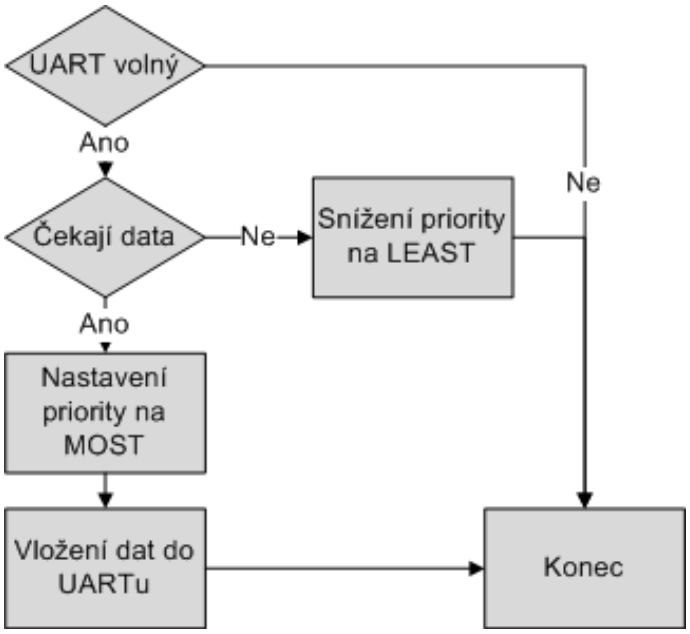
Problém byl vyřešen pomocí jednoduché třídy, jež pracuje jako kruhový buffer.

```
typedef struct CBUFFER_DATA
{
    unsigned char data[CBUFFER_SLOT_LENGTH];    //Data k odeslani
    OS_CONNECT target;                          //Cil dat
    ANSWER completed;                           //Hotovo
}CBUFFER_DATA;

class CBuffer
{
private:
    CBUFFER_DATA dataBuff[CBUFFER_SLOTS_COUNT];//Zasobnik s daty
    char dataTop;                              //Neodeslana data
    char dataCurrent;                          //Aktualni data
public:
    CBuffer(void);                             //Konstruktor
//Vlozeni dat
    ANSWER InsertData(OS_CONNECT target, unsigned char * input);
    ANSWER SendNextData(void);                 //Odeslani dat
    ANSWER DataPending(void);                  //Cekajici data
    ANSWER DataFull(void);                     //Zasobnik plny
    void Init();                               //Inicializace
    ~CBuffer(void);                            //Destruktor
};
```

Data k odeslání jsou uložena ve struktuře *CBUFFER_DATA*, přičemž u každého paketu je uložen cíl (Wifi nebo com port) a flag *completed* s informací, jestli byla tato sekvence zpracována.

Cyklus práce aplikace je uveden ve stavovém diagramu 4.9.



Obr. 4.9: Stavový diagram odesílání dat

Z principů, které jsou použity v ovládání sériové linky vyplývá, že nelze korektně požadovat odeslání dat přes com port a zároveň komunikovat s Wifi modulem. Tuto nevýhodu lze však snadno odstranit několika drobnými úpravami v knihovně *uart.cpp*. Toto řešení by však vyžadovalo dvojnásobné množství paměti.

Ovladač Wifi

Ovladač Wifi modulu, jak již z názvu vyplývá, ošetřuje spojení s Wifi modulem, vyhodnocuje zprávy přijaté od něj a detekuje stav modulu. Ovladač se může nacházet ve dvou režimech. Režim aktivního řízení způsobí, že ovladač odebírá data přijatá z modulu a vyhodnocuje je jako odpověď na příkazy. To ale není výhodná vlastnost, pokud komunikujeme s modulem, který je v režimu SerialNet. V tom případě je ovladač přepnut do neaktivního režimu a data, která přichází od Wifi, jsou zpracována přímo vyšší aplikací.

Komunikace s modulem probíhá na principu otázka/příkaz-odpověď. Komunikační protokol výrobce modulu nazval AT+i protokol. Název vychází z gramatiky příkazů, kdy každý příkaz začíná touto sekvencí. Každá odpověď modulu končí naopak sekvencí I/status, kde status identifikuje výsledek příkazu. V praxi je možné se setkat s odpověďmi typu I/OK, I/ERROR (n), I/ONLINE, I/OFFLINE.

Jelikož tento způsob komunikace je poměrně těžkopádný, podporuje modul režim *SerialNet*. Tento režim přepne modul do stavu, kdy udržuje aktivně spojení s protějškem pomocí TCP protokolu, případně UDP spojení nebo Telnet spojení, avšak

přijatá data okamžitě přeposílá na sériové rozhraní, případně data ze sériové linky posílá přes TCP nebo UDP protokol protějšku. Může při tom vystupovat jako server nebo klient (my používáme Wifi v roli klienta). Volitelně přes TCP/IP protokol lze aktivovat šifrování pomocí SSL. Vzhledem k tomu, že modul v tomto režimu nepřijímá žádné příkazy, není možné zjistit jeho aktuální stav, ani stav připojení. Proto je nezbytné tyto stavy detekovat jiným způsobem, například zasíláním specifických dat ze serveru apod. Ukončení tohoto stavu je možné dvěma způsoby. První z nich je odeslání sekvence "+++"přes sériový port. Druhý z nich je složitější, jedná se o vystavení log. 1 na jednom ze vstupů modulu, ten je uveden do Safe módu, kdy je možné jej překonfigurovat standardními příkazy.

Pro aktivaci a korektní činnost šifrovacích algoritmů je třeba vygenerovat (či jinak získat) certifikát vydaný důvěryhodnou certifikační autoritou. V práci je použit jako kořenový certifikát soubor vygenerovaný pomocí balíku OpenSSL, který je podepsaný sám sebou. Tímto certifikátem jsou podepsány certifikáty zařízení i certifikáty protějšku zařízení.

Inicializátor

Tato aplikace po startu provede zaslání inicializačních zpráv všem aplikacím, inicializuje Wifi modul do výchozího stavu a spustí výchozí aplikaci, menu. Aktivace Wifi modulu probíhá následujícím způsobem. Nejprve je zaslán modulu jednoduchý příkaz AT+i, na nějž modul musí odpovědět I/OK. Když se tak stane, je odeslána zpráva pro zjištění aktuální asociované Wifi sítě. V opačném případě je odeslána sekvence "+++"pro pokus o ukončení SerialNet režimu. Vždy je čekáno přibližně 1s na odpověď. Pokud ani jeden pokus není úspěšný, firmware považuje modul za nefunkční a zobrazí varovnou ikonu ve stavovém panelu.

4.1.4 Aplikace pro komunikaci s uživatelem

Aplikací v rámci názvosloví tohoto projektu je myšlen podprogram, který komunikuje s uživatelem pomocí grafického rozhraní. Těchto aplikací je ve firmwaru několik a jako jeden celek zajišťují ovládání přístroje.

Schéma grafického rozhraní nejlépe vystihuje obrázek 4.10. Zelenou barvou jsou vyznačeny položky menu, které jsou uživatelsky přístupné. Oranžovou barvou jsou vyznačeny aplikace, které jsou spouštěny pomocí nadřazených položek v menu. Pro přehlednost jsou vynechány aplikace Keyboard (klávesnice) a Dialog, které jsou spouštěny z některých aplikací za účelem získání dat od uživatele.

Aby bylo možné aplikace jednoduše vytvářet a upravovat, jsou postaveny na společném základu. Jako všechny ostatní úlohy v systému i tyto jsou tvořeny stavovým automatem, který zajišťuje, že je možné určit u každé přesně její stav, zaslat

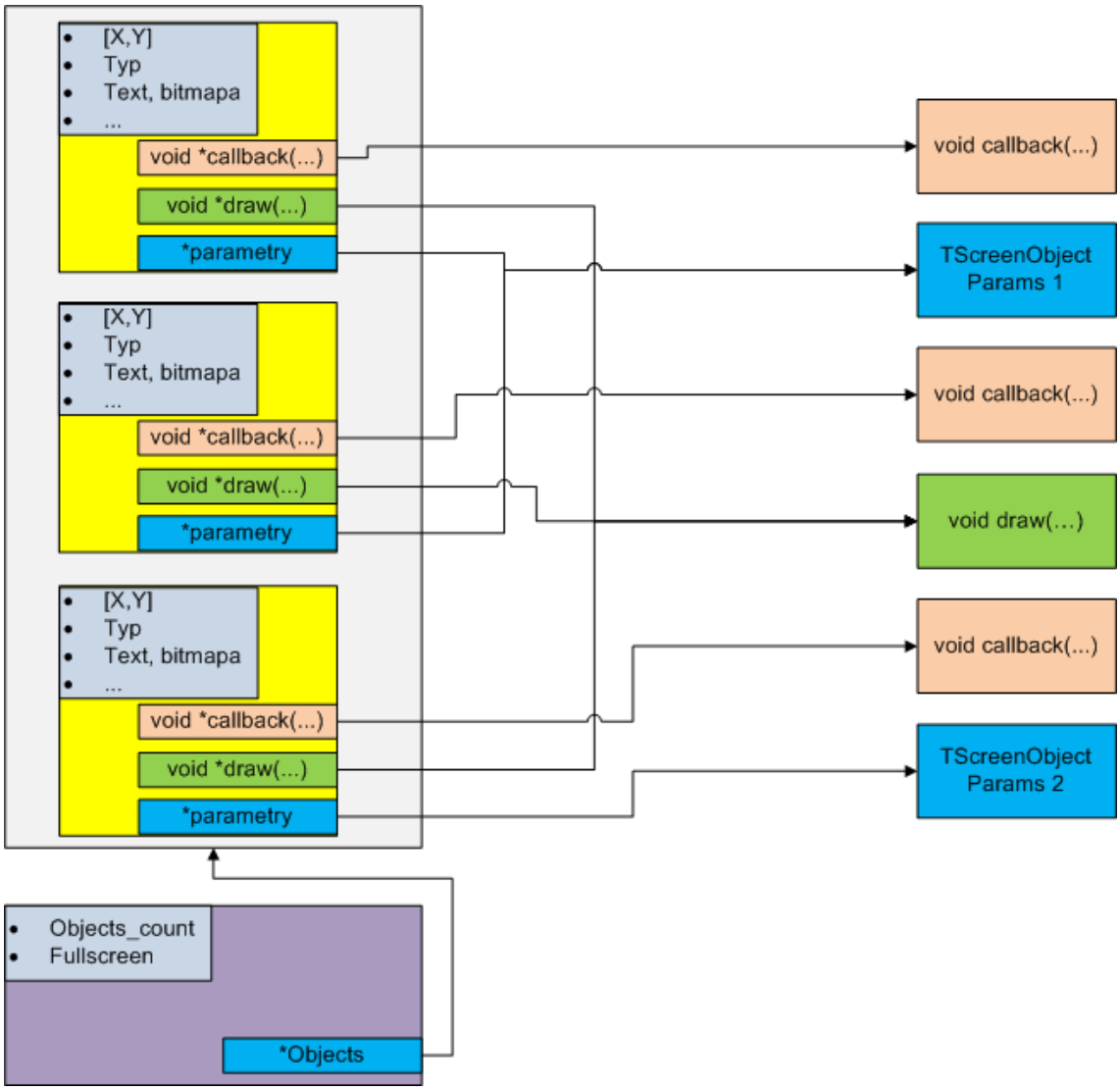
Obr. 4.10: Struktura grafického uživatelského rozhraní

jí zprávu či příkaz. Aplikace navíc obsahují grafické rozhraní reagující na podněty uživatele.

Grafické rozhraní aplikací

Za základní stavební kámen grafického rozhraní lze považovat třídy *TScreen, TScreenObject* a strukturu *TScreenObjectParams*. Struktura uspořádání objektů v paměti je zřejmá z obrázku 4.11. Pro úsporu paměti (programové i datové) je výhodné použité uspořádání, kdy několika objektům lze přiřadit společné parametry (výšku, šířku, font), případně stejné akční členy (kreslící funkce, reakční funkce apod.).

Pro správnou funkci navržených algoritmů je důležité, aby přiřazené funkce měly následující typ:



Obr. 4.11: Struktura uspořádání grafických objektů

```
void DrawCustomObject(TScreenObject * obj, OBJECT_MESSAGE msg)
{
    //telo funkce
}
```

Jako první parametr je předáván ukazatel na objekt, skrz nějž je funkce volána. Jako druhý parametr je předán pomocný parametr volání, kterým může být funkce konfigurována. Jako příklad lze uvést parametr *MSG_NO_BACKGROUND* či *MSG_CLICK*.

Předpokládá se, že každé aplikaci bude přiřazena jedna obrazovka.

Řídící algoritmy aplikací

Pro jednoduchou a spolehlivou spolupráci aplikací s plánovačem úloh je důležité, aby aplikace zachovávaly korektní chování k nadřazenému systému. Pro správu základních dat aplikací existuje pro každou aplikaci instance třídy *TApplicationInternal*. Bylo by zbytečné na tomto místě uvádět detailní definici této třídy, protože není problém prozkoumat ji přímo v kódu.

Podstatné je, že třída má za úkol provádět úkony nezbytné pro start aplikace (startem je myšlena registrace grafické oblasti na displeji pro kliknutí, spuštění stavového panelu, nastavení priority aplikace), zastavení aplikace (odregistrování grafické oblasti, apod.), zajišťuje spuštění jiné aplikace touto aplikací, získání výsledků po ukončení, překlad pozic kliknutí na index objektu v obrazovce.

Překlad pozice kliknutí na index objektu v obrazovce je zajištěn překladačem pozice. Tento překladač je definován funkcí *int translateClickPosition(POSITION *p)*. Při definování chování aplikace je možné provádět překlad pomocí interní metody tohoto typu, jež je součástí třídy *TApplicationInternal*, nebo je možné nadefinovat při inicializaci třídy vlastní překladač. Rozhodnutí, kterou z voleb použít, závisí na programátorovi a na požadavcích aplikace. Interní překladač postupně prochází veškeré objekty obrazovky, a pokud najde průnik objektu s pozicí, zavolá callback objektu s parametrem *msg = MSG_CLICK*.

V praxi může nastat situace, kdy takový přístup není možný. Pokud by v obrazovce bylo velké množství objektů, mohlo by prohledávání trvat příliš dlouho. Alternativní překladač může být potřeba i v případě, že aplikace používá dvě obrazovky. Potom je potřeba definovat, kdy patří kliknutí které obrazovce a zároveň správně ošetřit jejich callback. Toto je situace, která nastala například v aplikaci *MainMenu*.

4.1.5 Komunikace s PC s OS Windows

Komunikace s počítačem je zajištěna pomocí dvou aplikací, *Desktop* a *Console*. Rozhodnutí, kterou aplikaci použít vykoná nadřazená aplikace *Connector*.

Aplikace Desktop je určena pro komunikaci s OS Windows. Rozložení jejích grafických prvků je uvedeno na obrázku 4.12.

Aplikace s PC komunikuje pomocí krátkých zpráv. Na tomto místě budou popsány zprávy, které posílá zařízení počítači, opačný směr komunikace bude popsán v jedné z příštích kapitol. Každá zpráva má obecně formát:

```
p"D,x,y"p\r\n
```

Ve zprávě *p* značí typ odesílané zprávy, parametr *D* je zpřesňující parametr. Dále zde jsou parametry *x, y*, jež nesou samotný obsah sdělení.

Obr. 4.12: Ukázka rozložení grafických prvků aplikace *Desktop* v režimu Touchpadu

Obr. 4.13: Ukázka rozložení grafických prvků aplikace *Desktop* v režimu Plochy

Parametr p	**Popis**
m	Data týkající se kurzoru
k	Data týkající se klávesnice
a	Potvrzení

Tab. 4.1: Typy parametrů p ve zprávách

Vnitřní parametry zprávy jsou specifické pro každou ze zpráv, proto budou rozebrány tyto zprávy postupně v příslušných podsekcích.

Kurzor a příslušné zprávy

Řízení kurzoru je implementováno pomocí callbacku objektu Touchpad. Vzhledem k tomu, že implementace algoritmů pro odlišení kliknutí a tažení kurzorem prostřednictvím Touchpadu bylo poměrně komplikované a nespolehlivé, rozhodl jsem

se pro klikání vymezit zvláštní tlačítka (Pravé a levé tlačítko myši), jejichž callbacky doplňují chování Touchpadu.

Objekt Touchpad přijímá kliknutí a s pomocí ovladače dotykové plochy vyhodnocuje, jestli je prst (tužka, případně jiný předmět) přiložen na dotykovou plochu. V takovém případě tento pohyb periodicky vyhodnocuje a následně určuje vektor posunutí. Tento vektor je uložen do zprávy s parametrem $D = R$ a jako parametry x, y jsou do zprávy vloženy souřadnice směru posunutí.

Kliknutí tlačítka myši je vyhodnoceno callbackem příslušného objektu na displeji. Jako parametr D je použit $D = c$ pro levou myš a $D = p$ pro pravou myš. Parametry x, y jsou zde nulové, protože klikáme na pozici, jež se právě nachází pod kurzorem.

Aby nedocházelo k náhodnému stisknutí tlačítek okolo Touchpadu, je v callbacku každého tlačítka stanovena ochranná doba, po níž je tlačítko neaktivní po posledním dotyku Touchpadu.

Klávesnice a příslušné zprávy

Data klávesnice pro klávesnicový modul softwaru PC mají všechny jako pomocný parametr D stanoveno $D = P$. Do parametru x je vložen text pro odeslání. Text může být získán pomocí interní klávesnice zařízení nebo lze požadovat stisknutí zvláštních kláves (ALT+písmeno, CTRL+písmeno, F5 apod).

Zvláštní klávesy jsou získávány pomocí speciální klávesnice. Kódování těchto kláves je provedeno takovým způsobem, aby již nebylo nutné s přijatým textem nic dělat a Klávesnicový modul mohl data přímo zpracovat. Návod, jak zvláštní klávesy kódovat je k nalezení na stránkách společnosti Microsoft [24].

Potvrzovací zprávy a detekce chyb spojení

Přijetí každého balíku dat od počítače musí být potvrzeno tzv. potvrzovací zprávou. Tato zpráva nemá žádné vnitřní parametry (parametry uvnitř ní jsou na straně PC ignorovány). Pokud nejsou po dlouhý časový okamžik (cca. 15s) přijata žádná data, je odeslán požadavek na data, jenž je realizován touto zprávou. Pokud řídící software žádná data nemá, měl by odeslat zprávu NOP.

Přijímání dat je řešeno pomocí časovače. Předpokládá se totiž, že balík dat bude přicházet spojitě s mezerami mezi fragmenty menšími, než časový úsek. Problematická situace nastává, když jsou přijímána data pomocí šifrovaného spojení. Wifi modul totiž potřebuje delší čas na jejich dekódování a je tedy potřeba počítat s delší prodlevou. Tato je nastavena na 30ms při nešifrovaném spojení, 750ms při šifrovaném spojení. Je tedy třeba si uvědomit, že při požadavku na šifrované spojení bude ovládání počítače těžkopádnější.

Problém by bylo možné vyřešit kompresí dat, nicméně komprese dat je relativně náročná záležitost z hlediska výkonu i implementace, proto není v rámci této práce řešena.

4.1.6 Komunikace s PC s OS Linux

Komunikace se systémem Linux je z hlediska řídícího protokolu velmi jednoduchá. Spojení je totiž realizováno přímo s jádrem operačního systému prostřednictvím Telnet spojení. Ze zařízení se stává terminál prostřednictvím aplikace *Console*.

Jádro linuxového systému podporuje několik typů virtuálních terminálů (například xterm, vt100 apod.), volba typu terminálu definuje řídící sekvence (tzv. Escape sequence). Tento model komunikace vychází z historie linuxových systémů, kdy obrazovka a klávesnice byly pomocí společného kabelu s rozhraním nejčastěji RS232 připojeny k počítači. Složité grafické prvky obrazu nebylo možné pomocí takto pomalého rozhraní vykreslit v dostatečném čase, proto byly vytvořeny standardy, které umožňovaly operace jako přepínání fontů, pohyb s kurzorem apod., s jejichž pomocí bylo možné kreslit alespoň jednoduchou grafiku velice rychle. Příkladem takového rozhraní jsou programy jako *Powertop, Htop, Nano*.

Escape sequence

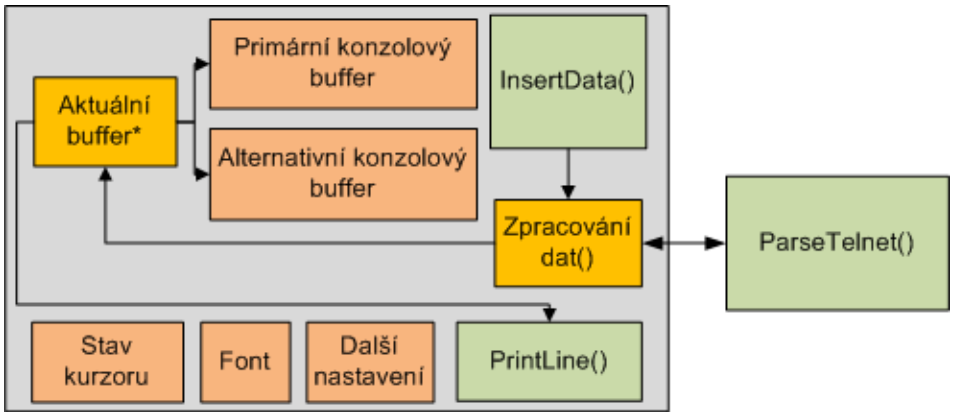
Řídících sekvencí je celá řada, jak již bylo řečeno, liší se podle typu terminálu. V základní konfiguraci (profil terminálu *network*) je grafika zcela vypnuta.

Formát escape sekvencí je následovný: Zpráva je uvozena znakem ESC (V ASCII tabulce číslo 27), poté následuje nepovinný definiční znak a parametry oddělené středníkem. Následuje ukončovací znak, jenž zároveň definuje sdělení příkazu. Vzhledem k množství možných příkazů není vhodné je zde všechny uvádět, lze je však nalézt například na adrese [14].

Zpracování příkazů probíhá uvnitř třídy *LBuffer*, jejíž instance je jádrem aplikace *Console*. Není vhodné zde uvádět kompletní definici této třídy, je vhodnější uvést její blokové schéma. To je na obrázku 4.14.

Jádrem třídy jsou dva buffery, primární a alternativní. Terminál má podle specifikace obsahovat dva obrazové buffery, přičemž jeden z nich je vždy aktivní a zároveň je zobrazován. Do tohoto bufferu jsou vkládány znaky, popřípdě jsou prováděny operace s aktivním bufferem podle aktuálního nastavení třídy (font, nastavení kurzoru apod).

Vložení dat do třídy je prováděno pomocí funkce *InsertData*, jež zpracovává textová data apod. Pokud se objeví znak Escape, předá metoda data funkci *ParseTelnet*. Ta provede zpracování těchto dat a následně i rekonfiguraci třídy nebo požadovanou operaci se znaky.

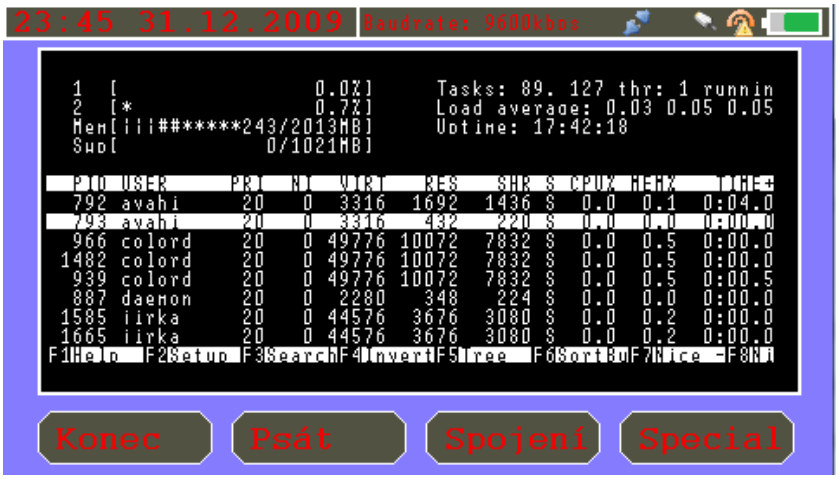


Obr. 4.14: Struktura třídy *LBuffer*

Lze si vyvodit, že nebylo třeba implementovat všechny sekvence, protože ne všechny jsou použitelné pro naše zařízení (Ovládání tiskárny apod.).

Vzhledem k tomu, že terminál nepodporuje všechny Escape sequence, nelze se spoléhat na jeho absolutně korektní chování ve všech situacích. Proto autor práce doporučuje v kritických situacích využívat programy bez grafického rozhraní, případně použít terminálový režim *network*, který je podporovaný absolutně.

Příklad aplikace spuštěné v terminálu zařízení je uveden na obrázku 4.15.



Obr. 4.15: Grafické rozhraní aplikace *Console*

Jelikož byl terminál VT100 monochromatický, není na něm možné zobrazovat barvy. V rámci experimentování byl ale učiněn pokus o implementaci profilu *xterm*. Jeho specifikace je ale velice rozsáhlá a proto byla implementace velice složitá. Souhrn specifikace lze nalézt například na [17].

4.1.7 Simulátor

Pro vytvoření simulátoru zařízení existovalo několik důvodů. Hlavním z nich byla nedostupnost hardwaru v počátcích vývoje zařízení, dalším důvodem byla snadnost ladění v prostředí Visual Studio oproti Cross Studio. Program se totiž postupem času rozrostl, a proto jeho vložení do paměti procesoru trvá mnohem déle, než jeho spuštění v okně systému Windows.

Simulátor samozřejmě nepodporuje všechny vlastnosti procesoru, není v něm například možné jednoduchým způsobem simulovat power-save mód, také nelze jednoduše simulovat hardwarový kurzor, registry hodin reálného času apod. Naopak je možné vytvoření spojení s Wifi modulem přes sériový port počítače.

Specifikům, která s sebou simulace hardwaru přináší, bylo třeba přizpůsobit komunikační knihovny (UART), některé části aplikací (Touchpad) a jiné.

Pro komunikaci s Linuxovým systémem byl použit emulátor nulového modemu *com0com*, ke stažení z [18]. Tímto programem byly vytvořeny dva páry sériových portů, jež se chovají jako propojené sériovým kabelem. Takto je možné jeden z portů nastavit jako výstupní pro Emulátor PC jako Virtual PC nebo VMware a druhý port použít v aplikaci simulátoru.

Podmínkou pro úspěšné použití simulátoru bylo, aby jádro softwaru bylo co nejshodnější pro reálný hardware i simuátor. Pokud je zajištěna vysoká míra shody, lze s úspěšností tvrdit, že změny provedené ve Visual Studio budou funkční i v Cross Works bez nutnosti jakýchkoli změn.

Souhrn rozdílů

Odlišností mezi reálným hardwarem a simulátorem jsou shrnuty v následujícím seznamu.

- Nutnost přidat algoritmus pro simulaci řadiče displeje.
- Definice jednotlivých odstínů barev jsou odlišné, protože je použita odlišná barevná hloubka.
- Jsou odlišné algoritmy pro práci s UARTem. Chybí možnost použít přerušení, čtení a zápis používají funkce send a recv.
- Hlavní program (grafické prostředí zařízení) je spouštěn jako vlákno aplikace simulátoru.
- Nejsou podporovány tahy na displeji, nelze proto plnohodnotně simulovat Touchpad.
- Časování událostí lze nativně provádět v intervalu min. 10ms, proto je upravena časovací funkce.

4.2 Software pro OS Linux

Spojení s Linuxem lze uskutečnit dvěma způsoby. První z nich je spojení přes sériovou linku. Toto spojení je z hlediska spolehlivosti výhodnější, neboť jediný problém, který může na kominikační lince nastat, je odpojení sériového kabelu. Výhodou této komunikace je i naprostá bezpečnost. Naproti tomu spojení přes Wifi je možné uskutečnit na téměř libovolnou vzdálenost. Nutnou podmínkou je samozřejmě existence Wifi sítě v okolí zařízení.

4.2.1 Spojení přes sériový kabel

Protože v linuxu je velmi jednoduché přesměrovat konzolová data na sériový port, není potřeba vytvářet žádný zvláštní software. Komunikaci lze navázat prostým zadáním následujícího příkazu do příkazového řádku:

```
telnet 127.0.0.1 > /dev/ttyS1 < /dev/ttyS1
```

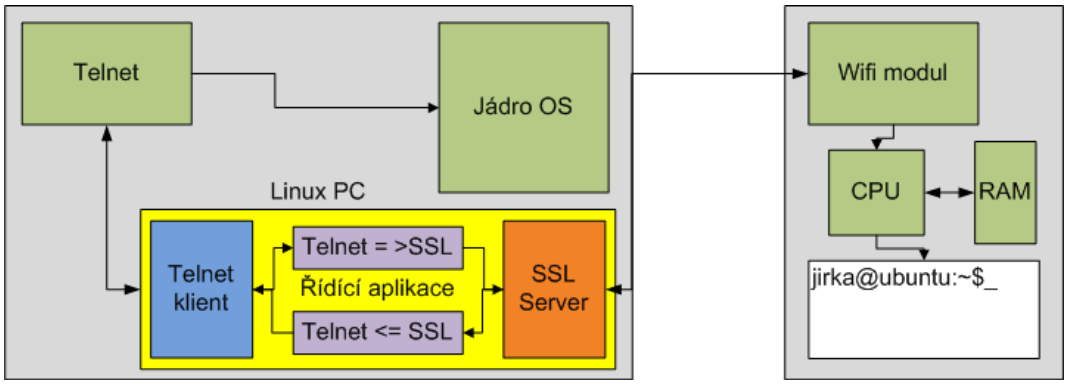
Dojde k otevření spojení s Telnetem lokálního stroje. Výstup tohoto spojení je přesměrován na sériový port *ttyS1*. Pokud by docházelo k dalšímu rozšíření projektu, bylo by vhodné otevřít toto spojení například při startu PC. Uživatel by měl možnost přihlásit se okamžitě po připojení zařízení.

4.2.2 Spojení přes TCP protokol

Ideálním způsobem by bylo navázání spojení přes SSH protokol přímo s aplikací standardně nainstalovanou na většině linuxových systémů. Tento postup je však poměrně složitý na implementaci (knihovny SSH by bylo potřeba naportovat do prostředí Cross Studio). Navíc by nebylo možné využít tak snadno SerialNet režim spojení Wifi. Výhodnější bylo vytvořit zabezpečené spojení pomocí integrovaného SSL šifrování v modulu. Struktura řídícího softwaru pak odpovídá obrázku 4.16. K realizaci softwaru je použita knihovna *OpenSSL* (ke stažení z [19]), jež obsahuje vše potřebné pro realizaci spojení pomocí SSL.

Jádrem softwaru jsou dvě struktury, *Telnet* a *SSLServer*, jež každá zajišťují obsluhu jednoho z rozhraní.

Komunikace pracuje na principu tunelu, stavový diagram spojení je uveden na obrázku 4.17. Na straně počítače je aktivní naslouchací port, jenž je zvolen defaultně na hodnotu 1025. Tuto hodnotu lze změnit argumentem předaným programu z příkazové řádky při spouštění. Naslouchání probíhá, dokud se neobjeví klient požadující připojení. Poté je provedena autentizace pomocí tzv. SSL handshake. V případě úspěchu při autentizaci je spojení na straně SSL považováno za navázané. Zároveň s touto událostí je vyvolán požadavek na spojení s localhost stanicí Telnet.



Obr. 4.16: Struktura spojení s OS linux

Po navázání Telnet spojení je provedeno vyjednání parametrů spojení (podrobnější informace lze nalézt na [20]) a zároveň je vyvolán požadavek na přihlášení uživatele.

Spojení může být ukončeno dvěma způsoby:

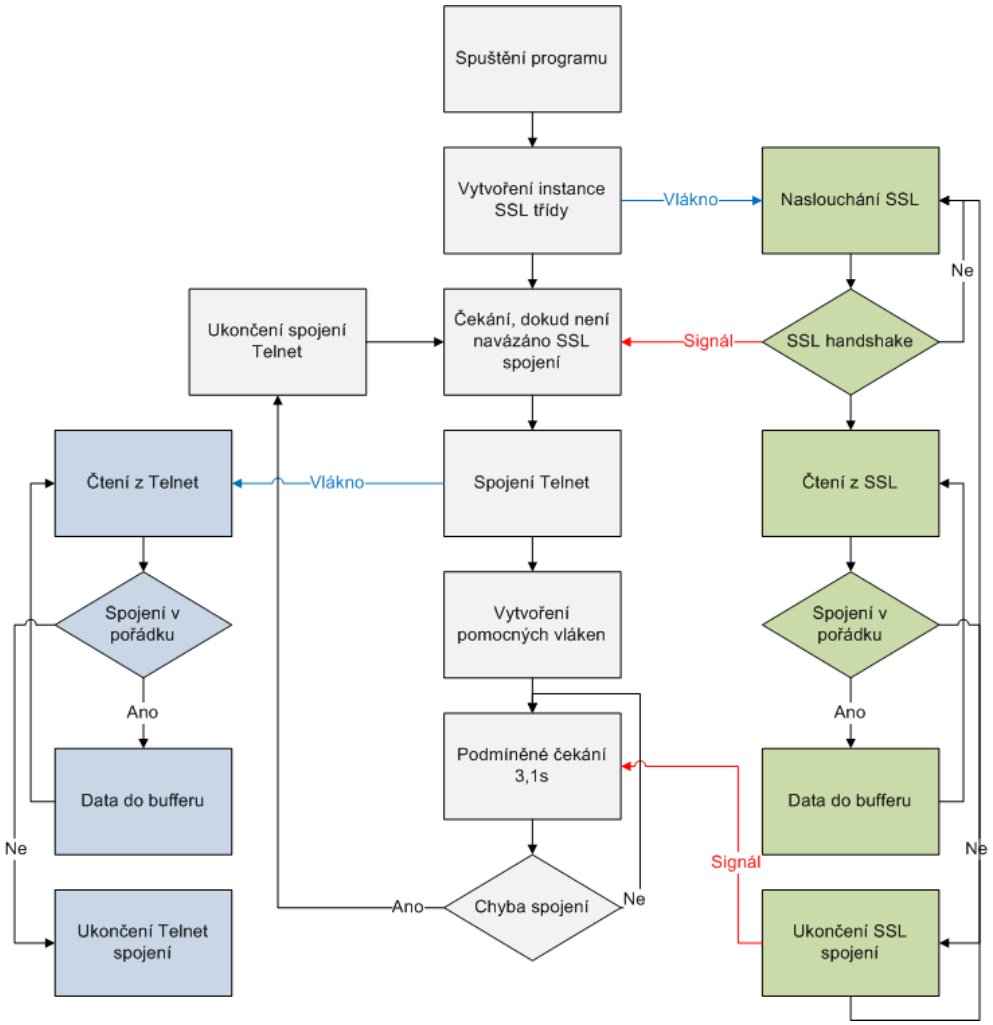
- Odhlášením uživatele
- Přerušením spojení

Detekce přerušení spojení je jednoduchá, funkce *recv* totiž vrací hodnotu menší než 0. Pokud dojde k přerušení Telnet spojení, je v instanci třídy obsluhující Telnet vystaven flag o přerušení spojení. Hlavní vlákno tento flag kontroluje a v případě problému je proveden reset tohoto spojení.

Pokud je přerušeno spojení ze strany SSL, je se spojením nakládáno jako se ztraceným. Dojde k ukončení Telnet komunikace a SSL spojení přejde do stavu naslouchání.

Pro efektivní fungování řízení spojení bylo nutné aplikaci rozdělit do několika vláken. Komunikace těchto vláken s hlavním vláknem je zajištěna pomocí podmíněných proměnných, samozřejmostí je užití mutexů pro vyloučení mnohonásobného přístupu k paměťovým strukturám.

Aby bylo možné data přesouvat mezi spojeními efektivně, je navržena struktura jednoduchého mostu, jež je patrná z diagramu 4.18. Hlavní vlákno po navázání spojení vytvoří dvě vlákna, jež jsou prakticky identická. Ta čekají na signál od struktur Telnet a SSLServer, přičemž každé 3 sekundy je čekání přerušeno a je kontrolován stav spojení. V případě, že spojení nejsou funkční, jsou obě vlákna ukončena.



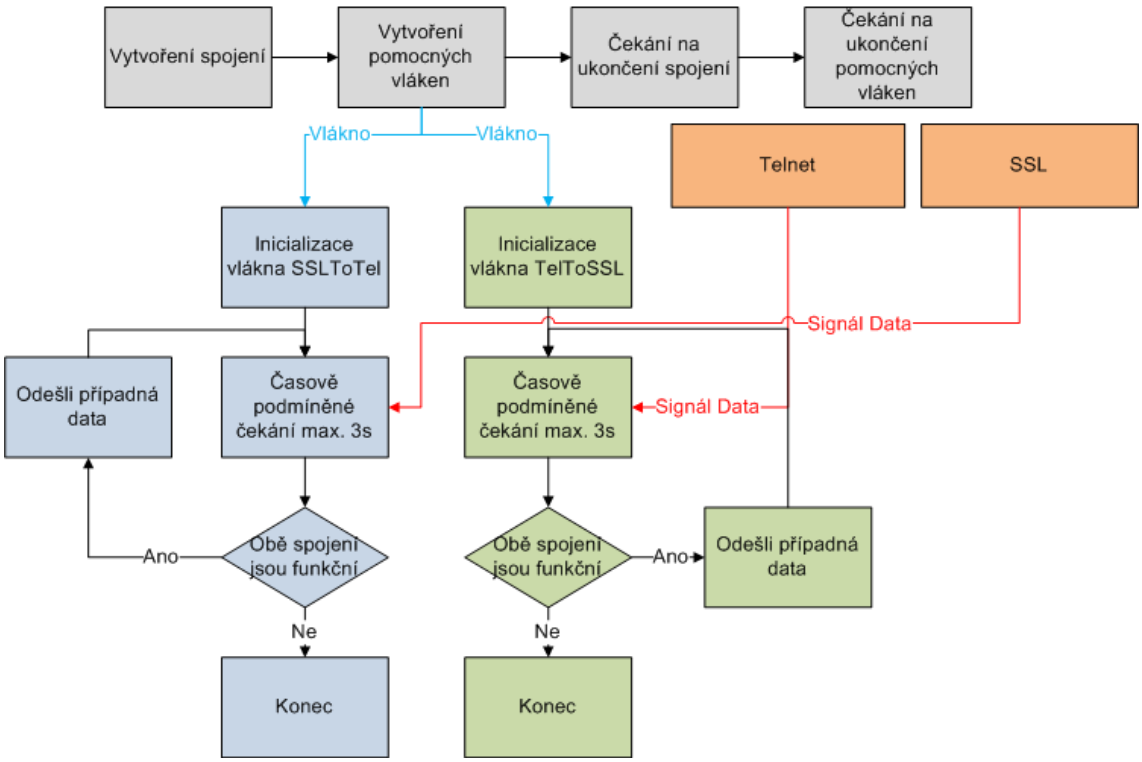
Obr. 4.17: Struktura řízení spojení s OS linux

4.3 Software pro OS Windows

Zajištění komunikace s tímto operačním systémem bylo o něco složitější. Bylo nutné totiž vytvořit kompletně celý komunikační protokol, vyřešit problematiku snímání obrazu, jeho přenos a zároveň zajistit dekódování dat přijímaných ze zařízení.

Proto byl software rozdělen opět na několik logicky funkčních celků, jež bylo možné odladit samostatně. Software je vytvořen v jazyce C#, přičemž jako aplikační rozhraní je použit tzv. .NET Framework ve verzi 4.0. Výhodou Frameworku je jeho komplexnost, podpora síťové komunikace včetně šifrování a grafická propracovanost jeho jednotlivých prvků.

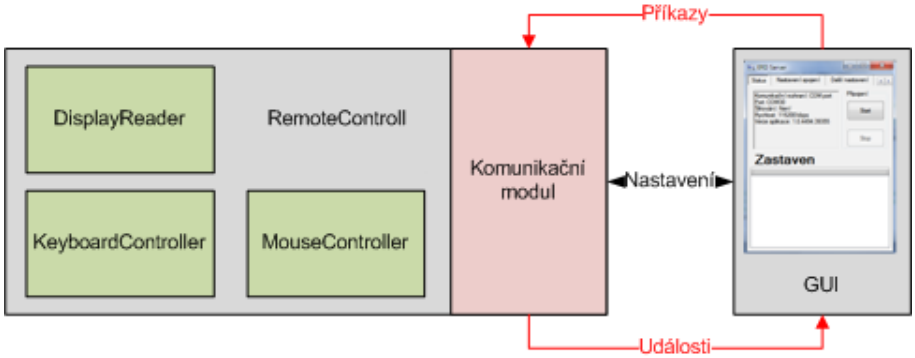
Nevýhodou Frameworku je nemožnost odlaďovat problémy na nižších vrstvách než dovolují základní třídy (není tedy například možné ladit problémy vzniklé při handshaku a podobně).



Obr. 4.18: Struktura mostu pro předávání dat

Aplikace je navržena tak, aby režim připojení bylo možné zvolit pouze z klientského zařízení. Filozofií při tvorbě aplikace byla snaha, aby uživatel pouze při startu klienta vybral komunikační rozhraní a jeho parametry a zapnul naslouchání. Pomocí argumentů příkazové řádky lze program přinutit, aby se spouštěl minimalizovaný, případně aby okamžitě začal naslouchat. Toho lze s výhodou využít, pokud chceme, aby se spouštěl při startu systému.

Software se zkládá z bloků dle schématu 4.19.



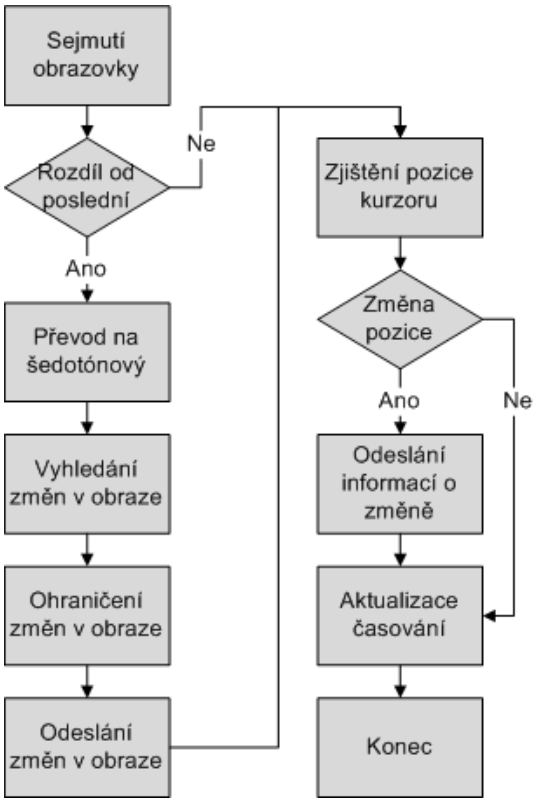
Obr. 4.19: Struktura řídícího software pro OS Windows

4.3.1 Modul pro čtení obrazovky

Čtení obrazovky zajišťuje instance třídy *DisplayReader*. Třída obsahuje vše potřebné pro sejmutí aktuálního stavu obrazovky a odeslání dat do výstupního datového proudu.

Algoritmus zpracování dat z obrazovky je postaven na softwarovém balíku *OpenCV* (Ke stažení z [21]). Ten je vytvořen v jazyce C++, pro jeho použitelnost v C# je tedy nutné použít wrapper, který umožní C# přístup k těmto funkcím. Jako wrapper byl použit balík *Emgu CV* (Ke stažení z [22]). Aby výsledný program správně pracoval, musí být všechny knihovny umístěny ve složce s výsledným programem.

Algoritmus zpracování dat z obrazovky nejlépe vystihuje diagram 4.20.



Obr. 4.20: Algoritmus získání změn v obraze

V každém průchodu je nejprve vytvořen snímek obrazovky v původním rozlišení, který je okamžitě převeden na rozlišení, které je přijímáno klientem[8]. Ten je porovnán s posledním získaným snímkem, přičemž je vytvořen rozdílový snímek. Tento snímek je převeden na šedotónový a je vypočtena jeho hodnota. Pokud je tato hodnota nenulová, došlo ke změně. Je tedy nutné nalézt souřadnice všech změn. Aby

[8]Doporučené rozlišení je stanoveno na 478x250, hodnotu je možno zvolit v nastavení.

byly sloučeny malé změny do větších, je provedena dilatace tohoto obrazu. Po nalezení souřadnic a zjištění rozměrů nejmenších obdélníků, do nichž lze změny uzavřít, jsou z originálního obrazu vyjmuty originální body a jejich hodnoty jsou přepočteny na formát použitelný v zařízení dle obrázku 4.5.

Data jsou vložena do zásobníku pro odesílání, ze kterého jsou odesílána postupně komunikačním modulem. Pokud je detekován snímek, který obsahuje data pro celou plochu, je celý zásobník vymazán a snímek je vložen na první místo v zásobníku.

Formát paketů není příliš komplikovaný, aby mohla být data co nejrychleji zpracována v klientském zařízení. Jsou definovány dva formáty datových paketů, které se vztahují k obrazu a jeden formát paketu, který se vztahuje ke kurzoru myši.

Úvodní znak	**Datová pole**	**Popis**
F	X, Y, Šířka, Výška, Obrazová data	Snímek nebo jeho část
O	X, Y	Offset obrazů
M	X, Y	Pozice kurzoru
NOP	nejsou	Prázdná zpráva

Tab. 4.2: Typy zpráv odesílaných z PC s OS Windows

Aby nedocházelo k zahlcování zařízení obrazovými daty (ta jsou často velmi objemná), vyžaduje řídící software potvrzení přijetí paketu vzdálenou stranou. Dokud není přijata zpráva s potvrzením ve specifickém formátu, není nic dalšího odesíláno. Zprávy vysílané zařízením jsou definovány v předchozím textu.

Nevýhodou zvoleného způsobu komunikace je, že není zabezpečena proti chybám na úrovni aplikační vrstvy (například chyba při komunikaci procesoru zařízení s modulem apod). Tento problém však v důsledku nedostatku času nebyl vyřešen. Prakticky totiž postačuje ochrana dat na úrovni TCP protokolu.

Během vývoje tohoto modulu byl řešen problém, jak efektivně sejmout plochu systému Windows. Situaci lze řešit několika postupy:

- Pomocí DirectX SDK
- Pomocí funkcí .NET 4.0
- Přímou komunikací s grafickým ovladačem videokarty

Byly otestovány pouze první dva postupy, třetí se ihned v počátku projevil jako velice složitý, proto od něj bylo upuštěno.

DirectX SDK je balík softwaru, jenž velmi úzce spolupracuje přímo s grafickým rozhraním systému, proto existoval předpoklad, že bude s jeho pomocí jednoduché získat aktuální obraz monitoru. Existuje několik postupů, bohužel jsem narazil na zásadní problém. Sejmutí plochy proběhlo velice rychle (řádově několik milisekund),

nicméně velmi dlouho trval převod mezi .NET rozhraním a DirectX funkcemi (přibližně 100ms).

Mnohem rychlejší a efektivnější byl nakonec druhý postup, kdy byla plocha sejmuta pomocí balíku .NET. Zpracování celého obrazu tak trvá přibližně 55ms v závislosti na aktuálním vytížení systému.

4.3.2 Modul pro ovládání myši a klávesnice

Pro ovládání kurzoru myši je použita funkce WinAPI *mouse_event*, jež byla pomocí *DLLImport* nawrappována do C#.

Wrapping vypadá takto:

```
[DllImport("user32.dll", parametry volani)]
private static extern void mouse_event(UInt32 dwFlags, uint dx, uint dy, ...);
```

Pomocí parametru *dwFlags* je stanovena akce, kterou má funkce s kurzorem provést. Mezi základní akce patří kliknutí, kliknutí na pozici, kliknutí pravým tlačítkem, apod.

Ve třídě *MouseControl* je funkce zapouzdřena, je tak zajištěno její jednodušší volání.

Problémem při řízení kurzoru bylo stanovení převodní charakteristiky. Z klienta vzdálené plochy jsou totiž přijímány pouze vektory posunutí. Jejich přímé použití je však nepohodlné, kurzor reaguje velice pomalu, ačkoli přesně. Pro uživatele však toto není nejvhodnější, protože kurzor je často přesunován na velkou vzdálenost. Vhodnější tedy bylo vytvoření převodní charakteristiky, která způsobí, že při malých vektorech posunutí bude kurzor posunut velmi přesně, při velkých posunech bude posunut nepřesně ale o to rychleji.

Převodní funkce je navržena dle následujících vztahů.

$$_{\Delta}X = \frac{sign(S_X).{S_X}^2}{250} \tag{4.3}$$

$$_{\Delta}Y = \frac{sign(S_Y).{S_Y}^2}{250} \tag{4.4}$$

Kde $_{\Delta}X$ a $_{\Delta}Y$ jsou výsledná posunutí kurzoru a S_X a S_Y jsou souřadnice vektoru posunutí přijaté komunikačním modulem.

Aby bylo chování kurzoru na zařízení co nejsvižnější, mají kurzorová data prioritu a jsou vkládána do odesílacího zásobníku vždy na první místo. Vzhledem k tomu, že je odesílána absolutní pozice kurzoru, je možné ostatní kurzorové pakety ze zásobníku před přidáním aktuálnějších odstranit.

Modul *KeyboardControl* plní pouze kosmetickou funkci. Jeho jedinou uživatelskou metodou je metoda *Press*, která zprostředkuje volání .NET metody *SendWait*.

4.3.3 Modul pro řízení komunikace

Modul je tvořen třídou *RemoteControl*. Instance třídy je vytvořena ihned při startu aplikace, nicméně aby ji bylo možné použít ke komunikaci, musí být nakonfigurována. Konfigurace a spuštění naslouchání je provedeno metodou *Start*. Metoda existuje ve třech přetíženích:

- `public bool Start(string port, int speed)`
- `public bool Start(int port)`
- `public bool Start(int port, X509Certificate2 cert)`

Pokud je volána první z metod, předpokládá se, že uživatel požaduje komunikaci přes sériový port. Je tedy otevřen port a zahájeno čekání na data při zadané komunikační rychlosti. Nepředpokládá se žádná parita a je očekáván pouze jeden stop bit.

Druhá z metod zahájí naslouchání na zadaném TCP portu. Pokud se objeví čekající klient, je okamžitě obsloužen.

Třetí z metod aktivuje šifrování. Pokud tedy na naslouchaném portu vznikne spojení, je proveden SSL handshake. V případě úspěchu již spojení pracuje stejně jako v předchozích případech.

Třída je implementována takovým způsobem, aby po aktivaci naslouchání a navázání spojení na nejnižší úrovni (tzn. TCP protokol nebo COM port) postačovalo pouze převzít objekt typu *Stream* ze spojení a ten předat zbytku aplikace. Ostatní třídy již s tímto objektem pracují jednotně bez rozlišování komunikačního rozhraní.

Vnitřní stavový automat třídy je uveden na obrázku 4.21.

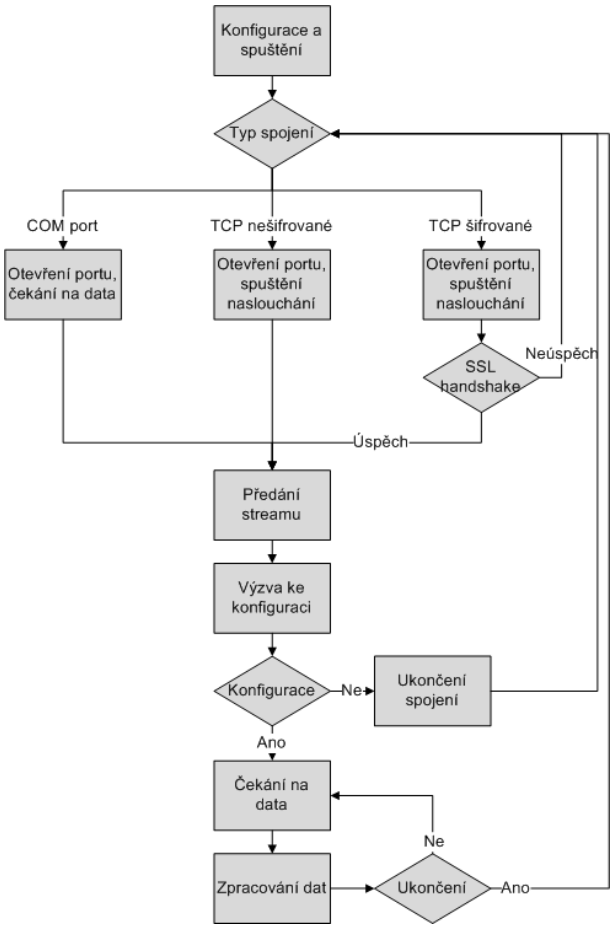
Pokud existuje klient, který se chce k programu připojit, je nejprve požádán o konfigurační informaci. Očekávaná odpověď může být dvojí:

- ERD,CT=Display
- ERD,CT=Peripheral

Pokud je přijata konfigurace *Display*, je spuštěno čtení obrazovky ve třídě *DisplayReader* a je zahájeno odesílání dat s obrazem. Pokud je režim *Peripheral*, není třeba odesílat žádná data, pouze se čeká na povely od uživatele pro klávesnici a kurzor.

Po přijetí jedné z konfiguračních vět je spojení považováno za funkční a je zahájena aktivní komunikace s klientem. Spojení lze ukončit dvěma způsoby. První z nich je přerušení komunikace a pád spojení. V takovém případě je aplikace uvedena do stavu naslouchání. Korektní způsob, jak ukončit spojení je přijetí ukončovací věty *!STOP!*. Potom je spojení uzavřeno (v případě TCP protokolu) a aplikace se vrací do stavu naslouchání.

Klientské zařízení by mělo být schopné detekovat stav spojení. Tento stav je detekován jednoduše z přijímaných dat. Klientské zařízení tedy vyžaduje, aby mu



Obr. 4.21: Stavový diagram třídy *RemoteControl*

byla alespoň jednou za 5 sekund poslána data, jinak o ně žádá zprávou stejnou, jako je akceptační zpráva. Pokud je tedy přijata konfigurace do periferního režimu, je spuštěn časovač, který odešle každých 5 sekund prázdný paket ve formátu *NOP*.

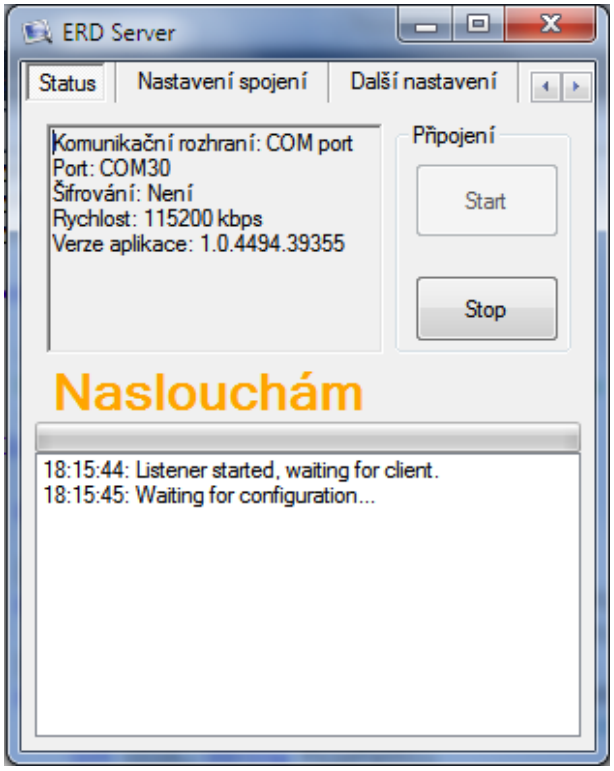
4.3.4 Grafické rozhraní

Pro pohodlné ovládání programu je v systému Windows naprostým standardem grafické rozhraní. Pro modul *RemoteControl* bylo tedy vytvořeno grafické rozhraní. Je navrženo tak, aby bylo možné okno aplikace minimalizovat na lištu. Ovládací program po zadání parametru *-m* minimalizuje okno, *-s* zase provede spuštění naslouchání.

Aby nebylo nutné po každém startu programu zadávat konfiguraci, je implementováno ukládání dat do INI konfiguračního souboru. Pro jednoduchý přístup k těmto datům je použita třída *INIFile* převzatá z [23]. Do tohoto souboru jsou uloženy veškeré informace o nastavení programu v okamžiku jeho vypnutí.

Pro komunikaci s nižšími (pracovními) třídami je použito systému delegátů, kteří v okamžiku vzniku události volají asociované metody jiných tříd.

Screenshot aplikace je na obrázku 4.22.



Obr. 4.22: Screenshot aplikace

Jelikož aplikace umožňuje přístup do systému z vnější sítě, je vhodné, aby i s certifikáty byla umístěna ve složce, do které má přístup pouze uživatel oprávněný používat ji.

5 ZÁVĚR

Diplomová práce se zabývala návrhem koncepce zařízení, jež mělo svému uživateli umožnit ovládání počítače PC. V rámci této práce bylo navrženo obvodové schéma, jež bylo následně realizováno a oživeno.

Pro vytvořený hardware byl vytvořen firmware a následně i řídící software pro počítač s operačním systémem Windows a Linux. Komunikace mezi klientským přístrojem a počítačem může být šifrována pomocí algoritmu SSLv3, nehrozí tak odposlouchávání komunikace ze strany třetích osob. Podpora šifrování je zajištěna pomocí balíku OpenSSL v Linuxu a .NET 4.0 ve Windows. Zpracování obrazu je vyřešena pomocí balíku OpenCV, aby bylo možné jej jednoduše použít v jazyce C#, je použit také wrapper Emgu.

Při použití výrobku se systémem Windows je zařízení schopno pracovat jako náhražka klávesnice, touchpadu, a obrazovky. Výkon zařízení v režimu obrazovky je poměrně nízký, což je způsobeno pomalou odezvou komunikačního rozhraní. V rámci pokračování tohoto projektu by tedy mohlo být přistoupeno ke komprimování obrazových dat a tím ke zrychlení odezvy zařízení.

Se systémem Linux komunikuje zařízení prostřednictvím protokolu Telnet, dovoluje tedy přístup ke konzolovému rozhraní PC a odtud provádění nejdůležitějších změn. Za tímto účelem jsou implementovány prvky pro grafické zobrazování programů jako editor Nano, případně monitorovací software Htop atp. Jako implementovaný terminálový profil je zvolen VT100, jenž dovoluje základní formátování dat ve dvou barvách.

Pro zařízení byl zvolen mikrokontrolér společnosti NXP, typ LPC2478, jehož výhodou je velké množství periferií, mezi něž patří zejména řadič pro barevný TFT displej a řadič paměti SDRAM. Tato periferie byla klíčová při výběru procesoru, protože zvolený displej jakožto typický zástupce rodiny dotykových displejů s rozlišením 480x272 pixelů vyžaduje specifický způsob řízení. Pro zařízení byla vybrána dynamická paměť RAM s kapacitou 16MB, zařízení má tak dostatek paměti pro provádění operací s obrazem.

Jako prvek pro síťovou komunikaci byl vybrán Wifi modul s integrovaným TCP/IP stackem a podporou šifrování SSL, nebylo tedy nutné řešit volbu komunikačního software. Toto řešení s sebou však přineslo několik problémů, protože do softwaru modulu nelze zasahovat (lze jej pouze konfigurovat pomocí AT+i protokolu), nelze odladit chyby, které se v jeho softwaru vyskytují. Jako druhé komunikační rozhraní nad rámec zadání práce byl vybrán sériový port. Alternativně deska podporuje i USB rozhraní, to však není v rámci práce oživeno.

LITERATURA

[1] *Datasheet:* LPC2478. N/A : NXP B.V., 2011. 93 s. Dostupné z WWW: <http://www.nxp.com/documents/data_sheet/LPC2478.pdf>.

[2] *User manual:* LPC24XX User manual. N/A : NXP B.V., 2008. 70 s. Dostupné z WWW: <http://www.nxp.com/documents/user_manual/UM10237.pdf>.

[3] *Datasheet:* MT48LC4M32B2. N/A : Micron Technology, Inc., 2001. 67 s. Dostupné z WWW: <http://download.micron.com/pdf/datasheets/dram/sdram/128MbSDRAMx32.pdf>.

[4] *Datasheet:* PT0434827T-A401. N/A : PALM TECHNOLOGY, NA. 22 s. Dostupné z WWW: <http://www.tme.eu/dok/06_optoelektronika/pt0434827t-a401.pdf>.

[5] *Datasheet:* TPS61040. N/A : TEXAS INSTRUMENTS, Inc., 2002. 26 s. Dostupné z WWW: <http://www.ti.com/lit/ds/slvs413f/slvs413f.pdf>.

[6] *Datasheet:* TPS63060. N/A : TEXAS INSTRUMENTS, Inc., 2012. 31 s. Dostupné z WWW: <http://www.ti.com/lit/ds/symlink/tps63060.pdf>.

[7] *Datasheet:* TSC2003. N/A : TEXAS INSTRUMENTS, Inc., 2007. 28 s. Dostupné z WWW: <http://www.ti.com/lit/ds/sbas162g/sbas162g.pdf>.

[8] *Datasheet:* MAX3232. N/A : TEXAS INSTRUMENTS, Inc., 2004. 20 s. Dostupné z WWW: <http://www.ti.com/lit/ds/symlink/max3232.pdf>.

[9] *Datasheet:* BQ2057. N/A : TEXAS INSTRUMENTS, Inc., 2002. 30 s. Dostupné z WWW: <http://www.ti.com/lit/ds/symlink/bq2057.pdf>.

[10] *Datasheet:* NanoSocket. N/A : Connect One Ltd., 2009. 23 s. Dostupné z WWW: <http://www.spezial.cz/pdf/Nano_Socket_iWiFi_DS.pdf>.

[11] *Datasheet* HX8257-A01. N/A : Himax Technologies, Inc, 2008. 70 s. Dostupné z WWW: <http://www.microtipsusa.com/pdfs/driver_controller_spec/HX8257-A01.pdf>.

[12] MACHÁČEK, J. *Terminály pro ovládání zařízení v rodinném domě*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 75 s. Vedoucí bakalářské práce Ing. Tomáš Macho, Ph.D..

[13] *Atmel AT45DB161D* : 16-megabit 2.5V or 2.7V DataFlash. NA : Atmel, 2010. 51 s. Dostupné z WWW: <http://www.atmel.com/dyn/resources/prod_documents/doc3500.pdf>.

[14] WILLIAMS, Paul. *VT100.net* [online]. 2006, 2006-06-25 [cit. 2012-05-14]. Dostupné z: <http://www.vt100.net/>.

[15] USUNOV, Tsvetan. *OLIMEX* [online]. 2012, 2012 [cit. 2012-05-14]. Dostupné z: <http://olimex.com/>.

[16] MB86291A 'Scarlet'. *FUJITSU*. Fujitsu [online]. 2012 [cit. 2012-05-14]. Dostupné z: <http://www.fujitsu.com/emea/services/microelectronics/gdc/gdcdevices/mb8629 scarlet.html>.

[17] MOY, Edward. Rtfm / Xterm / Escape Sequences. STEVENS, Michael. *ETLA* [online]. 1999 [cit. 2012-05-14]. Dostupné z: <http://rtfm.etla.org/xterm/ctlseq.html>.

[18] FROLOV, Vyacheslav. Null-modem emulator. GEEKNET, Inc. *Sourceforge* [online]. 2005, 2012-02-01 [cit. 2012-05-14]. Dostupné z: <http://sourceforge.net/projects/com0com/>.

[19] Binary Distributions. ENGELSCHALL, Ralf S. *OpenSSL: Cryptography and SSL/TLS toolkit* [online]. 2009, 10-May-2012 [cit. 2012-05-14]. Dostupné z: <http://www.openssl.org/related/binaries.html>.

[20] Telnet. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2012, 2012-May-8 [cit. 2012-05-14]. Dostupné z: <http://en.wikipedia.org/wiki/Telnet>.

[21] BORNET, Olivier. OpenCV. GEEKNET, Inc. *SourceForge* [online]. 2012 [cit. 2012-05-14]. Dostupné z: <http://sourceforge.net/projects/opencv/>.

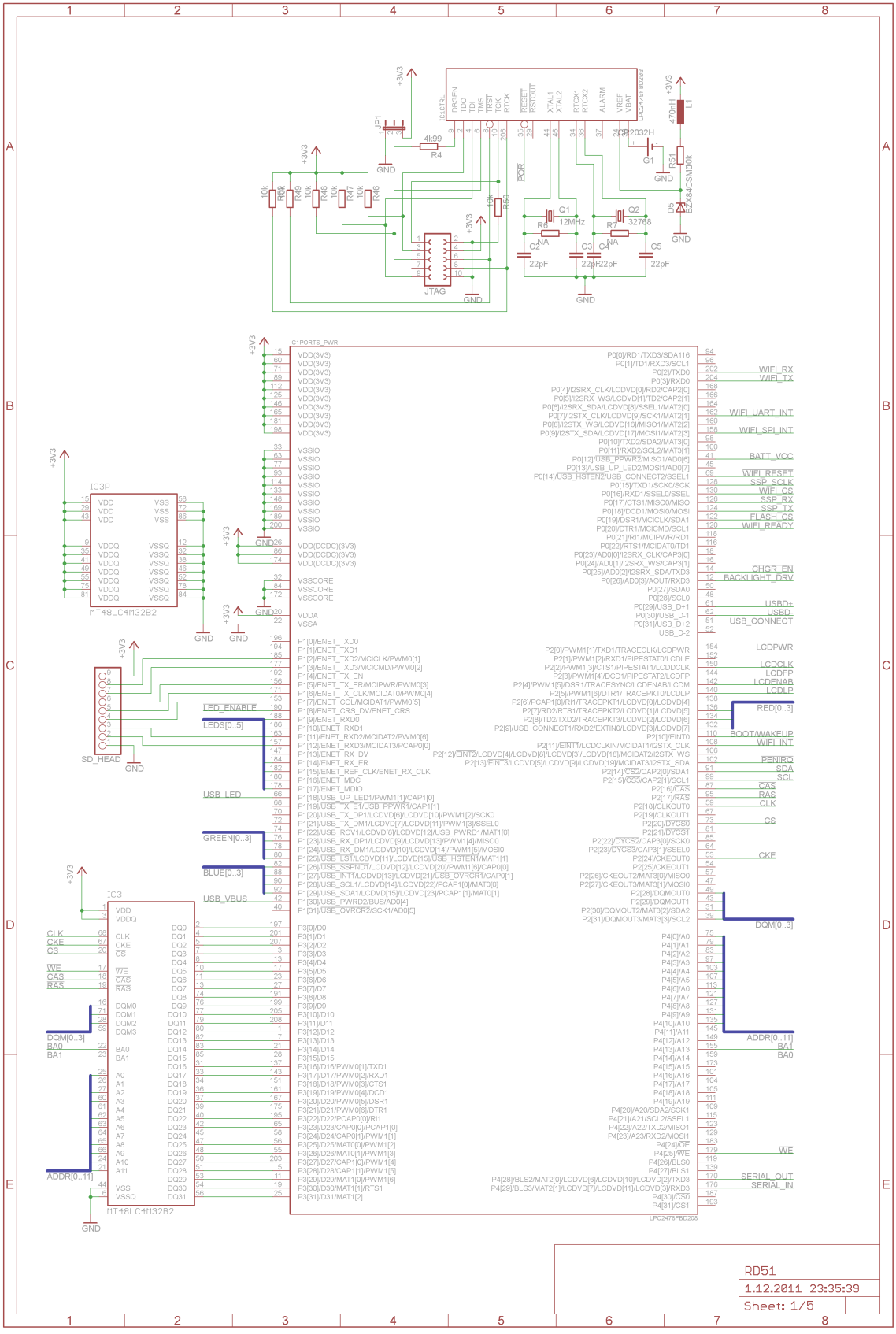
[22] CANMING. Emgu CV. GEEKNET, Inc. *SourceForge* [online]. 2012 [cit. 2012-05-14]. Dostupné z: <http://sourceforge.net/projects/emgucv/>.

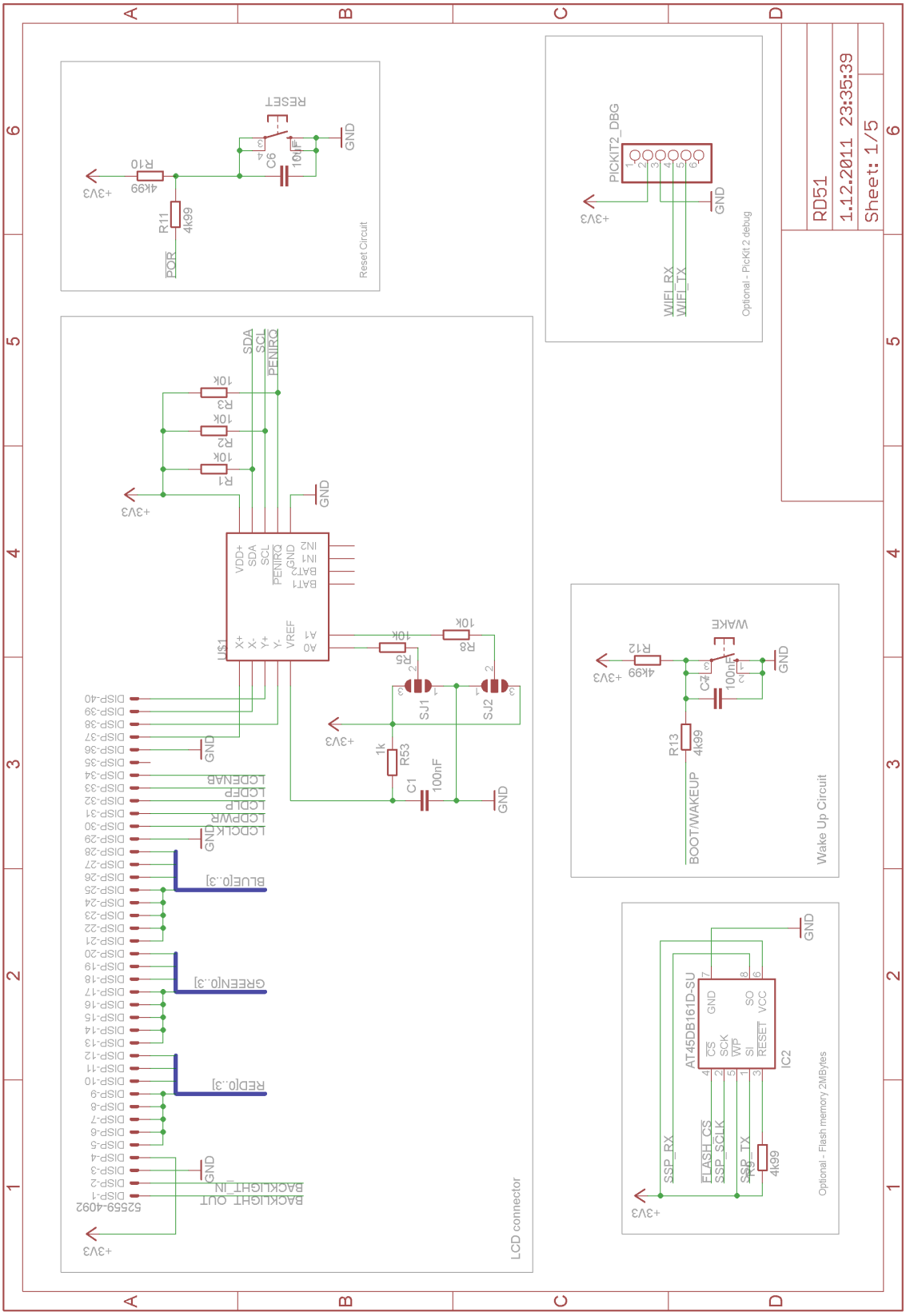
[23] BLAZINIX. An INI file handling class using C#. In: . *The Code Project: Your Development Resource* [online]. Canada, 2002 [cit. 2012-05-14]. Dostupné z: <http://www.codeproject.com/Articles/1966/An-INI-file-handling-class-using-C>.

[24] SendKeys Class. MICROSOFT. *MSDN* [online]. 2012 [cit. 2012-05-15]. Dostupné z: <http://msdn.microsoft.com/en-us/library/system.windows.forms.sendkeys.aspx>.

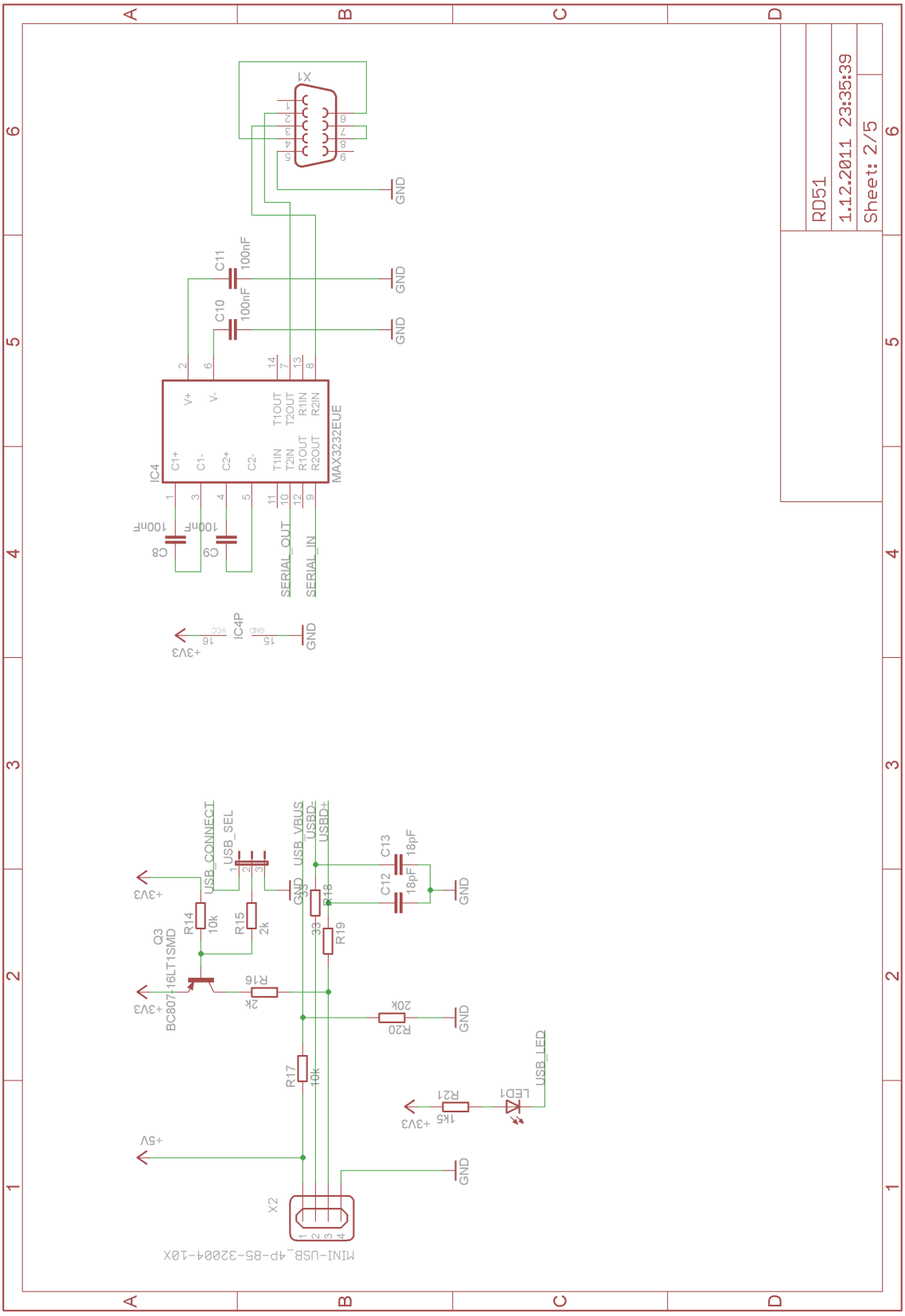
SEZNAM PŘÍLOH

A CELKOVÉ SCHÉMA ZAŘÍZENÍ

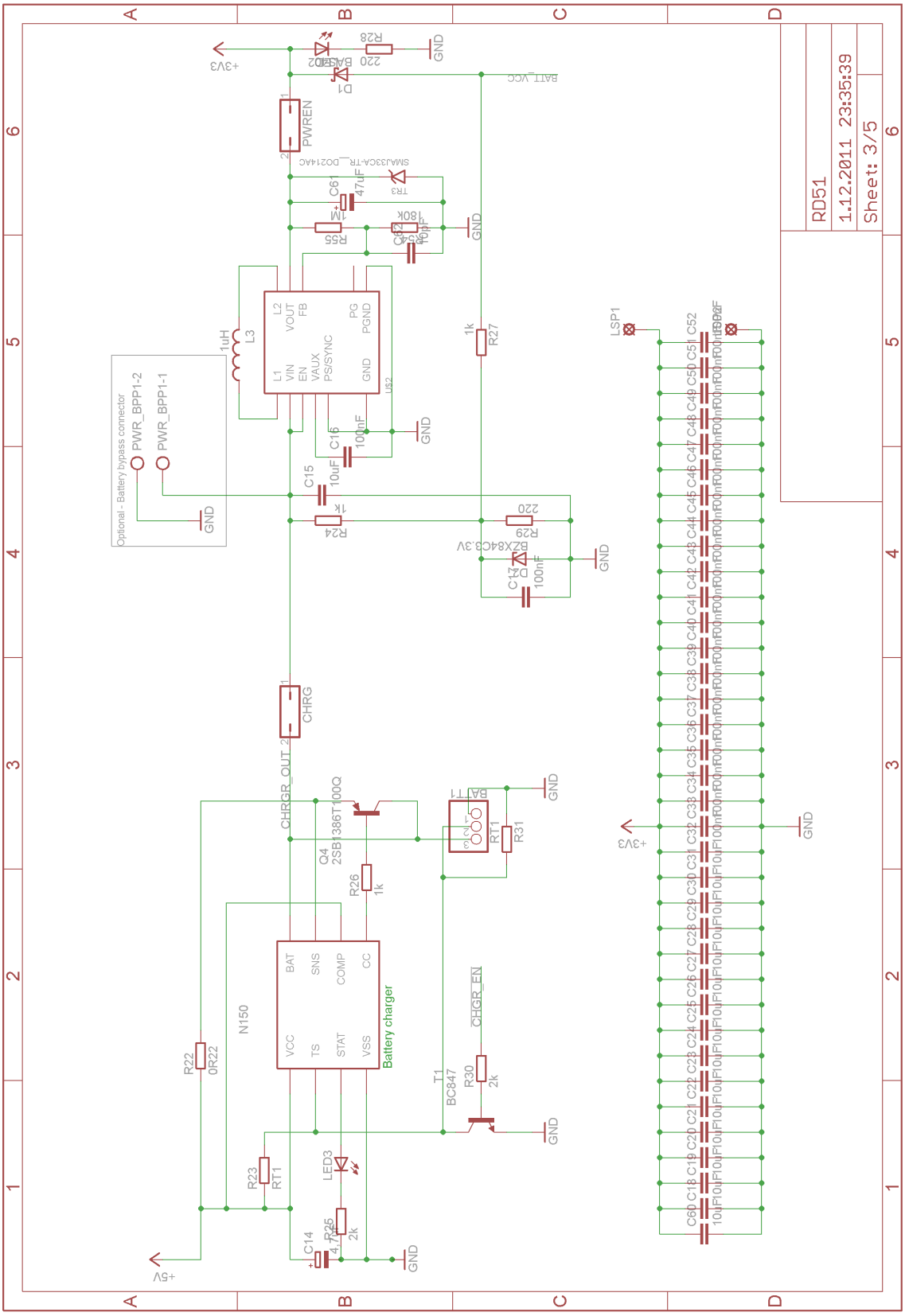




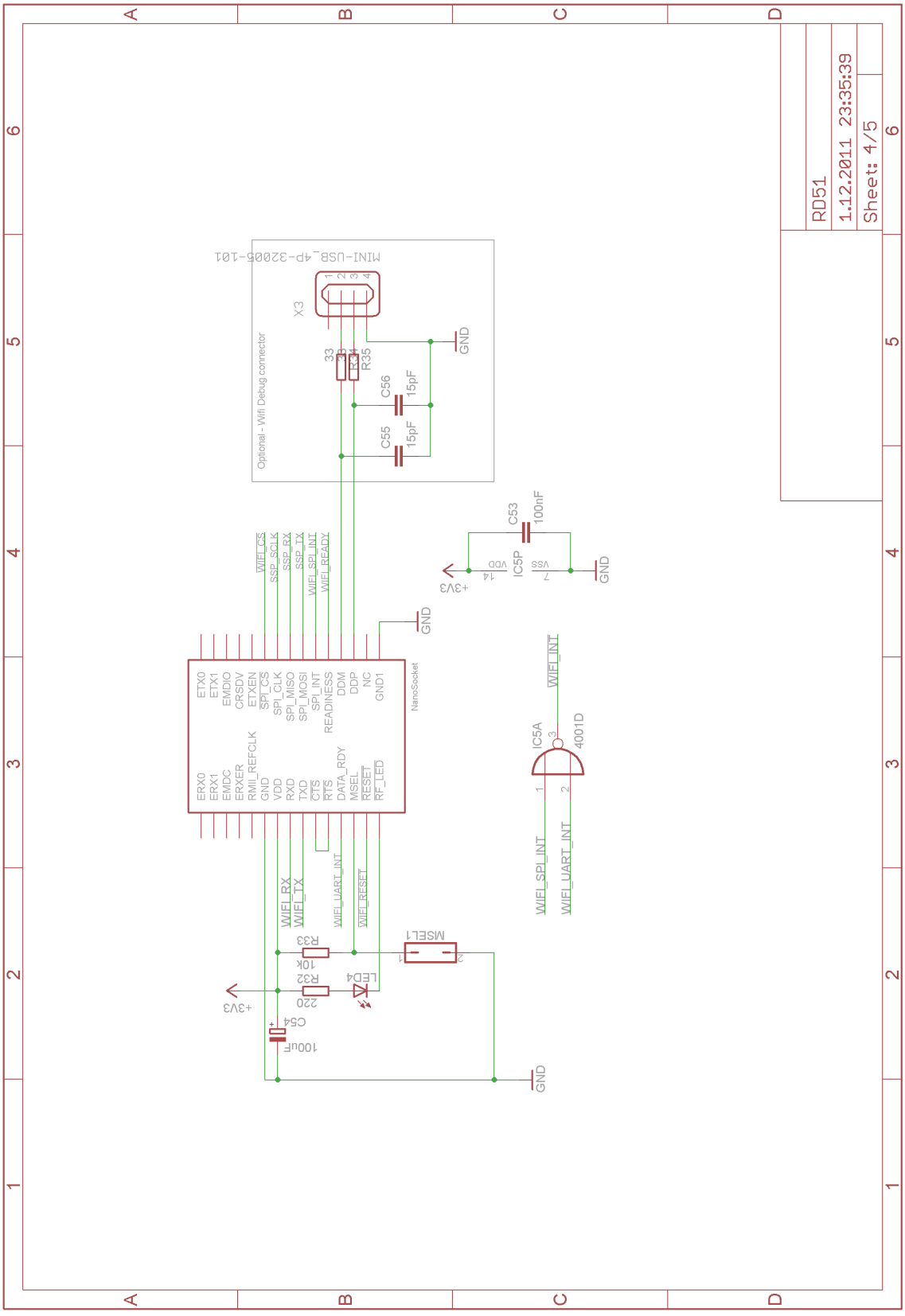
Obr. A.1: Pomocné obvody mikrokontroléru a konektor displeje



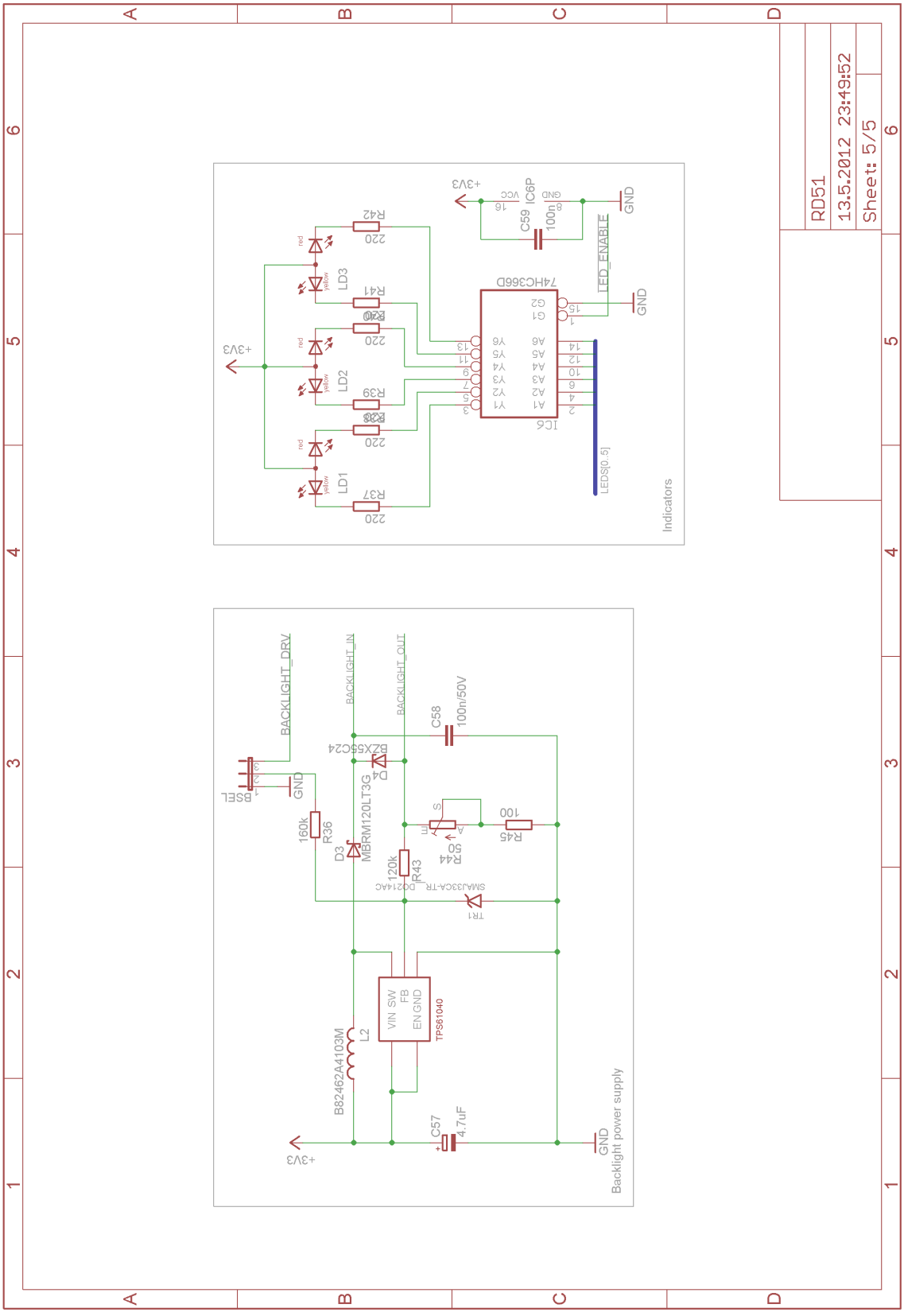
Obr. A.2: Konektor USB a RS232



Obr. A.3: Napájecí zdroj



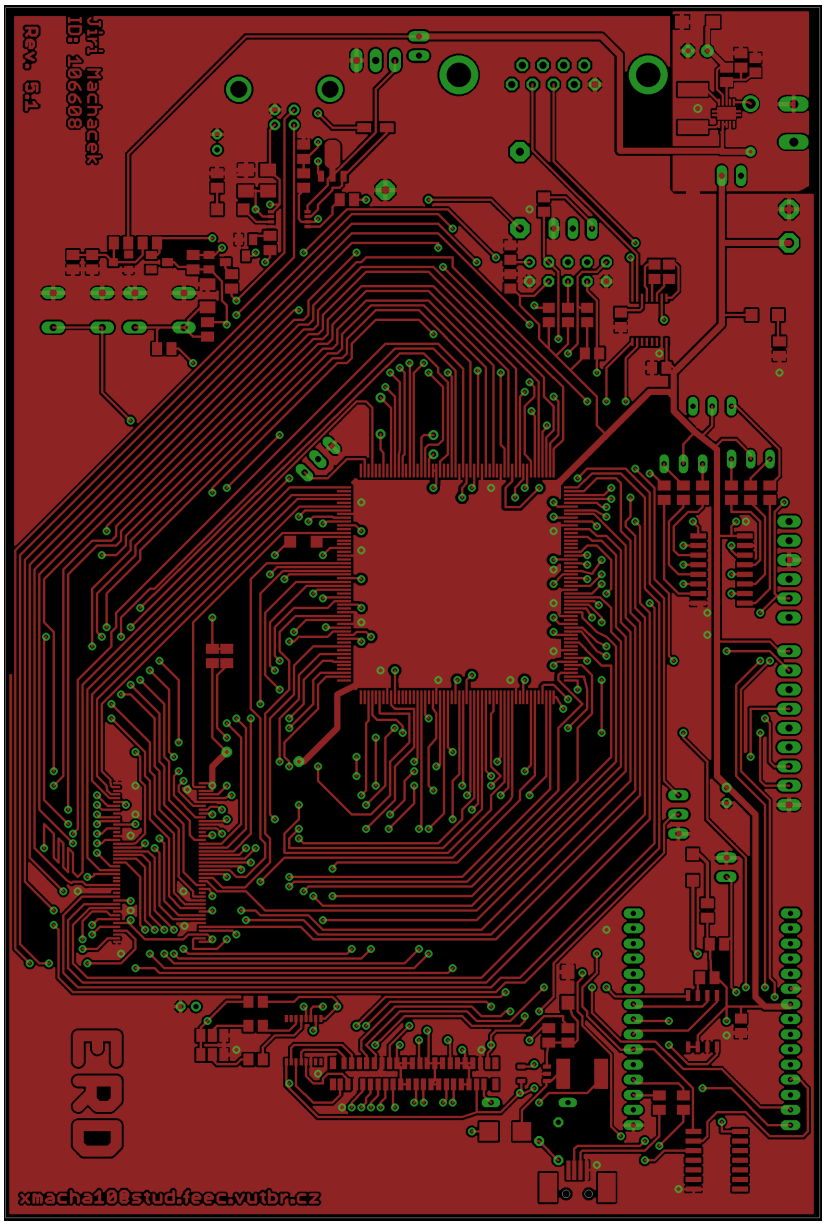
Obr. A.4: Wifi modul



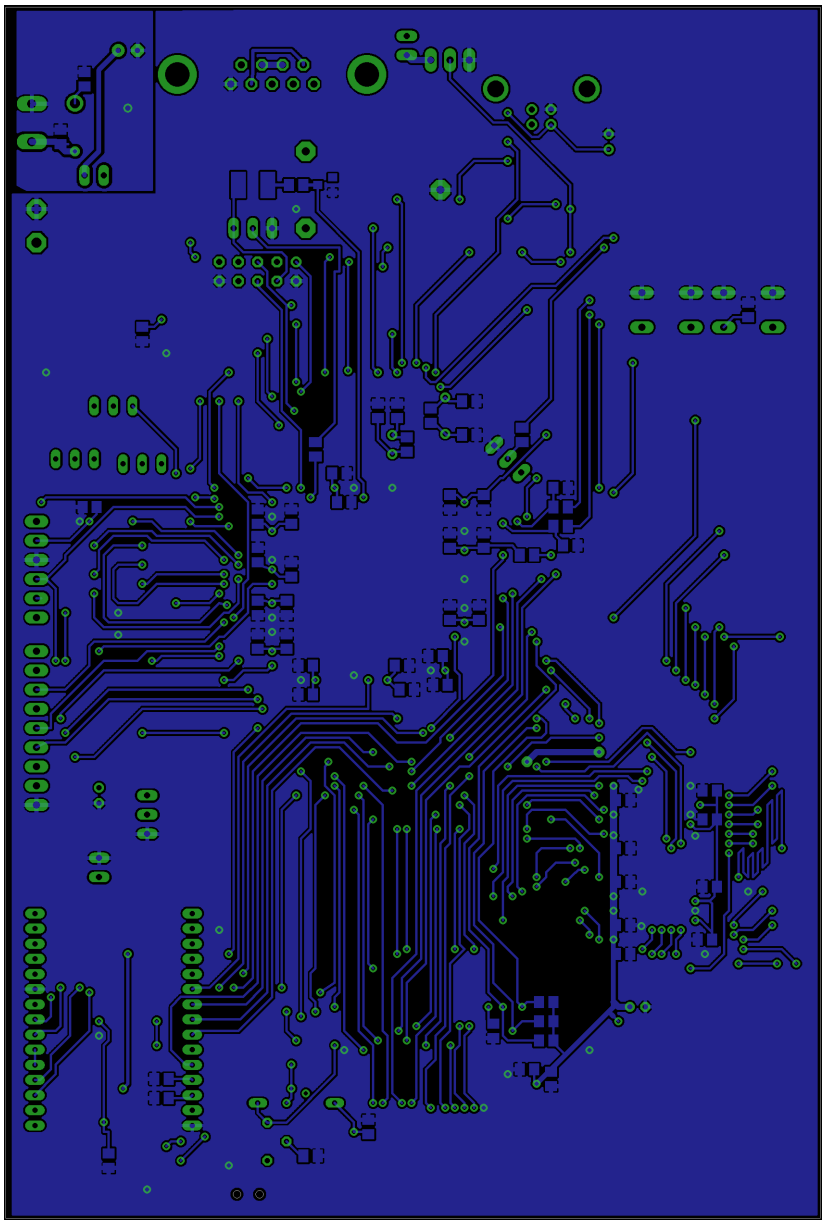
Obr. A.5: Zdroj podsvětlení a indikátory

B REALIZOVANÁ DESKA ZAŘÍZENÍ

B.1 Deska plošného spoje

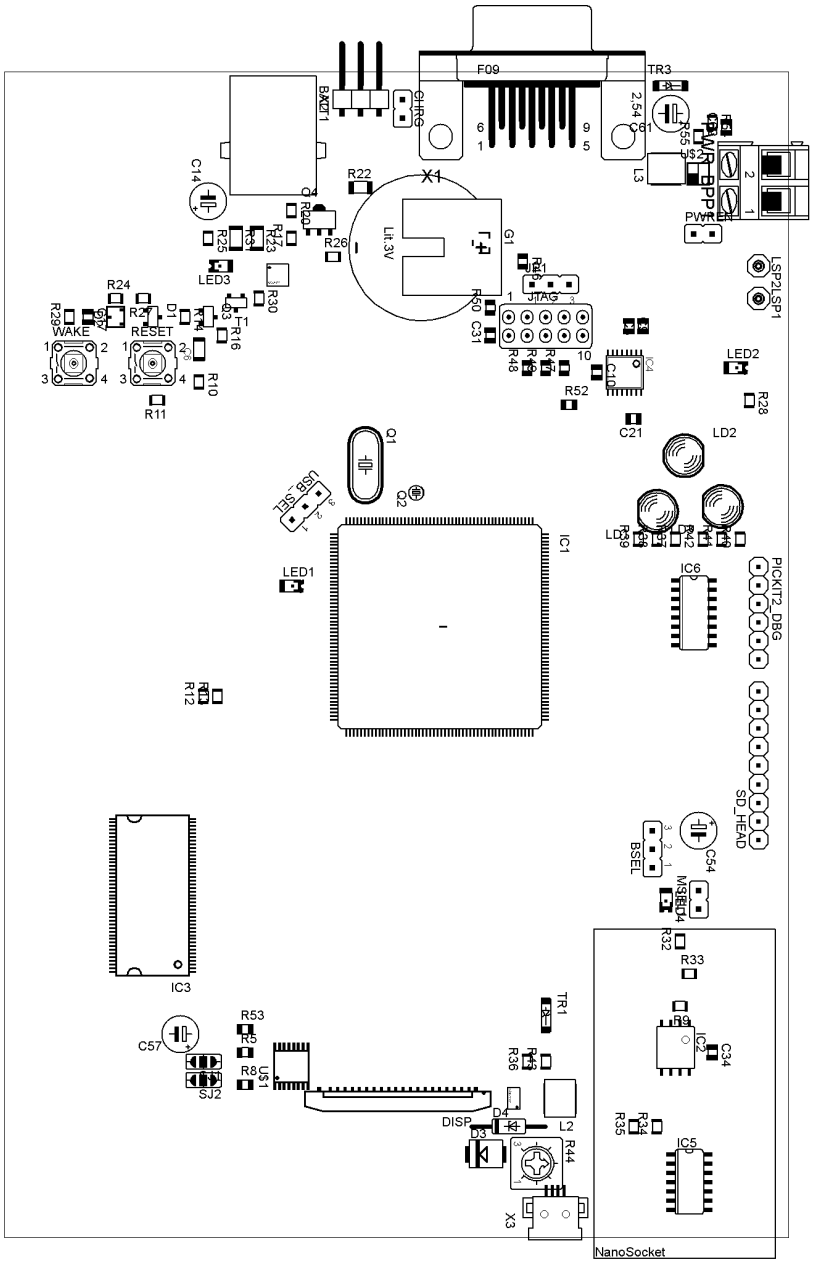


Obr. B.1: Deska plošného spoje - vrchní strana

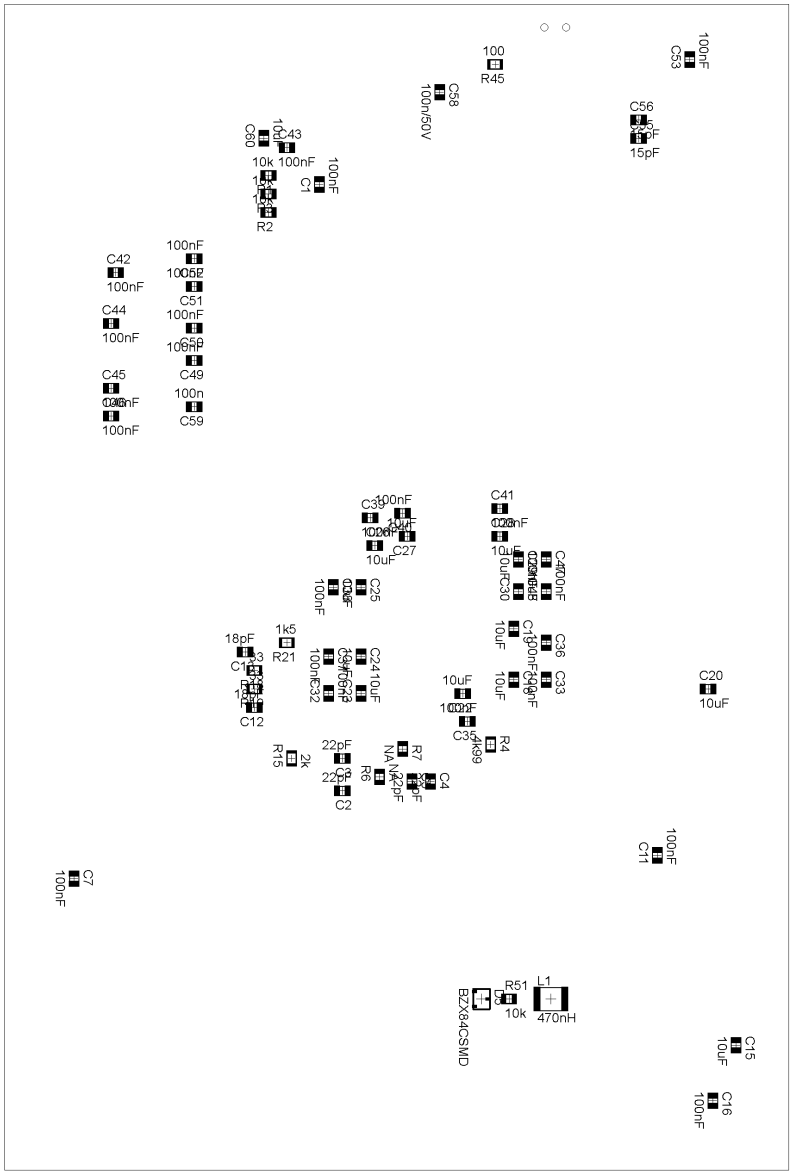


Obr. B.2: Deska plošného spoje - spodní strana

B.2 Rozmístění součástek



Obr. B.3: Rozmístění součástek na desce - vrchní strana



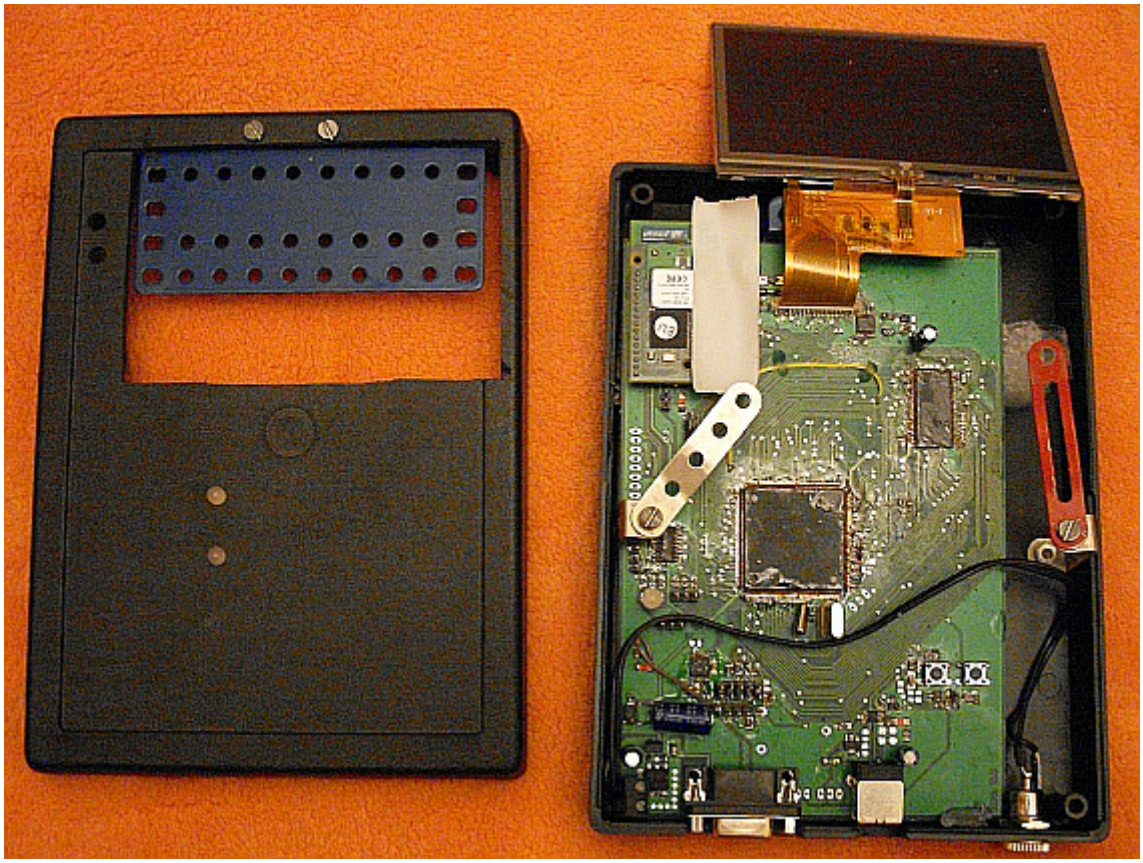
Obr. B.4: Rozmístění součástek na desce - spodní strana

B.3 Seznam součástek

Název součástky	Hondota	Počet	Typ	Dodavatel
C1, C7-C11, C16, C17, C32-C53, C58, C59	100nF	32	CL21B104KBCNNNC	TME
C6, C15, C18-C31	10uF	16	C0805C106K8PAC	TME
C2-C5	22pF	4	KER SMD 22P NPO 0805	GES
C12, C13	18pF	2	KER SMD 18P NPO 0805	GES
C55, C56	15pF	2	KER SMD 15P NPO 0805	GES
C14, C57	4,7uF	2	SD1J475M05011BB	TME
C54	100uF	1	SD1E107M05011PC	TME
C61	47uF	1	SD1E476M05011BB	TME
C62	10pF	1	KER SMD 10P NPO 0805	GES
Pinová lišta		30		GME
R28, R29, R32, R37-R42	220	9	CR0805 220R 5%	TME
R1-R3, R5, R8, R14, R17, R33, R46-R52	10k	15	CR0805 10K0 5%	GES
R4, R9-R13	4k99	6	SMD0805-4K99-1%	TME
R18, R19, R34, R35	33	4	CR0805 33R0 5%	GES
R24, R26, R27, R53	1k	4	CR0805 1K00 5%	GES
R15, R16, R25, R30	2k	4	HP05-2K75%	TME
R20	20k	1	CR0805 20k00 5%	GES
R21	1k5	1	CR0805 1k50 5%	GES
R22	0R22	1	SMD1206-0R22	TME
R36	160k	1	CR0805 160k00 5%	GES
R43	120k	1	CR0805 120k00 5%	GES
R44	50	1	PT10LV-50R	TME
R45	100	1	RC0805JR-07100R	TME
R55	1M	1	SMD0805-1M	TME
R54	180k	1	SMD0805-180k	TME
Q1	12MHz	1	12.00M-HC49	TME
Q2	32768Hz	1	32.768K-20PPM	TME
Q4		1	2SB1386T100Q	FARNELL
Q3		1	BC807-16	TME
LED1-LED4	Červená	4	KPT-3216ID	TME
LD1-LD3	Červená/Zelená	3	L-59EGW-CA	TME
L2	10uH	1	B82462A4103M	FARNELL
L1	470nH	1	DL1812-0.47	TME
L3	1uH	1	DL4N-1	TME
D1		1	BAS40-07	TME
D2, D5	3V3	2	BZX84C3V3	TME
D3		1	MBRM120LT3G	FARNELL
D4	24V	1	0.5W-24V	TME
DISP		1	52559-4092	FARNELL
JTAG		1	MLW10G	GME
G1	3V3	1	BAT-CR2032-H/MT	TME
IC1		1	LPC2478FBD208,551	FARNELL
IC2		1	AT45DB161D-SU	TME
IC3		1	MT48LC4M32B2P-6	TME
IC4		1	MAX3232CWE+	TME
IC5		1	4001-SMD	TME
IC6		1	74HC366	GME
WAKE, RESET		2	DTS644K	TME
TR1, TR3	3V3	2	SMAJ33CA-LF	TME
T1		1	BC847C	TME
U$1		1	TSC2003	TI
U$2		1	TPS63060	TI
U$3		1	BQ2057	TI
U$4		1	NanoSocket0	ConnectOne
U$5		1	TPS61040	TI
X1		1	DHP8-09M	TME
X2		1	292304-1	TME
X3		1	ESB33100000Z	TME
RWB_BPP1		1	ARK300/2	GME

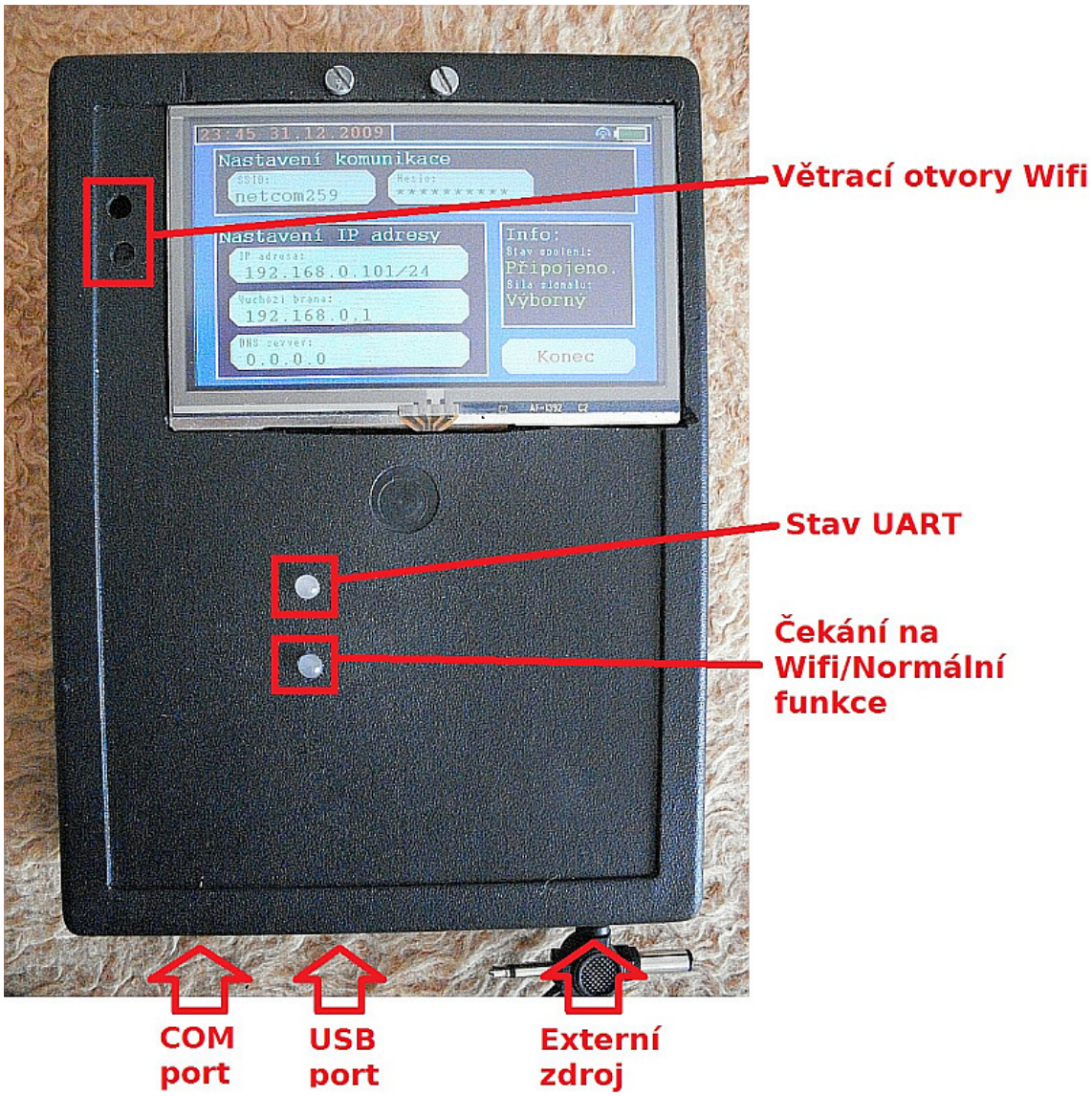
C VNITŘNÍ USPOŘÁDÁNÍ ZAŘÍZENÍ

Při montáži hotové desky výrobku s displejem do krabičky bylo potřeba vyřešit problém ukotvení LCD displeje na správném místě, vic C.1. Displej totiž nemá žádné mechanické úchyty, proto jako jediná možnost montáže připadá v úvahu nalepení displeje na modrou destičku přišroubovanou ve víku krabičky. Aby vznikla pevná opora displeje a zároveň bylo možné krabičku otevřít pro účely ladění softwaru, bylo potřeba vytvořit podpěry ve spodní části krabičky.



Obr. C.1: Vnitřní uspořádání výrobku

D VNĚJŠÍ VZHLED VÝROBKU



Obr. D.1: Vnější uspořádání výrobku