Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta
Katedra informatiky

USB zařízení pro převod sériové USB komunikace na paralelní komunikaci pro řízení externí logiky, funkční zařízení ovládané přes USB rozhraní včetně software

Bakalářská práce

Jonáš Simota

Školitel: Ing. Břetislav Bakala

České Budějovice 2012

Bibliografické údaje
Simota Jonáš, 2012: USB zařízení pro převod sériové USB komunikace na paralelní komunikaci pro řízení externí logiky, funkční zařízení ovládané přes USB rozhraní včetně software.
[USB device to convert serial USB communication to parallel communication for external control logic, functional devices controlled via USB interface including software. Bc. Thesis, in Czech.] – 50 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 14. prosince 2012 Jonáš Simota

Anotace

Tato bakalářská práce se zabývá návrhem a konstrukcí funkčního USB zařízení, návrhem a výrobou plošného spoje, programováním software a vytvořením uživatelského rozhraní pro ovládání zařízení přes PC. Hlavní funkcí celého zařízení je ovládání modulu RGB LED diod v různých režimech.

Abstract

The bachelor work deals with the design and construction of the functional USB device, design and manufacture of PCB, programming software and creating a user interface for controlling the device via the PC. The main function of the device is controlling RGB LEDs in module`s different modes.

Poděkování

Rád bych poděkoval panu Ing. Břetislavu Bakalovi za odborné rady ohledně realizace hardwaru, za jeho čas věnovaný konzultacím a za konečnou kontrolu celé práce.

Dále bych rád poděkoval panu RNDr. Jaroslavu Ichovi za rady ohledně psaní dokumentace a za rady a připomínky při diplomovém semináři.

Obsah

1 Úvod

Tato bakalářská práce se zabývá návrhem a konstrukcí funkčního USB zařízení, které je založené na vývojové desce FEZ Panda II a dále se skládá ze spínacího a napájecího modulu a modulu RGB LED diod. Funkčnost celého zařízení bude předvedena ovládáním RGB LED diod v různých režimech. Hlavní funkcí celého zařízení je zjištění barvy, která převažuje na obrazovce uživatele v reálném čase, předání informací o této barvě přes USB rozhraní vývojové desce a následné rozsvícení příslušné barvy RGB diodami. Tento efekt může být využíván při sledování filmů, hraní počítačových her nebo i při pouhém prohlížení fotografií. Mezi další možnosti patří např. manuální rozsvícení zadané barvy. Celý software je navržen v Microsoft Visual Studio 2010, uživatelská aplikace pro PC je naprogramována v jazyce C#, aplikace pro vývojovou desku pak v jazyce micro C#.

2 Cíle

Cílem tohoto projektu je navrhnout, zprovoznit a následně otestovat USB zařízení. Toto zařízení by mělo správně plnit svou hlavní funkci, ale zároveň by mělo být univerzální a mělo by zkušenějším uživatelům poskytnout možnost úprav a vylepšení, nebo dokonce i možnost využít zařízení pro vlastní potřebu k jiné funkci než bylo původně navrženo.

3 Přehled současného stavu

3.1 Rozbor problematiky

Zařízení tohoto typu se nedá zakoupit a to je hlavní důvod, proč jsem se rozhodl navrhnout vlastní. Vzorem pro toto zařízení jsou televizory značky Philips s technologií Ambilight. Televizory jsou na zadní straně osazeny RGB LED pásky, které osvětlují prostor za televizorem a toto osvětlení barevně navazuje na sledovaný obraz. Tyto televizory jsou řádově o několik tisíc dražší, než televizory bez této technologie. Moje řešení by mělo podobný efekt zpřístupnit i uživatelům, kteří sledují filmy, nebo hrají hry na PC s klasickým monitorem a to za dostupnou cenu.

3.2 Požadavky na hardware PC

Základním předpokladem k používání zařízení je stolní PC nebo notebook s USB portem. Celá aplikace není náročná na paměť RAM, ale na výpočty CPU. Kvůli tomu patří mezi minimální systémové požadavky alespoň dvou jádrový procesor. Zařízení samozřejmě lze zprovoznit i na pomalejším hardwaru, ale může docházet k prodloužení doby odezvy zařízení nebo ke zpomalení ostatních běžících aplikací (např. sekání sledovaného filmu).

3.3 Požadavky na vývojovou desku

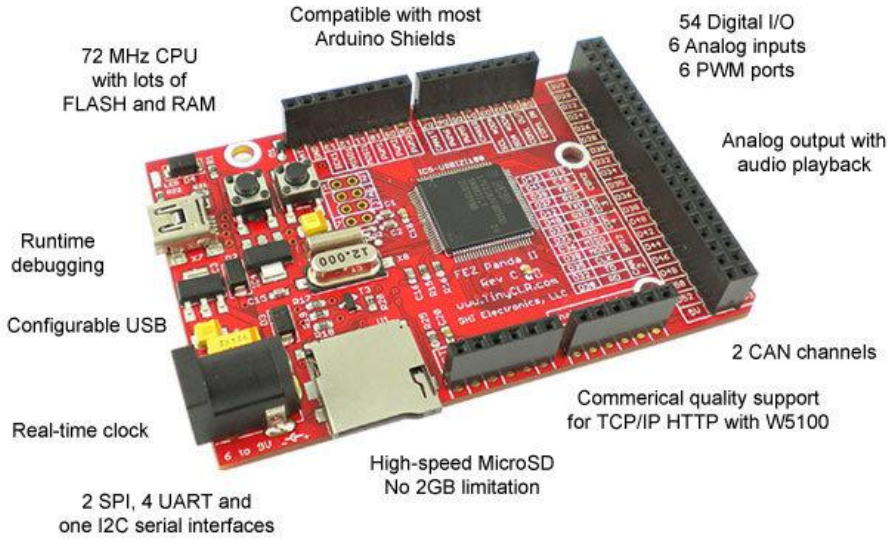
Požadavky na hardware, který bude řídit LED modul byly následující:

- minimálně 3 digitální výstupy
- minimální frekvence ovládání výstupů 1kHz
- USB rozhraní
- debugging v reálném čase
- možnost odesílat do zařízení vstupní data za chodu
- programovací jazyk C, C++ nebo C#

4 Přehled součástek a technologií

4.1 Vývojová deska FEZ Panda II

FEZ Panda II je mikropočítačová vývojová deska založená na .NET Micro Framework a je programovatelná přímo z vývojového prostředí Microsoft Visual Studio.



Obrázek č. 1 – vývojová deska FEZ Panda II

Hlavní vlastnosti:

72MHz. 32-bit ARM7 procesor

USB připojení pro run-time debugging

USB Debugging a Virtuální COM (CDC) mohou pracovat najednou

54x Digital I/O portů

Multi-Threading[1]

---

[1] GHI ELECTRONICS, GHI Electronics. *Snail Instruments* [online]. 2012 [cit. 2012-12-08]. Dostupné z: http://shop.snailinstruments.com/index.php?main_page=product_info&products_id=929&zenid=20b89ef8bc8661e7a6896ee8796b6e97

4.2 Spínací modul

Spínací modul slouží ke spínání RGB LED diod, které mají velký výkon (7,2W) a musí být napájeny z vlastního zdroje. Jedná se o jednoduchý spínací obvod se třemi tranzistory BD139 (pro každou barvu jeden) v zapojení se společným emitorem. Obvod sepneme přivedením signálu na bázi, na kterou je přes rezistor připojen digitální výstup z modulu FEZ.

4.3 RGB LED modul

Hlavní vlastnosti:

Uhel vyzařování světla: 120°
Spotřeba na metr [W/m]: 14,4 W
Počet LED na metr: 60
Barva: RGB
Napájecí napětí: 12V DC [2]

Obrázek č. 2 – RGB LED pásek

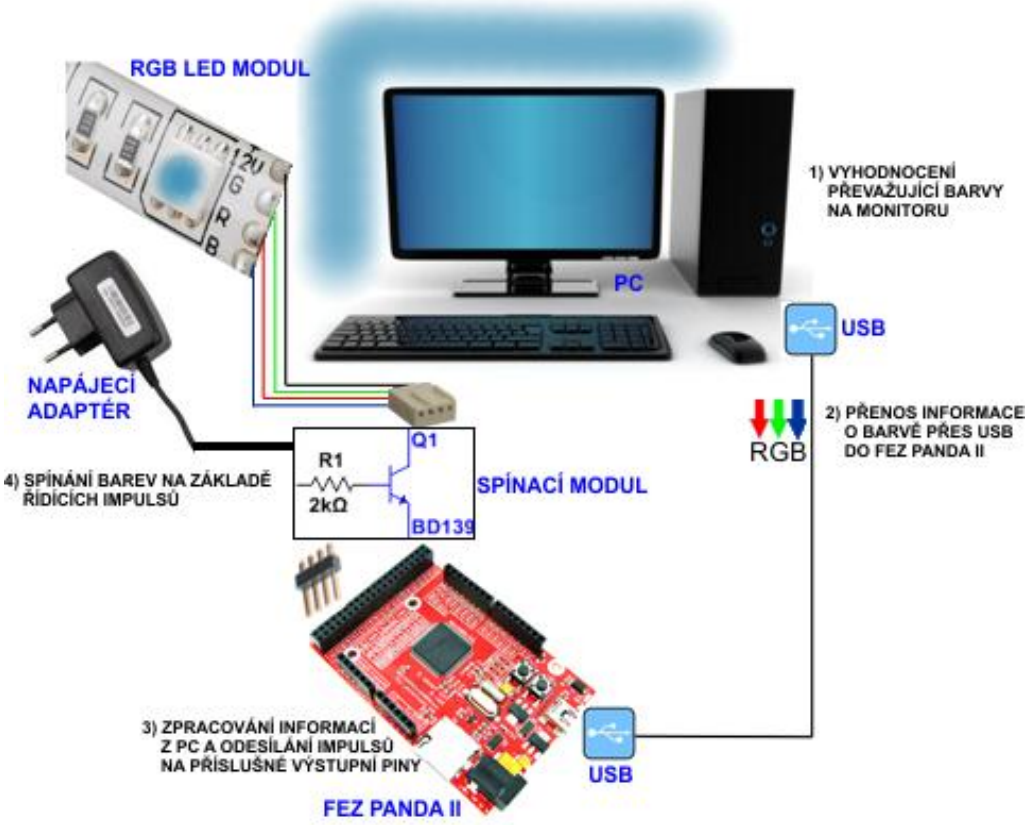
RGB LED modul se skládá ze 30 kusů výkonných RGB LED diod, které jsou umístěny na pásku dlouhém 50 cm a zalité silikonem. Pásek je dělitelný po 3 LED diodách (po 5cm). Tyto 5 cm dlouhé části jsou navzájem propojeny paralelně a jakkoliv dlouhý pásek lze tedy připojit na zdroj stejnosměrného napětí 12V s dostatečným maximálním proudem.

---

[2] LEDPASKY.COM, *LED Pásky* [online]. 2011 [cit. 2012-12-08]. Dostupné z: http://www.ledpasky.com/led-pasek-rgb-5050-oemcn-60-ledm-ip65-vodeodolny

5 Návrh řešení

5.1 Blokové schéma



Obrázek č. 3 – Blokové schéma

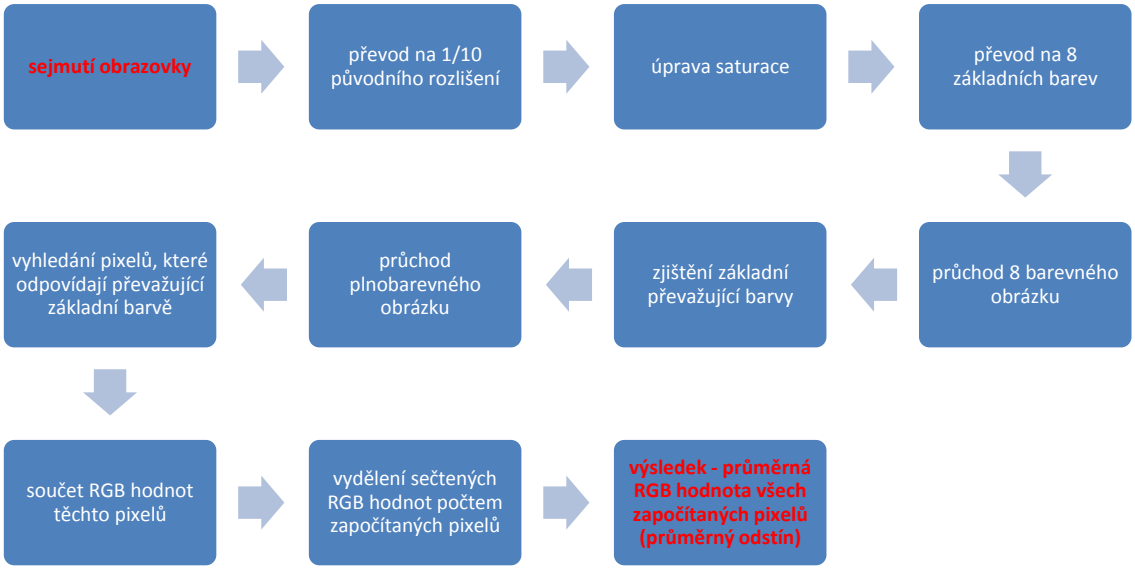
5.2 USB komunikace mezi vývojovou deskou a PC

Komunikace mezi PC a vývojovou deskou bude probíhat přes virtuální COM port. Na straně PC vytvoříme virtuální COM port a následně ho otevřeme. Vývojovou desku nastavíme do režimu virtuálního COM portu, aby byla připravena přijímat data. Vždy, když vypočítáme hodnotu převažující barvy na monitoru, na virtuální COM port odešleme informaci o této barvě ve formě tří hodnot typu BYTE (R, G, B), které budou v rozmezí od 0 do 20 (počet odstínů, které je zařízení schopné zobrazit je tedy $20^3 = 8\ 000$).

5.3 Řízení barev RGB LED modulu

Barvy RGB modulu budou v reálném čase nastavovány na základě hodnot přijímaných přes virtuální COM port. Jednotlivé odstíny budou nastaveny změnou frekvence blikání. Perioda blikání bude 20ms a hodnota z virtuálního COM portu bude přímo odpovídat době v milisekundách po jakou bude daná barva v této periodě rozsvícena(např. fialová barva s plným jasem bude mít hodnotu 20;0;20 a s jasem 50% bude hodnota 10;0;10).

5.4 Princip získávání informací o převažující barvě



Obrázek č. 4 – Princip zjišťování převažující barvy

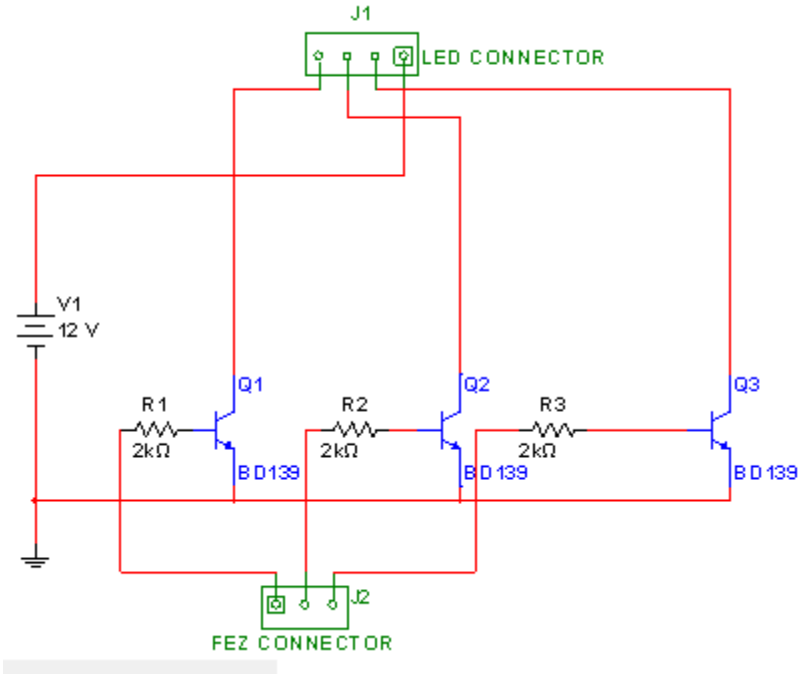
6 Realizace hardware

6.1 Vývojová deska FEZ Panda II

Jedná se o již sestavenou vývojovou desku, kterou není potřeba jakkoliv upravovat. Pořizovací cena je 900,-. K desce je dodáván USB kabel a na internetových stránkách výrobce je zdarma ke stažení software a ovladače pro operační systém Microsoft Windows.

6.2 Spínací modul

6.2.1 Schéma zapojení



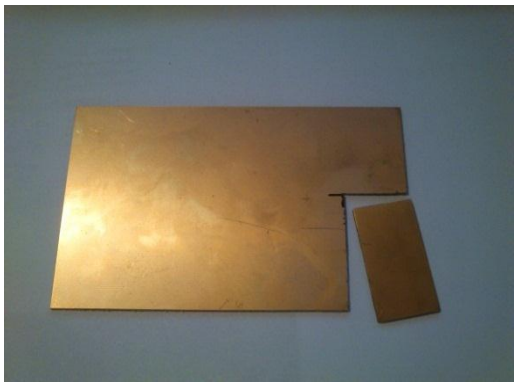
Obrázek č. 5 – Schéma zapojení spínacího modulu

6.2.2 Výroba plošného spoje

Plošný spoj jsem se rozhodl vyrobit metodou nažehlení toneru, která je podle mého názoru nejlepší, pokud plošný spoj vyrábíme v domácích podmínkách.

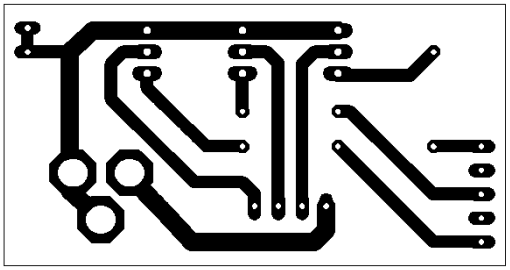
K výrobě budeme potřebovat: cuprextit, laserovou tiskárnu, lepící barevné papíry, mycí prostředek na nádobí, žehličku, papír na pečení, leptací roztok (např. chlorid železitý), ředidlo (např. odlakovač na nehty), vrtačku, pájku, cín a ochranný nátěr (např. roztok kalafuny a toluenu).

1) Z desky cuprextitu vyřízneme obdélník o potřebné velikosti, který důkladně očistíme a odmastíme např. přípravkem na nádobí.



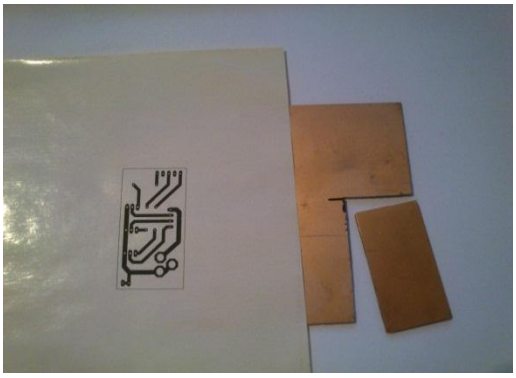
Obrázek č. 6 – Vyříznutá destička cuprextitu

2) Navrhneme plošný spoj (např. v programu Eagle)



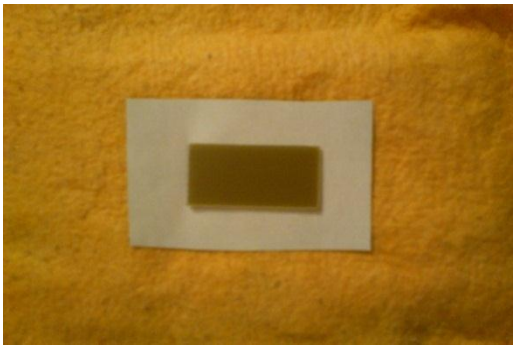
Obrázek č. 7 – Návrh plošného spoje v programu Eagle

3) Na lepící stranu lepícího papíru natiskneme návrh plošného spoje bez zrcadlového převrácení v nejvyšší tiskové kvalitě na laserové tiskárně



Obrázek č. 8 – Vytištěný návrh plošného spoje

4) Vyříznutý a očištěný cuprextit přiložíme měděnou stranou na vytištěný návrh



Obrázek č. 9 – Cuprextit na vytištěném návrhu

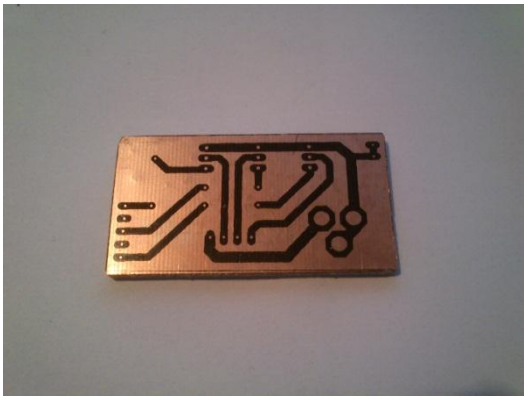
5) Okraje přeložíme, aby se destička nehýbala, přikryjeme papírem na pečení, žehličku nastavíme přibližně na poloviční výkon a mírným přítlakem cuprextit prohříváme 1-2 minuty

Obrázek č. 10 – Nažehlování toneru na cuprextit

6) Po nažehlení hodíme cuprextit i s nažehleným papírem do vody na 5-10 minut, papír se rozmočí a jde lépe sundat

Obrázek č. 11 – Rozmočující se papír

7) Po rozbalení by měl být toner přenesený z papíru na cuprextit



Obrázek č. 12 – Nažehlený toner

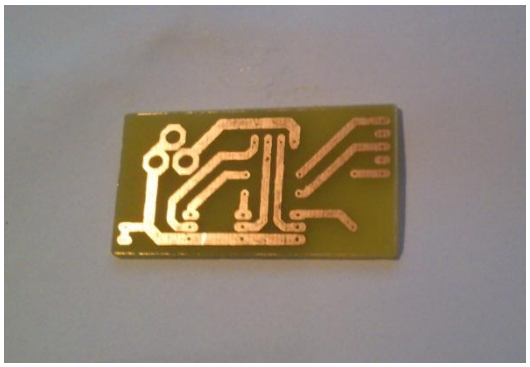
8) Cuprextit hodíme do leptacího roztoku, který by měl být ohřátý na 50°C (láhev s leptacím roztokem necháme několik minut pod proudem horké vody a urychlí se tak proces leptání), proces můžeme také urychlit mícháním

Obrázek č. 13 – Cuprextit v leptacím roztoku

9) Takto vypadá plošný spoj po vyleptání nezakrytých částí

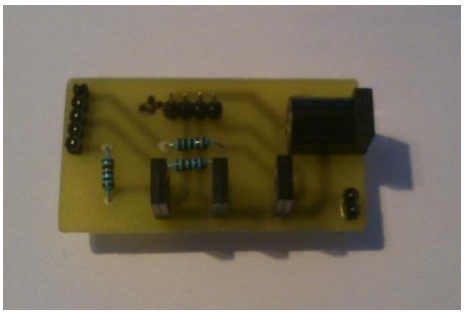
Obrázek č. 14 – Vyleptaný plošný spoj s vrstvou toneru

10) Toner setřeme ředidlem a vrtákem o velikosti 0,7mm vyvrtáme otvory pro součástky a po osazení natřeme spodní část ochranným nátěrem

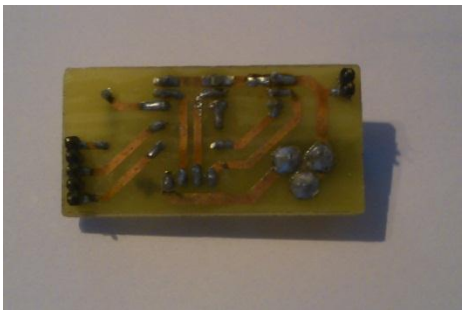


Obrázek č. 15 – Vyleptaný plošný spoj očištěný od toneru

11) Takto vypadá hotový plošný spoj po osazení součástkami



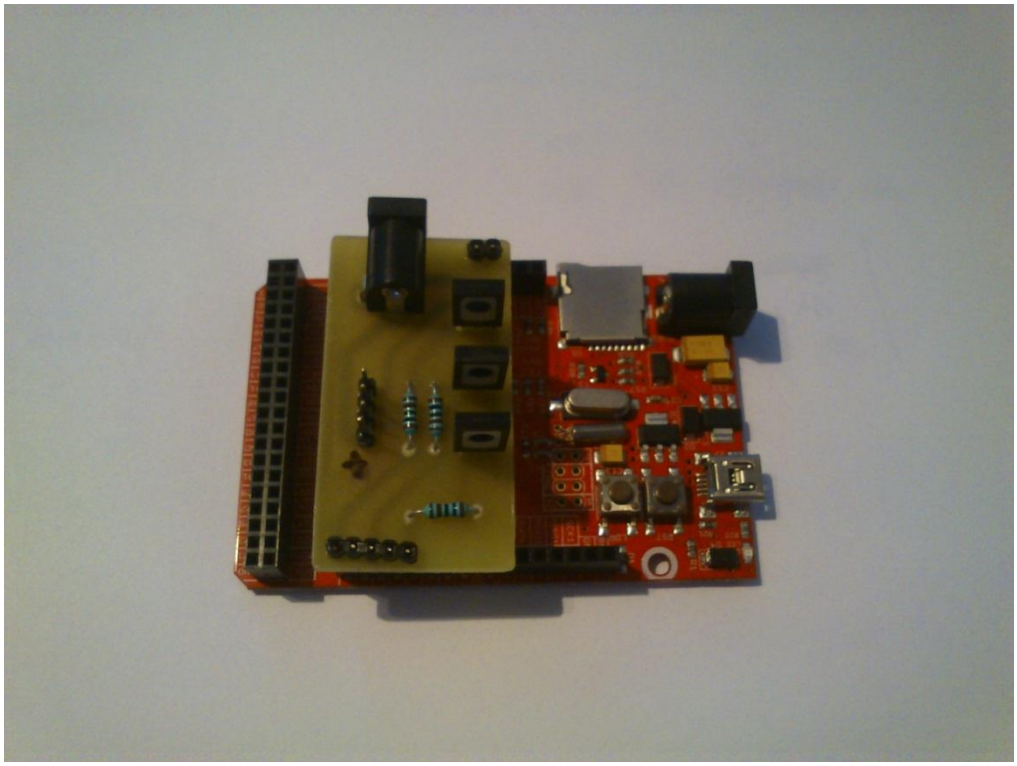
Obrázek č. 16 – Horní strana plošného spoje



Obrázek č. 17 – Spodní strana plošného spoje

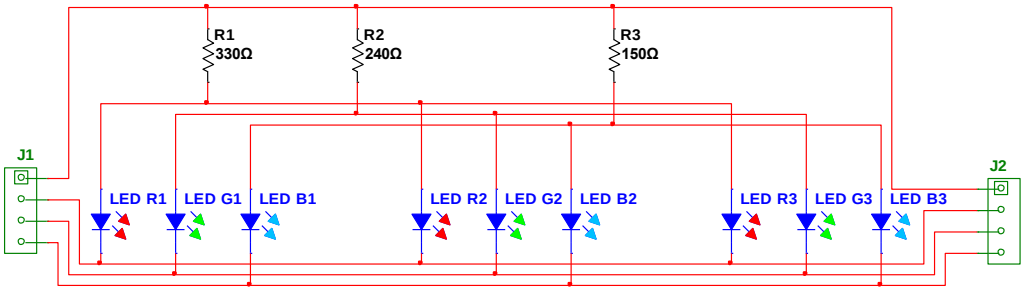
6.2.3 Připojení spínacího modulu

Spínací modul jsem navrhl tak, aby se dal přímo nasadit na vývojovou desku jako přídavný rozšiřující modul pomocí pinů, které jsou umístěny na spodní straně plošného spoje. Na horní straně je umístěn 4 pinový konektor pro připojení RGB LED modulu a napájecí konektor pro připojení napájecího adaptéru.



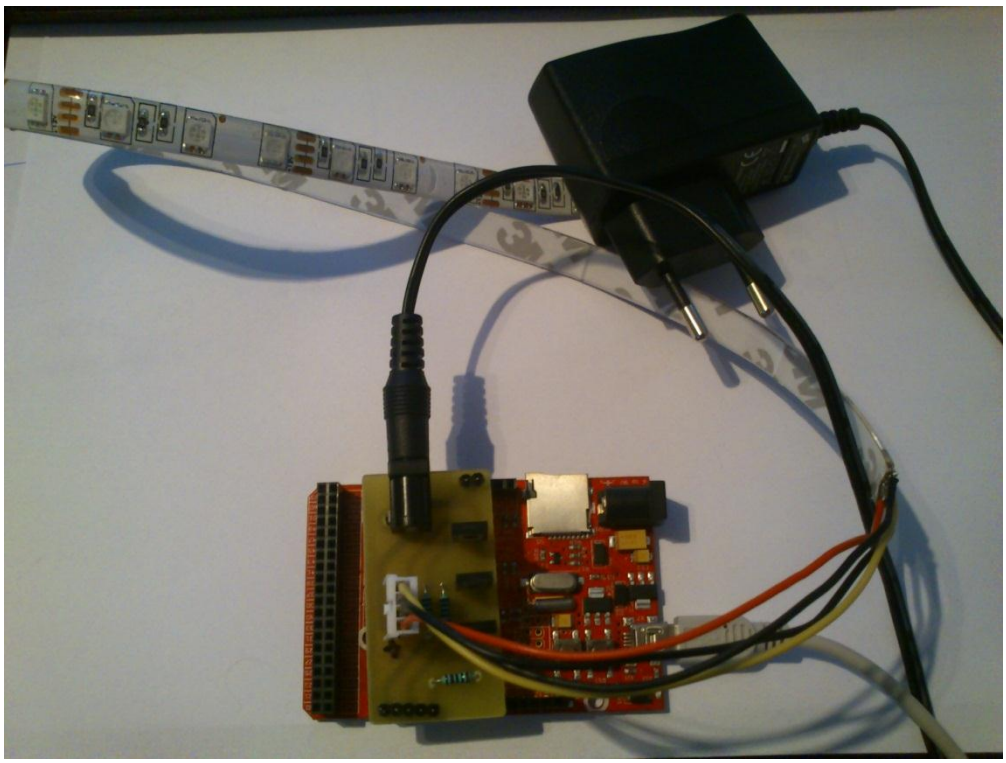
Obrázek č. 18 – Propojení spínacího modulu a vývojové desky

6.3 RGB LED modul



Obrázek č. 19 – Schéma vnitřního zapojení RGB LED pásku

6.4 Hotové zařízení



Obrázek č. 20 – Hotové zapojené zařízení

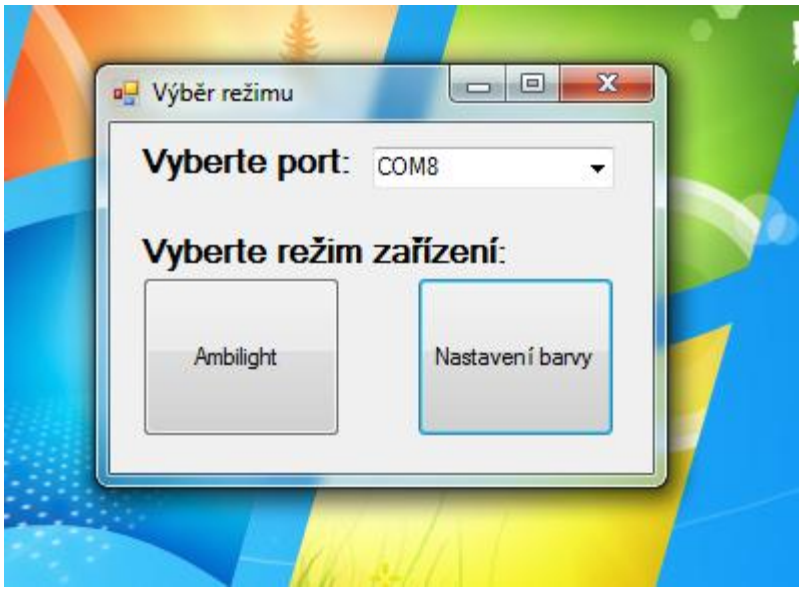
7 Implementace software

7.1 Postup instalace a spuštění

1) zařízení připojíme k počítači
2) nainstalujeme přiložené ovladače
3) po spuštění uživatelské aplikace můžeme zařízení začít používat

7.2 Software pro PC

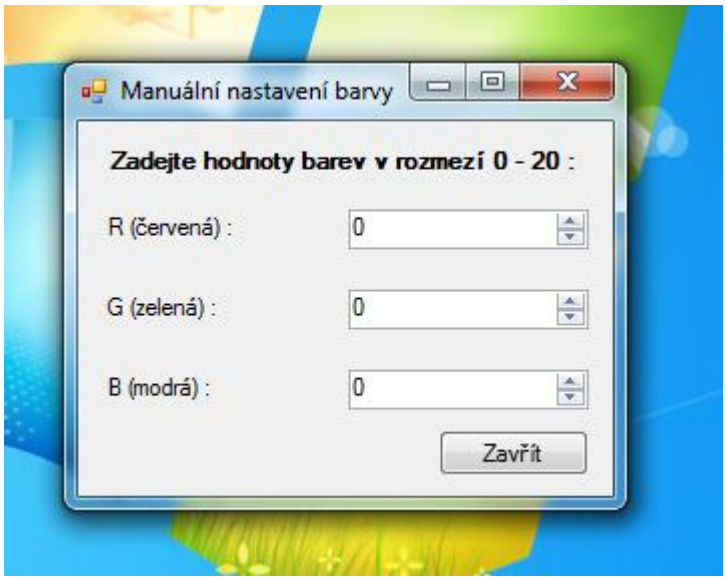
Na straně PC má uživatel možnost výběru z režimu Ambilight (automatický režim osvětlení podle převažující barvy na monitoru) nebo manuálního nastavení barvy. Pokud zařízení nefunguje s původním nastavením, je potřeba vybrat správný COM port pro odesílání informací.



Obrázek č. 21 – Výběr režimu osvětlení

7.2.1 Režim manuálního nastavení barvy

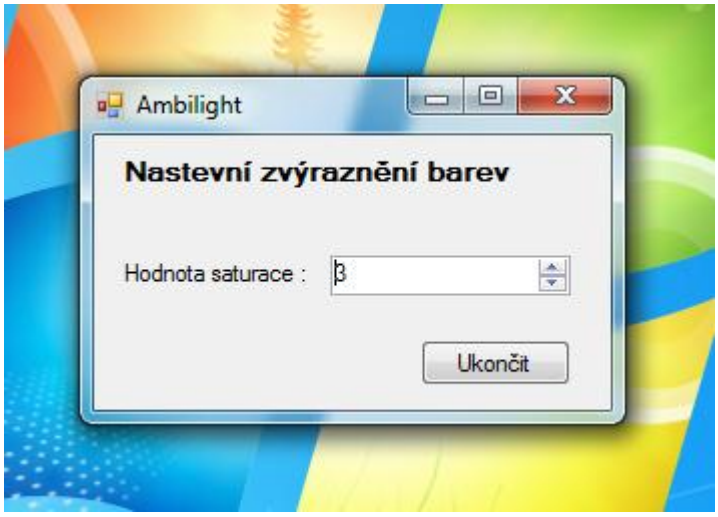
Uživatel má možnost nastavit si barvu světla manuálně zadáním hodnot RGB přímo v uživatelské aplikace a má na výběr ze všech 8000 odstínů, které je zařízení schopné zobrazit. Tento režim je vhodný pro efektivní barevné osvětlení interiéru pokud uživatel zrovna např. nesleduje žádný film. Hodnoty jsou nastavovány v rozmezí od 0 do 20 tlačítky nebo přímo zadáním číselné hodnoty.



Obrázek č. 22 – Manuální nastavení barvy

7.2.2 Režim osvětlení podle převažující barvy na obrazovce

Aplikace v tomto režimu běží na pozadí a nastavuje barvu osvětlení automaticky podle barvy, která v daném okamžiku převažuje na monitoru uživatele. Uživatel má možnost kalibrace nastavením hodnoty saturace (zvýraznění) barev – toto nastavení se neprojeví na monitoru, ale pouze na zařízení.



Obrázek č. 23 – Nastavení saturace v režimu Ambilight

7.2.3 Princip vyhodnocení převažující barvy

1) Sejmutí obrazovky tak jak ji vidí uživatel

Obrázek č. 24 – Snímek sejmuté obrazovky

2) Obrázek je převeden na 10x nižší rozlišení pro urychlení výpočtů a je provedena saturace podle hodnoty nastavené uživatelem

Obrázek č. 25 – Snímek obrazovky po změně rozlišení a úpravě saturace

3) Obrázek je převeden na 8 základních barev – nyní můžeme zjistit, která základní barva se vyskytuje nejčastěji a kde se na obrázku nachází

Obrázek č. 26 – Snímek převedený na 8 základních barev

4) Zjistili jsme, že nejčastěji se vyskytující barva je červená a kde se nachází (ostatní oblasti už pro nás nejsou důležité)

Obrázek č. 27 – Snímek obrazovky po změně rozlišení a úpravě saturace

5) Nyní do obrázku doplníme původní hodnoty červených pixelů, abychom mohli vypočítat průměrný červený odstín – RGB hodnoty všech pixelů sečteme a vydělíme celkovým počtem pixelů

Obrázek č. 28 – Snímek obrazovky po doplnění původních barev

Na tomto posledním obrázku můžeme také vidět příklad toho, kdy uživatel nastavil příliš vysokou hodnotu saturace – mezi červené pixely se po zvýraznění barev započítaly i hnědé oblasti z pozadí obrázku a dojde tak k nepatrnému zkreslení výsledného odstínu červené.

7.2.4 Odesílání informací o barvě

Výsledkem výpočtu převažující barvy jsou RGB hodnoty v rozsahu 0-255. Naše zařízení rozlišuje pouze 20^3 barev, takže převedeme původní hodnoty na rozsah 0-20 jednoduchým výpočtem RGB / 255 * 20.
Nyní máme 3 hodnoty v rozsahu 0-20, které potřebujeme do zařízení odeslat.
Nejprve vytvoříme v PC virtuální COM port příkazem:

```
SerialPort sp = new SerialPort("COM", 128000, Parity.None, 8, StopBits.One);
```

Následně port otevřeme příkazem:

```
sp.Open();
```

Nyní je virtuální COM port na straně PC otevřen a připraven odesílat data.
Data budeme odesílat ve formátu jednorozměrného pole o třech hodnotách (R, G, B) typů BYTE příkazem:

```
byte[] odeslani = new byte[3];
sp.Write(odeslani, 0, 3);
```

7.3 Software pro FEZ Panda II

7.3.1 Příjem informací o barvě

Zařízení nastavíme do režimu virtuálního COM portu příkazem:

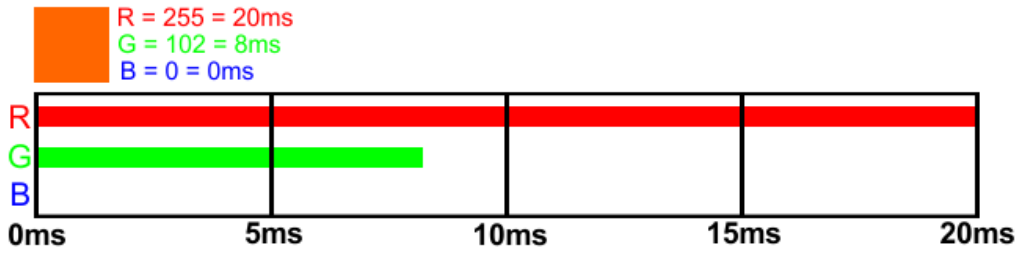
```
USBC_CDC port = USBClientController.StandardDevices.StartCDC_WithDebugging();
```

Poté můžeme přijímat informace odesílané z PC příkazem:

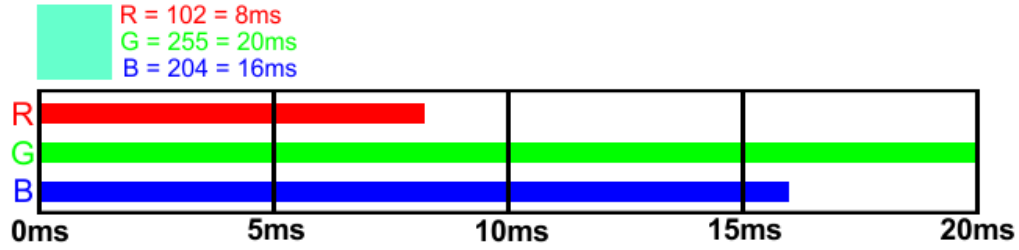
```
port.Read(buf, 0, 3);
```

7.3.2 Rozsvícení barvy podle vstupních hodnot

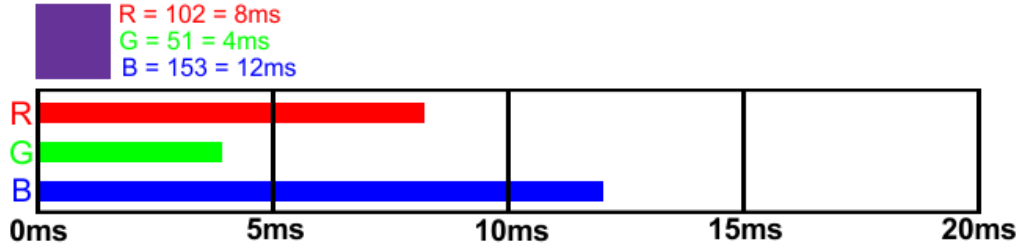
Vstupní hodnoty z virtuálního COM portu přímo odpovídají době v milisekundách, po kterou je daná barva rozsvícena v opakující se periodě 20ms. Tím vznikne požadovaný odstín a jas. Na následujících obrázcích je znázorněn princip blikání LED diod v periodě včetně převodu RGB hodnoty na hodnotu udávající počet milisekund pomocí vzorce RGB / 255 * 20.



Obrázek č. 29 – Znázornění principu míchání barev (oranžová)



Obrázek č. 30 – Znázornění principu míchání barev (azurová)

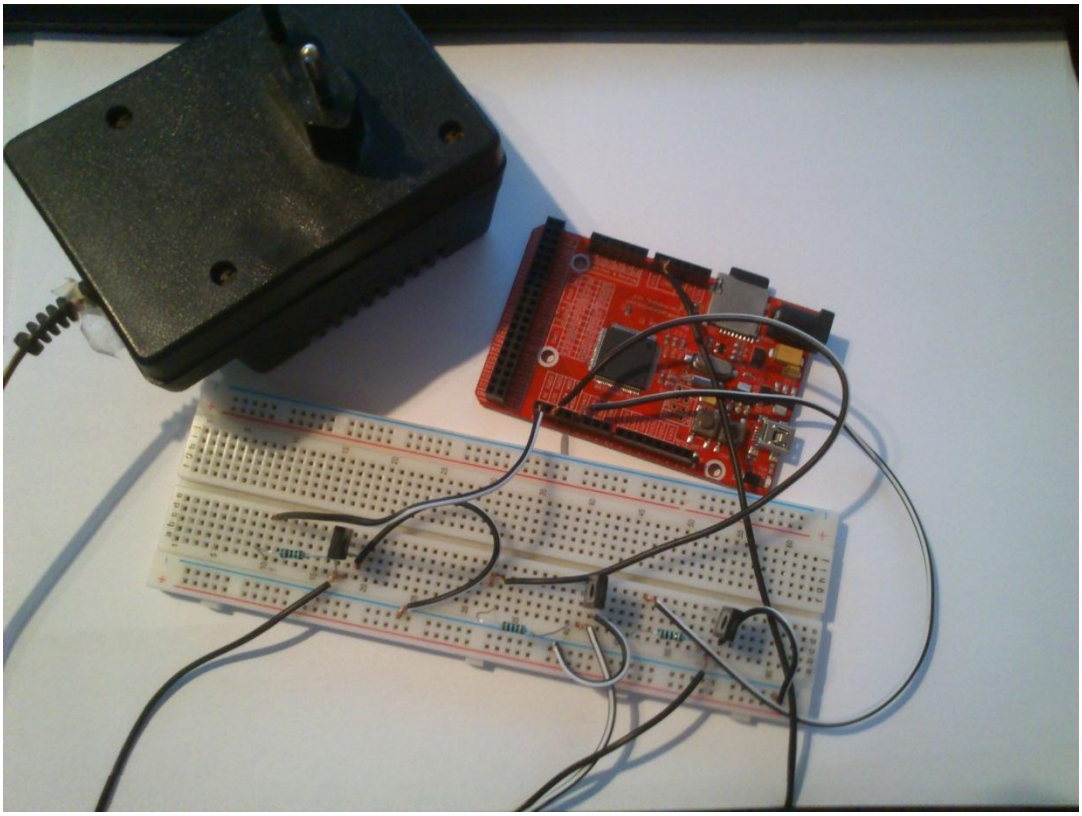


Obrázek č. 31 – Znázornění principu míchání barev (fialová)

8 Testování (HW i SW)

8.1 Testování funkčnosti HW

Funkčnost hardwaru byla testována na zkušebním zapojení, které bylo realizované na zkušební desce a s napájecím adaptérem, který měl regulovatelné napětí. Jednalo se o dočasné testovací řešení před výrobou plošného spoje.



Obrázek č. 32 – Testovací zapojení

8.2 Testování vyhodnocování převažující barvy

Testování bylo prováděno na velkém množství fotografií a obrázků, u kterých bylo okem zřetelné, jaká barva převažuje. Algoritmus funguje správně a vždy vyhodnotil převažující barvu podle mého odhadu.

8.3 Testování správného míchání barev

Míchání barev bylo testováno programem Microsoft Paint (malování). Změnou barvy papíru na celé obrazovce byly odzkoušeny všechny odstíny z palety barev a bylo sledováno, zda odstín na obrazovce odpovídá odstínu rozsvíceného RGB LED modulu. Testování odstínů proběhlo úspěšně a nebylo potřeba provádět žádnou kalibraci.

8.4 Testování celého projektu

Testování celého zařízení jako celku včetně uživatelské aplikace bylo prováděno na třech různých stanicích:

1) Typ: stolní PC – **2 jádra**

   Monitor: 22“ (1680x1050)

   CPU: AMD Athlon II X2 245 2.9GHz

   GPU: ATI Radeon 4650

   RAM: 4GB DDR2

   OS: Microsoft Windows 7 Professional 64-bit

   Hodnocení testu: Zařízení na tomto HW fungovalo podle mých představ, výkon byl dostačující a při spuštěné aplikaci bylo možné bez problému pracovat s ostatními aplikacemi nebo sledovat video bez sekání i ve vysokém rozlišení (1080p).

2) Typ: stolní PC – **1 jádro**

   Monitor: 24“ (1920x1080)

   CPU: Intel Pentium 4 3.06GHz

   GPU: nVidia GeForce 8400GS

   RAM: 2GB DDR

   OS: Microsoft Windows 7 Ultimate 32-bit

Hodnocení testu: Reakce zařízení byla podstatně pomalejší než u ostatních stanic (až 3x delší odezva mezi monitorem a zařízením), kvůli úplnému vytížení CPU docházelo také při sledování videa ve vyšším rozlišení (720p a vyšší) k sekání obrazu.

3) Typ: notebook Acer – **2 jádra**

   Monitor: 17“ (1360x768)

   CPU: Intel Core 2 Duo P8400 2.26GHz

   GPU: nVidia GeForce 9600M GS

   RAM: 4GB

   OS: Microsoft Windows 7 Professional 32-bit

   Hodnocení testu: Výsledky na této stanici byly velice podobné výsledkům stanice č.1, reakce zařízení byla ještě o něco rychlejší (zřejmě díky nižšímu rozlišení monitoru), ale rozdíl byl při pozorování téměř neviditelný.

9 Návrhy pro budoucí řešení

9.1 Snížení náročnosti na HW

Pokud je aplikace spuštěna v režimu Ambilight, probíhá neustálé snímání obrazovky a vyhodnocování pořízeného snímku. Snímání obrazovky ve vysokém rozlišení je hardwarově náročná operace a proto by bylo potřeba navrhnout řešení, jak získat pouze informace, které potřebujeme (informace o barvách) přímo z grafické karty nebo přes DirectX. Tím by se náročnost na hardware určitě snížila a také by se zmenšilo zpoždění mezi sejmutím obrazovky a rozsvícením LED modulu, které je způsobeno dlouhou dobou výpočtu (řádově desítky milisekund).

9.2 Odstranění viditelného blikání

Dalším problémem je blikání RGB LED diod při nízkých hodnotách jasu. Toto blikání je způsobeno mícháním barev změnou frekvence a pokud je jas nízký, barva svítí např. pouze 1ms z 20ms periody. Toto blikání by se dalo odstranit návrhem přídavného modulu, který by měnil jas plynou regulací napětí na jednotlivých barvách.

9.3 Návrh řešení pro další verzi zařízení

Další verze zařízení by měla být inspirována technologií Philips Ambilight Spectra 3. Tato technologie už nevypočítává převažující barvu na celé obrazovce jako předchozí verze, ale snímá pouze okraje obrazu, na které barevně navazuje osvětlení za TV, jak je vidět na následujícím obrázku.

Obrázek č. 33 – Philips Ambilight Spectra 3

Toho bych chtěl dosáhnout nahrazením obyčejného RGB LED pásku za RGB modul, který je osazen integrovanými čipy a je u něj možné nastavit barvu každé RGB LED diody nezávisle. Tím se samozřejmě zvýší pořizovací cena celého zařízení a také nebude tak univerzální jako první verze – bude muset být navrženo vždy pro konkrétní velikost monitoru, aby barvy správně navazovaly na obraz. Výsledek je ale mnohem efektivnější a při sledování filmů vyvolává iluzi, že je obraz opticky větší, než když koukáte na klasický monitor.

9.4 Další možnosti využití

Celé zařízení je založené na programovatelné vývojové desce, takže je velmi univerzální a po drobných úpravách ho můžeme použít k ovládání téměř čehokoliv. Pokud by jsme např. tranzistory na spínacím modulu nahradili spínacím relé, můžeme přes PC jednoduše ovládat osvětlení a elektroniku v celém domě pouhým kliknutím na příslušné tlačítko.

10 Závěr

V této bakalářské práci měly být splněny následující cíle:

- návrh a konstrukce funkčního USB zařízení
- výroba plošného spoje
- návrh a vytvoření softwaru a uživatelského rozhraní
- dokumentace

Myslím si, že všechny tyto cíle byly alespoň z velké části a v rámci mých možností, zkušeností a vědomostí splněny. Do této doby jsem zkoušel programovat aplikace pro zařízení pouze na emulátorech, ale při tomto projektu jsem si konečně vyzkoušel pracovat se skutečným USB zařízením. Další zkušeností byla určitě výroba plošného spoje. Dříve jsem vyráběl plošný spoj foto cestou – k tomu jsem ovšem potřeboval patřičné vybavení (zdroj UV světla). Metoda nažehlení toneru, kterou jsem použil je bez problému realizovatelná i v domácích podmínkách. Při návrhu software a uživatelské aplikace jsem využil své zkušenosti s programováním získané během studia a přiučil jsem se nové věci z oblasti práce s grafikou a programování v jazyce micro C#. Celé zařízení budu používat pro vlastní potřebu při sledování filmů a budu se ho pokoušet dále zdokonalovat.

11 Literatura a použité zdroje

Michal Dobeš – Zpracování obrazu a algoritmy v C# (ISBN: 978-80-7300-233-6)

David Matoušek – Udělejte si z PC… 2. díl (ISBN: 80-7300-072-5)

GHI Electronics FEZ Panda II - uživatelský manuál

http://www.ambx.com/

http://www.philips.cz/

http://social.msdn.microsoft.com/

http://www.codekeep.net/

12 Přílohy

12.1 Technické parametry FEZ Panda II

FEZ Panda II je mikropočítačová vývojová deska založená na open source platformě .NET Micro Framework.
Deska obsahuje 32-bit mikrokontrolér programovatelný z vývojového studia Visual Studio.

12.1.1 Hlavní vlastnosti

Postaven na NXP's LPC2387 micro-controller s GHI USBizi firmware/software package.
72MHz. 32-bit ARM7 procesor.
512 KB Flash (148KB pro uživatelská data).
96 KB RAM (62KB pro uživatelská data).
Kompatibilní s většinou Arduino shields.
USB připojení pro run-time debugging.
Specializované knihovny pro konfiguraci USB portu v režimu Host.
USB Debugging a Virtuální COM (CDC) mohou pracovat najednou.
Zabudovaná čtečka na Micro SD karty (4-bit vysokorychlostní s podporou SDHC, bez 2GB limitu) a se signálem pro detekci vložení karty.
54x Digital I/O portů.
6x 10-bit analog vstupů.
10-bit analog výstup (s přehráváním WAV audio).
6x Hardwarových PWM kanálů.
2x CAN kanály.
2KB RAM se záložní baterií.
Konfigurovatelné on-board LED a tlačítko.
4x UART serial porty.
OneWire rozhrání (dostupné na mnoho IO).
Zabudované hodiny reálného času (RTC) s optimálním krystalem.
Přístup do registrů procesoru.

Výstupní komparátor pro generování signálů s vysokou přesností.
RLP umožňující nahrát uživatelům nativní kód (C/Assembly) pro real-time aplikace.
Ethernet je podporován prostřednictvím čipu W5100 s plnou podporou TCP, UDP, HTTP, DHCP a DNS.
Propustnost ethernetu je 400Kbps. Ideální modul je FEZ Connect shield.
Parallel Port.
JTAG je deaktivován (k dispozici je jen při vymazání firmwaru).
Multi-Threading.
XML.
FAT Souborový Systém.
Kryptografie (AES a XTEA).
Aktualizace při běhu z SD, sítě a dalších.

12.1.2 Napájení

Přes USB port nebo pomocí externího stejnosměrného napětí o velikosti 6-9V (lze připojit obě napájení).
3.3V regulovaný DC výstup.
5.0V regulovaný DC výstup.
Digitální I/O jsou 3.3V ale 5V tolerantní.
Nízkonapěťové a hibernační módy.
Spotřeba v aktivním stavu: 103 mA.
Spotřeba v klidovém stavu: 65 mA.
Spotřeba při hibernaci: 3.75mA.

12.1.3 Prostředí

V souladu s RoHS /Lead-free.
Provozní teplota: -20 to 65°C.[1]

---

[1] GHI ELECTRONICS, GHI Electronics. *Snail Instruments* [online]. 2012 [cit. 2012-12-08]. Dostupné z: http://shop.snailinstruments.com/index.php?main_page=product_info&products_id=929&zenid=20b89ef8bc8661e7a6896ee8796b6e97

12.2 Technické parametry RGB LED pásku

Životnost [hod.]: 50 000 +

Typ uchycení: samolepící 3m páska na zadní straně

Uhel vyzařování světla: 120°

Výrobce: OEM CN

IP: 65

Šířka: 10 mm

Spotřeba na metr [W/m]: 14,4 W

Typ LED: 5050

Počet LED na metr: 60

Dělitelný po: 5 cm

Vodotěsnost: odolný vůči stříkající vodě

Výška: 0,22 mm

Barva: RGB

Napájecí napětí: 12V DC [2]

[2] LEDPASKY.COM, *LED Pásky* [online]. 2011 [cit. 2012-12-08]. Dostupné z: http://www.ledpasky.com/led-pasek-rgb-5050-oemcn-60-ledm-ip65-vodeodolny

12.3 Zdrojový kód aplikace pro PC

12.3.1 Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.IO.Ports;

namespace WindowsFormsApplication2
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new mode());

        }
    }
}
```

12.3.2 Mode.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Imaging;
using System.Drawing.Printing;
using System.IO;
using Colors;
using System.Drawing.Drawing2D;
using System.IO.Ports;
using System.Threading;

namespace WindowsFormsApplication2
{
    public partial class mode : Form
    {
        public mode()
        {
            InitializeComponent();
            // ukonceni procesu pri zavreni aplikace
            this.FormClosing += new FormClosingEventHandler(this.zavri);
        }

        private void zavri(object sender, System.ComponentModel.CancelEventArgs e)
        {
```

```
            Environment.Exit(0);
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.Hide();
            manual f = new manual();
            f.Show();

        }


        private void button1_Click(object sender, EventArgs e)
        {
            this.Hide();
            ambilight f = new ambilight();
            f.Show();
        }

    }
}
```

12.3.3 Manual.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.Threading;

namespace WindowsFormsApplication2
{
    public partial class manual : Form
    {
        // vytvoreni com portu
        static SerialPort sp = new SerialPort("COM8", 128000, Parity.None, 8,
StopBits.One);
        // vytvoreni noveho vlakna
        Thread t = new Thread(delegate() { vlakno(); });
        static byte[] odeslani = new byte[3];
        static int r, g, b;
        static bool pokracuj = true;

        public manual()
        {
            InitializeComponent();
            r = g = b = 0;
            pokracuj = true;
            // zavreni aplikace krizkem
            this.FormClosing += new FormClosingEventHandler(this.zavri);
            t.Start();
        }

        private void zavri(object sender, System.ComponentModel.CancelEventArgs e)
        {
            pokracuj = false;
```

```
            //dokonceni cinnosti beziciho vlakna
            t.Join();
            // zhasnuti led modulu
            odeslani[0] = 0;
            odeslani[1] = 0;
            odeslani[2] = 0;
            sp.Write(odeslani, 0, 3);
            // zavreni portu
            if (sp.IsOpen) sp.Close();
            // zruseni vlakna
            t.Abort();
            mode F1 = new mode();
            F1.Show();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            this.Close();
        }



        static void vlakno()
        {
           sp.WriteTimeout = 2000;
           // otevreni portu
           if(!sp.IsOpen)
               sp.Open();

            while (pokracuj)
            {
                // prepocet puvodnich hodnot RGB na hodnotu v milisekundach
                odeslani[0] = (byte)r;
                odeslani[1] = (byte)g;
                odeslani[2] = (byte)b;
                // odeslani hodnot na com port
                sp.Write(odeslani, 0, 3);
                Thread.Sleep(100);
            }
        }

        private void numericUpDown1_ValueChanged(object sender, EventArgs e)
        {
            r = (int)numericUpDown1.Value;

        }

        private void numericUpDown2_ValueChanged(object sender, EventArgs e)
        {
            g = (int)numericUpDown2.Value;
        }

        private void numericUpDown3_ValueChanged(object sender, EventArgs e)
        {
            b = (int)numericUpDown3.Value;
        }


    }
}
```

12.3.4 Ambilight.cs

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Imaging;
using System.Drawing.Printing;
using System.IO;
using Colors;
using System.Drawing.Drawing2D;
using System.IO.Ports;
using System.Threading;


namespace WindowsFormsApplication2
{
    public partial class ambilight : Form
    {
        // vytvoreni noveho vlakna
        Thread t = new Thread(delegate() { vlakno(); });

        public ambilight()
        {
            InitializeComponent();
            pokracuj = true;
            // ukonceni aplikace po zavreni krizkem
            this.FormClosing += new FormClosingEventHandler(this.zavri);
            //spusteni vlakna
            t.Start();
        }

        private void zavri(object sender, System.ComponentModel.CancelEventArgs e)
        {

            pokracuj = false;
            //dokonceni cinnosti beziciho vlakna
            t.Join();
            // zhasnuti led modulu
            odeslani[0] = 0;
            odeslani[1] = 0;
            odeslani[2] = 0;
            sp.Write(odeslani, 0, 3);
            // zavreni portu
            if (sp.IsOpen) sp.Close();
            // zruseni vlakna
            t.Abort();
            mode F1 = new mode();
            F1.Show();
        }

        [System.Runtime.InteropServices.DllImport("gdi32.dll")]
        public static extern bool DeleteObject(IntPtr hObject);

        [System.Runtime.InteropServices.DllImport("user32.dll")]
        public static extern int InvalidateRect(IntPtr hwnd, IntPtr rect, int
bErase);
```

```
        [System.Runtime.InteropServices.DllImport("user32.dll")]
        public static extern IntPtr GetDC(IntPtr hwnd);

        [System.Runtime.InteropServices.DllImport("gdi32.dll")]
        public static extern IntPtr CreateCompatibleDC(IntPtr hdc);

        [System.Runtime.InteropServices.DllImport("user32.dll")]
        public static extern int ReleaseDC(IntPtr hwnd, IntPtr hdc);

        [System.Runtime.InteropServices.DllImport("gdi32.dll")]
        public static extern int DeleteDC(IntPtr hdc);

        [System.Runtime.InteropServices.DllImport("gdi32.dll")]
        public static extern IntPtr SelectObject(IntPtr hdc, IntPtr hgdiobj);

        [System.Runtime.InteropServices.DllImport("gdi32.dll")]
        public static extern int BitBlt(IntPtr hdcDst, int xDst, int yDst, int w, int
h, IntPtr hdcSrc, int xSrc, int ySrc, int rop);
        static int SRCCOPY = 0x00CC0020;

        [System.Runtime.InteropServices.DllImport("gdi32.dll")]
        static extern IntPtr CreateDIBSection(IntPtr hdc, ref BITMAPINFO bmi, uint
Usage, out IntPtr bits, IntPtr hSection, uint dwOffset);
        static uint BI_RGB = 0;
        static uint DIB_RGB_COLORS = 0;

[System.Runtime.InteropServices.StructLayout(System.Runtime.InteropServices.LayoutKin
d.Sequential)]

        public struct BITMAPINFO
        {
            public uint biSize;
            public int biWidth, biHeight;
            public short biPlanes, biBitCount;
            public uint biCompression, biSizeImage;
            public int biXPelsPerMeter, biYPelsPerMeter;
            public uint biClrUsed, biClrImportant;

[System.Runtime.InteropServices.MarshalAs(System.Runtime.InteropServices.UnmanagedTyp
e.ByValArray, SizeConst = 256)]
            public uint[] cols;
        }

        static uint MAKERGB(int r, int g, int b)
        {
            return ((uint)(b & 255)) | ((uint)((r & 255) << 8)) | ((uint)((g & 255)
<< 16));
        }

        static public Image Resize(Image img, double percentage)
            // funkce pro zmenseni rozliseni sejmuteho obrazku
        {
            // zjisti velikost puvodniho obrazku
            int originalW = img.Width;
            int originalH = img.Height;

            // vypocet velikosti noveho obrazku podle procenta zmenseni
            int resizedW = (int)(originalW * percentage);
            int resizedH = (int)(originalH * percentage);

            // vytvoreni obrazku s novymi rozmery
```

```
            Bitmap bmp = new Bitmap(resizedW, resizedH);
            Graphics graphic = Graphics.FromImage((Image)bmp);
            graphic.InterpolationMode = InterpolationMode.HighQualityBicubic;
            graphic.DrawImage(img, 0, 0, resizedW, resizedH);
            graphic.Dispose();
            return (Image)bmp;
        }

        public struct PixelData
        {
            public byte Blue;
            public byte Green;
            public byte Red;
        }

        static unsafe int[] prevladajici(Bitmap obrazek, Bitmap obrazek_8)
        {
            int[] barvy = new int[8];
            Color c, d;
            int nejvetsi = 0;
            int index_barvy = 0, R = 0, G = 0, B = 0;
            int vR = 0, vG = 0, vB = 0;
            int pocet = 0;
            int[] vysledek = new int[3];
            BitmapData obrazek_8_data, obrazek_data;

            obrazek_8_data = obrazek_8.LockBits(new Rectangle(0, 0, obrazek_8.Width,
obrazek_8.Height), ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);
            obrazek_data = obrazek.LockBits(new Rectangle(0, 0, obrazek.Width,
obrazek.Height), ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);

            // vynulování pole barev
            for (int i = 0; i < 8; i++)
                barvy[i] = 0;

            // zjistuje, kolikrat se ktera zakladni barva na obrazku vyskytuje
            // posun na dalsi radek v obrazku
            for (int i = 0; i < obrazek_8.Height; i++)
            {
                int* ukzt_8 = (int*)((int)obrazek_8_data.Scan0 + i *
obrazek_8_data.Stride);
                // posun doprava po radku
                for (int j = 0; j < obrazek_8.Width; j++)
                {
                    c = Color.FromArgb(*ukzt_8);

                    if (c.R == 0 && c.G == 0 && c.B == 0)
                        barvy[0]++;
                    if (c.R == 0 && c.G == 0 && c.B == 255)
                        barvy[1]++;
                    if (c.R == 0 && c.G == 255 && c.B == 0)
                        barvy[2]++;
                    if (c.R == 0 && c.G == 255 && c.B == 255)
                        barvy[3]++;
                    if (c.R == 255 && c.G == 0 && c.B == 0)
                        barvy[4]++;
                    if (c.R == 255 && c.G == 0 && c.B == 255)
                        barvy[5]++;
                    if (c.R == 255 && c.G == 255 && c.B == 0)
                        barvy[6]++;
                    if (c.R == 255 && c.G == 255 && c.B == 255)
                        barvy[7]++;
```

```
                ukzt_8++;
            }
        }

        // zzjistuje jaka zakladni barva se na obrazku vyskytuje nejcasteji
        for (int i = 0; i < 8; i++)
        {
            if (barvy[i] > nejvetsi)
            {
                nejvetsi = barvy[i];
                index_barvy = i;
            }
        }

        if (index_barvy == 0) { R = 0; G = 0; B = 0; }
        if (index_barvy == 1) { R = 0; G = 0; B = 255; }
        if (index_barvy == 2) { R = 0; G = 255; B = 0; }
        if (index_barvy == 3) { R = 0; G = 255; B = 255; }
        if (index_barvy == 4) { R = 255; G = 0; B = 0; }
        if (index_barvy == 5) { R = 255; G = 0; B = 255; }
        if (index_barvy == 6) { R = 255; G = 255; B = 0; }
        if (index_barvy == 7) { R = 255; G = 255; B = 255; }

        // nyni vime, jaka zakladni barva je na obrazku nejcasteji
        // projdeme 8 barevny obrazek znovu a pokud na tuto barvu narazime,
zjistime hodnotu puvodni barvy pixelu z puvodniho snimku
        // vsechny tyto barvy budeme scitat a na konci vysledek vydelime celkovym
poctem pixelu, ktere jsme zapocitali
        // tim vypocitame průmerny odstnim nejcasteji se vyskytujici barvy
        for (int i = 0; i < obrazek_8.Height; i++)
        {
            int* ukzt = (int*)((int)obrazek_data.Scan0 + i *
obrazek_data.Stride);
            int* ukzt_8 = (int*)((int)obrazek_8_data.Scan0 + i *
obrazek_8_data.Stride);
            // znovu projdeme obrazek a hledame barvu, u ktere jsme zjistili, ze
se vyskytuje nejcasteji
            for (int j = 0; j < obrazek_8.Width; j++)
            {
                c = Color.FromArgb(*ukzt_8);
                // pokud na tuto barvu narazime
                if (c.R == R && c.G == G && c.B == B)
                {
                    // kod barvy zjistujeme z puvodniho plnobarevneho obrazku
                    d = Color.FromArgb(*ukzt);
                    vR += d.R; vG += d.G; vB += d.B;
                    pocet++;
                }
                ukzt++;
                ukzt_8++;
            }
        }

        obrazek.UnlockBits(obrazek_data);
        obrazek_8.UnlockBits(obrazek_8_data);

        // soucet hodnot RGB vydelime poctem vsech zapocitanych pixelu - vyjde
nam prumerny odstin
        vR /= pocet; vG /= pocet; vB /= pocet;

        vysledek[0] = vR;
        vysledek[1] = vG;
```

```
            vysledek[2] = vB;

            return vysledek;

        }

        static System.Drawing.Bitmap CopyToBpp(System.Drawing.Bitmap b, int bpp)
            // funkce převzatá z webu http://social.msdn.microsoft.com/ pro převod
snímku na 8 základních barev
        {
            if (bpp != 1 && bpp != 8)
                throw new System.ArgumentException("1 or 8", "bpp");

            int w = b.Width;
            int h = b.Height;
            IntPtr hbm = b.GetHbitmap(); // this is step (1)
            //
            // Step (2): create the monochrome bitmap.
            // "BITMAPINFO" is an interop-struct which we define below.
            // In GDI terms, it's a BITMAPHEADERINFO followed by an array of two
RGBQUADs
            BITMAPINFO bmi = new BITMAPINFO();
            bmi.biSize = 40;  // the size of the BITMAPHEADERINFO struct
            bmi.biWidth = w;
            bmi.biHeight = h;
            bmi.biPlanes = 1; // "planes" are confusing. We always use just 1. Read
MSDN for more info.
            bmi.biBitCount = (short)bpp; // ie. 1bpp or 8bpp
            bmi.biCompression = BI_RGB; // ie. the pixels in our RGBQUAD table are
stored as RGBs, not palette indexes
            bmi.biSizeImage = (uint)(((w + 7) & 0xFFFFFFF8) * h / 8);
            bmi.biXPelsPerMeter = 1000000; // not really important
            bmi.biYPelsPerMeter = 1000000; // not really important
            // Now for the colour table.
            uint ncols = (uint)1 << bpp; // 2 colours for 1bpp; 256 colours for 8bpp
            /* bmi.biClrUsed = ncols;
              bmi.biClrImportant = ncols;
              bmi.cols = new uint[256]; // The structure always has fixed size 256,
even if we end up using fewer colours
              if (bpp == 1) { bmi.cols[0] = MAKERGB(0, 0, 0); bmi.cols[1] =
MAKERGB(255, 255, 255); }
              else { for (int i = 0; i < ncols; i++) bmi.cols[i] = MAKERGB(i, i, i);
}*/

            bmi.biClrUsed = 8;
            bmi.cols = new uint[256];
            bmi.biClrImportant = 8;
            // int[] colv=new int[6]{0,51,102,153,204,255};

            //  for (int i = 0; i < 8; i++)  bmi.cols[i] = MAKERGB(colv[i / 36],
colv[(i / 6) % 6], colv[i % 6]);

            bmi.cols[0] = MAKERGB(0, 0, 0);
            bmi.cols[1] = MAKERGB(0, 0, 255);
            bmi.cols[2] = MAKERGB(0, 255, 0);
            bmi.cols[3] = MAKERGB(0, 255, 255);
            bmi.cols[4] = MAKERGB(255, 0, 0);
            bmi.cols[5] = MAKERGB(255, 0, 255);
            bmi.cols[6] = MAKERGB(255, 255, 0);
            bmi.cols[7] = MAKERGB(255, 255, 255);
            // For 8bpp we've created an palette with just greyscale colours.
            // You can set up any palette you want here. Here are some possibilities:
```

```
            // greyscale: for (int i=0; i<256; i++) bmi.cols[i]=MAKERGB(i,i,i);
            // rainbow: bmi.biClrUsed=216; bmi.biClrImportant=216; int[] colv=new
int[6]{0,51,102,153,204,255};
            //          for (int i=0; i<216; i++)
bmi.cols[i]=MAKERGB(colv[i/36],colv[(i/6)%6],colv[i%6]);
            // optimal: a difficult topic:
http://en.wikipedia.org/wiki/Color_quantization
            //
            // Now create the indexed bitmap "hbm0"
            IntPtr bits0; // not used for our purposes. It returns a pointer to the
raw bits that make up the bitmap.
            IntPtr hbm0 = CreateDIBSection(IntPtr.Zero, ref bmi, DIB_RGB_COLORS, out
bits0, IntPtr.Zero, 0);
            //
            // Step (3): use GDI's BitBlt function to copy from original hbitmap into
monocrhome bitmap
            // GDI programming is kind of confusing... nb. The GDI equivalent of
"Graphics" is called a "DC".
            IntPtr sdc = GetDC(IntPtr.Zero);       // First we obtain the DC for the
screen
            // Next, create a DC for the original hbitmap
            IntPtr hdc = CreateCompatibleDC(sdc); SelectObject(hdc, hbm);
            // and create a DC for the monochrome hbitmap
            IntPtr hdc0 = CreateCompatibleDC(sdc); SelectObject(hdc0, hbm0);
            // Now we can do the BitBlt:
            BitBlt(hdc0, 0, 0, w, h, hdc, 0, 0, SRCCOPY);
            // Step (4): convert this monochrome hbitmap back into a Bitmap:
            System.Drawing.Bitmap b0 = System.Drawing.Bitmap.FromHbitmap(hbm0);
            //
            // Finally some cleanup.
            DeleteDC(hdc);
            DeleteDC(hdc0);
            ReleaseDC(IntPtr.Zero, sdc);
            DeleteObject(hbm);
            DeleteObject(hbm0);
            //
            return b0;
        }

        public const float LumR = 0.3086f, LumG = 0.6094f, LumB = 0.0820f;
        public static ImageAttributes GetSaturationImageAttributes(float saturation)
        // funkce převzatá z webu http://www.codekeep.net/ pro úpravu saturace -
zvýraznění barev
        {
            ImageAttributes attr = new ImageAttributes();

            float satCompl = 1.0f - saturation;
            float satComplR = LumR * satCompl;
            float satComplG = LumG * satCompl;
            float satComplB = LumB * satCompl;

            ColorMatrix m = new ColorMatrix(new float[5][]    {
            new float[5] {satComplR + saturation,    satComplR,    satComplR,
0.0f, 0.0f},
            new float[5] {satComplG,    satComplG + saturation,    satComplG,
0.0f, 0.0f},
            new float[5] {satComplB,    satComplB,    satComplB + saturation,
0.0f, 0.0f},
            new float[5] {0.0f,    0.0f,    0.0f,    1.0f,    0.0f},
            new float[5] {0.0f,    0.0f,    0.0f,    0.0f,    1.0f} } );

            attr.ClearColorKey();
```

```
            attr.SetColorMatrix(m);
            return attr;
        }

        public static Image SaturateImage(Image source, float saturation)
        // funkce převzatá z webu http://www.codekeep.net/pro úpravu saturace -
zvýraznění barev
        {
            ImageAttributes attr = GetSaturationImageAttributes(saturation);
            Bitmap saturated = new Bitmap(source);
            using (Graphics g = Graphics.FromImage(saturated))
            {
                g.DrawImage(source, new Rectangle(0, 0, source.Width, source.Height),
                0, 0, source.Width, source.Height, GraphicsUnit.Pixel, attr);
                return saturated;
            }
        }

        // vysledny - snimek po uprave rozliseni a saturace, vysledny_8 - snimek
prevedeny na 8 barev
        static Image vysledny, vysledny_8;
        static byte[] odeslani = new byte[3];
        // vytvoreni virtualniho com portu
        static SerialPort sp = new SerialPort("COM8", 128000, Parity.None, 8,
StopBits.One);
        static bool pokracuj = true;
        static float saturace = 3;
        private static Bitmap bmpScreenshot;
        private static Graphics gfxScreenshot;

        static void vlakno()
        {
            // pole do ktereho ukladame RGB hodnotu vysledne prevazujici barvy
            int[] vysledek = new int[3];
            sp.WriteTimeout = 2000;
            // otevreni portu
            if (!sp.IsOpen) sp.Open();

            do
            {
                // vytvoreni prazdneho bitmap oprazku o rozmerech rozliseni hlavniho
monitoru
                bmpScreenshot = new Bitmap(Screen.PrimaryScreen.Bounds.Width,
Screen.PrimaryScreen.Bounds.Height, PixelFormat.Format32bppArgb);
                // prevod bitmap na graphics
                gfxScreenshot = Graphics.FromImage(bmpScreenshot);
                // screenshot obrazovky
                gfxScreenshot.CopyFromScreen(Screen.PrimaryScreen.Bounds.X,
Screen.PrimaryScreen.Bounds.Y, 0, 0, Screen.PrimaryScreen.Bounds.Size,
CopyPixelOperation.SourceCopy);
                // prevod snimku na 1/10 puvodniho rozliseni
                vysledny = Resize((Image)bmpScreenshot, 0.1);
                // uprava saturace podle hodnoty nastavene uzivatelem
                vysledny = SaturateImage(vysledny, saturace);
                // prevod obrazku na 8 zakladnich barev
                vysledny_8 = CopyToBpp((Bitmap)vysledny, 8);
                // zjisteni prevladajici barvy na snimku
                vysledek = prevladajici((Bitmap)vysledny, (Bitmap)vysledny_8);

                odeslani[0] = (byte)((float)vysledek[0] / 255 * 20);
                odeslani[1] = (byte)((float)vysledek[1] / 255 * 20);
                odeslani[2] = (byte)((float)vysledek[2] / 255 * 20);
```

```
                // odeslani hodnot na virtualni com port
                sp.Write(odeslani, 0, 3);
                // uvolneni pameti
                GC.Collect();

            } while (pokracuj);

            return;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void numericUpDown1_ValueChanged(object sender, EventArgs e)
        {
            saturace = (float)numericUpDown1.Value;
        }

    }
}
```

12.4 Zdrojový kód aplikace pro FEZ Panda II

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.USBClient;
using GHIElectronics.NETMF.Hardware.LowLevel;

namespace FEZ_Panda_II_Application1
{
    public class Program
    {
        static byte[] buf = new byte[3];

        static bool ledState = false;

        // inicializace vystupnich pinu
        static OutputPort R = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.Di0, ledState);
        static OutputPort G = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.Di2, ledState);
        static OutputPort B = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.Di4, ledState);


        // nastaveni casu svitu a casu zhasnuti v ms pro cervenou barvu
        static void RED()
        {
            while (true)
            {
                if (buf[0] != 0) R.Write(true);
                Thread.Sleep((int)buf[0]);
                R.Write(false);
                Thread.Sleep(20 - (int)buf[0]);
            }
        }

        // nastaveni casu svitu a casu zhasnuti v ms pro zelenou barvu
```

```
        static void GREEN()
        {
            while (true)
            {
                if(buf[1]!=0) G.Write(true);
                Thread.Sleep((int)buf[1]);
                G.Write(false);
                Thread.Sleep(20 - (int)buf[1]);
            }
        }

        // nastaveni casu svitu a casu zhasnuti v ms pro modrou barvu
        static void BLUE()
        {
            while (true)
            {
                if (buf[2] != 0) B.Write(true);
                Thread.Sleep((int)buf[2]);
                B.Write(false);
                Thread.Sleep(20 - (int)buf[2]);
            }
        }

        public static void Main()
        {
            // vytvoreni noveho portu
            USBC_CDC port;

            // nastaveni zarizeni do rezimu virtualniho COM portu
            port = USBClientController.StandardDevices.StartCDC_WithDebugging();

            // cyklus k overeni funkcnosti virtualniho COM portu
            while (true)
            {
                if (USBClientController.GetState() !=
USBClientController.State.Running)
                {
                    Debug.Print("waiting to connect.");
                    Thread.Sleep(600);
                }
                else
                {
                    Debug.Print("CDC connected.");
                    break;
                }
            }


            // vytvoreni vlaken pro kazdou barvu - vsechny barvy musi byt
rozsvecovany a zhasinany soucasne
                    Thread tr = new Thread(delegate() { RED(); });
                    Thread tg = new Thread(delegate() { GREEN(); });
                    Thread tb = new Thread(delegate() { BLUE(); });

            // spusteni vlaken
                    tr.Start();
                    tg.Start();
                    tb.Start();

            // nekonecny cyklus pro prijem informaci z PC
           while(true)
           {
```

```
            port.Read(buf, 0, 3);
        }

     }

    }

}
```