ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA APLIKOVANÉ ELEKTRONIKY A TELEKOMUNIKACÍ

DIPLOMOVÁ PRÁCE

Distribuovaná ovládací jednotka pro ozvučovací systém automobilu

Stanislav Brtna **2017**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2016/2017**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Stanislav BRTNA**
Osobní číslo:	**E15N0065P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Dopravní elektroinženýrství a autoelektronika**
Název tématu:	**Distribuovaná ovládací jednotka pro ozvučovací systém automobilu**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

Navrhněte koncepci ovládací jednotky pro ovládání ozvučovacího systému v automobilu. Ovládací jednotka bude využívat dotykového displeje a komunikaci pomocí rozhraní CAN.

1. Seznamte se s funkcemi ovládací jednotky a specifikujte potřebné signály hardwarového rozhraní a základní strukturu uživatelského ovládacího rozhraní s využitím dotykového displeje.
2. Na základě předchozí analýzy navrhněte blokovou strukturu zařízení a specifikujte vhodný řídicí mikrokontrolér a dotykový displej.
3. Proveďte návrh obvodového řešení zařízení. Zařízení realizujte a oživte.
4. Navrhněte a realizujte knihovnu funkcí pro zobrazení vhodných ovládacích prvků s volbou možnosti vysílání obecně definovaných zpráv pomocí rozhraní CAN. Do použitého mikrokontroléru implementujte odpovídající firmware.
5. Diskutujte možnost uživatelské konfigurace a propojení zařízení s nadřazeným systémem.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah kvalifikační práce: **40 - 60 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce.

Vedoucí diplomové práce: **Ing. Petr Weissar, Ph.D.**
Katedra aplikované elektroniky a telekomunikací

Datum zadání diplomové práce: **3. července 2017**

Termín odevzdání diplomové práce: **21. srpna 2017**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 3. července 2017

Abstrakt

Práce popisuje návrh a výrobu řídící jednotky ozvučovacího systému s využitím kitu STM32F769I-DISCO.

Klíčová slova

STM32, CAN, Grafické rozhraní pro mikrokontroléry, GR2, SVS

Abstract

This master's thesis describes development of control unit for car audio system. Main component of that control unit is STM32F769I-DISCO development board from STM.

Keywords

STM32, CAN, Graphical user interface for micro-controllers, GR2, SVS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

……

podpis

V Plzni dne 21.8. 2017 Stanislav Brtna

Obsah

Seznam symbolů a zkratek

API	-	zkratka pro Application Programming Interface, aplikační programové rozhraní
CAN	-	Controller Area Network, datová sběrnice
GNU	-	Gnu Není Unix
LCD	-	liquid crystal display, display z tekutých krystalů
SD	-	Secure Digital, standard v oblasti paměťových karet
px	-	pixel, obrazový bod

1 Úvod

Cílem této diplomové práce je vyvinout distribuovanou řídicí jednotku pro ozvučovací systém v automobilu. Autor si není přesně jist tím, co všechno má být na řídící jednotce distribuované, tak se předpokládá, že vyvíjená řídící jednotka má být spíše součástí distribuovaného ozvučovacího systému, jen takto totiž zadání dává nějaký smysl. (Distribuovanost systému naznačuje jeho rozdělení na několik dílčích modulů jež spolu komunikují pomocí zpráv (balíků).)

V dalších kapitolách této práce jsou blíže specifikovány požadavky na výsledné zařízení, navrženo blokové schéma hardware, popsán návrh schématu. Dále je důkladně popsáno aplikační rozhraní a možnosti konfigurace řídícího softwaru. V závěru jsou zhodnoceny výsledky práce.

2 Shrnutí zadání

Zadání této diplomové práce specifikuje požadavky na návrh a ovládací jednotky pro distribuovaný ozvučovací systém automobilu. Tím se rozumí, že tato ovládací jednotka bude po rozhraní CAN ovládat zatím blíže nespecifikované audio zařízení. Uživatel bude s ovládací jednotkou pracovat pomocí jejího dotykového LCD displeje. Na displeji bude pro uživatelovo potěšení zobrazeno grafické rozhraní. Grafické rozhraní bude sestávat z několika obrazovek, na každou z nich bude možnost umístit ovládací prvky a přiřadit jim jistou funkcionalitu, spočívající například ve změně aktuální obrazovky nebo odeslání zprávy po rozhraní CAN.

3 Analýza existujících řešení

První část zadání se zabývá definicí problému. Autor práce předpokládal, že se tato část má zabývat specifikací konkrétního zařízení, jehož podoba je nastíněna dalšími body zadání. V první verzi práce bylo autorovi vytčeno, že neobsahuje analýzu existujících řešení podobné problematiky. Tato kapitola se věnuje právě analýze řešení tak, jak by k němu bylo přistupováno v komerčním sektoru.

3.1 Analýza zadání

Zadání diplomové práce přichází s předem navrženým konceptem zařízení. Toto zařízení by mělo disponovat blíže nespecifikovaným dotykovým LCD displejem, mělo by být řízeno nespecifikovaným mikrokontrolérem a jeho účel je ovládat audiosystém po sběrnici CAN. Firmware mikrokontroléru má umožňovat vykreslení uživatelského rozhraní na displeji a má umožnit prvkům uživatelského rozhraní přiřadit zprávy CAN.

Takto navržené zařízení jistě bude plnit svůj účel. Ale jedná se o nejlepší možnou cestu k řešení (tušeného) problému? Odpověď je nejistá, zadání nespecifikuje onen problém. Přistupme tedy k extrakci problému.

3.2 Analýza problému

Ubíráním funkčně nepodstatných prvků zadání, se dostaneme k potřebě ovládat audiosystém pomocí zařízení s dotykovým displejem. Jaký problém toto zařízení má řešit a jaký je jeho budoucí přínos? Začneme trochu oklikou ze strany audiosystému. Interiér moderního automobilu je vybaven velkým počtem reproduktorů, např. Škoda Superb jich má v základní výbavě osm. Díky jejich prostorovému rozmístění je možno úpravou signálu putujícího ke konkrétním reproduktorům vytvářet zajímavé efekty (typicky vnášení upraveného zvuku motoru do kabiny automobilu). Pro tvorbu takovýchto efektů je automobil vybaven digitálním signálovým procesorem (DSP). Jako většina věcí ve světě automotive technologií je velmi obtížné (i v univerzitním prostředí) modifikovat toto DSP za účelem vyzkoušený nově vyvíjených technologií.

To je onen problém, jež má řešit zařízení specifikované zadáním. Toto zařízení společně s nějakým komerčně dostupným a snadno modifikovatelným DSP utvoří systém, jež umožní snadnou prezentaci inovativních technologií přímo ve voze tímto systémem osazeným.

3.3 Analýza předkládaného řešení

Posuďme tedy nakolik předložené řešení ve své realizaci plní tuto potřebu. Začněme u dotykového LCD, takovéto panely, ovladatelné poměrně nevýkonným mikrokontrolérem, je problém pořídit ve velikosti a rozlišení, jež ve srovnání s komerčním řešením nepůsobí archaicky, nemluvě o faktické nesehnatelnosti takových panelů pro použití v automotive. (Automotive výrobci si nechávají takovéto panely vyrábět na zakázku, případně ve vlastní továrně provádějí jejich montáž spojením dotykového panelu a LCD, typ. Continental)

Tento problém je snadno řešitelný užitím ARM SoC. V komerčním sektoru by takovéto řešení představovalo standard . V hobby prostředí se vyloženě nabízí využít vývojové desky Raspberry-Pi, jež poskytuje dostatečný výkon a možnost stavby uživatelského rozhraní nad sofistikovanými grafickými knihovnami (případně webovým rozhraním). Rozvineme-li dále tuto myšlenku postavit zařízení nad Raspberry-Pi, musíme zvolit vhodnou zobrazovací jednotku a způsob komunikace s audiosystémem. Displej je možno zvolit buďto originální sedmi-palcové dotykové LCD k Raspberry-Pi, nebo jakýkoli LCD panel s HDMI konektorem pro displej a dotykovým kontrolérem s podporou

GNU/Linuxu. Komunikace se zbytkem audiosystému pomocí sběrnice CAN je v případě vyžití Raspberry-Pi problematičtější (neobsahuje CAN periferii), ale vzhledem k absenci přesnější specifikace audio protikusu, můžeme usuzovat, že dané zařízení by mohlo být ovládáno i jiným způsobem, pro který má Raspberry-Pi potřebnou periferii (Sériový port, I2C, SPI, Ethernet…)

Řešení předložené zadáním tedy trpí na malý výběr rozumných LCD displejů, na relativně malý výkon mikrokontrolérů, obtížnost a malou flexibilitu softwarového řešení. Při využití ARM SoC tyto problémy zanikají. Bohužel je tu jeden problém, který mají obě tato řešení a tím je finální montáž ve voze.

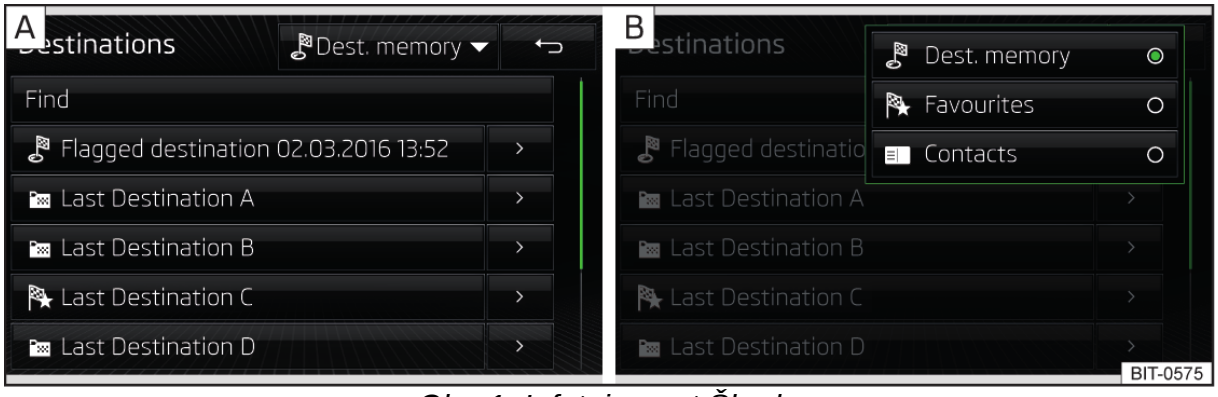
Jakékoli dodatečné zařízení v oblasti palubní desky bude vyžadovat buďto velmi agresivní zásah do interiéru vozu a nebo bude umístěno v nevzhledné "bradavici".

U vývojového prototypu a technologického demonstrátoru se s tímto problémem můžeme smířit. Ovšem pokud chceme přijít s moderním a čistým řešením, není ani jedna možnost přístupná. Řešením tohoto problému je ovládat audiosystém z velmi výkonného zařízení, vybaveného dotykovým displejem, kterým dnes disponuje téměř každý a které velmi často může být a nebo i je umístěno na palubní desce vozu.

3.4 Moderní a čisté řešení

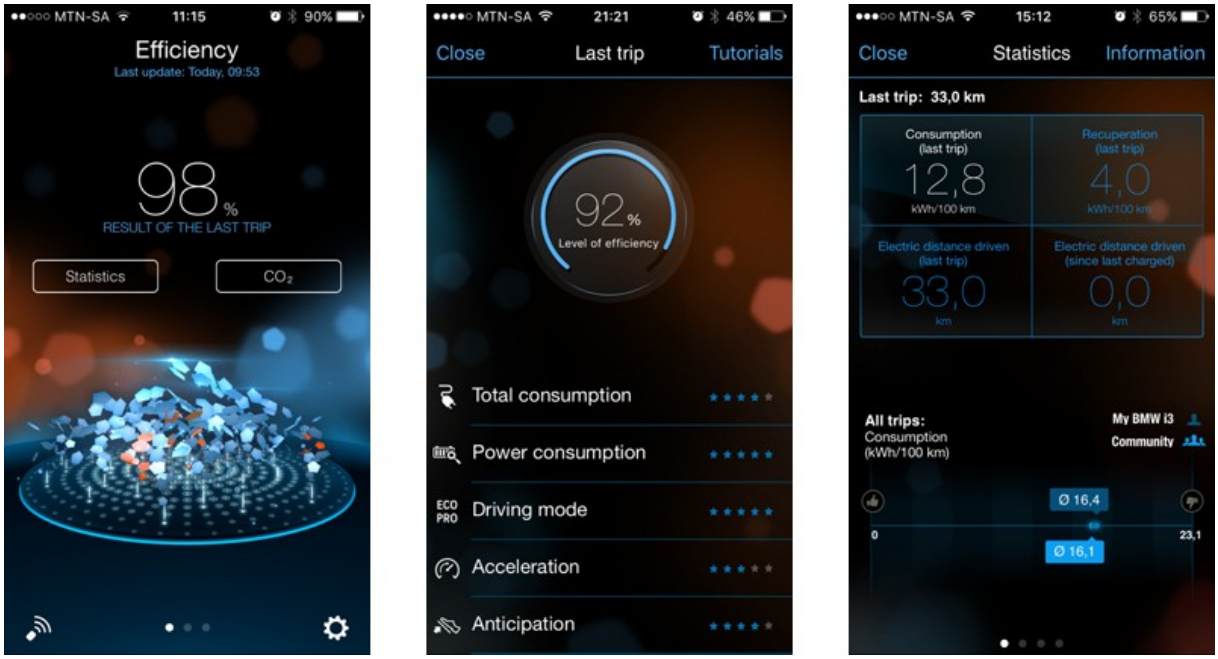
Zařízení o kterém je řeč v posledním odstavci předchozí kapitoly je samozřejmě mobilní telefon. S jeho dostatečným výkonem, kvalitním displejem a snadnou dostupností, není problém přijít s líbivou aplikací, jež bude například po rozhraní bluetooth komunikovat s Raspberry-Pi, které bude ovládat ono DSP.

3.5 Ukázky podobných komerčních řešení



Obr. 1: Infotainment Škoda

Obr. 2: Mobilní aplikace automobilky Tesla



Obr. 3: Mobilní aplikace automobilky BMW

Zajímavá je též mobilní aplikace Škoda Motor Sound, jejíž funkcionalita je velmi podobná funkcionalitě námi popisovaného zařízení. Tato aplikace modifikuje nastavení audiosystému vozu a umožňuje modifikaci zvuku motoru skrze reproduktory. Automobilka se v tomto případě rozhodla pro mobilní aplikaci, namísto modifikace software vozu.

3.6 Závěr analýzy

Bohužel toto hezké čisté a moderní řešení má s předloženým zadáním společné jen to, že by splnilo shodný účel, vyřešilo shodný problém. Autor nepředpokládá, že vypracovat zadání úplně jinak než jak je zadáno by bylo k jeho prospěchu. Proto je práce vypracována dle zadání.

Všechna řešení předložená touto analýzou jsou zcela realizovatelná v prostředí Západočeské Univerzity.

4 Bližší specifikace

Ve zbytku této kapitoly budou upřesněny požadavky na hardware a software.

4.1 Požadované grafické prvky

Grafické rozhraní řídící jednotky by mělo umožňovat vykreslení následujících grafických prvků. Tento seznam byl vypracován společně se zadavatelem práce.

4.1.1 Textové pole

Textové pole je jeden ze základních prvků jakéhokoli grafického uživatelského rozhraní, umožňuje programátorovi snadné vykreslení textu, který může informovat uživatele.

4.1.2 Tlačítko

Tlačítko umožňuje uživateli ovlivnit chod programu a zároveň může (svým popiskem) uživatele informovat o způsobu jakým bude chod programu ovlivněn. Tlačítko umožňuje vložit uživateli pouze logickou hodnotu.

4.1.3 Posuvník

Posuvník umožňuje uživateli předat programu numerickou hodnotu v určitém, předem daném, rozsahu. V této konkrétní aplikaci je potřeba implementovat jak jednorozměrný, tak dvojrozměrný posuvník.

4.2 Požadavky na konfigurovatelnost

Dalším požadavkem je snadná konfigurovatelnost uživatelského rozhraní. Toho bude dosaženo umístěním konfigurace prvků uživatelského rozhraní na SD kartu do snadno upravitelného textového souboru. Pro většinu úprav software řídící jednotky tedy nebude potřeba nic jiného, než počítač se čtečkou SD karet a textovým editorem.

4.3 Požadavky na hardware a volba discovery kitu od STM

Základním požadavkem na hardware je použití rozumného dotykového LCD. Dále pak mikrokontrolér o dostatečném výkonu aby dokázal plynule ovládat displej a dotykovou vrstvu. Posledním hardwarovým požadavkem je implementace CAN rozhraní a možnost napájení celé řídící jednotky z palubní sítě automobilu.

4.3.1 Displej

V ideálním případě by bylo využito dotykového displeje přímo pro automobilový průmysl. Bohužel tyto displeje si výrobci autoelektroniky nechávají vyrábět na zakázku a není možné je běžně zakoupit. Další problém by byl s pořízením dokumentace k takovémuto

displeji.

Na trhu dostupné dotykové displeje jsou zase příliš drahé, málo dostupné, jejich rozhraní je často špatně dokumentované. Proto bylo užito kitu STM32F769I-DISCO.

4.3.2 STM32F769I-DISCO

Tento vývojový kit od firmy STM je osazen poměrně kvalitním kapacitním dotykovým displejem, obsahuje procesor řady F7 a jedná se o špičku mezi vývojovými kity STM. Přes špičkové parametry je cena kitu poměrně nízká (2395.55 Kč na TME.cz). Cena displeje srovnatelného s displejem tohoto kitu by byla oproti ceně kitu dvojnásobná.

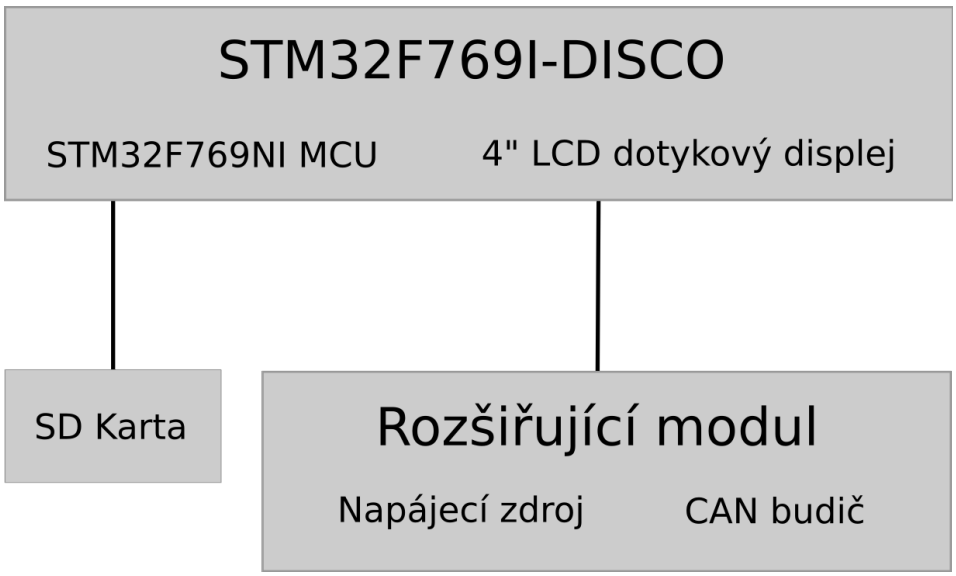
Další pozitivum užití tohoto kitu je možnost jeho budoucí znovupoužitelnosti pro jinou bakalářskou nebo diplomovou práci.

Užití tohoto kitu velmi zjednodušuje návrh hardware i software. Kit obsahuje většinu potřebných obvodů, je skvěle dokumentován a jsou k němu dostupné mnohé příklady ukázkových zdrojových kódů. Potřebné obvody které kit neobsahuje (zdroj a CAN budič) budou umístěny na rozšiřující modul, který bude s kitem propojen skrz rozšiřující konektor.

Obr. 4: Kit STM32F769I-DISCO

5 Návrh blokové struktury

Vzhledem k vyšší komplikovanosti užitého softwarového řešení, je části software věnována celá kapitola. Blokové schéma hardwarové části je uvedeno na obrázku 5.



Obr. 5: Blokové schéma

5.1 Hardware

Bloková struktura hardware systému je z velké míry podřízena realizaci převážné většiny hardware pomocí kitu STM32F769I-DISCO. Důvody pro volbu právě tohoto kitu jsou popsány v předchozí kapitole.

Zbytek hardware tvoří rozšiřující modul, jehož hlavním účelem bude implementace CAN budiče a napájecího zdroje.

6 Návrh obvodového řešení

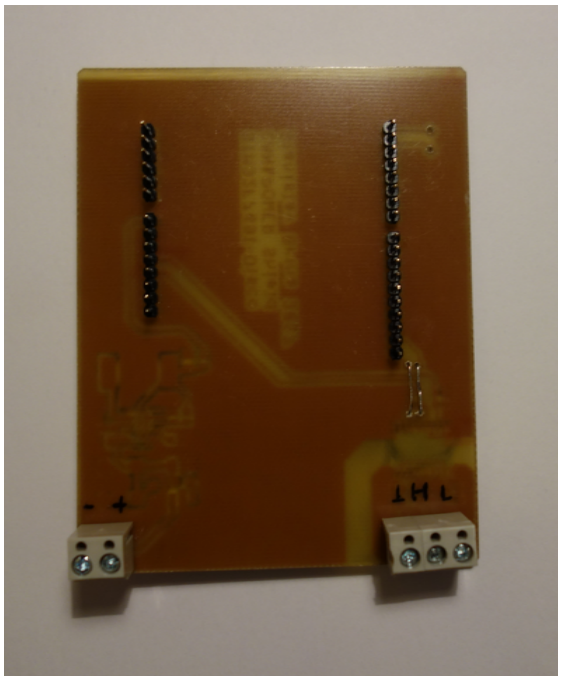
Návrh obvodového řešení se užitím kitu STM32F769I-DISCO velmi zjednodušil, dohromady není moc co navrhovat, většinu už implementuje užitý kit.

6.1 Rozšiřující modul

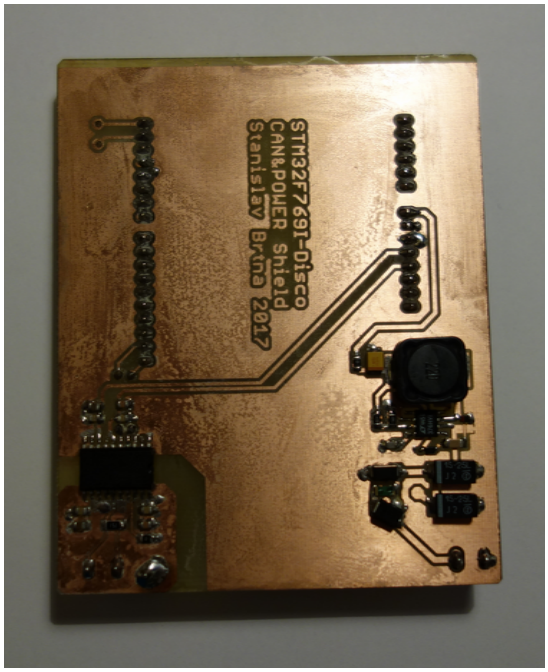
Ani tady se toho moc světoborného neděje, na rozšiřujícím modulu jsou dva integrované obvody, zapojené přesně podle katalogu. Prvním je integrovaný CAN budič s galvanickým oddělením, ADM3053. Druhým je spínaný zdroj realizovaný pomocí LT1376, s výstupním napětím 8 V. Kompletní schéma se nachází v příloze, v další kapitole věnované osazení a oživení se můžete podívat na obrázky osazené desky.

7 Osazení a oživení

DPS rozšiřovacího modulu byla vyrobena foto-cestou (filmová šablona je součástí příloh). Dále byl plošný spoj ručně odvrtán a osazen. Zdroj 8 V a CAN budič funguje dle předpokladů.



Obr. 7: Rozšiřující modul, vrchní strana.



Obr. 6: Rozšiřující modul, spodní strana.

8 Řídící software

Požadavky na řídící software jsou následující: Řídící software musí umožňovat zobrazení již specifikovaných ovládacích prvků. Těmto ovládacím prvkům by mělo být možno nastavit obecně definované CAN zpráv.

Jako rozhraní pro odesílání CAN zpráv byla zvolena knihovní funkce z HAL knihovny od STM32. K vykreslení grafických prvků a zpracování údajů z dotykové vrstvy byla použita dále popsaná knihovna GR2, vyvinutá v rámci této diplomové práce.

Nad knihovnou GR2 běží interpret skriptovacího jazyka SVS, který zpracovává konfigurační soubor s nastavením grafických prvků.

8.1 Knihovna GR2

Knihovna byla vytvořena s primárním účelem poskytnout funkční grafické rozhraní pro systémy s dotykovým displejem řízené mikrokontrolérem. Důraz byl při návrhu knihovny kladen na maximální jednoduchost jejího nasazení a použití.

8.1.1 Oblast využití

Primární oblastí využití knihovny GR2 je tvorba grafického rozhraní pro zařízení postavená na mikrokontrolérech. Konkrétně na mikrokontrolérech firmy STM. Ovšem zdrojové kódy jádra knihovny nejsou závislé na platformě. Knihovna je použitelná všude, kde

je třeba vykreslení základnách prvků grafického rozhraní a následná možnost zpracování uživatelského vstupu z dotykového rozhraní.

8.1.2 Vlastnosti

Knihovna GR2 umožňuje vykreslit na zobrazovací jednotku (většinou obrazovku) prvky grafického rozhraní. Dále umožňuje přijmout uživatelský vstup a zpracovat ho do podoby události konkrétního prvku grafického rozhraní, pro další zpracování uživatelskou obslužnou rutinou. Grafické prvky jsou vykresleny předem vygenerovaným fontem s variabilní šířkou znaku a podporou českých znaků v kódování UTF-8. Při kompilaci knihovny je možno nastavit maximální počet zobrazitelných grafických prvků a tím knihovnu přizpůsobit paměťovým požadavkům konkrétní aplikace. Knihovna nevyužívá dynamické alokace paměti a je nezávislá na dalším software.

Pro každou platformu je třeba specifických ovladačů: ovladače vykreslování a ovladače uživatelského vstupu. Ovladače jsou k dispozici pro platformu STM32 a PC (pomocí knihovny SDL2).

8.1.3 Příklad použití

Ukázka kódu jež vytvoří tlačítko s obsluhou.

```
//inicializace proměnných do kterých budou uloženy identifikátory
uint16_t screen;
uint16_t button;
uint16_t text;
//volání konstruktoru vrátí identifikátor
screen = pscg_add_screen();
//s identifikátorem poté můžeme dále pracovat
text = pscg_add_text(2, 0, 22, 3, "hello world", screen);
button = pscg_add_button(2, 3, 16, 6, "button", screen);
//vykreslení
LCD_setDrawArea(0, 0, LCD_W, LCD_H);
pscg_draw_screen(0, 0, 800, 479, screen, 1);

while(!exit){ //vykreslovací smyčka

  //překreslení
  pscg_draw_screen(0, 0, 800, 479, screen,0);

  //předání souřadnic dotyku a příslušné události
  pscg_touch_input(0, 0,800,479,touchX,touchY,ev,screen);
  if (pscg_get_event(button)==EV_RELEASED){
    //obslužný kód
    printf("button pressed\n");
    //vymazání události
    pscg_set_event(button,EV_NONE);
  }

}
```

Obr. 8: Takto se zobrazí předchozí kód

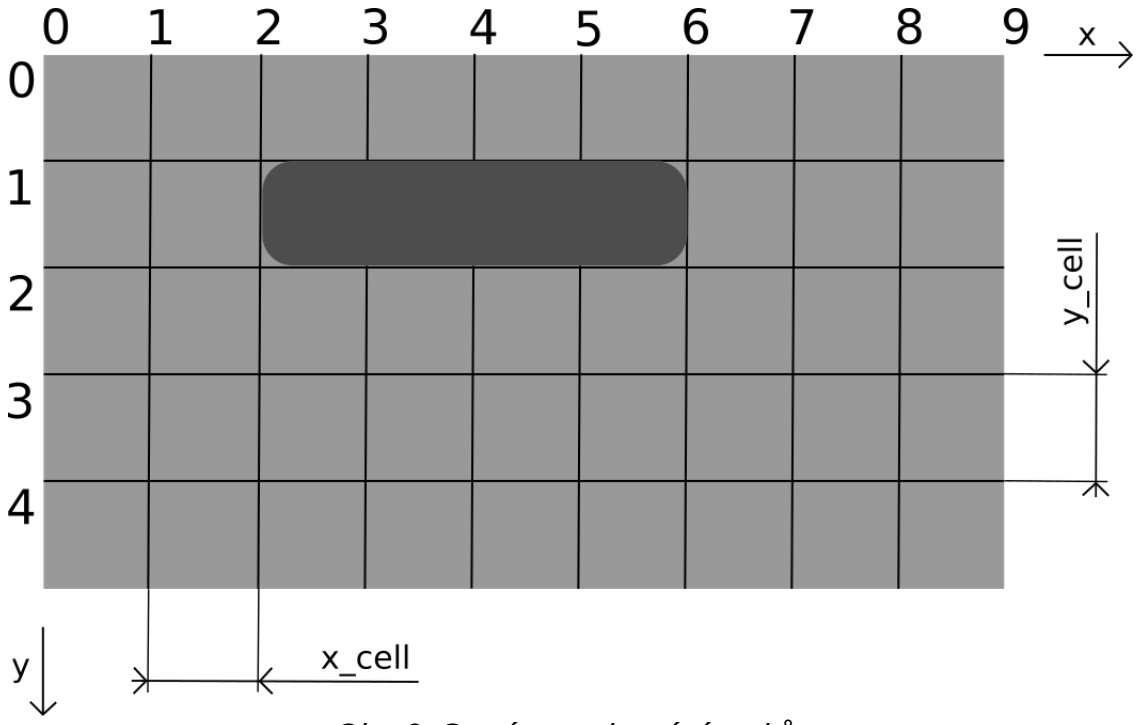
Popis a ukázka prvků knihovny

Primárním prvkem je obrazovka (*screen*), na obrazovku je možno umístit další prvky a definovat nebo měnit jejich parametry. Obrazovka je v knihovně GR2 prvek kontejnerového typu, do něhož je možno přidat další prvky, včetně dalších obrazovek. Druhým kontejnerovým prvkem GR2 je rámec, do rámce je možno vložit vždy jen jeden prvek typu obrazovka. Každý prvek má mírně odlišnou sadu parametrů, podle konkrétní potřeby.

Parametry prvku obrazovka

Parametry prvku typu obrazovka udávají do značné míry chování prvků na dané obrazovce. Zásadní jsou v tomto případě parametry *x_cell* a *y_cell*, udávající rozměr mřížky k níž jsou prvky obrazovky přichyceny. Globální parametry pozice prvku (*x1*, *y1*, *x2*, *y2*) jsou vázány právě na tuto mřížku.

Například: objekt na obrázku má parametry:
x1 = 2, *y1* = 1, *x2* = 6, *y2* = 2

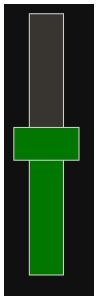
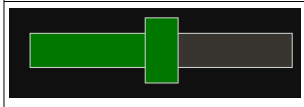


Obr. 9: Systém pozicování prvků.

Prvku je možno určit pozici v pixelech při velikosti mřížky:
x_cell = 1, *y_cell = 1*

Další parametry obrazovky jsou *x_scroll* a *y_scroll*. Tyto parametry udávají posun obsahu obrazovky oproti jejímu vykreslovacímu „oknu“. Modifikací těchto parametrů lze snadno posunout obsahem obrazovky v ose *x* a nebo *y*.

Poznámka: Ačkoli jsou parametry bez-znaménkového (unsigned) typu, je možno do nich po přetypování uložit (signed) záporné číslo, vzhledem ke způsobu nakládání s těmito parametry, dojde při vykreslení k zápornému posunu obsahu, přesně dle očekávání.

Název	Popis	Parametry	Ukázka
text	Textové pole.	*str* – text	
button	Tlačítko.	*str* – text tlačítka *event* – událost tlačítka	
slider_v	Vertikální posuvník.	*value* – hodnota posuvníku *param* – maximální hodnota posuvníku	
slider_h	Horizontální posuvník.		
progbar	Indikátor průběhu.	*value* – hodnota průběhu *param* – maximální hodnota průběhu	

Tabulka 1: Grafické prvky knihovny GR2

Globální parametry prvků:

Parametr	Popis
x1	x levého horního rohu prvku [x mřížky obrazovky]
y1	y levého horního rohu prvku [y mřížky obrazovky]
x2	x pravého spodního rohu prvku [x mřížky obrazovky]
y2	y pravého spodního rohu prvku [y mřížky obrazovky]
visible	určuje, zda bude prvek vykreslen
modified	upozorní vykreslovací funkci o potřebě překreslení prvku (většinou se nastaví automaticky)
screen	identifikátor (id) obrazovky, kam prvek přísluší

Tabulka 2: Globální parametry prvků

Parametry rámce

Rámec má všechny parametry jako ostatní elementy, parametr *value* udává, jaká obrazovka se bude na jeho místě vykreslovat. Snadnou modifikací parametru *value* lze pohodlně přepínat pod-obrazovky.

8.1.4 Popis aplikačního rozhraní (lcd_basics)

Nejprv budou popsány potřebné funkce vykreslovací knihovny *lcd_basics*.

Název	Popis
Název funkce	LCD_init
Prototyp funkce	uint8_t LCD_init(uint16_t x_size, uint16_t y_size, lcdOrientationType orient);
Chování	Inicializuje ovladač zobrazovací jednotky a nastaví základní parametry zobrazovací jednotky
Parametry	x_size – horizontální rozlišení zobrazovací jednotky [px] y_size – vertikální rozlišení zobrazovací jednotky [px] orient – orientace displeje
Poznámka	Nutno inicializovat LCD před použitím knihovny, funkce vyžaduje funkční ovladač zobrazovací jednotky.

Tabulka 3: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_setDrawArea
Prototyp funkce	void LCD_setDrawArea(uint16_t x1,uint16_t y1,uint16_t x2,uint16_t y2);
Chování	Nastaví oblast pro vykreslování dle parametrů
Parametry	x1 – x pravého horního rohu [px] y1 – y pravého horního rohu [px] x2 – x levého spodního rohu [px] y2 – y levého spodního rohu [px]
Poznámka	Před každým voláním vykreslovací funkce grafického rozhraní (*pscg_draw_screen*) je nutno ručně nastavit vykreslovací oblast.

Tabulka 4: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_setSubDrawArea
Prototyp funkce	void LCD_setSubDrawArea(uint16_t x1,uint16_t y1,uint16_t x2,uint16_t y2);
Chování	Omezí předem nastavenou vykreslovací oblast dle předem daných souřadnic.
Parametry	x1 – x pravého horního rohu [px] y1 – y pravého horního rohu [px]

	x2 – x levého spodního rohu [px] y2 – y levého spodního rohu [px]
Poznámka	

Tabulka 5: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	LCD_set_orientation
Prototyp funkce	void LCD_set_orientation(lcdOrientationType orient);
Chování	Nastaví orientaci LCD
Parametry	orient – orientace
Poznámka	Po nastavení nutno znovu překreslit

Tabulka 6: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	LCD_rotr_x
Prototyp funkce	uint16_t LCD_rotr_x(uint16_t x1, uint16_t y1);
Chování	Vrací souřadnici x dle aktuální orientace displeje, na základě aktuální orientace obrazovky.
Parametry	x1 – vstupní souřadnice x [px] y1 – vstupní souřadnice y [px]

Tabulka 7: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	LCD_rotr_y
Prototyp funkce	uint16_t LCD_rotr_y(uint16_t x1, uint16_t y1);
Chování	Vrací souřadnici y dle aktuální orientace displeje, na základě aktuální orientace obrazovky.
Parametry	x1 – vstupní souřadnice x [px] y1 – vstupní souřadnice y [px]

Tabulka 8: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_get_x_size
Prototyp funkce	uint16_t LCD_get_x_size();
Chování	Vrací šířku displeje

Tabulka 9: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_get_y_size
Prototyp funkce	uint16_t LCD_get_y_size();
Chování	Vrací výšku displeje

Tabulka 10: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_getDrawArea
Prototyp funkce	void LCD_getDrawArea(LCD_drawArea *a);
Chování	Předá souřadnice vykreslovací oblasti
Parametry	*a – ukazatel na strukturu souřadnic kam budou zapsána výstupní data
Poznámka	Toto je jediná možnost jak získat souřadnice vykreslovací oblasti.

Tabulka 11: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_setDrawAreaS
Prototyp funkce	void LCD_setDrawAreaS(LCD_drawArea *a);
Chování	Nastaví vykreslovací oblast dle parametrů
Parametry	*a – ukazatel na strukturu souřadnic
Poznámka	Pro rychlé a pohodlné nastavení vykreslovací oblasti z výstupu funkce LCD_getDrawArea.

Tabulka 12: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_MixColor
Prototyp funkce	uint16_t LCD_MixColor(uint8_t r, uint8_t g,uint8_t b);
Chování	Převádí RGB888 barvu na RGB565
Parametry	r – hodnota červené g – hodnota zelené b – hodnota modré

Tabulka 13: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_DrawPoint
Prototyp funkce	void LCD_DrawPoint(uint16_t x,uint16_t y, uint16_t color);
Chování	Vykreslí bod danou barvou
Parametry	x,y – souřadnice [px] color – 16bit barva

Tabulka 14: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_Fill
Prototyp funkce	void LCD_Fill(uint16_t color);
Chování	Vyplní LCD danou barvou
Parametry	color – 16bit barva

Tabulka 15: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_DrawLine
Prototyp funkce	void LCD_DrawLine(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t color);
Chování	Vykreslí čáru
Parametry	x1, y1, x2, y2 – souřadnice [px] color – 16bit barva

Tabulka 16: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_DrawRectangle
Prototyp funkce	void LCD_DrawRectangle(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t color);
Chování	Vykreslí obrys obdélníku.
Parametry	x1, y1, x2, y2 – souřadnice [px] color – 16bit barva

Tabulka 17: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_FillRect
Prototyp funkce	void LCD_FillRect(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2,uint16_t Color);
Chování	Vykreslí vyplněný obdélník.
Parametry	x1, y1, x2, y2 – souřadnice [px] color – 16bit barva

Tabulka 18: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_DrawCircle
Prototyp funkce	void LCD_DrawCircle(uint16_t x0, uint16_t y0, uint16_t radius, uint16_t color);
Chování	Vykreslí kružnici.
Parametry	x0, y0 – souřadnice středu [px] radius – poloměr [px] color – 16bit barva

Tabulka 19: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_FillCircle
Prototyp funkce	void LCD_FillCircle(uint16_t x0, uint16_t y0, uint16_t radius, uint16_t color);
Chování	Vykreslí kruh.
Parametry	x0, y0 – souřadnice středu [px] radius – poloměr [px] color – 16bit barva

Tabulka 20: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_DrawText_ext
Prototyp funkce	void LCD_DrawText_ext(uint16_t x, uint16_t y, uint16_t color, uint8_t *text);
Chování	Vykreslí text.
Parametry	x, y – souřadnice levého horního rohu textu [px] color – 16bit barva text – nulou ukončený řetězec
Poznámka	Vykresluje fontem s variabilní šířkou znaku. Umožňuje vykreslení českých znaků kódovaných v UTF8.

Tabulka 21: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_Text_Get_Width
Prototyp funkce	uint16_t LCD_Text_Get_Width(uint8_t *text, uint16_t count);
Chování	Vrací šířku textu po vykreslení aktuálním fontem.
Parametry	text – nulou ukončený řetězec count – počet znaků pro které se bude počítat šířka
Poznámka	Pokud count je rovno nule, tak bude vrácena šířka celého řetězce, až do ukončovacího znaku.

Tabulka 22: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_Draw_Get_Font_Height
Prototyp funkce	uint16_t LCD_Draw_Get_Font_Height();
Chování	Vrací výšku textu po vykreslení aktuálním fontem.
Parametry	text – nulou ukončený řetězec count – počet znaků pro které se bude počítat výška
Návratová hodnota	Pokud count je rovno nule, tak bude vrácena výška celého řetězce, až do ukončovacího znaku.

Tabulka 23: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_Draw_Get_Font_Width
Prototyp funkce	uint16_t LCD_Draw_Get_Font_Width();
Chování	Vrátí šířku aktuálního fontu.

Tabulka 24: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_Text_Draw_Cursor
Prototyp funkce	void LCD_Text_Draw_Cursor(uint16_t x, uint16_t y, uint8_t *text, uint16_t pos, uint16_t Color);
Chování	Vykreslí kurzor na dané pozici v textu.
Parametry	x, y – souřadnice levého horního rohu textu [px] pos – pozice kurzoru color – 16bit barva text – nulou ukončený řetězec

Tabulka 25: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_Set_Sys_Font
Prototyp funkce	void LCD_Set_Sys_Font(uint8_t size);
Chování	Nastaví systémový font na font dané velikosti
Parametry	size – velikost
Poznámka	Velikosti fontů jsou dány při kompilaci. Přidání dalších fontů není triviální.

Tabulka 26: Popis funkce (viz tabulka)

Název	Popis
Název funkce	LCD_Get_Font_Size
Prototyp funkce	uint8_t LCD_Get_Font_Size();
Chování	Vrací velikost aktuálního fontu.

Tabulka 27: Popis funkce (viz tabulka)

Rozhraní ovladače zobrazovací jednotky

Tyto funkce je třeba reimplementovat pro každou zobrazovací jednotku.

Název	Popis
Název funkce	lcd_hw_Draw_Point
Prototyp funkce	void lcd_hw_Draw_Point(uint16_t color);
Chování	Vykreslí bod dle aktuálních souřadnic daných lcd_hw_set_xy
Parametry	color – 16bit barva

Tabulka 28: Popis funkce (viz tabulka)

Název	Popis
Název funkce	lcd_hw_set_xy
Prototyp funkce	void lcd_hw_set_xy(uint16_t px1, uint16_t py1,uint16_t px2,uint16_t py2);
Chování	Nastaví vykreslovací rozsah na hardware zobrazovací jednotky.
Parametry	px1, py1, px2, py2 – souřadnice vykreslovací oblasti [px]

Tabulka 29: Popis funkce (viz tabulka)

Název	Popis
Název funkce	lcd_hw_init
Prototyp funkce	uint8_t lcd_hw_init();
Chování	Provádí inicializaci hardware zobrazovací jednotky.

Tabulka 30: Popis funkce (viz tabulka)

Příklad ovladače zobrazovací jednotky pro SDL2. Jedná se pouze o kód funkcí ovladače, předpokládá inicializovaný renderer. Plně funkční projekt je přiložen, z hlediska ovladače pro SDL2 jsou zajímavé soubory *svs-sdl.c* a *GR2/lcd_io.c*.

```
uint16_t hx1, hy1, hx2, hy2, cx, cy; //pomocné globální proměnné

uint8_t lcd_hw_init(){ //v tomto případě není nutno inicializovat hw
  return 0;
}

void lcd_hw_set_xy(uint16_t px1, uint16_t py1,uint16_t px2,uint16_t py2){
  hx1=px1;
  hx2=px2;
  hy1=py1;
  hy2=py2;
  cx=0;
  cy=0;
  return;
}

//vykreslovací funkce s převodem barvy z R5-G6-B5 na R8-G8-B8
void ExtDrawPoint(int x, int y, uint16_t color){
 uint8_t r,g,b;
  r= (uint8_t)(((float)((color>>11)&0x1F)/32)*256);
  g= (uint8_t)(((float)(((color&0x07E0)>>5)&0x3F)/64)*256);
  b= (uint8_t)(((float)(color&0x1F)/32)*256);

  SDL_SetRenderDrawColor( gRenderer,r,g,b, 0xFF );
  SDL_RenderDrawPoint(gRenderer, x, y);
}

//emulace chování lcd řadiče
void lcd_hw_Draw_Point(uint16_t color){
  ExtDrawPoint(cx+hx1,cy+hy1, color);
  if (cx<(hx2-hx1)){
    cx++;
  }else{
    cx=0;
    if (cy<(hy2-hy1)){
      cy++;
    }else{
      cy=0;
    }
  }
  return;
}
```

Struktury a typy modulu lcd_basics

Název	lcdOrientationType
Popis	Enumerační typ pro orientaci LCD
Hodnoty	OR_NORMAL OR_ROT_LEFT OR_ROT_RIGHT OR_UPSIDE_DOWN

Tabulka 31: Popis struktury lcdOrientationType

Název	LCD_drawArea
Popis	Typ pro předávání souřadnic vykreslovací oblasti
Prvky	uint16_t x1; uint16_t y1; uint16_t x2; uint16_t y2;

Tabulka 32: Popis struktury LCD_drawArea

8.1.5 Popis aplikačního rozhraní (pscg)

Popis aplikačního rozhraní modulu *pscg* knihovny GR2.

Vykreslovací a vstupní funkce

Název	Popis
Název funkce	pscg_touch_input
Prototyp funkce	uint8_t pscg_touch_input(uint16_t x1, uint16_t y1,uint16_t x2, uint16_t y2,uint16_t touch_x,uint16_t touch_y,gr2EventType event, uint16_t screen);
Chování	Zpracuje dotyk v obrazovce a nastaví příslušnou událost konkrétnímu grafickému prvku. **Návratová hodnota:** **0** pokud dotyk nebyl zpracován, **1** pokud byl zpracován **2** pokud byl zpracován aktivním textovým polem.
Parametry	Souřadnice na kterých je vykreslena obrazovka, jejíž dotyk chceme zpracovat: x1 – x levého horního rohu obrazovky [px] y1 – y levého horního rohu obrazovky [px] x2 – x pravého spodního rohu obrazovky [px] y2 – y pravého spodního rohu obrazovky [px] touch_x – souřadnice x dotyku touch_y – souřadnice y dotyku event – událost dotyku screen – obrazovka pro kterou se má událost zpracovat
Poznámka	Funkce by se měla zavolat se shodnými parametry vykreslovacích souřadnic a vykreslené obrazovky jako funkce pscg_draw_screen.

Tabulka 33: Popis funkce pscg_touch_input

Název	Popis
Název funkce	pscg_draw_screen
Prototyp funkce	void pscg_draw_screen(uint16_t x1, uint16_t y1,uint16_t x2, uint16_t y2, uint16_t screen, uint8_t all);
Chování	Vykreslí danou obrazovku na základě daných souřadnic.
Parametry	x1 – x levého horního rohu obrazovky [px] y1 – y levého horního rohu obrazovky [px] x2 – x pravého spodního rohu obrazovky [px]

	y2 – y pravého spodního rohu obrazovky [px] screen – obrazovka pro kterou se má událost zpracovat all – 1 pokud má dojít k úplnému překreslení, jinak 0
Poznámka	Vykreslí obrazovku včetně jejích prvků (včetně pod-obrazovek a jejich prvků). Ve většině případů stačí nepřekreslovat obrazovku celou (parametr *all*), knihovna sama překreslí jen modifikované prvky rozhraní

Tabulka 34: Popis funkce pscg_draw_screen

Název	Popis
Název funkce	pscg_draw_end
Prototyp funkce	void pscg_draw_end();
Chování	Interní procedura knihovny, kterou je nutno zavolat pokaždé po dokončení vykreslování.
Poznámka	

Tabulka 35: Popis funkce pscg_draw_end

Konstruktory prvků

Konstruktor je funkce jejíž činností je inicializovat nový grafický prvek, nastavit mu hodnoty parametrů a předat jeho identifikátor (id). Identifikátor slouží pro další práci s prvkem grafického rozhraní, jako je čtení jeho událostí nebo nastavování dalších parametrů. Grafické rozhraní je navrženo tak, aby byla práce s prvky maximálně unifikovaná.

Systém souřadnic v pscg:

Grafické prvky jsou v pscg zarovnány v mřížce, jejíž velikost je dána parametry obrazovky. Parametry mřížky lze pro každou obrazovku modifikovat změnou parametrů *x_cell* a *y_cell*.

Název	Popis
Název funkce	pscg_add_screen
Prototyp funkce	uint16_t pscg_add_screen();
Chování	Konstruktor obrazovky. Vytvoří novou obrazovku. **Návratová hodnota:** **id** nové obrazovky

Tabulka 36: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_add_button
Prototyp funkce	uint16_t pscg_add_button(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint8_t *str, uint16_t screen);
Chování	Konstruktor tlačítka. Vytvoří nové tlačítko daných parametrů. **Návratová hodnota:** **id** nového tlačítka
Parametry	x1 – x levého horního rohu prvku [x mřížky obrazovky] y1 – y levého horního rohu prvku [y mřížky obrazovky] x2 – x pravého spodního rohu prvku [x mřížky obrazovky] y2 – y pravého spodního rohu prvku [y mřížky obrazovky] str – popisek tlačítka, ukazatel na nulou ukončený řetězec screen – obrazovka, kam bude prvek přidán
Poznámka	

Tabulka 37: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_add_slider_v
Prototyp funkce	uint16_t pscg_add_slider_v(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t param, uint16_t value, uint16_t screen);
Chování	Konstruktor vertikálního posuvníku. Vytvoří nový vertikální posuvník daných parametrů. **Návratová hodnota:** **id** nového posuvníku
Parametry	x1 – x levého horního rohu prvku [x mřížky obrazovky] y1 – y levého horního rohu prvku [y mřížky obrazovky] x2 – x pravého spodního rohu prvku [x mřížky obrazovky] y2 – y pravého spodního rohu prvku [y mřížky obrazovky] param – maximální hodnota posuvníku value – aktuální hodnota posuvníku screen – obrazovka, kam bude prvek přidán
Poznámka	

Tabulka 38: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_add_slider_h
Prototyp funkce	uint16_t pscg_add_slider_h(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t param, uint16_t value, uint16_t screen);
Chování	Konstruktor horizontálního posuvníku. Vytvoří nový horizontální posuvník daných parametrů. **Návratová hodnota:** **id** nového posuvníku
Parametry	x1 – x levého horního rohu prvku [x mřížky obrazovky] y1 – y levého horního rohu prvku [y mřížky obrazovky] x2 – x pravého spodního rohu prvku [x mřížky obrazovky] y2 – y pravého spodního rohu prvku [y mřížky obrazovky] param – maximální hodnota posuvníku value – aktuální hodnota posuvníku screen – obrazovka, kam bude prvek přidán
Poznámka	

Tabulka 39: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_add_text
Prototyp funkce	uint16_t pscg_add_text(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint8_t *str, uint16_t screen);
Chování	Konstruktor textového pole. Vytvoří nové textové pole daných parametrů. **Návratová hodnota:** **id** nového textového pole
Parametry	x1 – x levého horního rohu prvku [x mřížky obrazovky] y1 – y levého horního rohu prvku [y mřížky obrazovky] x2 – x pravého spodního rohu prvku [x mřížky obrazovky] y2 – y pravého spodního rohu prvku [y mřížky obrazovky] str – text textového pole, ukazatel na nulou ukončený řetězec screen – obrazovka, kam bude prvek přidán
Poznámka	

Tabulka 40: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_add_progbar_v
Prototyp funkce	uint16_t pscg_add_progbar_v(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t param, uint16_t value, uint16_t screen);
Chování	Konstruktor indikátoru průběhu. Vytvoří nový indikátor průběhu daných parametrů. **Návratová hodnota:** **id** nového indikátoru průběhu
Parametry	x1 – x levého horního rohu prvku [x mřížky obrazovky] y1 – y levého horního rohu prvku [y mřížky obrazovky] x2 – x pravého spodního rohu prvku [x mřížky obrazovky] y2 – y pravého spodního rohu prvku [y mřížky obrazovky] param – maximální hodnota indikátoru value – aktuální hodnota indikátoru screen – obrazovka, kam bude prvek přidán
Poznámka	Indikátor se vykreslí buďto horizontální a nebo vertikální, záleží na velikosti prvku, jestli je širší než vyšší a nebo opačně.

Tabulka 41: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_add_icon
Prototyp funkce	uint16_t pscg_add_icon(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint8_t *str, uint8_t *str2, uint16_t screen);
Chování	Konstruktor ikony. Vytvoří novou ikonu daných parametrů. Nutno při překladu povolit *PPM_SUPPORT_ENABLED* a poskytnout funkci pro vykreslení *.ppm* souborů. Soubory *ppm* nutno dodat 64x64 px nebo 32x32 px. **Návratová hodnota:** **id** nového textového pole
Parametry	Obdobně jako *pscg_add_button*. str2 – název souboru ikony, ukazatel na nulou ukončený řetězec
Poznámka	**Ve výchozím stavu zakázáno.**

Tabulka 42: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_add_frame
Prototyp funkce	uint16_t pscg_add_frame(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t val, uint16_t screen);
Chování	Konstruktor rámce. Vytvoří nový rámec. **Návratová hodnota:** **id** nového rámce
Parametry	x1 – x levého horního rohu prvku [x mřížky obrazovky] y1 – y levého horního rohu prvku [y mřížky obrazovky] x2 – x pravého spodního rohu prvku [x mřížky obrazovky] y2 – y pravého spodního rohu prvku [y mřížky obrazovky] val – id obrazovky obsahu rámce screen – obrazovka, kam bude prvek přidán

Tabulka 43: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_add_cbutton
Prototyp funkce	uint16_t pscg_add_cbutton(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint8_t *str, uint16_t screen);
Chování	Konstruktor tlačítka s definovanou barvou. **Návratová hodnota:** **id** nového barevného tlačítka
Parametry	x1 – x levého horního rohu prvku [x mřížky obrazovky] y1 – y levého horního rohu prvku [y mřížky obrazovky] x2 – x pravého spodního rohu prvku [x mřížky obrazovky] y2 – y pravého spodního rohu prvku [y mřížky obrazovky] str – popisek tlačítka, ukazatel na nulou ukončený řetězec screen – obrazovka, kam bude prvek přidán
Poznámka	Parametr *value* určuje barvu.

Tabulka 44: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_get_value
Prototyp funkce	uint16_t pscg_get_value(uint16_t id);
Chování	Vrátí parametr *value*. **Návratová hodnota:** **value**
Parametry	id – identifikátor prvku jehož parametr chceme obdržet

Tabulka 45: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_set_value
Prototyp funkce	void pscg_set_value(uint16_t id, uint16_t val);
Chování	Umožňuje nastavit parametr *value*.
Parametry	id – identifikátor prvku jehož parametr chceme nastavit val – nová hodnota parametru

Tabulka 46: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_set_screen
Prototyp funkce	void pscg_set_screen(uint16_t id, uint16_t val);
Chování	Umožňuje nastavit parametr *screen*.
Parametry	id – identifikátor prvku jehož parametr chceme nastavit val – nová hodnota parametru

Tabulka 47: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_get_param
Prototyp funkce	uint16_t pscg_get_param(uint16_t id);
Chování	Umožňuje nastavit parametr *param*.
Parametry	id – identifikátor prvku jehož parametr chceme obdržet

Tabulka 48: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_set_param
Prototyp funkce	void pscg_set_param(uint16_t id, uint16_t val);
Chování	Umožňuje nastavit parametr *param*.
Parametry	id – identifikátor prvku jehož parametr chceme nastavit val – nová hodnota parametru

Tabulka 49: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_get_x1
Prototyp funkce	uint16_t pscg_get_x1(uint16_t id);
Chování	Vrátí parametr *x1*. **Návratová hodnota:** **x1**
Parametry	id – identifikátor prvku jehož parametr chceme obdržet

Tabulka 50: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_get_x2
Prototyp funkce	uint16_t pscg_get_x2(uint16_t id);
Chování	Vrátí parametr *x2*. **Návratová hodnota:** **x2**
Parametry	id – identifikátor prvku jehož parametr chceme obdržet

Tabulka 51: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_get_y1
Prototyp funkce	uint16_t pscg_get_y1(uint16_t id);
Chování	Vrátí parametr *y1*. **Návratová hodnota:** **y1**
Parametry	id – identifikátor prvku jehož parametr chceme obdržet

Tabulka 52: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_get_y2
Prototyp funkce	uint16_t pscg_get_y2(uint16_t id);
Chování	Vrátí parametr *y2*. **Návratová hodnota:** **y2**
Parametry	id – identifikátor prvku jehož parametr chceme obdržet

Tabulka 53: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_set_x1
Prototyp funkce	void pscg_set_x1(uint16_t id, uint16_t val);
Chování	Umožňuje nastavit parametr *x1*.
Parametry	id – identifikátor prvku jehož parametr chceme nastavit val – nová hodnota parametru

Tabulka 54: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_set_x2
Prototyp funkce	void pscg_set_x2(uint16_t id, uint16_t val);
Chování	Umožňuje nastavit parametr *x2*.
Parametry	id – identifikátor prvku jehož parametr chceme nastavit val – nová hodnota parametru

Tabulka 55: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_set_y1
Prototyp funkce	void pscg_set_y1(uint16_t id, uint16_t val);
Chování	Umožňuje nastavit parametr *y1*.
Parametry	id – identifikátor prvku jehož parametr chceme nastavit val – nová hodnota parametru

Tabulka 56: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_set_y2
Prototyp funkce	void pscg_set_y2(uint16_t id, uint16_t val);
Chování	Umožňuje nastavit parametr *y2*.
Parametry	id – identifikátor prvku jehož parametr chceme nastavit val – nová hodnota parametru

Tabulka 57: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_set_str
Prototyp funkce	void pscg_set_str(uint16_t id, uint8_t *str);
Chování	Umožňuje nastavit parametr *txt*.
Parametry	id – identifikátor prvku jehož parametr chceme nastavit str – nová hodnota parametru, ukazatel na nulou ukončený řetězec

Tabulka 58: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_get_str
Prototyp funkce	uint8_t * pscg_get_str(uint16_t id);
Chování	Umožňuje získat parametr *txt*. **Návratová hodnota:** **txt** – hodnota parametru *txt*, ukazatel na nulou ukončený řetězec
Parametry	id – identifikátor prvku jehož parametr chceme obdržet

Tabulka 59: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_set_modified
Prototyp funkce	void pscg_set_modified(uint16_t id);
Chování	Nastaví prvku příznak *modified*, čímž vyvolá jeho překreslení při dalším průchodu vykreslovací smyčkou.
Parametry	id – identifikátor prvku jehož příznak chceme nastavit

Tabulka 60: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_set_visible
Prototyp funkce	void pscg_set_visible(uint16_t id, uint16_t vis);
Chování	Umožňuje nastavit parametr *visible*.
Parametry	id – identifikátor prvku jehož příznak chceme nastavit vis – nová hodnota parametru *visible*

Tabulka 61: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_text_set_size
Prototyp funkce	void pscg_text_set_size(uint16_t id, uint16_t size);
Chování	Umožňuje nastavit velikost textu prvku.
Parametry	id – identifikátor prvku jehož příznak chceme nastavit size – nová hodnota velikosti textu, viz *LCD_Set_Sys_Font*
Poznámka	Funguje jen u prvku pscg_text.

Tabulka 62: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_text_deactivate
Prototyp funkce	void pscg_text_deactivate();
Chování	Deaktivuje aktivní textové pole v režimu editace.
Poznámka	

Tabulka 63: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_text_deactivate
Prototyp funkce	void pscg_text_deactivate();
Chování	Deaktivuje aktivní textové pole v režimu editace.
Poznámka	

Tabulka 64: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_bread_apply_butter
Prototyp funkce	food pscg_bread_apply_butter(uint16_t id, uint16_t val);
Chování	Namaže chleba máslem (příznak *butter*)
Parametry	id – identifikátor prvku typu pscg_*chleba* jehož příznak chceme nastavit val – nová hodnota příznaku *butter*

Tabulka 65: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_text_get_editable
Prototyp funkce	uint8_t pscg_text_get_editable(uint16_t id);
Chování	Umožňuje získat hodnotu příznaku *editable* textového pole. **Návratová hodnota:** **0** – příznak nenastaven **1** – příznak nastaven
Parametry	id – identifikátor prvku typu pscg_*text* jehož příznak chceme nastavit

Tabulka 66: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_set_x1y1x2y2
Prototyp funkce	void pscg_set_x1y1x2y2(uint16_t id, uint16_t x1,uint16_t y1, uint16_t x2,uint16_t y2);
Chování	Umožňuje prvku nastavit parametry x1, x2, y1 a y1.
Parametry	id – identifikátor prvku jehož parametr chceme nastavit x1 – nová hodnota parametru *x1* y1 – nová hodnota parametru *y1* x2 – nová hodnota parametru *x2* y2 – nová hodnota parametru *y2*

Tabulka 67: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_get_event
Prototyp funkce	gr2EventType pscg_get_event(uint16_t id);
Chování	Umožňuje získat hodnotu parametru *event.* **Návratová hodnota:** **event –** událost typu *gr2EventType*
Parametry	id – identifikátor prvku jehož parametr chceme obdržet

Tabulka 68: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_set_event
Prototyp funkce	void pscg_set_event(uint16_t id, gr2EventType val);
Chování	Umožňuje prvku nastavit parametr *event.*
Parametry	id – identifikátor prvku jehož parametr chceme nastavit val – nová hodnota parametru (typu gr2EventType)
Poznámka	Po obsluze události (přečtením parametru event), je nutno touto funkcí ručně nastavit parametr na hodnotu *EV_NONE*.

Tabulka 69: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_clear_screen_ev
Prototyp funkce	void pscg_clear_screen_ev(uint16_t id);
Chování	Umožňuje nastavit všem prvkům dané obrazovky parametr *event* na *EV_NONE*.
Parametry	id – identifikátor **obrazovky**
Poznámka	Nahrazením jednoho volání *pscg_clear_screen_ev* více voláními *pscg_set_event(id, EV_NONE)* lze dosáhnout rychlejšího vykonání kódu.

Tabulka 70: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_get_xscroll
Prototyp funkce	uint16_t pscg_get_xscroll(uint16_t id);
Chování	**Návratová hodnota:** **xscroll –** parametr *xscroll* [px]
Parametry	id – identifikátor **obrazovky**
Poznámka	Tímto parametrem disponují pouze obrazovky.

Tabulka 71: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_get_yscroll
Prototyp funkce	uint16_t pscg_get_yscroll(uint16_t id);
Chování	Umožňuje obdržet parametr y*scroll*. **Návratová hodnota:** **yscroll –** parametr y*scroll* [px]
Parametry	id – identifikátor **obrazovky**
Poznámka	Tímto parametrem disponují pouze obrazovky.

Tabulka 72: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_set_xscroll
Prototyp funkce	void pscg_set_xscroll(uint16_t id, uint16_t val);
Chování	Umožňuje nastavit parametr *xscroll*.
Parametry	id – identifikátor **obrazovky** val – nová hodnota parametru [px]
Poznámka	Tímto parametrem disponují pouze obrazovky.

Tabulka 73: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_set_yscroll
Prototyp funkce	void pscg_set_yscroll(uint16_t id, uint16_t val);
Chování	Umožňuje nastavit parametr *yscroll*.
Parametry	id – identifikátor **obrazovky** val – nová hodnota parametru [px]
Poznámka	Tímto parametrem disponují pouze obrazovky.

Tabulka 74: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_set_x_cell
Prototyp funkce	void pscg_set_x_cell(uint16_t id, uint16_t val);
Chování	Umožňuje nastavit parametr *x_cell*.
Parametry	id – identifikátor **obrazovky** val – nová hodnota parametru [px]
Poznámka	Tímto parametrem disponují pouze obrazovky.

Tabulka 75: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_set_y_cell
Prototyp funkce	void pscg_set_y_cell(uint16_t id, uint16_t val);
Chování	Umožňuje nastavit parametr y_*cell*.
Parametry	id – identifikátor **obrazovky** val – nová hodnota parametru [px]
Poznámka	Tímto parametrem disponují pouze obrazovky.

Tabulka 76: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_get_x_cell
Prototyp funkce	uint16_t pscg_get_x_cell(uint16_t id);
Chování	Umožňuje obdržet parametr *x_cell.* **Návratová hodnota:** **x_cell –** hodnota parametru *x_cell*. [px]
Parametry	id – identifikátor **obrazovky**
Poznámka	Tímto parametrem disponují pouze obrazovky.

Tabulka 77: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_get_y_cell
Prototyp funkce	uint16_t pscg_get_y_cell(uint16_t id);
Chování	Umožňuje obdržet parametr *y_cell.* **Návratová hodnota:** **y_cell –** hodnota parametru *y_cell*. [px]
Parametry	id – identifikátor **obrazovky**
Poznámka	Tímto parametrem disponují pouze obrazovky.

Tabulka 78: Popis funkce (viz tabulka)

Funkce pro práci s globálním nastavením barev

Název	Popis
Název funkce	pscg_set_border_color
Prototyp funkce	void pscg_set_border_color(uint16_t col);
Chování	Umožňuje nastavit globální barvu *border_color*.
Parametry	col – nová barva [RGB565]
Poznámka	Touto barvou jsou vykreslovány okraje prvků.

Tabulka 79: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_get_border_color
Prototyp funkce	uint16_t pscg_get_border_color();
Chování	Umožňuje obdržet globální barvu *border_color*. **Návratová hodnota:** **border_color** – hodnota globální barvy *border_color*. [RGB565]
Poznámka	Touto barvou jsou vykreslovány okraje prvků.

Tabulka 80: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_set_text_color
Prototyp funkce	void pscg_set_text_color(uint16_t col);
Chování	Umožňuje nastavit globální barvu *text_color*.
Parametry	col – nová barva [RGB565]
Poznámka	Touto barvou je vykreslován text.

Tabulka 81: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_get_text_color
Prototyp funkce	uint16_t pscg_get_text_color();
Chování	Umožňuje obdržet globální barvu *text_color*. **Návratová hodnota:** **text_color** – hodnota globální barvy *text_color*. [RGB565]
Poznámka	Touto barvou je vykreslován text.

Tabulka 82: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_set_background_color
Prototyp funkce	void pscg_set_background_color(uint16_t col);
Chování	Umožňuje nastavit globální barvu *background_color*.
Parametry	col – nová barva [RGB565]
Poznámka	Touto barvou je vykresleno pozadí obrazovek.

Tabulka 83: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_get_background_color
Prototyp funkce	uint16_t pscg_get_background_color();
Chování	Umožňuje obdržet globální barvu *background_color*. **Návratová hodnota:** **background_color** – hodnota globální barvy *background_color*. [RGB565]
Poznámka	Touto barvou je vykresleno pozadí obrazovek.

Tabulka 84: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_set_fill_color
Prototyp funkce	void pscg_set_fill_color(uint16_t col);
Chování	Umožňuje nastavit globální barvu *fill_color*.
Parametry	col – nová barva [RGB565]
Poznámka	Touto barvou je vykresleno pozadí prvků (tlačítek atp.).

Tabulka 85: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_get_fill_color
Prototyp funkce	uint16_t pscg_get_fill_color();
Chování	Umožňuje obdržet globální barvu *fill_color*. **Návratová hodnota:** **fill_color** – hodnota globální barvy *fill_color*. [RGB565]
Poznámka	Touto barvou je vykresleno pozadí prvků (tlačítek atp.).

Tabulka 86: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_set_active_color
Prototyp funkce	void pscg_set_active_color(uint16_t col);
Chování	Umožňuje nastavit globální barvu *active_color*.
Parametry	col – nová barva [RGB565]
Poznámka	Touto barvou jsou vykresleny aktivní prvky.

Tabulka 87: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	
Prototyp funkce	uint16_t pscg_get_active_color();
Chování	Umožňuje obdržet globální barvu *active_color*. **Návratová hodnota:** **active_color** – hodnota globální barvy *active_color*. [RGB565]
Poznámka	Touto barvou jsou vykresleny aktivní prvky.

Tabulka 88: Popis funkce (viz tabulka)

Název	**Popis**
Název funkce	pscg_reset_all
Prototyp funkce	void pscg_reset_all();
Chování	Vymaže všechny grafické prvky.
Poznámka	Knihovnu není nutno před prvním použitím resetovat.

Tabulka 89: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_destroy
Prototyp funkce	void pscg_destroy(uint16_t id);
Chování	Odstraní prvek daného id.
Parametry	id – identifikátor prvku
Poznámka	Pro odstranění prvku typu obrazovky použijte *pscg_destroy_screen.*

Tabulka 90: Popis funkce (viz tabulka)

Název	Popis
Název funkce	pscg_destroy_screen
Prototyp funkce	void pscg_destroy_screen(uint16_t id);
Chování	Odstraní screen a všechny jeho podřazené prvky.
Parametry	id – identifikátor obrazovky

Tabulka 91: Popis funkce (viz tabulka)

Struktury a typy modulu pscg

Název	gr2EventType
Popis	Enumerační typ pro druh události
Hodnoty	EV_NONE EV_PRESSED EV_HOLD EV_LONGHOLD EV_RELEASED

Tabulka 92: Popis enumeračního typu gr2EventType

8.1.6 Popis zdrojových souborů knihovny

Soubor	Popis
fonts/*	Hlavičkové soubory užitých fontů.
GR2.h	Hlavní hlavičkový soubor.
lcd_basics.c	Modul *lcd_basics* – obsahuje elementární grafické funkce.
lcd_basics.h	Hlavičkový soubor modulu *lcd_basics*.
lcd_io.c	Ovladač zobrazovací jednotky pro SDL2.
LICENSE.TXT	Licenční informace.
pscg.c	Modul *pscg*, obsahuje vykreslovací funkci obrazovek, vstupní funkci a funkce pro práci s parametry prvků.
pscg.h	Hlavičkový soubor modulu *pscg*.
pscg_draw.c	Modul *pscg*, obsahuje vykreslovací funkce prvků.
pscg_elements.c	Modul *pscg*, obsahuje konstruktory prvků.

Tabulka 93: Popis Zdrojových souborů knihovny GR2

Poznámka: Pokud by došlo k nějakým nedorozuměním ohledně autorova autorského podílu na popsané grafické knihovně, je každý soubor opatřen licenční hlavičkou s příslušným autorem daného souboru. U všech souborů (krom *lcd_basics.c*) je prvotním autorem Stanislav Brtna. U modulu *lcd_basics* byly převzaty některé elementární grafické funkce ze zdrojů, jež se posléze autorovi už nepodařilo dohledat a nebyl čas tyto funkce nahradit funkcemi dohledatelného původu.

Užité fonty byly vygenerovány ze svobodných true-type fontů programem GLCDFontCreator2. Autorem tohoto programu je pravděpodobně F. Maximilian Thiele, ale webové stránky projektu nefungují. Takto generované fonty byly následně ručně upraveny aby fungovaly s knihovnou GR2.

8.1.7 Detailní funkcionalita

Autor nepředpokládá, že by došlo k nějakému silnějšímu nasazení grafické knihovny GR2, proto je její popis omezen na dokumentaci jejího API. Popis detailní funkcionality bude z těchto omezen a v případě nějakých záludných dotazů se prosím obraťte na autora.

API knihovny je postaveno tak, aby účinně oddělilo uživatele od vnitřních procesů knihovny. Hlavním takovým procesem je kam se vlastně nové prvky ukládají. Soubor *pscg.c* definuje pole struktur typu *pscgElement*, které obsahuje grafické prvky. Dále definuje pole typu *pscgScreen* kam se ukládají data obrazovek (*x_cell*, *y_scroll* atp.) Konstruktory grafických prvků projdou pole prvků dokud nenarazí na první prvek s parametrem *valid* rovným nule a zde inicializují hodnoty parametrů, vrátí pak index tohoto prvku v poli prvků. Smazání grafického prvku se provádí analogicky nastavením parametru *valid* na nulu.

Dalším důležitým procesem je vykreslení grafických prvků. Vykreslovací funkce dostane identifikátor „kořenové“ obrazovky, nastaví vykreslovací oblast dle parametrů, poté prochází pole prvků, hledajíc validní prvky s parametrem screen rovným identifikátoru kořenové obrazovky, takové prvky jsou poté vykresleny. Pokud je takovým prvkem další obrazovka (pod-obrazovka), je vykreslovací oblast uložena a následně oříznuta na rozměry dle pozice pod-obrazovky, pod-obrazovka je následně vykreslena rekurzivním voláním funkce pro vykreslení obrazovky. Po návratu z této funkce je vykreslovací oblast obnovena z předem uložených dat.

Vykreslení je možno ovlivnit parametrem *all*, v takovém případě budou vykresleny všechny prvky, v opačném případě jen prvky s nastaveným parametrem *modified*. Při překreslení prvku je parametr *modified* nastaven na nulu.

Funkce pro zpracování uživatelského vstupu funguje analogicky k funkci vykreslovací, jen namísto vykreslení objektů jim nastaví příslušnou událost, pokud souřadnice dotyku překrývají souřadnice, kde je prvek vykreslen.

Pro detailnější popis interních procesů se obraťte do zdrojového kódu, případně na autora.

8.2 Skriptovací jazyk SVS

Skriptovací jazyk SVS vznikl pro potřebu přesunout důležitou část funkcionality z firmware procesoru na záznamové médium k procesoru připojené (např. SD kartu), za účelem zjednodušení modifikace této funkcionality. V některých aplikacích je potřeba jít za hranice běžných konfiguračních souborů, jež umožňují statická nastavení požadovaných hodnot a do konfigurace přesunout i rozhodovací logiku aplikace. Dnes dostupné skriptovací jazyky pro mikrokontroléry si kladou za cíl přesunout do skriptu veškerou funkcionalitu aplikace, což nemusí být vždy žádoucí. SVS si klade za cíl umožnit co možná nejsnazší provázání skriptu s nativní aplikací. Výsledné řešení je pak takové, kdy většina aplikace je nativní kód a pouze ty části funkcionality, které je potřeba často modifikovat se nacházejí v konfiguračním skriptu. Interpret jazyka SVS je napsán v jazyce C, což umožňuje jeho snadný přenos na mnoho platforem. Interpret jazyka SVS je svobodný software, šířený pod standardní MIT licencí.

8.2.1 Syntaxe jazyka SVS

Struktura skriptu

Strukturu jazyka SVS předvedeme na tradičním „Hello World!“ programu. Program začíná definováním funkce *main*, následované blokem příkazů ({}) obsahujícím volání systémové funkce *print*, která na standardní výstup vytiskne textovou konstantu „Hello world!“. Jednořádkový komentář je uvozen znakem #, víceřádkové komentáře jazyk v aktuální verzi (0.6) neumožňuje.

```
function main{
  #takto vypadá komentář
  sys print("Hello world!");
}
```

Klíčová slova
Všechna klíčová slova jsou case-sensitive.

Klíčové slovo	Funkcionalita
end	Ukončí načítání souboru. Není nutné psát na konec souboru.
break	Přeruší provádění smyčky.
function	Uvozuje funkci.
if, else	Uvozuje podmíněné vykonání kódu.
local	Definuje lokální proměnnou.
return	Návrat z funkce, umožňuje předat návratovou hodnotu.
sys	Uvozuje systémové volání.
while	Uvozuje smyčku typu while.

Tabulka 94: Klíčová slova jazyka SVS

Další řídící znaky

Řídící znak	Funkcionalita
(a)	Začátek a konec závorky, využito buďto u příkazů sys, volání funkcí, podmíněných výrazů a nebo u matematických operací.
{ a }	Začátek a konec bloku. Blok umožňuje vykonání bloku příkazů jako jednoho příkazu, užitečné například u podmíněných výrazů… (Snad Vám nemusím vysvětlovat jak funguje blok.)
;	Středník ukončuje řádek s příkazem.*
,	Odděluje argumenty systémového volání.*
=	Operace přiřazení.
.	Odděluje desetinná místa při zadávání konstant.

Tabulka 95: Řídící znaky jazyka SVS

*Z historických důvodů se jak čárka tak středním překládá na stejný token, proto jsou oba operátory zaměnitelné. Ovšem slušnost je použít čárku k oddělení argumentů a středník k ukončení příkazů.

Funkce

Funkce jsou uvozeny klíčovým slovem *function* následovaným názvem funkce.

Název funkce musí začínat malým nebo velkým písmenem (obojí bez diakritiky), případně znakem podtržítka "_". Další znaky mohou být malá nebo velká písmena (obojí bez diakritiky), znaky "_" a nebo čísly. Délka názvu funkce je omezená nastavením běhového prostředí, definicí makra *NAME_LENGTH* (ve výchozím stavu 14 znaků).

Funkce může být volána interpretem jazyka SVS, nebo jinou funkcí. Volání funkce je ukázáno v příkladu. Funkce v aktuální verzi interpretu neumožňují předávání argumentů. Argumenty je možno v případě nouze předávat pomocí globální proměnné. Funkce umožňuje předat návratovou hodnotu.

V jednom SVS souboru je omezen maximální počet funkcí (definice makra *FUNCTION_TABLE_L*). Ve výchozím stavu je maximální počet funkcí 15.

```
function main{
  hello();
  sys print(""+osm());
}

function hello{
  sys print("Hello");
}

function osm{
  return 8;
}
```

Výpis programu:

```
Hello
8
```

V ukázce funkce *hello* vytiskne na standardní výstup text „Hello" a funkce *osm* vrátí hodnotu 8.

Příkaz *return* je možné použít i bez návratové hodnoty. Také je možno nepoužít příkazu return vůbec, na konci bloku příkazů funkce dojde automaticky k návratu. Mírným problémem návratových hodnot je předem neznámý typ této hodnoty, více o typech hodnot prozradí následující kapitola.

Proměnné

Proměnné jazyka SVS není nutno inicializovat, inicializace se provádí při zavedení programu. Proměnné jsou ve výchozím stavu všechny globální, jejich hodnotu je možné číst i zapisovat ze všech částí programu.

Názvy proměnných mají stejná omezení jako názvy jako názvy funkcí. V jednom programu může existovat funkce i proměnná stejného názvu.

Maximální počet proměnných je omezen (definice *VAR_TABLE_L*). Ve výchozím stavu je maximální počet proměnných 150.

Klíčové slovo *local* umožňuje v bloku příkazů vytvořit lokální proměnnou, která zanikne po vykonání tohoto bloku. Viz ukázka.

```
function main{
  a=5;
  {
    local a;
    a=8;
    {
      local a;
      a="Karel";
      sys print("a="+a);
    }
    sys print("a="+a);
  }
  sys print("a="+a);
}
```

Výstup:

```
a=Karel
a=8
a=5
```

Typy hodnot v SVS

Z předchozích ukázek je patrné, že proměnné SVS mohou nabývat hodnot jak číselných tak textových. Tyto typy hodnot jsou omezeny v operacích, které je nad nimi možno provádět. Ve výchozím stavu je každá proměnná inicializována na celé číslo s hodnotou 0.

Typ	Inicializace	Popis

celé číslo	a=5;
	b=-97;
	c=0x3FF;
		Celé číslo. Definováno jako int32_t. Možno zadat i hexadecimálně.

desetinné číslo	a=6.22;
	b=8.5;
	c=3.0;
		Desetinné číslo. Definováno jako 32bit float.*

Textový řetězec	a=“milan“; b=“auto“;	Textový řetězec, maximální délka řetězce je omezena volným místem textové paměti.

Tabulka 96: Tabulka typů hodnot, zároveň s ukázkou definic konstant.

* Pozor na operace s desetinnými a celými čísly! Např. *3 + 1.5* vyvolá chybu, správně je *3.0 + 1.5*. Typicky pozor na 0 a 0.0 .

Pokud je potřeba užít znaku #, například v textové konstantě, je třeba napsat znak dvakrát (##). Takováto sekvence znaků nebude považována za komentář. Při potřebě přesunu na nový řádek v textové konstantě je možnost buďto odřádkovat přímo ve zdrojovém souboru případně použít escape znak.

Escape znak	Význam	Popis
\n	Newline	přesun na nový řádek
\\	Backslash	zpětné lomítko
\“	Double quote	uvozovky

Tabulka 97: Tabulka podporovaných escape znaků

Operátory

Jazyk SVS umožňuje provádění základních početních operací a spojování řetězců. Vzhledem k různým typům hodnot je prováděna typová kontrola a operace napříč typy jsou omezeny. Zde je seznam operátorů společně s jejich významem.

Operátor	Funkcionalita
+	Sčítání: Sčítá numerické hodnoty, spojuje řetězce. Při použití s numerickou hodnotou a řetězcem převede numerickou hodnotu na nový řetězec a oba řetězce spojí.
-	Odčítání.
*	Násobení.
/	Při použití nad celými čísly dělí beze zbytku, při použití nad čísly s pohyblivou desetinou čárkou dělí se zbytkem.
%	Zbytek po celočíselném dělení.
==	Rovno. Vrací jedničku pokud jsou oba operandy shodné. (Použitelné jak na čísla tak na řetězce)
!=	Nerovno. Vrací jedničku pokud jsou oba operandy rozdílné. (Použitelné jak na čísla tak na řetězce)
<=	Menší nebo rovno.
<	Menší než.
>=	Větší nebo rovno.
>	Větší než.

Tabulka 98: Tabulka operátorů

Priorita (od nejvyšší po nejnižší)	Operátory
1.	*, /, %,
2.	+, -
3.	+ (mezi řetězci a nebo řetězci a čísly)
4.	==, !=, <=, >=, <, >

Tabulka 99: Tabulka priority operátorů

Operátory stejné priority se vykonávají zleva doprava. Výrazy v závorkách mají přednost, závorky do sebe lze neomezeně zanořovat.

	řetězec	**celé číslo**	**desetinné číslo**
řetězec	+, ==, !=	+	+
celé číslo	X	*, /, %, +, -, ==, !=, <=, >=, <, >	
desetinné číslo	X		*, /, +, -, ==, !=, <=, >=, <, >

Tabulka 100: Možnosti operací napříč typy.

Přetypování (převod z jednoho typu hodnoty na jiný) není ve výchozím stavu podporováno. Výjimku tvoří operátor součtu (+) použitý mezi řetězcem a jedním z číselných typů, takováto operace převede číslo na řetězec, spojí oba řetězce do nového a ten předá dál. Převod čísel s pohyblivou desetinnou čárkou není IEEE 754 kompatibilní a měl by být používán na vlastní nebezpečí. Některé operace pro přetypování obsahuje základní knihovna funkcí (*pcBasicWrap*).

Ukázka

```
a=98;
a=a+1; #iknrementace o 1
a=(74*(12+5))/(-1); #záporná čísla je ve výpočtu nutno dát do závorky

b=1;
#pokud je b rovno nule c bude 8, pokud je b rovno 1, c bude 5
c=(b==0)*8+(b==1)*5;

b=3.89;
c=b-0.5;

d=“ahoj“;
e=d+“ světe“
#výsledný řetězcec bude „ahoj světe“

a=5*2;
f=“výsledek: “+a;
#výsledný řetězec bude „výsledek: 10“
```

Logické operátory

Vzhledem k absenci logických operátorů je třeba použít operátorů numerických a rozdílnou prioritu sčítání a násobení ošetřit vhodným použitím závorek.

Podmíněné příkazy a smyčky

Podmíněné příkazy umožňují základní větvení programu. Podmíněné příkazy do sebe lze vnořovat.

Příkazy if a else

Základním řídícím příkazem je příkaz *if*. Příkaz *if* musí být následován závorkou, obsahující výraz složený z operátorů, proměnných, konstant a volání funkcí. Pokud výsledek tohoto výrazu nemá hodnotu nula, je proveden příkaz následující příkaz *if*. Takovým příkazem může být i blok příkazů. Pokud však má výraz hodnotu nula, je příkaz (případně blok příkazů) přeskočen a vykoná se až příkaz následující.

Volitelně je možno použít konstrukce příkazů *if-else*, kde příkaz (blok příkazů) za příkazem *if* je následován příkazem *else* a dalším příkazem (blokem příkazů). Takováto konstrukce umožňuje po splnění podmínky vykonání *if* větve a přeskočení *else* větve, v opačném případě (podmínka nesplněna) je *if* větev přeskočena a *else* větev vykonána.

Ukázka příkazu if:

```
if (1)
    sys print("True");

if (b<10){
    sys print("5");
  b=b+1;
}else{
  sys print("reset!");
  b=0;
}

if (a==5)
    sys print("5");
  else if(a==6)
    sys print ("6");
  else if(a==8)
    sys print("8");
  else
    sys print("not 5, not 6, not 8");
```

Dobrou praxí je dávat příkazy vždy do složených závorek.

Příkaz while

Jediným příkazem pro realizaci programové smyčky v SVS je příkaz *while*. Příkaz *while* musí být následován závorkou s výrazem podmínky po jejíž splnění má být proveden (podmínka je stejného formátu jako pro příkaz *if*). Vykonávání smyčky je možné uvnitř smyčky přerušit příkazem *break*.

Ukázka příkazu while:

```
while (a<5){
  a=a+1;
  sys print("a:" +a);
}

while(1){
  if(a<5){
    a=a+1;
  }else{
    break;
  }
}
```

Užití příkazu *break* mimo smyčku se nedoporučuje, v aktuální verzi SVS dojde v tomto případě k ukončení vykonávání programu.

Stejně jako programovací jazyk BASIC neměl příkaz *while*, nemá SVS příkaz *for*. Tedy alespoň prozatím. V případě nutnosti smyčky typu for prostě v těle smyčky *while* inkrementujte proměnnou jež je součástí podmínky vykonání smyčky, viz předešlé příklady.

Systémové funkce wapperu basics

Běhové prostředí jazyka SVS umožňuje snadné navázání příkazů skriptu SVS na nativní kód v jazyce C. Vrstva umožňující toto provázání se skládá z tzv. wrapperů. Základní systémové funkce pro propojení běhového prostředí SVS s nativním kódem představuje wrapper *pcBasicWrap*. V tomto wrapperu je také několik funkcí jež by se měly v budoucnosti přesunout přímo mezi základní příkazy SVS, rozšíření tohoto wrapperu je snadnou cestou pro přidání další funkcionality.

Volání systémových funkcí se provede příkazem *sys* následovaným názvem funkce a v závorce uvedenými parametry systémové funkce. Systémové funkce mohou mít návratovou hodnotu, kterou je možno užít ve výrazech přiřazení hodnoty i podmínky. Za klíčovým slovem *sys* musí být vždy alespoň jeden mezerník. Viz následující ukázka.

```
sys print("Hello World");

a=sys test();

if(sys test()){
  B=5;
}
```

Systémové funkce wrapperu basics

Název	real
Popis	Převede textový řetězec na celé číslo.
Volání	sys real(str);
Návratová hodnota	Celočíselná hodnota obsažená v řetězci
Parametry	str – řetězec
Poznámka	Nefunguje pro čísla s pohyblivou desetinnou čárkou.

Tabulka 101: Popis systémové funkce (jazyk SVS)

Název	dbg
Popis	Nastaví režim ladění. Na standardní výstup budou v režimu ladění vypisovány informace o běhu programu.
Volání	sys dbg(lvl);
Návratová hodnota	0
Parametry	lvl – celé číslo, 1 – spustí režim ladění, 0 – vypne režim ladění
Poznámka	V tomto režimu je zapnuto ladění pro matematické výrazy a řídící příkazy. Toto je režim vhodný pro hledání většiny chyb.

Tabulka 102: Popis systémové funkce (jazyk SVS)

Název	dbgGc
Popis	Nastaví ladící režim garbage-collectoru.
Volání	sys dbgGc(lvl);
Návratová hodnota	0
Parametry	lvl – celé číslo, 1 – spustí režim ladění, 0 – vypne režim ladění
Poznámka	

Tabulka 103: Popis systémové funkce (jazyk SVS)

Název	dbgCache
Popis	Nastaví ladící režim cache tokenů.
Volání	sys dbgCache(lvl);
Návratová hodnota	0
Parametry	lvl – celé číslo, 1 – spustí režim ladění, 0 – vypne režim ladění
Poznámka	

Tabulka 104: Popis systémové funkce (jazyk SVS)

Název	profiler
Popis	Nastaví režim profilování.
Volání	sys profiler(enable);
Návratová hodnota	0
Parametry	enable – celé číslo, 1 – povolí režim profilování, 0 – zakáže režim profilování
Poznámka	

Tabulka 105: Popis systémové funkce (jazyk SVS)

Název	GC
Popis	Vyvolá rutinu garbage-collectoru.
Volání	sys GC(toFree);
Návratová hodnota	0
Parametry	toFree – celé číslo, udává počet znaků, které se má garbage-collector pokusit uvolnit
Poznámka	Pouze pro speciální účely, běžně je garbage-collector volán automaticky.

Tabulka 106: Popis systémové funkce (jazyk SVS)

Název	info
Popis	Vypíše informace o prostředí SVS na standardní výstup. Jedná se hlavně o hodnoty ze souboru *svs_limits.h* .
Volání	sys info();
Návratová hodnota	0
Parametry	
Poznámka	

Tabulka 107: Popis systémové funkce (jazyk SVS)

Název	print
Popis	Vypíše text na standardní výstup.
Volání	sys print(str);
Návratová hodnota	0
Parametry	Str – textový řetězec (případně výraz na něj vedoucí), jež má být vypsán na standardní výstup.
Poznámka	

Tabulka 108: Popis systémové funkce (jazyk SVS)

Implementace polí pomocí wrapperu basics

Název	arrayNew
Popis	Přidá nové pole.
Volání	sys arrayNew(len);
Návratová hodnota	0 pokud inicializace selže id nového pole pokud je inicalizace úspěšná
Parametry	len – celé číslo udávající délku pole
Poznámka	

Tabulka 109: Popis systémové funkce (jazyk SVS)

Název	arrayGet
Popis	Umožňuje přečíst hodnotu prvku z pole.
Volání	sys arrayGet(id, index);
Návratová hodnota	Vrátí hodnotu prvku pole. Hodnota může být jak textový řetězec, tak celé číslo nebo číslo s pohyblivou desetinnou čárkou.
Parametry	id – identifikátor pole index – index prvku
Poznámka	

Tabulka 110: Popis systémové funkce (jazyk SVS)

Název	arraySet
Popis	Umožňuje zadat hodnotu prvku pole.
Volání	sys arraySet(id, index, val);
Návratová hodnota	0
Parametry	id – identifikátor pole index – index prvku val – nová hodnota prvku.
Poznámka	Hodnota může být jak textový řetězec, tak celé číslo nebo číslo s pohyblivou desetinnou čárkou.

Tabulka 111: Popis systémové funkce (jazyk SVS)

Chybové výpisy, problémy a řešení

Jedním z důležitých parametrů jakéhokoli programovacího nástroje je detekce chyb programu. Následný výpis chybového stavu musí být dostatečně podrobný a přesný aby bylo možno chybu programu rychle odhalit a opravit. Běhové prostředí jazyka SVS odhaluje chyby ve zdrojovém souboru dvěma způsoby:

1. Chyby odhalitelné při tokenizaci.

Program v jazyce SVS je nejprve převeden z textového formátu na formát pole tokenů, jež jsou následně vykonávány. Během této operace mohou být odhaleny a vypsány následující chyby:

- liché počty závorek složených i normálních
- duplicitní funkce
- nekorektní názvy funkcí a proměnných
- nekorektní escape znaky
- nekorektní znaky ve zdrojovém souboru
- přeplnění globálního textového pole

Po detekci chyby je chyba vypsána na standardní výstup. SVS se poté pokusí vypsat řádek na kterém chybu detekovalo. Při detekci chyby na úrovni tokenizace nedojde ke spuštění programu.

2. Chyby odhalené za běhu programu

Za běhu programu mohou být odhaleny další chyby, zejména související s typy hodnot. Dále je zde ošetřeno volání neexistujících funkcí a neexistujících systémových funkcí.

- typová kontrola výrazů a volání
- volání nedefinovaných funkcí
- volání nedefinovaných systémových funkcí
- chybějící závorky, čárky a středníky
- přeplnění globálního textového pole

Chyba vyvolaná za běhu programu vyvolá opuštění funkce skriptu, chyba je poté případně obsloužena nativním kódem. Ve výchozím stavu se vypíše na standardní výstup a program se ukončí. SVS umožňuje i další způsoby zotavení se z chybového stavu, jako například volání obslužné funkce na straně skriptu, jež nastaví výchozí hodnoty atp.

8.2.2 SVS a optimalizace

Tato kapitola hlouběji rozebere některé klíčové funkční bloky SVS, jejichž znalost není k programování v jazyce SVS bezprostředně nutná, ale hodí se pro zrychlení běhu programu a nebo při řešení neočekávaných chybových stavů.

Tokenizer

Jak už bylo naznačeno, virtuální stroj jazyka SVS nevykonává program v textové podobě, ale nejprve jej převede na pole tokenů, jež jsou následně vykonány. Toto převádění není příliš optimalizováno, tudíž je možno strukturou programu ovlivnit rychlost vykonávání výsledného kódu. Jedná se hlavně o práci s konstantami, ta není optimalizovaná a tak je možno vhodnou ruční optimalizací konstant dosáhnout výkonnostního zlepšení.

```
#neoptimální
a=1+1+1+1+1+b;
#optimální
a=5+b;
```

Další takovouto optimalizací například může být eliminace nadbytečných závorek.

Cachache tokenů

Virtuální stroj jazyka SVS byl primárně navržen pro mikrokontroléry s kapacitou paměti 20 kB a vyšší. Aby bylo možno do paměti uložit delší program, disponuje SVS odkládacím souborem tokenů, jež se uloží na paměťové médium, ze kterého je načítán zdrojový soubor. Tokeny bezprostředně nutné pro běh programu jsou uloženy v RAM mikrokontroléru, v mezipamněti (chache) tokenů. Tato mezipaměť není v aktuální verzi příliš dobře implementována. Řešení je funkční, ale trpí výkonnostními problémy, při změně obsahu chache dojde ke zpomalení vykonávání programu. Ideální by tedy bylo, když by se vykonávaný program vešel celý do mezipaměti. Bohužel aktuálně neexistují žádné prostředky, jak toto zajistit.

Garbage collector

Všechny textové řetězce programu v SVS jsou uloženy v globálním textovém poli. Toto textové pole má omezenou délku. Každá operace, jež generuje novou hodnotu typu textového řetězce tuto hodnotu vkládá na konec tohoto globálního textového pole. Nejčastější takovouto operací je sčítání řetězců, případně převod čísla na textový řetězec. Po určité době vykonávání takovýchto operací by se globální textové pole naplnilo a program by se přestal vykonávat. Aby došlo k zabránění této situaci, obsahuje virtuální stroj SVS svůj garbage-collector (uklízeč odpadu), jež kontroluje stav naplnění globálního textového pole, v případě, že je toto pole před naplněním, garbage-collector vymaže zbytečné řetězce. Nalezení zbytečných textových řetězců probíhá variací algoritmu *mark and sweep* s následnou

defragmentací globálního textového pole. Tato operace není dobře optimalizovaná a způsobí zastavení běhu programu dokud nedojde jejímu dokončení. Z tohoto důvodu je třeba dbát jisté opatrnosti a otestovat, zda program nevytváří více řetězců, než po něm stihne GC uklidit, v takové situaci totiž dojde ke znatelnému zpomalení vykonávání programu. Obranou proti tomuto problému je volání systémové funkce GC, když běh programu dovoluje mírné zpomalení.

8.2.3 Aplikační rozhraní pro jazyk C:

Primární zaměření jazyka SVS bylo jeho snadné začlenění do nativní aplikace v jazyce C. SVS lze snadno použít na linuxovém desktopu (případně nad MS Windows pod Cygwin) nebo na mikrokontrolérech STM (testovány řady stm32f1x, stm32f4x a stm32f7x), vybavených SD kartou a knihovnou FATfs.

Využití SVS interpretu ve vlastní aplikaci

Tato kapitole se zabývá konfigurací běhového prostředí jazyka SVS a jeho začleněním do aplikace v jazyce C.

Nastavení

Nastavení běhového prostředí SVS je možno upravit úpravou souboru svs_limits.h. Tento soubor obsahuje definice maker, jež ovlivňují různá nastavení. Viz následující tabulka.

Makro	Popis	Výchozí hodnota
TOKEN_CACHE_DISABLED	Zakáže ukládaní souborů do odkládacího souboru.	0
TOKEN_LENGTH	Maximální počet tokenů v RAM	500
NAME_LENGTH	Maximální délka názvu funkce nebo proměnné.	15
FILE_NAME_L	Maximální délka názvu souboru.	32
FUNCTION_TABLE_L	Maximální počet funkcí.	15
VAR_TABLE_L	Maximální počet proměnných.	150
STRING_FIELD_L	Délka globálního textového pole.	1500
SYSCALL_TABLE_L	Kolik různých sys volání může obsahovat jeden soubor.	50
SYSCALL_WRAPPERS	maximální počet wrapperů	5
LOCAL_VARIABLES_ENABLED	Pokud definováno, jsou povoleny lokální proměnné.	
USE_FLOAT	Pokud definováno, je povolena práce s čísly s pohyblivou desetinnou čárkou	

Tabulka 112: Konfigurační makra preprocesoru.

Konfigurací tohoto souboru je možno zásadně ovlivnit paměťovou náročnost interpretu jazyka SVS.

Inicializace interpretu

Prvním krokem je přidání hlavičkového souboru *svs_basics.h* z adresáře SVS. Při použití SVS wrapperů je nutno definovat prototypy inicializačních funkcí těchto wraperů. Například:

```
void pcBasicWrapInit();
void svsGr2WrapInit();
```

Dále inicializace struktury typu *svsVM* držící stav programu:

```
svsVM s;
```

Poté můžeme zavolat inicializační funkce wrapperů.

```
pcBasicWrapInit();
svsGr2WrapInit();
```

Načtení programu je závislé na platformě a způsobu načtení souboru. Výchozí funkcí pro načítání souboru je *loadApp*. Tato funkce v závislosti na definování makra preprocesoru "*PC*" použije buďto funkce knihovny FATfs nebo standardní systémové funkce pro práci se soubory.

Název funkce	loadApp
Prototyp funkce	uint8_t loadApp(uint8_t * fname, uint8_t * name, svsVM *s, uint8_t mode)
Popis	Resetuje, inicializuje a načte program do struktury typu svsVM.
Parametry	fname – název souboru name – název programu s – svsVM mode – mód načítání, standardně 0
Návratová hodnota	0 – pokud byl program úspěšně načten 1 – pokud došlo během načítání k chybě

Tabulka 113: Popis funkce loadApp (jazyk C)

Pokud je potřeba načtení souboru jiným způsobem, je možno modifikovat, či úplně nahradit funkci *tokenGetch* ze souboru *svs_load.c* tak, aby mohl být program načítán například ze sériové linky, paměti flash atp.

Provádění kódu

Po úspěšném zavedení programu nyní můžeme zavolat funkci z prostředí SVS skriptu. K tomuto účelu slouží funkce *commExec*.

```
commExec("main", &s);
```

Název funkce	commExec
Prototyp funkce	uint16_t commExec(uint8_t * name, svsVM *s);
Popis	Vykoná funkci skriptu SVS.
Parametry	name – název funkce s – svsVM
Návratová hodnota	Pozice tokenu na které funkce skončila.

Tabulka 114: Popis funkce commExec (jazyk C)

Po provedení dané funkce je třeba ověřit, zda nedošlo při vykonávání skriptu k chybě a případně tuto chybu vypsat nebo jinak ošetřit.

```
if (errCheck(&s)){ //detekce chyby
  errSoftPrint(&s); //tisk chyby
  //zde může být případně další obsluha
}
```

Získání návratové hodnoty funkce spočívá v přečtení typu návratové hodnoty ze struktury svsVM (prvek *commRetType*) a vyčtení konkrétní hodnoty jednou z následujících funkcí. Tyto funkce snad nepotřebují podrobné vysvětlení, uvedeny jsou jejich prototypy.

```
int32_t getRetValInt(svsVM *s);
uint8_t * getRetValStr(svsVM *s);
float getRetValFlt(svsVM *s);
```

Typ uložený v může nabývat hodnot:

Hodnota	**Typ**
0	int32_t
1	Ukazatel na nulou zakončený řetězec. (uint8_t *)
2	float

Tabulka 115: Numerické hodnoty typů proměnných.

U návratové hodnoty typu řetězec musíme počítat s tím, že při dalším provedení jakékoli funkce daného svsVM můžeme o textový řetězec přijít, díky tomu, že ho vymaže garbage collector. Proto je potřeba ho ideálně zkopírovat do nějakého bufferu.

Příklad použití:

```
if (s.commRetType==0){
  printf("Return: %i\n",getRetValInt(&s));
}else if(s.commRetType==1){
  printf("Return: %s\n", getRetValStr(&s));
}else if(s.commRetType==2) {
  printf("Return: %f\n",getRetValFlt(&s));
}
```

Při volání funkce z prostředí SVS se může hodit předem ověřit existenci této funkce pomocí *functionExists*.

Název funkce	functionExists
Prototyp funkce	uint8_t functionExists(uint8_t *name, svsVM *s);
Popis	Zjistí zda sané svsVM obsahuje danou funkci.
Parametry	name – název funkce s – svsVM
Návratová hodnota	1 – funkce existuje 2 – funkce neexistuje

Tabulka 116: Popis funkce functionExists (jazyk C)

Tvorba wrapperu C funkcí:

Funkce wrapperu spočívá v poskytnutí nativních funkcí jazyka C do prostředí skriptu SVS. Pro přidání wrapperu není nutno zasahovat do kódu interpretu SVS. Přidání je provedeno inicializační funkcí, jež přidá odkaz na obslužnou funkci wrapperu do globálního pole SVS wrapperů.

Příklad inicializační funkce:

```
#include "SVS/svs_basics.h"

// hlavička obslužné funkce wrapperu
uint8_t exampleWrapper(varRetVal *result, argStruct *argS, svsVM *s);

//inicializační funkce wrapperu
void exampleWrapper Init(){
  addSysWrapper(exampleWrapper); //funkce SVS pro přidání wrapperu
}
```

Obslužná funkce wrapperu je volána automaticky když interpret SVS narazí na příkaz *sys*. Jsou jí předány hodnoty argumentů, typy argumentů, jejich počet a název systémového volání. Obslužná funkce provede vyhodnocení těchto hodnot. Na základě toho je buďto volání obslouženo a funkce ukončena. Nebo funkce ukončena aniž by volání obsloužila. Případně dojde k vyvolání chyby a ukončení funkce (typicky když nesedí typy argumentů).

```
uint8_t exampleWrapper(varRetVal *result, argStruct *argS, svsVM *s){
  uint8_t argType[11];

  if (sysFuncMatch(argS->callId,"test",s)){ //vyhodnocení názvu sys funkce
    argType[1]=0; //nastavení typů argumetů
    argType[2]=0;
    argType[3]=1;

    if(sysExecTypeCheck(argS, argType, 3,s)){ //kontola argumentů
      return 0;
    }
    // obsluha
    printf("Voláná sys test funkce!  arg1=%i arg2=%i arg3=%s\n", argS->
      arg[1].val_s, argS->arg[2].val_s, s->stringField+argS-
>arg[3].val_str);

    // předání návratové hodnoty
    result->value.val_s=8;
    result->type=0;
    return 1;
  }

  //zde může být obsluha dalších systémových volání
  return 0;
}
```

Předávání hodnot mezi SVS a C

Nejjednodušší metodou je použití již popsaných systémových funkcí a zavedení SVS sys funkcí typu *getValX* a *setValX*. Případně je možné využít návratové hodnoty volané funkce. Při předávání řetězce do prostředí SVS je třeba použít funkce *strNew*, pro přidání nového řetězce do globálního textového pole.

Název funkce	strNew
Prototyp funkce	uint16_t strNew(uint8_t *str, svsVM *s);
Popis	Přidá nový řetězec do globálního textového pole dané svsVM
Parametry	str – nulou ukončený řetězec s – svsVM
Návratová hodnota	id nového řetězce

Tabulka 117: Popis funkce strNew (jazyk C)

Pokud na druhou stranu dostaneme v hodnotě proměnné řetězec, jedná se o index v globálním textovém poli. Je proto třeba k této hodnotě přičíst ukazatel na začátek globálního textového pole, abychom dostali ukazatel na požadovaný řetězec (povšimněte si třetího argumentu u obsluhy volání *test* v příkladu *exampleWrapper*).

8.3 GR2 wrapper pro SVS

GR2 wrapper pro SVS rozšiřuje knihovnu systémových funkcí programovacího jazyka SVS o volání z grafické knihovny GR2. Umožňuje pak programu běžícímu v prostředí SVS vytvářet a obsluhovat grafické prvky této knihovny.

Jednotlivé popsané funkce mají přísně definované typy argumentů a typy návratových hodnot, v tabulce je celočíselný typ označen *num* a typ textového přetěžce *txt*.

Většina funkcí přesně kopíruje svojí předlohu v knihovně GR2, pro detaily ohledně fungování knihovny GR2 se obraťte na příslušnou kapitolu.

Název	pAddScreen
Popis	Vytvoří novou obrazovku.
Volání	sys pAddScreen();
Návratová hodnota	id nové obrazovky

Tabulka 118: Popis funkce viz tabulka (jazyk SVS)

Název	pDestroyScr
Popis	Vymaže obrazovku a všechny její prvky.
Volání	sys pDestroyScr(id);
Parametry	id (num) – identifikátor obrazovky
Poznámka	Pokud se na obrazovce nachází objekt typu rámce, nedojde ke smazání obrazovky v tomto rámci.

Tabulka 119: Popis funkce viz tabulka (jazyk SVS)

Název	pAddFrame
Popis	Vytvoří nový rámec
Volání	sys pAddFrame(x1, y1, x2, y2, value, screen);
Návratová hodnota	id (num) – identifikátor nového prvku
Parametry	x1 (num) – x levého horního rohu prvku [x mřížky obrazovky] y1 (num) – y levého horního rohu prvku [y mřížky obrazovky] x2 (num) – x pravého dolního rohu prvku [x mřížky obrazovky] y2 (num) – y pravého dolního rohu prvku [y mřížky obrazovky] value (num) – id obrazovky jež má být zobrazena v rámci screen (num) – id obrazovky prvku

Tabulka 120: Popis funkce viz tabulka (jazyk SVS)

Název	pAddButton
Popis	Vytvoří nové tlačítko.
Volání	sys pAddButton(x1, y1, x2, y2, str, scrId);
Návratová hodnota	id (num) – identifikátor nového prvku
Parametry	x1 (num) – x levého horního rohu prvku [x mřížky obrazovky] y1 (num) – y levého horního rohu prvku [y mřížky obrazovky] x2 (num) – x pravého dolního rohu prvku [x mřížky obrazovky] y2 (num) – y pravého dolního rohu prvku [y mřížky obrazovky] str (txt) – text prvku screen (num) – id obrazovky prvku
Poznámka	

Tabulka 121: Popis funkce viz tabulka (jazyk SVS)

Název	pAddSliderV
Popis	Vytvoří nový vertikální posuvník.
Volání	sys pAddSliderV(x1, y1, x2, y2, z_kolika, kolik, screen);
Návratová hodnota	id (num) – identifikátor nového prvku
Parametry	x1 (num) – x levého horního rohu prvku [x mřížky obrazovky] y1 (num) – y levého horního rohu prvku [y mřížky obrazovky] x2 (num) – x pravého dolního rohu prvku [x mřížky obrazovky] y2 (num) – y pravého dolního rohu prvku [y mřížky obrazovky] z_kolika (num) – maximální hodnota posuvníku kolik (num) – hodnota posuvníku screen (num) – id obrazovky prvku

Tabulka 122: Popis funkce viz tabulka (jazyk SVS)

Název	pAddSliderH
Popis	Vytvoří nový horizontální posuvník.
Volání	sys pAddSliderH(x1, y1, x2, y2, z_kolika, kolik, screen);
Návratová hodnota	id (num) – identifikátor nového prvku
Parametry	x1 (num) – x levého horního rohu prvku [x mřížky obrazovky] y1 (num) – y levého horního rohu prvku [y mřížky obrazovky] x2 (num) – x pravého dolního rohu prvku [x mřížky obrazovky] y2 (num) – y pravého dolního rohu prvku [y mřížky obrazovky] z_ kolika (num) – maximální hodnota posuvníku kolik (num) – hodnota posuvníku screen (num) – id obrazovky prvku
Poznámka	

Tabulka 123: Popis funkce viz tabulka (jazyk SVS)

Název	pAddBar
Popis	Přidá nový indikátor průběhu.
Volání	sys pAddBar(x1, y1, x2, y2, z_kolika, kolik, scrId);
Návratová hodnota	id (num) – identifikátor nového prvku
Parametry	x1 (num) – x levého horního rohu prvku [x mřížky obrazovky] y1 (num) – y levého horního rohu prvku [y mřížky obrazovky] x2 (num) – x pravého dolního rohu prvku [x mřížky obrazovky] y2 (num) – y pravého dolního rohu prvku [y mřížky obrazovky] z_ kolika (num) – maximální hodnota posuvníku kolik (num) – hodnota posuvníku scrId (num) – id obrazovky prvku
Poznámka	Indikátor se zobrazí buďto horizontální nebo vertikální, v závislosti na rozměrech prvku.

Tabulka 124: Popis funkce viz tabulka (jazyk SVS)

Název	pAddText
Popis	Přidá nové textové pole.
Volání	sys pAddText(x1, y1, x2, y2, str, scrId);
Návratová hodnota	id (num) – identifikátor nového prvku
Parametry	x1 (num) – x levého horního rohu prvku [x mřížky obrazovky] y1 (num) – y levého horního rohu prvku [y mřížky obrazovky] x2 (num) – x pravého dolního rohu prvku [x mřížky obrazovky] y2 (num) – y pravého dolního rohu prvku [y mřížky obrazovky] str (txt) – text textového pole scrId (num) – id obrazovky prvku

Tabulka 125: Popis funkce viz tabulka (jazyk SVS)

Název	pDestroy
Popis	Smaže prvek daného id.
Volání	sys pDestroy(id);
Parametry	id (num) – id prvku
Poznámka	Pro mazání obrazovek použijte pDestroyScr.

Tabulka 126: Popis funkce viz tabulka (jazyk SVS)

Název	pGetValue
Popis	Vrátí hornotu prvku.
Volání	sys pGetValue(id);
Návratová hodnota	val (num) – hodnota prvku (parametr *value*)
Parametry	id (num) – id prvku

Tabulka 127: Popis funkce viz tabulka (jazyk SVS)

Název	pSetValue
Popis	Nastaví hodnotu prvku.
Volání	sys pSetValue(id, val);
Parametry	id (num) – id prvku val (num) – hodnota prvku (parametr *value*)

Tabulka 128: Popis funkce viz tabulka (jazyk SVS)

Název	pGetEvent
Popis	Vrátí událost prvku. (parametr *event*)
Volání	sys pGetEvent(id);
Návratová hodnota	event (num) – numerická hodnota následujícího významu: 0 – *EV_NONE* 1 – *EV_PRESSED* 2 – *EV_HOLD* 3 – *EV_RELEASED*
Parametry	id (num) – id prvku

Tabulka 129: Popis funkce viz tabulka (jazyk SVS)

Název	pSetEvent
Popis	Nastaví událost prvku. (parametr *event*)
Volání	sys pSetEvent(id, val);
Parametry	id (num) – id prvku val (num) – hodnota, viz pGetEvent

Tabulka 130: Popis funkce viz tabulka (jazyk SVS)

Název	pClrScrEv
Popis	Nastaví události všem prvkům dané obrazovky na EV_NONE (hodnota 0)
Volání	sys pClrScrEv(scrId);
Parametry	
Poznámka	V rámci optimalizace rychlosti vykonání programu je lepší použít více volání pSetEvent.

Tabulka 131: Popis funkce viz tabulka (jazyk SVS)

Název	pSetVisible
Popis	Nastaví viditelnost prvku. (parametr *visible*)
Volání	sys pSetVisible(id, val);
Parametry	id (num) – id prvku val (num) – hodnota viditelnosti
Poznámka	0 – prvek není viditelný 1 – prvek je viditelný

Tabulka 132: Popis funkce viz tabulka (jazyk SVS)

Název	pSetScreen
Popis	Nastaví danému prvku danou obrazovku.
Volání	sys pSetScreen(id, scr);
Parametry	id (num) – id prvku scr (num) – nová obrazovka prvku
Poznámka	Tímto příkazem lze přidat podobrazovku.

Tabulka 133: Popis funkce viz tabulka (jazyk SVS)

Název	pSetStr
Popis	Nastaví textový parametr daného prvku.
Volání	sys pSetStr(id, str);
Parametry	id (num) – id prvku str (txt) – nová textová hodnota prvku
Poznámka	Platné pro prvky *text* a *button*.

Tabulka 134: Popis funkce viz tabulka (jazyk SVS)

Název	pGetStr
Popis	Vrátí textovou hodnotu prvku.
Volání	sys pGetStr(id);
Návratová hodnota	str (txt) – textová hodnota prvku
Parametry	id (num) – id prvku

Tabulka 135: Popis funkce viz tabulka (jazyk SVS)

Název	pSetXYXY
Popis	Umožňuje nastavit všechny souřadnice prvku najednou,
Volání	sys pSetXYXY(id, x1, y1, x2, y2);
Parametry	id (num) – id prvku x1 (num) – x levého horního rohu prvku [x mřížky obrazovky] y1 (num) – y levého horního rohu prvku [y mřížky obrazovky] x2 (num) – x pravého dolního rohu prvku [x mřížky obrazovky] y2 (num) – y pravého dolního rohu prvku [y mřížky obrazovky]
Poznámka	Vhodné například k nastavení pozice pod-obrazovky.

Tabulka 136: Popis funkce viz tabulka (jazyk SVS)

Název	pSetTxtSize
Popis	Umožňuje nastavit velikost textu textového pole.
Volání	sys pSetTxtSize(id, val);
Parametry	id (num) – id prvku val (num) – nová hodnota velikosti textu
Poznámka	Hodnota velikosti není libovolná a je dána užitými fonty knihovny GR2.

Tabulka 137: Popis funkce viz tabulka (jazyk SVS)

Parametry obrazovek:

Název	pGetXscroll
Popis	Umožňuje obdržet parametr *xscroll.*
Volání	sys pGetXscroll(id);
Návratová hodnota	xscroll (num) – parametr *xscroll*
Parametry	id (num) – id obrazovky

Tabulka 138: Popis funkce viz tabulka (jazyk SVS)

Název	pSetXscroll
Popis	Umožňuje nastavit parametr *xscroll.*
Volání	sys pSetXscroll(id, val);
Parametry	id (num) – id obrazovky val (num) – nová hodnota parametru

Tabulka 139: Popis funkce viz tabulka (jazyk SVS)

Název	pGetYscroll
Popis	Umožňuje obdržet parametr y*scroll.*
Volání	sys pGetYscroll(id);
Návratová hodnota	yscroll (num) – parametr y*scroll*
Parametry	id (num) – id obrazovky

Tabulka 140: Popis funkce viz tabulka (jazyk SVS)

Název	pSetYscroll
Popis	Umožňuje nastavit parametr y*scroll.*
Volání	sys pSetYscroll(id, val);
Parametry	id (num) – id obrazovky val (num) – nová hodnota parametru

Tabulka 141: Popis funkce viz tabulka (jazyk SVS)

Název	pGetXcell
Popis	Umožňuje obdržet parametr *x_cell.*
Volání	sys pGetXcell(id);
Návratová hodnota	x_cell (num) – parametr *x_cell*
Parametry	id (num) – id obrazovky

Tabulka 142: Popis funkce viz tabulka (jazyk SVS)

Název	pSetXcell
Popis	Umožňuje nastavit parametr *x_cell.*
Volání	sys pSetXcell(id, val);
Parametry	id (num) – id obrazovky val (num) – nová hodnota parametru

Tabulka 143: Popis funkce viz tabulka (jazyk SVS)

Název	pGetYcell
Popis	Umožňuje obdržet parametr y*_cell.*
Volání	sys pGetYcell(Id);
Návratová hodnota	y_cell (num) – parametr y*_cell*
Parametry	id (num) – id obrazovky

Tabulka 144: Popis funkce viz tabulka (jazyk SVS)

Název	pSetYcell
Popis	Umožňuje nastavit parametr *y_cell.*
Volání	sys pSetYcell(id, val);
Parametry	id (num) – id obrazovky val (num) – nová hodnota parametru

Tabulka 145: Popis funkce viz tabulka (jazyk SVS)

8.4 CAN wrapper pro SVS

Poslední součástí softwarového řešení je wrapper pro SVS umožňující nastavení hlavní obrazovky aplikace, popisku aplikace a odesílání CAN zpráv.

Název	pSetMainScr
Popis	Umožňuje nastavit hlavní obrazovku aplikace.
Volání	sys pSetMainScr(id);
Parametry	id (num) – identifikátor obrazovky

Tabulka 146: Popis funkce viz tabulka (jazyk SVS)

Název	pSetScrName
Popis	Umožňuje nastavit popisek obrazovky.
Volání	sys pSetScrName(name);
Parametry	name (txt) – popisek obrazovky

Tabulka 147: Popis funkce viz tabulka (jazyk SVS)

Název	canSetId
Popis	Nastaví id pro odesílané zprávy.
Volání	sys canSetId(id, ext_id);
Parametry	id (num) – id zprávy ext_id (num) – extId zprávy, pokud je rovno nule, je použito jen standardní id

Tabulka 148: Popis funkce viz tabulka (jazyk SVS)

Název	canSetRTR
Popis	Umožňuje nastavit RTR příznak odesílané zprávě.
Volání	sys canSetRTR(rtr);
Parametry	rtr (num) – pokud 1, je nastaven příznak RTR

Tabulka 149: Popis funkce viz tabulka (jazyk SVS)

Název	canSend
Popis	Odešle zprávu přes rozhraní CAN.
Volání	sys canSend(len, d0, d1, d2, d3, d4, d5, d6, d7);
Parametry	len – délka zprávy <0;8> d0 … d7 – data zprávy

Tabulka 150: Popis funkce viz tabulka (jazyk SVS)

8.5 Software: Shrnutí a vývojový proces

Zadání specifikuje aby firmware mikrokontroléru umožňoval definování CAN zpráv prvkům grafického rozhraní. Grafické rozhraní je řešeno již popsanou knihovnou GR2. Konfigurace grafických prvků je řešena konfiguračním skriptem SVS s wrapperem pro knihovnu GR2. Odesílání zpráv po sběrnici CAN zajišťují HAL funkce z knihoven dodávaných výrobcem užitého mikrokontroléru (STM) obalené wrapperem pro SVS. Takto vypadá "softwarový stack" na straně firmware, přenechává uživateli kompletní kontrolu nad podobou grafického rozhraní a formátem komunikace s audiosystémem, to vše dostatečně rychle a stabilně.

Tato kapitola se bude věnovat předpokládanému procesu nastavení grafického rozhraní finální aplikace. Návrhu tohoto rozhraní, jeho implementaci, testování a nasazení. Protože dodat softwarové řešení bez vývojových postupů nástrojů je špatné.

8.5.1 Návrh Rozhraní

Prvotní myšlenkou uživatelského rozhraní musí být vždy jeho účel (úkol jež má plnit). Po ujasnění jeho účelů je nutno přistoupit k specifikaci "výrazových prostředků", k otázce konkrétního využití konkrétních prvků pro ten který účel.

Příklad: Chceme mít možnost přepínat mezi dvěma presety audiosystému odesláním dvou zpráv. Zpráva A nastaví audiosystému preset P1, zpráva B nastaví preset P2. Tohoto výsledku můžeme dosáhnou jedním tlačítkem, jež bude formou přepínače přepínat mezi presety. Stejně tak můžeme tohoto výsledku dosáhnout dvěma tlačítky.

Poté co si rámcově ujasníme filosofii ovládání naší aplikace, můžeme přistoupit dalšímu kroku návrhu a tím je vizualizace.

8.5.2 Vizualizace

Nejjednodušším a silně doporučovaným vizualizačním nástrojem jest papír a tužka. Rozkreslení jednotlivých obrazovek aplikace pomůže s ujasněním si vzájemných vazeb mezi prvky a odhalí celkovou strukturu aplikace.

Za vizualizační nástroj je možné považovat také dále popsaný simulátor řídící jednotky, jež umožní zobrazit aplikaci přesně tak jako řídící jednotka.

8.5.3 Implementace

Implementace funkcionality spočívá ve vytvoření konfiguračního skriptu. S jazykem SVS a systémovými funkcemi jste již seznámeni z předchozích kapitol, přejdeme tedy

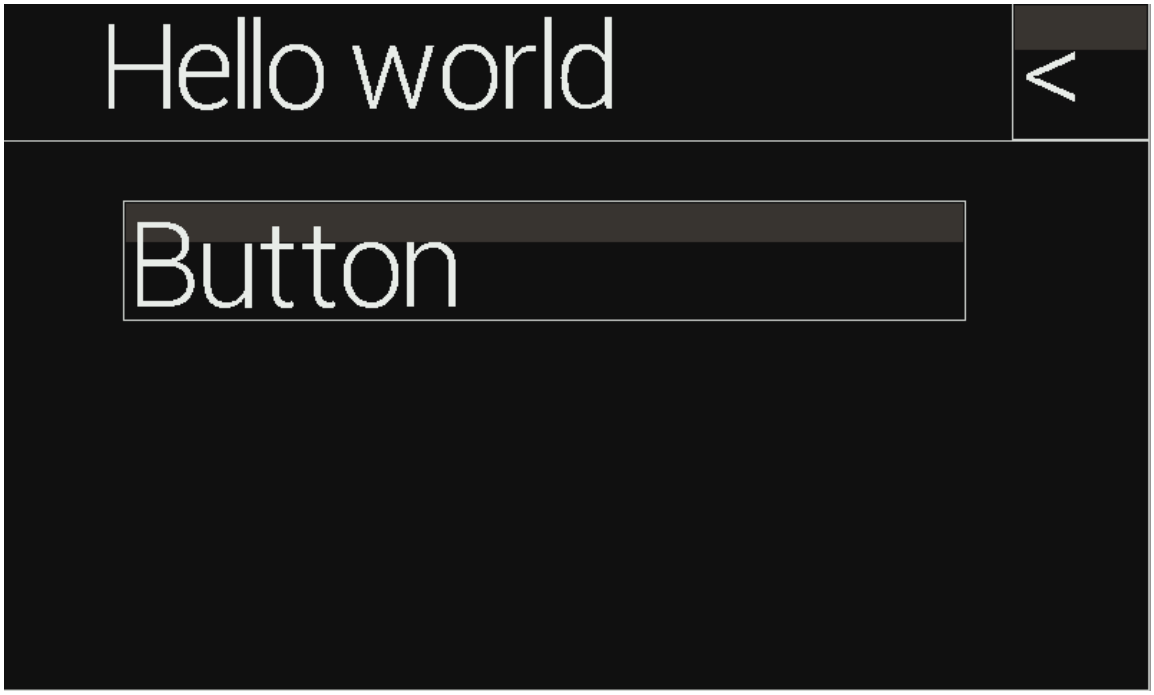
k praktickým částem implementace. Ke tvorbě konfiguračního souboru potřebujeme textový editor nastavený tak, aby kódování znaků bylo provedeno v **UTF-8** a ukončení řádku bylo **UNIXového typu** (pouze newline).

Dále budeme si implementaci ukážeme na ukázkových programech. Prvním z nich bude hello world. V textovém editoru nastaveném dle předchozího odstavce vytvoříme soubor *hello.svs* s následujícím obsahem.

```
function init {
  #konstruktory elementů
  screen = sys pAddScreen();

  button = sys pAddButton( 2, 1, 16, 3, "Button", screen);

  #nastavení hlavní obazovky
  sys pSetMainScr(screen);
  sys pSetScrName("Hello world");

  #nastavení CAN zprávy
  sys canSetId(600 , 0 );
  sys canSetRTR(0);
}

function update {
  #obsluha tlačítka
  if (sys pGetEvent(button)==3){
    sys canSend(3 ,1,2,3,0,0,0,0,0);
    sys pSetEvent(button,0);
  }
}
```

Konfigurační soubor je rozdělen na dvě funkce, funkci *init* a funkci *update*. Funkce *init* je provedena po zavedení programu, funkce *update* je prováděna periodicky. Ve funkci *init* vytvoříme obrazovku, jejíž *id* uložíme do proměnné *screen* a tlačítko, jehož *id* uložíme do proměnné *button*. Poté nastavíme hlavní obrazovku na naší obrazovku a nastavíme její popisek na textovou konstantu „Hello world“.

Funkce update je určena k aktualizaci prvků a provádění akcí na základě uživatelských událostí. Obsluha v ukázce nejprve ověří, zda událost tlačítka je 3 (EV_RELEASED), poté odešle po sběrnici CAN tři byty hodnot 1, 2 a 3. Nakonec dojde k vymazání události tlačítka.

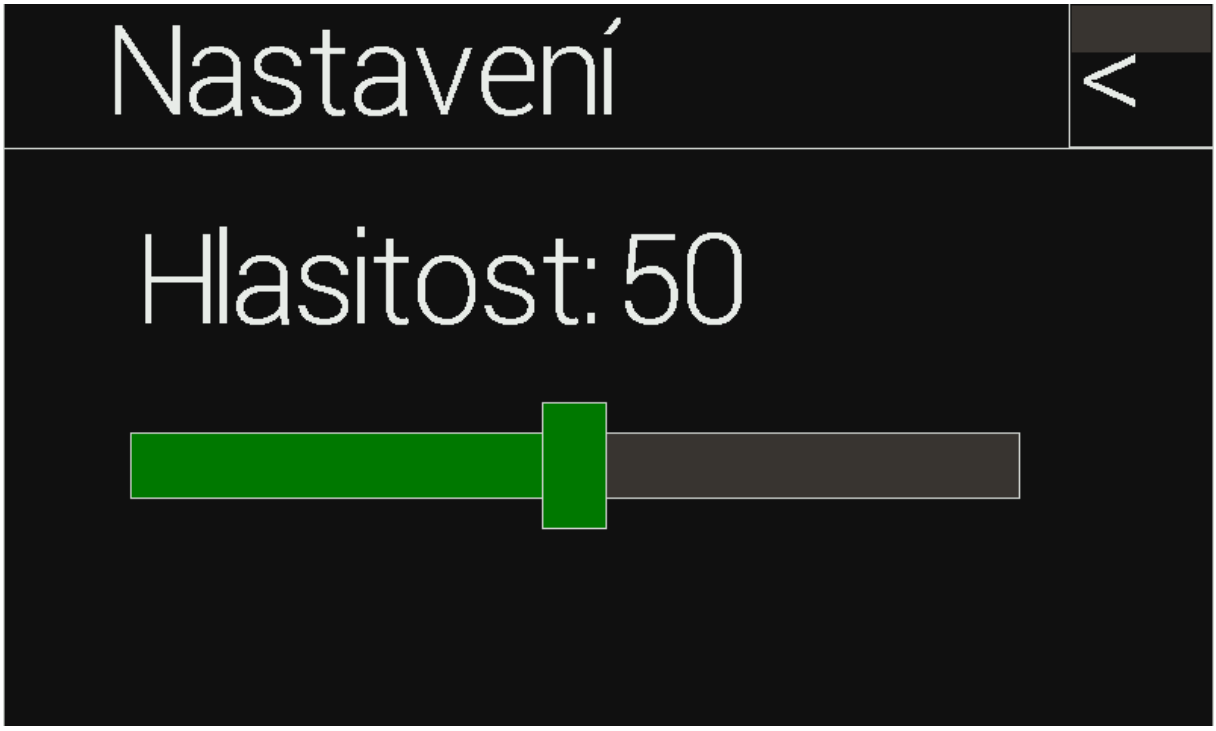


Obr. 10: Ukázka hello.svs

Další typickou úlohou je implementace posuvníku. V obsluze si můžeme povšimnout aktualizace textového popisku hlasitosti.

```
function init{
  screen = sys pAddScreen();
  text = sys pAddText(2, 1, 16, 3, "Hlasitost: "+50, screen);
  slider = sys pAddSliderH(2, 4, 16, 6, 100, 50, screen);

  sys pSetMainScr(screen);
  sys pSetScrName("Nastavení");

  sys canSetId(600 , 0 );
  sys canSetRTR(0);

  sld_old=sys pGetValue(slider);
}

function update{

  if(sys pGetEvent(slider)){
    if(sys pGetValue(slider)!=sld_old){
      sld_old=sys pGetValue(slider);
      sys canSend(3 ,0xAA,0xBB,sys pGetValue(slider),0,0,0,0,0);
      #aktualizujeme hodnotu textového pole
      sys pSetStr(text, "Hlasitost: "+sys pGetValue(slider));
    }
    sys pSetEvent(slider,0);
  }

}
```

Obsluha nejprve ověří událost posuvníku, poté ověří, zda se hodnota posuvníku skutečně změnila a pokud jsou tyto podmínky pravdivé, odešle zprávu a aktualizuje textový popisek.



Obr. 11: Ukázka posuvníku (slider.svs)

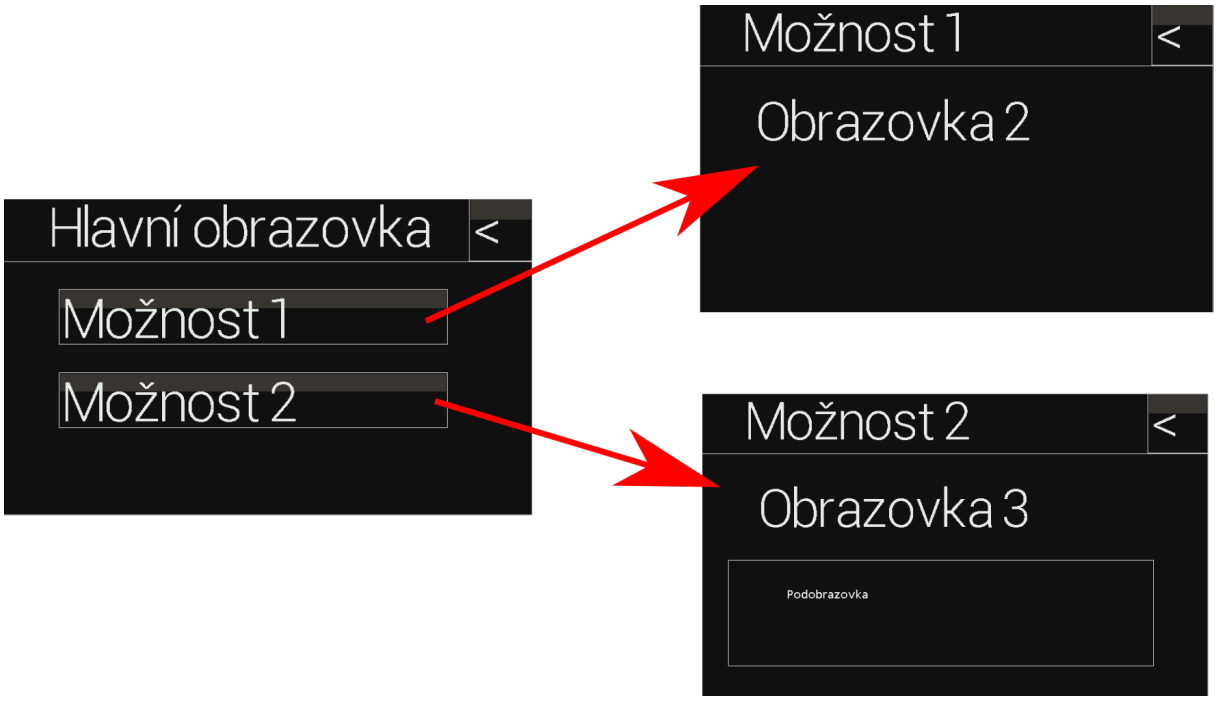
Posledním ukázkovým programem je program s více obrazovkami a do sebe zanořenými pod-obrazovkami. Na tomto příkladu si můžete povšimnout, že na pořadí v jakém jsou inicializovány grafické prvky nezáleží, jejich vykreslení je vždy podmíněno jen identifikátorem jejich obrazovky a pozicí na této obrazovce. Obrazovky jsou také jen dalším z prvků a je možno je přidat na existující obrazovku. Přecházení mezi obrazovkami je realizováno přepínáním hlavní obrazovky. Při tom je možno změnit popisek obrazovky. O návrat na předchozí obrazovku se stará tlačítko zpět v pravém horním rohu.

```
function init{
  #konstruktory obrazovek
  screen = sys pAddScreen();
  screen2 = sys pAddScreen();
  screen3 = sys pAddScreen();
  subscreen = sys pAddScreen();

  #nastavení hlavní obrazovky
  sys pSetMainScr(screen);
  sys pSetScrName("Hlavní obrazovka");

  #prvky hlavní obrazovky
  btn1 = sys pAddButton( 2, 1, 16, 3, "Možnost 1", screen);
  btn2 = sys pAddButton( 2, 4, 16, 6, "Možnost 2", screen);

  #textové prvky rozlišující obrazovky
  sys pAddText( 2, 1, 16, 3, "Obrazovka 2", screen2);
  sys pAddText( 2, 1, 16, 3, "Obrazovka 3", screen3);

  #nastavení podobrazovky
  sys pSetXYXY(subscreen, 1, 4, 17, 8);
  sys pSetScreen(subscreen, screen3);

  #vytvoření nového textového prvku s nastavením velikosti textu
  sys pSetTxtSize(sys pAddText( 2, 1, 16, 3, "Podobrazovka", subscreen),
18);

}

function update {
  if (sys pGetEvent(btn1)==3){
    sys pSetMainScr(screen2);
    sys pSetScrName("Možnost 1");
    sys pSetEvent(btn1,0);
  }

  if (sys pGetEvent(btn2)==3){
    sys pSetMainScr(screen3);
    sys pSetScrName("Možnost 2");
    sys pSetEvent(btn2,0);
  }
}
```

V ukázce je též použita modifikace velikosti textu textového pole, aniž by se ukládal identifikátor textového pole do nějaké proměnné. Velikost fontu není možno nastavit na libovolnou hodnotu, fonty jsou dány během kompilace a firmware řídící jednotky obsahuje velikosti fontů viz tabulka:

Velikosti fontů		
18	32	87

Tabulka 151: Velikosti fontů



Obr. 12: Ukázka programu (screens.svs)

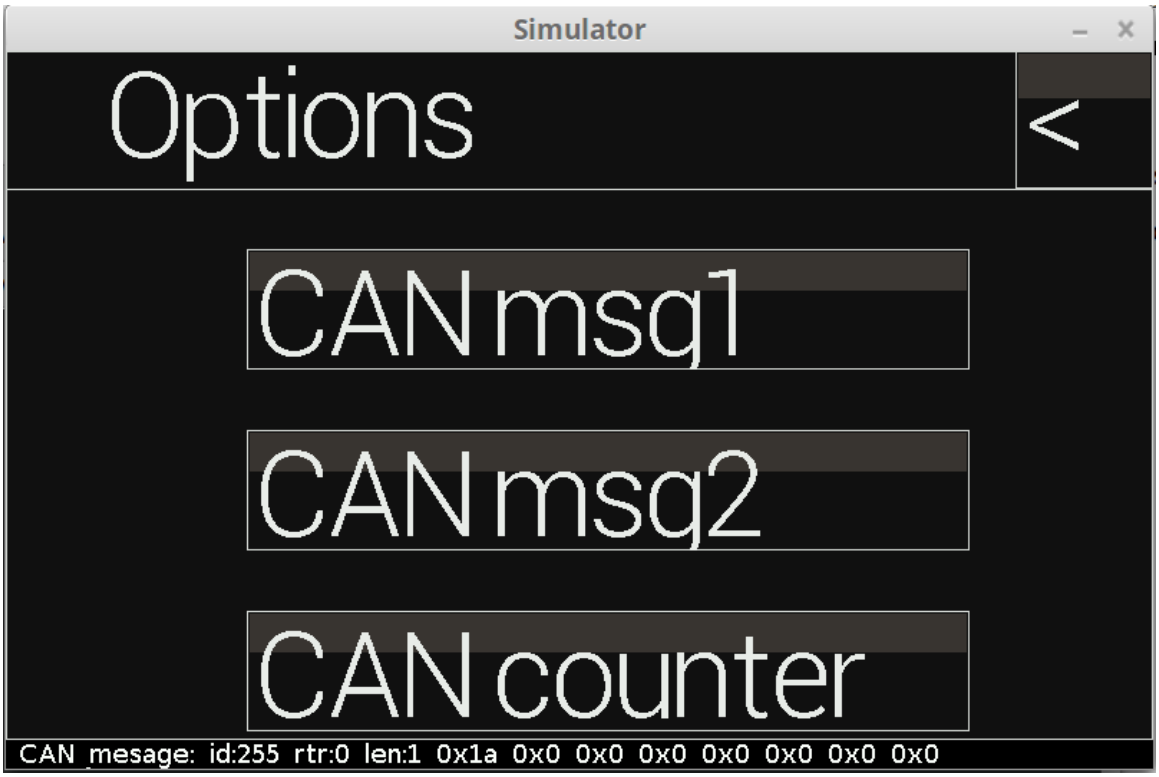
Na CD přiloženém k diplomové práci jsou soubory všech tří ukázkových programů. Dále jsou zde navíc ukázkové programy z předchozí verze diplomové práce.

8.5.4 Simulace

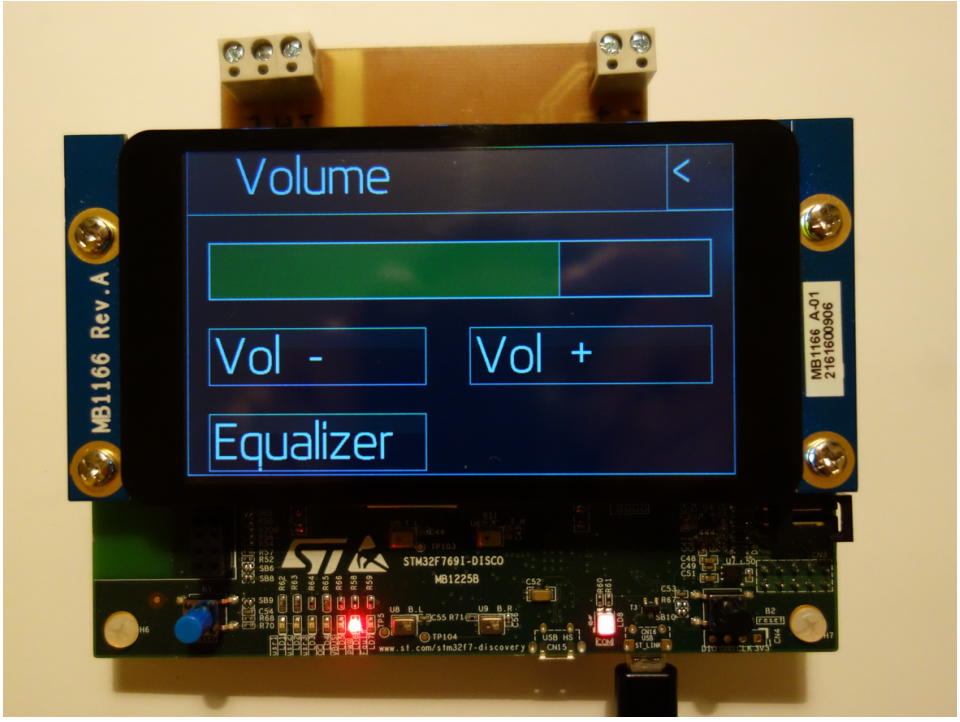
Čtenářovu pozornému oku jistě neušlo, že snímky obrazovky na uvedených příkladech nejsou fotografiemi displeje řídící jednotky. Jedná se o snímky obrazovky ze simulátoru, kde programové vybavení řídící jednotky neběží nad knihovnami mikrokontroléru, ale nad knihovnou SDL2 na PC. Tento simulátor umožňuje otestovat konfiguraci řídící jednotky v pohodlí PC, bez nutnosti nahrávání konfigurace do zařízení. Kompatibilita není stoprocentní, ale je dostatečná pro průběžné testování aplikace.

Simulátor využívá snadné přenositelnosti grafické knihovny GR2 a skriptovacího jazyka SVS. Existují verze pro GNU/Linux i MS Windows. Simulátor fungoval bez problémů na všech testovaných počítačích, tedy krom počítače určeného k prezentaci diplomových prací během Státní závěrečné zkoušky, ale tam nefungoval (kupodivu) ani powerpoint. Každopádně od tohoto debaklu byl simulátor těžce optimalizován a nyní by měl fungovat lépe. Minimální požadavky simulátoru jsou MS Windows XP a vyšší, v případě GNU/Linuxu vyžaduje simulátor knihovnu SDL2 a běžící X server.

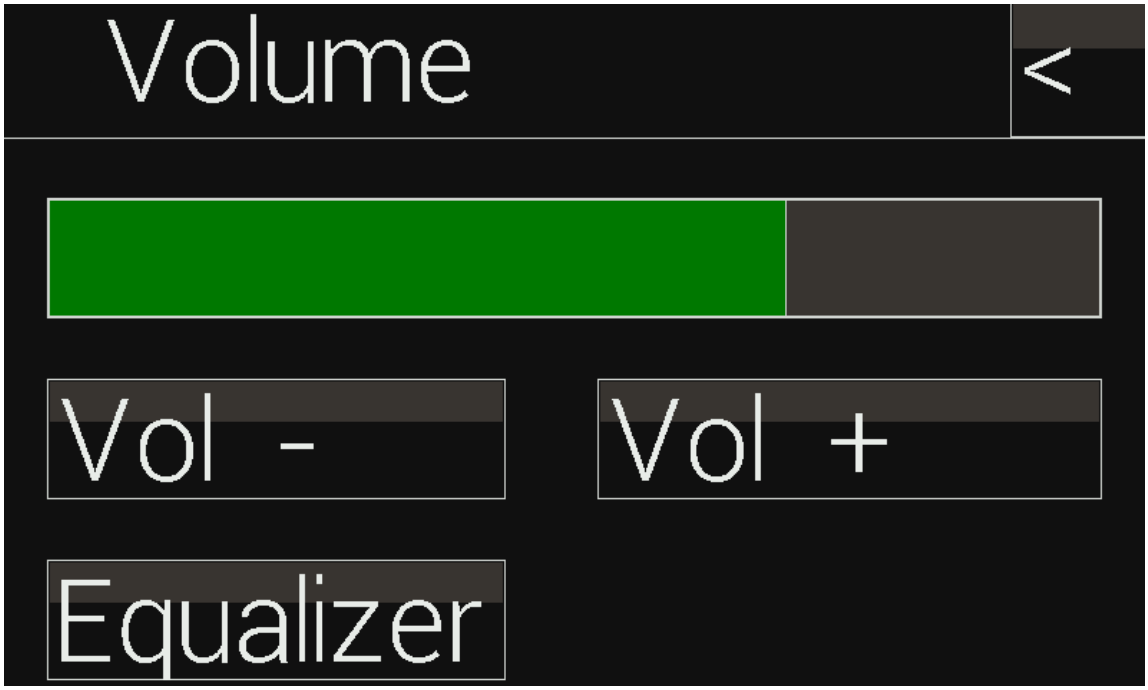
Ideální je spustit simulátor z příkazové řádky, aby byl vidět případný chybový výstup. Prvním parametrem simulátoru je název svs souboru, jež má být zaveden. Simulátor ve spodní části okna vypisuje poslední CAN zprávu jež byla simulátorem jakoby odeslána.



Obr. 13: Ukázka rozhraní simulátoru.



Obr. 14: Ukázka běhu zařízení



Obr. 15: Ukázka té samé obrazovky v simulátoru.

Simulátor tedy použijeme ke snadnému odladění chyb v konfiguraci řídící jednotky. Spuštění simulátoru vypadá z příkazové řádky například takto:

```
D:\DP-Win>DP_sim.exe hello.svs
```

Samozřejmě je možné si ke spouštění simulace vytvořit *.bat* soubor, na přiloženém CD je takových souborů několik.

Po dostatečném odladění konfigurace v simulátoru můžeme přejít k nasazení konfigurace na řídící jednotku.

8.5.5 Nasazení

Řídící jednotka je vybavena slotem na microSD kartu. Takovouto kartu (testovány byly karty SD a SDHC, tzn. do 32 GB) je nutno naformátovat na systém souborů FAT32. Na takto připravenou kartu nakopírujeme konfigurační soubor a přejmenujeme jej na **app.svs** Kartu není nutno formátovat pokaždé, ale v případě výskytu neočekávané chyby je dobré začít naformátováním karty.

Jednotka po startu, případně restartu automaticky tento soubor načte a vykoná. V případě výskytu problémů, jež se neprojevily v simulátoru, je nutno připojit se pomocí PC k řídicí jednotce na její sériový port, nastavený na 9600 bd a zkontrolovat případný chybový výpis.

9 Závěr

Využití vývojového kitu se oplatilo zrychlením vývoje hardware i software. Hardwarová část diplomové práce není příliš rozsáhlá ani robustní a existuje zejména z důvodu ověření funkce software.

Software vyvinutý v rámci této diplomové práce je jejím stěžejním bodem. Jedná se o grafickou knihovnu GR2 a skriptovací jazyk SVS.

Grafická knihovna GR2 je na první pohled jen další z grafických knihoven pro mikrokontroléry, kterých je na trhu mnoho. Na druhý pohled má některé zajímavé funkce a vlastnosti, kterými se dokáže vyrovnat dalším podobným produktům. Jedná se zejména její snadnou přenositelnost, nasaditelnost a možnost propojení s knihovnou SDL2 umožňující rychlé prototypování grafického rozhraní embedded aplikace. Podpora fontů s dynamickou šířkou znaku a české diakritiky v kódování UTF8 jsou další výhody knihovny GR2. Na druhou stranu je značně limitující, že knihovna vnitřně pracuje s 16bit barvami namísto 24bit. Dalším problémem je absence grafického konfigurátoru grafického rozhraní (interface builder) pro GR2.

Skriptovací jazyk SVS si klade v této aplikaci za úkol usnadnit nastavení grafického rozhraní a to nejen co se týče předem definovaných zpráv grafických prvků, ale i celkové logiky a funkcionality aplikace. Ve světě skriptovacích jazyků fungujících na mikrokontrolérech se řadí na ne úplně obvyklou pozici jazyka, jehož místo v aplikaci je až na samotném vrchu, skripty tedy teoreticky nejsou příliš závislé na konkrétním hardware. Správa hardware je přenechána nativním systémovým funkcím, což vede teoreticky k vyššímu výkonu. Prakticky je tento výkon zpětně použit ke zjednodušení implementace, případně k zjednodušení práce se skriptem.

Celé toto softwarové řešení umožňuje nejen rychlé prototypování výsledné aplikace, ale hlavně rychlé prototypování s minimem aplikačně-specifických softwarových nástrojů, což je v embedded světě málo vídané. V celé kapitole 8.5, věnované návrhu, testování a nasazení konfigurace řídící jednotky například není ani slovo o jakékoli kompilaci, jediný nezbytně nutný nástroj je textový editor. Výslednou robustností se tento sytém samozřejmě nemůže vyrovnat komerčnímu řešení, ovšem je málo které komerční řešení lze nasadit pouze textovým editorem a čtečkou karet. Zda je to důležitá výhoda a nebo zbytečnost, to ať posoudí čtenář sám.

10 Seznam literatury a informačních zdrojů

[1] *GRIES, David. Compiler Construction for Digital Computers.* John Wiley and Sons, New York, 1971. ISBN-13: 978-0471327769.

[2] *RICHTA, K., BRŮHA I. Programovací jazyk C*. Ediční středisko ČVUT, Praha, 1991. ISBN 80-01-00704-9.

[3] *OLEHLA, J., OLEHLA M. a kol. BASIC u mikropočítačů.* NADAS, Praha, 1988, ISBN 31-044-88.

[4] Katalogový list integrovaného obvodu Linear Technology LT1375. *K dispozici na přiloženém CD.*

[5] Katalogový list integrovaného obvodu Analog Devices ADM3053. *K dispozici na přiloženém CD.*

[6] Katalogový list integrovaného obvodu STM STM32F769. *K dispozici na přiloženém CD.*

[7] Schéma vývojového kitu STM32F769I-DISCO. *K dispozici na přiloženém CD.*

Přílohy

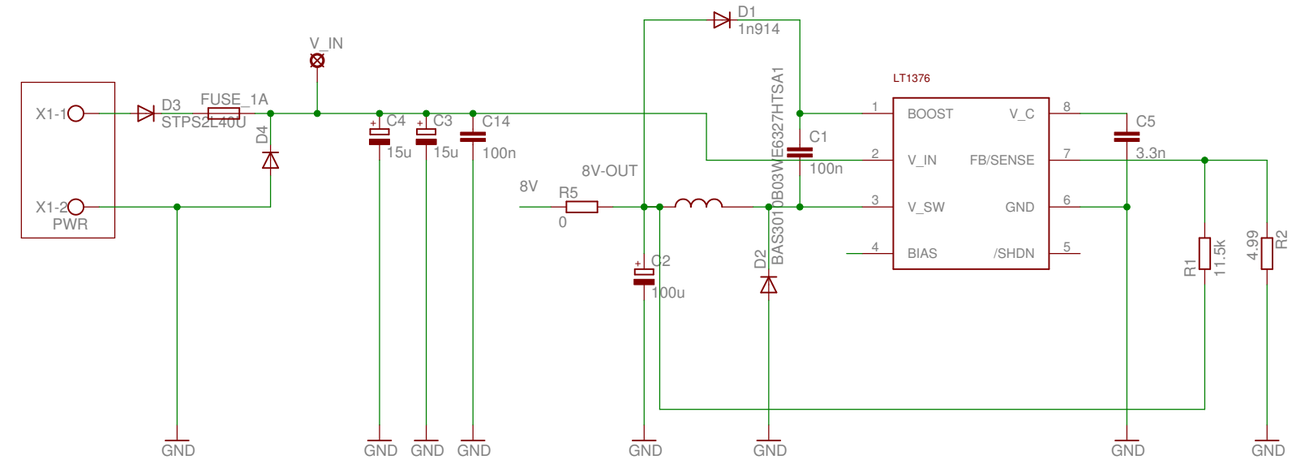
Obsah přiloženého CD:

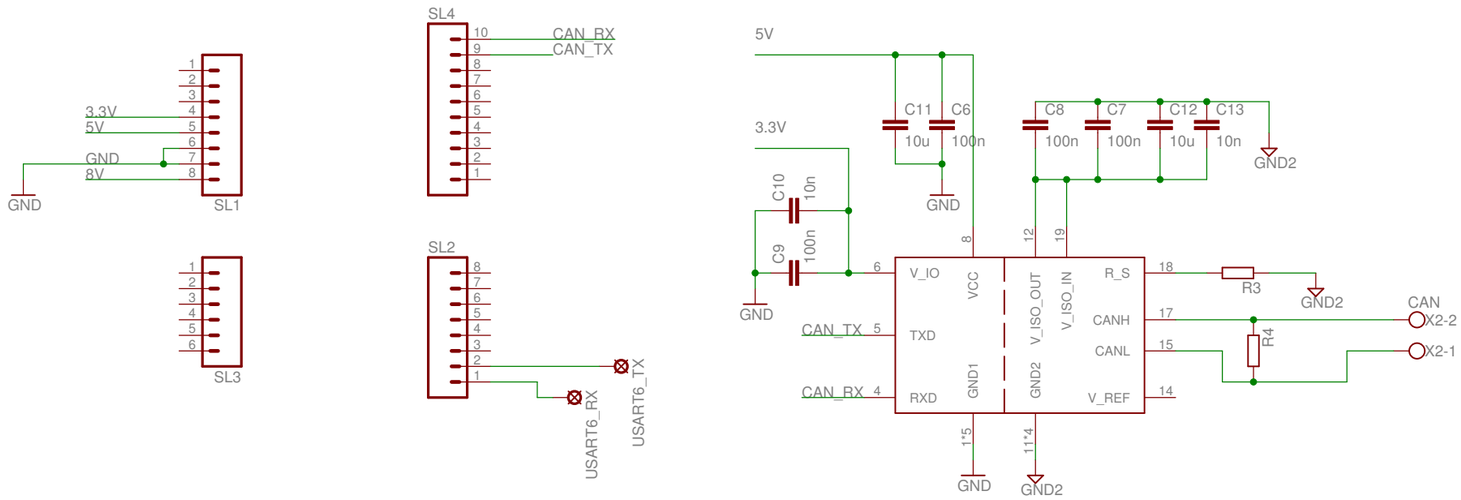
- Zdrojové kódy
  - zdrojový kód aplikace
  - zdrojový kód simulátoru pro PC
  - příklady použití konfiguračního souboru
- Podklady k výrobě DPS, ve formátu EAGLE
- Tato práce ve formátu PDF
- Katalogové listy použitých součástek ve formátu PDF

Příloha 1:
Filmová předloha použitá k osvitu DPS.

(pokud se zde nenachází, tak je ve druhém výtisku této diplomové práce)

Příloha 2:
Kompletní schéma zapojení





STM32F769I-DISCO CAN & POWER Expansion

Stanislav Brtna 2017