KATEDRA
ELEKTROTECHNOLOGIE

ČESKÉ VYSOKÉ UČENÍ
TECHNICKÉ V PRAZE

FAKULTA ELEKTROTECHNICKÁ

VZOROVÉ PŘÍKLADY PRO ŠKOLNÍ MIKROPOČÍTAČ

BAKALÁŘSKÁ PRÁCE

BŘEZEN 2023

MILOŠ
MLEJNEK

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Mlejnek** Jméno: **Miloš** Osobní číslo: **501423**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra elektrotechnologie**

Studijní program: **Elektrotechnika, energetika a management**

Specializace: **Aplikovaná elektrotechnika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Vzorové příklady pro školní mikropočítač

Název bakalářské práce anglicky:

Sample programs for the school kit

Pokyny pro vypracování:

1. Seznamte se s konstrukcí a vybavením školního mikropočítače.
2. Navrhněte koncepci programového vybavení.
3. Zpracujte a ověřte funkci hardwarového rozhraní
4. Navrhněte, zpracujte a ověřte vzorové příklady pro použité periférie.
5. Vývoj realizujte s potporou verzovacího systému.
6. Zpracujte dokumentaci s využitím vhodného nástroje pro generování dokumentace.

Seznam doporučené literatury:

[1] STM, „RM0016 Reference manual STM8S and STM8A microcontroller families". STMicroelectronics, 2011. https://www.st.com/resource/en/reference_manual/cd00190271-stm8s-series-and-stm8af-series-8bit-microcontrollers-stmicroelectronics.pd
[2] Scott Chacon, Pro Git, První. Praha: CZ.NIC, z. s. p. o. [Online]. Available: http://www.root.cz/knihy/pro-git/
[3] Doxygen. Documentation https://www.doxygen.nl/manual/index.html

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Karel Künzel, CSc. katedra elektrotechnologie FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **09.02.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

Ing. Karel Künzel, CSc. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

PODĚKOVÁNÍ

Rád bych poděkoval svému vedoucímu Ing. Karlu Künzelovi, CSc. za jeho nápomocnost, ochotu a čas který mi při vypracování bakalářské práce věnoval.

PROHLÁŠENÍ

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 25. května 2023

.....

ABSTRAKT

Bakalářská práce se zabývá školním mikroprocesorovým přípravkem určeným pro výuku mikroprocesorů. Jako ukázka pro práci s mikroprocesory jsou zpracovány vzorové příklady pro jednotlivé periferie mikroprocesorového přípravku. Při zpracování jednotlivých úloh bylo využito verzovacího systému a nástroje pro generování dokumentace.

Klíčová slova: mikroprocesor, verzovací systém, generování dokumentace, analogově číslicový převodník, motorek, I2C, UART, SPI, reproduktor

ABSTRACT

The Bachelor thesis deals with school microprocessor kit designed for teaching microprocessors. Sample examples for individual microprocessor peripherals are processed as a demonstration of microprocessor work. During the processing of individual tasks, a version control system and documentation generation tools were used.

Keywords: microprocessor, version control system, documentation generation, analog-to-digital converter, motor, I2C, UART, SPI, speaker

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Tato bakalářská práce se zabývá školním mikroprocesorovým přípravkem určeným pro výuku mikroprocesorů. Cílem práce je návrh a vytvoření vzorových příkladů pro jednotlivé periferie školního mikropočítače s využitím podpory verzovacího systému a nástroje pro generování dokumentace. Možnosti mikropočítače udává konstrukce a vybavení, se kterými je nezbytné se při práci seznámit. Vzorové příklady mají demonstrovat princip a využití jednotlivých periferií. Využití verzovacího systému umožnuje uchování historie změn souboru nebo sady souborů v průběhu času a následné navrácení ke kterékoli verzi s následným rozvíjením dané verze v nové větvi. Vygenerování dokumentace pro jednotlivé vzorové příklady poskytuje přehledný popis programů umožňující snazší pochopení řešené problematiky.

KAPITOLA 1: ANALÝZA PROBLÉMU

1.1 Vývojové kity

Vývojový kit je přípravek pro vývoj a testování aplikací nebo hardwaru. Tyto přípravky jsou dodávány k mikroprocesorům od samotných výrobců mikroprocesorů nebo vznikají nezávisle na výrobci součástek. V komerčním světě jsou využívány jako přechodné řešení mezi původní myšlenkou a prvním prototypem elektrického zařízení. Vývojové kity jsou zároveň velmi užitečné pro výuku mikroprocesorů kvůli testování programů na reálných zařízeních. Reálná zařízení poskytují okamžitou zpětnou vazbu na vytvořený program v různých podobách. Nejjednodušší zpětnou vazbou může být pouhé blikání LED. Studenti tak získají zkušenosti s prací s periferiemi, které poskytuje vývojový kit jako například tlačítka, potenciometry, reproduktor nebo inkrementální čidlo.

Pro výuku mikroprocesorů mohou být využity vývojové kity od Arduino, Raspberry PI, PIC mikrokontrolérů, ARM Cortex nebo STM. Volba mezi osmi bitovými a třiceti dvou bitovými mikroprocesory záleží na konkrétním vzdělávacím cíli. Třiceti dvou bitové mikroprocesory jsou výkonnější a poskytující tvorbu složitějších projektů. Osmi bitové mikroprocesory jsou vhodnější pro jednodušší projekty a poskytují snazší pochopení práce s mikroprocesory. Někteří výrobci používají stejné periferie pro osmi a třiceti dvou bitové mikroprocesory, což zjednodušuje přechod pro práci z osmibitových na třiceti dvou bitové mikroprocesory.

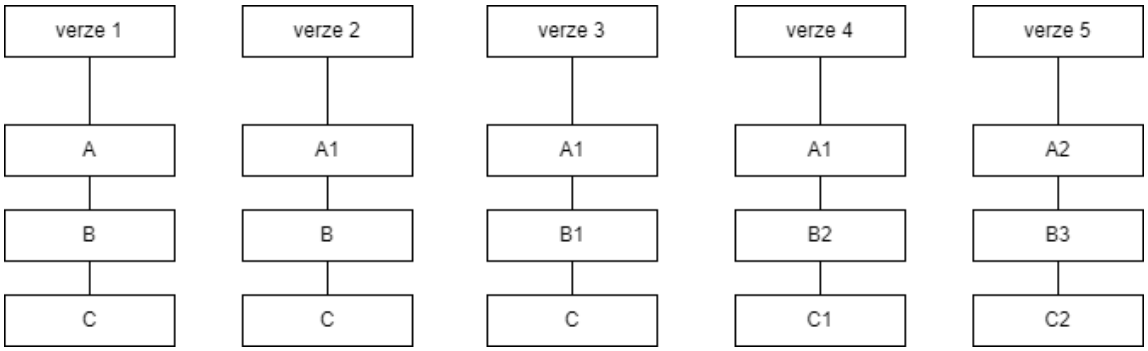
1.2 Zálohování a verzování

Během práce na nejrůznějších projektech s mikroprocesory může dojít ke ztrátě dosud vytvořené práce nebo ke změně v projektu, která může narušit chod programu a návrat k funkční verzi projektu bývá velmi náročný. Kvůli nenávratným ztrátám souborů se používají zálohovací programy. Zálohování je vytvoření kopie dat uložené na jiném zařízení nebo úložišti.

Verzovací systémy umožňují návrat do historie provedených změn v programech. Uživatel může následně danou historickou verzi využít a vytvořit novou větev programu s jiným přístupem k řešení daného problému. Představiteli verzovacích systémů jsou například CVS, Apache Subversion, Git, Bazaar nebo Mercurial. Pro širokou podporu a bezplatnost je v této práci využit verzovací systém Git.

1.2.1 Git

Git je verzovací systém umožňující spolupráci na projektu v týmu, zálohování projektu, uchování všech verzovaných souborů s přehledem o práci jiných spolupracovníků. Uživatelé si soubory stahují z centrálního úložiště do svých pracovních adresářů a v případě kolapsu serveru se pro obnovení zkopíruje repozitář od libovolného uživatele zpět na server. Git bere data jako sadu snímků, kdy při uložení projektu dojde k uložení snímků všech souborů a uložení reference na dané snímky. V případě uložení bez změny souborů dojde pouze k uložení odkazu na předchozí identický soubor. Celá historie projektu je uložena na lokálním disku, což zaručuje velkou rychlost prováděných operací. *[1]*

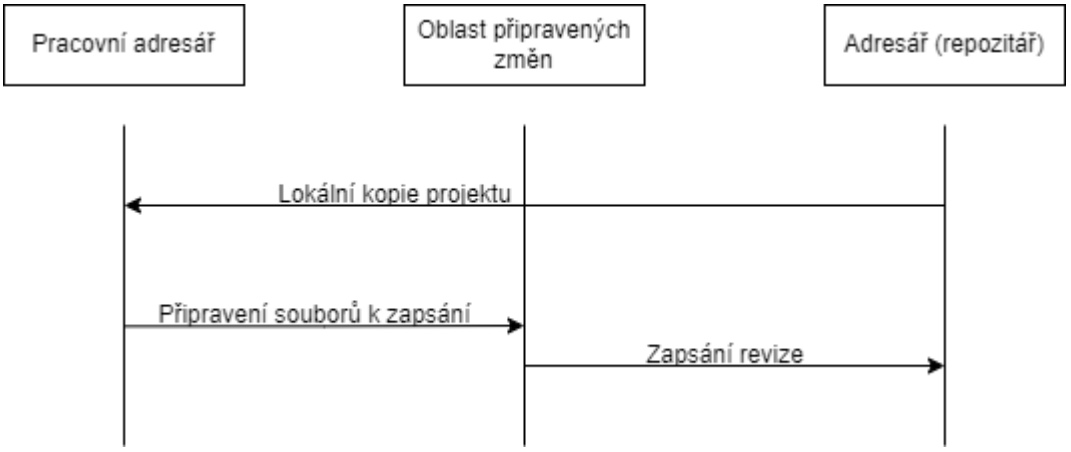


Obr. 1-1 Git – ukládání snímků

V první verzi projektu byly vytvořeny tři soubory A, B, C. V druhé verzi projektu byl upraven soubor A na A1 a dva zbylé soubory B, C byly zanechány bez změny. Ve verzi tři byl změněn soubor B na B1 a zbylé dva soubory A1 a C byly zanechány bez změny. Takovým to způsobem probíhá ukládání změn v každé další verzi projektu.

Pro náhled změn provedených v souboru provede Git lokální výpočet rozdílů mezi aktuální verzí souboru a vybraného minulého souboru. Další výhodou je ukládání revizí bez připojení k internetu. Načtení změn do systému proběhne po následném připojení k internetu. Před jakýmkoli uložením do systému Git proběhne součet, který umožní identifikaci dané operace. Toto zamezuje ztrátě jakékoliv informace. Tento součet se nazývá otisk SHA-1 a je to řetězec čtyřiceti hexadecimálních znaků. Git ukládá soubory do své databáze podle těchto otisků.

Pro spravované soubory jsou využívány tři stavy: zapsáno (commited), změněno (modified) a připraveno k zapsání (staged). Stav zapsáno znamená uložení dat v lokální databázi. Stav změněno má význam provedení změn bez zapsání do databáze. Stav připraveno k zapsání má význam změněného souboru určeného k zapsání v další revizi zapsáno. *[1]*

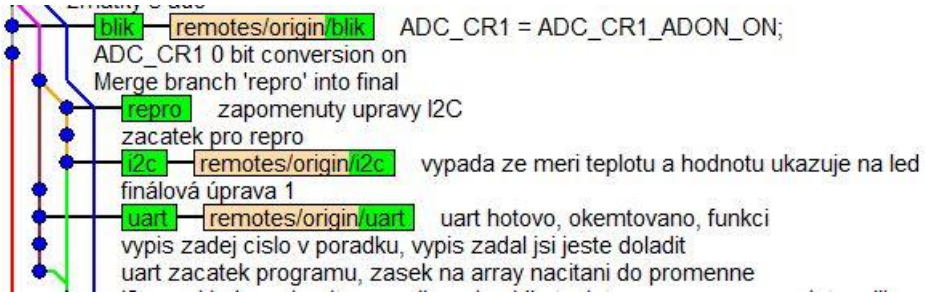


Obr. 1-2 Git – stavy spravovaných souborů

Pro práci s Gitem se používá příkazový řádek Git, grafické rozhraní Git Gui nebo Git Bash. Git Gui usnadňuje uživateli práci tím, že není potřeba výpisu jednotlivých příkazů pro požadovanou činnost. Příklady základních příkazů pro Git příkazový řádek jsou uvedeny níže. *[1]*

- ***git add jméno souboru***: příkaz pro sledování nových souborů, tedy sledovaný soubor je připraven k zapsání a pokud v tomto okamžiku dojde k zapsání revize bude v historickém snímku verze souboru po spuštění git add
- ***git commit –m "popis změny souboru"****:* tento příkaz zapisuje změny souborů, které byly připraveny k zapsání příkazem git add, příznak –m umožňuje napsat zprávu o revizi do uvozovek

- ***git checkout –b jméno větve***: příkaz pro vytvoření nové větve projektu s přepnutím na novou větev
- **git push:** příkaz pro zapsání na vzdálený repozitář
- ***git status:*** příkaz pro zjištění stavu jednotlivých souborů, dojde k vypsání souborů připravených k zapsaní a souborů, které byly změněné, ale nedošlo u nich k připravení k zapsání
- ***git log:*** příkaz umožňující výpis provedených revizí v repozitáři v chronologickém pořadí
- ***gitk –all:*** příkaz pro zobrazení přehledu všech větví



Obr. 1-3 Git – větve a komentáře k jednotlivým revizím

Na obrázku lze vidět jednotlivé větve projektu zvýrazněné zelenou barvou. Zároveň si lze povšimnout komentářů pro každou revizi.

1.3 Tvorba dokumentace

Neoddělitelnou součástí práce na jakémkoli projektu je tvorba dokumentace. Tvůrce programu by měl zdokumentovat jednotlivé kroky své práce pro zpětné porozumění řešení a další vývoj. V případě nedodržení tohoto pravidla by se práce s nezdokumentovaným projektem dala přirovnat k luštění hádanky pro nalezení myšlenky schované v řádcích nesrozumitelného programu. Takovéto hledání myšlenky řešení stojí mnohdy mnohem více času než vytvoření zcela nového řešení. Ve zdokumentovaném programu se také snadněji hledají chyby v projektu. Dále je také klíčová změna dokumentace při změně programu po porozumění novějšího řešení. Dodržení těchto pravidel poskytuje efektivní práci v projektu a možnost rozvíjení programu. Pro dokumentování projektů se využíváno mnoho nástrojů pro tvorbu dokumentace. Příkladem mohou být Sphinx, Docurium, Natural Docs nebo Doxygen.

1.3.1 Doxygen

Doxygen je nástroj umožňující tvorbu dokumentace pomocí specifických komentářů. Dokumentace je možná v široké škále programovacích jazyků jako C, C#, C++, Java, Python a další. Zároveň je možné komentáře psát v několika mateřských jazycích. Pro náhled na zdokumentovaný projekt lze dokumentaci exportovat do mnoha formátů HTML, Latex, RTF, Objective-C, D, IDL, VHDL, PHP, Tcl nebo Fortan. Pro snadné zorientování v projektu je možná tvorba grafů a diagramů pomocí Graphviz. Pro editaci konfiguračního souboru lze využít grafické rozhraní Doxywizard.

Obr. 1-4 Doxygen – výstupní HTML formát

Pro dokumentaci se používají specifické komentáře pomocí, kterých Doxygen generuje dokumentaci. Například pro popis daného souboru se používá následující syntaxe.

```
1  /**
2   *\file motor.c
3   *\author Miloš Mlejnek
4   *\date 10.3. 2023
5   *\version 1.0
6   *\brief Dvoufázový motorek
7   */
```

Obr. 1-5 Doxygen – úvod souboru

Pro detailní popis proměnné lze využít syntaxi: */*!< popis proměnné*/*

```
16 unsigned char pozice; /*!< 0-250 je rozmezí pohybu motorku, doprava počítám od 0 do 250, doleva odečítám 250 až 0*/
17 unsigned char pozice_max = 250; /*!< krajní pravá poloha*/
18 unsigned char polohaDEK; /*!< poloha dekodéru*/
19 unsigned char DEK; /*!< uložení hodnoty z Dčidla*/
20 T_dekoder dekoderPC; /*!<struktura z motor.h, T_stavDek stav, unsigned char pocetHran*/
21 unsigned char INKC; /*!< pro rychlost motorku*/
22 unsigned char INKC_INI= 50;
23 unsigned char stav; /*!< stav pohybu - pravá, levá, stojí, jede*/
```

Obr. 1-6 Doxygen – popis proměnné

Pro stručný popis funkce lze využít syntaxi:

```
12 /*! \brief Nastavení PB_ODR
13  *
14  *  Na PB_ODR  na 0 - 7 bitu LED a zároveň na 0-3 bitu vstupy do budiče a následně do motorku \n
15  *  PB_DDR - nastavení brány B na výstup\n
16  *  PB_CR1 - nastavení push-pull \n
17  *  PB_ODR - 1 znamená zhasnuto, 0 znamená rozsvíceno
18  */
19 void PBledInit(void)
20 {
21       PB_CR1 = PB_CR1_CW;
22       PB_DDR = PB_DDR_CW;
23       PB_ODR = PB_ODR_INI;
24 }
```

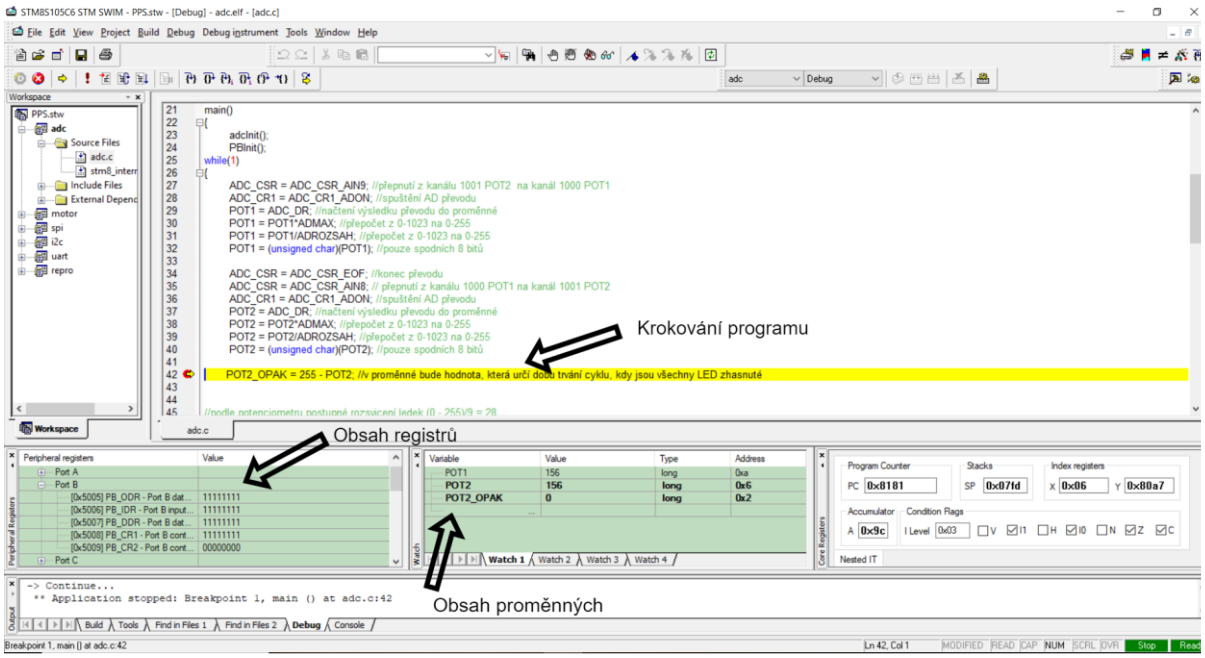
Obr. 1-7 Doxygen – stručný popis funkce

Další často využívané komentáře mohou být *class* jméno třídy značící komentující blok pro třídy, *attention* pro upozornění na důležitost věci, *image* pro vložení obrázku, *return* označující návratovou hodnotu a mnoho dalších. Komentáře použije tvůrce podle řešeného problému. Možnosti komentářů lze najít v manuálu doxygenu i s vysvětlivkami. [2]

1.4 Vývojové prostředí

Vývojové prostředí, Integrated Development Environment se zkratkou IDE, je software pro vývoj programů v konkrétním programovacím jazyce. Pro vývoj programů je potřeba editoru pro vytvoření zdrojového souboru, kompilátoru pro překlad zdrojového souboru a debuggeru k ladění souboru. Během ladění programu je užitečné krokování jednotlivých instrukcí programu nebo náhled do proměnných a registrů.

Výběr vývojového prostředí závisí na programovacím jazyku a rodině mikroprocesorů, pro který je aplikace vyvíjena. Dále může hrát roli kompatibilita se softwarem pro tvorbu automatické dokumentace. V případě této práce je vybráno vývojové prostředí podle mikroprocesoru STM8 a programovacího jazyka C. V této práci je použit vývojový kit STM8S Discovery kit s doporučením pro používání vývojového prostředí IAR, Embedded Workbench pro STM8 nebo ST Visual Develop. Pro tuto práci bylo vybráno vývojové prostředí St Visual Develop, které je určené pro vývojové a testovací účely poskytované bezplatně s překladačem Cosmic C.



Obr. 1-8 Vývojové prostředí ST Visual Develop

Na obrázku vývojového prostředí ST Visual Develop lze vidět editor s hlavním programem, kde lze jednotlivé instrukce krokovat. Dále si lze všimnout náhledu do periferních registrů nebo obsahu proměnných. V pravém rohu obrázku je náhled do jádrových registrů jako střadače, indexového registru, programového čítače nebo zásobníku. Způsoby krokování lze vybrat na horní liště vývojového prostředí s ikonami závorek.

KAPITOLA 2: POPIS PŘÍPRAVKU

2.1 Školní přípravek

Školní mikroprocesorový přípravek je postaven na procesorové desce STM8S-DISCOVERY vybavený mikroprocesorem STM8S105C6. Využity jsou všechny vývody prostřednictvím čtyř propojovacích konektorů. Pro práci ve vývojovém prostředí ST Visual Studio je třeba připojit osobní počítač. Pro komunikaci mezi přípravkem a osobním počítačem je využívána komunikace Swim ST-LINK.

Přehled vstupních zařízení

- čtyři tlačítka BT1 – BT4
- dva přepínače SW1 a SW2
- rotační inkrementální snímač IRC
- potenciometry POT1 a POT2
- optický snímač krajní polohy lineárního pohonu

Přehled výstupních zařízení

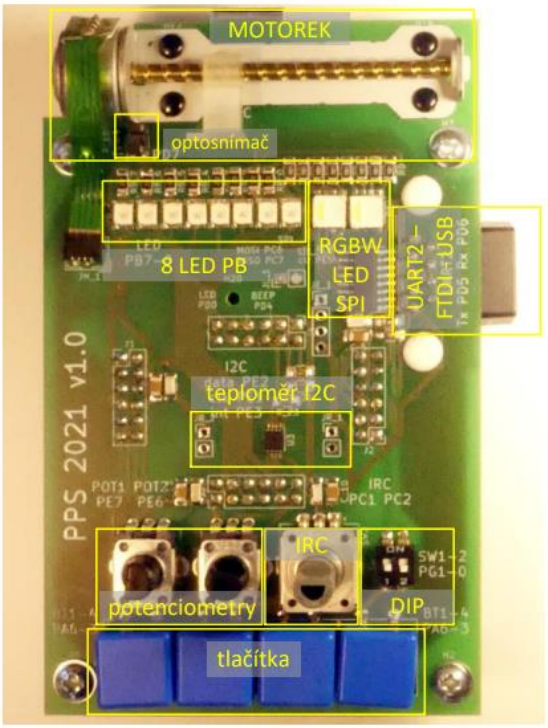
- 8 zelených 2 mA LED
- 2 čtyřbarevné RGBW LED
- dvoufázový motor s permanentními magnety a lineárním šroubovým posunem
- mini reproduktor
- dva topné rezistory pro snímač teploty

Přehled komunikačních a ovládacích obvodů

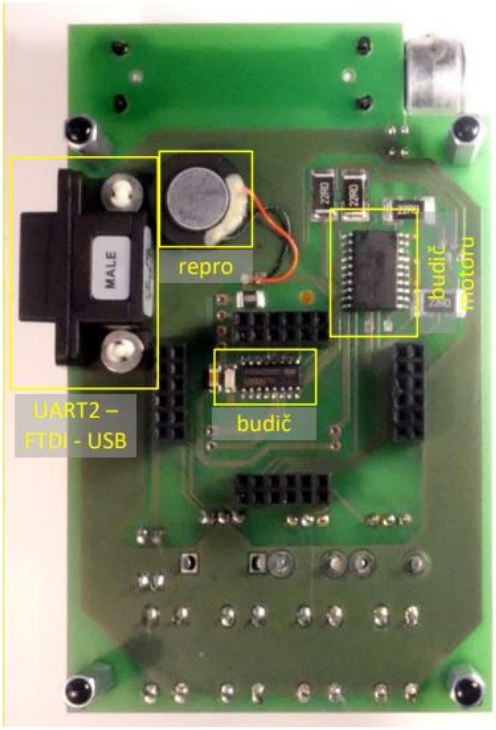
- FTDI převodník UART – USB (mini USB konektor)
- posuvný registr s pamětí a výstupním budičem pro ovládání RGBW diod – SPI komunikace
- čidlo teploty s nastavitelnou mezí a hysterezí pro signál překročení teploty-I2C komunikace budič motorku typu L293D
- tranzistorové pole typu ULN2003

Nastavení desky STM8S-DISCOVERY

- volba napájení
  - aplikační deska je navržena pro napájení 5 V
  - při napájení 3,3 V není dostatečné napětí pro napájení motorku, ostatní periferie fungují
- dotykový senzor
  - dotykový senzor na desce STM8S-DISCOVERY je připojen na bity PC1 a PC2
  - aplikační deska používá tyto vstupy pro inkrementální snímač
  - deaktivace dotykového senzoru viz kapitola 2.4 „Single touch sensing“ *[3]*



Obr. 2-1 Přípravek – pohled shora [7]



Obr. 2-2 Přípravek – pohled zdola [7]

KAPITOLA 3: VZOROVÉ PŘÍKLADY

3.1 Potenciometry POT1, POT2 a osm zelených LED

3.1.1 Analogově číslicový převodník a zelené LED

Přípravek STM8S-DISCOVERY obsahuje analogově číslicový převodník ADC1 s šestnácti kanály. Převod převodníků může být prováděn v nepřetržitém (continuous) režimu nebo v samostatném (single) režimu.

Pro převod hodnot je využit převodník ADC1 s větším množstvím možností v nastavení. Převodník převádí hodnoty z potenciometrů POT1 a POT2. Potenciometr POT1 se nachází na sedmém bitu brány E a potenciometr POT2 se nachází na šestém bitu brány E.

AD převodník je spuštěn po nastavení bitu ADON v registru ADC_CR1. Pro začátek převodu je třeba nastavit bit ADON do jedničky s druhým zapsáním instrukce. Po provedení převodu je sedmý bit registru ADC_CSR v jedničce. Převedená data se nahrají do registru ADC_DR. [1]

3.1.2 Zadání vzorového příkladu

Pomocí potenciometru POT1 postupně rozsvěcujte LED na bráně B a pomocí potenciometru POT2 měňte jas rozsvícených LED.

3.1.3 Nastavení

- Konstanta ADC_CR1_INI pro registr ADC_CR1 – nastaví dělení frekvence mikroprocesoru 2 MHz (fMASTER/18), režim samotného převodu v CONT bitu a spuštění převodu ADON bitem
- Konstanta ADC_CR1_ADON pro registr ADC_CR1 – spustí AD převodník
- Konstanta ADC_CR2_INI pro registr ADC_CR2 – nastaví zarovnání převedených dat doprava, tedy osm nejméně významných bitů bude v registru ADC_DRL a dva zbývající bity v registru ADC_DRH
- Konstanta ADC_CSR_AIN8 pro registr ADC_CSR – nastaví čtení z kanálu 1000, kde je potenciometr POT1
- Konstanta ADC_CSR_AIN9 pro registr ADC_CSR – nastaví čtení z kanálu 1001, kde je potenciometr POT2
- Konstanta ADC_CSR_EOF – ukončení převodu AD převodníku
- Konstanta ADC_TDRH_INI pro registr ADC_TDRH – snížení spotřeby statické energie pro vstupy obou potenciometrů na pinu PE6 a PE7
- Konstanta PB_DDR_INI pro registr PB_DDR – nastaví bránu B jako výstupní
- Konstanta PB_ODR_INI pro registr PB_ODR – zhasnutí celé brány B
- Konstanta PB_CR1_INI pro registr PB_CR1 – nastaví push–pull výstup

Inicializace zmíněných registrů proběhne v souboru *adcInit.c*. Možnosti nastavení lze najít v dokumentu UM0817 User Manual *[4].*

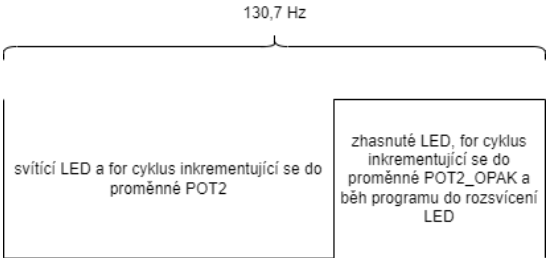
3.1.4 Popis řešení zadání

Hlavní programová smyčka probíhá v souboru *adc.c*. Nejprve se zavolají funkce inicializace pro AD převodník a brány B. Poté se spustí převod z potenciometru POT1. Výsledek převodu se převede z desetibitového čísla na osmibitové. Výsledné osmibitové číslo je uloženo do proměnné *POT1*. Pro čtení z potenciometru POT2 je ukončen převod z potenciometru POT1 a změněn kanál pro čtení. Čtení z potenciometru POT2 proběhne analogicky jako u potenciometru POT1 a výsledek je uložen do proměnné *POT2*.

```
33      ADC_CSR = ADC_CSR_AIN9; //přepnutí z kanálu 1001 POT2  na kanál 1000 POT1
34      ADC_CR1 = ADC_CR1_ADON; //spuštění AD převodu
35      POT1 = ADC_DR; //načtení výsledku převodu do proměnné
36      POT1 = POT1*ADMAX; //přepočet z 0-1023 na 0-255
37      POT1 = POT1/ADROZSAH; //přepočet z 0-1023 na 0-255
38      POT1 = (unsigned char)(POT1); //pouze spodních 8 bitů
39
40      ADC_CSR = ADC_CSR_EOF; //konec převodu
41      ADC_CSR = ADC_CSR_AIN8; // přepnutí z kanálu 1000 POT1 na kanál 1001 POT2
42      ADC_CR1 = ADC_CR1_ADON; //spuštění AD převodu
43      POT2 = ADC_DR; //načtení výsledku převodu do proměnné
44      POT2 = POT2*ADMAX; //přepočet z 0-1023 na 0-255
45      POT2 = POT2/ADROZSAH; //přepočet z 0-1023 na 0-255
46      POT2 = (unsigned char)(POT2); //pouze spodních 8 bitů
```

Obr. 3-1 POT1, POT2 a zelené LED – analogově číslicový převod z potenciometrů

Pro řízení jasu byla zvolena metoda rychlého rozsvěcení a zhasínání LED, které lidské oko nezaznamená. Nejvyšší frekvence rozlišení lidského oka je 60 Hz, z čehož vyplývá, že interval mezi rozsvícením a zhasnutím brány B musí odpovídat frekvenci vyšší než 60 Hz. Tento interval byl vytvořen dvěma *for* cykly řídícími se proměnnou *POT2*. První *for* cyklus je určen pro svícení LED a jeho inkrementace probíhá do hodnoty proměnné POT2. Po prvním *for cyklu* dojde k zhasnutí LED a vstupu do druhého *for* cyklu, který se inkrementuje do dopočtu proměnné *POT2*, uložené v proměnné *POT2_OPAK*. LED jsou zhasnuté až do podmínky, která určuje kolik LED bude rozsvícených. Pro změření doby a frekvence intervalu byl využit simulátor se stopkami. V případě tohoto programu, překladače a vývojového prostředí je interval dlouhý 7,65 ms, což znamená frekvenci 130,7 Hz. Pro změření intervalu je třeba mít správně zvolenou frekvenci běhu mikroprocesoru a to 2 MHz.



Obr. 3-2 POT1, POT2 a zelené LED – interval pulzně šířkového řízení jasu

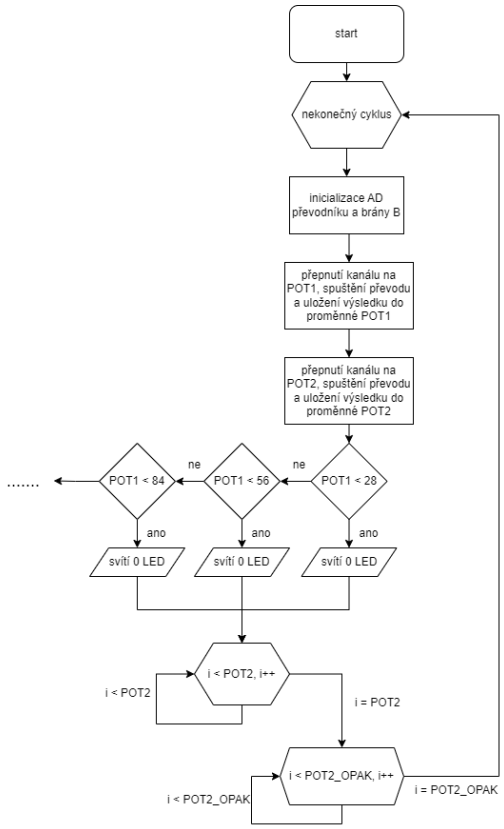
Podle hodnoty proměnné *POT1* je určen počet rozsvícených LED pomocí konstant nadefinovaných v souboru *adc.h* a podmínek if a else.

```
51  //podle potenciometru postupné rozsvícení ledek (0 - 255)/9 = 28
52  if(POT1 < 28)
53  {
54      PB_ODR = LED0;
55
56  }else if(POT1 < 56)
57  {
58      PB_ODR = LED1;
59
60  }else if(POT1 < 84)
61  {
62      PB_ODR = LED2;
63
64  }else if(POT1 < 112)
65  {
66      PB_ODR = LED3;
67
68  }else if(POT1 < 140)
69  {
70      PB_ODR = LED4;
71
```

Obr. 3-3 POT1, POT2 a zelené LED – podmínky pro určení počtu svítících LED

```
68  #define LED0 0b11111111
69  #define LED1 0b11111110
70  #define LED2 0b11111100
71  #define LED3 0b11111000
72  #define LED4 0b11110000
73  #define LED5 0b11100000
74  #define LED6 0b11000000
75  #define LED7 0b10000000
76  #define LED8 0b00000000
```

Obr. 3-4 POT1, POT2 a zelené LED – konstanty pro počet svítících LED

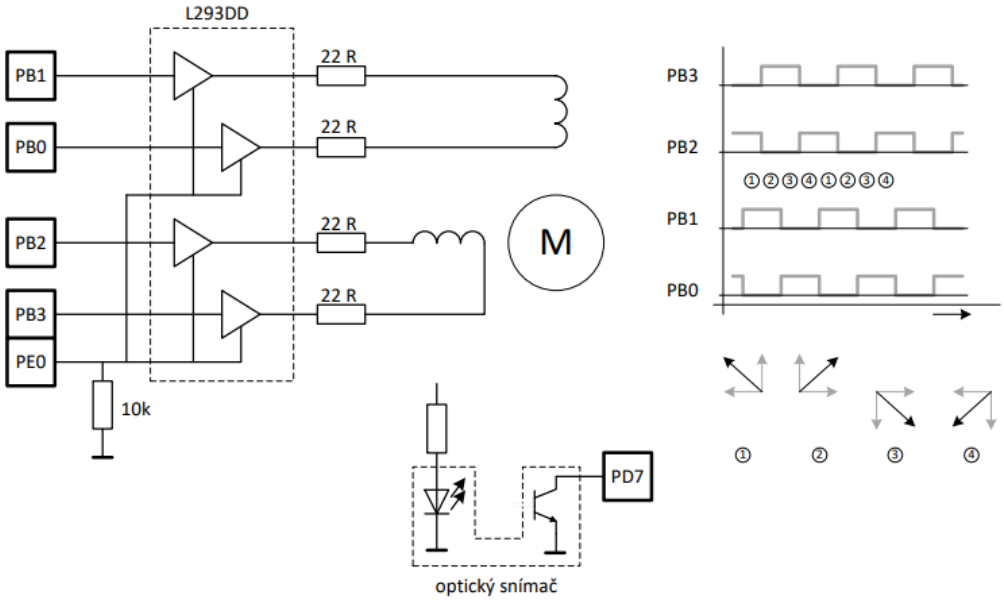


Obr. 3-5 POT1, POT2 a zelené LED – vývojový diagram

3.2 Motorek, čidlo polohy, tlačítka a inkrementální čidlo

3.2.1 Motorek

Jednou z dalších možností přípravku je dvoufázový motorek s budičem L293D. Pro ovládání motorku jsou použity čtyři dolní bity na bráně B. Motorek otáčí šroubovým převodem s jezdcem. Pro určení levé krajní polohy je použito optický snímač na sedmém bitu brány D. Pro ochranu motorku a vinutí jsou v sérii s každým vinutím dva rezistory. *[7]*



Obr. 3-6 Motorek – zapojení dvoufázového motorku [7]

3.2.2 Zadání vzorového příkladu

Úkolem vzorového příkladu pro dvoufázový motorek je pojezd motorku pomocí tlačítek a inkrementálního čidla. Motorek bude jezdit ze strany na stranu. Pro určení polohy motorku využijte optický snímač polohy v levé pozici šroubového převodu. Pro prvotní určení polohy je třeba nejprve dojet do levé krajní pozice s optickým snímačem a poté například s každým pohybem doprava inkrementovat proměnou. Při zmáčknutí tlačítka BT1 na šestém pinu brány A dojde ke změně pohybu motorku doleva, v případě již správného směru pohybu motorku se nic nezmění. V případě zmáčknutí tlačítka čtyři BT4 na třetím pinu brány A dojde ke změně směru pohybu motorku doprava, pokud již motorek jel směrem doprava, nic se nezmění. Zmáčknutím tlačítek dva nebo tři na čtvrtém a pátém pinu brány A dojde k zastavení pohybu motorku. Pro následný rozjezd je třeba zmáčknout tlačítko BT1 pro pohyb doleva nebo tlačítko BT4 pro pohyb doprava. Pro změnu rychlosti využijte inkrementální čidlo na prvním a druhém piny brány C. Napájení dvou vinutí motorku je ovládáno z výstupních pinů 0-3 na bráně B, což kromě samotného pohybu motorku způsobí rozsvěcení LED na daných pinech. Dosažení maximální nebo minimální rychlosti motorku bude signalizovat osmá LED na bráně B.

3.2.3 Nastavení

- Konstanta PB_DDR_INI pro registr PB_DDR – nastaví bránu B jako výstupní
- Konstanta PB_ODR_INI pro registr PB_ODR – zhasnutí celé brány B
- Konstanta PB_CR1_INI pro registr PB_CR1 – nastaví push-pull výstup
- Konstanta PD_DDR_OPT pro registr PD_DDR – na sedmém pinu je optický snímač, který se nastaví do vstupního režimu
- Konstanta PD_CR1_OPT pro registr PD_CR1 – nastaví vstup na push-pull pro optický snímač
- Konstanta PC_CR1_IRC pro registr PC_CR1 – nastaví vstup na push-pull pro inkrementální čidlo
- Konstanta PE_DDR_MOT pro registr PE_DDR – nastaví jako výstupní nultý pin brány E pro odblokování budiče motorku
- Konstanta PE_CR1_MOT pro registr PE_CR1 – nastaví push-pull výstup na nultém pinu brány E pro odblokování budiče motorku
- Konstanta PA_CR1_BT pro registr PA_CR1 – nastaví push-pull vstup pro tlačítka BT1 – BT4
- Konstanta TIM4_ARR_INI pro registr TIM4_ARR – nastaví hodnotu, do které čítá čítač
- Konstanta TIM4_PSCR_INI pro registr TIM4_PSCR – nastaví hodnotu pro dělení frekvence mikroprocesoru
- Konstanta TIM4_CR1_INI pro registr TIM4_CR1 – umožní čítaní čítače
- Konstanta TIM4_IER_INI pro registr TIM4_IER – umožní vyvolání přerušení

Při přetečení čítače TIM4 dojde k vstupu do podprogramu přerušení. Pro vstup do podprogramu přerušení je třeba podprogram definovat v souboru *stm8_interrupt_vector.c* a následně podprogram uvést do tabulky vektorů v tomto souboru. Správný vektor přerušení pro čítač TIM4 lze nalézt na straně 45 manuálu *[5].*

Inicializace registrů LED na bráně B, čidla polohy, tlačítek, čítače a inkrementálního čidla proběhne v souboru *motorInit.c.*

Možnosti nastavení lze najít v dokumentu UM0817 User Manual *[4].*

3.2.4 Popis řešení zadání

Hlavní programová smyčka probíhá v souboru *motor.c*. Nejprve se zavolají funkce pro inicializaci registrů a povolí se vyvolání přerušení. Poté se vstoupí do nekonečné smyčky. Jako první se

v nekonečné smyčce zkontroluje stav optického snímače levé polohy. V případě, že se běžec nachází v levé poloze, vynuluje se proměnná pozice a do proměnné stav je uložena jednička pro směr pohybu doprava.

```
45      if(PD_IDR &(1<<7)) // levá pozici? na 7 bitu je optický snimac leve polohy (1000 0000)
46      {
47          pozice = 0; //levý kraj, začnu počítat od 0 do 250 a 250 je pravý okraj
48          stav |= 1UL << 1; //změna směru pohybu doprava
49      }
```

Obr. 3-7 Motorek – kontrola levé polohy

Dále dojde ke kontrole stisknutí tlačítek BT1 – BT4. V případě stisknutého tlačítka BT1 dojde ke změně směru doprava, v případě stisknutí BT4 dojde ke změně směru doleva a pokud je stisknuté jedno z tlačítek BT2 nebo BT3 dojde k zastavení motorku. Opětovný rozjezd motorku umožní jedno z tlačítek BT1 pro pohyb doprava nebo BT4 pro pohyb doleva.

```
51      if(!(PA_IDR &(1<<6))) // pokud stisknuté BT1 na 6. bitu (úplně vlevo tlačítko)
52      {
53          stav &= ~(1UL <<1);  //1.bit do 0 ZMĚNA SMĚRU DOLEVA
54          stav |= 1UL << 0;  // 0.bit do 1 STAV JEDU
55
56      }else if(!(PA_IDR &(1<<3))) // pokud stisknuté BT4 na 3. bitu (úplně vpravo tlacitko)
57      {
58          stav |= 1UL << 1;  //1.bit do 1 ZMĚNA SMĚRU DOPRAVA
59          stav |= 1UL << 0;  //0.bit do 1 STAV JEDU
60
61      }else if(!(PA_IDR &(1<<4))||!(PA_IDR &(1<<5))) // pokud jsou stisknuté tlačítka BT2 nebo BT3
62      {
63          stav &= ~(1UL <<0); //0.bit do 0, ZASTAVENÍ
64      }
```

Obr. 3-8 Motorek – kontrola stisknutí tlačítek

Po kontrole tlačítek dojde ke kontrole maximální a minimální rychlosti motorku a případnému rozsvícení osmé LED na bráně B. Funkce pro kontrolu maxima a minima se nachází v souboru *motorInit.c*.

```
66      DEKMI = dekoderMP(&dekoderPC);
67      DEKMA = dekoderMR(&dekoderPC);
68
69      if(DEKMI==1 || DEKMA ==1)
70      {
71          PB_ODR &= ~(1UL <<7);
72
73      }else{
74
75          PB_ODR |= 1UL << 7;
76      }
```

Obr. 3-9 Motorek – kontrola maxima a minima rychlosti motorku

```
214     /*! \brief porovná jestli je dosaženo maximální rychlosti*/
215     bool dekoderMP(T_dekoder*filter)
216     {
217     return filter -> pocetHran == DEKODER_MAX;
218     }
219     /*! \brief porovná jestli bylo dosaženo minimální rychlosti*/
220     bool dekoderMR(T_dekoder*filter)
221     {
222     return filter -> pocetHran == DEKODER_MIN;
223     }
```

Obr. 3-10 Motorek – funkce pro kontrolu maximální a minimální rychlosti motorku

Kdykoli během běhu hlavního programu může být vyvoláno přerušení. Při vyvolání přerušení dojde k vstupu do podprogramu přerušení. V podprogramu přerušení se nejprve dekrementuje proměnná *INKC*, která určuje, jak často se bude spínat vinutí motorku natočením inkrementálního čidla. Pro čtení hodnoty natočené inkrementálním čidlem je určena funkce *dekoder* v souboru *motorInit.c*. Do této funkce vstupují výsledky kanálu inkrementálního čidla a proměnná *dekoderPC* s informací o předchozím stavu dekodéru a počtu hran.

```
86          if(!INKC == 0)
87          {
88                  INKC--;
89          }
90          polohaDEK = dekoder(&dekoderPC, (PC_IDR &(1<<1)), (PC_IDR &(1<<2)));
91          DEK = polohaDEK/4;
```

Obr. 3-11 Motorek – řízení rychlosti čtením hodnoty z inkrementálního čidla

Pokud dojde k dosažení nuly proměnné *INKC* dojde k vstupu do podmínky, která spíná vinutí motorku.

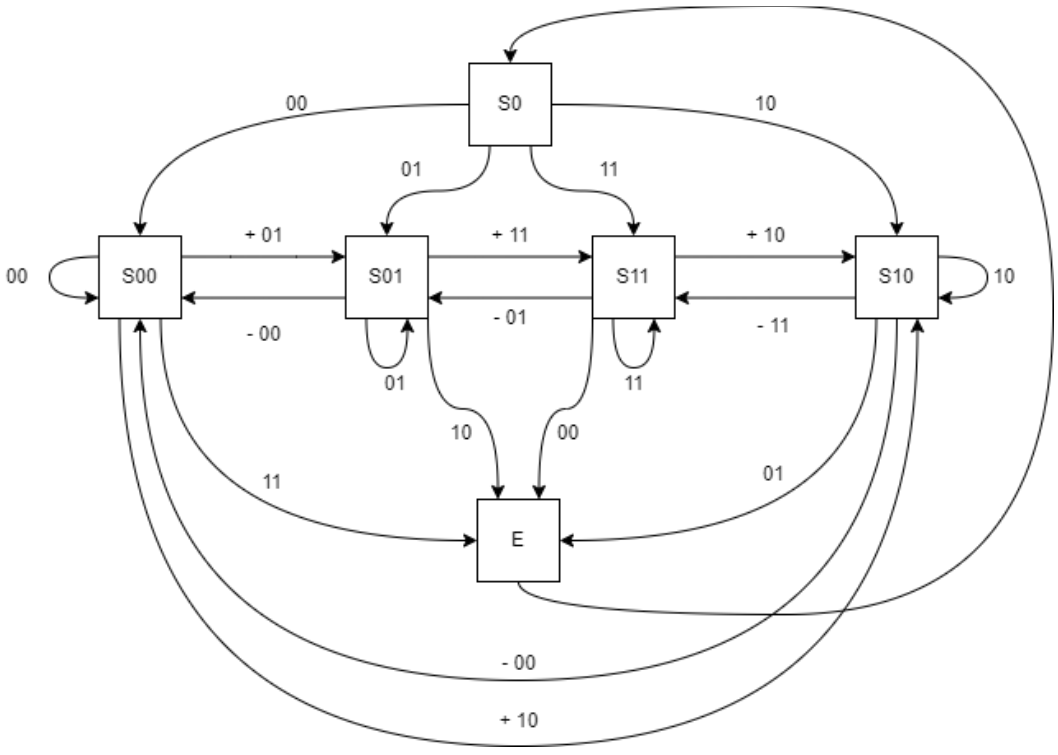
```
93  if((stav &(1 << 0))&&(INKC == 0)) // motorek musí být v pohybu a proměnná INCK je musí dekrementovat do nuly
94  {
95      polohaDEK = dekoder(&dekoderPC, (PC_IDR &(1<<1)), (PC_IDR &(1<<2)));
96      INKC = DEK;
97
98          if(sepnout == 0)
99          {
100             if(PB_ODR &(1<<7))
101             {
102                 PB_ODR = NULAZ;
103
104             }else{
105
106                 PB_ODR = NULAR;
107             }
108
109
110         }else if(sepnout == 1)
111         {
112             if(PB_ODR &(1<<7))
113             {
114                 PB_ODR = JEDNAZ;
115
116             }else{
117
118                 PB_ODR = JEDNAR;
119             }
120
121         }else if(sepnout == 2)
122         {
123             if(PB_ODR &(1<<7))
124             {
125                 PB_ODR = DVAZ;
126
127             }else{
128
129                 PB_ODR = DVAR;
```

Obr. 3-12 Motorek – spínání vinutí motorku

Po spínací kombinaci motorku se v podprogramu přerušení inkrementuje proměnná *pozice* a v případě dosažení hodnoty 250, což je pravá poloha, je uložena do proměnné *stav* hodnota pro pohyb motorku doleva. Zároveň se inkrementuje proměnná *sepnout* pro vybrání další spínací kombinace. Pokud jede motorek doleva, proměnné *pozice* a *sepnout* se dekrementují. Podprogram přerušení je ukončen shozením vlajky přerušení.

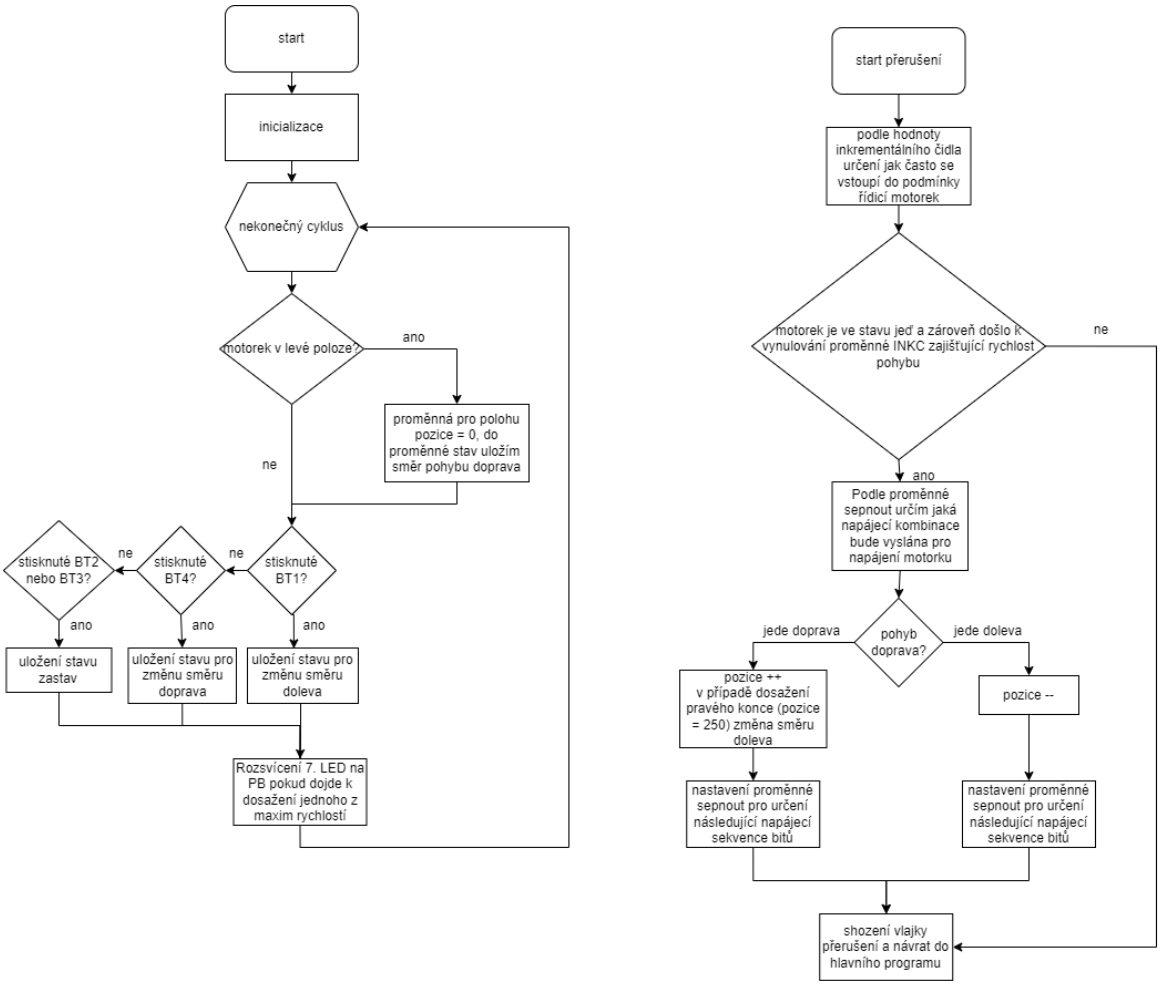
```
144         if(stav &(1 << 1)) // pokud jedu doprava
145         {
146               pozice++;
147
148               if(pozice == pozice_max)// pokud jsem na pravem konci
149               {
150                  stav &= ~(1UL <<1);  // zmena smeru, jedu doleva
151               }
152
153               sepnout++;
154
155               if(sepnout==4)
156               {
157                  sepnout = 0;
158               }
159
160         }else // pokud jedu doleva
161         {
162            pozice--;
163
164               if(sepnout==0)
165               {
166                  sepnout = 4;
167               }
168               sepnout--;
169         }
170      }
171      TIM4_SR = 0; //shození interrupt flag
172   }
```

Obr. 3-13 Motorek – určení polohy a příští spínací kombinace



Obr. 3-14 Motorek – stavový diagram inkrementálního čidla, funkce dekoder

Obr. 3-15 Motorek – stavy stavového diagramu pro funkci dekoder



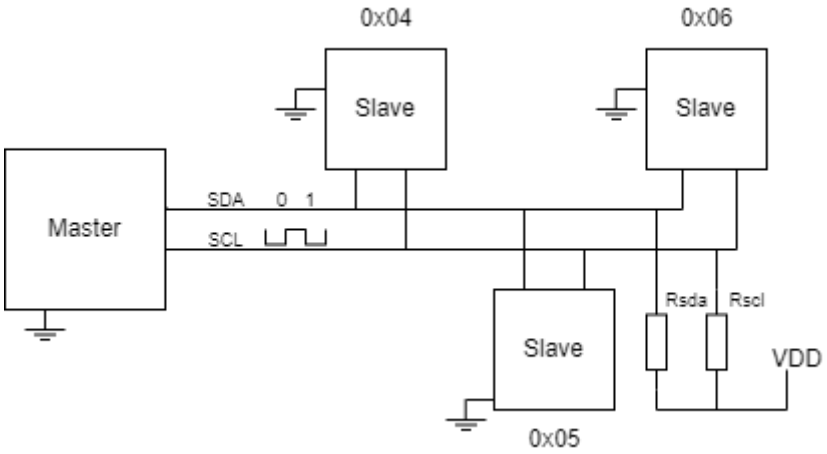
Obr. 3-16 Motorek – vývojový diagram

3.3 Komunikační protokol I2C

3.3.1 Princip I2C

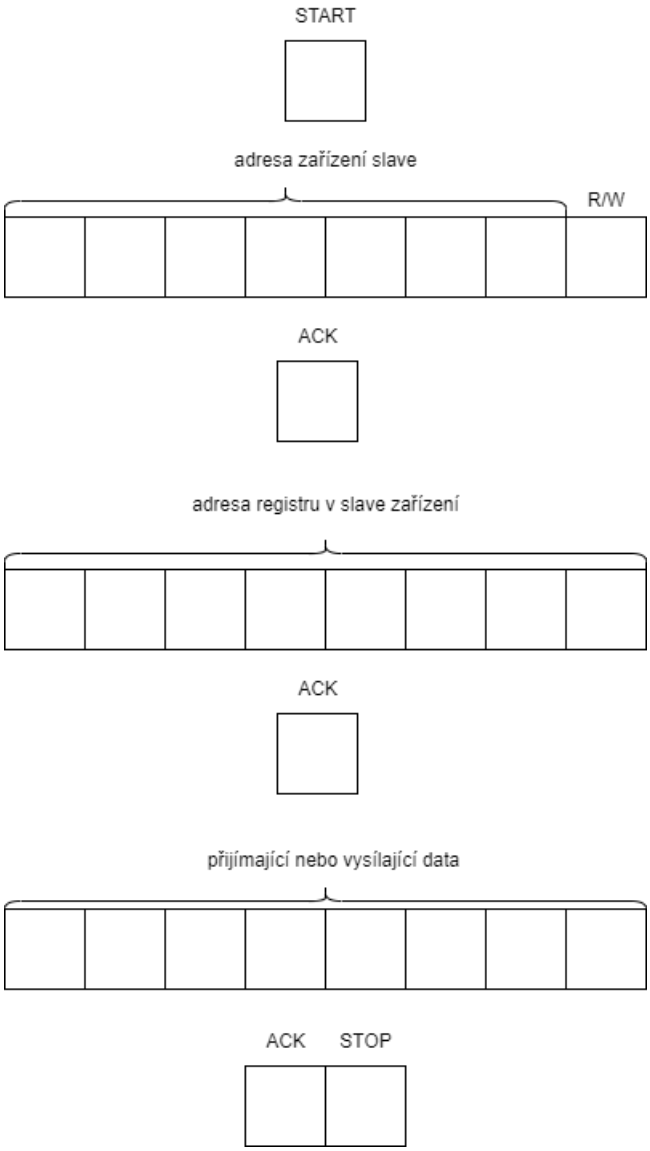
I2C, inter-integrated circuit, je sériový komunikační protokol používaný pro přenos dat mezi integrovanými obvody. Komunikace je možná mezi jedním nebo několika nadřízenými zařízeními zvanými masters a podřízenými zařízeními zvané slaves. Tento komunikační protokol využívá pro přenos dat datový vodič SDA (serial data) a hodinový vodič SCL (serial clock). Vodiče jsou obousměrné, využívané vždy pouze v daném směru v závislosti na režimu čtení nebo zápisu. Komunikaci řídí master zařízení určující, jaká data jsou posílána a jakému zařízení slave. Každé zařízení slave má svou adresu pro komunikaci se zařízením master. V případě napětí na vodičích nedochází k žádnému přenosu dat Napětí ve vodiči má význam logické jedničky. Pro začátek komunikace musí být vysílající zařízení připojeno k GND, čímž vznikne nulové napětí s významem

logické nuly. Součástí celého uspořádání jsou rezistory s hodnotami odporu v kiloohmech. Hodnota rezistoru omezuje rychlost přenosu s větším odporem rychlost přenosu klesá. Tyto rezistory zamezují zkratu mezi zdrojem napětí a zemí. Každé zařízení má open-drain kolektor pro vodič SDL a SDA.



Obr. 3-17 I2C – princip

Přenos začíná vysláním start bitu od master zařízení. Za start bitem následuje sedmibitová adresa slave zařízení, se kterým bude probíhat komunikace. Po adresování adresované slave zařízení vyšle acknowledge bit zpět k master zařízení pro potvrzení komunikace. Následuje osmibitová adresa registru, se kterým chce master komunikovat v daném slave zařízení. Po přijetí adresy, slave zařízení znovu vyšle acknowledge bit. V případě, že chce master zařízení poslat data do slave zařízení, následuje sekvence osmi bitů nesoucí informaci, po které znovu slave zařízení vyšle zpět acknowledge bit. Pokud chce master zařízení číst z daného registru slave zařízení je sekvence osmi bitů vyslána master zařízení a master zařízení vysílá zpět k slave zařízení acknowledge bit. Jestliže chce master zařízení data ve více než osmi bitech, tak po každé sekvenci osmi bitů musí být vyslán acknowledge bit. Pro ukončení komunikace vyšle master zařízení stop bit.

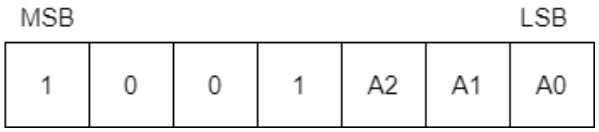


Obr. 3-18 I2C – význam bitů při komunikaci I2C

Výhodami I2C komunikace je komunikace pouze po dvou vodičích, možnost více master zařízení, přehled o doručení dat díky acknowledge bitu a možnost změny rychlosti přenosu podle vybraného slave zařízení, se kterým se bude komunikovat.

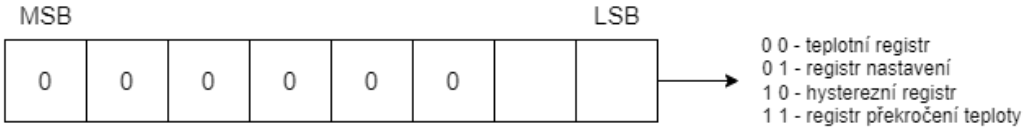
3.3.2 Teplotní senzor LM75B

Tento teplotní senzor měří teplotu a následně hodnotu teploty převede na digitální číslo. Dále tento senzor umožňuje detekci překonání přednastavené maximální teploty. Měření teploty probíhá s přesností 0,125 °C. Měření teploty probíhá každých 100 ms, kdy se na konci převodu na digitální hodnotu hodnota porovná s maximální přednastavenou hodnotou. Výsledek převodu je uložen do jedenáctibitového čísla. Pro čtení převedených dat se využívá sériové dvoulinky I2C, kdy teplotní senzor je zařízení slave a mikroprocesor jako zařízení master. Mikroprocesor poskytuje hodinový signál SCL a řídí datový vodič SDA. Adresa teplotního čidla je dána sedmibitovou adresou, kdy čtyři nejvyšší bity jsou pevně určené propojením obvodu v čidle. Tři nejméně významné bity jsou značeny A2, A1 a A0 a určeny piny, ke kterým jsou připojeny. Piny jsou připojeny k napájecímu napětí V_{CC} pro logickou 1 nebo k GND pro logickou 0. *[6]*



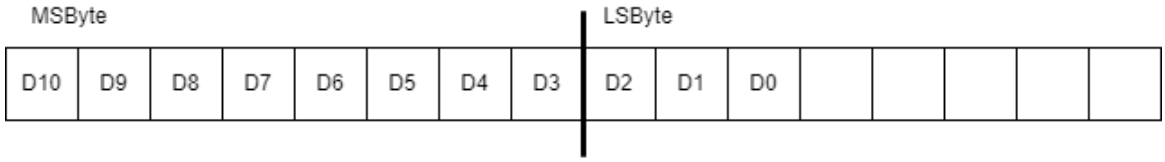
Obr. 3-19 I2C – adresa teplotního čidla

Pro práci s teplotním čidlem využíváme několika registrů. Pro určení, se kterým registrem chce uživatel pracovat, je určen registr ukazatel (Pointer registr). Tento registr je osmibitový a dva jeho nejméně významné bity určují hodnotu jednoho z následujících čtyř registrů a to teplotního (temperature) registru, registru nastavení (configuration registr), hysterezního (hysteresis registr) registru a registru překročení teploty (overtemperature shutdown registr). Zbylé bity jsou rovné nule. *[6]*

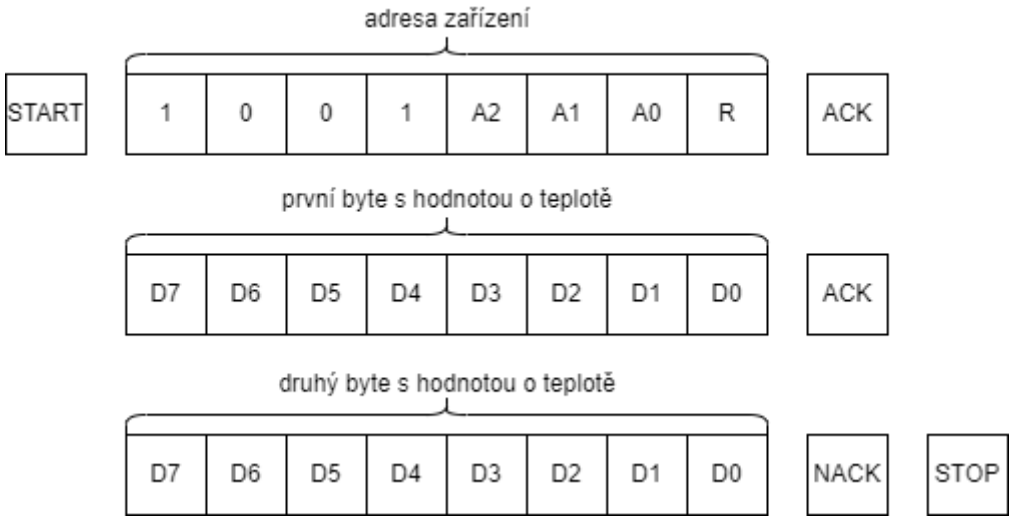


Obr. 3-20 I2C – pointer registr

Do teplotního registru se ukládá výsledek A-D převodu po změření teploty. Tento registr je šestnáctibitový, kdy pouze jedenáct nejvýznamnějších bitů nese informaci o teplotě. Nejvíce významný bit tohoto jedenáctibitového čísla udává znaménko teploty. Pro vyjádření teploty z tohoto čísla je třeba číslo vynásobit hodnotou 0,125. Přes linku I2C je přenášena šestnáctibitová hodnota teploty s prvním vysláním start bitu, následnou osmibitovou adresou zařízení, s bitem pro čtení nebo zápisu, ukončená acknowledge bitem. Po adrese zařízení následuje šestnáctibitová hodnota teploty po osmi bitech přerušená acknowledge bitem. Přenos je ukončen not acknowledge bitem a stop bitem. *[6]*

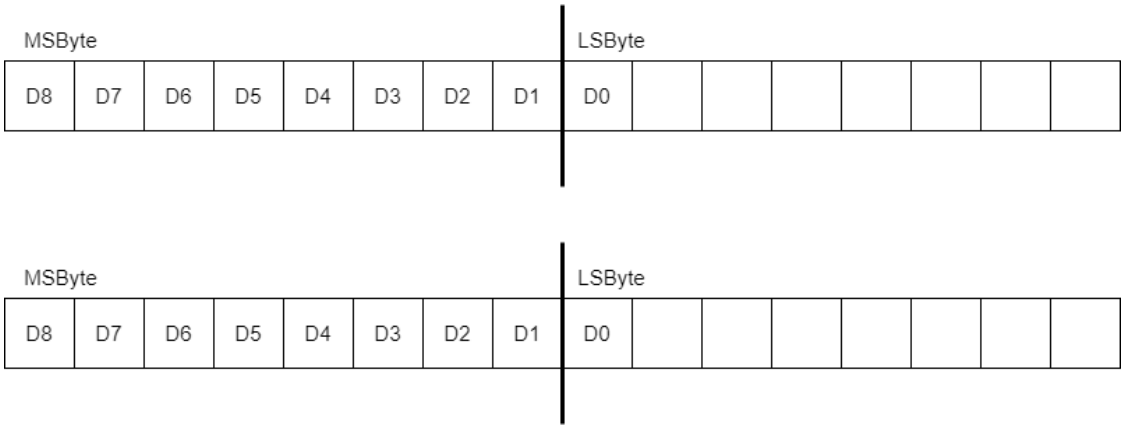


Obr. 3-21 I2C – teplotní registr



Obr. 3-22 I2C – přenos hodnoty teploty

Registr nastavení je osmibitový registr, který se používá pro nastavení zařízení. V této práci je použité výchozí nastavení pro registr nastavení. Registr překročení teploty a hysterezní registr jsou registry pro uložení hodnot teplotních maxim. Na konci každého převodu se převedená data porovnají s daty uloženými v těchto dvou registrech. Oba registry jsou šestnáctibitové, kdy pouze devět nejvíce významných bitů jsou určeny pro uložení teplotních maxim. Na porovnání naměřené převedené hodnoty reaguje takzvaný OS output, který v režimu normálního chodu nastavuje stav zařízení podle porovnání. *[6]*



Obr. 3-23 I2C – registr překročení teploty a hysterezní registr

3.3.3 Zadání vzorového příkladu

Vzorový příklad sériové komunikace I2C je komunikace mezi mikroprocesorem jako master zařízení a teplotním čidlem LM75B jako slave zařízení. Teplotní čidlo má měřit teplotu na rezistorech PD3 a PD2. Topení do rezistorů mají signalizovat svítící LED na PD3 a PD2. Pro spuštění vytápění rezistorů mají sloužit přepínače SW1 a SW2. Naměřená hodnota teploty má být následně převedena na digitální číslo a binárně se zobrazit na LED PB0 – PB7.

3.3.4 Nastavení

- Konstanta PB_DDR_INI pro registr PB_DDR – nastaví bránu B jako výstupní
- Konstanta PB_ODR_INI pro registr PB_ODR – zhasnutí celé brány B
- Konstanta PB_CR1_INI pro registr PB_CR1 – nastaví push-pull výstup
- Konstanta PG_CR1_DIP pro registr PG_CR1 – nastaví push-pull vstup pro přepínače
- Konstanta PD_DDR_TEP pro registr PD_DDR – nastaví výstup pro topné rezistory a LED na bráně D
- Konstanta PD_CR1_TEP pro registr PD_CR1 – nastaví push-pull výstup pro topné rezistory a LED na bráně D
- Konstanta I2C_FREQR_INI pro registr I2C_FREQR – nastaví časování periferních vnitřních hodin na 16 MHz
- Konstanta I2C_TRISER_INI pro registr I2C_TRISER – nastaví maximální dobu trvání zpětné vazby SCL
- Konstanty I2C_CCRL_INI a I2C_CCRH_INI pro registr I2C_CCRL – kontrola hodin SCL
- Konstanta I2C_CR1_INI pro registr I2C_CR1 – umožnění periferií

V souboru *I2CInit.c* proběhne inicializace jednotlivých periferií. Možnosti nastavení lze najít v dokumentu UM0817 User Manual *[4].*

3.3.5 Řešení vzorového příkladu

V hlavní smyčce programu nejprve dojde k zavolání funkcí pro inicializaci potřebných periferií. Poté se před vstupem do nekonečné smyčky se zavolá funkce pro nastavení teplotního čidla. Do proměnné *adresa* se ukládá adresa registru hystereze a do proměnné *teplota* hodnota hystereze. Následně se zavolá funkce *write*, kde dojde k zapsání požadované hodnoty hystereze. Stejný postup platí pro zapsání teploty, která nemá být překročena do registru překročení teploty. Zapsání probíhá následovně:

- V registru I2C_CR2 se nastaví přijetí acknowledge bitu
- V registru I2C_CR2 se nastaví generování start bit
- Počká se na vygenerování start bitu
- Do registru I2C_DR se zapíše adresa zařízení, do kterého se bude zapisovat
- Počká se na shodu adresy se zařízením
- Počká se na vyčištění datového registru
- Dojde k poslání adresy registru, do kterého se bude zapisovat
- Počká se na vyčištění datového registru
- Dojde k poslání prvního bytu maximální hodnoty hystereze nebo hodnoty překročení teploty
- Počká se na vyčištění datového registru
- Dojde k poslání druhého bytu maximální hodnoty hystereze nebo hodnoty překročení teploty
- Počká se na vyčištění datového registru
- Počká se na úspěšné poslání dat
- Dojde k ukončení generování

```
71  void write()
72  {
73      I2C_CR2 |= 1UL << 2; //návrat acknoledge bitu po přijetí dat nebo adresy
74      I2C_CR2 |= 1UL << 0; //začátek generování jakmile je vodič volný
75
76      while(!(I2C_SR1 &(1<<0)) ); //čeká se na vygenerování start bitu
77
78      I2C_DR = TEMP_ADRW; //zápis do DR registru vyčistí SR1 0bit, vyšle se adresa zařízení, se kterým se bude komunikovat
79
80      while(!(I2C_SR1 &(1<<1)) ); //shoda s přijatou adresou
81
82      prazdny = I2C_SR3; //přečte se  SR3 - vyčistí se SR1
83
84      while(!(I2C_SR1 &(1<<7)) ); //dokud není prázdný datový registr
85
86      I2C_DR = adresa; //vyslání adresy registru do kterého se bude zapisovat
87
88      while(!(I2C_SR1 &(1<<7)) ); //dokud není prázdný datový registr
89
90      I2C_DR = teplota;
91
92      while(!(I2C_SR1 &(1<<7)) ); //dokud není prázdný datový registr
93
94      I2C_DR =  teplota >>16;
95
96      while(!(I2C_SR1 &(1<<7)) ); //dokud není prázdný datový registr
97      while(!(I2C_SR1 &(1<<2)) ); //dokud nejsou data úspěšně poslána
98
99      I2C_CR2 |= 1UL << 1; //konec generování po posledním bytovém přenosu nebo po poslání podmínky start
100 }
```

Obr. 3-24 I2C – funkce write nastavující teplotní čidlo

Po nastavení hysterezního registru a registru překročení teploty se přejde z režimu zapisování do teplotního čidla na režim čtení z teplotního čidla. Poté se vstoupí do nekonečného cyklu, kde se pravidelně čte teplota z teplotního čidla a přečtená hodnota se pošle na bránu B pomocí funkce LED.

- V registru I2C_CR2 se nastaví přijetí acknowledge bitu
- V registru I2C_CR2 se nastaví generování start bit
- Počká se na vygenerování start bitu
- Do registru I2C_DR se zapíše adresa zařízení, ze kterého se bude číst
- Počká se na shodu adresy se zařízením
- Počká se na vyčištění datového registru
- V registru I2C_CR2 se nastaví nenavrácení acknowledge bitu
- Dojde k ukončení generování
- Výsledek převodu je uložen do proměnné *prevod*

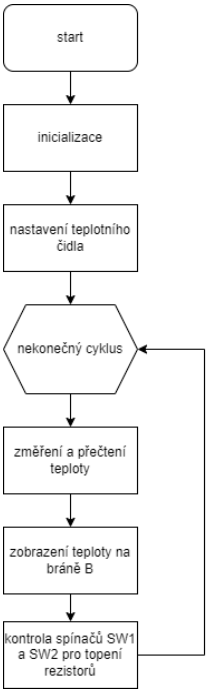
```
129  void read()
130  {
131      I2C_CR2 |= 1UL << 2; //návrat acknoledge bitu po přijetí dat nebo adresy
132      I2C_CR2 |= 1UL << 0; // začátek generování jakmile je vodič volný
133
134      while(!(I2C_SR1 &(1<<0)) );   //čeká se na vygenerování start bitu
135
136      I2C_DR = TEMP_ADRR; //zápis do DR registru vyčistí SR1 0bit, vyšle se adresa zařízení, se kterým se bude komunikovat
137
138      while(!(I2C_SR1 &(1<<1)) ); //shoda s přijatou adresou
139
140      prazdny = I2C_SR3; //přečte se  SR3 - vyčistí se SR1
141
142      while(!(I2C_SR1 &(1<<6)) ); //dokud není prázdný datový registr
143
144      I2C_CR2 &= ~(1UL <<2); //není navrácen acknoledge bit
145      I2C_CR2 |= 1UL << 1; //konec generování po posledním bytovém přenosu nebo po poslání podmínky start
146
147      prevod  = I2C_DR;
148
149      while(!(I2C_SR1 &(1<<6)) );  //dokud není prázdný datový registr
150
151      prevod  =  prevod << 8;
152      prevod = prevod + I2C_DR;
153  }
```

Obr. 3-25 I2C – funkce read čtoucí teplotu z teplotního čidla

Jako poslední se ve funkci *topeni* zkontroluje, jestli jsou sepnuty spínače SW1 nebo SW2 na PG0 a PG1. V případě sepnutí spínačů se rozsvítí LED na PD2, PD3 a zároveň se začnou zahřívat topné rezistory.

```
170   void topeni()
171   {
172       if((PG_IDR &(1<<0))&&(PG_IDR&(1<<1)))
173       {
174           PD_ODR = 0b00001100; // SW1 a SW2 OFF
175
176       }else if((!(PG_IDR &(1<<0)))&&(!(PG_IDR&(1<<1))))
177       {
178           PD_ODR = 0b00000000; // SW1 a SW2 ON
179
180       }else if((!(PG_IDR &(1<<0)))&&(PG_IDR&(1<<1)))
181       {
182           PD_ODR = 0b00000100; // SW1 OFF SW2 ON
183
184       }else if((PG_IDR &(1<<0))&&(!(PG_IDR&(1<<1))))
185       {
186           PD_ODR = 0b00001000; // SW1 ON SW2 OFF
187
188       }
189
190   }
```

Obr. 3-26 I2C – funkce topeni

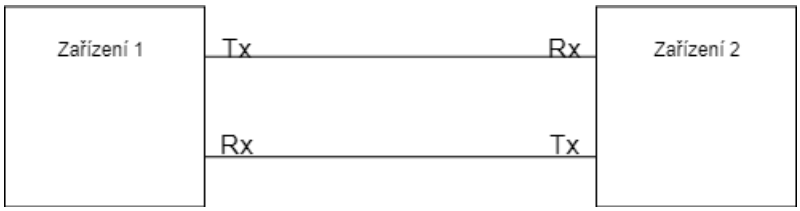


Obr. 3-27 I2C – vývojový diagram

3.4 Komunikační protokol UART

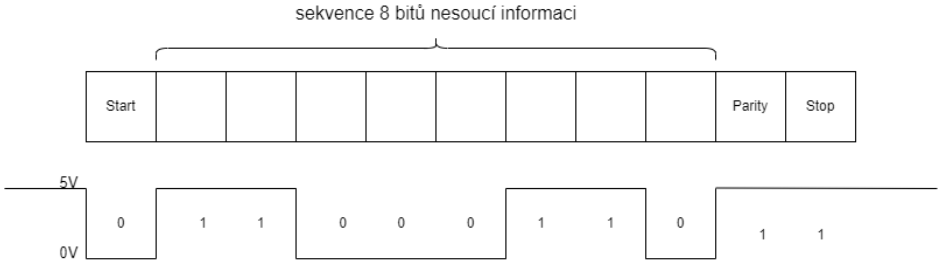
3.4.1 Princip UART komunikace

Uart, Universal Asynchronous Receiver/Transmitter, je asynchronní komunikační protokol umožňující komunikaci mezi dvěma a více zařízeními. Příklad použití je komunikace mezi počítačem a periferním zařízením jako motor nebo senzor. Komunikace může probíhat mezi několika zařízeními master, mezi jedním zařízením master a několika zařízení slave nebo pouze pro komunikaci mezi dvěma zařízeními bez rozdělení na master a slave. Ke komunikaci jsou využívány dva vodiče Rx pro čtení a Tx pro zápis. Jeden z vodičů používá jedno zařízení pro zápis a následně druhé zařízení ze stejného vodiče přijímá informaci od prvního zařízení. U druhého vodiče je princip obrácený.



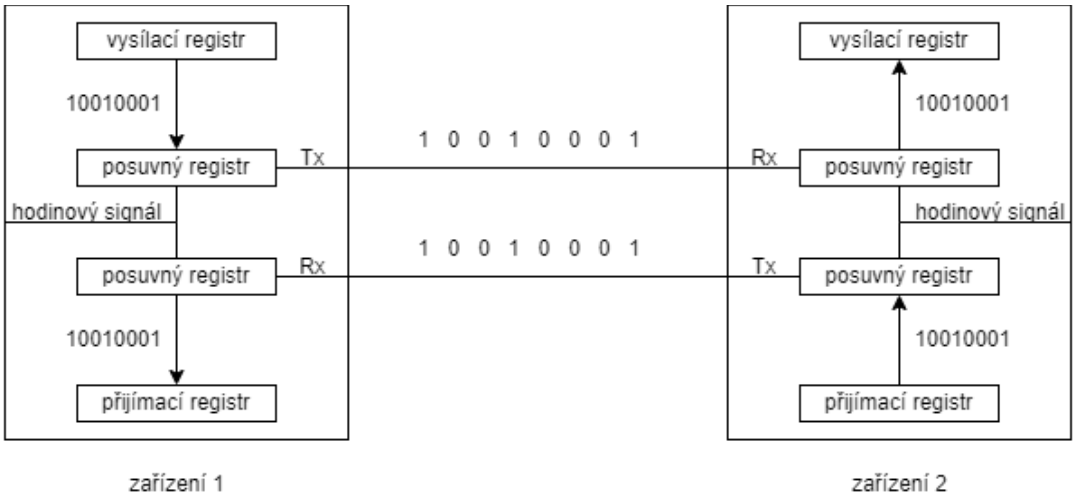
Obr. 3-28 UART komunikace

Komunikace na krátké vzdálenosti probíhá v rozmezí napětí 0-5 V se sekvencí deseti až jedenácti bitů. Pokud je ve vodiči napětí 5 V, znamená to, že neprobíhá žádný přenos. Pro začátek komunikace musí vysílající zařízení vygenerovat nulové napětí pro jednu periodu nastavených hodin. Nulové napětí během jedné periody hodin se nazývá start bit. Po start bitu následuje kombinace osmi bitů nesoucí informaci. V této kombinaci bitů nulové napětí znamená bitovou nulu a napětí 5 V bitovou jedničku. Pro konec komunikace po sekvenci osmi bitů vyšle vysílající zařízení bitovou jedničku značící stop bit. Před stop bitem je možné používat parity bit pro kontrolu, jestli nedošlo k problému během přenosu informace v sekvenci osmi bitů.



Obr. 3-29 UART – bitová kombinace

Před samotným přenosem vysílající zařízení přesune najednou obsah vysílacího registru do posuvného registru. Z posuvného registru jsou následně data vysílána bit po bitu do posuvného registru druhého zařízení. Následně se data najednou přesunou z posuvného registru do přijímacího registru.



Obr. 3-30 UART – průběh přenosu

Výhodami UART komunikace je jednoduchost komunikačního rozhraní, méně potřebného hardwaru a žádné komplikace s adresováním zařízení. Nevýhodami jsou nutná synchronizace hodinového signálu, přenosová rychlost, kdy přenosová rychlost se řídí největší možnou rychlostí

nejpomalejšího zařízení. Dále zde není žádná zpětná vazba o přijetí informace od vysílající zařízení. Pro komunikaci na větší vzdálenosti dochází k rušení hladiny 5 V, a proto se využívá na větší vzdálenosti hladin +13 V a -13 V s konvertory například RS232 nebo RS422.

3.4.2 Zadání vzorového příkladu

Úkolem je, aby mikroprocesor posílal uživateli na počítač přes aplikaci PuTTy návrh na zadání čísla 1-8. Uživatel by měl nejprve obdržet výzvu pro zadání čísla a následně pod sebou vypsané možnosti čísel. Při každé nabídce jednoho z čísel se na mikroprocesoru zároveň rozsvítí led dioda odpovídající danému číslu. Pokud uživatel zadá jedno z vybraných čísel rozsvítí se následně daný počet led diod a mikroprocesor vyšle zpět do počítače zprávu obsahující zadané číslo. V případě zadání jiného znaku nebo čísla, než je v nabídce mikroprocesor vyšle uživateli zprávu zpět do PuTTy o zadání znaku mimo požadovaný rozsah.

3.4.3 Nastavení

- Konstanta PB_DDR_INI pro registr PB_DDR – nastaví bránu B jako výstupní
- Konstanta PB_ODR_INI pro registr PB_ODR – zhasnutí celé brány B
- Konstanta PB_CR1_INI pro registr PB_CR1 – nastaví push-pull výstup
- Konstanta UART2_PSCR_INI pro registr UART2_PSCR – dělení mikroprocesorových hodin jedničkou
- Konstanta UART2_BRR1_INI pro registr UART_BRR1 – nastaví množství přenesených bitů za sekundu, 9600 bd
- Konstanta UART2_BRR2_INI pro registr UART_BRR2 – nastaví množství přenesených bitů za sekundu, 9600 bd
- Konstanta UART2_CR1_INI pro registr UART_CR1 – spustí komunikaci UART s prvním start bitem, osmi datovými bity a n-tým stop bitem, nastaví metodu probuzení a nečinnost parity bitu
- Konstanta UART2_CR2_INI pro registr UART2_CR2 – nastaví generování přerušení na konci přenosu, pokud jsou data po přijetí připravena k přečtení nebo pokud dojde k chybě přeskočení

V souboru *uartInit.c* dojde k inicializaci potřebných periferií. Možnosti nastavení lze najít v dokumentu UM0817 User Manual *[4].*

3.4.4 Řešení vzorového příkladu

V hlavním programu nejprve dojde k zavolání funkcí pro inicializaci. Následně se vstoupí do nekonečné smyčky, kde nejprve dojde k zavolání funkce pro odeslání nového řádku na terminál PuTTy. Následně dojde k vyslání řetězce znaků *Zadej cislo 1-8* zakončeného znakem pro začátek nového řádku a znakem skoku na nový řádek.

```
45      novyradek(); //odeslání nového řádku
46
47      for(j = 0; j < 15; j++ )
48      {
49          while(!(UART2_SR &(1<<6))); //dokud není ukončen přenos
50          UART2_DR = zadej[j]; // řetězec Zadej cislo 1-8
51          while(!(UART2_SR &(1<<6))); //dokud není ukončen přenos
52      }
53
54      novyradek();
```

Obr. 3-31 UART – vyslání výzvy pro zadání čísla

Následně se vstoupí do cyklu vypisování čísel jedna až osm pomocí řetězce se současným rozsvícením LED v pořadí podle nabízených možností čísel. Zároveň v tomto *for* cyklu je podmínka pro kontrolu, jestli nedošlo k vyslání znaku ze strany počítače. Pokud došlo k zapsání na straně počítače, dojde k rozsvícení zadaného počtu LED na mikroprocesorovém přípravku a výpisu zadaného čísla v terminálu PuTTy. Jestliže zadaný znak není číslo, nerozsvítí se žádná LED a dojde k výpisu na terminál PuTTy, že je uživatel mimo rozsah se zadaným znakem. Po zpracování načteného znaku z počítače program opustí *for* cyklus a celý algoritmus se opakuje od prvního řetězce *Zadej cislo 1-8.*

```
56          for( i = 0; i < 8; i++)
57          {
58              PB_ODR = cisla [i]; // postupné rozsvěcení LED
59
60
61              if(UART2_SR &(1<<5)) //pokud je načten znak z počítače
62              {
63                  prijem = UART2_DR; // uložení načteného znaku
64                  receive(); // rozsvícení LED v případě správného zadání znaku
65                  cekej(); // zpoždění
66                  nacteno =1; // pro opuštění for cyklu
67                  nacten(); // výpis zadaného znaku nebo upozornění na vstup mimo rozsah
68
69
70              }else{
71
72                  UART2_DR = vypis[i]; // postupný výpis čísel 1-8
73                   while(!(UART2_SR &(1<<6))); // čekám na konec přenosu
74                   zpozdeni(); //zpoždění
75                   novyradek(); // výpis nového řádku
76              }
77
78              if(nacteno ==1)
79              {
80                   i = 8;
81                   nacteno = 0;
82              }
83
84          }
```

Obr. 3-32 UART – for cyklus s postupným výpisem čísel, rozsvícení LED a kontrolou pro načtení znaků

Posílání dat z mikroprocesoru do počítače probíhá pomocí registru UART2_DR. Do tohoto registru se nahraje znak a následně se čeká na konec převodu dat podle registru UART2_SR na šestém bitu. To samé platí u posílání dat, kdy načtený znak z počítače je uložen v registru UART2_DR, a proto je uložen do proměnné.

Zadaný znak z počítače je uložen z registru UART2_DR do proměnné *prijem*. Následně se zavolá funkce *receive*, která v případě zadání čísla v požadovaném rozsahu, rozsvítí počet LED podle zadaného čísla. Jednotlivé podmínky hledají shodu s čísly jedna až osm v ascii kódu.

```
91   void receive(void)
92   {
93       if(prijem == 49)
94       {
95           PB_ODR = LED0;
96
97       }else if(prijem == 50)
98       {
99           PB_ODR = LED1;
100
101      }else if(prijem == 51)
102      {
103          PB_ODR = LED2;
104
105      }else if(prijem == 52)
106      {
107          PB_ODR = LED3;
108
109      }else if(prijem == 53)
110      {
111          PB_ODR = LED4;
112
113      }else if(prijem == 54)
114      {
115          PB_ODR = LED5;
116
117      }else if(prijem == 55)
118      {
119          PB_ODR = LED6;
120
121      }else if(prijem == 56)
122      {
123          PB_ODR = LED7;
124      }
125  }
```

Obr. 3-33 UART – funkce receive pro rozsvícení požadovaného počtu LED

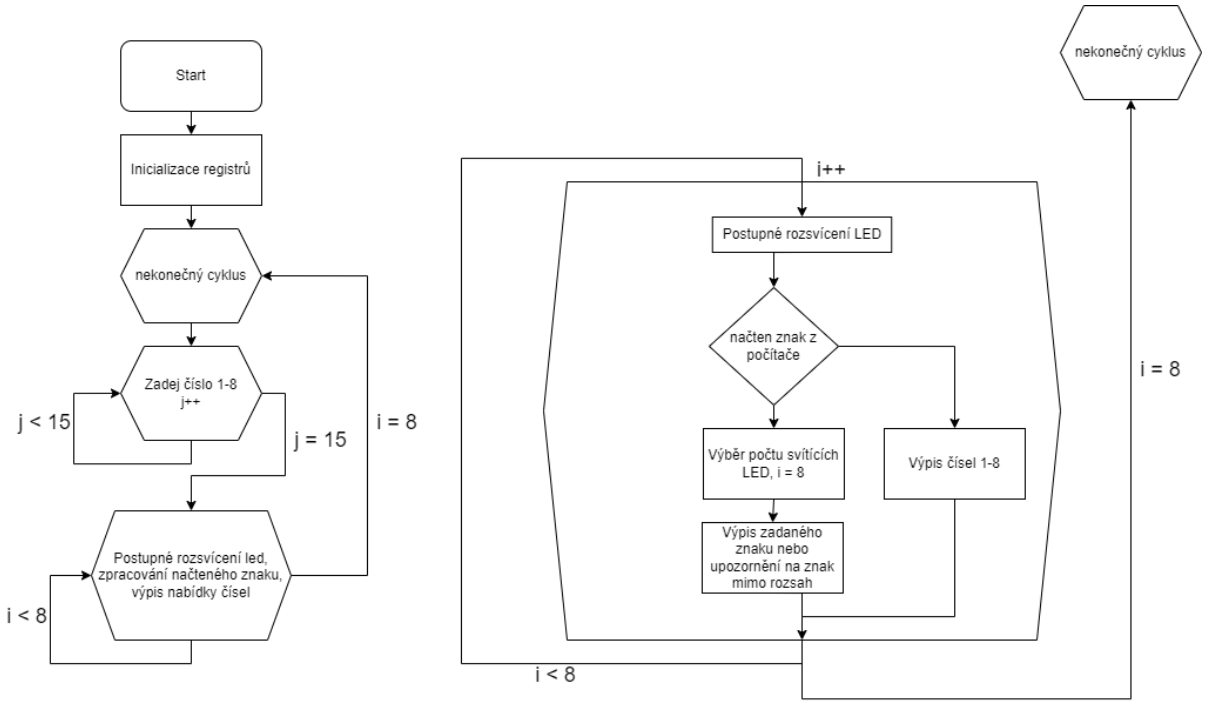
```
51   #define LED0 0b11111110
52   /*!<
53    svítí 1
54   */
55   #define LED1 0b11111100
56   /*!<
57    svítí 2
58   */
59   #define LED2 0b11111000
60   /*!<
61    svítí 3
62   */
63   #define LED3 0b11110000
64   /*!<
65    svítí 4
66   */
67   #define LED4 0b11100000
68   /*!<
69    svítí 5
70   */
71   #define LED5 0b11000000
72   /*!<
73    svítí 6
74   */
75   #define LED6 0b10000000
76   /*!<
77    svítí 7
78   */
79   #define LED7 0b00000000
80   /*!<
81    svítí 8
82   */
```

Obr. 3-34 UART – konstanty pro funkci receive

Dále se zavolá funkce *nacten*, která vypíše uživateli znak, který zadal nebo vypíše upozornění na znak mimo rozsah. Odeslání do terminálu PuTTy znovu proběhne přes registr UART2_DR ve *for* cyklu s kontrolou konce přenosu v registru UART2_SR. Na závěr se pošle do počítače na terminál PuTTy zadaný znak.

```
130  void nacten(void)
131  {
132      if(prijem > 48 && prijem <57)
133      {
134          for(k = 0; k < 11; k++)
135          {
136              while(!(UART2_SR &(1<<6))); //dokud není ukončen přenos
137              UART2_DR = zadano[k];
138              while(!(UART2_SR &(1<<6))); //dokud není ukončen přenos
139          }
140
141      }else{
142
143          for(k = 0; k < 17; k++)
144          {
145              while(!(UART2_SR &(1<<6))); //dokud není ukončen přenos
146              UART2_DR =rozsah[k];
147              while(!(UART2_SR &(1<<6))); //dokud není ukončen přenos
148          }
149      }
150
151      UART2_DR = prijem;
152      while(!(UART2_SR &(1<<6))); //dokud není ukončen přenos
153      novyradek();
154      zpozdeni();
155  }
```

Obr. 3-35 UART – ukázka funkce nacten

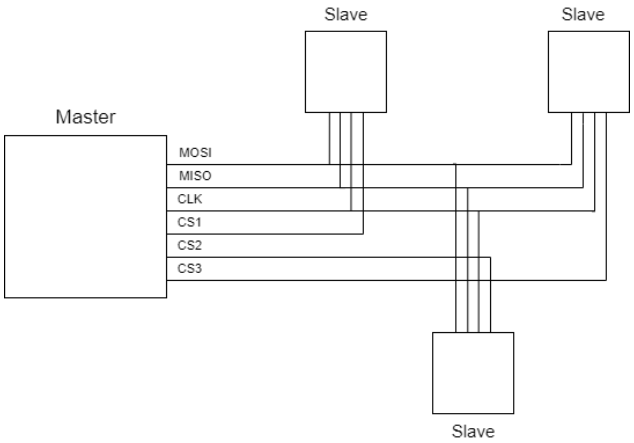


Obr. 3-36 UART – vývojový diagram

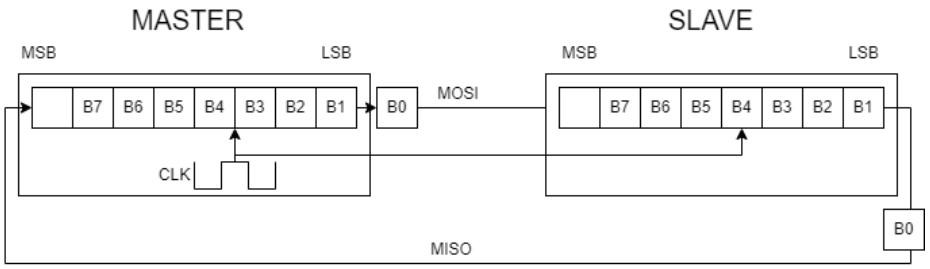
3.5 Komunikační protokol SPI

3.5.1 Princip SPI

SPI, serial peripheral interface, je sériový synchronní komunikační protokol. Zařízení mezi sebou komunikují se stejným hodinovým signálem. Rychlost komunikace může dosahovat 8 Mbit/s. Jedno zařízení master může být připojeno k několika zařízením slave. Data jsou vysílána ve dvou vodičích MOSI a MISO. Vodič MOSI znamená master out a slave in, kdy master zařízení posílá data zařízení slave, které data přijímá. Vodič MISO znamená master in a slave out, kdy naopak slave zařízení vysílá data a master zařízení data přijímá. Dále je pro komunikaci potřeba hodinový signál poskytnutý zařízením master. Pro výběr slave zařízení, se kterým chce zařízení master komunikovat se používá vodič chip select. Komunikace s daným zařízením slave neprobíhá, pokud je na vodiči chip select napětí 5 V. Pro komunikaci s daným zařízením slave musí napětí na vodiči chip select klesnout k nule.



Obr. 3-37 SPI–princip



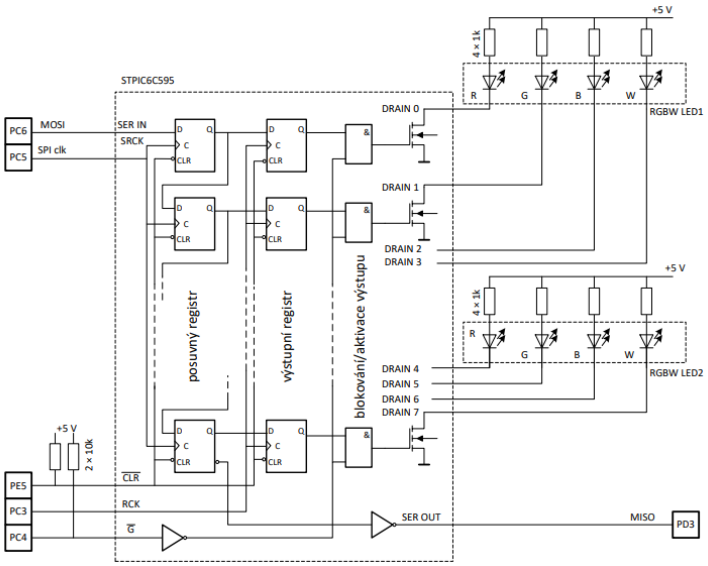
Obr. 3-38 SPI – přenos

Samotný přenos začíná vygenerováním hodinového signálu od zařízení master. Tím se se vyšle nejméně významný bit zařízení master na místo nejvíce významného bitu zařízení slave a naopak nejméně významný bit zařízení slave se vyšle na místo nejvíce významného bitu zařízení master.

3.5.2 Nastavení

- Konstanta PB_DDR_INI pro registr PB_DDR – nastaví bránu B jako výstupní
- Konstanta PB_ODR_INI pro registr PB_ODR – zhasnutí celé brány B
- Konstanta PB_CR1_INI pro registr PB_CR1 – nastaví push-pull výstup
- Konstanta SPI_CR1_INI pro registr SPI_CR1 – nejvíce významný bit bude odeslán jako první, spuštění periferie SPI, určení počtu symbolů přenesených za sekundu, určení polarity hodin a fáze hodin, vybrání master konfigurace
- Konstanta SPI_CR2_INI pro registr SPI_CR2 – nastaví jednosměrnou komunikaci, režim přijímání a full duplex komunikaci
- Konstanta PE_DDR_SPI pro registr PE_DDR – nastaví pátý pin jako výstupní pro SPI CLR
- Konstanta PE_CR1_SPI pro registr PE_CR1 – nastaví pátý pin jako push-pull pro SPI CLR
- Konstanta PC_DDR_SPI pro registr PC_DDR – nastaví výstup pro SPI RCK, SPI/G a SPI clock
- Konstanta PC_CR1_SPI pro registr PC_CR1 – nastaví push-pull pro SPI MISO, SPI MOSI, SPI RCK, SPI/G a SPI clock

V souboru *spiInit.c* proběhne inicializace jednotlivých periferií. Možnosti nastavení lze najít v dokumentu UM0817 User Manual *[4].*



Obr. 3-39 SPI – připojení posuvného registru [7]

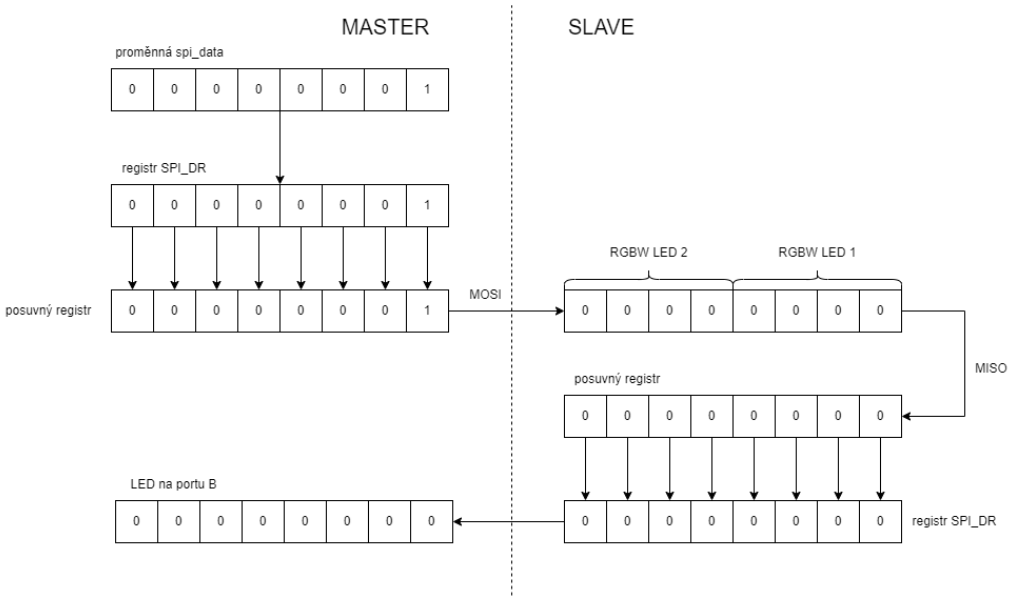
3.5.3 Řešení vzorového příkladu

V hlavním souboru *spi.c* jako první dojde k zavolání inicializačních funkcí, dojde k ukončení nulování posuvného a výstupního registru a proměnné *spi_data* se přiřadí počáteční hodnota. Poté se vstoupí do nekonečné smyčky. V nekonečné smyčce se do registru SPI_DR nahraje hodnota, která bude odeslána na RGBW LED. Vždy se jedná o jedničku, která se každý cyklus posune o jednu pozici doleva. V případě vynulování proměnné *spi_data* pro posun jedničky se jednička znovu nahraje na nultou pozici proměnné *spi_data*.

```
28      PBledInit();
29      spiInit();
30      spi_data = SPI_START;
31      PE_ODR |= 1UL << 5; //negace CLEAR posuvný a výstupní registr
32
33      while (1)
34      {
35          SPI_DR = spi_data;
36          spi_data = spi_data << 1; //posun o 1 doleva
37          if(spi_data == 0) // v případě rotace 1000 0000 dostaneme 0000 0000 a chceme 0000 0001
38          {
39              spi_data = 1;
40          }
```

Obr. 3-40 SPI – inicializace a posun jedničky

Data se z registru SPI_DR se nahrají do posuvného registru. Pro zapsání dat z posuvného registru do výstupního registru se nejprve odblokuje výstup a pomocí RCK na třetím bitu brány C dojde k zapsání dat. Tímto se rozsvítí jedna z RGBW LED. Po zapsání se zároveň přečte obsah registru SPI_DR zapsáním obsahu do proměnné *prijato*. Hodnota proměnné *prijato* se zneguje a pošle na bránu B. Jelikož první čtení z registru SPI_DR je nulové, bude svítící LED na bráně B zpožděna o jedno rozsvícení při poslání první hodnoty na RGBW LED.

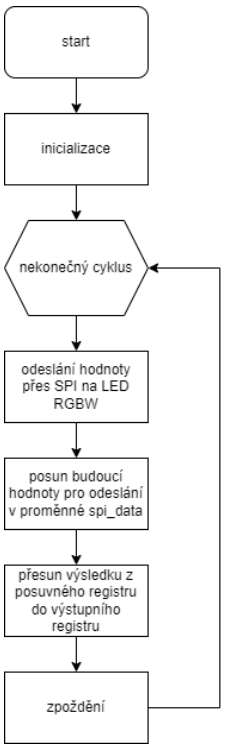


Obr. 3-41 SPI – vyslání první hodnoty z mikroprocesoru na RGBW LED

Na závěr dojde cyklus zpoždění zajistí viditelnost změn rozsvěcení a zhasínání.

```
42          PC_ODR |= 1UL << 4; //odblokování výstupu G
43
44          PC_ODR |= 1UL << 3; //umožnění převodu výsledku posuvného registru do výstupního registru RCK
45          PC_ODR &= ~(1UL <<3); //konec převodu výsledku posuvného registru do výstupního registru RCK
46
47          PC_ODR &= ~(1UL <<4);//zablokování výstupu G
48
49          prijato = SPI_DR;
50          PB_ODR = 255 -prijato; //rozsvícení PB
51
52
53          for(i = 0; i < 200; i++) //zpoždění
54          {
55              for(j = 0; j < 100; j++)
56              {
57                  for(k = 0; k < 3; k++){}
58              }
59          }
```

Obr. 3-42 SPI – práce s posuvným a výstupním registrem



Obr. 3-43 SPI – vývojový diagram

3.6 Reproduktor

3.6.1 Zadání vzorového příkladu

Vyberte si libovolné čtyři slyšitelné frekvence zvuku pro reproduktor a následně jednotlivé frekvence přiřaďte tlačítkům BT1 – BT4. Při stisknutí jednoho z tlačítek začne reproduktor vydávat zvuk dané frekvence. V případě, že není žádné tlačítko stisknuté, reproduktor nevydává žádný zvuk. Stisknutím jednoho z tlačítek se zároveň rozsvítí všechny LED na bráně B. Frekvenci zvuku vytvořte voláním přerušení pomocí časovače dva.

3.6.2 Nastavení

- Konstanta PB_DDR_INI pro registr PB_DDR – nastaví bránu B jako výstupní
- Konstanta PB_ODR_INI pro registr PB_ODR – zhasnutí celé brány B
- Konstanta PB_CR1_INI pro registr PB_CR1 – nastaví push-pull výstup

- Konstanta PA_CR1_BT pro registr PA_CR1 – nastaví push-pull vstup pro tlačítka BT1 – BT4
- Konstanta TIM2_ARR_INI pro registr TIM2_ARR – nastaví hodnotu, do které čítá čítač
- Konstanta TIM2_PSCR_INI pro registr TIM2_PSCR – nastaví hodnotu pro dělení frekvence mikroprocesoru
- Konstanta TIM2_CR1_INI pro registr TIM2_CR1 – umožní čítaní čítače
- Konstanta TIM2_IER_INI pro registr TIM2_IER – umožní vyvolání přerušení
- Konstanta PD_DDR_BEEP pro registr PD_DDR – čtvrtý pin jako výstupní pro reproduktor
- Konstanta PD_CR1_BEEP pro registr PD_CR1 – čtvrtý pin jako push-pull pro reproduktor

V souboru *reproInit.c* proběhne inicializace jednotlivých periferií. Možnosti nastavení lze najít v dokumentu UM0817 User Manual *[4].*

Pro vyvolávání přerušení pomocí časovače dva je třeba vytvořit funkci v souboru *stm8_interrupt_vector.c* a zapsat ji do tabulky vektorů. Vektor, do kterého zapsat funkci podprogramu přerušení lze nalézt na straně 44 v tomto manuálu *[5]*.

3.6.3 Řešení vzorového příkladu

Nejprve je třeba spočítat, s jakou frekvencí bude reproduktor spínán. Samotný mikroprocesor pracuje na 2 MHz a v nastavení jsem zvolil dělení této frekvence osmi. Pin s reproduktorem nastavuji do jedničky každé druhé vyvolání přerušení. Čítač bude tedy inkrementován s frekvencí 3906,25 Hz.

$$f_{inkrementace} = \frac{2 \cdot 10^6}{2^8 \cdot 2} = 3906{,}25\ Hz \tag{3-1}$$

Tab. 3-1 Tabulka zvolených frekvencí

Konstanta pro registr TIM2_ARR	Frekvence volání přerušení
1	3906,25 Hz
2	1953,125 Hz
4	976,56 Hz
8	488,28 Hz

Samotný algoritmus začne zavoláním funkcí pro inicializaci jednotlivých periferií. Následně se přejde do nekonečného cyklu. Během běhu tohoto cyklu budou všechny LED na bráně B zhasnuty. Dále probíhá kontrola stisknutí jednoho z tlačítek BT1 – BT4. V případě stisknutí jednoho z tlačítek dojde k nastavení registru TIM2_ARR, který udává hodnotu, do které čítač čítá. Zároveň dojde k rozsvícení LED na bráně B.

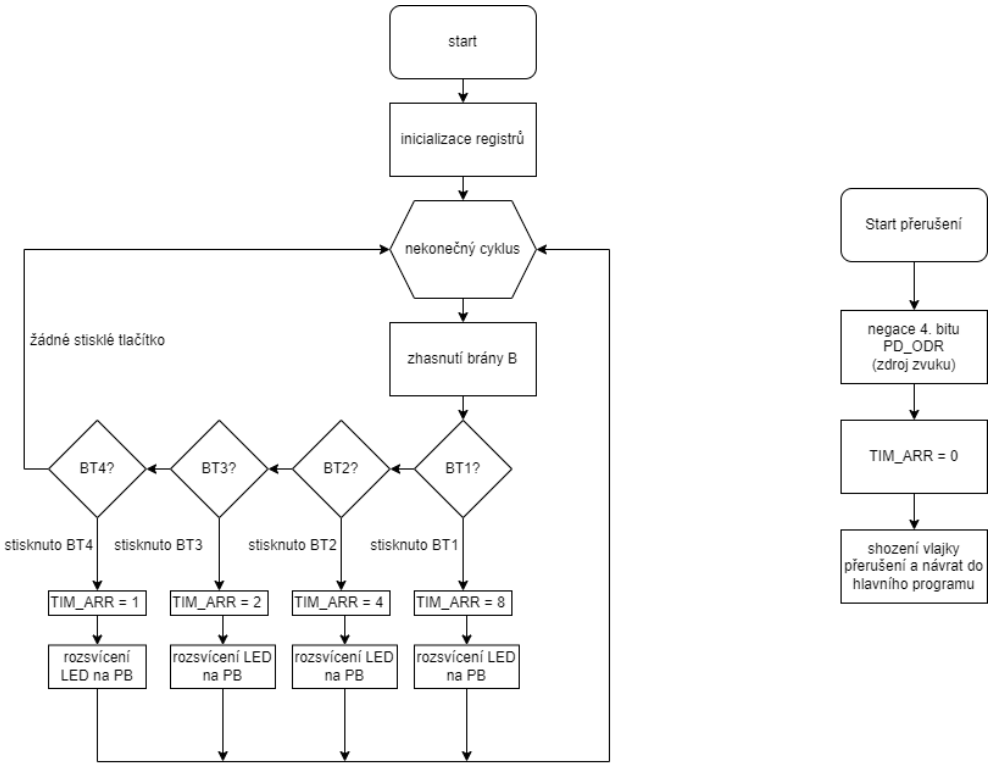
```
25      while (1)
26      {
27
28          PB_ODR = 0b11111111;
29
30          if(!(PA_IDR &(1<<6))) //pokud zmáčknuté BT1 na 6. bitu (úplně vlevo tlačítko)
31          {
32              TIM2_ARR =8;
33              PB_ODR = 0b00000000;
34
35          }else if(!(PA_IDR &(1<<5))) // pokud není zmáčknuté tlačítko BT3
36          {
37              TIM2_ARR =4;
38              PB_ODR = 0b00000000;
39
40          }else if(!(PA_IDR &(1<<4))) // pokud není zmáčknuté tlačítko BT3
41          {
42              TIM2_ARR =2;
43              PB_ODR = 0b00000000;
44
45
46          }else if(!(PA_IDR &(1<<3))) //pokud není zmáčknuté tlačítko BT4
47          {
48              TIM2_ARR =1;
49              PB_ODR = 0b00000000;
50          }
51      }
52  }
```

Obr. 3-44 Reproduktor – hlavní program

Při nastavení registru TIM2_ARR se čítá do nastavené hodnoty a následně se vyvolá přerušení. V podprogramu přerušení se zneguje čtvrtý bit na bráně D, registr TIM2_ARR se nastaví do nuly a na závěr se shodí vlajka přerušení.

```
58  void int_t2(void)
59  {
60      PD_ODR ^= (1 << 4); //negace
61      TIM2_ARR =0;
62      TIM2_SR1 = ~(1UL <<0); // shození vlajky přerušení
63  }
```

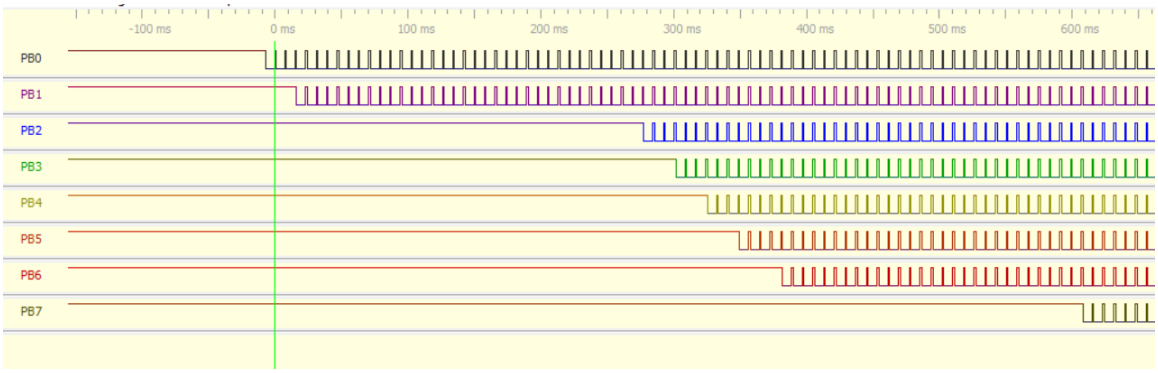
Obr. 3-45 Reproduktor – podprogram přerušení



Obr. 3-46 Reproduktor – vývojový diagram

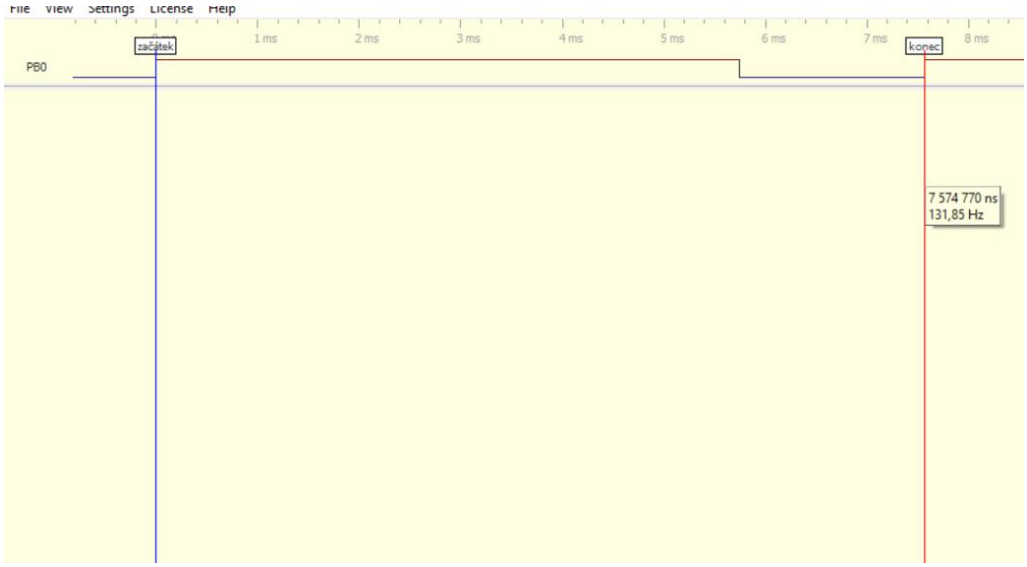
KAPITOLA 4: OVĚŘENÍ FUNKCE VZOROVÝCH PŘÍKLADŮ

4.1 Potenciometry POT1, POT2 a osm zelených LED



Obr. 4-1 Postupné rozsvěcení LED pomocí potenciometru POT1

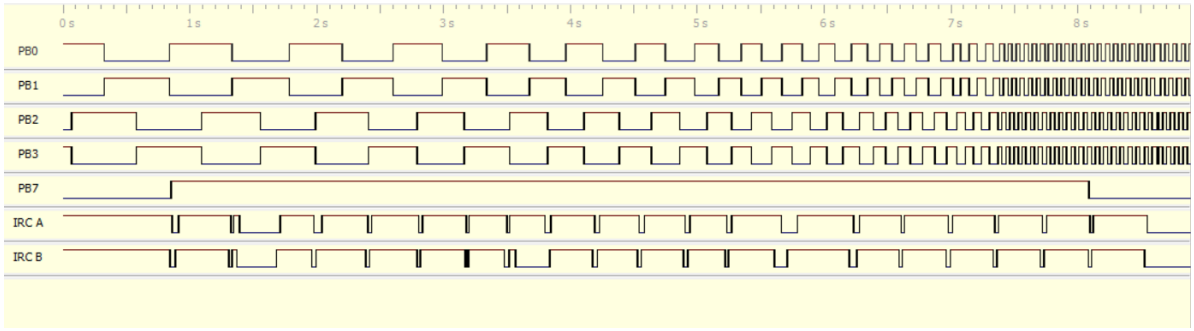
Obrázek znázorňuje postupné rozsvěcení zelených LED na bráně B pomocí potenciometru POT1. Nepravidelnost rozsvěcení LED je způsobené nerovnoměrným otáčením potenciometru rukou uživatele.



Obr. 4-2 Detail intervalu pulzně šířkového řízení jasu

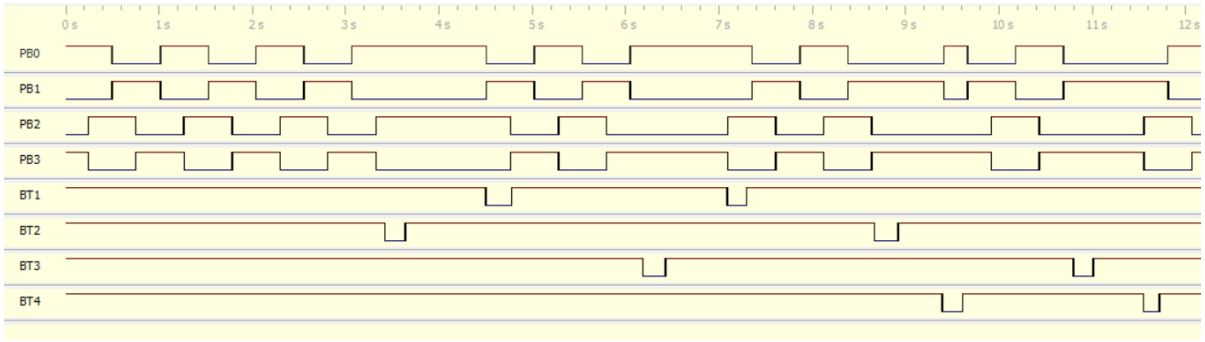
Obrázek ukazuje detail intervalu pulzně šířkového řízení jasu. První část intervalu v jedničce odpovídá zhasnuté LED na nultém pinu brány B a druhá část odpovídá rozsvícené LED. Naměřený časový interval odpovídá přibližně časovému intervalu změřenému během simulace.

4.2 Motorek, čidlo polohy, tlačítka a inkrementální čidlo



Obr. 4-3 Zvyšování rychlosti pomocí inkrementálního čidla

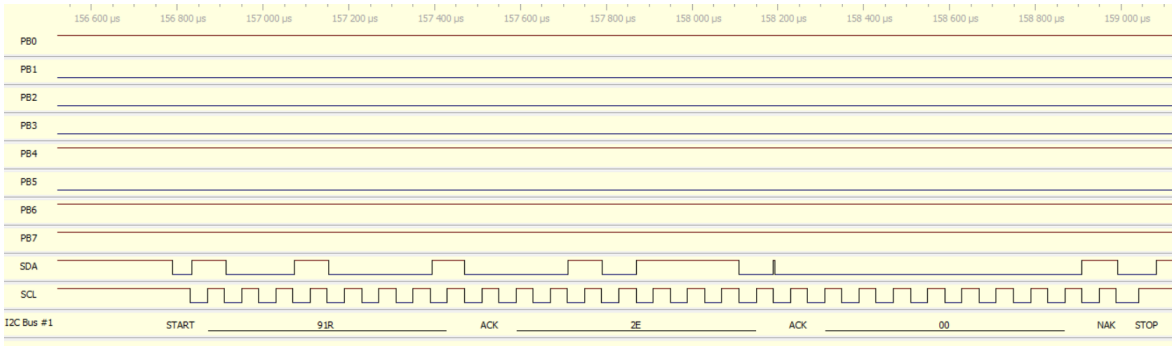
Na obrázku lze vidět postupné zrychlování motorku z nejpomalejší rychlosti po největší rychlost. Dosažení maximální a minimální rychlosti je symbolizováno rozsvícením osmé LED na sedmém pinu brány B.



Obr. 4-4 Ovládání motorku pomocí tlačítek

Na obrázku lze vidět stisknutí tlačítka BT2, které motorek zastaví. Po stisknutí tlačítka BT1 se motorek znovu rozjede. Při stisknutí tlačítka BT3 se motorek znovu zastaví a rozjede se až po stisknutí tlačítka BT1. V případě znovu zastavení motorku tlačítkem BT2 se motorek rozjede po stisknutí tlačítka BT4. Stisknutím tlačítka BT3 se motorek znovu zastaví a rozjede se po stisknutí tlačítka BT4.

4.3 Komunikační protokol I2C

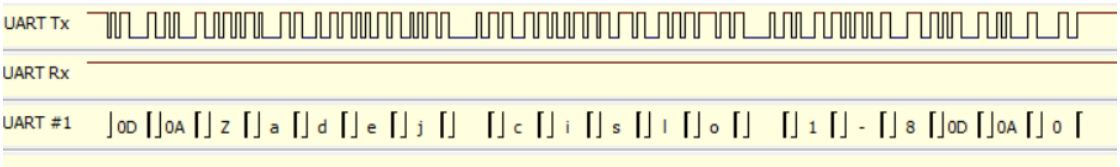


Obr. 4-5 Přenos hodnoty teploty po lince I2C

Na obrázku lze vidět přenos přes komunikační protokol I2C. Přenos začíná start bitem a pokračuje adresou zařízení 91 v hexadecimálním vyjádření a písmenem R pro čtení ze zařízení. Hexadecimální číslo 91 je v binární soustavě 10010001. Po adrese zařízení následuje první byte

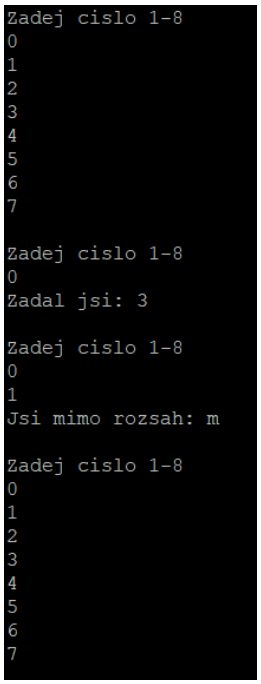
v hexadecimální soustavě 2E ukončeným acknowledge bitem. Hexadecimální číslo 2E je v binární soustavě 00101110. Po acknowledge bitu následuje druhý byte 00 v binární soustavě 00000000. Komunikace je ukončena bitem not acknowledge a stop bitem. Informaci o teplotě udává jedenáct nejvíce významných bitů, tedy číslo 00101110000 v decimální soustavě je to číslo 368. Po vynásobení čísla 368 číslem 0,125 se dojde k výsledku 46 °C. Pokud se podíváme na bránu B, tak lze spatřit číslo 11010001, kdy hodnota v nule znamená svítící LED. Jestliže se číslo 11010001 přepíše tak, aby hodnota jedna znamenala svítící LED, poté se dojde k číslu 00101110, které v decimální soustavě znamená číslo 46. Brána B, tedy správně ukazuje změřenou hodnotu poslanou po sériové lince I2C.

4.4 Komunikační protokol UART



Obr. 4-6 Přenos znaků přes sériovou linku UART „Zadej cislo 1-8“

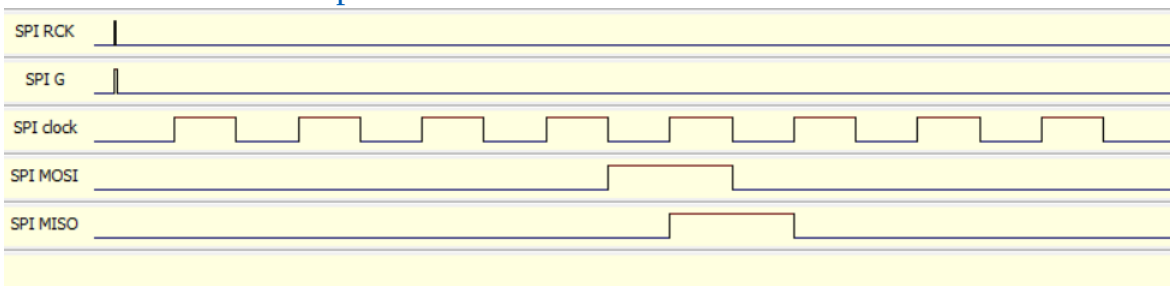
Na obrázku lze vidět aktivní vodič pro zápis Tx, který vysílá zprávu *Zadej cislo 1-8* se znaky pro posun kurzoru na nový řádek a znaku posun kurzoru na začátek řádku.



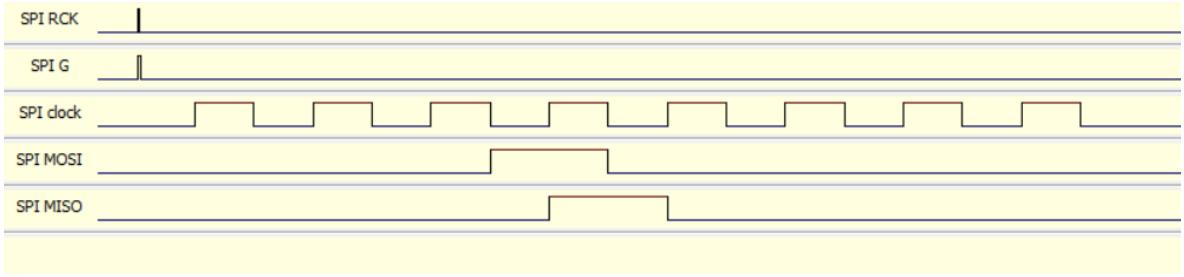
Obr. 4-7 Výpis na terminálu PuTTy

Obrázek z terminálu PuTTy dokazuje správnou funkčnost zadaného příkladu pro komunikaci UART. Mikroprocesor vysílá na terminál PuTTy *Zadej cislo 1-8* s následnou nabídkou jednotlivých čísel. V případě zadání jednoho z čísel se zadané číslo vypíše na terminál. Pokud uživatel zadá znak mimo čísla 1-8, mikroprocesor vypíše uživateli na terminál *Jsi mimo rozsah:* se znakem, který uživatel zadal.

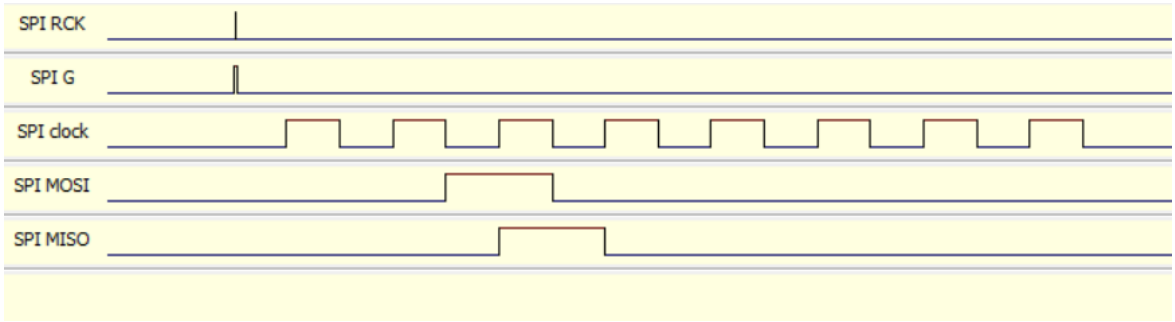
4.5 Komunikační protokol SPI



Obr. 4-8 SPI – odeslání čísla na MOSI 00001000

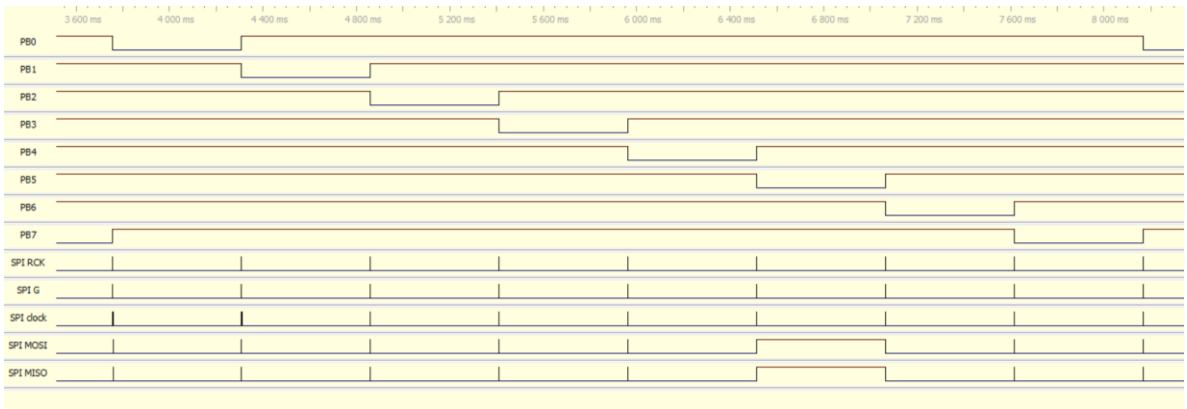


Obr. 4-9 SPI – odeslání čísla na MOSI 00010000



Obr. 4-10 SPI – odeslání čísla na MOSI 00100000

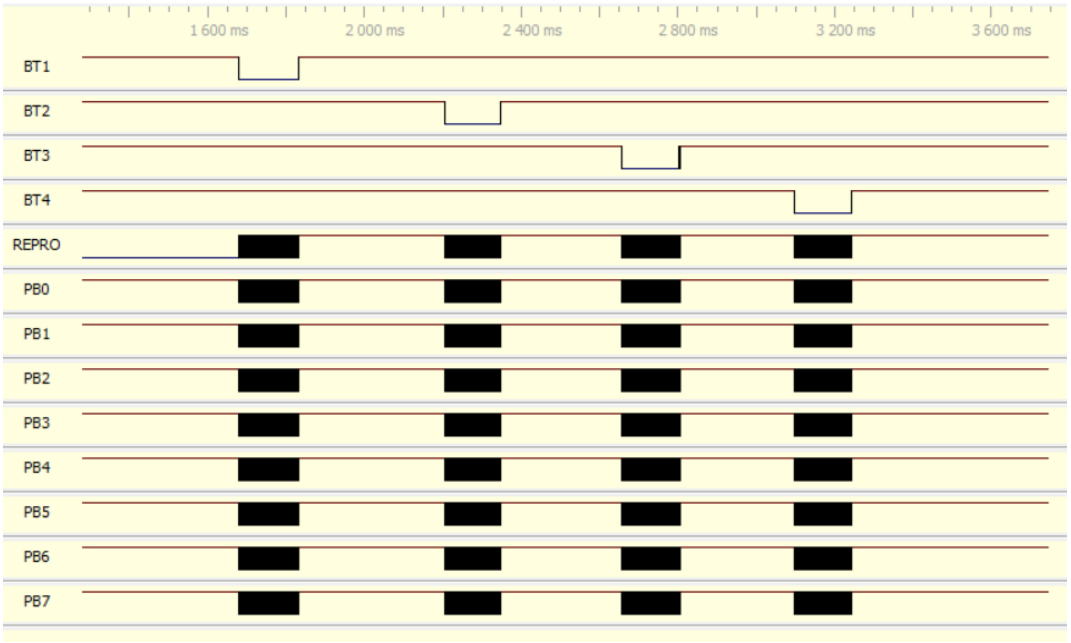
Snímky ukazují postupné odeslání čísel 00001000, 00010000 a 00100000. Tyto čísla jsou dána proměnnou *spi_data* vložené do registru SPI_DR a vyslané na vodič MOSI. Přes vodič MISO je poslána předchozí hodnota vodiče MOSI.



Obr. 4-11 SPI – postupné rozsvěcení LED na bráně B

Obrázek ukazuje postupné rozsvěcení LED hodnotami poslanými přes vodiče MOSI a MISO. Po vyslání posledních dat zůstávají vodiče MISO a MOSI v jedničce do začátku dalšího přenosu inicializovaným hodinovým signálem.

4.6 Reproduktor



Obr. 4-12 Stisknutí tlačítek BT1-BT4 a reakce na reproduktoru a bráně B pomocí LED

Obrázek ukazuje postupné stisknutí tlačítek BT1, BT2, BT3 a BT4. Při každém stisknutí tlačítka se rozsvítí všechny LED na bráně B a reproduktor začne vydávat zvuk.

KAPITOLA 5: ZHODNOCENÍ PRÁCE

Prvním úkolem této práce bylo seznámení s konstrukcí a vybavením školního mikropočítače. Tomuto úkolu se věnovala kapitola s popisem přípravku vysvětlující možnosti poskytující školním přípravkem. Následně v kapitolách pro jednotlivé periferie byl vysvětlen princip jednotlivých periferií pro pochopení práce s periferiemi.

Druhým úkolem bylo navržení koncepce programového vybavení. Pro tuto práci bylo použito jedno z mnoha možností vývojových prostředí, a to vývojové prostředí ST Visual Develop.

Třetím a čtvrtým úkolem této práce bylo navržení, zpracování a ověření funkcí jednotlivých periferií. Tento úkol je zpracován v kapitole vzorové příklady a kapitole ověření. Kapitola vzorové příklady jako první vysvětluje princip periferií, pro které jsou následně vytvořené zadání vzorových příkladů použití dané periferie. Před samotným řešením vzorových příkladů bylo popsáno nastavení dané periferie konstantami nadefinované v projektu v hlavičkových souborech použité v inicializačních souborech. Poté jsou popsány řešení jednotlivých vzorových příkladů. V kapitole ověření funkce vzorových příkladů byla ověřena funkce jednotlivých vzorových příkladů. Vzorové příklady fungují správně podle jednotlivých zadání.

Pátým úkolem bylo využití verzovacího systému pro zpracování vzorových příkladů. Jako verzovací systém byl vybrán systém Git popsaný v první kapitole. Při vypracování vzorových příkladů bylo tedy využito verzovacího systému Git.

Šestým a posledním úkolem bylo zpracování dokumentace vzorových příkladů pomocí vhodného nástroje pro generování dokumentace. Jako nástroj pro generování dokumentace byl vybrán nástroj Doxygen popsaný v první kapitole. Při vypracování vzorových příkladů bylo tedy využito nástroje Doxygen.

Tato práce uvede začátečníka do světa mikroprocesorů a může posloužit jako stavební kámen pro realizaci nejrůznějších myšlenek a nápadů s využitím mikroprocesorů.

LITERATURA

[1] *Chacon, Scott. Pro Git. [online]. 2.* vydání. Berkeley: Apress, 2014 *[cit. 2023-05-02]. ISBN 978-1484200773.* Dostupné z: https://knihy.nic.cz/files/nic/edice/scott_chacon_pro_git.pdf

[2] DOXYGEN. Commands in the Doxygen input file *[online]. [cit. 2023-05-02].* Dostupné z: https://doxygen.nl/manual/commands.html#cmdreturn

[3] STMicroelectronics. STM8S-Discovery: STM8S105C6T6 MCU discovery kit. *[online].* 2017 *[cit. 2023-05-02].* Dostupné z: https://www.st.com/resource/en/user_manual/cd00250600-stm8sdiscovery-stmicroelectronics.pdf

[4] STMicroelectronics RM0016: STM8S Series and STM8AF Series 8-bit Microcontrollers, Reference Manual *[online]. [cit. 2023-05-02].* Dostupné z: https://www.st.com/resource/en/user_manual/cd00250600-stm8sdiscovery-stmicroelectronics.pdf

[5] STMicroelectronics. STM8S105C6- Mainstream Value line 8-bit MCU with 32 Kbytes Flash, 24 MHz CPU, integrated EEPROM, STM8S105C6T6 datasheet *[online].* 2021 *[cit. 2023-05-02].* Dostupné z: https://www.st.com/resource/en/datasheet/stm8s105c6.pdf

[6] NXP Semiconductors. LM75B: Digital temperature sensor and thermal watchdog *[online].* Rev. 6. September 2019 *[cit. 2023-05-02].* Dostupné z: https://www.nxp.com/docs/en/data-sheet/LM75B.pdf

[7] *KÜNZEL, K.,* Přípravek PPS 2021: provizorní studijní materiál [interní materiál]. Verze 1.1. Praha, 2021. 16 s. ČVUT FEL v Praze.

PŘÍLOHA A: DOKUMENTACE DOXYGEN

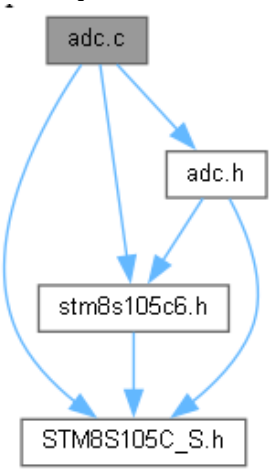
A.1 Potenciometry POT1, POT2 a osm zelených LED

A.1.1 adc.c

Potenciometry POT1, POT2 a osm zelených LED.

```
#include "stm8s105c6.h"
#include "stm8s105c_s.h"
#include "adc.h"
```

Graf závislostí na vkládaných souborech pro adc.c:



Funkce

- **main** ()

  hlavní program před nekonečnou smyčkou proběhne inicializace brány B a AD převodníku

  funkce nejprve načte výsledek AD převodu z kanálu AIN9 POT1 pro postupné rozsvěcení LED, výsledek zpracuje na osmibitové číslo

  následně dojde k načtenái výsledku na kanálu AIN8 z POT2, výsledek se zracuje na osmibitové číslo podle hodnot zpracovaných hodnot se rozsvítí určitý počet LED s určitou intenzitou svícení

Proměnné

- long **POT1**
- long **POT2**
- long **POT2_OPAK**
- int **i**

Detailní popis

Potenciometry POT1, POT2 a osm zelených LED.

Autor

Miloš Mlejnek

Datum

1.3. 2023

Verze

1.0

Dokumentace funkcí

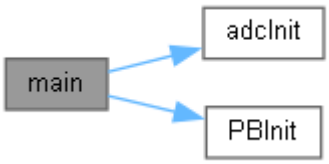
main ()

hlavní program před nekonečnou smyčkou proběhne inicializace brány B a AD převodníku

funkce nejprve načte výsledek AD převodu z kanálu AIN9 POT1 pro postupné rozsvěcení LED, výsledek zpracuje na osmibitové číslo

následně dojde k načtenái výsledku na kanálu AIN8 z POT2, výsledek se zracuje na osmibitové číslo podle hodnot zpracovaných hodnot se rozsvítí určitý počet LED s určitou intenzitou svícení

Tato funkce volá...



Dokumentace proměnných

int i

pro vnitřní cyklus for cyklu na vytvoření zpoždění

long POT1

pro převod z adc kanálu 1000

long POT2

pro převod z adc kanálu 1001

long POT2_OPAK

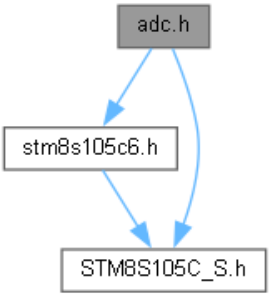
pro 1023-adc

A.1.2 adc.h

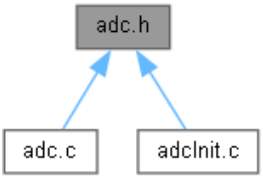
Potenciometry POT1, POT2 a osm zelených LED - konstanty.

```
#include "stm8s105c6.h"
#include "stm8s105c_s.h"
```

Graf závislostí na vkládaných souborech pro adc.h:



Následující graf ukazuje, které soubory přímo nebo nepřímo vkládají tento soubor:



Definice maker

- #define **ADC_CR1_INI** 0b01110000
- #define **ADC_CR1_ADON** 0b01110001

- #define **ADC_CR2_INI** 0b00001000
- #define **ADC_CSR_AIN8** 0b00001000
- #define **ADC_CSR_AIN9** 0b00001001
- #define **ADC_CSR_EOF** 0b10001001
- #define **ADC_TDRH_INI** 0b00000011
- #define **PB_DDR_INI** 0b11111111
- #define **PB_ODR_INI** 0b11111111
- #define **PB_CR1_INI** 0b11111111
- #define **ADROZSAH** 1023
- #define **ADMAX** 255
- #define **LED0** 0b11111111
- #define **LED1** 0b11111110
- #define **LED2** 0b11111100
- #define **LED3** 0b11111000
- #define **LED4** 0b11110000
- #define **LED5** 0b11100000
- #define **LED6** 0b11000000
- #define **LED7** 0b10000000
- #define **LED8** 0b00000000

Funkce

- void **adcInit** (void)
  inicializace AD převodníku

- void **PBInit** (void)
  inicializace brány B

Detailní popis

Potenciometry POT1, POT2 a osm zelených LED - konstanty.

Autor

Miloš Mlejnek

Datum

1.3. 2023

Verze

1.0

Dokumentace definic maker

#define ADC_CR1_ADON 0b01110001

spustí AD převodník

#define ADC_CR1_INI 0b01110000

nastaví dělení frekvence mikroprocesoru 2 MHz (fMASTER/18), režim samotného převodu v CONT bitu a spuštění převodu ADON bitem

#define ADC_CR2_INI 0b00001000

nastaví zarovnání převedených dat doprava, tedy osm nejméně významných bitů bude v registru ADC_DRL a dva zbývající bity v registru ADC_DRH

#define ADC_CSR_AIN8 0b00001000

nastaví čtení z kanálu 1000, kde je potenciometr POT1

#define ADC_CSR_AIN9 0b00001001

nastaví čtení z kanálu 1001, kde je potenciometr POT2

#define ADC_CSR_EOF 0b10001001
ukončení převodu AD převodníku
#define ADC_TDRH_INI 0b00000011
snížení spotřeby statické energie pro vstupy obou potenciometrů na pinu PE6 a PE7
#define ADMAX 255
rozsah PB_ODR 8 bit
#define ADROZSAH 1023
rozsah ad převodníku 10 bit
#define LED0 0b11111111
#define LED1 0b11111110
#define LED2 0b11111100
#define LED3 0b11111000
#define LED4 0b11110000
#define LED5 0b11100000
#define LED6 0b11000000
#define LED7 0b10000000
#define LED8 0b00000000
#define PB_CR1_INI 0b11111111
nastaví push–pull výstup
#define PB_DDR_INI 0b11111111
nastaví bránu B jako výstupní
#define PB_ODR_INI 0b11111111
zhasnutí celé brány B

Dokumentace funkcí
void adcInit (void)

inicializace AD převodníku
Tuto funkci volají...

void PBInit (void)

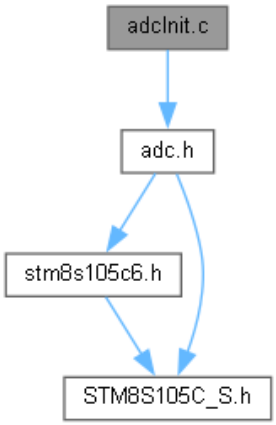
inicializace brány B
Tuto funkci volají...

A.1.3 adcInit.c

Potenciometry POT1, POT2 a osm zelených LED - inicializace.

```
#include "adc.h"
```

Graf závislostí na vkládaných souborech pro adcInit.c:



Funkce

- void **adcInit** ()
  inicializace AD převodníku

- void **PBInit** ()
  inicializace brány B

Detailní popis

Potenciometry POT1, POT2 a osm zelených LED - inicializace.

Autor

Miloš Mlejnek

Datum

22.4. 2023

Verze

1.0

Dokumentace funkcí

void adcInit (void)

inicializace AD převodníku

Tuto funkci volají...

void PBInit (void)

inicializace brány B

Tuto funkci volají...

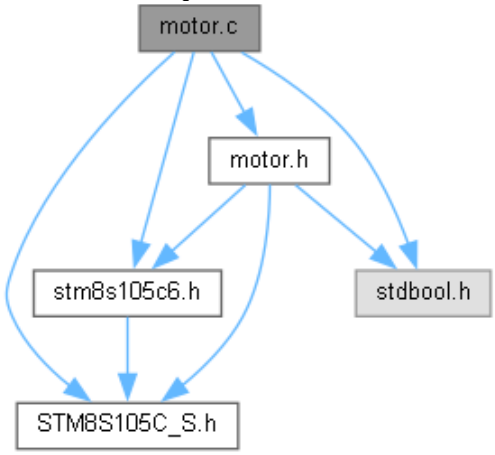
A.2 Motorek, čidlo polohy, tlačítka a inkrementální čidlo

A.2.1 motor.c

Motorek, čidlo polohy, tlačítka a inkrementální čidlo.

```
#include "stm8s105c6.h"
#include "stm8s105c_s.h"
#include "motor.h"
#include <stdbool.h>
```

Graf závislostí na vkládaných souborech pro motor.c:



Funkce

- far interrupt void **T4Init** (void)
  podprogram přerušení pravidelně se dekrementuje proměnná DEK pro vstup do podmínky, kde dochází k spínání vinutí motorku, čímž je řízena rychlost ¨funkce zajišťuje spínání vinutí motorku

- **main** ()
  hlavní program nejprve proběhne inicializace brány B, optického čidla, inkrementálního čidla, motorku, tlačítek, timeru 4, funkce pro určení polohy v nekonečné smyčce se zkontroluje stisknutí tlačítek BT1-BT4 a jestli bylo dosaženo maximální nebo minimální rychlosti pohybu motorku a popřiadě se rozsvítí osmá LED brány B

- void **motInit** (void)
  Nastavení Portu E pro odblokování budiče.

- void **docidlaInit** (void)
  Uvodní nastavení proměnných pro ovládání motorku.

Proměnné

- unsigned char **pozice**
- unsigned char **pozice_max** = 250
- unsigned char **polohaDEK**
- unsigned char **DEK**
- **T_dekoder dekoderPC**
- unsigned char **INKC**
- unsigned char **INKC_INI** = 50
- unsigned char **stav**
- unsigned char **sepnout**
- bool **DEKMI**
- bool **DEKMA**

Detailní popis

Motorek, čidlo polohy, tlačítka a inkrementální čidlo.

Autor

Miloš Mlejnek

Datum

10.3. 2023

Verze

1.0

Dokumentace funkcí

void docidlaInit (void)

Uvodní nastavení proměnných pro ovládání motorku.

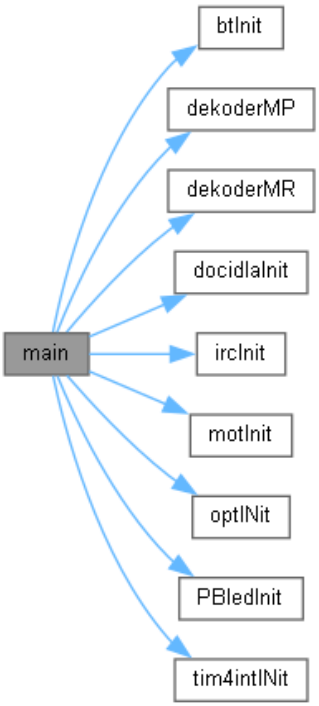
po dojezdu do čidla polohy se nastaví proměnná pozice do nuly

Tuto funkci volají...

main ()

hlavní program nejprve proběhne inicializace brány B, optického čidla, inkrementálního čidla, motorku, tlačítek, timeru 4, funkce pro určení polohy v nekonečné smyčce se zkontroluje stisknutí tlačítek BT1-BT4 a jestli bylo dosaženo maximální nebo minimální rychlosti pohybu motorku a popřiadě se rozsvítí osmá LED brány B

Tato funkce volá...



void motInit (void)

Nastavení Portu E pro odblokování budiče.

PE_CR1 0.bit do 1 pro odblokování budiče

PE_DDR 0.bit do 1 - output mode

Tuto funkci volají...

void T4Init (void)

podprogram přerušení pravidelně se dekrementuje proměnná DEK pro vstup do podmínky, kde dochází k spínání vinutí motorku, čímž je řízena rychlost ¨funkce zajišťuje spínání vinutí motorku

podprogram přerušení

Tato funkce volá...

Dokumentace proměnných

unsigned char DEK

uložení hodnoty z Dčidla

bool DEKMA

uložení výsledku pro dosazení maximalní rychlosti

bool DEKMI

uložení výsledku pro dosazení minimalní rychlosti

T_dekoder dekoderPC

struktura z **motor.h**, T_stavDek stav, unsigned char pocetHran

unsigned char INKC

pro rychlost motorku

unsigned char INKC_INI = 50

unsigned char polohaDEK

poloha dekodéru

unsigned char pozice

0-250 je rozmezí pohybu motorku, doprava počítám od 0 do 250, doleva odečítám 250 až 0

unsigned char pozice_max = 250

krajní pravá poloha

unsigned char sepnout

pro vybrání kombinace spínání motorku

unsigned char stav

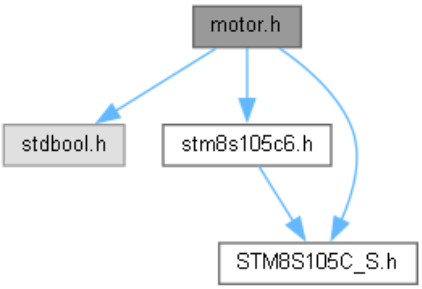
stav pohybu - pravá, levá, stojí, jede

A.2.2 motor.h

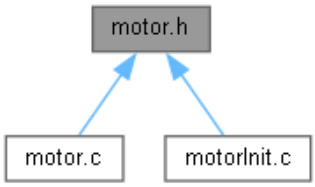
Motorek, čidlo polohy, tlačítka a inkrementální čidlo - konstanty.

```
#include <stdbool.h>
#include "stm8s105c6.h"
#include "stm8s105c_s.h"
```

Graf závislostí na vkládaných souborech pro motor.h:



Následující graf ukazuje, které soubory přímo nebo nepřímo vkládají tento soubor:



Datové struktury

- struct **T_dekoder**

Definice maker

- #define **PD_DDR_OPT** 0b00000000
- #define **PD_CR1_OPT** 0b10000000
- #define **PC_CR1_IRC** 0b00000110
- #define **PB_DDR_CW** 0b11111111
- #define **PB_ODR_INI** 0b11111111
- #define **PB_CR1_CW** 0b11111111
- #define **PE_CR1_MOT** 0b00000001
- #define **PE_DDR_MOT** 0b00000001
- #define **PA_CR1_BT** 0b01111000
- #define **TIM4_ARR_INI** 0b11111010
- #define **TIM4_PSCR_INI** 0b00000111
- #define **TIM4_CR1_INI** 0b00000001
- #define **TIM4_IER_INI** 0b00000001
- #define **DEKODER_MAX** 64
- #define **DEKODER_MIN** 0
- #define **NULAZ** 0b11111001
- #define **JEDNAZ** 0b11111010
- #define **DVAZ** 0b11110110
- #define **TRIZ** 0b11110101
- #define **NULAR** 0b01111001
- #define **JEDNAR** 0b01111010
- #define **DVAR** 0b01110110
- #define **TRIR** 0b01110101

Výčty

- enum **T_stavDek** { **S0**, **S00**, **S01**, **S11**, **S10**, **E** }

Funkce

- void **PBledInit** (void)
  Inicializace PB_ODR.

- void **PDledInit** (void)
- void **optINit** (void)
  Inicializace čidla polohy.

- void **ircInit** (void)
  inicializace inkrementálního čidla

- void **motInit** (void)
  Nastavení Portu E pro odblokování budiče.

- void **btInit** (void)
  Inicializace Portu A - tlačítka BT1, BT2, BT3, BT4 na bitech 6-3.

- void **tim4intINit** (void)
  inicializaceTIM4

- void **docidlaInit** (void)
  Uvodní nastavení proměnných pro ovládání motorku.

- unsigned char **dekoder** (**T_dekoder** *filter, bool stopaA, bool stopaB)
  funkce pro inkrementální čidlo

- bool **dekoderMP** (**T_dekoder** *filter)
  porovná jestli je dosaženo maximální rychlosti

- bool **dekoderMR** (**T_dekoder** *filter)
  porovná jestli bylo dosaženo minimální rychlosti

Detailní popis

Motorek, čidlo polohy, tlačítka a inkrementální čidlo - konstanty.

Autor

Miloš Mlejnek

Datum

10.3. 2023

Verze

1.0

Dokumentace definic maker

#define DEKODER_MAX 64

maximální hodnota natočená na inkrementalním čidle

#define DEKODER_MIN 0

minimální hodnota natočená na inkrementalním čidle

#define DVAR 0b01110110

kombinace napájení motoru 2

#define DVAZ 0b11110110

kombinace napájení motoru 2

#define JEDNAR 0b01111010

kombinace napájení motoru 1

#define JEDNAZ 0b11111010

kombinace napájení motoru 1

#define NULAR 0b01111001

kombinace napájení motoru 0

#define NULAZ 0b11111001

kombinace napájení motoru 0

#define PA_CR1_BT 0b01111000

nastaví push-pull vstup pro tlačítka BT1 – BT4

#define PB_CR1_CW 0b11111111

nastaví push-pull výstup

#define PB_DDR_CW 0b11111111

nastaví bránu B jako výstupní

#define PB_ODR_INI 0b11111111

zhasnutí celé brány B ″

#define PC_CR1_IRC 0b00000110

nastaví vstup na push-pull pro inkrementální čidlo

#define PD_CR1_OPT 0b10000000

nastaví vstup na push-pull pro optický snímač

#define PD_DDR_OPT 0b00000000

na sedmém pinu je optický snímač, který se nastaví do vstupního režimu

#define PE_CR1_MOT 0b00000001

nastaví push-pull výstup na nultém pinu brány E pro odblokování budiče motorku

#define PE_DDR_MOT 0b00000001

nastaví jako výstupní nultý pin brány E pro odblokování budiče motorku

#define TIM4_ARR_INI 0b11111010

nastaví hodnotu, do které čítá čítač

#define TIM4_CR1_INI 0b00000001

umožní čítaní čítače

#define TIM4_IER_INI 0b00000001

umožní vyvolání přerušení

#define TIM4_PSCR_INI 0b00000111

nastaví hodnotu pro dělení frekvence mikroprocesoru

#define TRIR 0b01110101

kombinace napájení motoru 3

#define TRIZ 0b11110101

kombinace napájení motoru 3

Dokumentace výčtových typů

enum T_stavDek

Hodnoty výčtu:

S0	počáteční stav
S00	kanál A = 0 kanál B = 0
S01	kanál A = 1 kanál B = 0
S11	kanál A = 1 kanál B = 1
S10	kanál A = 0 kanál B = 1
E	neočekávaná hodnota

Dokumentace funkcí

void btInit (void)

Inicializace Portu A - tlačítka BT1, BT2, BT3, BT4 na bitech 6-3.

Inicializace Portu A - tlačítka BT1, BT2, BT3, BT4 na bitech 6-3.

Tuto funkci volají...

unsigned char dekoder (T_dekoder * *filter*, bool *A*, bool *B*)

funkce pro inkrementální čidlo

stavový diagram

podle kanálů A,B určení stavu a podle stavu se přičítá počet hran podle, kterých se řídí rychlost

Tuto funkci volají...

bool dekoderMP (T_dekoder * *filter*)

porovná jestli je dosaženo maximální rychlosti

maximum pro nejpomalejší chod

Tuto funkci volají...

bool dekoderMR (T_dekoder * *filter*)

porovná jestli bylo dosaženo minimální rychlosti

maximum pro nejrychlejší chod

Tuto funkci volají...

void docidlaInit (void)

Uvodní nastavení proměnných pro ovládání motorku.

po dojezdu do čidla polohy se nastaví proměnná pozice do nuly

Tuto funkci volají...

void ircInit (void)

inicializace inkrementálního čidla

Tuto funkci volají...

void motInit (void)

Nastavení Portu E pro odblokování budiče.
PE_CR1 0.bit do 1 pro odblokování budiče
PE_DDR 0.bit do 1 - output mode

Tuto funkci volají...

void optINit (void)

Inicializace čidla polohy.

Tuto funkci volají...

void PBledInit (void)

Inicializace PB_ODR.
Inicializace PB_ODR.
Inicializace PB_ODR.
Inicializace PB_ODR.
Inicializace PB_ODR.

void PDledInit (void)
void tim4intINit (void)

inicializaceTIM4

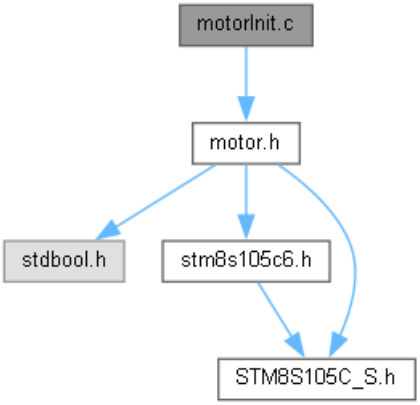
Tuto funkci volají...

A.2.3 motorInit.c

Motorek, čidlo polohy, tlačítka a inkrementální čidlo - inicializace.

```
#include "motor.h"
```

Graf závislostí na vkládaných souborech pro motorInit.c:



Funkce

- void **PBledInit** (void)
  inicializace PB_ODR LED

- void **optINit** (void)
  Inicializace čidla polohy.

- void **btInit** (void)
  Inicializace Portu A - tlačítka BT1, BT2, BT3, BT4 na bitech 6-3.

- void **tim4intINit** (void)
  inicializaceTIM4

- void **ircInit** (void)
  inicializace inkrementálního čidla

- unsigned char **dekoder** (**T_dekoder** *filter, bool A, bool B)
  funkce pro inkrementální čidlo

- bool **dekoderMP** (**T_dekoder** *filter)
  porovná jestli je dosaženo maximální rychlosti

- bool **dekoderMR** (**T_dekoder** *filter)
  porovná jestli bylo dosaženo minimální rychlosti

Detailní popis

Motorek, čidlo polohy, tlačítka a inkrementální čidlo - inicializace.

Autor

Miloš Mlejnek

Datum

10.3. 2023

Verze

1.0

Dokumentace funkcí

void btInit (void)

Inicializace Portu A - tlačítka BT1, BT2, BT3, BT4 na bitech 6-3.

unsigned char dekoder (T_dekoder * *filter*, bool *A*, bool *B*)

funkce pro inkrementální čidlo
podle kanálů A,B určení stavu a podle stavu se přičítá počet hran podle, kterých se řídí rychlost
Tuto funkci volají...

bool dekoderMP (T_dekoder * *filter*)

porovná jestli je dosaženo maximální rychlosti
maximum pro nejpomalejší chod
Tuto funkci volají...

bool dekoderMR (T_dekoder * *filter*)

porovná jestli bylo dosaženo minimální rychlosti
maximum pro nejrychlejší chod
Tuto funkci volají...

void ircInit (void)

inicializace inkrementálního čidla

Tuto funkci volají...

void optINit (void)

Inicializace čidla polohy.

Tuto funkci volají...

void PBledInit (void)

inicializace PB_ODR LED
Inicializace PB_ODR.

void tim4intINit (void)

inicializaceTIM4

Tuto funkci volají...

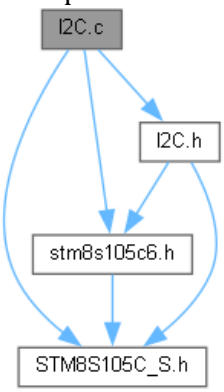
A.3 Komunikační protokol I2C

A.3.1 I2C.c

Komunikační protokol I2C.

```
#include "stm8s105c6.h"
#include "stm8s105c_s.h"
#include "I2C.h"
```

Graf závislostí na vkládaných souborech pro I2C.c:



Funkce

- **main** ()
  hlavní program
- void **nastaveni** ()
  funkce pro nastavení registrů
- void **write** ()
  funkce pro zápis do registrů
- void **writetoread** ()
  funkce změní režim zápisu na režim čtení
- void **read** ()
  funkce pro čtení teploty
- void **LED** ()
  přepočet teploty a zobrazení na bráně B
- void **topeni** ()
  funkce pro přitápění

Proměnné

- unsigned int **teplota**

- unsigned int **hyst_t** = 0b0001001000000000
- unsigned int **over_t** = 0b0001001100000000
- unsigned char **prazdny**
- unsigned char **adresa**
- long **prevod**
- int **i**
- int **j**
- int **k**

Detailní popis

Komunikační protokol I2C.

Autor

Miloš Mlejnek

Datum

22.3. 2023

Verze

1.0

Dokumentace funkcí

void LED (void)

přepočet teploty a zobrazení na bráně B

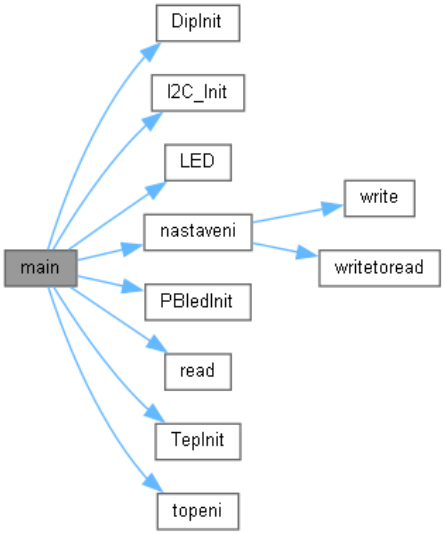
Tuto funkci volají...

main ()

hlavní program

nejprve proběhne inicializace brány B, přepínačů, topných odporů, komunikace I2C funkce pravidelně čte teplotu na rezistorech z teplotního čidla a hodnotu zobrazuje na bráně B pomocí LED binárně

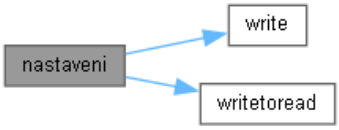
Tato funkce volá...



void nastaveni (void)

funkce pro nastavení registrů

Tato funkce volá...



Tuto funkci volají...

void read (void)

funkce pro čtení teploty

Tuto funkci volají...

void topeni (void)

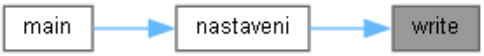
funkce pro přitápění

Tuto funkci volají...

void write (void)

funkce pro zápis do registrů

Tuto funkci volají...



void writetoread (void)

funkce změní režim zápisu na režim čtení

Tuto funkci volají...

Dokumentace proměnných

unsigned char adresa

adresa registrů

unsigned int hyst_t = 0b0001001000000000

int i

int j

int k

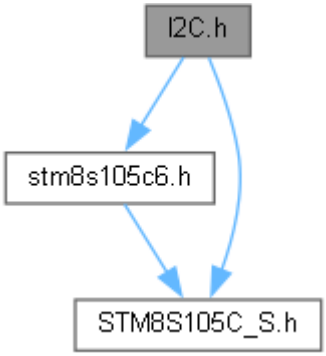
unsigned int over_t = 0b0001001100000000
unsigned char prazdny
čtení I2C_SR3 vyčistí registr I2C_SR1
long prevod
přepočet teploty
unsigned int teplota
hodnota teploty

A.3.2 I2C.h

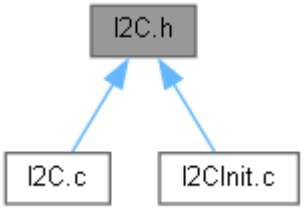
Komunikační protokol I2C - konstanty.

```
#include "stm8s105c6.h"
#include "stm8s105c_s.h"
```

Graf závislostí na vkládaných souborech pro I2C.h:



Následující graf ukazuje, které soubory přímo nebo nepřímo vkládají tento soubor:



Definice maker

- #define **PB_DDR_INI** 0b11111111
- #define **PB_ODR_INI** 0b11111111
- #define **PB_CR1_INI** 0b11111111
- #define **PG_CR1_DIP** 0b00000011
- #define **PD_DDR_TEP** 0b00001100
- #define **PD_CR1_TEP** 0b00001100
- #define **I2C_FREQR_INI** 0b00010000
- #define **I2C_TRISER_INI** 0b00010001
- #define **I2C_CCRL_INI** 0b01010000
- #define **I2C_CCRH_INI** 0b00000000
- #define **I2C_CR1_INI** 0b00000001
- #define **HYSTEREZE** 0b00000010
- #define **TEMP_ADRW** 0b10010000
- #define **OVERTEMP** 0b00000011
- #define **PTR_TEMP** 0b00000000
- #define **TEMP_ADRR** 0b10010001

Funkce

- void **PBledInit** (void)

Inicializace PB_ODR.

- void **DipInit** (void)
  Inicializace přepínačů

- void **TepInit** (void)
  Inicializace topných odporů a LED na bráně D.

- void **I2C_Init** (void)
  Inicializace komunikace I2C.

- void **nastaveni** (void)
  funkce pro nastavení registrů

- void **write** (void)
  funkce pro zápis do registrů

- void **read** (void)
  funkce pro čtení teploty

- void **writetoread** (void)
  funkce změní režim zápisu na režim čtení

- void **LED** (void)
  přepočet teploty a zobrazení na bráně B

- void **topeni** (void)
  funkce pro přitápění

Detailní popis

Komunikační protokol I2C - konstanty.

Autor

Miloš Mlejnek

Datum

22.3. 2023

Verze

1.0

Dokumentace definic maker

#define HYSTEREZE 0b00000010

#define I2C_CCRH_INI 0b00000000

umožnění periferií

#define I2C_CCRL_INI 0b01010000

kontrola hodin SCL

#define I2C_CR1_INI 0b00000001

Peripheral enable

#define I2C_FREQR_INI 0b00010000

nastaví časování periferních vnitřních hodin na 16 MHz

#define I2C_TRISER_INI 0b00010001

nastaví maximální dobu trvání zpětné vazby SCL

#define OVERTEMP 0b00000011

#define PB_CR1_INI 0b11111111

nastaví push-pull výstup

#define PB_DDR_INI 0b11111111

nastaví bránu B jako výstupní

#define PB_ODR_INI 0b11111111

zhasnutí celé brány B

#define PD_CR1_TEP 0b00001100

nastaví push-pull výstup pro topné rezistory a LED na bráně D

#define PD_DDR_TEP 0b00001100

nastaví výstup pro topné rezistory a LED na bráně D

#define PG_CR1_DIP 0b00000011

nastaví push-pull vstup pro přepínače

#define PTR_TEMP 0b00000000

#define TEMP_ADRR 0b10010001

adresa LM75B - address read

#define TEMP_ADRW 0b10010000

address write

Dokumentace funkcí

void DipInit (void)

Inicializace přepínačů

Tuto funkci volají...

void I2C_Init (void)

Inicializace komunikace I2C.

Tuto funkci volají...

void LED (void)

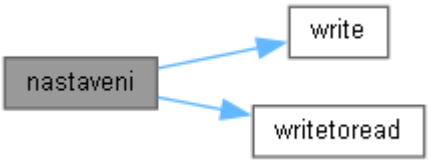
přepočet teploty a zobrazení na bráně B

Tuto funkci volají...

void nastaveni (void)

funkce pro nastavení registrů

Tato funkce volá...



Tuto funkci volají...

void PBledInit (void)

Inicializace PB_ODR.
Inicializace PB_ODR.
Inicializace PB_ODR.
Inicializace PB_ODR.
Inicializace PB_ODR.

Tuto funkci volají...

void read (void)

funkce pro čtení teploty

Tuto funkci volají...

void TepInit (void)

Inicializace topných odporů a LED na bráně D.

Tuto funkci volají...

void topeni (void)

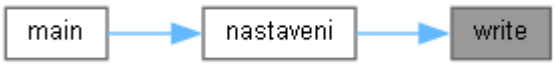
funkce pro přitápění

Tuto funkci volají...

void write (void)

funkce pro zápis do registrů

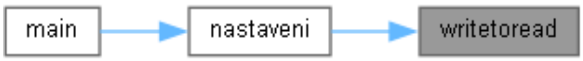
Tuto funkci volají...



void writetoread (void)

funkce změní režim zápisu na režim čtení

Tuto funkci volají...

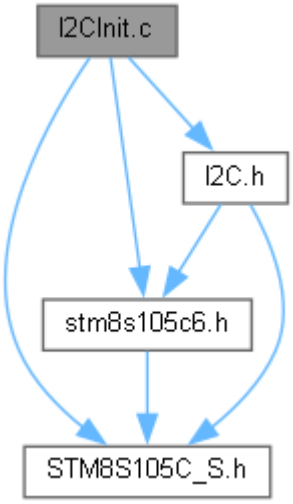


A.3.3 I2CInit.c

Komunikační protokol I2C - inicializace.

```
#include "stm8s105c6.h"
#include "stm8s105c_s.h"
#include "I2C.h"
```

Graf závislostí na vkládaných souborech pro I2CInit.c:



Funkce

- void **PBledInit** (void)
  Inicializace PB_ODR.

- void **DipInit** (void)
  Inicializace přepínačů

- void **TepInit** (void)
  Inicializace topných odporů a LED na bráně D.

- void **I2C_Init** (void)

Inicializace komunikace I2C.

Detailní popis

Komunikační protokol I2C - inicializace.

Autor

Miloš Mlejnek

Datum

22.3. 2023

Verze

1.0

Dokumentace funkcí

void DipInit (void)

Inicializace přepínačů

inicializace PG_CR1

Tuto funkci volají...

void I2C_Init (void)

Inicializace komunikace I2C.

Tuto funkci volají...

void PBledInit (void)

Inicializace PB_ODR.

void TepInit (void)

Inicializace topných odporů a LED na bráně D.

Tuto funkci volají...

A.4 Komunikační protokol UART

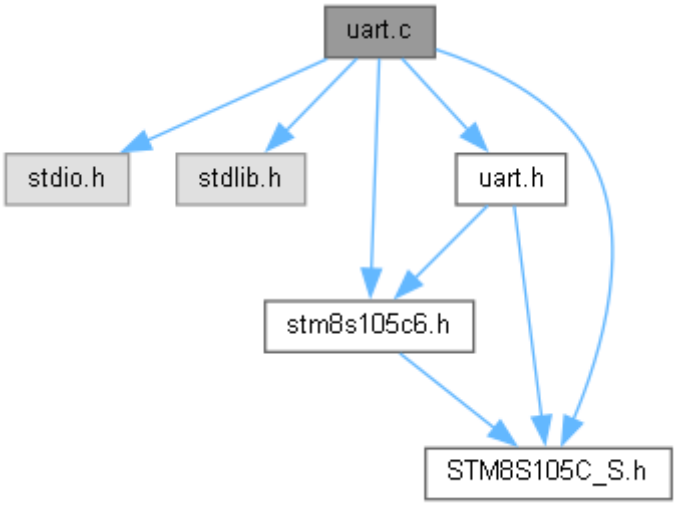
A.4.1 uart.c

Komunikační protokol UART.

```
#include <stdio.h>
#include <stdlib.h>
#include "uart.h"
#include "stm8s105c6.h"
```

```
#include "stm8s105c_s.h"
```

Graf závislostí na vkládaných souborech pro uart.c:



Funkce

- **main** ()
  hlavní program nejprve inicializace procesor konstantně posílá po lince Zadej číslo 1-8 a následně vypíše možnosti
  kdykoli během výpisu může uživatel zadat číslo 1-8 , kdy se rozsvítí číslem daný počet ledek
  v případě jiného znaku dostne uživatel výpis Jsi mimo rozsah

- void **receive** (void)
  kolik ledek se rozsvítí

- void **nacten** (void)
  výpis načteného znaku

- void **novyradek** (void)
  odeslaní nového řádku

Proměnné

- int **i**
- int **j**
- int **k**
- int **cisla** [] = { 254, 253, 251, 247, 239, 223, 191, 127}
- unsigned char **zadej** [] = "Zadej cislo 1-8"
- unsigned char **vypis** [] = "01234567"
- unsigned char **zadano** [] = "Zadal jsi: "
- unsigned char **rozsah** [] = "Jsi mimo rozsah: "
- unsigned char **prijem**
- char **radek** [] = {'\r','\n'}
- int **nacteno**

Detailní popis

Komunikační protokol UART.

Autor

Miloš Mlejnek

Datum

22.3. 2023

Verze

1.0

Dokumentace funkcí

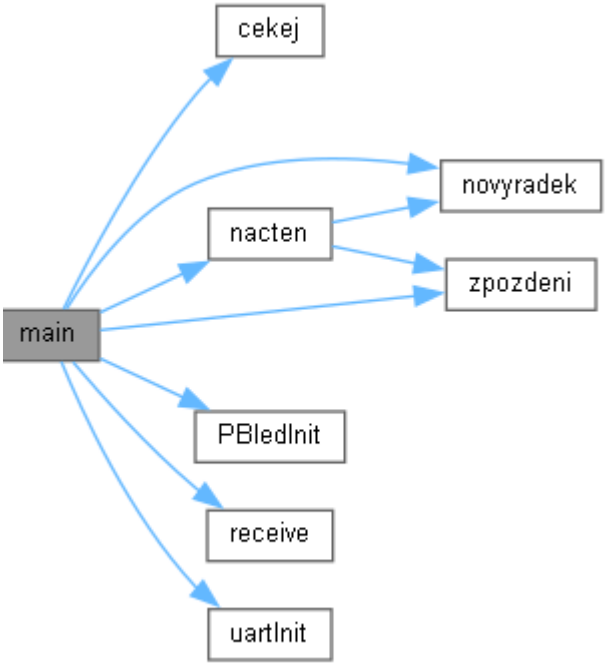
main ()

hlavní program nejprve inicializace procesor konstantně posílá po lince Zadej číslo 1-8 a následně vypíše možnosti

kdykoli během výpisu může uživatel zadat číslo 1-8 , kdy se rozsvítí číslem daný počet ledek

v případě jiného znaku dostne uživatel výpis Jsi mimo rozsah

Tato funkce volá...

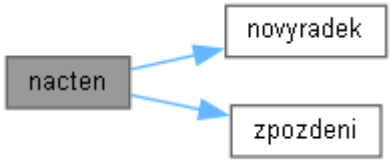


void nacten (void)

výpis načteného znaku

vypis nacteneho znaku

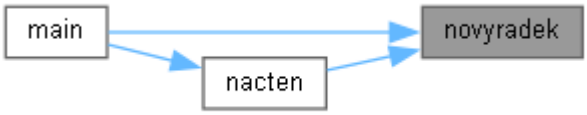
Tato funkce volá...



Tuto funkci volají...

void novyradek (void)

odeslaní nového řádku
odeslani noveho radku
Tuto funkci volají...



void receive (void)

kolik ledek se rozsvítí
kolik ledek se rozsviti
Tuto funkci volají...

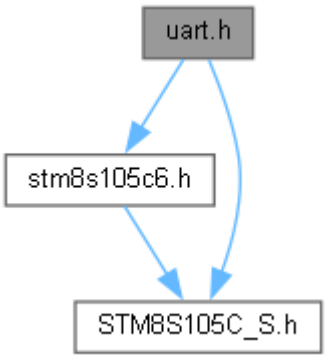
Dokumentace proměnných
int cisla[] = { 254, 253, 251, 247, 239, 223, 191, 127}
254 = 11111110, 253 = 11111101, ..., čísla pro svícení LED
int i
pro for cyklus
int j
pro for cyklus
int k
pro for cyklus
int nacteno
pro vyskočení z vypisovacího cyklu
unsigned char prijem
uložení hodnoty z UART2_DR, kterou zadal uživatel na počítači
char radek[] = {'\r','\n'}
unsigned char rozsah[] = "Jsi mimo rozsah: "
unsigned char vypis[] = "01234567"
unsigned char zadano[] = "Zadal jsi: "
unsigned char zadej[] = "Zadej cislo 1-8"

A.4.2 uart.h

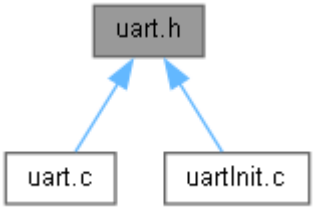
Komunikační protokol UART - konstanty.

```
#include "stm8s105c6.h"
#include "stm8s105c_s.h"
```

Graf závislostí na vkládaných souborech pro uart.h:



Následující graf ukazuje, které soubory přímo nebo nepřímo vkládají tento soubor:



Definice maker

- #define **PB_DDR_INI** 0b11111111
- #define **PB_ODR_INI** 0b11111111
- #define **PB_CR1_INI** 0b11111111
- #define **UART2_PSCR_INI** 0b00000001
- #define **UART2_BRR1_INI** 0b00001101
- #define **UART2_BRR2_INI** 0b00000000
- #define **UART2_CR1_INI** 0b00000000
- #define **UART2_CR2_INI** 0b01101100
- #define **LED0** 0b11111110
- #define **LED1** 0b11111100
- #define **LED2** 0b11111000
- #define **LED3** 0b11110000
- #define **LED4** 0b11100000
- #define **LED5** 0b11000000
- #define **LED6** 0b10000000
- #define **LED7** 0b00000000
- #define **nic** 0b11111111

Funkce

- void **PBledInit** (void)
  Inicializace PB_ODR.

- void **DipInit** (void)
  inicializace PG_CR1

- void **uartInit** (void)
  inicializaceUART2

- void **zpozdeni** (void)
  softwarove zpozdeni pro blikani

- void **receive** (void)

kolik ledek se rozsviti

- void **cekej** (void)
  softwarove zpozdeni pro prijem

- void **nacten** (void)
  vypis nacteneho znaku

- void **novyradek** (void)
  odeslani noveho radku

Detailní popis

Komunikační protokol UART - konstanty.

Autor

Miloš Mlejnek

Datum

25.3. 2023

Verze

1.0

Dokumentace definic maker

#define LED0 0b11111110

svítí 1

#define LED1 0b11111100

svítí 2

#define LED2 0b11111000

svítí 3

#define LED3 0b11110000

svítí 4

#define LED4 0b11100000

svítí 5

#define LED5 0b11000000

svítí 6

#define LED6 0b10000000

svítí 7

#define LED7 0b00000000

svítí 8

#define nic 0b11111111

zhasnuto

#define PB_CR1_INI 0b11111111

nastaví push-pull výstup

#define PB_DDR_INI 0b11111111

nastaví bránu B jako výstupní

#define PB_ODR_INI 0b11111111

zhasnutí celé brány B

#define UART2_BRR1_INI 0b00001101

nastaví množství přenesených bitů za sekundu, 9600 bd

#define UART2_BRR2_INI 0b00000000

nastaví množství přenesených bitů za sekundu, 9600 bd

#define UART2_CR1_INI 0b00000000

spustí komunikaci UART s prvním start bitem, osmi datovými bity a n-tým stop bitem, nastaví metodu probuzení a nečinnost parity bitu

#define UART2_CR2_INI 0b01101100

nastaví generování přerušení na konci přenosu, pokud jsou data po přijetí připravena k přečtení nebo pokud dojde k chybě přeskočení

#define UART2_PSCR_INI 0b00000001

dělení mikroprocesorových hodin jedničkou

Dokumentace funkcí

void cekej (void)

softwarove zpozdeni pro prijem

Tuto funkci volají...

void DipInit (void)

inicializace PG_CR1

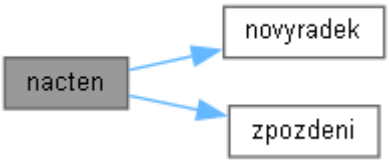
inicializace PG_CR1

void nacten (void)

vypis nacteneho znaku

vypis nacteneho znaku

Tato funkce volá...



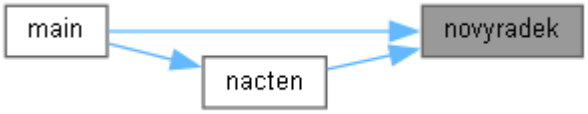
Tuto funkci volají...

void novyradek (void)

odeslani noveho radku

odeslani noveho radku

Tuto funkci volají...



void PBledInit (void)

Inicializace PB_ODR.
Inicializace PB_ODR.
Inicializace PB_ODR.
Inicializace PB_ODR.
Inicializace PB_ODR.

void receive (void)

kolik ledek se rozsviti
kolik ledek se rozsviti
Tuto funkci volají...

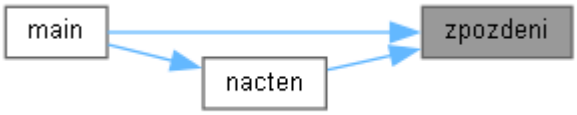
void uartInit (void)

inicializaceUART2
inicializaceUART2
Tuto funkci volají...

void zpozdeni (void)

softwarove zpozdeni pro blikani

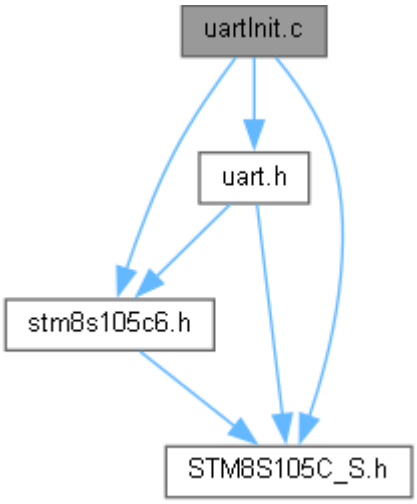
Tuto funkci volají...



A.4.3 uartInit.c

```
#include "uart.h"
#include "stm8s105c6.h"
#include "stm8s105c_s.h"
```

Graf závislostí na vkládaných souborech pro uartInit.c:



Funkce

- void **PBledInit** (void)
  inicializace PB_ODR

- void **uartInit** (void)
  inicializace UART2

- void **zpozdeni** (void)
  softwarove zpozdeni pro blikani

- void **cekej** (void)
  softwarove zpozdeni pro prijem

Dokumentace funkcí

void cekej (void)

softwarove zpozdeni pro prijem

Tuto funkci volají...

void PBledInit (void)

inicializace PB_ODR

Inicializace PB_ODR.

Tuto funkci volají...

void uartInit (void)

inicializace UART2

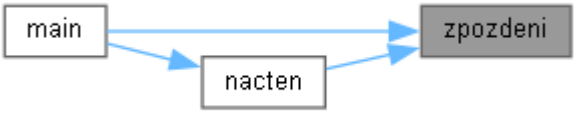
inicializaceUART2

Tuto funkci volají...

void zpozdeni (void)

softwarove zpozdeni pro blikani

Tuto funkci volají...



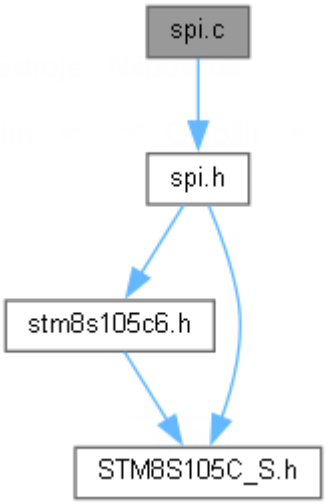
A.5 SPI Komunikace

A.5.1 spi.c

Komunikační protokol SPI.

```
#include "spi.h"
```

Graf závislostí na vkládaných souborech pro spi.c:



Funkce

- **main** ()
  hlavní program

Proměnné

- unsigned char **spi_data**
- int **i**
- int **j**
- int **k**
- unsigned char **prijato**

Detailní popis

Komunikační protokol SPI.

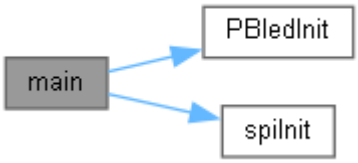
Autor
Milos Mlejnek
Datum
17.3. 2023
Verze
1.0

Dokumentace funkcí
main ()

hlavní program

nejprve proběhne inicializace brány B, spi komunikačního protokolu funkce vysílá pravidelně hodnoty přes registr SPI_DR na RGBW LED(barevné LED) a zároveň se čte hodnota z SPI_DR a posílá na bránu B pro svícení zelných LED rozsvícená zelená LED symbolizuje předchozí hodnotu poslanou do RGBW LED

Tato funkce volá...



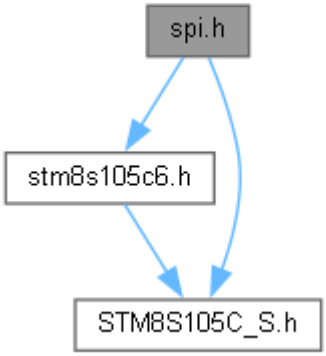
Dokumentace proměnných
int i
pro zpoždění
int j
pro zpoždění
int k
pro zpoždění
unsigned char prijato
přepočet SPI_DR na PB_ODR
unsigned char spi_data
rotace 1

A.5.2 spi.h

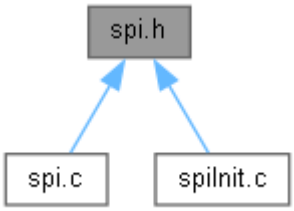
Komunikační protokol SPI - konstanty.

```
#include "stm8s105c6.h"
#include "stm8s105c_s.h"
```

Graf závislostí na vkládaných souborech pro spi.h:



Následující graf ukazuje, které soubory přímo nebo nepřímo vkládají tento soubor:



Definice maker

- #define **PB_DDR_INI** 0b11111111
- #define **PB_ODR_INI** 0b11111111
- #define **PB_CR1_INI** 0b11111111
- #define **SPI_CR1_INI** 0b01111100
- #define **SPI_CR2_INI** 0b00000011
- #define **PE_DDR_SPI** 0b00100000
- #define **PE_CR1_SPI** 0b00100000
- #define **PC_DDR_SPI** 0b00111000
- #define **PC_CR1_SPI** 0b11111000
- #define **SPI_START** 0b00000001

Funkce

- void **PBledInit** (void)
  Inicializace PB_ODR.

- void **spiInit** (void)
  inicializace pro zprovoznění komunikace SPI

- void **res_spi** (void)

Detailní popis

Komunikační protokol SPI - konstanty.

Autor

Milos Mlejnek

Datum

17.3. 2023

Verze

1.0

Dokumentace definic maker

#define PB_CR1_INI 0b11111111

nastaví push-pull výstup

#define PB_DDR_INI 0b11111111

nastaví bránu B jako výstupní

#define PB_ODR_INI 0b11111111

zhasnutí celé brány B

#define PC_CR1_SPI 0b11111000

nastaví push-pull pro SPI MISO, SPI MOSI, SPI RCK, SPI/G a SPI clock

#define PC_DDR_SPI 0b00111000

nastaví výstup pro SPI RCK, SPI/G a SPI clock

#define PE_CR1_SPI 0b00100000

nastaví pátý pin jako push-pull pro SPI CLR

#define PE_DDR_SPI 0b00100000

nastaví pátý pin jako výstupní pro SPI CLR

#define SPI_CR1_INI 0b01111100

nejvíce významný bit bude odeslán jako první, spuštění periferie SPI, určení počtu symbolů přenesených za sekundu, určení polarity hodin a fáze hodin, vybrání master konfigurace

#define SPI_CR2_INI 0b00000011

nastaví jednosměrnou komunikaci, režim přijímání a full duplex komunikaci

#define SPI_START 0b00000001

v proměnné spi_data bude rotovat 1 doleva a toto je její počáteční poloha

---

Dokumentace funkcí

void PBledInit (void)

Inicializace PB_ODR.

Inicializace PB_ODR.

Inicializace PB_ODR.

Inicializace PB_ODR.

Inicializace PB_ODR.

void res_spi (void)

void spiInit (void)

inicializace pro zprovoznění komunikace SPI

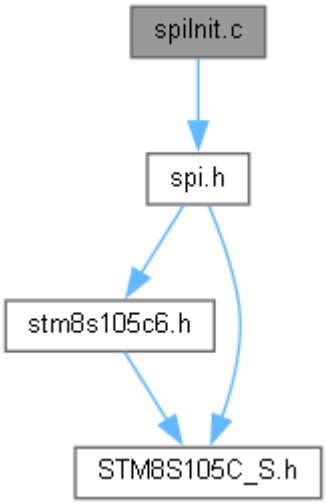
Tuto funkci volají...

A.5.3 spiInit.c

Komunikační protokol SPI - inicializace.

```
#include "spi.h"
```

Graf závislostí na vkládaných souborech pro spiInit.c:



Funkce

- void **PBledInit** (void)
  inicializace PB_ODR

- void **spiInit** (void)
  inicializace pro zprovoznění komunikace SPI

Detailní popis

Komunikační protokol SPI - inicializace.

Autor

Milos Mlejnek

Datum

17.3. 2023

Verze

1.0

Dokumentace funkcí

void PBledInit (void)

inicializace PB_ODR

Inicializace PB_ODR.

void spiInit (void)

inicializace pro zprovoznění komunikace SPI

Tuto funkci volají...

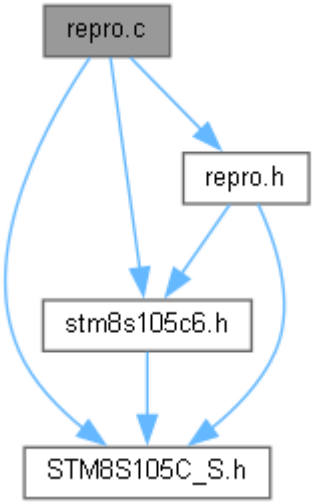
A.6 Reproduktor

A.6.1 repro.c

Reproduktor.

```
#include "stm8s105c6.h"
#include "stm8s105c_s.h"
#include "repro.h"
```

Graf závislostí na vkládaných souborech pro repro.c:



Funkce

- **main** ()
  hlavní program nejprve inicializace funkce kontroluje stisknutí tlačítek BT1-BT4 a při stisknutí nastaví jak často se má vyvolávat přerušení, kde negováním PD_ODR na čtvrtém bitu vyvolám zvuk

- void **int_t2** (void)
  podprogram přerušení

Detailní popis

Reproduktor.

Autor

Miloš Mlejnek

Datum

2.4. 2023

Verze

1.0

Dokumentace funkcí

void int_t2 (void)

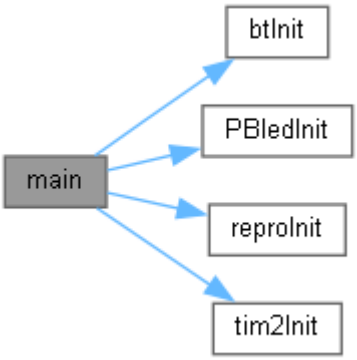
podprogram přerušení

neguje PD_ODR na čtvrtém bitu a tak vznikne zvuk

main ()

hlavní program nejprve inicializace funkce kontroluje stisknutí tlačítek BT1-BT4 a při stisknutí nastaví jak často se má vyvolávat přerušení, kde negováním PD_ODR na čtvrtém bitu vyvolám zvuk

Tato funkce volá...

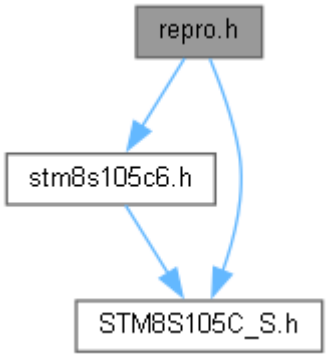


A.6.2 repro.h

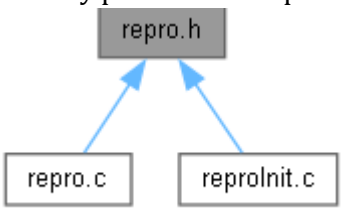
Reproduktor - konstanty.

```
#include "stm8s105c6.h"
#include "stm8s105c_s.h"
```

Graf závislostí na vkládaných souborech pro repro.h:



Následující graf ukazuje, které soubory přímo nebo nepřímo vkládají tento soubor:



Definice maker

- #define **TIM2_ARR_INI** 10
- #define **TIM2_PSCR_INI** 8
- #define **TIM2_CR1_INI** 0b00000001
- #define **TIM2_IER_INI** 0b00000001
- #define **PD_DDR_BEEP** 0b00010000
- #define **PD_CR1_BEEP** 0b00010000
- #define **PB_DDR_INI** 0b11111111
- #define **PB_ODR_INI** 0b11111111
- #define **PB_CR1_INI** 0b11111111

- #define **PA_CR1_BT** 0b01111000

Funkce

- void **tim2Init** (void)
  inicializace timeru 2 pro vyvolávání přerušení

- void **reproInit** (void)
  inicializace brány D pro využití reproduktoru

- void **PBledInit** (void)
  Inicializace PB_ODR.

- void **btInit** (void)
  Inicializace Portu A - tlačítka BT1, BT2, BT3, BT4 na bitech 6-3.

- far interrupt void **int_t2** (void)
  podprogram přerušení

Detailní popis

Reproduktor - konstanty.

Autor

Miloš Mlejnek

Datum

2.4. 2023

Verze

1.0

Dokumentace definic maker

#define PA_CR1_BT 0b01111000

nastaví push-pull vstup pro tlačítka BT1 – BT4

#define PB_CR1_INI 0b11111111

nastaví push-pull výstup

#define PB_DDR_INI 0b11111111

nastaví bránu B jako výstupní

#define PB_ODR_INI 0b11111111

zhasnutí celé brány B

#define PD_CR1_BEEP 0b00010000

čtvrtý pin jako push-pull pro reproduktor

#define PD_DDR_BEEP 0b00010000

čtvrtý pin jako výstupní pro reproduktor

#define TIM2_ARR_INI 10

nastaví hodnotu, do které čítá čítač

#define TIM2_CR1_INI 0b00000001

umožní čítaní čítače

#define TIM2_IER_INI 0b00000001

umožní vyvolání přerušení

#define TIM2_PSCR_INI 8

nastaví hodnotu pro dělení frekvence mikroprocesoru

Dokumentace funkcí

void btInit ()

Inicializace Portu A - tlačítka BT1, BT2, BT3, BT4 na bitech 6-3.

Inicializace Portu A - tlačítka BT1, BT2, BT3, BT4 na bitech 6-3.

far interrupt void int_t2 (void)

podprogram přerušení

neguje PD_ODR na čtvrtém bitu a tak vznikne zvuk

void PBledInit (void)

Inicializace PB_ODR.

Inicializace PB_ODR.

Inicializace PB_ODR.

Inicializace PB_ODR.

Inicializace PB_ODR.

void reproInit (void)

inicializace brány D pro využití reproduktoru

Tuto funkci volají...

void tim2Init (void)

inicializace timeru 2 pro vyvolávání přerušení

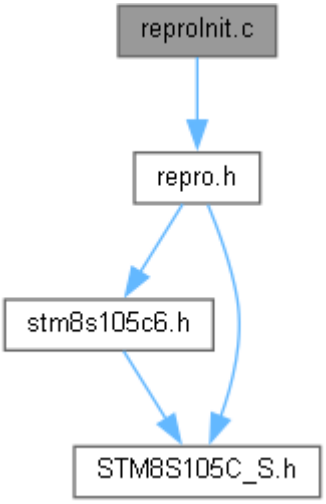
Tuto funkci volají...

A.6.3 reproInit.c

Reproduktor - inicializace.

```
#include "repro.h"
```

Graf závislostí na vkládaných souborech pro reproInit.c:



Funkce

- void **tim2Init** ()
  inicializace timeru 2 pro vyvolávání přerušení

- void **reproInit** ()
  inicializace brány D pro využití reproduktoru

- void **btInit** ()
  inicializace brány A pro využití tlačítek

- void **PBledInit** ()
  inicializace brány B

Detailní popis

Reproduktor - inicializace.

Autor

Miloš Mlejnek

Datum

2.4. 2023

Verze

1.0

Dokumentace funkcí

void btInit (void)

inicializace brány A pro využití tlačítek

Inicializace Portu A - tlačítka BT1, BT2, BT3, BT4 na bitech 6-3.

Tuto funkci volají...

void PBledInit (void)

inicializace brány B
Inicializace PB_ODR.

void reproInit (void)

inicializace brány D pro využití reproduktoru

Tuto funkci volají...

void tim2Init (void)

inicializace timeru 2 pro vyvolávání přerušení

Tuto funkci volají...