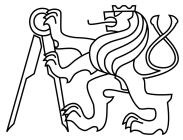
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Programovací nástroj pre mikrokontroléry rodiny ST10F
Student:	Ján Sučan
Vedoucí:	Ing. Radek Dobiáš, Ph.D., MBA
Studijní program:	Informatika
Studijní obor:	Informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce zimního semestru 2017/18

Pokyny pro vypracování

Priemyselný partner rieši možnosť programovania mikrokontrolérov rodiny ST10F aj pod novými verziami operačného systému Windows a rozšírenie funkcionality oproti súčasnému riešeniu od výrobcu mikrokontrolérov.

Vykonajte prieskum existujúcich prostriedkov pre programovanie mikrokontrolérov ST10F168 a ST10F269. Naštudujte dokumentáciu k mikrokontrolérom týkajúcu sa zápisu, čítania a mazania programovej pamäte FLASH. Navrhnite a implementujte software pre vykonávanie týchto operácií s programovou pamäťou mikrokontrolérov. Software bude realizovaný ako spustiteľný súbor a bude funkčný v operačných systémoch MS Windows od verzie Windows 7 až po verziu Windows 10 a v súčasných verziách operačných systémov GNU/Linux. Výsledok práce vhodnými prostriedkami otestuje.

Práca bude vedená formou projektu s tým, že študent bude vykonávať funkciu manažéra, analyzátora, výkonného pracovníka. Vedúci práce vykonáva funkciu sponzora. Zákazníkom je pracovník vývoja vo firme AŽD Praha s.r.o.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 22. září 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalárska práca

Programovací nástroj pre mikrokontroléry rodiny ST10F

Ján Sučan

Vedúci práce: Ing. Radek Dobiáš, Ph.D., MBA

1. februára 2017

Poďakovanie

Ďakujem vedúcemu mojej bakalárskej práce za rady, pripomienky a udávanie smeru. Firme AŽD Praha s.r.o. ďakujem za ochotné poskytovanie vývojového hardwaru, ktorý sa odo mňa vracal nie vždy nepoškodený.

Babičke Štesovej ďakujem za podporu počas môjho štúdia a nespočetné požehnania.

Ďakujem Jozefovi Masárovi, svojmu dlhoročnému spoluhráčovi, za jeho nehynúce kamarátstvo a za tie najlepšie stredoškolské roky.

Mojim drahým rodičom ďakujem za ich nekonečnú podporu, za to, že sa na mňa nehnevali aj vtedy, keď mohli a za to, že mi umožňujú kráčať v živote svojou cestou. Každý by si prial mať na svojej strane takých hrdinov. Mám vás rád.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohoto zákona týmto udeľujem bezvýhradné oprávnenie (licenciu) k užívaniu tejto mojej práce, a to vrátane všetkých počítačových programov ktoré sú jej súčasťou alebo prílohou a tiež všetkej ich dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo užívať. Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela (vrátane komerčného využitia). Toto oprávnenie je časovo, územne a množstevne neobmedzené.

V Prahe 1. februára 2017 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na túto prácu

Sučan, Ján. *Programovací nástroj pre mikrokontroléry rodiny ST10F*. Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Cieľom bakalárskej práce je vytvorenie multiplatformového nástroja umožňujúceho zápis, čítanie a mazanie programovej pamäte mikrokontrolérov rodiny ST10F a jeho otestovanie. Na základe analýzy existujúcich prostriedkov pre programovanie mikrokontrolérov, literatúry k týmto mikrokontrolérom a dokumentácie operačných systémov Microsoft Windows a GNU/Linux je navrhnutá a implementovaná nová aplikácia. Výsledkom práce je multiplatformový programovací nástroj pre zápis, čítanie a mazanie programovej pamäte mikrokontrolérov ST10F168 a ST10F269.

Programovací nástroj je používaný vo firme AŽD Praha s.r.o. pre údržbu elektronických jednotiek koľajových obvodov. Je prospešný technikom a vývojovým pracovníkom, ktorí sa zaoberajú údržbou a vývojom systémov osadených týmito programovateľnými súčiastkami.

V prílohe práce je možné nájsť zdrojové kódy programovacieho nástroja.

Kľúčové slová návrh a implementácia programovacieho nástroja, mikrokontroléry, ST10F, FLASH pamäť, Windows, Linux, FreeBSD, AŽD Praha s.r.o, Assembler, C++

Abstract

Aim of this bachelor thesis is a creation of cross-platform tool allowing to write, read and erase a program memory of microcontrollers of ST10F family and its testing. Based on analysis of existing means for microcontroller programming, literature about these microcontrollers and documentation of Microsoft Windows and GNU/Linux operating systems, a new application is designed and implemented. Result of the thesis is a cross-platform programming tool for writing, reading and erasing of a program memory of ST10F168 and ST10F269 microcontrollers.

The programming tool is used by AŽD Praha s.r.o. company for maintenance of electronic units of railway electric circuits. It is beneficial to technicians and developers, who deal with maintenance and development of a systems which have these programmable devices planted.

In attachment of the thesis, source codes of the programming tool can be found.

Keywords desing and implementation of a programming tool, microcontrollers, ST10F, FLASH memory, Windows, Linux, FreeBSD, AŽD Praha s.r.o, Assembler, C++

Obsah

Zoznam obrázkov

Úvod

Programovateľné elektronické súčiastky našli využitie v aplikáciach pre spracovávanie dát a riadenie už pred pár desiatkami rokov a tento trend trvá dodnes. Aj v dnešnej dobe sú v prevádzke systémy, ktoré sú osadené veľmi starými programovateľnými súčiastkami, plnia kritické úlohy a ktorých vytvorenie a nasadenie s použitím nových programovateľných súčiastok by bolo veľmi náročné a nákladné. Programovacie nástroje pre prácu s týmito starými súčiastkami však stratili krok s hlavným technologickým prúdom osobných počítačov. Preto je pri údržbe systémov veľmi ťažké pracovať s týmito súčiastkami s pomocou moderných osobných počítačov.

Kapitola 1

Cieľ práce

Cieľom teoretickej časti práce je preskúmať existujúce prostriedky pre programovanie mikrokontrolérov ST10F168 a ST10F269, naštudovať dokumentáciu k týmto mikrokontrolérom a k operačným systémom MS Windows a GNU-/Linux. Po dohode so zadávateľom budú upresnené požiadavky na výsledný programovací nástroj.

Praktická časť práce sa bude zaoberať návrhom a realizáciou software, ktorý umožní vykonávať operácie mazania, zápisu a čítania internej programovej pamäte FLASH mikrokontrolérov ST10F168 a ST10F269. Výsledok práce bude pod podporovanými operačnými systémami otestovaný na splnenie cieľových vlastností.

1.1 Motivácia

Pretože pre množstvo starších programovateľných súčiastok nie sú k dispozícii programovacie nástroje, ktoré by boli bez problémov funkčné s technológiami moderných operačných systémov, rozhodol som sa vytvoriť programovací nástroj, ktorý bude slúžiť pre prácu s týmito súčiastkami a bude bez problémov funkčný na nových osobných počítačoch.

Výsledok práce bude prospešný technikom a vývojovým pracovníkom, ktorí sa zaoberajú údržbou systémov osadených staršími programovateľnými súčiastkami používaných v kritických aplikáciach a pre ktoré neexistujú pohodlne použiteľné programovacie nástroje.

Zadávateľ, firma AŽD Praha s.r.o., bude programovací nástroj používať pre údržbu elektronických jednotiek koľajových obvodov, ktoré sa ešte v súčasnosti používajú v prevádzke.

1.2 Štruktúra práce

V práci sa zaoberám analýzou existujúcich prostriedkov pre prácu s programovateľnými súčiastkami rodiny ST10F od firmy STMicroelectronics, návrhom aplikácie pre zápis, čítanie a mazanie programovej pamäte súčiastky a otestovaním výslednej aplikácie na splnenie týchto vlastností.

V prvej časti sa venujem analýze existujúcich prostriedkov pre programovanie súčiastok z rodiny ST10F, analýze elektroniky, na ktorej bude prebiehať vývoj aplikácie, rozborom možností, ktoré tieto súčiastky ponúkajú programátorovi a dokumentáciou cieľových operačných systémov pre prevádzku výslednej aplikácie. V druhej časti sa zaoberám návrhom základu aplikácie a použitými vývojovými nástrojmi a programovacími jazykmi. V tretej časti popisujem realizáciu aplikácie, zmenu návrhových rozhodnutí, ktoré vyvstali až v priebehu práce a riešenie problémov, na ktoré som pri vývoji narazil. Štvrtá časť popisuje testovanie aplikácie na splnenie požadovaných vlastností na podporovaných operačných systémoch.

Kapitola 2

Analýza

2.1 Upresnenie zadania po dohode so zadávateľom

Po dohode so zadávateľom témy bakalárskej práce boli upresnené požiadavky na výsledný programovací nástroj. Požiadavky sa delia na funkčné a nefunkčné.

Upresňujúce funkčné požiadavky:

- nástroj bude pracovať s internou FLASH pamäťou programu ST10F168 aj ST10F269,
- dáta sa z a do FLASH budú čítať a zapisovať od adresy nula,
- jeho používateľské rozhranie bude riadkové,
- bude multiplatformný na úrovni zdrojového kódu.

Upresňujúce nefunkčné požiadavky:

- nástroj nebude zaisťovať prepínanie MCU do režimu bootstrap zavádzača,
- bude predpokladať, že na sériovom porte je pripojený MCU s už aktivovaným režimom bootstrap zavádzača,
- nebude pracovať so zámkami a ochranami internej FLASH pamäte programu ani predpokladať, že sú na pamäti aktivované.

2.2 Existujúce prostriedky pre programovanie ST10F

Programovacie prostriedky v súčastnosti dostupné pre ST10F rozdelím vzhľadom na ich technickú realizáciu podľa toho, či sa jedná len o software pre PC, alebo o hardware s obslužným software. Doplnkovú elektroniku MCU (napájanie, reset, taktovanie a pod.), zabezpečujúcu nutné minimum pre programovanie neuvažujem ako súčasť hardwarových programovacích prostriedokov,

pretože je vyžadovaná aj pre bežnú prevádzkovú činnosť. Za hardwarové programovacie prostriedky považujem tie, ktoré sú viazané na špeciálny obslužný software a bez neho nie sú použiteľné ako funkčný celok pre programovanie MCU.

Počiatočným bodom z ktorého som pri prieskume vychádzal boli zoznamy *FLASH Utilities* a *Device Programmers* na stránkach [1] a [2]. Firmy [3], [4] a aj [5] už produkty pre programovanie ST10F168 a ST10F269 vyradili z ponuky. Odkazy vedúce na webové stránky spomínaných firiem buď nie sú funkčné, alebo sú presmerované na iné stránky firiem s nerelevantnými informáciami. Tiež použitie vyhľadávania priamo na webových stránkach firiem [3], [4] a [5] neviedlo na žiadne programovacie nástroje pre MCU ST10F168 ani ST10F269.

2.2.1 ST10Flasher

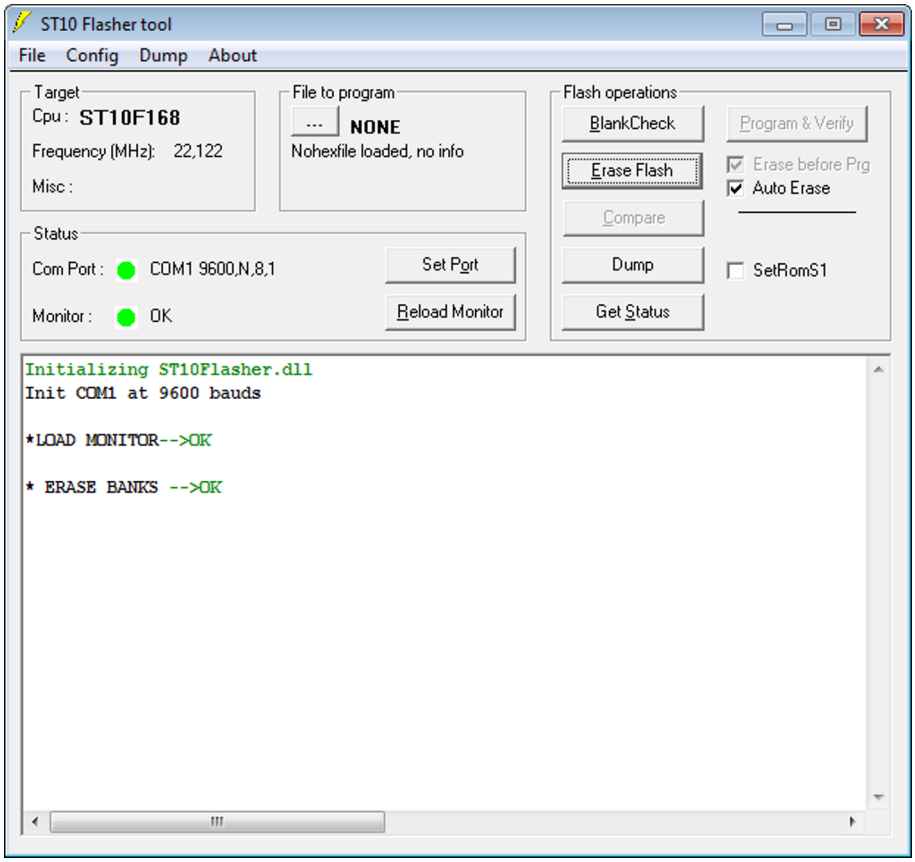
ST10Flasher bol vytvorený firmou [5], výrobcom rodiny MCU ST10F, a uvoľnený k voľnému používaniu. Zdrojové kódy ST10Flasher som nenašiel. Nebol však predstavený ako produkčný nástroj pre programovanie. To sa uvádza v informáciach o programe prístupných v grafickom rozhraní aplikácie ST10Flasher, znázornenom na obrázku 2.1. Podľa [6] podporuje len operačné systémy Windows 95, Windows 98 a NT. Z rodiny ST10F podporuje MCU ST10F167, ST10F168, ST10F169, ST10F269 a ST10F280.

Nástroj už nie je dostupný na oficiálnych webových stránkach firmy [5]. Nenašiel som ani webové stránky iných firiem, kde by sa dal získať. Najčastejšie odkazy na stiahnutie som nachádzal na fórach v príspevkoch jednotlivcov. Odkazy viedli na úložiská súborov a neboli funkčné (napr. [7]). Kópiu programu som získal od pracovníka vo firme AŽD Praha s.r.o.

Program pri spustení číta konfiguračný súbor STARTUP.INI v inštalačom adresári. V tomto súbore je možné nastaviť názov používaného komunikačného portu. V grafickom používateľskom rozhraní je možné vybrať len COM1, COM2, COM3 a COM4. Cez konfiguračný súbor som skúšal nastavovať aj vyššie čísla COM portov, ktoré mi operačný systém dovolil použiť. Od COM5 po COM9 je program funkčný. Pri nastavení COM10 už hlási chybu pri otváraní sériového portu. GUI neposkytuje používateľovi informáciu o priebehu operácií, len o výsledku.

Funkčne sa nástroj delí na

- grafické používateľské rozhranie,
- dynamicky linkovateľnú knižnicu implementujúcu funkcionalitu pre programovanie,
- a súbory firmware vo formáte hex pre podporované MCU.



Obr. 2.1: ST10Flasher

2.2.2 Ostatné softwarové nástroje

Medzi ďalšie softwarové nástroje patria práve tie, založené na dynamicky linkovateľnej knižnici ST10Flasher ale s vlastným používateľským rozhraním. Jedná sa o snahy jednotlivcov k vytvoreniu nástrojov odvodených od ST10Flasher (napr. [8]).

Firma AŽD Praha s.r.o. má pre svoje účely vyvinuté vlastný programovací nástroj, založený na dynamicky linkovateľnej knižnici ST10Flasher. Nemožnosť použitia ľubovoľných čísel COM portov a závislosť na platforme MS Windows tak stále zostáva.

2.2.3 Hardwarové programovacie prostriedky

Hardwarové programovacie prostriedky poskytujú firmy [9] a [10]. Riešenia poskytované týmito firmami sú v cenách stoviek eur.

Riešenie [11] je určené pre použitie s hardwarom Martex Box III a je cielené pre programovanie MCU ST10F použitých v elektronike automobilov.

Software aj hardware je uzavretý. Na uvedených webových stránkach nie je možné dohľadať, aké verzie MS Windows nástroj podporuje a ani či podporuje operačné systémy GNU/Linux.

Firma [10] ponúka univerzálne programátory. [12] podporuje približne 97 000 druhov programovateľných súčiastok. V uvedenom zdroji je k dispozícii archív s ovládacím software na stiahnutie. Software aj hardware je uzavretý. V informáciach k programátoru je uvedené, že jeho obslužný software podporuje operačné systémy MS Windows od verzie Windows 2000 po Windows 8. Informácie o podpore GNU/Linuxu som nenašiel.

2.3 Programátorský model ST10F

Informácie v tejto podkapitole sú prevzaté z [13] a [14]. Jedná sa o informácie nutné k návrhu firmware pre podporu mazania, zápisu a čítania programovej pamäte FLASH.

2.3.1 Adresový priestor

MCU ST10F používajú 24-bitovú fyzickú adresu a sú tak schopné pracovať s adresovým priestorom až do veľkosti 16 MiB. Tento adresový priestor je rozdelený na 256 segmentov o veľkosti 64 KiB. Každý segment sa delí na 4 stránky, každá s veľkosťou 16 KiB. Do adresového priestoru je mapovaná pamäť programu, pamäť dát, registre a V/V porty.

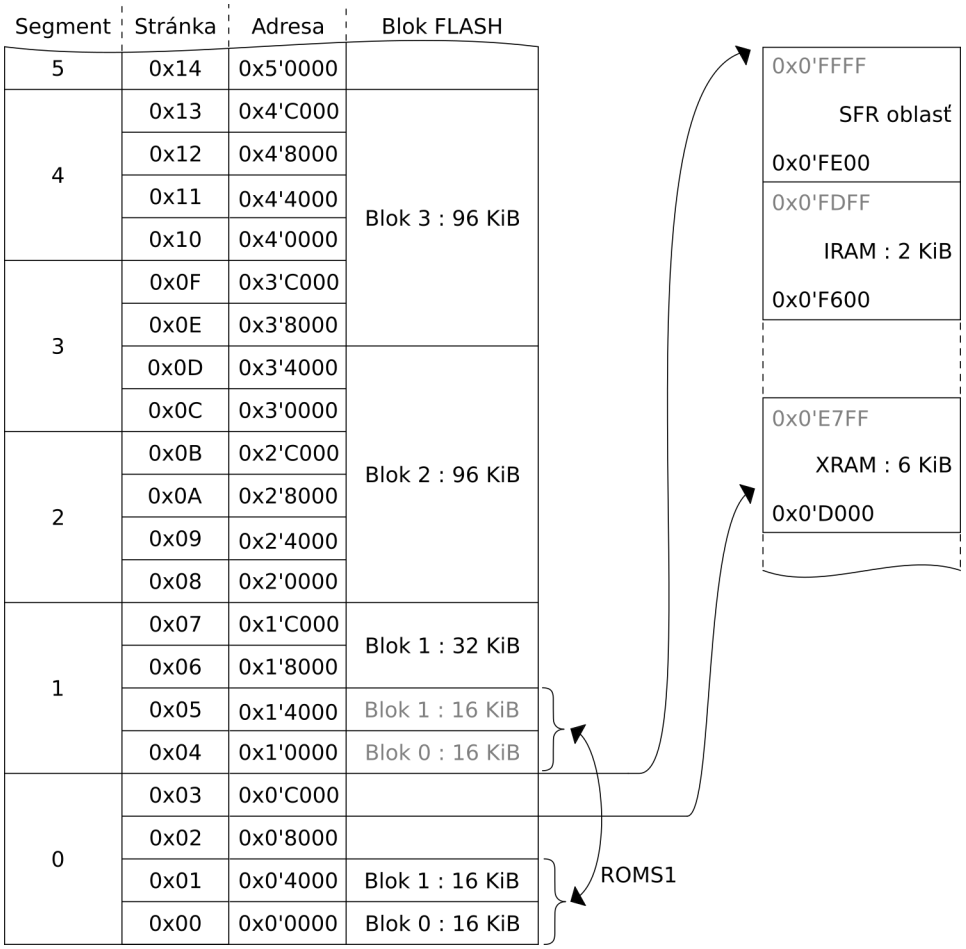
Väčšina oddelených adresových priestorov umiestnených v puzdre MCU je namapovaných do segmentu 0. Sú to napr. interná RAM, SFR, GPR a periférne zariadenia. Segment 0 sa nazýva aj systémový segment.

Horných 224 KiB FLASH pamäte programu je vždy mapovaných do segmentov 1, 2, 3 a 4. Mapovanie spodných 32 KiB FLASH pamäte programu sa dá meniť nastavením bitu ROMS1 v registri SYSCON. Pre ROMS1 = 0 je spodných 32 KiB mapovaných do segmentu 0 od adresy 0 a pre ROMS1 = 1 do segmentu 1.

FLASH pamäť má zavedené vlastné delenie na bloky. Vďaka tomuto deleniu je možné namiesto celej pamäte pred zápisom dát zmazať len jednotlivé zapisované bloky. Delenie sa medzi ST10F168 a ST10F269 odlišuje počtom a veľkosťou blokov.

Pamäťový priestor ST10F168 je znázornený na obrázku 2.2. Podoba je zjednodušená a zahrňuje len spomínané aspekty adresového priestoru. Ukazuje segmenty s namapovanými blokmi FLASH.

Pamäťový priestor ST10F269 v uvedenom rozsahu sa líši rozdielnou veľkosťou blokov FLASH a rozdelením XRAM na dve oblasti (XRAM1 a XRAM2), namiesto jednej, a jej veľkosťou. Pri využívaní adries len v rozsahu XRAM1 je použitie XRAM na oboch MCU z pohľadu programátora rovnaké. V ďalšom texte budem preto považovať pojmy *XRAM* a *XRAM1* za rovnaké.



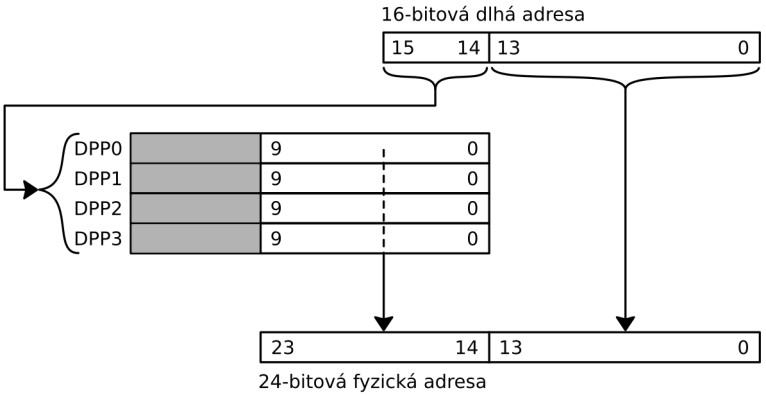
Obr. 2.2: Adresový priestor ST10f168

2.3.2 Spôsoby adresovania

Spôsob adresovania určuje, ako inštrukcia pristupuje k dátam, s ktorými pracuje. Spôsoby adresovania používané MCU ST10F sú:

- **Krátke adresovanie** - pri výpočte fyzickej adresy sa použije špecifická báza, ku ktorej je pripočítaná hodnota spočítaná z krátkej adresy obsiahnutej v inštrukčnom kóde. Krátke adresovanie sa používa pri adresovaní bitovo adresovateľných oblastí pamäte alebo pri adresovaní registrov.

- **Dlhé adresovanie** - 16-bitová dlhá adresa obsiahnutá v inštrukčnom kóde je pre účely výpočtu 24-bitovej fyzickej adresy rozdelená na dve časti. 14-bitová časť (bity 0 až 13) je offset. 2-bitová časť (bity 14 až 15) je číslo od 0 do 3 a je to adresa jedného zo štyroch DPP registrov (DPP0 až DPP3). DPP registre sú 16-bitové, ale používa sa len prvých 10 bitov



Obr. 2.3: Spôsob dlhého adresovania

(bity 0 až 9). Hodnota DPP registru vynásobená veľkosťou stránky pamäte (16 KiB) tvorí bázu, ku ktorej sa pripočíta 14-bitový offset v dlhej adrese. Tak je získaná 24-bitová fyzická adresa. DPP registre sú pri resete MCU nastavené tak, že každá dlhá adresa je mapovaná na rovnakú fyzickú adresu v segmente 0. Dlhé adresovanie je znázornené na obrázku 2.3.

- **Nepriame adresovanie** - kombinuje spôsoby krátkeho a dlhého adresovania. 16-bitová dlhá adresa je uložená v niektorom z GPR. V inštrukčnom kóde je uložená 4-bitová adresa, ktorá je použitá podľa spôsobu krátkeho adresovania pre určenie adresy GPR obsahujúceho 16-bitovú dlhú adresu, ktorá je potom použitá podľa spôsobu dlhého adresovania vo výpočte fyzickej adresy.

- **Konštanty** - dáta s ktorými inštrukcia pracuje, sú uložené priamo v jej inštrukčnom kóde. MCU ST10F z dôvodu úspory pamäte pre uloženie inštrukčných kódov pracujú s veľkosťami konštánt 3, 4, 8 alebo 16 bitov.

- **Adresovanie cieľov vetvenia programu** - ciele volaní a skokov môžu byť adresované relatívne, absolútne a nepriamo. Adresu v pamäti programu pre vykonávanie ďalšej inštrukcie určuje obsah registrov IP a CSP. CSP je ukazovateľ na segment pamäťového priestoru a IP je offset v rámci segmentu. Relatívne adresovanie mení IP pripočítaním 16-bitovej hodnoty so znamienkom. Pri absolútnom adresovaní je IP naplnený novou hodnotou. Pri relatívnom adresovaní inštrukcia obsahuje adresu GPR v ktorom je uložená nová hodnota IP. Hodnota registra CSP sa dá meniť len jeho naplnením novou hodnotou. Čítanie inštrukcie z nepárnej adresy má za následok vyvolanie nemaskovateľného prerušenia.

2.3.3 Prístup ku GPR

Pri krátkom spôsobe adresovania registrov pre všeobecné použitie určuje bázu hodnota v registri CP. Slová v 32 bajtoch od báze (vrátane) sú používané ako GPR. Vďaka tomu je možné pri volaní podprogramov rýchlo zachovávať hodnoty registrov. Namiesto ich ukladania na zásobník stačí zmeniť hodnotu CP. Kvôli prúdovému spracovaniu inštrukcií sa ale zmena prejaví až na druhej nasledujúcej inštrukcii, preto sa často za inštrukciu zmeny CP pridáva prázdna inštrukcia NOP. Inštrukcia NOP nie je nutná vtedy, ak za inštrukciou zmeny CP nenasleduje inštrukcia, využívajúca GPR.

2.3.4 Mechanizmus prekrytia DPP registrov

Mechanizmom prekrytia DPP registrov sa dá dočasne vyradiť použitie hodnôt DPP registrov pri dlhom adresovaní. Tento mechanizmus je prístupný inštrukciami EXTP, EXTPR, EXTS a EXTSR.

Tieto inštrukcie prijímajú dva operandy. Prvým je buď priama hodnota použijúca sa pri výpočte fyzickej adresy namiesto hodnoty registra DPP (inštrukcie bez prípony R), alebo register obsahujúci túto hodnotu (inštrukcie s príponou R). Druhým operandom je 2-bitové číslo ktoré určuje, pre koľko nasledujúcich inštrukcií bude prekrytie platné. Prekryť DPP je možné maximálne pre štyri nasledujúce inštrukcie.

V inštrukciách EXTP a EXTPR znamená konečná hodnota prvého operandu adresu stránky. Tá je použitá rovnako ako je popísané v dlhom spôsobe adresovania.

V inštrukciách EXTS a EXTSR znamená konečná hodnota prvého operandu adresu segmentu. Z dlhej 16-bitovej adresy sa pri výpočte 24-bitovej fyzickej adresy použije všetkých 16 bitov ako offset v segmente. Zvyšných 8 bitov fyzickej adresy je určených adresou segmentu. Adresa segmentu z operandu sa teda vynásobí veľkosťou segmentu 64 KiB a pripočíta sa k nej 16-bitový offset z dlhej adresy.

2.3.5 Zásobník

Zásobníkové operácie ovplyvňujú registre SP, STKUN, STKOV a hodnota STKSZ (bity 13 až 15) v registri SYSCON. MCU ST10F umožňujú používať zásobník v dvoch módoch: kruhový a lineárny. Mód a zároveň veľkosť zásobníka je volený podľa tabuľkovej hodnoty STKZ z katalógového listu MCU. Zásobník sa môže nachádzať len v internej RAM a rastie smerom k nižším adresám.

Lineárny mód slúži pre veľkosti zásobníka viac ako 1 KiB. Všetky ostatné nastavenia STKSZ (okrem nepoužitých) znamenajú voľbu kruhového zásobníka o vybranej veľkosti.

Registre STKUN a STKOV slúžia na ochranu pre podtečením a pretečením zásobníka. STKUN je najvyššia platná adresa pre zásobník a STKOV

je najnižšia platná adresa pre zásobník. Ak dôjde k prekročeniu týchto medzí, je vyvolané prerušenie. Prerušenia vyvolané podtečením alebo pretečením zásobníka sú nemaskovateľné.

Podľa zvolenej veľkosti zásobníka sa líši počet významných bitov SP. V kruhovom móde zásobníka dochádza k hardwarovému výpočtu adresy pre uloženie dátového slova z významných bitov SP. K nim je do šírky 16 bitov doplnená opačná časť bitov z registru STKUN. Adresa v SP sa tak môže líšiť od skutočnej hodnoty použitej pre uloženie dátového slova.

Ak programátor nepočíta s pretečením zásobníka, adresa v SP pri korektnom nastavení STKUN a STKOV, obsahuje reálnu adresu pre uloženie dátového slova.

2.3.6 Režim bootstrap zavádzača

Režim bootstrap zavádzača ponúka možnosť, ako pri hardwarovom resete MCU nahrať programový kód do RAM MCU cez sériové rozhranie a predať mu riadenie, namiesto spustenia programového kódu od adresy 0 v segmente 0 pamäte programu.

Ak je pri hardwarovom resete úroveň na pine P0L.4 (BSL) log. 0, spustí sa programový kód z Boot-ROM MCU. Tento kód inicializuje MCU aj sériové rozhranie ASC0 v móde: 8 dátových bitov, 1 stop bit, bez parity. MCU očakáva na sériovom rozhraní príjem bajtu s hodnotou 0x00. Pevná komunikačná rýchlosť nie je nastavená. Namiesto toho, MCU analyzuje priebeh signálu na sériovej linke pri vysielaní nulového bajtu protistranou a podľa toho nastaví rýchlosť komunikácie. MCU pri meraní signálu na sériovej linke používa časovač a ďalej jeho hodnotu používa pre výpočet hodnoty pre nastavenie rýchlosti sériovej linky. V týchto krokoch môže dôjsť k odchýlke reálnej od výpočtom určenej rýchlosti. Rýchlosť nastavená na strane komunikujúcej protistrany a na strane MCU sa nesmú líšiť o viac ako 2,5 %.

Po tom MCU odošle protistrane potvrdzovací bajt 0xD5, prijme 32 bajtov, ktoré nahrá do internej RAM od adresy 0xFA40 a predá na adresu 0xFA40 riadenie.

2.3.7 Zápis, čítanie a mazanie programovej FLASH pamäte

ST10F168 používa iné programované rozhranie k operáciam s FLASH pamäťou ako ST10F269. Oba podporujú mazanie blokov vstavanej programovej FLASH, čítanie slova z pamäte, zápis slova do pamäte. ST10F269 naviac podporuje zmazanie celej pamäte jedným príkazom.

2.3.7.1 ST10F168

ST10F168 má vo vstavanej pamäti, neprístupnej programátorovi, uložené podprogramy implementujúce operácie s FLASH pamäťou. Tieto algoritmy sú označované skratkou STEAK. Informácie o STEAK som čerpal z [15].

Programátor môže vyvolať konkrétnu operáciu naplnením registrov R0 až R4 argumentami operácie a vykonaním odomykacej sekvencie inštrukcií.

Odomykacia sekvencia pozostáva z dvoch tesne po sebe nasledujúcich zápisov do adresového priestoru. Prvý zápis používa priame adresovanie v inštrukcii vo formáte `MOV mem, Rwn`, kde `mem` je adresa a `Rwn` je jeden z registrov R6 až R15. Druhý zápis používa nepriame adresovanie inštrukciou vo formáte `MOV [Rwm], Rwn`, kde `Rwm` je jeden z registrov R6 až R15 naplnených hodnotou `mem` z predchádzajúcej inštrukcie a `Rwn` je jeden z registrov R6 až R15.

Po odomykacej sekvencii, prejde spracovávanie programového kódu na STEAK podprogramy. STEAK skontroluje argumenty a podmienky pre programovaciu operáciu a ak sú korektné, vykoná ju. Po dokončení STEAK podprogramov sa riadenie vráti za poslednú inštrukciu odomykacej sekvencie. Návratový kód značiaci úspech alebo popisujúci chybu je v registri R0.

STEAK používa pre svoje operácie zásobník a preto je potrebné vyhradiť na ňom dostatok miesta. V katalógovom liste výrobca doporučuje minimálne 50 slov.

Pri všetkých operáciach vyžaduje STEAK v registri R4 dĺžku periódy hodinového signálu procesoru v nanosekundách. Túto hodnotu používa pre časovanie operácií s FLASH pamäťou. Podľa dokumentácie k STEAK je povolený rozsah 30 až 200. Experimentálne som ale zistil, že maximálna hodnota je 199. Frekvencia, popisovaná touto hodnotou, by sa podľa výrobcu nemala veľmi líšiť od skutočnej frekvencie procesora. Výrobca však presnú odchýlku neuvádza. Píše, že ak je rozdiel veľmi veľký (približne 1,5 násobok a viac), výsledok programovacej operácií nie je zaručený.

ST10Flasher používa automatickú detekciu frekvencie MCU. Rozhodol som sa však pre manuálne zadanie frekvencie používateľom, pretože detekcia je možná jedine cez časovanie komunikácie na sériovom rozhraní a sériová komunikácia v OS je zaisťovaná viacerými vrstvami a časové odchýlky na sériovej linke sa nedajú všeobecne presne určiť. Pri použití programovacieho nástroja pre kritické aplikácie nie je možné povolovať žiadne nedefinovateľné odchýlky.

2.3.7.2 ST10F269

V ST10F269 sa konkrétna operácia volí sekvenciou zápisov na špecifické adresy v adresovom priestore. Adresy musia ležať v aktívnej oblasti FLASH, tzn. na logických adresách, na ktoré sú mapované adresy FLASH pamäte. Je možné použiť len nepriame adresovanie cez register. Východzí mód pamäte je čítanie.

Jeden zápis sa nazýva cyklus. Rôzne operácie majú rôzny počet cyklov (1 až 12). V priebehu posledného cyklu kontrolér FLASH spustí operáciu a vykonávanie programu pokračuje ďalej. V prípade operácie pre mazanie blokov sa operácia spustí až po uplynutí 96 mikrosekund po 6. cykle. V tomto čase sa očakávajú ďalšie cykly špecifikujúce ďalšie bloky pre zmazanie. Operácie s FLASH pamäťou prebiehajú paralelne s vykonaním programu.

Informácia o priebehu a výsledku operácie je vo *FLASH status* registri. Spôsob získania jeho hodnoty sa líši pre operácie zápisu a mazania. V prípade operácie zápisu je adresa stavového registra adresou zapisovaného slova. Pri operácii zápisu je jeho adresou ľubovoľná adresa v bloku, ktorý je mazaný.

Bit FSB.3 informuje o uplynutí 96 mikrosekúnd v príkaze mazania blokov. Bit je nastavený na 0 po poslednom príkaze v sérii a nastavený na 1, po uplynutí času.

Pre detekciu skončenia operácie je možné použiť dva spôsoby:

- **Toggle bity** - ak operácia prebieha, každé čítanie vráti opačnú logickú hodnotu. Operácia je skončená, ak dve po sebe nasledujúce čítania vrátia rovnakú hodnotu bitu; 1 pre operáciu mazania a naposledy programovaný bit pre operáciu zápisu.

- **Data polling** - pri zápise má bit FSB.7 hodnotu doplnku 7. bitu zapisovaného slova. Po skončení zápisu má hodnotu 7. bitu zapisovaného slova. V priebehu mazania má hodnotu 0 a po skončení hodnotu 1.

FSB.5 indikuje chybu. Ak je nastavený na 1 v priebehu operácie, nastala chyba. Po dokončení operácie v prípade úspechu, má hodnotu 1 po mazaní a hodnotu naposledy programovaného bitu pri zápise. Je nulovaný príkazom Read/Reset. Z toho vyplýva, že hodnotu bitu FSB.5 je pre správnu interpretáciu nutné kontrolovať zároveň s príznakom dokončenia operácie.

2.3.8 Identifikácia mikrokontroléra

ST10F168 aj ST10F269 majú rovnaký protokol bootstrap zavádzača: prijatie bajtu 0x00, odoslanie bajtu 0xD5 a prijatie 32 bajtov. Pretože sa ale spôsoby mazania, zápisu a čítania programovej pamäte líšia, potrebuje protistrana vedieť s akým modelom MCU komunikuje. Oba MCU obsahujú registre IDMANUF a IDCHIP. Hodnoty registrov jednoznačne identifikujú výrobcu a model MCU.

2.4 Vývojový hardware s ST10F168 a ST10F269

Vývojový hardware som získal od firmy AŽD Praha s.r.o. K dispozícii som mal dve verzie jednotiek AVR3, ktoré sa používajú v súboroch koľajových prijímačov elektronických koľajových obvodov AŽD typ KOA1. Koľajové obvody typu KOA1 sú popísané na [16].

Jedna verzia bola osadená MCU ST10F269. Je ukázaná na obrázku 2.4. Druhá bola osadená MCU ST10F168. Tieto dve verzie sú funkčne kompatibilné, na prvý pohľad nerozoznateľné a pri ich analýze som vychádzal z jednej spoločnej schémy. Tú mi na vyžiadanie poskytla firma AŽD Praha s.r.o.

Napájanie MCU na doske plošných spojov a signály pre jeho programovanie sú vyvedené na konektor na zadnej strane dosky. Konektor má 32 kontaktov, ale pre účely programovania je využitých len šesť, v prípade verzie s ST10F168, a päť, v prípade verzie s ST10F269. Z uvedených piatich kontaktov sú dva na pripojenie napájacieho napätia 24 V, dva na pripojenie kontaktov sériovej linky (vysielanie a príjem) a jeden na signál BSL pre uvedenie MCU do režimu bootstrap zavádzača.

Kontakty sériovej linky sú z konektoru priamo privedené na kontakty MCU pre jeho sériové rozhranie ASC0. Kontakt BSL je pripojený k obvodu, ktorý zaisťuje inverziu BSL signálu a ošetrenie rušenia. Pri nepripojenom kontakte konektoru, alebo jeho pripojení na zem, je báza NPN tranzistoru pripojená na zem, tranzistor je uzavretý a na vývode P0L.4 (BSL) je vďaka vnútorným V/V obvodom MCU v dobe hardwarového resetu vzorkovaná úroveň 1. Režim bootstrap zavádzača sa neaktivuje. Ak je ale na kontakt konektoru privedené napätie 5 V, tranzistor sa otvorí a na P0L.4 sa dostane 0 V. V dobe hardwarového resetu je vtedy vzorkovaná úroveň log. 0 a aktivuje sa režim bootstrap zavádzača. Kondenzátor pripojený medzi zem a bázu má za úlohu zviesť rušivé frekvencie k zemi, aby rušením nedošlo v dobe hardwarového resetu k aktivácii režimu bootstrap zavádzača.

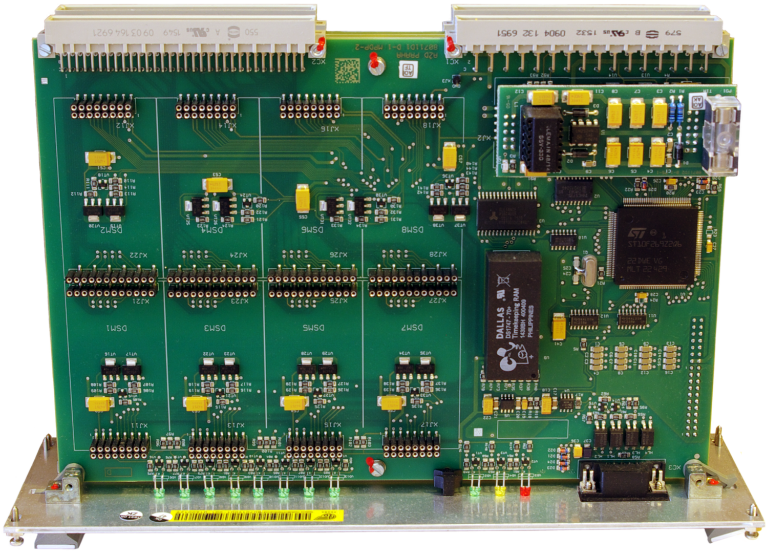
Všetkých päť kontaktov verzie s ST10F269 je presne rovnako použitých aj pre verziu s ST10F168. Tá má naviac ešte na jeden vývod konektoru pripojené napätie 12 V využívané internou FLASH pamäťou pre jej programovanie. Vývod konektoru je priamo pripojený k vývodu Vpp (RPD) MCU. K RPD je pripojený obvod tvorený rezistorom a kondenzátorom. Kondenzátor zaisťuje pomalý nábeh napätia z 0 V na 5 V. V dobre resetu je nutné, aby bola po určitú dobu na RPD log. 0. Rezistor tiež obmedzuje prúd, tečúci zo zdroja 12 V do zdroja 5 V.

Na doske je osadený kryštál 11,0592 MHz a vývody P0H.7, P0H.6, P0H.5 MCU sú nastavené na logické hodnoty 1, 0 a 1. Toto troj-bitové číslo má význam vynásobenia frekvencie generovanej kryštálom hodnotou 2. Taktovacia frekvencia procesoru MCU je teda 22,1184 MHz.

2.5 Ovládanie sériového portu

Sériová komunikácia sa medzi operačnými MS Windows a GNU/Linux značne odlišuje. V OS systémoch Windows sú funkcie pre ovládanie sériového portu súčasťou Win32 API. Situácia je uľahčená spätnou kompatibilitou. Preto jedna implementácia nad Win32 API bude funkčná v OS od MS Windows 7 po MS Windows 10. Informácie o ovládaní sériového portu cez Win32 API som čerpal z [17] a [18].

GNU/Linux síce nie je oficiálne certifikovaný POSIX systém, ale funkcie a dátové štruktúry z normy POSIX sú implementované. GNU/Linux poskytuje aj vlastné rozšírenie POSIX API pre sériovú komunikáciu, ktoré napr.



Obr. 2.4: Doska osadená mikrokontrolérom ST10F269

umožňuje nastavovať ľubovoľnú rýchlosť sériovej komunikácie. Rozhodol som sa však používať len POSIX API, aby som zbytočne neobmedzoval prenositelnosť software na systémy, ktoré sériovú komunikáciu podľa POSIX implementujú. Informácie o ovládaní sériového portu na unixových systémoch som čerpal z [19] a [20].

2.5.1 Win32 API

Win32 API poskytuje pre sériovú komunikáciu dva spôsoby V/V:

- **asynchrónny (prekrývaný)** - dovoľuje po zavolaní V/V operácie pokračovať vo vykonávaní vlákna. Požiadavky pre V/V z viacerých vláken sú serializované operačným systémom. Spracovávanie návratov volaní je komplikovanejšie, pretože volanie operácie sa môže tiež správať aj ako neprekrývaná a aplikácia musí byť pripravená aj na okamžité ošetrenie výsledku operácie, okrem odloženého ošetrenia.

- **synchrónny (neprekrývaný)** - blokuje vlákno s volaním V/V operácie až do dokončenia operácie. Ak jedno vlákno volá V/V operáciu, ďalšie vlákna môžu tiež zavolať V/V operáciu, ale budú blokované, až pokým neskončí pôvodná operácia. Serializácia prístupu k portu je na aplikácii.

Rozhodol som sa použiť neprekrývaný V/V, pretože je jednoduchší a pre účely programovacieho nástroja postačujúci. Blokovanie volaní bude využité ako synchronizácia pre implementáciu jednoduchého protokolu komunikácie medzi software a firmware.

2.5.1.1 Otvorenie a zatvorenie sériového portu

Sériový port sa otvorí funkciou `CreateFile()`, ktorá okrem otvorenia súborov slúži aj pre otvorenie zariadení. Zariadenie ale nemá v súborovom systéme svoj súbor, ktorý by ho reprezentoval. V reťazci cesty k otváranému zdroju sa musí použiť špeciálna sekvencia znakov \\.\ určujúca, že reťazec nie je cesta k súboru v súborovom systéme, ale označenie zariadenia v mennom priestore zariadení. Napríklad pre otvorenie COM portu 137 sa uvedie cesta \\.*COM137*. Ďalšími parametrami funkcie sa určuje otvorenie zariadenia pre zápis aj čítanie a pre neprekrývaný mód.

Funkcia vracia identifikátor otvoreného zariadenia, ktorý sa používa ako identifikácia zariadenia vo všetkých ďalších funkciách pre prácu s ním. Identifikátor je dátového typu `HANDLE`. V prípade neúspechu pri otváraní funkcia `CreateFile()` vráti hodnotu `INVALID_HANDLE_VALUE`.

Funkcia `CloseHandle()` handle zatvorí zariadenie špecifikované jej parametrom dátového typu `HANDLE`.

2.5.1.2 Nastavenie parametrov komunikácie

Pre zistenie maximálnej komunikačnej rýchlosti v baudoch sa používa funkcia `GetCommProperties()`, ktorá naplní informačnú dátovú štruktúru `COMMPROP` informáciami o COM porte. Jej člen `dwMaxBaud` môže nadobúdať hodnoty z konečnej množiny konštánt popisujúcich maximálnu možnú komunikačnú rýchlosť. Maximum môže byť fixná hodnota, alebo definovaná používateľom.

Funkcia `SetCommState()` nastavuje parametre portu podľa dátovej štruktúry `DCB`. Získať štruktúru `DCB` portu je možné funkciou `GetCommState()`.

V štruktúre sa nastavuje počet dátových bitov, počet stop bitov, parita a komunikačná rýchlosť. Hodnoty pre `dwMaxBaud` v štruktúre `COMMPROP` sa nemôžu priraďovať do člena `BaudRate` v štruktúre `DCB`. Hodnotou `BaudRate` je priamo počet baudov. Niektoré štandardné rýchlosti sú definované ako pomenované konštanty.

2.5.1.3 Vysielanie a príjem dát po sériovej linke

Funkcia `WriteFile()` zapíše do zariadenia zadaný počet bajtov z dátového poľa určeného ukazovateľom. Vo výstupnom parametri vracia funkcia počet skutočne zapísaných bajtov. Ten sa môže líšiť od požadovaného počtu bajtov pre zápis. Návratový kód vyjadruje úspech alebo neúspech vo funkcii.

`ReadFile()` pracuje veľmi podobne. Prijíma počet bajtov pre prečítanie do dátového poľa určeného ukazovateľom. Vo výstupnom parametri vracia počet skutočne prečítaných bajtov, čo sa môže odlišovať od požadovaného počtu. Návratový kód vyjadruje úspech alebo neúspech vo funkcii.

2.5.1.4 Časové limity dátových prenosov

`GetCommTimeouts()` naplní dátovú štruktúru `COMMTIMEOUTS` pre určený port. `COMMTIMEOUTS` obsahuje tri hodnoty určujúce časové limity pre čítanie a dve pre zápis. Všetky hodnoty sú v milisekundách. Nastavenia z dátovej štruktúry sa na port aplikujú volaním funkcie `SetCommTImeouts()`. Význam členov štruktúry `COMMTIMEOUTS` je nasledujúci:

- `ReadIntervalTimeout` je maximálny čas medzi prijatím dvoch po sebe nasledujúcich bajtov,
- `ReadTotalTimeoutMultiplier` určuje celkový limit pre jedno volanie `ReadFile()` vynásobením s počtom bajtov pre prečítanie,
- `ReadTotalTimeoutConstant` sa pripočítava k výslednému časovému limitu pre čítanie,
- `WriteTotalTimeoutMultiplier` určuje celkový limit pre jedno volanie `WriteFile()` vynásobením s počtom bajtov pre zápis,
- `WriteTotalTimeoutConstant` sa pripočítava k výslednému časovému limitu pre zápis.

2.5.1.5 Návratové kódy funkcií Win32 API

Mnou naštudované funkcie Win32 API vracajú nulovú hodnotu v prípade chyby a nenulovú v prípade úspechu. Nenulový kód špecifikujúci poslednú chybu sa získa volaním funkcie `GetLastError()`. Na textový reťazec s popisom chyby sa kód prevedie funkciou `FormatMessageA()`. Pre prevod používa systémové tabuľky s chybovými správami vo zvolenom jazyku. Alokuje pamäť pre buffer pre uloženie znakového reťazca s popisom chyby. Táto pamäť sa uvoľní volaním `LocalFree()`. Písmeno `A` v názve funkcie `FormatMessageA()` značí, že funkcia vracia ASCII reťazec, nie reťazec širokých znakov.

2.5.2 POSIX

Rozhranie pre sériovú komunikáciu na unixových systémoch má odlišný návrh, ako na systémoch MS Windows. Historicky boli cez sériovú linku k unixovému počítaču pripojené textové terminály, cez ktoré so systémom interagovali používatelia. Preto je v konfigurácii sériového portu veľa nastavení týkajúcich sa spracovávania a transformácie znakov posielaných po sériovej linke.

2.5.2.1 Otvorenie a zatvorenie sériového portu

Sériový port sa otvorí funkciou `open()`. Sériový port je reprezentovaný súborom v súborovom systéme. Na súčasných operačných systémoch GNU/Linux sú súbory zariadení umiestnené vo virtuálnom súborovom systéme. Funkcia `open()` má dva parametre. Prvým je cesta k súboru zariadenia a druhým sú príznaky s konfiguráciou otváraného súboru.

Funkcia vracia deskriptor otvoreného súboru zariadenia, ktorý sa používa pre jeho identifikáciu vo všetkých ďalších funkciách pre prácu so zariadením. Identifikátor je dátového typu `int`. V prípade neúspechu pri otváraní funkcia `open()` vráti hodnotu -1 a premenná `errno` je nastavená na kód chyby, ktorý je možné ďalej spracovať pre určenie presnej príčiny chyby.

Funkcia `close()` handle zatvorí súbor zariadenia identifikovaný deskriptorom predaného jej ako jediný parameter.

2.5.2.2 Nastavenie parametrov komunikácie

Parametre sériového portu sú uložené v štruktúre `struct termios`. Funkcia `tcgetattr()` naplní zadanú štruktúru nastaveniami pre zariadenie sériového portu podľa deskriptoru. Funkcia `tcsetattr()` aplikuje nastavenia portu zo zadanej štruktúry na port podľa deskriptoru súboru zariadenia. Nastavenie prenosovej rýchlosti v štruktúre `struct termios` vykonávajú funkcie `cfsetispeed()` a `cfsetospeed()`. Prvá nastavuje rýchlosť prijímania a druhá rýchlosť vysielania. Prvým parametrom je ukazovateľ na štruktúru a druhým je pomenovaná konštanta pre danú rýchlosť, dostupná prostredníctvom systémových hlavičkových súborov.

Funkciou pre nastavenie parametrov otvoreného súboru zariadenia je `fcntl()`. Nastavuje sa ňou, že čítanie zo súboru bude blokujúce.

2.5.2.3 Vysielanie a príjem dát po sériovej linke

Vysielanie a príjem je realizovaný rovnako ako zápis a čítanie z a do štandardného súboru. Funkcia `write()` zapíše do súboru podľa deskriptoru zadaný počet bajtov z pamäte podľa ukazovateľa. Vráti počet zapísaných bajtov alebo -1. V druhom prípade je premenná `errno` je nastavená na kód chyby.

Funkcia `read()` sa pokúsi prečítať zo súboru podľa deskriptoru zadaný počet bajtov do pamäte podľa ukazovateľa. Vráti počet prečítaných bajtov alebo -1. V druhom prípade je premenná `errno` nastavená na kód chyby.

2.5.2.4 Časové limity dátových prenosov

Pre časovanie čítania zo sériového portu som sa rozhodol použiť funkciu `select()` pre synchrónne multiplexovanie V/V súborových operácií. Funkcia umožňuje čakať na zadané udalosti na súborových deskriptoroch po zadanú dobu.

Čas preberá v štruktúre `struct timeval` v sekundách a mikrosekundách. Funkcia `select()` môže čakať na tri typy udalostí: zápis do súboru je možný, čítanie zo súboru možné (existujú dáta pre prečítanie) a výnimka na súbore. Medzi parametrami preberá funkcia príslušné tri množiny sledovaných súborových deskriptorov. Tieto množiny reprezentuje dátový typ `fd_set`. Makrom `FD_ZERO` sa množina nuluje, makrom `FD_SET` sa do množiny pridáva deskriptor.

Ostatné makrá nie sú pre moje účely použité, keďže budem sledovať len jeden typ udalosti na jednom deskriptore.

`select()` vracia kladné číslo, čo je počet deskriptorov, na ktorých nastala sledovaná udalosť. Nulu vracia po uplynutí časového limitu a -1 pri chybe. V premennej `errno` nastaví kód popisujúci chybu.

2.5.2.5 Návratové kódy funkcií

Funkcie majú buď definované svoje vlastné špecifické návratové hodnoty, alebo využívajú mechanizmus hlásenia založený na premennej `errno`. V tomto prípade funkcia pri neúspechu vráti -1 a globálna premenná `errno` je nastavená na kód chyby. Tento kód je možné previesť na znakový reťazec s popisom chyby funkciou `strerror()`.

Kapitola 3

Návrh podpory programovania

Pre vývoj nástroja pre programovanie som zvolil agilnú metodiku. Je charakterizovaná na [21]. Uprednostnil som ju preto, že vývoj sa neuberá presnými cestami, ale vývojár si môže ďalšie kroky k dosiahnutiu cieľa voliť ľubovoľne. Veľmi sa blíži k modelu vývoja v oblasti slobodného software, s ktorým mám najviac skúseností. Považujem ju za obzvlášť vhodnú pre malé projekty, v ktorých si dokáže udržať prehľad jeden vývojár. Tiež umožňuje flexibilne reagovať na nové poznatky, ktoré sa objavujú až pri implementácii návrhu.

Táto kapitola popisuje návrhové rozhodnutia nutné pre začatie realizácie. Návrh ďalších rysov vývoja nástroja pre programovanie budem diskutovať až v kapitole o realizácii a tam aj popisovať okolnosti vývoja, čo viedli ku zmene pôvodných návrhov.

3.1 Model nasadenia

Mechanizmus bootstrap zavádzača je obecný a dá sa využiť aj pre manipuláciu s programovou pamäťou MCU. Program, ktorý bude operovať s FLASH pamäťou musí byť vykonávaný procesorom MCU. Nie je možné požadovanú funkcionalitu realizovať len v software pre PC, pretože cez linky sériového rozhrania nie je možné komunikovať s kontrolérom programovej FLASH pamäte v MCU.

Programovací nástroj bude preto rozdelený na dve časti: software pre PC a firmware pre MCU. Tieto dve časti budú spolu obojsmerne komunikovať cez sériovú linku.

3.2 Tok programu vo firmware mikrokontroléra

Bootstrap zavádzač prijíma len 32 bajtov programového kódu a to je pre implementáciu manipulácie s FLASH nedostatočné. Firmware bude preto pozostávať z troch oddelených častí a zavedenie firmware bude trojúrovňové.

Obr. 3.1: Model nasadenia programovacieho nástroja

Do RAM MCU sa najprv nahrá a spustí 32 bajtov firmware prvej úrovne cez bootstrap zavádzač. Hlavnou úlohou firmware prvej úrovne je prekonať obmedzenie bootstrap zavádzača na 32 bajtov prijatého a spusteného programového kódu. Firmware druhej a tretej úrovne bude prijímaný firmware prvej úrovne a preto môže mať už veľkosť väčšiu ako 32 bajtov. Firmware prvej úrovne prijme programový kód druhej úrovne a zapíše ho do RAM tak, aby neprepísal sám seba a predá druhej úrovni riadenie.

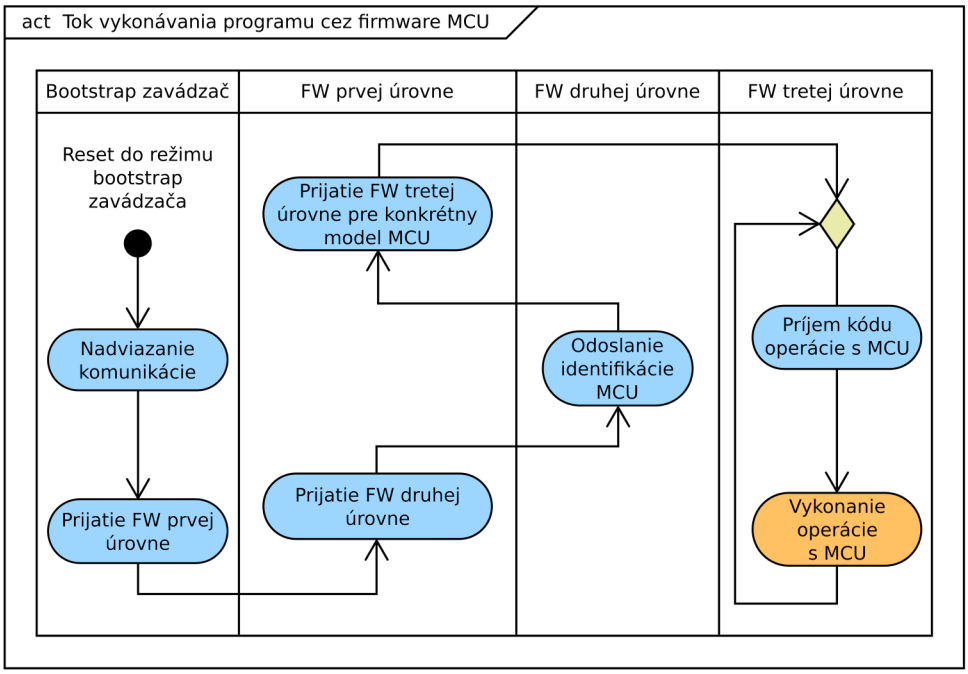
Firmware druhej úrovne prečíta identifikačné registre MCU a odošle ich software v PC. Podľa toho software bude vedieť, aký MCU je pripojený na sériovom rozhraní a vyberie správny firmware tretej úrovne s podporou operácií pre konkrétny model. Z kódu druhej úrovne sa riadenie vráti zase na prvú úroveň. Tam sa bude očakávať prijatie firmware tretej úrovne.

Po spustení tretej úrovne firmware sa spustí príkazová slučka. Bude prijímať kódy operácií po sériovej linke a podľa toho spustí danú operáciu. Konkrétna operácia môže odosielať alebo prijímať ďalšie dáta. Vďaka príkazovej slučke je možné vykonávať s MCU viac operácií bez nutnosti hardwarového resetu a opakovaného nahrávania firmware v režime bootstrap zavádzača.

Príkazové rozhranie bude pre software v PC pre rôzne MCU rovnaké. Navrhnutý tok programu cez úrovne firmware znázorňuje diagram na obrázku 3.2.

3.3 Komunikačný protokol medzi software a firmware

V tejto podkapitole popíšem komunikačný protokol medzi software v PC a firmware v MCU, zo strany MCU. Komunikačný protokol ilustrujú UML diagramy v prílohe C. Sú to diagramy aktivít. Činnosti podfarbené modrou farbou sa týkajú odosielania a prijímania informácií po sériovej linke. V ostatných činnostiach komunikácia po sériovej linke neprebieha. Diagramy nepopisujú implementačné detaily vo firmware MCU. Tie je možné nájsť v zdrojových kódoch v prílohe B. Komunikačný protokol je pre oba modely podporovaných MCU rovnaký.



Obr. 3.2: Tok vykonávania programu cez firmware mikrokontroléra

Chybové kódy používané protokolom majú rozsah jeden bajt. Nenulový bajt je kód chyby, nulový bajt kód úspechu.

Chybová komunikácia je jednosmerná. Software sa dozvie o chybe vo firmware, ale nie naopak (napr. pri skončení software na signál klávesovej skratky CTRL-C).

Komunikáciu po sériovej linke pri zavádzaní firmware tretej úrovne do RAM MCU znázorňuje diagram C.3. Software po svojom spustení vždy pošle nulový bajt. Na nulový bajt reaguje bootstrap zavádzač, aj príkazová slučka (diagram C.4). Podľa prijatého bajtu sa software dozvie, v akom stave sa MCU nachádza. Keď prijme identifikačný bajt bootstrap zavádzača, vie že MCU nie je inicializovaný a spustí viacstupňový proces, ktorého výsledkom je nahranie a spustenie firmware tretej úrovne v pamäti RAM MCU. Keď prijme identifikačný bajt príkazovej slučky, vie že už môže posielať kód operácie.

Mazanie blokov je na diagrame C.5. Mazanie celej internej FLASH pamäte na diagrame C.6. ST10F168 mazanie celej pamäte jedným príkazom pre kontrolér FLASH nepodporuje. Simuluje ho mazaním všetkých blokov FLASH. Komunikčný protokol je ale v prípade oboch modelov MCU rovnaký.

Zápis do FLASH je na diagrame C.7. Zápis na diagrame C.8. V oboch sa odosiela po prečítaných/zapísaných 1024 bajtoch počet zostávajúcich bajtov pre prečítanie/zapísanie. Stav čítača sa tiež odosiela po skončení operácie. Toto slúži na synchronizáciu komunikácie. Odoslanie hodnoty čítača ukazuje

diagram C.9.

Pre prijímanie príkazov v príkazovej slučke a informácií pre spustenie operácie som navrhol spôsob prijatia dát, ktorý je odolnejší proti chybám prenosu. Nazval som ho *bezpečný príjem*. Firmware prijaté dáta odošle späť software. Software sa tým dozvie, či firmware prijal správne dáta. Software prijaté dáta znova odošle firmware. Firmware nakoniec odpovie kódom chyby, alebo úspechu.

Bezpečný príjem slova a dvojslova, je na diagramoch C.1 a C.2. Príkazová slučka potrebuje *bezpečne prijať* všetky bajty, okrem nulového bajtu, aby tak simulovala prijatie nulového bajtu bootstrap zavádzačom. Preto si bezpečný príjem implementuje sama.

3.4 Návrh software

Pretože primárne používam operačné systémy GNU/Linux, budem programovací nástroj vyvíjať na tejto platforme s ohľadom na prenositelnosť. Jediná platformovo závislá časť software je ovládanie sériového portu. Táto časť bude od zvyšku platformovo nezávislej časti software oddelená použitím rozhrania. Nástroj najprv odladím na OS GNU/Linux. Potom na systéme MS Windows 7 vyviniem a otestujem podporu pre systémy MS Windows. Z informácií o kompatibilite funkcií Win32 API na [18] vyplýva, že jediná implementácia bude na úrovni zdrojového kódu funkčná aj na vyšších verziách operačných systémov MS Windows, ktoré má programovací nástroj podporovať.

Software bude vo forme spustiteľného súboru pre operačný systém. Spustiteľný súbor bude obsahovať firmware v binárnej podobe, ktorý bude nahrávaný do RAM MCU.

3.5 Voľba jazyka implementácie

Firmware som sa rozhodol písať v jazyku Assembler. V [22] je doporučované písať kód pre manipuláciu s FLASH pamäťou v jazyku Assembler, aby sa zabránilo rôznym optimalizáciam kompilátora jazyka vyššej úrovne (napr. jazyka C). Pri príkazoch pre kontrolér FLASH musí byť použité jedine nepriame adresovanie cez register. Podobne to platí pre MCU ST10F168. Na zvyšnú časť kódu firmware už by bol jazyk vyššej úrovne zbytočný, pretože tok programu bude veľmi jednoduchý a výhody jazyka vyššej úrovne (selekcie, iterácie) by neboli využité.

Pre implementáciu software som zvolil najprv jazyk C. Rozhodol som sa tak preto, že s jazykom C mám skúsenosti, prekladač jazyka C je dostupný na operačných systémoch GNU/Linux, MS Windows a iných. Z jazyka C je možné priamo volať funkcie zo systémových knižníc, hlavne čo sa týka komunikácie cez sériový port. Nie je potrebné inštalovať žiadne knižnice, ktoré

by poskytovali API pre volanie systémových funkcií sériovej linky hostiteľského systému, ako napr. v jazyku Java.

3.6 Použité vývojové nástroje

3.6.1 Vývoj firmware

Pre vývoj firmware v jazyku Assembler som si vybral vývojové prostredie µVision 5 a vývojové nástroje C166 (prekladač, linker) od firmy Keil. Informácie o vývojových nástrojoch C166 som čerpal z [23].

Tieto produkty majú na [24] veľmi obsiahlu dokumentáciu a sú tam k dispozícii na stiahnutie zadarmo vo forme skúšobných verzií. Podmienky ich použitia a obmedzenia sú uvedené na [25].

Keil µVision 5 oficiálne podporuje len operačné systémy MS Windows. Zistil som však, že bez chýb funguje v OS GNU/Linux s vrstvou kompatibility Wine. Vývojové prostredie som používal v OS openSUSE Leap 42.1 64-bit s nainštalovaným Wine 1.8.4.

Skúšobné verzie nemajú časové obmedzenia, ale majú limitovanú veľkosť výstupných objektových súborov na 4 KiB. Po zvážení limitu na objektový kód som zistil, že táto hranica bude pre programový kód v jazyku Assembler dostatočná. Maximum funkcionality chcem presunúť do SW na PC. Uvažoval som tak, že priemerná veľkosť inštrukčného kódu je približne 2,8 bajtu (vychádzal som z informácií v [26]), $4096/2,8 \approx 1462$. Katalógové listy [13] a [14] pre MCU obsahujú príklady programov v jazyku Assembler pre programovanie FLASH a tie majú spolu približne 120 inštrukcií. To znamená $120 \times 2,8 = 336$ bajtov a to je približne $\frac{1}{12}$ z 4096 bajtov.

3.6.2 Vývoj software

Pre vývoj software som si vybral nástroje projektu [27]. Projekt predstavuje kompletné prostredie pre beh GCC pre podporu natívnych binárnych súborov pre 32-bitové a 64-bitové verzie operačných systémov MS Windows. Snahou bolo použiť rovnaké nástroje na MS Windows aj GNU/Linux, pre odstránenie nutnosti oboznamovať sa s odlišnými nástrojmi pre kompiláciu. Ďalším dôvodom bolo dodržanie jednotného systému pre zostavovanie software v podporovaných operačných systémoch.

GCC je systém kompilátorov známy hlavne z unixových operačných systémov a je štandardnou súčasťou väčšiny distribúcii GNU/Linux. Súčasťou inštalácie mingw-w64 je prekladač jazyka C a C++, linker a nástroje pre spracovávanie Makefile súborov obsahujúcich informácie o závislostiach medzi zdrojovými súbormi pre automatické zostavovanie software.

3.6.3 Systém pre zostavovanie software

Pre softwarové projekty pozostávajúce z viacerých zdrojových súborov by bola manuálna kompilácia a linkovanie jednotlivých zdrojových súborov do výsledného spustiteľného súboru veľmi zdĺhavá. Hlavne pri zmene len niektorých zo zdrojových súborov je kompilácia a linkovanie len dotknutých častí software náchylná na opomenutie závislostí medzi zdrojovými súbormi.

Systém pre zostavovanie software dohliada na závislosti medzi zdrojovými súbormi a pri ich zmene sám podľa nich určí, ktoré je nutné znovu spracovať a ako vytvoriť aktualizovaný spustiteľný súbor. Môže sa skladať z viacerých nástrojov.

Rozhodol som sa použiť program GNU make, pretože je štandardnou súčasťou unixových operačných systémov a pre operačný systém MS Windows je dostupný v projekte mingw-w64. GNU make po svojom zavolaní bez parametrov spracováva súbor so štandardným názvom Makefile v aktuálnom adresári. V Makefile sú uvedené závislosti medzi zdrojovými súbormi a spôsob, ako zdrojové súbory spracovávať (nemusí sa jednať len o proces kompilácie a linkovania).

Súbory Makefile môžu byť vytvárané manuálne alebo automaticky. Vzhľadom na malý rozsah programovacieho nástroja (v jednotkách zdrojových súborov) som si v tejto bakalárskej práci zvolil manuálne vytvorenie súborov Makefile.

Odhad rozsahu sa však vo fáze realizácie ukázal ako nesprávny. To viedlo k rozšíreniu systému pre zostavovanie software o automatické generovanie Makefile súborov.

3.7 Vkladanie dát do zdrojových súborov software

Aby bol program jednoducho premiestniteľný v rámci súborového systému konkrétneho OS a aj medzi rôznymi PC, rozhodol som sa všetky dátové zdroje vložiť priamo do spustiteľného súboru programu. Program nebude vyžadovať inštaláciu a bude jednoducho premiestniteľný ako jediný súbor v rámci súborového systému OS a medzi rôznymi PC.

Operačný systém MS Windows umožňuje do spustiteľných súborov vkladať dátové zdroje, ktoré program využíva. Tento spôsob sa nazýva *Application Resources* ale je závislý na platforme MS Windows.

Pretože program má byť multiplatformný, zvolil som si možnosť transformácie dát do dátovej štruktúry v zdrojovom kóde v programovacom jazyku implementácie.

Aby som si nemusel vytvárať vlastný nástroj pre transformáciu, hľadal som na internete. Našiel som jednoduchý nástroj [28], ktorý podporuje unixové systémy aj MS Windows. Vypisuje súbory v šestnástkovej, desiatkovej alebo osmičkovej sústave. Tiež má možnosť vypísať súbor ako deklaráciu poľa v jazyku C obsahujúceho bajty súboru. Program XD prijíma názov vstup-

ného a výstupného súboru. Je uvoľnený jeho autorom pre použitie ako *Public domain*.

Kapitola 4

Realizácia

4.1 Prepojenie PC a mikrokontroléra

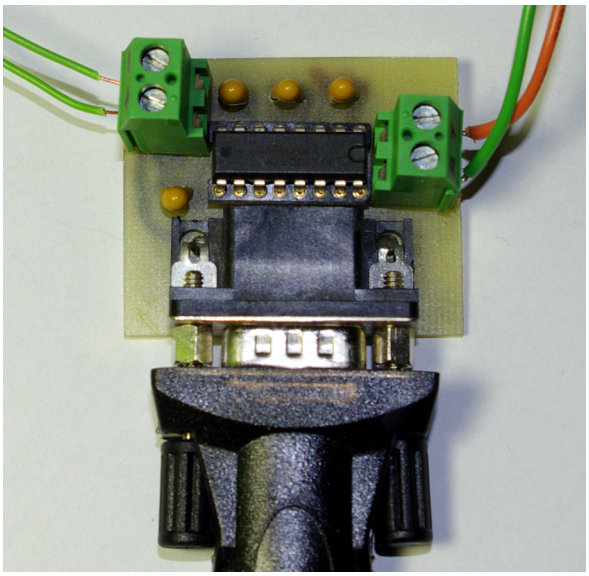
Programovací nástroj som vyvíjal a testoval na notebooku Lenovo THINKPAD X220i, ktorý nemá sériový port. Použil som preto prevodník medzi USB 2.0 a RS-232 od výrobcu PremiumCord, na obrázku 4.1. Na CD dodanom k prevodníku boli ovládače pre MS Windows 2000, Windows Server 2008, Windows XP, Windows Vista, Windows 7, Windows 8, GNU/Linux a Mac OS v8.6 a vyššie.

Sériové rozhranie MCU pracuje s TTL úrovňami signálov. Pre prevod signálov medzi úrovňami RS-232 a TTL som použil obvod MAX232IN od firmy Texas Instruments v typickom zapojení uvedenom v katalógovom liste [29]. Prípravok vlastnej výroby ukazuje obrázok 4.2.

Jednotky AVR3 a prevodník medzi RS-232 a TTL som napájal spínaným adaptérom Vigan VSZ-24-01 s výstupným napätím 24V a max. výstupným prúdom 1A.

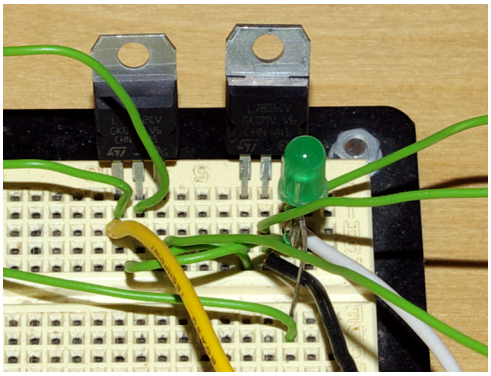
MCU ST10F168 vyžaduje pre programovanie FLASH pamäte priviesť na

Obr. 4.1: Prevodník medzi USB 2.0 a RS-232 PremiumCord

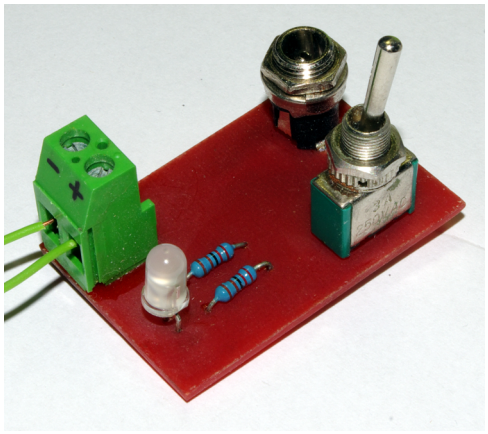


Obr. 4.2: Prevodník medzi RS-232 a TTL úrovňami signálov

Obr. 4.3: Spínaný napájací adaptér Vigan VSZ-24-01



Obr. 4.4: Zapojenie stabilizátorov napätia na kontaktnom poli



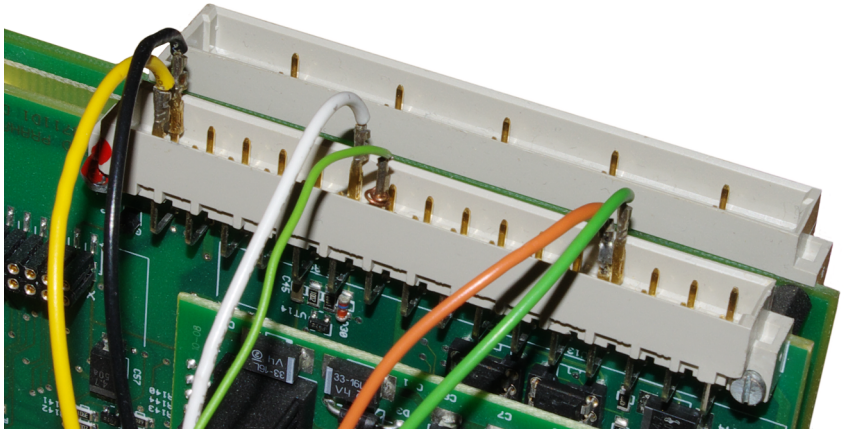
Obr. 4.5: Spínací prípravok pre napájacie napätie

svoj vývod č. 84 napätie 12V. Obvod MAX232 je zase napájaný 5V. Napätie 12V som získal použitím stabilizátora L78S12CV a 5V použitím stabilizátora L7805CV. Informácie o zapojení stabilizátorov som čerpal z [30] a [31]. Zapojenie obvodu so stabilizátormi na kontaktnom poli je na obrázku 4.4

Pri skúšaní firmware som potreboval MCU často resetovať pre vstup do režimu bootstrap zavádzača. K tomuto účelu som hneď za napájací adaptér zaradil spínač s indikáciou stavu dvojfarebnou LED. Ak je spínač zopnutý, svieti LED zeleno. Ak je spínač rozopnutý, svieti LED červeno. Vďaka tomu je možné rozlišovať okrem spomenutých dvoch stavov ešte jeden: ak napájací adaptér nie je zapojený v zásuvke, LED nesvieti. Osadené rezistory obmedzujú prúd tečúci LED. Prípravok pre spínanie napájania je tiež vlastnej výroby.

Pripojenie dosky AVR3 s MCU ST10F168 je na obrázku 4.6. Pripojené sú:

- napájacie napätie 24 V pre AVR3 (žltý a čierny vodič),
- kontakty sériovej linky (zelený a oranžový vodič napravo),



Obr. 4.6: Pripojenie dosky s ST10F168

- signál BSL pre vstup do režimu bootstrap zavádzača pri hardwarovom resete MCU (biely vodič),
- a napätie 12 V pre programovanie FLASH (zelený vodič v strede).

Pripojenie dosky s ST10F269 sa líši len v odpojení napätia pre programovanie FLASH. Schémy uvedenej podpornej elektroniky sú v prílohe D.

4.1.1 Rozdiel v napájaní ST10F168 a ST10F269

Najprv som vyvíjal podporu pre programovanie ST10F269, ktorý pripojenie 12 V na kontakt konektoru na vývojovej doske nevyžadoval. Keď som potom začal s vývojom podpory pre ST10F168, po spustení STEAK operácie zápisu MCU prestal odpovedať na sériovom rozhraní a spracovávanie programu sa zdanlivo zastavilo. Keď som ale predal STEAK programom nesprávnu hodnotu niektorého parametra, všetko šlo správne podľa dokumentácie. STEAK vrátil kód chyby a vykonávanie programu pokračovalo. Vyskúšal som preto ST10Flasher. S tým mazanie ani zápis FLASH ST10F168 nefungovali tiež. Z toho som nabral podozrenie, že je to chyba dosky s ST10F168. Preto som kontaktoval pracovníka firmy AŽD Praha s.r.o. a programovanie sme vyskúšali vo firme s ich technickými prostriedkami pre pripojenie dosiek k PC a s programom ST10Flasher. Tam všetko fungovalo.

Vrátil som sa ku hľadaniu príčiny chyby na mojej strane. Vyskúšal som spustiť operácie zápisu slova, dvojslova. Vždy MCU prestal odpovedať. Potom mi napadlo vyskúšať ešte spustiť mazanie blokov. Pri nej sa vykonávanie programu nezastavilo a STEAK vrátil chybový kód. V [15] bol popísaný význam kódu. Kód signalizoval, že nie je prítomné napätie 12 V na kontakte Vpp puzdra MCU. Podrobnejšie som si preštudoval katalógový list ST10F168 [13] a našiel som informáciu, že pre programovanie FLASH je potrebné na

vývod 84 puzdra PQFP144 MCU pripojiť napätie 12 V. Zo schémy vývojovej dosky som vysledoval pripojenie toho vývodu na vývod 14C konektoru dosky a napätie 12V dodal pomocou stabilizátora L78S12CV.

Potom už programovanie s ST10Flasher fungovalo, ale s novým vyvýjaným programovacím nástrojom stále nie a MCU prestával odpovedať. V hľadaní riešenia som pokračoval a identifikoval som problém s rozložením pamäte MCU, ktorý bližšie popisujem v podkapitole 4.5. To, že sa spracovávanie programu zastavilo pri operácii zápisu, ale nie mazania bolo spôsobené tým, že STEAK podprogram pre mazanie, na rozdiel od podprogramov pre zápis, nezapĺňal zásobník tak, aby si prepísal dôležité registre. Preto prebehol správne a vrátený chybový kód bol stopou k riešeniu problému. Po jeho vyriešení už programovanie ST10F168 fungovalo.

4.2 Úprava programu XD

Pre moje potreby som si nástroj XD upravil. Úpravy boli robené pre možnosť vkladania binárnych dát s firmware MCU do software v jazyku C++ a pre vkladanie textu s nápovedou programu, ako zdrojových súborov programu v jazyku C++. Vstavanú nápovedu nástroja XD som príslušne pozmenil a pridal informácie o modifikácii.

Pre vkladanie dát firmware som zmenil existujúcu funkcionalitu výpisu binárnych dát. Do výstupného súboru som pridal *include guard* definíciu pre ošetrenie viacnásobného vloženia hlavičkového súboru. Zmenil som dátový typ položky poľa na `uint8_t` pre zaistenie dátového rozsahu jeden bajt. Pridal som definíciu konštanty uvádzajúcej veľkosť definovaného poľa v bajtoch. Všetky definície som uzavrel do menného priestoru, pre zabránenie kolízii názvov symbolov v programe. Výpis dátovej definície si používateľ vyžiada prepínačom `-d` s názvom poľa a názov menného priestoru uvedie za prepínačom `-p`. Vzor výstupu pre vkladanie binárnych dát je vo výpise 4.1.

Zdrojový kód 4.1: Vzor výstupu XD pre binárne dáta

```
#ifndef FW_STAGE_1_HPP
#define FW_STAGE_1_HPP 1

namespace FwCommon {

uint8_t fw_stage_1[] = {
    47,137,15,0,230,240,0,224,154,183
};

const size_t fw_stage_1_length = 10;

}

#endif
```

Pre pre vkladanie obsahu textových súborov pre použitie v programe som pridal prepínač `-t`. Výstup tiež obsahuje *include guard* a menný priestor. Prístup k textu je realizovaný prostredníctvom definovanej metódy, ktorá text vracia v type `string`. Pre postupnú konštrukciu reťazca používa objekt triedy `std::ostringstram`. Do textu som potreboval vložiť vlastný text, konkrétne do nápovedy k programu názov programu. Preto XD pri prechádzaní textu hľadá reťazec `%ARG%` a ten v návratovom reťazci nahradí argumentom funkcie. Vzor výstupu pre vkladanie obsahu textového súboru je vo výpise 4.2.

Zdrojový kód 4.2: Vzor výstupu XD pre textový súbor

```
#ifndef HELP_MESSAGE_HPP
#define HELP_MESSAGE_HPP 1

namespace HelpMessage {

std::string
getText(const std::string & arg)
{
    std::ostringstream os;

    os << "Usage: " << arg << " OPERATION" << std::endl;
    os << "" << std::endl;
    os << "    OPERATION  read, write" << std::endl;

    return os.str();
}

}

#endif
```

Takto môže byť text nápovedy udržovaný a modifikovaný ako samostatný jednoduchý textový súbor `HelpMessage.txt` v zdrojovom strome namiesto toho, aby bol text súčasťou zdrojového kódu. Nevýhodou druhého spomenutého spôsobu je rozdelenie textu na riadky. Prekladač jazyka C++ nerozoznáva znaky nového riadka v reťazcových konštantách. Tie je treba uvádzať napr. pomocou manipulátora `std::endl` alebo kontrolnej sekvencie `\n`. Pri zmene rozdelenia textu na riadky je potrebné zmeniť rozloženie týchto manipulátorov a pre čitateľnosť ešte aj rozložiť text do viacerých volaní metód alebo priradení tak, aby riadky s textom v zdrojovom súbore zodpovedali zamýšľanému riadkovaniu textu a neboli len jedným dlhým riadkom. Pri udržovaní textu v obyčajnom textovom súbore a jeho automatickom spracovaní starosti s formátovaním textu pre programovací jazyk odpadajú.

Zdrojový kód `xd.c` upraveného programu XD som začlenil do zdrojového stromu do adresára `utils/xd/`. Zaradil som ho do procesu zostavovania výsledného spustiteľného súboru aplikácie a tak pri zmene XD dôjde tiež k jeho prekompilovaniu.

4.3 Zmena programovacieho jazyka software

V analýze som si pre implementáciu zvolil jazyk C. Program som navrhol tak, aby som mohol chybové hlášky vypisovať a program ukončovať len na jednom mieste programu (v jednej funkcii alebo jednom zdrojovom súbore) a tak sa dalo ukončovanie behu programu prehľadne kontrolovať. Ako miesto pre ukončovanie programu som zvolil funkciu `main()`. Pri zanorení programu na viac ako jedno volanie bolo nutné predávať informácie smerom k volajúcemu cez viacero funkcií. Každá funkcia musela mať možnosť, ako predať informáciu o výskyte chyby. To znamenalo buď využitie návratovej hodnoty funkcie, alebo jej výstupného parametra, ktorý by sa inak mohol vynechať v prípade, kedy funkcia samotná žiadne informácie o chybe volajúcemu predávať nepotrebovala. Tiež to znamenalo veľa selekcií pre kontrolu návratovej hodnoty volaných funkcií v programe. To programový kód zneprehľadňovalo.

Preto som sa rozhodol prepísať software do jazyka C++ a využiť výnimky. Odpadla tak potreba predávania informácie o chybe cez viacero úrovní volaní funkcií a potreba selekcií v programe. V jazyku C++ môžem chybu vyhodiť ako výnimku a zachytiť ju až o niekoľko úrovní vyššie vo funkcii `main()`, kde ju spracujem a beh program ukončím. Zjednodušili sa tak signatúry funkcií a metód, zjednotilo sa spracovanie chybových stavov a program sa tak sprehľadnil.

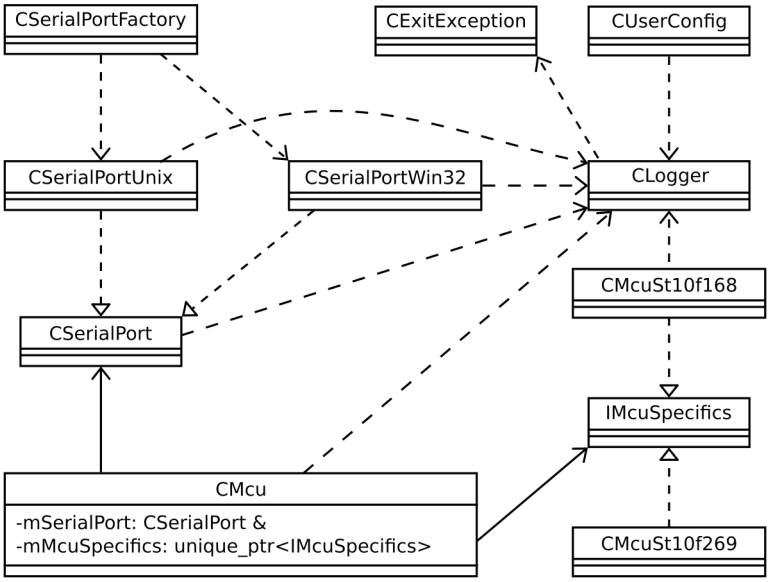
Pri písaní v jazyku C++ som objavil ďalšie rysy tohoto jazyka, ktoré program sprehľadnili a zjednodušili. Vďaka objektovému návrhu sa mohla funkcionalita programu zapúzdriť do logických celkov, a mohli sa použiť menné priestory pre zjednodušenie tvorby názvov symbolov v programe. Pracovník firmy AŽD s.r.o. mi poradil využiť automatickú správu pamäte pomocou chytrých ukazovateľov (angl.*smart pointers*) jazyka C++. Odpadla tak nutnosť manuálneho uvoľňovania pamäte. Tak sa eliminovali ďalšie možné chyby v programe.

4.4 Software

Do software som presunul väčšinu funkcionality a rozdelil som ho do častí, ktoré sa premietli do návrhu tried. Vzťahy medzi triedami znázorňuje diagram 4.7. Diagram je zjednodušený pre prehľadnosť. Uvádza len dátové členy podieľajúce sa na vzťahoch asociácií tried a neuvádza metódy tried.

4.4.1 Používateľské rozhranie

Programovací nástroj má riadkové používateľské rozhranie. Jeho prvým argumentom je názov operácie (zápis, čítanie, mazanie a pod.) a za ním nasledujú prepínače a názov vstupného alebo výstupného súboru. Prepínače, ktoré prijímajú hodnotu, od nej musia byť oddelené medzerou. Poradie prepínačov a súboru môže byť ľubovoľné.



Obr. 4.7: Návrhový diagram tried

4.4.2 Hlavný program

Hlavným zdrojovým súborom je `main.cpp`. Na začiatku funkcie `main()` sa zaregistruje obsluha pre ukončenie programu na signál SIGINT vyvolaný klávesovou skratkou CTRL+C. V tejto obsluhe sa zatvorí otvorený sériový port a program sa ukončí. Po registrovaní obsluhy signálu sa vytvorí objekt triedy `CSerialPort` s podporou operácií so sériovým portom pre danú platformu.

Vytvorí sa objekt triedy `CUserConfig` a jeho konštruktoru sa bez zmeny predá počet argumentov `argc` a ich pole `argv`, ktoré obdržala od volajúceho funkcia `main()`. Trieda `CUserConfig` spracuje argumenty a umožní sa pomocou svojich metód dotazovať na konfiguráciu programu.

Následne sa podľa používateľsky zvolenej konfigurácie vykoná zvolená operácia s príslušnými argumentami. Argumenty pre konkrétnu operáciu kontroluje trieda `CUserConfig`. Operácia musí byť zadaná, inak sa vypíše chybové hlásenie a program skončí. Pre operácie

- výpisu vstavanej nápovedy,
- výpisu informácií o programe (názov, verzia),
- a výpisu rýchlostí sériového portu

sa nekomunikuje s MCU. Objekt triedy `CMcu` sa preto nevytvára. Pre operácie

- identifikácie pripojeného MCU,
- mazania,
- čítania,

- a zápisu do FLASH

sa vytvorí objekt triedy `CMcu`.

Trieda `CMcu` zapúzdruje funkcionalitu súvisiacu s pripojeným MCU. V konštruktore preberá objekt sériového portu, s ktorým má operovať.

Výnimky sa zachytávajú a program končí s návratovým kódom obsiahnutým v objekte výnimky. Pri ukončení programu sa sériový port zatvorí.

Ak si používateľ vyžiadal výpis rýchlostí podporovaných sériovým portom, zavolá sa metóda `CSerialPort::getSpeeds()` sériového portu a ňou vrátený zoznam sa vypíše na štandardný výstup.

Ak pri operácii mazania používateľ nezadal koľko bajtov sa má prečítať, prečíta sa celá pamäť. To sa odráža v preťaženej metóde `CMcu::read()`. Ak má len jeden argument, je to príznak toho, či sa má vypisovať percentuálna informácia o priebehu čítania. Ak má dva argumenty, prvým je počet bajtov pre prečítanie a druhý je znovu príznak vypisovania informácie o postupe. Po prečítaní pamäte sa dáta zapíšu do súboru.

Ak používateľ pre operáciu mazania zadal zoznam blokov pre mazanie, zmažú sa uvedené bloky. Ak nezadal, zmaže sa celá pamäť. Obe možnosti implementuje preťažená metóda `CMcu::erase()`. Bez argumentu maže celú pamäť, inak prijíma zoznam blokov pre zmazanie.

Pri operácii zápisu sa prečítajú dáta zo vstupného súboru. Ak si používateľ vyžiadal mazanie celej pamäte, zmaže sa celá. Inak sa mažú len bloky, podľa rozsahu zapisovaných dát. Po zápise sa na požiadanie používateľa z pamäte prečíta zapisovaný počet bajtov a porovná sa s dátami prečítanými zo vstupného súboru. Toto slúži ako kontrola zápisu. Predvolene sa zápis nekontroluje spätným prečítaním, pretože chyby zápisu do FLASH detekuje kontrolér v MCU.

Pri písaní funkcií pre zápis a čítanie zo súboru som narazil na platformovú odlišnosť. Druhý parameter funkcie `fopen()` určuje mód prístupu k súboru (`"r"` čítanie a `"w"` zápis). Na platforme MS Windows však pri takomto nastavení prebiehajú na dátach transformácie znakov konca riadku a odriadkovania (súbory sa predvolene otvárajú ako textové). Preto mi čítanie zo súboru dávalo niekedy nesprávne hodnoty a podobne tiež pri zápise do súboru. Na MS Windows som musel otvoriť súbor v binárnom režime. Voľbu módu podľa platformy ukazuje výpis 4.3.

Zdrojový kód 4.3: Mód otvorenia súboru podľa platformy

```
#ifdef UNIX
    char mode[3] = "r";
#else
    char mode[3] = "rb";
#endif

f = fopen(fpath.c_str(), mode);
```

4.4.3 Spracovanie používateľskej konfigurácie

Trieda `CUserConfig` je deklarovaná v súbore `UserConfig.cpp`. Má za úlohu spracovať konfiguráciu predanú do `main()` funkcie. V konštruktore sa inicializujú dátové členy. Argumenty reprezentované v štýle jazyka C premennými `argc` a `argv` sa prevedú do C++ vektoru, ktorý sa potom predá metóde `CUserConfig::parseCommandLine()`.

Spracovanie argumentov v `CUserConfig::parseCommandLine()` funguje tak, že sa najprv prejdú všetky argumenty a spracujú sa tie, spoločné pre všetky operácie. Sú to

- názov sériového portu,
- voľba komunikačnej rýchlosti,
- výrečný mód,
- vypisovanie priebehu operácií čítania a zápisu
- a taktovacia frekvencia pripojeného MCU.

Spracované argumenty sa z vektoru odstránia. Program ďalej pracuje s vektorom argumentov, bez spracovaných spoločných prepínačov, nastavení a ich hodnôt.

Ďalej sa získa prvý argument a považuje sa za názov operácie. Ak nie je známy, program skončí s chybou. Ak operácia vyžaduje spracovanie svojich vlastných argumentov, zavolá sa metóda pre spracovanie príslušná k danej operácií. Je to jedna z metód

- `CUserConfig::parseEraseArguments()`,
- `CUserConfig::parseWriteArguments()`,
- `CUserConfig::parseReadArguments()`.

Pri spracovávaní hodnôt nastavení sa kontroluje ich platnosť (napr. či sa jedná o nezáporné číslo). Konfiguračné hodnoty sa ukladajú do dátových členov triedy. Na obsah týchto členov a hodnoty nastavení sa môže dotazovať programátor s použitím metód. Príkladom môže byť operácia mazania. Informácia o jej vyžiadaní je prístupná cez metódu `isEraseSet()` a zoznam blokov pre zmazanie vracia metóda `getEraseBlockList()`.

Zdrojový kód 4.4: Metódy pre dotazovanie používateľskej konfigurácie

```
bool
CUserConfig::isEraseSet()
{
    return mErase;
}

list<unsigned int>
CUserConfig::getEraseBlockList()
{
    return mEraseBlockList;
}
```

Trieda `CUserConfig` tiež poskytuje metódu pre získanie textu vstavanej nápovedy. Využíva funkciu `HelpMessage::getText()` z hlavičkového súboru `help_message.hpp` generovaného programom XD z textového súboru s nápovedou.

Zdrojový kód 4.5: Metóda pre získanie textu vstavanej nápovedy

```
const string
CUserConfig::getHelpMessage(const string & execName) const
{
    return HelpMessage::getText(execName);
}
```

4.4.4 Sériová komunikácia

Trieda `CSerialPort` reprezentuje v programe sériový port. Dedia od nej triedy `CSerialPortUnix` a `CSerialPortWin32`. Program pracuje s týmito triedami, implementujúcimi platformovo závislý prístup k sériovému portu, pomocou polymorfizmu cez dátový typ nadtypu (`CSerialPort`).

Trieda `CSerialPort` obsahuje platformovo nezávislé metódy. Niektoré z nich volajú metódy z tried `CSerialPortUnix` a `CSerialPortWin32` závislé na platforme. Triedy `CSerialPortUnix` a `CSerialPortWin32` obsahujú platformovo závislé metódy.

Pri kompilácii programu sa podľa platformy na ktorej prebieha zostavenie programu vytvorí statická knižnica s triedou `CSerialPortUnix` alebo `CSerialPortWin32`. Nadradený zdrojový kód používa len platrofmovo nezávislé metódy cez dátový typ nadtypu (`CSerialPort`).

Od toho, objekt akej triedy (pre akú platformu) má nadradený kód použiť, je odtienený pomocou návrhového vzoru *Factory*. Nadradený kód len vytvorí objekt triedy `CSerialPortFactory` a zavolá jej jedinú metódu `getSerialPort()`. Tá vráti chytrý ukazovateľ na objekt triedy `CSerialPort` s podporou sériového portu pre konkrétnu platformu.

4.4.4.1 Platformovo nezávislé metódy

Trieda `CSerialPort` obsahuje metódy pre nastavovanie časového limitu pre operáciu čítania. Hodnota je v milisekundách a je uložená v dátovom člene triedy. Metóda `setReadTimeout()` nastaví časový limit na zadanú hodnotu. Metóda `setDefaultTimeout()` nastaví predvolený časový limit 3 000 ms.

Metódy `sendSafeByte()`, `sendSafeWird()` a `sendSafeDoubleWord()` zaisťujú odosielanie bajtu, slova a dvojslova bezpečným spôsobom podľa komunikačného protokolu medzi SW a FW.

Metóda `write()` odosiela dáta na sériový port. Je blokujúca. Využíva platformovo závislú metódu `writeSingle()`. Pretože operačný systém môže zapísať do portu len časť žiadaných dát a zvyšok až pri ďalšom volaní, volá

sa `writeSingle()` až kým sa nezapíšu všetky dodané dáta. Potom, ak je počet zapísaných bajtov menší, ako minimálny počet zapisovaných bajtov, dáta sú doplnené bajtami 0xFF na požadované minumum. Metóda doplňovania je potrebná, pretože bootstrap zavádzač aj FW prvej úrovne očakáva prijatie pevného počtu bajtov, až potom pokračuje. Ak má firmware menej bajtov, je potrebné ešte odoslať výplňové bajty.

Metóda `writeWord()` využíva `write()` a zapisuje do portu slovo.

Metóda `read()` je blokujúca. Používa platformovo závislú metódu `readSingle()`. Podobne ako pri zápise dát, môže operačný systém vrátiť pri čítaní z portu menej bajtov, ako je požadované. Preto sa v `read()` volá `readSingle()` až pokým sa neprečíta požadovaný počet bajtov. `readSingle()` musí však vrátiť aspoň jeden bajt, inak sa to považuje za chybu.

Metódy `readWord()` a `readDoubleWord()` sú blokujúce a čítajú slovo a dvojslovo.

4.4.4.2 Sériová komunikácia cez Win32 API

Podpora sériovej komunikácie pomocou funkcií Win32 API je v súbore `SerialPortWin32.cpp`.

Metóda `getLastErrorAsString()` prevádza kód chyby funkcií Win32 API na znakový reťazec s popisom chyby.

`getMaxSpeed()` zistí maximálnu podporovanú komunikačnú rýchlosť. Konštanta určujúcu maximálnu rýchlosť sa nedá priamo použiť v nastavení rýchlosti sériového portu a je potrebné ju konvertovať. Z toho dôvodu je v konštruktore inicializovaný zoznam možných maximálych rýchlostí a im príšlušných hodnôt pre konfiguráciu sériového portu. Zoznam je tiež doplnený o znakový reťazec s popisom rýchlosti, pre jednoduchší výpis používateľovi. Návratovou hodnotou `getMaxSpeed()` je práve odkaz na príslušný prvok prevodného zoznamu.

Metóda `getSpeeds()` vráti zoznam textových reťazcov s popisom možných komunikačných rýchlostí pre výpis používateľovi.

Metóda `open()` volá metódu `openPort()`. `openPort()` otvorí, ale nekonfiguruje, sériový port. `openPort()` je volaná aj pri získavaní zoznamu podporovaných rýchlostí portu, preto je zdieľaná funkcionalita umiestnená vo zvláštnej metóde `openPort()`. `open()` ďalej nastaví parametre komunikácie.

Metóda `close()` zatvorí otvorený sériový port.

Metóda `setTimeouts()` nastaví konfiguráciu portu pre časovanie zápisu a čítania na sériovom porte.

Metóda `writeSingle()` implementuje platformovo závislú funkcionalitu pre zápis dát do sériového portu. `readSingle()` implementuje funkcionalitu pre čítanie z portu. Je platformovo závislá.

4.4.4.3 Sériová komunikácia na POSIX systémoch

Podpora sériovej komunikácie na POSIX systémoch je v súbore `SerialPortUnix.cpp`.

V konštruktore triedy sa nachádza kód, ktorý inicializuje prevodný zoznam pre rýchlosti komunikácie. Jedna položka zoznamu má dva prvky. Reťazec s textovým popisom rýchlosti a konštantu príslušnej rýchlosti pre zápis do konfigurácie portu. Kód pre naplnenie zoznamu je určený pri kompilácií. Zoznam sa tak naplní len rýchlosťami, ktorých konštanty sú definované na systéme, kde kompilácia prebieha.

Avšak nie všetky rýchlosti zo zoznamu naplneného v konštruktore sú podporované zariadením sériového portu pri behu aplikácie. Metóda `getDeviceSpeeds()` preto každú rýchlosť z tohoto zoznamu skúsi nastaviť na port a vracia zoznam naplnený len rýchlosťami, ktoré sa skutočne na zariadení sériového portu dajú nastaviť.

Metóda `getSpeeds()` vráti zoznam textových reťazcov s popisom možných komunikačných rýchlostí pre výpis používateľovi. Využíva prevodný zoznam vrátený metódou `getDeviceSpeeds()` a z neho čerpá textové reťazce podporovaných komunikačných rýchlostí.

Metóda `open()` volá metódu `openPort()`. `openPort()` otvorí, ale nekonfiguruje, sériový port. `openPort()` je volaná aj pri získavaní zoznamu podporovaných rýchlostí portu, preto je zdieľaná funkcionalita umiestnená v zvláštnej metóde `openPort()`. `open()` ďalej nastaví parametre komunikácie. Volá metódu `findSpeed()` zisťujúci, či je používateľom vyžiadaná rýchlosť komunikácie v zozname rýchlostí podporovaných sériovým portom. Metódou `setSpeed()` nastaví komunikačnú rýchlosť otvorenému portu.

Sériová komunikácia má v unixových systémoch väčší význam ako v systémoch MS Windows. Cez sériovú linku boli k počítaču pripojené textové terminály, cez ktoré so systémom interagovali používatelia. Preto je v konfigurácii sériového portu veľa nastavení týkajúcich sa spracovávania a transformácie znakov posielaných po sériovej linke a riadenia toku. Pri vývoji som narazil na poškodzovanie znakov (bajtov z pamäte FLASH MCU) posielaných po linke. Zistil som, že k poškodeniu dochádzalo na znakoch odriadkovania. Po vypnutí nepotrebných znakových transformácií a spracovávaní znakov pre kontrolu toku na linke, bol problém vyriešený.

Metóda `close()` zatvorí otvorený sériový port.

`writeSingle()` implementuje platformovo závislú funkcionalitu pre zápis dát do sériového portu. `readSingle()` implementuje funkcionalitu pre čítanie z portu. `readSingle()` sa stará aj o časové limity a časovanie blokujúceho čítania z portu.

4.4.5 Operácie s mikrokontrolérom

Trieda `CMcu` reprezentuje MCU. Jej verejné metódy sú `ident()`, `erase()`, `read()` a `write()`. Vytvára spoločnú abstrakciu nad rôznymi modelmi MCU. Informácie pre konkrétny model MCU získava z triedy implementujúcej rozhranie IMcuSpecifics deklarované v `McuSpecifics.hpp`.

V konštruktore triedy `CMcu` sa na sériové rozhranie odošle nulový bajt. Bajt odpovede sa číta dovtedy, dokým sa nenarazí na platný identifikačný bajt bootstrap zavádzača, alebo príkazovej slučky firmware tretej úrovne. Maximálne sa však prečíta 128 bajtov. Tento prístup je ochranou pred nevyčisteným prijímacím bufferom sériového rozhrania v operačnom systéme. Nájdenie platného identifikačného bajtu je zároveň synchronizáciou komunikácie.

Ak bol prijatý identifikačný bajt bootstrap zavádzača, MCU sa pošle 32 bajtov firmware prvej úrovne a následne sa pošle kód firmware druhej úrovne, ktorý prečíta a odošle hodnoty identifikačných registrov MCU. Tie sa príjmu a podľa nich sa vytvorí objekt triedy implementujúci rozhranie `IMcuSpecifics`. Potom sa MCU pošle firmware tretej úrovne a prijatím nulového bajtu sa počká na dokončenie jeho inicializácie.

Ak bol prijatý identifikačný bajt príkazovej slučky firmware tretej úrovne, znamená to, že na sériovej linke je pripojený MCU, ktorý už vykonáva kód firmware tretej úrovne. Získajú sa len identifikačné registre a podľa nich sa vytvorí objekt triedy implementujúci rozhranie `IMcuSpecifics`. Informácie pre ST10F168 zapúzdruje trieda `CMcuSt10f168` v súbore `McuSt10f168/McuSt10f168.cpp` a pre ST10F269 ich zapúzdruje trieda `CMcuSt10f269` v súbore `McuSt10f269/McuSt10f269.cpp`.

Objekt triedy `IMcuSpecifics` poskytuje prístup k špecifickým informáciam o MCU bez toho, aby musel volajúci poznať, o aký model MCU sa jedná. Týmito špecifickými informáciami sú

- označenie MCU,
- dáta firmware tretej úrovne,
- veľkosti a počet blokov FLASH pamäte,
- celková veľkosť FLASH pamäte,
- časový limit pre čakanie na zmazanie pamäte,
- konfiguračné informácie,
- a mapovanie chybových kódov vrátených MCU na reťazce s popisom chyby pre výpis používateľovi.

Metóda `erase()` je preťažená a je možné ju volať tromi spôsobmi. So zoznamom blokov pre zmazanie zmaže dané bloky. Číslo každého bloku musí byť platné. Zostaví sa bitová maska popisujúca mazané bloky, príkazovej slučke sa odošle kód operácie mazania, konfiguračné dáta pre MCU a bitová maska. Potom sa čaká na prijatie výsledku operácie. Ak nebude prijatý v časovom limite pre mazanie FLASH, program skončí chybou.

Volanie `erase()` s dvomi argumentami maže zadaný rozsah pamäte od štartovacej adresy (vrátane) po koncovú adresu (vrátane). Metóda podľa informácií o blokoch FLASH pamäte a mazaného rozsahu zostaví zoznam blokov pre zmazanie a zavolá svoju preťaženú verziu, prijímajúcu zoznam blokov pre zmazanie.

Volanie `erase()` bez argumentov maže celú pamäť. Pošle príkazovej slučke kód operácie, konfiguračné informácie a počká na prijatie výsledku.

Pri zápise do FLASH v metóde `write()` sa najprv kontroluje veľkosť dát pre zápis. Potom sa zistí, či je to nepárny počet bajtov a ak áno, dáta sa pri posielaní MCU na konci ešte o jeden bajt doplnia tak, aby firmware tretej úrovne prijal párny počet bajtov. To je nutné preto, že MCU zapisuje dáta po slovách, nie po bajtoch. Príkazovej slučke vo firmware sa pošle kód operácie, konfiguračné dáta a 32-bitové číslo určujúce počet bajtov pre zápis. Podľa toho firmware vie, ktoré slovo pre zápis do pamäte je posledné a kedy má skončiť. Dáta sa posielajú po 1024 bajtových blokoch. Po každom bloku, alebo po odoslaní posledného slova dát pre zápis, software prijme stavový kód a počet bajtov, ktoré ešte zostáva do FLASH zapísať. Software sčíta svoj počet odoslaných bajtov s prijatou hodnotou zostávajúcich bajtov. Ten sa musí rovnať veľkosti zapisovaných dát s prihliadnutím na výplňový bajt. Tak sa komunikácia synchronizuje. Ak stavový kód signalizuje chybu, zápis skončí.

Čítanie FLASH metódou `read()` pracuje analogicky ako zápis. Prebehne kontrola zadanej veľkosti dát pre prečítanie. Ak je to nepárny počet bajtov, prečíta sa ešte jeden bajt naviac a pred vrátením dát volajúcemu sa výplňový bajt zahodí. Príkazovej slučke sa pošle kód operácie, konfiguračné dáta a počet bajtov, ktoré ma MCU od adresy 0 odoslať. Synchronizácia je rovnaká ako pri zápise.

4.4.6 Spracovávanie chýb a výpisov

Pre distribúciu informácie o chybe programu som použil systém výnimiek. Okrem chybovej správy som ale potreboval v objekte použitom pre výnimky uložiť aj návratový kód, s ktorým program končí. Preto som si vytvoril vlastnú triedu `CExitException`, ktorá dedí od triedy `std::runtime_error`. Deklaroval som si vlastný konštruktor, ktorý okrem reťazca so správou prijíma návratový kód programu, a metódu `getReturnValue()` pre získanie tejto hodnoty z objektu triedy. Trieda je deklarovaná v súboroch `ExitException.cpp` a `ExitException.hpp`.

Pre možnosť lepšieho spracovávania výpisov programu som namiesto priameho vypisovania na štandardný výstup alebo štandardný chybový výstup, zvolil ich presmerovanie do jedného miesta programu. Týmto miestom sú statické metódy triedy `CLogger`. Tento prístup tiež umožňuje konfigurovať voliteľné vypisovanie informačných hlásení a vypisované chybové, varovné a informačné správy spoločne formátovať. Zjednodušujú sa úpravy formátu výpisov, pretože sa formátovanie správy sústreďuje na jednom mieste.

Metóda `error()` prijíma chybovú správu a návratový kód programu. Na štandardný chybový výstup vypíše správu s predponou `ERROR:` a vyhodí vo výnimke objekt triedy `CExitException`. Metódu `error()` ukazuje výpis 4.6.

Zdrojový kód 4.6: Metóda error() triedy CLogger

```
void
CLogger::error(const string & msg, int returnValue)
{
    cerr << "ERROR:  " << msg << endl;
    throw CExitException(msg, returnValue);
}
```

Program sa ukončuje na chyby volaním metódy `error()`. Príklad je na 4.7.

Zdrojový kód 4.7: Ukončovanie programu pri chybe

```
CLogger::error("Cannot allocate memory", EXIT_MCU);
```

Metóda `warning()` prijíma varovnú správu a vypíše ju na štandardný chybový výstup s predponou `WARNING:`.

Metóda `info()` prijíma informačnú správu. Kontroluje príznak výrečnosti programu z používateľskej konfigurácie a ak má byť program výrečný, vypíše správu s predponou `INFO:` na štandardný výstup. Inak k výpisu správy nedôjde. Vypisovanie informačných hlásení sa ovláda metódou `setLogInfo()`.

Vypisovanie priebehu dátových operácií s FLASH pamäťou implementuje metóda `progress()`. Prijíma dve hodnoty. Prvá je aktuálny počet bajtov, druhá je celkový počet bajtov pre zápis alebo pre čítanie. Podľa toho metóda spočíta percentuálny postup operácie a túto informáciu vypíše na štandardný výstup.

Poslednou metódou triedy je pomocná metóda `decToHex()` prijímajúca celé číslo a vracajúca reťazec s číslom v šestnástkovej sústave s predponou `0x`. Súvisí s textovým výstupom programu a preto je zaradená v triede `CLogger`.

Program rozlišuje rôzne návratové kódy. Boli volené tak, aby popisovali časť programu, kde chyba nastala. Aby bolo možné hodnoty ľahko spravovať, sú definované direktívami preprocesoru v hlavičkovom súbore `ExitCodes.hpp`. Každá časť programu, rozdelenému vzhľadom na chybové oblasti, musí mať unikátny návratový kód. Návratové kódy sú celé čísla od jedna a vyššie.

4.5 Rozloženie pamäte mikrokontroléra

Adresa firmware prvej úrovne je pevne určená bootstrap zavádzačom. Je to adresa 0xFA40. Na túto adresu sa uloží 32 bajtov kódu firmware prvej úrovne prijatých bootstrap zavádzačom.

Dáta som sa rozhodol umiestniť do rozšírenej RAM XRAM1. V internej RAM je od adresy 0xFA40 uložený firmware prvej úrovne a preto najväčšia súvislá oblasť v IRAM je od adresy 0xF600 do adresy 0xFA3F (vrátane), tj. 1088 bajtov. V XRAM1 je k dispozícii 2048 bajtová oblasť. Pri vývoji

firmware som najprv používal IRAM oblasť. Ako sa ale postupne zväčšovala veľkosť firmware tretej úrovne, presunul som kód z IRAM do XRAM1.

Po tomto presune som ale narazil na chybu v MCU ST10F269, ktorá viedla k zničeniu MCU. Keď som spustil operáciu mazania, MCU mi po resete prestal odpovedať na nulový bajt a nedal sa aktivovať mód bootstrap zavádzača. Manipulovať s FLASH pamäťou už nebolo možné. Myslel som si, že je to chyba elektroniky vývojovej dosky s ST10F269. Kontaktoval som sponzora a programovanie sme vyskúšali vo firme AŽD Praha s.r.o s ich technickými prostriedkami a programom ST10Flasher. Programovanie tiež nefungovalo. Od sponzora som dostal novú dosku osadenú MCU ST10F269. Po pripojení dosky k mojim vývojovým prostriedkom a spustení mazania pamäte sa chyba opakovala znovu. MCU prestal odpovedať na nulový bajt. Až potom som začal prehľadávať informačné zdoje a na komunitnom fóre [32] som našiel príspevok v ktorom bola vysvetlená príčina. Citujem text príspevku (preklad vlastný):

> *„Nedávno bol objavený efekt na ST10F269, ktorý by mohol spôsobovať problém, ktorý máte. Pozri popis nižšie.*
> ------
> *Keď sú splnené všetky nasledujúce podmienky:*
> *– ST10F269 je v režime bootstrap zavádzača*
> *– Program beží z XRAM alebo externej pamäte*
> *– Flash je mapovaná v segmente 0 (ROMS1, bit12 registru SYSCON je nulový)*
> *– Je spustený príkaz mazania alebo zápisu Flash*
> *– Inštrukcia JUMP, CALL, alebo RET (ľubovoľná z každého typu) je vykonaná bezprostredne za posledným cyklom príkazu*
> *Jadro ST10 pošle chybný signál do flash pamäte, výsledkom čoho je výber TestFlash*
> *namiesto používateľskej flash. Za týchto okolností sa TestFlash poškodí (vrátane bootstrap kódu).*
> *Riešenie:*
> *Vložte inštrukciu NOP medzi posledný cyklus príkazu Flash a ktorúkoľvek z inštrukcií JUMP, CALL alebo RET.*
> ----------
> *Mohli by ste prosím skontrolovať váš software v súvislosti s týmto problémom?“*

Ako riešenie som porušil dve z vymenovaných podmienok. FLASH pamäť na ST10F269 som premapoval do segmentu 1 a pridal dve inštrukcie NOP za posledný cyklus každého príkazu pre kontrolér FLASH. Tým sa problém s programovaním ST10F269 vyriešil.

Aj keď sa mi časom podarilo zmenšiť veľkosť firmware tretej úrovne pri plnom zachovaní funkčnosti, usúdil som, že by v budúcnosti mohlo byť nutné

rozšíriť firmware a preto som kód ponechal v XRAM1, aj keby sa zmestil do IRAM.

Posledných 42 bajtov XRAM1 som vyhradil pre tabuľky statických dát. Veľkosť som určil ručným spočítaním podľa najväčšej z tabuliek vo firmware tretej úrovne.

O rozloženie oblastí zásobníka a registrových bánk som sa začal zaujímať až pri nefungujúcich STEAK operáciach v ST10F168. To sa prejavovalo tak, že keď som spustil operáciu mazania alebo zápisu FLASH, MCU prestal odpovedať. Ak som pre danú operáciu nastavil nesprávne argumenty, STEAK program správne detekoval chybné hodnoty a k žiadnemu zaseknutiu nedochádzalo. Pri hľadaní riešenia som na [33] našiel zdrojové kódy firmware veľmi starej vývojovej verzie ST10Flasher. Ďalším zdrojom z ktorého som čerpal bol dokument [34], v ktorom sú podrobnejšie analyzované STEAK programy a obsahuje zmienku, že výrobcom udávaná minimálna veľkosť zásobníka 100 bajtov pre STEAK nedostačuje a je potrebných aspoň 130 bajtov.

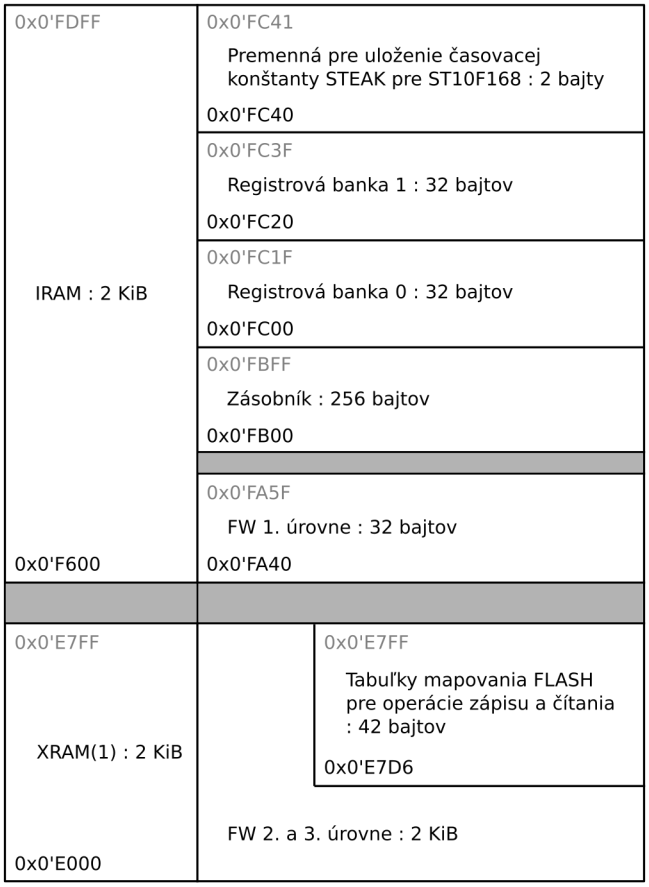
Tieto informácie ma naviedli k príčine nefungujúcich STEAK operácií. Chyba bola v nesprávnom rozložení a veľkosti zásobníka a prvej registrovej banky. Predvolená lokácia prvého bajtu zásobníka je na adrese 0xFA3F a zásobník rastie smerom k nižším adresám. Predvolené umiestnenie banky registrov je od adresy 0xFA00 vyššie.

Keď sa spustil program STEAK algoritmov, začal plniť zásobník a tak si prepísal niektoré GPR, ktoré sám používal a došlo k nedefinovanému chovaniu programu. Pri predaní nesprávnych argumentov pre STEAK všetko fungovalo preto, že sa ešte nestihol zásobník naplniť tak, aby prepísal dôležité registre. Ak mal ale STEAK správne argumenty, pokračoval vo svojom behu ďalej a naplnenie zásobníka sa zvýšilo.

Veľkosť zásobníka som preto nastavil na 256 bajtov a jeho bázovú adresu zvolil 0xFC00. Adresy pre dáta zásobníka sú v IRAM na adresách od 0xFB00 do 0xFBFF.

Nad zásobník som umiestnil dve registrové banky. Predvolenú registrovú banku 0 od adresy 0xFC00 (vrátane) a registrovú banku 1 od adresy 0xFC20. Druhú adresovú banku som využil v podprogramoch, ktoré manipulujú s príliš mnoho univerzálnymi registrami. Je jednoduchšie a rýchlejšie prepnúť používanú banku registrov pri vstupe do podprogramu a späť pri jeho opustení, ako ukladať na zásobník GPR inštrukciou PUSH a potom ich vyberať inštrukciou POP.

Banka registrov sa mení dvomi inštrukciami. Nastavením novej hodnoty registru CP inštrukciou SCXT a čakaním inštrukciou NOP na účinok tejto zmeny. Pri použití niektorého z univerzálnych registrov hneď za inštrukciou SCXT pre zmenu CP vznikajú chyby, ktoré sa s nedostatkom pozornosti veľmi ťažko dohľadávajú. Pri vývoji sa mi stalo, že som použil GPR hneď za inštrukciou SCXT. Kompilátor takýto druh chýb nedetekuje a pri rýchlej kontrole zdrojového kódu sa javí, akoby sa nastavovala hodnota registru už v novej



Obr. 4.8: Rozloženie pamäte MCU

banke. Kvôli prúdovému spracovaniu inštrukcií sa ale mení hodnota registru ešte v starej registrovej banke.

Rozloženie pamäťového priestoru je na obrázku 4.8.

4.6 Firmware

Firmware pre MCU som vyvíjal v prostredí Keil µVision 5. Pre každú zo štyroch častí firmware som založil samostatný projekt.

V prostredí µVision som používal pre spustenie procesu kompilácie a linkovania klávesovú skratku. Tá spúšťa tento proces pre projekt označený ako *aktívny* nie ten, ktorého zdrojový súbor je zobrazený v editačnom okne. Dovtedy, než som začal venovať väčšiu pozornosť, ktorý projekt je takto označený sa mi občas stávalo, že som robil zmeny v zdrojovom kóde firmware a premýšľal nad tým, prečo sa táto zmena neprejavuje na funkčnosti.

Výstupom procesu kompilácie a linkovania je binárny súbor, pre účely debuggeru a pre nahrávanie do hardwaru podporovaného prostredím µVision. Tento súbor som nemohol zapísať do RAM MCU, pretože obsahoval okrem kódu programu mnoho iných informácií. Tento problém som vyriešil aktiváciou generovania výstupného súboru vo formáte Intel Hex v nastaveniach projektu a jeho prevodom na binárne dáta. V nastaveniach projektu som našiel možnosť spúšťania používateľom definovaných príkazov po kompilácii a linkovaní. Definoval som si príkaz, ktorý prevedie Intel Hex súbor na binárny. Tento binárny súbor už obsahoval len kód programu. Pre konverziu som použil nástroj [35] pod licenciou BSD pre MS Windows. V systéme GNU/Linux som ho spúšťal tiež pomocou Wine.

4.6.1 Spoločné časti zdrojového kódu

Pre zdieľanie spoločných častí zdrojového kódu som použil direktívu `$INCLUDE`, ktorá na svoje miesto vloží text súboru, na ktorý odkazuje.

Do zdrojového kódu každej časti firmware bol vložený súbor `REG167.INC` s definíciami bitov a registrov MCU, dodávaný s vývojovým prostredím µVision. Tak som mohol namiesto bitových adries a adries registrov používať ich názvy.

V súbore `COMMON.INC` som uviedol len definície registrov, ktoré neboli obsiahnuté v `REG167.INC` a konštanty pre adresy oblastí RAM (program, premenné, zásobník, registrové banky) a hodnoty príkazov komunikačného protokolu.

Do súboru `SUBROUTINES.INC` som vložil podprogramy používané vo viac ako dvoch iných zdrojových súboroch. Obsahuje podprogramy pre odoslanie hodnôt identifikačných registrov, kontrolu počítadiel dát pre operácie čítania a zápisu, pre bezpečný príjem slova a dvojslova, cez sériové rozhranie a pre vysielanie a príjem bajtov a slov cez sériové rozhranie.

V súbore `SHELL.INC` je spoločná časť pre firmware tretej úrovne. Je to príkazová slučka, kde sa prijíma kód operácie vyžadovanej od software v PC. Po prijatí bajtu sa určí, či je to nulový bajt.

Ak áno, znamená to, že software odosiela nulový bajt s cieľom dostať bajt 0xD5 ako odpoveď od bootstrap zavádzača. V príkazovej slučke sa ako odpoveď pošle bajt 0xAB. Podľa toho software zistí, že v MCU už beží firmware tretej úrovne a že môže rovno posielať kód vyžadovanej operácie a nemusí nahrávať firmware prvej a druhej úrovne.

Ak príkazová slučka prijme nenulový bajt, znamená to, že software v PC už vie, že v MCU beží firmware tretej úrovne a bajt odoslal bezpečným spôsobom. To znamená, že software očakáva kontrolné prijatie bajtu odoslaného do MCU a ak obdrží hodnotu, ktorú poslal, odošle potvrdenie správneho ukončenia bezpečného odoslania bajtu. Tak sa znižuje riziko spustenia nevyžiadanej operácie rušením na sériovej linke.

4.6.2 Firmware prvej a druhej úrovne

Firmware prvej úrovne je prijímaný bootstrap zavádzačom, zapísaný do RAM na pevne definovanú adresu 0xFA40 a je mu predané riadenie. Najprv aktivuje oblasť XRAM1 a potom v slučke prijíma bajty firmware vyšších úrovní a ukladá ich na od adresy 0xE000 (vrátane) po adresu 0xE7FF (vrátane). Počet prijatých bajtov určuje práve z rozsahu týchto adries. Prijme teda vždy 2048 bajtov. Po prijatí všetkých týchto bajtov predá riadenie na ich začiatok v RAM, tj. na adresu 0xE000.

Firmware druhej úrovne je prijatý, uložený do RAM a spustený firmware prvej úrovne. Jeho jediným účelom je odoslanie hodnôt identifikačných registrov IDMANUF a IDCHIP cez sériové rozhranie. Podľa toho software v PC vie, aký model MCU je pripojený na sériovej linke a aký firmware tretej úrovne má zvoliť pre odoslanie do MCU. Potom skočí na adresu 0xFA40, čo je adresa firmware prvej úrovne, ktorý zasa prijme 2048 bajtov firmware tretej úrovne, uloží ich na adresu 0xE000 a spustí. Tým prepíše kód firmware druhej úrovne.

4.6.3 Firmware tretej úrovne pre ST10F168

Na začiatku zdrojového súboru sú vložené definície konštánt, bitov a registrov MCU. Po nich je definovaná adresa v RAM, použitá pre premennú pre uloženie časovacej konštanty, ktorá je predávaná STEAK v R4. Premenná je umiestnená hneď za registrovou bankou 1. Potom nasleduje dátová sekcia s tabuľkami statických dát pre ovládanie programu pri čítaní a zápise.

Zvyšok je tvorený sekciou programového kódu, začínajúceho od adresy 0xE000 v XRAM1. Na začiatku programu dôjde k inicializácii. Aj keď je inicializácia rovnaká pre ST10F168 aj ST10F269, rozhodol som sa túto časť nezjednocovať s ohľadom na možné budúce pridanie podpory ďalších MCU, kde by sa inicializácia líšila. V inicializácii sa nastaví použitie registrovej banky 0, zapne sa detekcia preplnenia prijímacieho bufferu pre sériový kanál, veľkosť zásobníka a hranice jeho pretečenia a podtečenia. Priradia sa hodnoty DPP registrom tak, aby každá dlhá adresa zodpovedala rovnakej fyzickej. Po ukončení inicializácie sa pošle software nulový bajt. Potom sa vloží direktívou `$INCLUDE (SHELL.INC)` príkazová slučka, z ktorej sú volané podprogramy operácií.

Pri chybe sa odošle jej kód a riadenie sa vracia do príkazovej slučky. Chyba môže nastať napr. pri bezpečnom príjme dát, alebo chybe STEAK.

Bezpečný príjem dát je použitý len pre dáta o veľkosti jednotiek bajtov. Pri operácii zápisu sa dáta pre zápis prijímajú normálne, aby bol zápis čo najrýchlejší. V prípade potreby si môže používateľ programovacieho nástroja vyžiadať kontrolu zapísaných dát.

Na začiatku každej operácie sa prijmú konfiguračné dáta. Firmware tretej úrovne pre každý MCU musí prijímať rovnaké množstvo dát. Ako ich ale spracuje, je len na ňom. ST10F168 prijme jedno slovo časovacej konštanty

pre STEAK. ST10F269 tieto dáta nevyužíva. Toto je preto, aby mohlo byť rozhranie firmware tretej úrovne pre každý MCU rovnaké.

Podprogram `ERASE_CHIP` maže celú FLASH pamäť. Zmení registrovú banku, prijme časovaciu konštantu. Pretože ST10F168 nemá možnosť zmazať celú FLASH jednou STEAK operáciou, volá sa `ERASE_BLOCKS` s nastavením pre zmazanie všetkých blokov (cieľ volania je až za príjmom konfiguračných dát a masky blokov). Registrová banka sa zmení späť na banku 0 a riadenie sa z podprogramu vráti volajúcemu.

Podprogram `ERASE_BLOCKS` tiež používa druhú registrovú banku. Prijme konfiguračné dáta a masku blokov pre zmazanie. Maska obsahuje log. 1 na bitovej pozícii, ktorej číslo je číslo mazaného bloku. Nastavia sa parametre v registroch pre určenie STEAK operácie. V slučke sa pomocou bitovej rotácie prejdú významné bity masky a tam, kde je v maske nájdená log. 1, zavolá sa odomykacia sekvencia pre začatie mazania príslušného bloku. Po skončení mazania blokov sa odošle slovo signalizujúce výsledok operácie.

Podprogram odomykacej sekvencie `UNLOCK_SEQUENCE` nasleduje po vložení spoločných podprogramov direktívou `$INCLUDE (SUBROUTINES.INC)`. Hodnoty používaných registrov R0 až R4 a R6 sa uložia inštrukciou PUSH a obnovia inštrukciou POP. Nepoužil som prepínanie registrových bank, pretože by som musel z prepínanej banky prekopírovať hodnoty registrov do novej banky a pre každé kopírovanie by bolo nutné použiť dve inštrukcie, pretože ST10F nemá inštrukciu pre presun dát z pamäte do pamäte. Po uložení obsahu registrov sa register R4 naplní časovacou konštantou a operácia sa spustí sekvenciou priameho a nepriameho zápisu na aktívnu adresu v pamäťovom priestore FLASH. Len pri týchto dvoch zápisoch používa mechanizmus prekrytia DPP registrov inštrukciou EXTS. Pre niekoľko málo inštrukcií je to jednoduchšie a rýchlejšie, ako nastavovať hodnoty DPP registrov. Dve nasledujúce NOP inštrukcie sú nutné pre zabránenie možným komplikáciam s prúdovým spracovaním inštrukcií v podprogramoch STEAK. Potom sa preberie návratová hodnota STEAK do registru R14 pre informovanie volajúceho a obnovia sa uložené registre.

Podprogram `READ` používa druhú registrovú banku. Prijme konfiguračné dáta a počet bajtov pre prečítanie od adresy 0 vo FLASH. Nastaví počítadlo bajtov bloku pre synchronizáciu komunikácie a bázovú adresu tabuľky s dátami popisujúcimi mapovanie FLASH do adresového priestoru. Obsah FLASH sa číta práve cez adresový priestor. Jeden prvok tabuľky sú tri slová. Prvé slovo popisuje segment, druhé slovo je štartovacia adresa v segmente a tretie slovo je počet slov, ktoré sa nachádzajú v danom segmente. FLASH v ST10F168 má tri bloky, ale je mapovaná do šiestich oblastí delených hranicami segmentov. Obsah tabuľky mapovania FLASH je ukázaný vo výpise 4.8.

Ak pri čítaní slov z FLASH prekročím hranicu oblasti popísanej v tabuľke, vyberie sa z tabuľky ďalší prvok, nastavia sa registre pre správne adresovanie a v čítaní sa pokračuje od začiatku ďalšej oblasti. Pri každom prečítanom a odoslanom slove sa zavolá podprogram `CHECK_COUNT`, ktorý zníži hodnotu počítadla celkovo prečítaných bajtov a hodnotu počítadla bajtov v 1024 baj-

Zdrojový kód 4.8: Definícia tabuľky mapovania FLASH

```
FLASH_MAPPING:
        DW 0,0,(32*1024)/2
        DW 1,8000h,(32*1024)/2
        DW 2,0000h,(64*1024)/2
        DW 3,0000h,(32*1024)/2
        DW 3,8000h,(32*1024)/2
        DW 4,0000h,(64*1024)/2
```

tovom bloku. Po prečítaní 1024 bloku alebo po prečítaní všetkých slov sa odošle obsah celkového počítadla pre synchronizáciu komunikácie. Návratová hodnota `CHECK_COUNT` určuje, či sa má vo vykonávaní čítania riadeného tabuľkou pokračovať, alebo už je hotovo. `READ` pre adresovanie používa mechanizmus prekrytia DPP registrov inštrukciou EXTS.

Zápis do pamäte realizovaný v podprograme `WRITE` je tiež riadený rovnakou tabuľkou a funguje analogicky. Prijme konfiguračné dáta a počet bajtov pre zápis od adresy 0 vo FLASH. Nastaví počítadlo bajtov bloku pre synchronizáciu komunikácie a bázovú adresu tabuľky s mapovaním FLASH. Prvé slovo vyberané z tabuľky sa upraví na príkaz pre STEAK, obsahujúci číslo segmentu v spodných štyroch bitoch (napr. pre číslo segmentu *2* má príkaz hodnotu 0x55A*2*). Zo sériovej linky sa prijme slovo a zavolaním odomykacej sekvencie sa spustí zápis slova do FLASH. Práca s tabuľkou a počítadlami pri priechode adresami FLASH je rovnaká ako pri čítaní. Na konci pošle podprogram cez sériové rozhranie kód popisujúci výsledok operácie. Tiež pri chybe príjmu slova, podprogram skončí.

4.6.4 Firmware tretej úrovne pre ST10F269

Firmware pre ST10F269 je veľmi podobný ako pre ST10F168. Dôvodom je, že majú prostredníctvom sériovej linky implementovať rovnaké rozhranie. Operácie s FLASH sú tiež riadené tabuľkou mapovania. FLASH pamäť ST10F269 má sedem blokov a je mapovaná do siedmich oblastí adresového priestoru delených hranicami segmentov.

V inicializácii FLASH je spodných 32 kB premapovaných do segmentu 1 kvôli chybe s vymazaním pamäte bootstrap zavádzača (4.5). Zvyšok inicializácie je rovnaký, ako pre ST10F168. Potom je do programu vložená príkazová slučka.

Na začiatku každej operácie sa prijmú konfiguračné dáta, ale firmware pre ST10F269 ich nevyužíva. Toto je nutné pre jednotnosť rozhrania firmware.

ST10F269 má už príkaz pre zmazanie celej FLASH pamäte. Podprogram `ERASE_CHIP` zmení registrovú banku, prijme časovaciu konštantu a vykoná príkazové cykly zápisom definovaných hodnôt na definované adresy, využívajúc

prekrytie DPP registrov. Počká na dokončenie mazania a odošle informáciu o výsledku.

Podprogram `ERASE_BLOCKS` zmení registrovú banku, prijme konfiguračné dáta a masku blokov pre zmazanie. Vykoná úvodné príkazové cykly pre spustenie mazania blokov FLASH. V slučke prejde masku a podľa nej pridá ešte ďalší cyklus popisujúci každý blok pre zmazanie. Počká na zmazanie blokov a odošle informáciu o výsledku. Príkazové cykly zaraďujúce do mazania konkrétne bloky musia zapisovať na platné adresy v danom bloku. Túto informáciu podprogram získava z tabuľky mapovania FLASH.

Čítanie pamäte v podprograme `READ` je až na reset kontroléru FLASH úplne rovnaké, ako podprogram `READ` pre ST10F168 a je parametrizované tabuľkou mapovania FLASH. Reset kontroléru som pridal pre istotu, aby bola pred zahájením čítania pamäť skutočne v móde čítania a nie napr. v čakaní na nedokončenú sériu príkazových cyklov. Príkaz *Reset* tiež nuluje príznaky chyby z minulých operácií.

Zápis v podprograme `WRITE` je veľmi podobný, ako pre ST10F168. Líši sa len spôsobom zápisu slova do FLASH (príkazové cykly namiesto STEAK) a pridaním čakania na dokončenie zápisu. Podprogram `CHECK_COUNT` zdieľa s firmware ST10F168 cez súbor `SUBROUTINES.INC`.

Na konci je podprogram `OP_WAIT` pre čakanie na výsledok operácií mazania a zápisu. Najprv som používal pre čakanie spôsob *Data polling*. Ten som vybral náhodne z dvojice *Data polling* a *Toggle bity*. Pre moje účely mali byť podľa [22] oba spôsoby ekvivalentné.

4.6.5 Chyby mikrokontroléra ST10F269

Pri vývoji a testovaní som však narazil na dva problémy so spôsobom *Data polling*. Ako riešenie som vyskúšal zmeniť spôsob detekcie dokončenia na *Toggle bity* a v oboch prípadoch to bolo úspešné.

Prvý problém sa objavil pri presune firmware používajúceho *Data polling* z IRAM do XRAM1. Pri zápise náhodných dát (zdrojom bolo linuxové zariadenie `/dev/urandom`) o veľkosti celej FLASH 256 KiB zápis končil chybou. ST10F269 nemá tak presné informovanie o chybe, ako STEAK na ST10F168, preto som nemal ako chybu presnejšie dohľadať. Skúšal som zápis na rýchlostiach 19 200 Bd, 115 200 Bd a 230 400 Bd. V štyroch pokusoch s rýchlosťou 230 400 Bd a pri aktivovanom vypisovaní percentuálnej informácie o postupe zápisu, zápis skončil postupne na 24 %, 33 %, 24 % a 42 %. Po skončení zápisu s chybou, MCU prestal na sériovom rozhraní odpovedať. Vždy bol teda nutný reset. Potom mi napadlo použiť detekciu pomocou *Toggle bitov* a problém sa vyriešil. Z toho dôvodu som túto chybu viac neanalyzoval.

Druhý problém sa objavil pri zápise všetkých slov FLASH s hodnotou 0x0000 a detektciou zmazania blokov spôsobom *Data polling*. Po takejto operácii zápisu a následnom spustení mazania ľubovoľnej množiny blokov, ktorá nemazala siedmy blok, mazanie blokov skončilo vypršaním časového limitu

a MCU prestal odpovedať. Po hardwarovom resete a teda aj novom nahraní firmware, MCU začal odpovedať, ale mazanie blokov bez siedmeho stále nefungovalo a MCU prestával odpovedať. Oprava nastala až vtedy, keď sa mi podarilo spustiť mazanie celej pamäte, alebo len siedmeho bloku.

Keď som ale vyskúšal zapísať celú FLASH náhodnými dátami, chyba sa nevyskytla. Postupom binárneho vyhľadávania som chybu v dátach pre zápis do FLASH zúžil na jediné slovo. Ak slovo na adrese nula v bloku sedem FLASH pamäte malo hodnotu 0x0000, chyba sa objavila. Pre testovanie som si vygeneroval náhodné dáta veľkosti celej FLASH, kde spomínané slovo v bloku sedem bolo nenulové. Potom som tieto dáta skopíroval a pozmenil len to konkrétne slovo na hodnotu 0x0000. Testovacie súbory sa teda líšili len v jedinom slove. Po zápise dát s nulovým slovom MCU úspešne vykonával všetky operácie čítania, mazania celej FLASH, mazania blokov zasahujúce siedmy blok a zápisu, pred ktorým sa mazala buď celá FLASH, alebo siedmy blok. To všetko až pokým sa nespustilo mazanie, ktoré nezasahovalo siedmy blok a MCU prestal odpovedať. Pri zápise dát s nenulovým slovom sa chyba neobjavovala.

Použité testovacie súbory je možné nájsť v prílohe B v adresári `src/error-data/`. Súbor `ok.bin` je ten, s ktorým programovanie prebiehalo v poriadku. Súbor `bad.bin` sa od súboru `ok.bin` líši v jednom nulovom slove a s ním sa chyba prejavovala.

Pretože som si na predchádzajúcej popisovanej chybe overil, že riešením by mohlo byť použiť namiesto *Data polling Toggle bity* vyskúšal som to a problém sa vyriešil. Odvtedy som túto chybu viac neanalyzoval.

4.7 Rozšírenie systému pre zostavovanie software

Počas práce na programe som postupne pridával zdrojové súbory do projektu. Pri každej zmene som musel manuálne modifikovať Makefile súbor a zohľadňovať závislosti zdrojových súborov. Stávalo sa mi, že aj keď som Makefile súbor modifikoval tak, aby bolo možné výsledný spustiteľný súbor zostaviť, tak som opomenul niektoré závislosti. Pri modifikácii súborov a následnom spustení príkazu `make` s cieľom zostaviť spustiteľný súbor programu sa v tom lepšom prípade program nezostavil kvôli detekovanej chybe a v tom horšom prípade, kedy sa nezmenili signatúry funkcií sa program zostavil z objektových súborov rôznych verzií a chyba nebola detekovaná.

Preto som sa rozhodol použiť program, ktorý by Makefile súbory generoval automaticky. Takýto program preskenuje len uvedené zdrojové `.cpp` súbory, zohľadňujúc aj hlavičkové súbory, sám zistí závislosti a vygeneruje korektné Makefile súbory.

Zvolil som si program CMake, pretože sa jedná o rozšírený multiplatformný program s obsiahlou dokumentáciou. Naviac má zabudovanú jednoduchú podporu pre testovanie software, ktorú som sa rozhodol tiež využiť pre testovanie programovacieho nástroja. Domovská stránka CMake je na [36].

CMake spracováva konfiguráciu v jeho jednoduchom skriptovacom jazyku v súboroch `CMakeLists.txt`. Pracuje tak, že podľa svojej konfigurácie a testov systému (aká je platforma, aké sú prítomné vývojové nástroje a pod.), na ktorom je spustený, vygeneruje súbor pre natívny systém pre zostavenie software. Dokáže generovať unixové Makefile súbory, MinGW Makefile súbory, súbory pre MS Visual Studio a Xcode a ďalšie.

4.8 CMake

V tejto podkapitole popíšem konfiguračné súbory CMake, ktoré sa týkajú zostavovania spustiteľného súboru aplikácie. CMake súbory s konfiguráciou testov CTest popíšem v kapitole o testovaní programovacieho nástroja.

Hlavný súbor pre CMake je uložený v koreňovom adresári so zdrojovými kódmi programovacieho nástroja. Najprv sa definuje minimálna verzia CMake, ktorá môže byť použitá pre spracovanie konfiguračného súboru. Ďalej sa definuje názov projektu, jeho verzia a pridajú sa nastavenia kompilátora. Názov a verzia sú potom prístupné v premenných pevných názvov a môžu byť v konfiguračnom súbore ďalej použité. Nasledujú definície vlastných premenných s cestami k adresáru s projektami Keil µVision a k cestám k jednotlivým binárnym súborom firmware spoločným pre oba podporované modely MCU (firmware prvej a druhej úrovne).

Aby som mohol používať hodnoty z CMake súboru v zdrojovom kóde jazyka C++, je potrebné vytvoriť definície C++ preprocesoru. Definuje sa názov programu a verzia. Tie budú použité vo výpise informácií o aplikácii.

Potom sa detekuje platforma, na ktorej prebieha kompilácia. CMake automaticky definuje premenné príslušné k jednotlivým kategóriám operačných systémov na hodnotu `true` alebo `false`. Platforma MS Windows sa detekuje cez premennú `WIN32`. Ak je pravdivá, pridá sa definícia C++ preprocesoru s názvom platformy `WIN32` a názov predvoleného sériového portu. Ak sa nejedná o platformu Windows, pridajú sa podobné definície, ale pre unixové systémy.

Ďalej inštruujem CMake, aby zostúpil do adresára pre program XD aby sa zostavil najprv ten. V adresári `utils/xd/` je umiestnený súbor `CMakeLists.txt` s informáciami pre zostavenie programu XD. Určuje sa v ňom minimálna verzia CMake, názov projektu, parametre kompilátora a z akých zdrojových súborov sa zostaví spustiteľný binárny súbor programu XD.

V hlavnom konfiguračnom súbore sa potom pokračuje pridaním cesty, kde sa budú hľadať súbory, vložené do zdrojových súborov direktívou `#include` preprocesoru. To je nutné, pretože hlavičkové súbory generované XD sú umiestnené v adresári, kde prebieha zostavovanie programu. Inak by ich preprocesor nenašiel.

Následne si definujem pomocnú funkciu `generate_headers_from_binaries()` pre generovanie hlavičkových súborov s firmware. Funkcia sama prejde zoznam

súborov pre spracovanie a pre každý spustí program XD so správnymi argumentami. Základom je CMake funkcia `add_custom_command()`, ktorá definuje, aké súbory príkaz vytvára, na akých súboroch závisí a samotný príkaz. Informácie o závislostiach a výstupoch CMake používa pre korektné zostavovanie aplikácie

Hneď za definíciou funkcie dôjde k volaniu definovanej funkcie a vygenerovaniu hlavičkových súborov z binárnych súborov firmware prvej a druhej úrovne.

Potom zaistím vygenerovanie hlavičkového súboru s textom vstavanej nápovedy k programu. Analogicky ako pri generovaní hlavičkových súborov s firmware, použijem program XD v CMake funkcii `add_custom_command()`.

Pre úspešné vyhľadanie hlavičkových súborov ešte pridám koreňový adresár zdrojového stromu a jeho podadresáre `McuSt10f168`, `McuSt10f269` a `SerialPort`.

Vo vymenovaných podadresároch sa nachádzajú časti programu. Pre ich zaradenie do výsledného spustiteľného súboru som sa rozhodol najprv ich zostaviť ako statické knižnice a potom prilinkovať k hlavnému programu. Funkciou `add_subdirectory()` zostúpim do každého z týchto troch adresárov a spracujem súbor `CMakeLists.txt`.

Statické knižnice s podporou pre konkrétny model MCU sa zostavia podobne. V `CMakeLists.txt` nastavím cesty k súborom firmware tretej úrovne pre MCU, pridám nutné adresáre pre hľadanie hlavičkových súborov, vygenerujem hlavičkové súbory s firmware a pridám definíciu vytváranej statickej knižnice (jej názov a zdrojové a hlavičkové súbory, z ktorých pozostáva). Na záver ešte pridám nastavenia pre C++ kompilátor.

V `CMakeLists.txt` v adresári `SerialPort` využívam informáciu o platforme zo CMake premennej `WIN32`. Ak je pravdivá, zostavím statickú knižnicu s operáciami so sériovým portom pre platformu Windows. V opačnom prípade zostavím knižnicu s podporou pre unixové systémy. Na konci tiež pridám nastavenia C++ kompilátora.

Ďalšie je zostavenie hlavného programu. Vo funkcii `add_executable()` vymenujem zdrojové a hlavičkové súbory potrebné pre zostavenie hlavného programu `main`. Potom prilinkujem statické knižnice.

Nakoniec ešte definuje nastavenia C++ kompilátora, ktoré sa majú použiť pri zostavovaní hlavného programu.

4.9 Zostavenie spustiteľného súboru aplikácie

Zostavenie spustiteľného súboru aplikácie sa delí na dve časti:

1. zostavenie binárnych súborov firmware vo vývojovom prostredí Keil µVision,

2. zostavenie spustiteľného súboru programovacieho nástroja.

Zdrojový kód 4.9: Zostavenie spustiteľného súboru aplikácie pomocou CMake

```
/src/impl> mkdir build
/src/impl/build> cd build
/src/impl/build> cmake ..
-- The C compiler identification is GNU 5.3.0
-- The CXX compiler identification is GNU 5.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
...
-- Configuring done
-- Generating done
-- Build files have been written to: /src/impl/build
/src/impl/build> make
Scanning dependencies of target SerialPort
[  4%] Building CXX object SerialPort/CMakeFiles/\
SerialPort.dir/SerialPortFactory.cpp.o
[  9%] Building CXX object SerialPort/CMakeFiles/\
SerialPort.dir/SerialPort.cpp.o
...
[ 95%] Building CXX object CMakeFiles/main.dir/main.cpp.o
[100%] Linking CXX executable main
[100%] Built target main
/src/impl/build> ./main version
Main 1.0.0
/src/impl/build>
```

Súbory firmware som mohol zostaviť len v prostredí, kde sa dá prevádzkovať vývojové prostredie Keil µVision.

Zostavenie spustiteľného súboru aplikácie predpokladá existenciu binárnych súborov firmware. Pomocou CMake sa na unixových systémoch spustiteľný súbor zostaví v krokoch:

1. vytvorenie adresára pre zostavenie,
2. prechod do adresára pre zostavenie,
3. spustenie príkazu `cmake` s uvedenou cestou ku koreňovému adresáru zdrojového stromu aplikácie,
4. spustením príkazu `make`.

Zostavenie aplikácie pomocou CMake a jej spustenie ukazuje výpis 4.9. Pracovným adresárom pri spustení prvého príkazu je koreňový adresár zdrojového stromu. Výpis je pre prehľadnosť skrátený.

Spustiteľný súbor aplikácie má na systémoch GNU/Linux a FreeBSD názov `main`. Zostavenie na MS Windows popisujem v kapitole 5.

Kapitola 5

Testovanie

Vykonané testy overovali, či je programovací nástroj funkčný podľa požiadavkov vytýčených v cieli práce a v upresnení zadania po dohode so zadávateľom (podkapitola 2.1). Do množiny testovacích operačných systémov MS Windows a GNU/Linux som sa rozhodol pridať ešte systém FreeBSD, aby som lepšie otestoval ovládanie sériového portu podľa normy POSIX pre unixové operačné systémy. Tiež som pre kompiláciu na FreeBSD vyskúšal odlišný prekladač od GCC, Clang. Clang je na FreeBSD predvoleným kompilátorom. Cieľom bolo vylúčiť použitie programovacích konštrukcií, špecifických pre GCC a tak ukázať, že bude možné program s minimálnymi zmenami skompilovať aj inými prekladačmi.

Rozhodol som sa okrem dynamických testov, vyžadujúcich spustenie programu použiť aj statické testy. Statické testy kontrolujú chyby alebo nesprávne programátorské praktiky v zdrojovom kóde, ktoré nemusí odhaliť prekladač.

Všetky testy prebiehali ako systémové *Black Box* testy, pretože sa nimi overovala funkčnosť na rozhraní. V prípade vyvinutého programovacieho nástroja je týmto rozhraním používateľské rozhranie programovacieho nástroja tvorené parametrami spustiteľného súboru. Rovnakým spôsobom bude s programom interagovať používateľ alebo volajúci programový kód, napr. pri použití nástroja v skriptoch.

Snažil som sa čo najviac testov automatizovať. V niekoľkých prípadoch som zvolil testovanie s manuálnym zásahom. Jedná sa o kompiláciu a inštaláciu programovacieho nástroja na testovacom operačnom systéme, voľbu sériových komunikačných rýchlostí, voľbu sériového portu. Vykonanie týchto testov s manuálnym zásahom je jednoduchšie a v prípade voľby rýchlostí by úplne automatické testovanie nebolo možné. Dôvodom je, že keď sa inicializuje sériové rozhranie MCU v režime bootstrap zavádzača na konkrétnu rýchlosť, rýchlosť už nie je možné zmeniť bez manuálneho resetu MCU, tj. bez nového vstupu do režimu bootstrap zavádzača.

5.1 Automatické systémové testy

K vytvoreniu automatických testov som použil CTest distribuovaný ako súčasť CMake. Funkcionalita testovania sa sprístupní zavolaním `enable_testing()` v konfiguračnom súbore CMake. Test sa pridá volaním funkcie `add_test()` s potrebnými argumentami. Vykonanie všetkých pridaných testov sa spustí príkazom `ctest` v adresári použitom pre zostavovanie software. Pri ladení programu v závislosti od výsledkov testov je veľmi užitočné nastavenie `-I`, pomocou ktorého môže používateľ zadať len určité testy pre vykonanie.

MCU ST10F168 a ST10F269 sa líšia napr. rozdielnym delením FLASH pamäte na bloky. Časť testov je preto nutné vykonávať zvlášť pre ST10F168 a zvlášť pre ST10F269. Parametrom `CONFIGURATIONS` funkcie `add_test()` sa dá určiť, že test sa vykoná len v prípade, že používateľ si vyžiadal jednu z daných *konfigurácii* pri spustení príkazu `ctest` nastavením názvu konfigurácie pomocou `-C`. Príklad priebehu testov s použitím konfigurácie a výberu testov ukazuje výpis 5.1. V príklade je zvolená konfigurácia *st10f168* a vykoná sa každý 45. test od prvého.

Zdrojový kód 5.1: Príklad testovania programom CTest

```
$ ctest -C st10f168 -I ,,45
Test project /home/jan/tmp/skola/BP/sw/build
    Start  1: 168_MissingOption
1/2 Test  #1: 168_MissingOption ...     Passed    0.75 sec
    Start 46: 168_Write1B
2/2 Test #46: 168_Write1B ...........***Failed   15.70 sec

50% tests passed, 1 tests failed out of 2

Total Test time (real) =  16.46 sec

The following tests FAILED:
         46 - 168_Write1B (Failed)
Errors while running CTest
$
```

Funkcia `add_test()` je veľmi jednoduchá. Ako parametre môže prijímať len názov testu, príkaz s argumentami, názov konfigurácie a pracovný adresár. Pri vykonávaní testu sa spustí uvedený príkaz s argumentami. Ak je jeho návratová hodnota nula, test je úspešný. Ak je nenulová, test je neúspešný.

Pre programovací nástroj som ale potreboval určovať pre testy napr. konkrétnu návratovú hodnotu, ktorá má znamenať úspech, binárny súbor s dátami pre zapísanie do FLASH MCU a binárny súbor s overenými správnymi dátami, ktorý sa porovná s dátami spätne z pamäte prečítanými a tak sa určí výsledok operácie.

Spôsob implementácie rozšírených testov je nasledujúci. Ako príkaz pre spustenie sa predá funkcii `add_test()` samotný `cmake` a nastavením `-P` sa mu predá cesta k CMake skriptu, ktorý má vykonať. Argumenty sa skriptu predajú nastaveniami `-D`. `-D` vytvorí v prostredí skriptu premenné, nastavené na

príslušné hodnoty. Hodnoty z týchto premenných môže skript použiť. Príklad pridania rozšíreného testu ukazuje výpis 5.2. V príklade sú všetky hodnoty získavané z premenných.

Zdrojový kód 5.2: Príklad pridania rozšíreného testu

```
add_test(NAME ${TestNamespace}${NAME}
  CONFIGURATIONS ${TestConfigurations}
  COMMAND ${CMAKE_COMMAND}
  -DTestArgs=${ARGS}
  -DTestConfigOptions=${CONFIG_OPTIONS}
  -DTestExitCode=${EXITCODE}
  -DTestBlessedFile=${BLESSEDFILE}
  -DTestFile=${FILE}
  -P ${SCRIPT}
  )
```

Rozlíšil som štyri typy testov. Každý test som umiestnil do jedného súboru. Rozdelenie som sa rozhodol realizovať preto, aby som nemusel zadávať názvy operácií programovacieho nástroja medzi jeho argumenty manuálne a aby som mohol pre každý typ testu vykonať špecifickú prípravu a kontrolu.

- `erase.cmake` - pri teste mazania môže byť nutné inicializovať obsah FLASH pamäťe do určitého stavu. Z toho dôvodu skript najprv skontroluje, či je zadaná cesta k súboru pre zápis do FLASH. Ak áno, zostaví argumenty pre programovací nástroj pre zápis do FLASH, programovací nástroj spustí operáciu zápisu a skontroluje, či je návratový kód rovný nule. Potom sa zostavia argumenty pre programovací nástroj pre operáciu mazania, nástroj sa spustí s operáciou mazania a skontroluje sa, či návratová hodnota odpovedá tej, zadanej volajúcim tohoto skriptu. Po mazaní sa skontroluje, či bola skriptu dodaná cesta k súboru pre zápis a zároveň cesta k súboru s dátami, ktoré zodpovedajú obsahu FLASH pamäte po úspešnej operácii mazania (*požehnaný súbor*, angl. *blessed file*). Ak áno, prečíta sa obsah FLASH, skontroluje sa návratový kód operácie čítania a prečítané dáta sa porovnajú s obsahom požehnaného súboru.

  Skript je možné volať dvomi spôsobmi. Buď bez cesty k súboru pre zápis a zároveň bez cesty k požehnanému súboru, alebo s oboma cestami. Prvý spôsob je použitý v prípade, kedy sa testujú len nastavenia operácie mazanie, aby si programátor nemusel vymýšľať cesty k súborom a zaisťovať ich existenciu, aj keď nebudú využité. Príkladom je testovanie chybných alebo chýbajúcich argumentov. Druhý spôsob je použitý pri ostatných testoch mazania.

- `read.cmake` - skript kontroluje, či má zadanú cestu k požehnanému súboru, s ktorým sa porovnajú dáta prečítané z FLASH. Ak áno, pridá medzi argumenty operácie čítania cestu k súboru, kde sa uložia dáta prečítané z FLASH. Potom spustí test a skontroluje, či návratová hodnota

odpovedá tej, zadanej volajúcim tohoto skriptu. Po prečítaní pamäte, ak je zadaná cesta k požehnanému súboru porovnajú sa tieto súbory na zhodu.

Skript je možné volať dvomi spôsobmi z rovnakých dôvodov, ako je to vysvetlené pre operáciu mazania.

- `write.cmake` - skript najprv zostaví argumenty pre operáciu zápisu a čítania. Potom spustí operáciu zápisu a skontroluje, či návratová hodnota odpovedá tej, zadanej volajúcim skriptu. Nakoniec, ak bola zadaná cesta k súboru pre zápis do FLASH a zároveň cesta k požehnanému súboru, prečíta sa obsah FLASH a porovná sa s požehnaným súborom na zhodu.

  Tak ako skripty `erase.cmake` a `read.cmake`, je možné z rovnakých dôvodov skript volať dvomi spôsobmi.

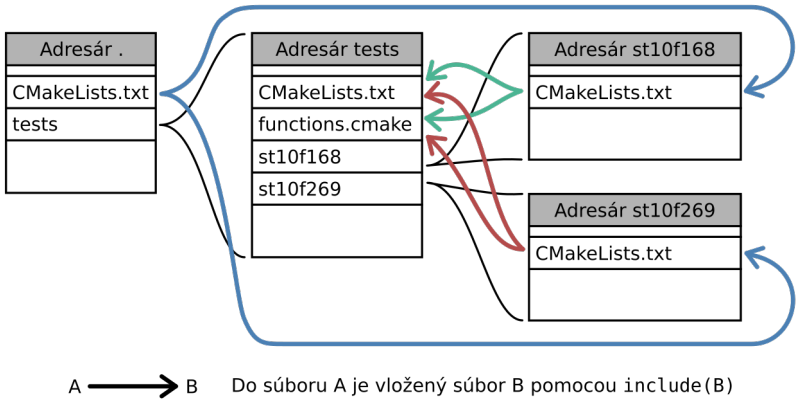
- `other_operations.cmake` - najjednoduchší skript pre všetky ostatné testy. Najprv zostaví argumenty nástroja, potom ho spustí a porovná návratovú hodnotu s tou, zadanou volajúcim. Medzi argumenty nedopĺňa názov operácie.

Pomocné makrá v súbore `functions.cmake`:

- `M_SET_CONFIG_OPTIONS` - pridá medzi nastavenia programu rýchlosť komunikácie, port a frekvenciu MCU, ak sú definované.
- `M_ADD_TEST_SCRIPT` - pridá rozšírený test. Podobne ako na výpise 5.2.

Pomocné funkcie v súbore `functions.cmake`:

- `EXEC_TEST` - spustí programovací nástroj so zadanými argumentami a skontroluje jeho návratovú hodnotu so zadanou. Test skončí s chybou aj v prípade, že neprebehne do 120 sekúnd.
- `COMPARE_FILES` - porovná dva súbory. Využíva pri tom samotný program `cmake` s nastavením `-E compare_files`. Takto spustený program porovná súbory na zhodu obsahu.
- `ADD_ERASE_TEST` - pridá medzi argumenty programovacieho nástroja nastavenia pre rýchlosť komunikácie, port a frekvenciu a pridá rozšírený test mazania.
- `ADD_READ_TEST` - pridá medzi argumenty programovacieho nástroja nastavenia pre rýchlosť komunikácie, port a frekvenciu a pridá rozšírený test čítania.
- `ADD_WRITE_TEST` - pridá medzi argumenty programovacieho nástroja nastavenia pre rýchlosť komunikácie, port a frekvenciu a pridá rozšírený test zápisu.



Obr. 5.1: Hierarchia CMake súborov s definíciou testov

- `ADD_NORMAL_TEST` - pridá medzi argumenty programovacieho nástroja nastavenia pre rýchlosť komunikácie, port a frekvenciu a pridá obecný rozšírený test.
- `ADD_WITHOUT_CONFIG_TEST` - nepridáva medzi argumenty programovacieho nástroja žiadne nastavenia a pridá rozšírený test. Táto funkcia sa používa pre pridanie testu, pri ktorom sa vyžaduje maximálna kontrola nad argumentami programovacieho nástroja. Používa sa len pre kontrolu spracovania argumentov.

Po sprístupnení funkcionality testovania v konfiguračnom súbore `CMakeLists.txt` zavolaním `enable_testing()` , sa spustí pridanie testov definovaných v súboroch pre ST10F168 a ST10F269. To ukazuje výpis 5.3.

Zdrojový kód 5.3: Pridanie testov z ďalších súborov

```
enable_testing ()

include(${CMAKE_SOURCE_DIR}/tests/st10f168/CMakeLists.txt)
include(${CMAKE_SOURCE_DIR}/tests/st10f269/CMakeLists.txt)
```

Hierarchiu CMake súborov definujúcich automatické testy ukazuje obrázok 5.1. Na obrázku sú v adresároch uvedené len súbory týkajúce sa automatických testov. Adresáre a súbory s testovacími dátami sú pre prehľadnosť vynechané.

Pri niektorých testoch záleží na poradí ich vykonávania. Príkladom môže byť test automatického mazania blokov pri zápise. Do celej FLASH pamäte sa najprv zapíše hodnota 0x00. Potom sú do pamäte zapisované stále väčšie dáta a sleduje sa, v akom stave je obsah celej FLASH po každom zápise. Tak sa skracuje čas testovania. Pred každým zápisom nie je nutná inicializácia obsahu FLASH do určitého stavu, ale využíva sa stav z predchádzajúceho testu. Ostatné testy sa môžu spúšťať samostatne a nezáleží na tom, kde v sérii testov sú zaradené.

5.1.1 Spoločné testy

Pridávajú sa do konfigurácie pre ST10F168 aj ST10F269. Pred názov testu sa pridá prípona `168_` alebo `269_` podľa toho, či sa pridáva do konfigurácie pre ST10F168 alebo pre ST10F269. Oblasti funkcionality testované spoločnými testami:

- neuvedený názov operácie,
- identifikácia pripojeného MCU,
- formát hodnôt pre nastavenia,
- zadaný názov neznámej operácie,
- chýbajúce cesty k vstupnému a výstupnému súboru,
- neznáme nastavenie,
- chýbajúca hodnota pre nastavenie,
- viac ako jeden vstupný alebo výstupný súbor,
- neplatný názov sériového portu,
- a nemožnosť otvorenia vstupného a výstupného súboru.

Informácie o konkrétnych testoch je možné získať v súbore `src/impl/tests/CMakeLists.txt` v prílohe B.

5.1.2 Testy pre ST10F168

Volaním `include()` sa sprístupnia funkcie a makrá zo súboru `functions.cmake`. Nastavia sa premenné určujúce konfiguráciu, do ktorej budú patriť pridávané testy (v tomto prípade *st10f168*). Pridajú sa testy bez konfigurácie pre ST10F168. Volaním `include()` sa pridajú testy spoločné pre ST10F168 aj ST10F269 a potom sa už len pridávajú testy pre ST10F168. Aby som zabránil kolízii názvov testov, k názvu testov pre ST10F168 sa pridá predpona `168_`. Oblasti funkcionality testované pre ST10F168:

- neuvedená frekvencia pre operáciu identifikácie MCU,
- neuvedená frekvencia pre ostatné operácie,
- mazanie blokov pri operácií mazania,
- zápis do FLASH pamäte,
- automatické mazanie blokov pri zápise,
- a čítanie z FLASH pamäte.

Informácie o konkrétnych testoch je možné získať v súbore `src/impl/tests/st10f168/CMakeLists.txt` v prílohe B.

5.1.3 Testy pre ST10F269

Volaním `include()` sa sprístupnia funkcie a makrá zo súboru `functions.cmake`. Nastavia sa premenné určujúce konfiguráciu, do ktorej budú patriť pridávané testy (v tomto prípade *st10f269*). Pridajú sa testy bez konfigurácie pre

ST10F269. Volaním `include()` sa pridajú testy spoločné pre ST10F168 aj ST10F269 a potom sa už len pridávajú testy pre ST10F269. Aby som zabránil kolízii názvov testov, k názvu testov pre ST10F269 sa pridá predpona `269_`.Oblasti funkcionality testované pre ST10F269:

- uvedená frekvencia MCU (pre ST10F269 sa ignoruje),
- mazanie blokov pri operácií mazania,
- zápis do FLASH pamäte,
- automatické mazanie blokov pri zápise,
- a čítanie z FLASH pamäte.

Informácie o konkrétnych testoch je možné získať v súbore `src/impl/tests/st10f269/CMakeLists.txt` v prílohe B.

5.2 Statická analýza zdrojového kódu

Pre statickú analýzu C++ kódu som použil nástroj Cppcheck. Nainštaloval som ho z repozitárov linuxovej distribúcie openSUSE. Statické testy prebiehali v prostredí nasledujúcich verzií software:

- Operačný systém GNU/Linux, distribúcia openSUSE Leap 42.1 64-bit
- Cppcheck 1.70

Cppcheck som spustil v príkazovom riadku v adresári so zdrojovými kódmi (pozri 5.4). V podadresári `build` sú súbory vzniknuté zostavovaním software pomocou CMake. Zostavenie software je v tomto prípade nutné preto, že hlavičkové súbory s dátami firmware a nápovedou k programu v zdrojovom strome neexistujú a generujú sa automaticky až pri zostavovaní software. Po zostavení sa budú nachádzať v podadresári `build`.

Príkazom `find` som našiel všetky zdrojové súbory C++ pre otestovanie v súborom strome začínajúcom aktuálnym adresárom. Príkazom `grep` som zo zoznamu ciest k nájdeným súborom vyradil zdrojové súbory z adresárov použitých pre zostavovanie software a zvyšné cesty som pomocou `xargs` predal programu `cppcheck` pre spracovanie. Prepínač `--check-config` zapína testovanie zdrojového kódu vo všetkých sekciách podmieneného prekladu (uzavretých medzi direktívy preprocesoru `#ifdef`, `#ifndef` a `#endif`). `--enable=all` zapína všetky kontroly, ktoré Cppcheck vie. `--force` povoľuje kontrolu zdrojových súborov s veľkým množstvom konfigurácie. V mojom prípade je to zdrojový súbor `SerialPort/SerialPortUnix.cpp`, kde je osemnásť sekcií `#ifdef` použitých pre detekciou sériových rýchlostí podporovaných unixovým systémom, na ktorom prebieha kompilácia. Nastaveniami `-I` sa pridávajú adresáre obsahujúce hlavičkové súbory, aby ich Cppcheck pri spracovávaní zdrojových súborov vedel nájsť. Okrem štandardných zdrojových adresárov pridávam aj adresár `build` obsahujúci hlavičkové súbory generované pri zostavovaní software. Prepínač `--suppress=missingIncludeSystem` potlačuje chybové hlásenia

Zdrojový kód 5.4: Spustenie statických testov Cppcheck

```
$ find . -name "*.cpp" | grep -v build | xargs cppcheck \
    --check-config --enable=all --force -I . \
    -I ./SerialPort -I ./build -I ./McuSt10f168 \
    -I ./McuSt10f269 -I ./build/McuSt10f168 \
    -I ./build/McuSt10f269 --suppress=missingIncludeSystem
```

spôsobené nenájdenými systémovými hlavičkovými súbormi. Tie som do kontroly nezahrnul.

Statická analýza C++ kódu pomocou Cppcheck odhalila jednu nepoužívanú metódu a odchytávanie výnimky hodnotou, nie pomocou referencie. Nájdené chyby som opravil a spustil kontrolu ešte raz. Potom už prebehla bez chýb.

5.3 Scenár testovania

Štartovací bod scenára testovania je zmena v zdrojových kódoch programovacieho nástroja. Po nej sa vždy začína krokom **S.1** algoritmu:

S.1 Zostaviť binárne súbory firmware. V prípade chyby opraviť a ísť na **S.1**.

S.2 Označiť každý podporovaný OS ako *neotestovaný*.

S.3 Statická analýza zdrojového kódu. V prípade chyby opraviť a ísť na **S.1**.

S.4 Ak je už každý z podporovaných OS označený ako *otestovaný*, ísť na **S.11**.

S.5 Zvoliť ďalší podporovaný OS, označený ako *neotestovaný*.

S.6 Zostaviť spustiteľný súbor programovacieho nástroja na zvolenom OS. V prípade chyby opraviť a ísť na **S.1**.

S.7 Spustiť automatické testy pre ST10F168 a ST10F269 s komunikačnými rýchlosťami

- 19 200 Bd - doporučovaná výrobcom ako bezpečná rýchlosť,
- 57 600 Bd - stredná rýchlosť vzhľadom na najnižšie maximum spomedzi podporovaných MCU na testovacích doskách,
- maximálna rýchlosť pre ST10F168 a ST10F269.

V prípade chyby opraviť a ísť na **S.1**.

S.8 Spustiť operáciu zápisu celej pamäte s kontrolou spätným prečítaním s tromi rôznymi názvami sériového portu. V prípade chyby opraviť a ísť na **S.1**.

S.9 Nainštalovať programovací nástroj. To je test, že je možné program zaradiť do umiestnení, kde operačný systém hľadá spustiteľné súbory a spustiť cez príkazový riadok bez zadania plnej cesty. V prípade chyby opraviť a ísť na **S.1**.

S.10 Označiť zvolený OS ako *otestovaný*. Ísť na **S.5**.

S.11 Koniec testovania.

Statickú analýzu stačí vykonať len raz pre všetky podporované OS, pretože zdrojový strom programovacieho nástroja je pre všetky OS vždy rovnaký.

Maximálne rýchlosti som našiel experimentálne a vyberal som spomedzi štandardných rýchlostí sériovej linky (celé násobky 19 200 Bd). Na testovacích doskách pre ST10F168 je maximálna rýchlosť 115 200 Bd a pre ST10F269 je to 230 400 Bd. V iných zapojeniach MCU môžu byť rýchlosti iné. Záleží na taktovacej frekvencii MCU a elektronike na sériovej linke.

Uvedené maximálne rýchlosti sú platné pre operačný systém, na ktorom vývoj prebiehal a pre priame použitie prevodníka medzi USB a RS-232. Vo virtualizovanom prostredí sa tieto rýchlosti nedali efektívne použiť.

5.4 Voľba a prevádzka testovacích operačných systémov

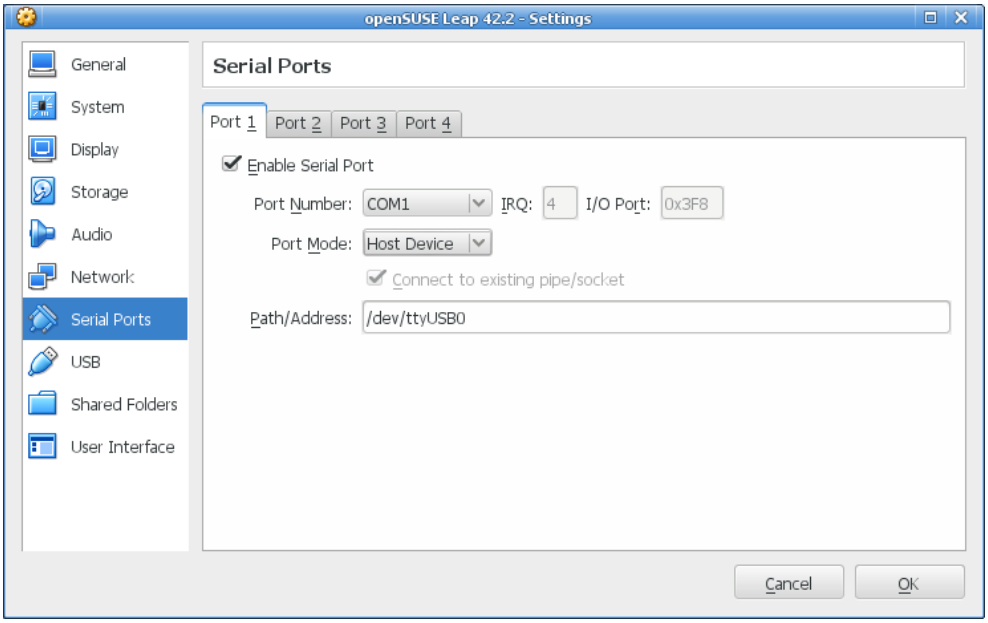
Programovací nástroj som vyvíjal a testoval na notebooku Lenovo THINKPAD X220i. Notebook má predinštalovaný systém MS Windows 7. GNU/Linux som si nainštaloval sám. Je to distribúcia openSUSE Leap 42.1 64-bit.

Programovací nástroj som testoval vždy na čistej inštalácií operačného systému s nainštalovaným software nutným pre zostavenie a testovanie aplikácie. Testovacie systémy som prevádzkoval vo virtualizovanom prostredí VirtualBox 5.1.10.

Systémy MS Windows verzií 7, 8.1 a 10 som získal vo forme obrazov pre VirtualBox z [37]. Namiesto verzie Windows 8, stačí testovanie na Windows 8.1. Windows 8.1 je aktualizovaný Windows 8, nie *nový* operačný systém.

Pri voľbe distribúcií GNU/Linux pre testovanie som vychádzal z dvoch hlavných prúdov distribúcií, čo sa týka aktuálnosti v nich obsiahnutého software. Distribúciu openSUSE Leap 42.2 64-bit som vybral ako zástupcu distribúcií pre bežné použitie. Získal som ju z [38]. Pozostáva z aktuálneho, aj keď menej otestovaného software. Distribúciu CentOS som zvolil ako zástupcu *enterprise* distribúcií. Vychádza zo zdrojových kódov systému Red Hat Enterprise Linux. Obsahuje starší ale dobre otestovaný software. CentOS som získal z [39].

Operačný systém FreeBSD som získal z [40]. Zvolil som aktuálnu verziu 11.0 pre architektúru amd64.



Obr. 5.2: Sprístupnenie sériového portu hostiteľa virtuálnemu stroju

Virtualizovaným operačným systémom som musel sprístupniť sériový port hostiteľského systému. To sa dalo nastaviť v konfigurácii každého virtuálneho stroja. V hostiteľskom operačnom systéme GNU/Linux je prevodník medzi USB a RS-232 reprezentovaný súborom zariadenia `/dev/ttyUSB0`. Modul jadra hostiteľského systému pre prevodník je `ftdi_sio`. Virtualizovanému systému je sériový port viditeľný ako štandardný sériový port založený na obvode 16550 UART. Nastavenie ukazuje obrázok 5.2. Pre prenos zdrojových súborov testovanej aplikácie do prostredia virtualizovaného systému som použil sieťové spojenie medzi hostiteľským a virtualizovaným systémom. Na hostiteľskom systéme som spustil FTP server, z virtualizovaného systému som k nemu pristupoval a prenášal dáta.

5.5 MS Windows

Operačné systémy MS Windows majú výbornú spätnú kompatibilitu. Preto bol postup testovania na všetkých troch verziách MS Windows rovnaký.

Ako FTP klienta pre získanie zdrojových súborov programovacieho nástroja som využil štandardného prieskumníka súborov. V adresovom riadku prieskumníka som na FTP úložisko pristúpil zadaním cesty vo formáte `ftp://login@host`, kde `login` je používateľské meno na severi a `host` je adresa FTP serveru.

Pri inštalácii mingw-w64 som najprv skúšal použiť inštalátor, ktorý automaticky stiahne najaktuálnejšie nástroje podľa zvolenej konfigurácie. Vždy

to ale skončilo chybou sťahovania. Preto som si podľa zvolenej konfigurácie vyhľadal potrebný archív s danou konfiguráciou manuálne a rozbalil. Archív bol vo formáte 7z a pre rozbalenie som nainštaloval program 7-Zip 16.04.

Testovanie prebiehalo v nasledujúcich verziách software:

- Operačný systém Windows 7 Enterprise, 32-bit, 6.1.7601 Service Pack 1 Build 7601
- Operačný systém Windows 8.1 Enterprise Evaluation, 32-bit, 6.3.9600 N/A Build 9600
- Operačný systém Windows 10 Enterprise Evaluation 64-bit, 10.0.14393 N/A Build 14393
- mingw-w64 i686-6.2.0-release-win32-dwarf-rt_v5-rev1
- CMake 3.7.1

Zo *štart menu* som spustil GUI CMake. V okne som vybral cestu k zdrojovým súborom software (*Browse Source*) a adresáru, kde sa bude software zostavovať (*Browse Build...*). Po stlačení tlačidla *Configure* sa ukázalo dialógové okno s voľbou generátoru výstupu a kompilátorov. Generátor som zvolil *MinGW Makefiles* a vybral možnosť *Specify native compilers*. Na ďalšej obrazovke dialógu som zadal cestu k

- C kompilátoru `C:\Users\IEUser\Desktop\mingw32\bin\gcc`
- C++ kompilátoru `C:\Users\IEUser\Desktop\mingw32\bin\g++`

Cestu ku kompilátoru pre Fortran som nezadal. Výber som potvrdil. Zobrazila sa chybová hláška, že nastala chyba v procese konfigurácie a konfiguračné súbory môžu byť neplatné. Dôvodom bolo, že CMake nevedel nájsť make program a shell. Zaškrtol som voľbu *Advanced* vedľa vyhľadávacieho poľa a v zozname premenných sa mi zobrazili chybné premenné. V tomto prípade to boli `CMAKE_MAKE_PROGRAM` a `CMAKE_SH`. Po kliknutí na pole *Value* som mohol nastaviť hodnoty týchto premenných. Zadal som

- pre `CMAKE_MAKE_PROGRAM` hodnotu `C:\Users\IEUser\Desktop\mingw32\bin\mingw32-make`
- pre `CMAKE_SH` hodnotu `C:\Windows\System32\cmd.exe`

Ešte bolo treba upraviť nastavenia kompilátorov, aby sa štandardné knižnice C a C++ linkovali k programu staticky. Inak nebolo možné program spustiť. Nastavil som

- `CMAKE_CXX_FLAGS` na `-static-libgcc -static-libstdc++`
- `CMAKE_C_FLAGS` na `-static-libgcc -static-libstdc++`

Po stlačení *Configure* konfiguračný proces prebehol v poriadku. Po stlačení *Generate* sa vygenerovali Makefile súbory. V príkazovom riadku som prešiel do adresára pre zostavenie software a spustil `mingw32-make` zadaním plnej

cesty `C:\Users\IEUser\Desktop\mingw32\bin\mingw32-make`. Výsledný spustiteľný súbor mal názov `main.exe`.

Testy som spúšťal príkazom `ctest` so zadaním plnej cesty k príkazu. Testovanie odhalilo nekonečné čakanie programu pri prekročení časového limitu prístupu k sériovému portu. Po oprave testovanie prebehlo bez chýb.

Názvy sériového portu som menil v správcovi zariadení. Spustí sa zadaním príkazu `devmgmt.msc` s právami administrátora. V zozname som našiel sériový port, klikol naň pravým tlačítkom myši a zvolil *Properties*. V dialógovom okne som prešiel na záložku *Port Settings* a klikol na tlačítko *Advanced...*. V ďalšom dialogovom okne je zoznam *COM Port Number*, v ktorom je možné zvoliť názov pre COM port.

Pri testovaní som zistil, že aj keď operačný systém vo volaní Win32 API informuje programovací nástroj, že sériový port podporuje používateľsky definovanú rýchlosť, nedá sa v skutočnosti nastaviť. Experimentálne nájdené maximum bolo 115 200 Bd.

5.6 GNU/Linux

Pri testovaní v distribúciách GNU/Linux sa vyskytol problém s rýchlosťami sériového portu podobne, ako pri testovaní v MS Windows. Vypisovanie dostupných rýchlostí v GNU/Linux fungovalo správne a dali sa nastaviť. Avšak, aj pri nastavení 230 400 Bd operácie trvali rovnako dlho, ako pri 115 200 Bd.

5.6.1 OpenSUSE

Pre získanie zdrojových súborov z FTP som použil príkaz
`wget -r ftp://pouzivatelske_meno:heslo@host/adresar` .

Do čistej inštalácie som doinštaloval balíky cmake, gcc, gcc-c++ so závislosťami. Používateľský účet, použitý pri testovaní, som musel priradiť do skupiny `dialout`, aby som mohol pristupovať k zariadeniu `/dev/ttyS0`. To malo vlastníka `root` a skupinu `dialout` a obe mali práva pre zápis a čítanie. Ostatným bol prístup odopretý. Pred zostavením software som editoval v súbore `CMakeLists.txt` cestu k zariadeniu sériového portu pre unixové systémy na `/dev/ttyS0`.

Testovanie v distribúcii openSUSE prebiehalo v prostredí nasledujúcich verzií software:

- Distribúcia openSUSE Leap 42.2 64-bit
- Linuxové jadro 4.4.27
- glibc 2.22
- GCC 4.8.5
- CMake 3.5.2

Pri testoch s rôznym názvom sériového portu som využil toho, že zariadenia sú reprezentované súbormi. Vytvoril som jednoducho symbolický od-

kaz na súbor zariadenia `/dev/ttyS0`. Symbolické odkazy som vytváral v adresári, v ktorom prebiehalo zostavenie aplikácie. Skúšal som názvy `ttyUSB1234`, `SeriovyPort123456` a `4a:ce4e-4d06d45`. Vždy som musel aplikáciu zostaviť s nastavením nového názvu súboru zariadenia, aby sa zmena premietla do testov.

Automatické testovanie odhalilo nesprávne nastavenie veľkosti blokov pamäte FLASH ST10F168. Chyba sa objavila pri automatických testoch mazania blokov. Po oprave testovanie prebehlo bez chyby.

5.6.2 CentOS

Verzia CMake v repozitároch CentOS 7 bola 2.8.11. Vyskúšal som upraviť minimálnu potrebnú verziu CMake v hlavnom konfiguračnom súbore `CMakeLists.txt`, ale aplikáciu nebolo možné zostaviť. Verzia 2.8.11 neobsahuje niektoré funkcie verzie 3.3.2, ktorú som používal pri vývoji. Namiesto úpravy CMake súborov pre zostavenie aplikácie som sa rozhodol skompilovať na CentOS novšiu verziu CMake. Pri tomto postupe je menšie riziko zanesenia chyby do CMake súborov a aplikáciu nebolo nutné znovu testovať na všetkých podporovaných systémoch. Ďalším dôvodom bolo, že na ostatných testovacích operačných systémoch prevažuje novšia verzia CMake. Preto som prípad s CMake na CentOS riešil ako výnimku.

Po inštalácii som musel nakonfigurovať sieťové pripojenie. Reštartoval som sieťové rozhranie príkazmi `ifdown enp0s3` a `ifup enp0s3`. Nainštaloval som balíky wget, gcc, gcc-c++. Skompiloval som novú verziu CMake zo zdrojových kódov a používal tú.

Testovanie v distribúcii CentOS prebiehalo v prostredí nasledujúcich verzií software:

- Distribúcia CentOS 7 64-bit
- Linuxové jadro 3.10.0
- glibc 2.17
- GCC 4.8.5
- CMake 3.7.1

Testovanie prebehlo bez chýb.

5.7 FreeBSD

Vo FreeBSD je predvoleným kompilátorom Clang od verzie 10.0. Predtým sa používal GCC. Clang je súčasťou základného systému. Nainštaloval som len CMake. Zariadenie sériového portu bolo `/dev/ttyu0`. Malo práva len pre používateľa `root` pre zápis a čítanie. Skupina súboru bola `wheel`. Používateľa, pod ktorým som testoval, som priradil do skupiny `wheel` a skupine dal práva pre čítanie a zápis.

Vypisovanie dostupných rýchlostí vo FreeBSD fungovalo správne a dali sa nastaviť. Maximálna rýchlosť bola 115 200 Bd a trvanie operácií jej zodpovedalo.

Testovanie v distribúcii FreeBSD prebiehalo v prostredí nasledujúcich verzií software:

- Operačný systém FreeBSD 11.0-RELEASE-p1 64-bit
- Clang 3.8.0
- CMake 3.6.2

Neuvádzam verziu štandardnej knižnice jazyka C ani verziu jadra. FreeBSD má svoju vlastnú implementáciu štandardnej knižnice C. Jadro systému a základné nástroje používateľského priestoru tvoria jeden vývojový celok a sú vyjadrené verziou operačného systému FreeBSD.

Názov sériového portu som menil rovnako v GNU/Linux, symbolickými odkazmi. Skúšal som rovnaké názvy ako pri testovaní v GNU/Linux.

Testovanie prebehlo bez chýb.

Záver

V práci som sa zaoberal analýzou, návrhom a realizáciou software pre vykonávanie operácií mazania, zápisu a čítania internej programovej pamäte FLASH mikrokontrolérov ST10F168 a ST10F269 a testovaním výsledku práce na splnenie cieľových vlastností pod podporovanými operačnými systémami.

Počas práce som narazil na chyby mikrokontroléra ST10F269. Podarilo sa mi ich vyriešiť vďaka informáciam z literatúry a vlastným experimentovaním.

Vytvorený programovací nástroj umožňuje mazať, zapisovať a čítať internú programovú FLASH pamäť mikrokontrolérov ST10F168 a ST10F269 a identifikovať pripojený mikrokontrolér. Pre každý z operačných systémov MS Windows od verzie Windows 7 až po Windows 10, GNU/Linux a FreeBSD je realizovaný ako spustiteľný súbor. Program bol otestovaný v prostredí týchto systémov. Má riadkové používateľské rozhranie, vstavaný návod na použitie, umožňuje výpis informácie s názvom a verziou programu, umožňuje výpis dostupných komunikačných rýchlostí a voľbu sériovej komunikačnej rýchlosti a ľubovoľného sériového portu. Má voľbu pre výpis informačných hlásení o svojom vykonávaní a informáciu o percentuálnom postupe čítania a zápisu pamäte FLASH.

Používateľ môže mazať celú pamäť, alebo len ľubovoľné jednotlivé bloky FLASH pamäte. Pre čítanie môže používateľ špecifikovať, koľko bajtov od adresy nula sa má prečítať a názov výstupného súboru, kam bude prečítaný obsah uložený. Pri zápise sa dá určiť, či sa budú mazať len zapisované bloky, alebo celá pamäť a názov súboru obsahujúceho dáta zapisované do FLASH od adresy nula. Pri zápise si môže používateľ vyžiadať kontrolu spätným prečítaním zapisovanej oblasti pamäte. Program považuje vstupné a výstupné súbory za binárne.

V budúcnosti by bolo možné aplikáciu rozšíriť o podporu formátu Intel Hex pre vstupné a výstupné súbory. Tiež by bolo možné pridať možnosť špecifikácie štartovacej adresy čítania a zápisu používateľom a podporu pre ďalšie modely mikrokontrolérov.

Literatúra

[1] ARM Ltd and ARM Germany GmbH.: *STMicroelectronics ST10F168*. [online]. (2016). [cit. 2016-11-22]. Dostupné z: `http://www.keil.com/dd/chip/3184.htm`

[2] ARM Ltd and ARM Germany GmbH.: *STMicroelectronics ST10F269*. [online]. (2016). [cit. 2016-11-22]. Dostupné z: `http://www.keil.com/dd/chip/3236.htm`

[3] *System on Modules, OEM-ODM Services*. [online]. PHYTEC America, LLC. (2016). [cit. 2016-11-20]. Dostupné z: `http://phytec.com/`

[4] *Hitex Embedded Systems Safety Security Tools Engineering*. [online]. Hitex GmbH. (2016). [cit. 2016-11-20]. Dostupné z: `https://www.hitex.com/en/`

[5] *Home - STMicroelectronics*. [online]. STMicroelectronics. (2016). [cit. 2016-11-20]. Dostupné z: `http://www.st.com/content/st_com/en.html`

[6] STMicroelectronics [online]: *ST10 Flasher DLL. 2001. [cit. 2016-11-17].* Dostupné z: `http://www.mikrocontroller.net/attachment/286818/ST10FlasherDLL.pdf`

[7] *Digital Kaos - support forum for dreambox receivers and general cable and satellite tv enthusiasts - Interface ST10Flasher*. [online]. [cit. 2016-10-18]. Dostupné z: `http://www.digital-kaos.co.uk/forums/showthread.php/67491-Interface-ST10Flasher/`

[8] *Forum - Custom Frontend tool for ST10Flasher.dll - STMicroelectronics*. [online]. [cit. 2016-11-22]. Dostupné z: `http://bit.ly/2fyk7kL`

[9] Martech: *Martech*. [online]. 2011. [cit. 2016-11-22]. Dostupné z: `http://shop.martech.pl/en/`

[10] Dataman Ltd.: *Device Programmers & ISP Programming by Dataman Programmers*. [online]. 2016. [cit. 2016-11-22]. Dostupné z: http://www.dataman.com/

[11] Martech: *ST10 Flasher - Martech*. [online]. [cit. 2016-11-22]. Dostupné z: http://shop.martech.pl/en/programmer/82-st10-flasher.html

[12] Dataman Ltd.: *Dataman 48Pro2 Super Fast Universal ISP Programmer*. [online]. [cit. 2016-11-22]. Dostupné z: http://www.dataman.com/programmers/dataman-48pro2-super-fast-universal-isp-programmer.html

[13] STMicroelectronics [online]: *ST10F168 16-BIT MCU WITH 256K BYTE FLASH MEMORY AND 8K BYTE RAM. 2002. [cit. 2016-08-20].* Dostupné z: http://www.keil.com/dd/docs/datashts/st/st10f168_ds.pdf

[14] STMicroelectronics [online]: *ST10F269 16-BIT MCU WITH MAC UNIT, 256K BYTE FLASH MEMORY AND 12K BYTE RAM. 2013. [cit. 2016-08-20].* Dostupné z: http://www.st.com/resource/en/datasheet/st10f269.pdf

[15] STMicroelectronics [online]: *ST10F168 ST EMBEDDED ALGORITHM KERNEL (STEAK) FOR FLASH PROGRAMMING / ERASING. 2000. [cit. 2016-11-20].* Dostupné z: https://my.st.com/public/STe2ecommunities/mcu/Lists/ST10/Attachments/1022/AN1247.pdf

[16] AŽD Praha s.r.o. [online]: *ELEKTRONICKÉ KOLEJOVÉ OBVODY AŽD TYP KOA1. 2010. [cit. 2016-11-28].* Dostupné z: http://www.azd.cz/admin/files/Dokumenty/pdf/Produkty/Kolejove/30-KOA-1.pdf

[17] *Serial Communications in Win32*. [online]. Microsoft. (1995). [cit. 2016-10-28]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms810467.aspx

[18] *Windows Dev Center*. [online]. Microsoft. (2016). [cit. 2016-10-28]. Dostupné z: https://developer.microsoft.com/en-us/windows

[19] *Serial Programming Guide for POSIX Operating Systems*. [online]. Sweet, M. R. (1999). [cit. 2016-10-28]. Dostupné z: https://www.cmrr.umn.edu/~strupp/serial.html

[20] *The Open Group Base Specifications Issue 7*. [online]. The IEEE and The Open Group. (2016). [cit. 2016-10-28]. Dostupné z: http://pubs.opengroup.org/onlinepubs/9699919799/

[21] *Manifesto for Agile Software Development.* [online]. Beedle, M. a kol. (2001). [cit. 2016-10-18]. Dostupné z: http://agilemanifesto.org/

[22] STMicroelectronics [online]: *AN1496 APPLICATION NOTE. Flash Programming / Reprogramming ST10F269 / ST10F280. ROGER, A. 2013. [cit. 2016-08-20].* Dostupné z: http://www.st.com/resource/en/application_note/cd00004343.pdf

[23] ARM Ltd and ARM Germany GmbH.: *Keil XC16x/C16x/ST10 Microcontroller Development Tools.* [online]. (2016). [cit. 2016-10-20]. Dostupné z: http://www.keil.com/c166/

[24] ARM Ltd and ARM Germany GmbH.: *C166 Product Manuals.* [online]. (2016). [cit. 2016-10-20]. Dostupné z: http://www.keil.com/support/man_c166.htm

[25] ARM Ltd and ARM Germany GmbH.: *Keil Evaluation Software Limitations.* [online]. (2016). [cit. 2016-10-20]. Dostupné z: http://www.keil.com/demo/limits.asp

[26] STMicroelectronics [online]: *ST10 FAMILY PROGRAMMING MANUAL. 2013. [cit. 2016-08-20].* Dostupné z: http://www.st.com/resource/en/programming_manual/cd00147146.pdf

[27] *MinGW-w64 - for 32 and 64 bit Windows.* [online]. [cit. 2016-11-17]. Dostupné z: https://sourceforge.net/projects/mingw-w64/

[28] *xd: Extended Dump and Load Utility.* [online]. Walker, J. (2000). [cit. 2016-11-17]. Dostupné z: https://www.fourmilab.ch/xd/

[29] Texas Instruments [online]: *MAX232x Dual EIA-232 Drivers/Receivers. 2014. [cit. 2016-11-12].* Dostupné z: http://www.ti.com/lit/ds/symlink/max232.pdf

[30] STMicroelectronics [online]: *L78S 2 A positive voltage regulator IC. 2014. [cit. 2016-08-11-12].* Dostupné z: https://www.soselectronic.cz/productdata/12/64/38/126438/L78S.pdf

[31] STMicroelectronics [online]: *L78 Positive voltage regulator ICs. 2016. [cit. 2016-08-11-12].* Dostupné z: http://www.st.com/resource/en/datasheet/l78.pdf

[32] *Forum - ST10F269 on chip Flash/Boostrap - STMicroelectronics.* [online]. [cit. 2016-11-21]. Dostupné z: https://my.st.com/dbc9692a

[33] *Forum - ST10F168 Monitor001.hex source code - STMicroelectronics.* [online]. [cit. 2016-11-20]. Dostupné z: https://my.st.com/cd9230ae

[34] Hitex GmbH [online]: *In-System Programming Of The ST10F168 FLASH. [cit. 2016-11-20].* Dostupné z: http://hitx7.securesites.net/iuk/c166/pfunc.pdf

[35] *Hex2bin Intel Hex or Motorola Hex file converter.* [online]. Pelletier, J. [cit. 2016-11-19]. Dostupné z: https://sourceforge.net/projects/hex2bin/

[36] *CMmake.* [online]. Kitware, Inc. [cit. 2016-11-17]. Dostupné z: https://cmake.org/

[37] *Free Virtual Machines from IE8 to MS Edge - Microsoft Edge Development.* [online]. Microsoft. (2016). [cit. 2016-10-28]. Dostupné z: https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/

[38] *software.opensuse.org:.* [online]. SUSE and others. (2011). [cit. 2016-10-28]. Dostupné z: https://software.opensuse.org/422/en

[39] *Download CentOS.* [online]. The CentOS Project. (2016). [cit. 2016-10-28]. Dostupné z: https://www.centos.org/download/

[40] *Download FreeBSD.* [online]. The FreeBSD Project. (2016). [cit. 2016-10-28]. Dostupné z: https://www.freebsd.org/where.html

Dodatok **A**

Zoznam použitých skratiek

API Application Programming Interface

ASCII American Standard Code for Information Interchange

BSD Berkeley Software Distribution

CP Context Pointer

CSP Code Segment Pointer

DPP Data Page Pointer

FTP File Transfer Protocol

FW Firmware

GCC GNU Compiler Collection

GNU GNU's Not Unix

GPR General-Purpose Register

GUI Graphical User Interface

IP Instruction Pointer

IRAM Internal Random Access Memory

ISO International Organization for Standardization

LED Light-Emitting Diode

MCU Microcontroller Unit

MS Microsoft

OS Operačný systém

PC Personal Computer

POSIX Portable Operating System Interface

RAM Random Access Memory

SP Stack Pointer

STEAK ST Embeded Algorithm Kernel

SW Software

TTL Tranzistorovo-tranzistorová logika

UML Unified Modeling Language

USB Universal Serial Bus

V/V Vstupno/Výstupný

XRAM Extended Random Access Memory

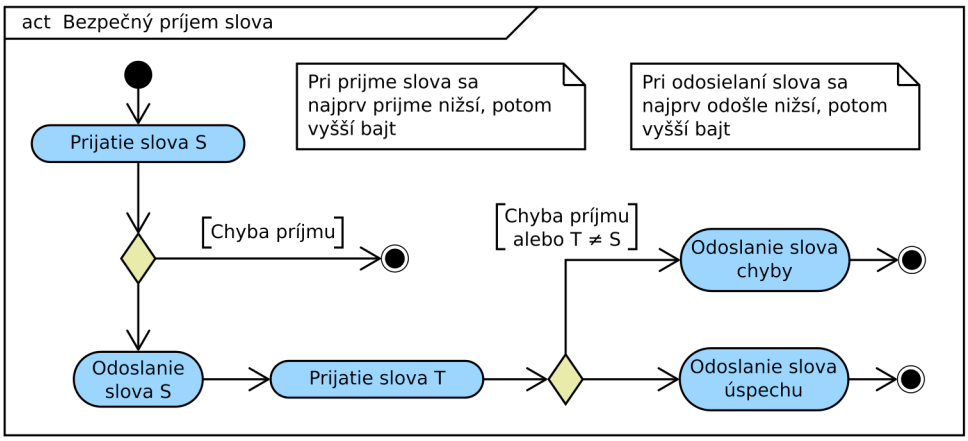
Dodatok **B**

Obsah priloženého CD

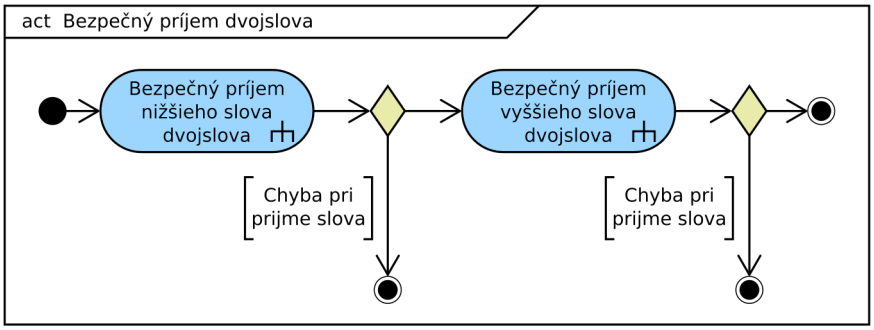
- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresár so spustiteľnými súbormi programovacieho nástroja
- `src`
  - `error-data` ... binárne súbory pre reprodukciu chyby MCU ST10F269 popisovanej v časti 4.6.5
  - `impl` ..... zdrojové kódy programovacieho nástroja
    - `firmware` ..... zdrojové kódy firmware
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `BP_Sučan_Ján_2017.pdf` ..... text práce vo formáte PDF

Dodatok C

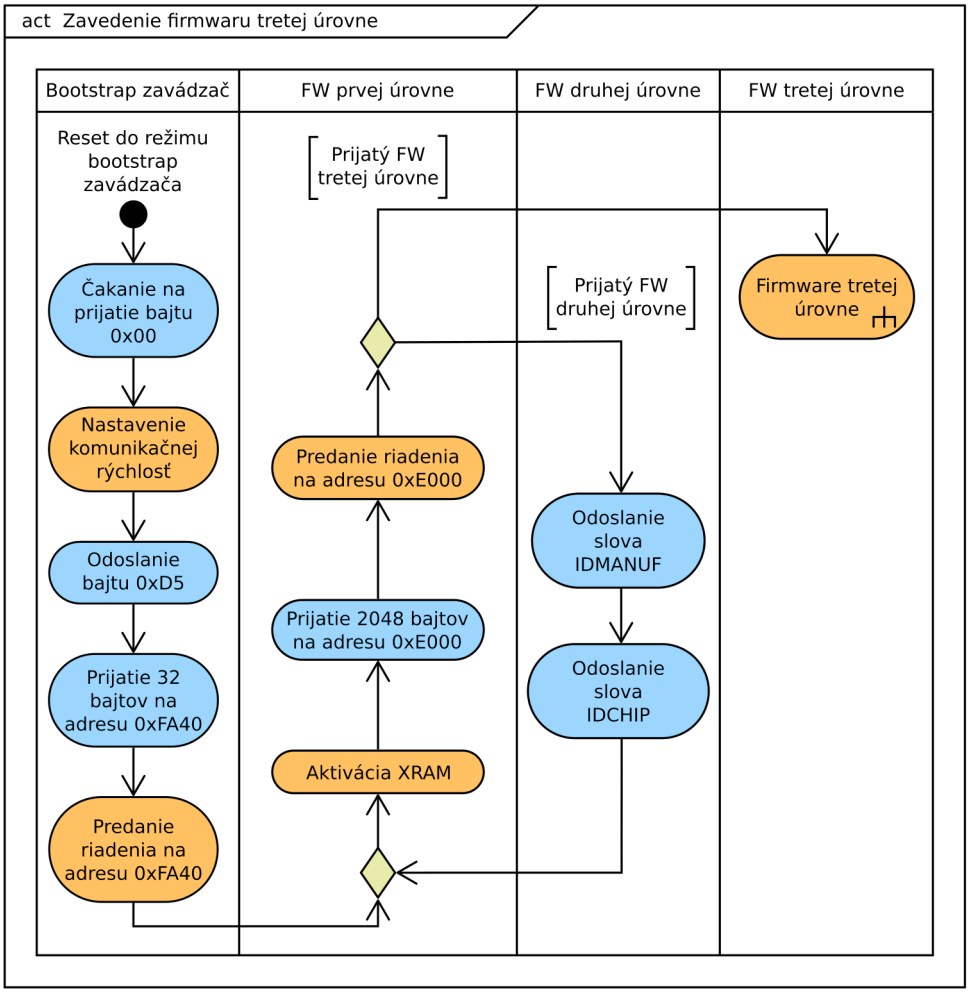
UML diagramy komunikačného protokolu



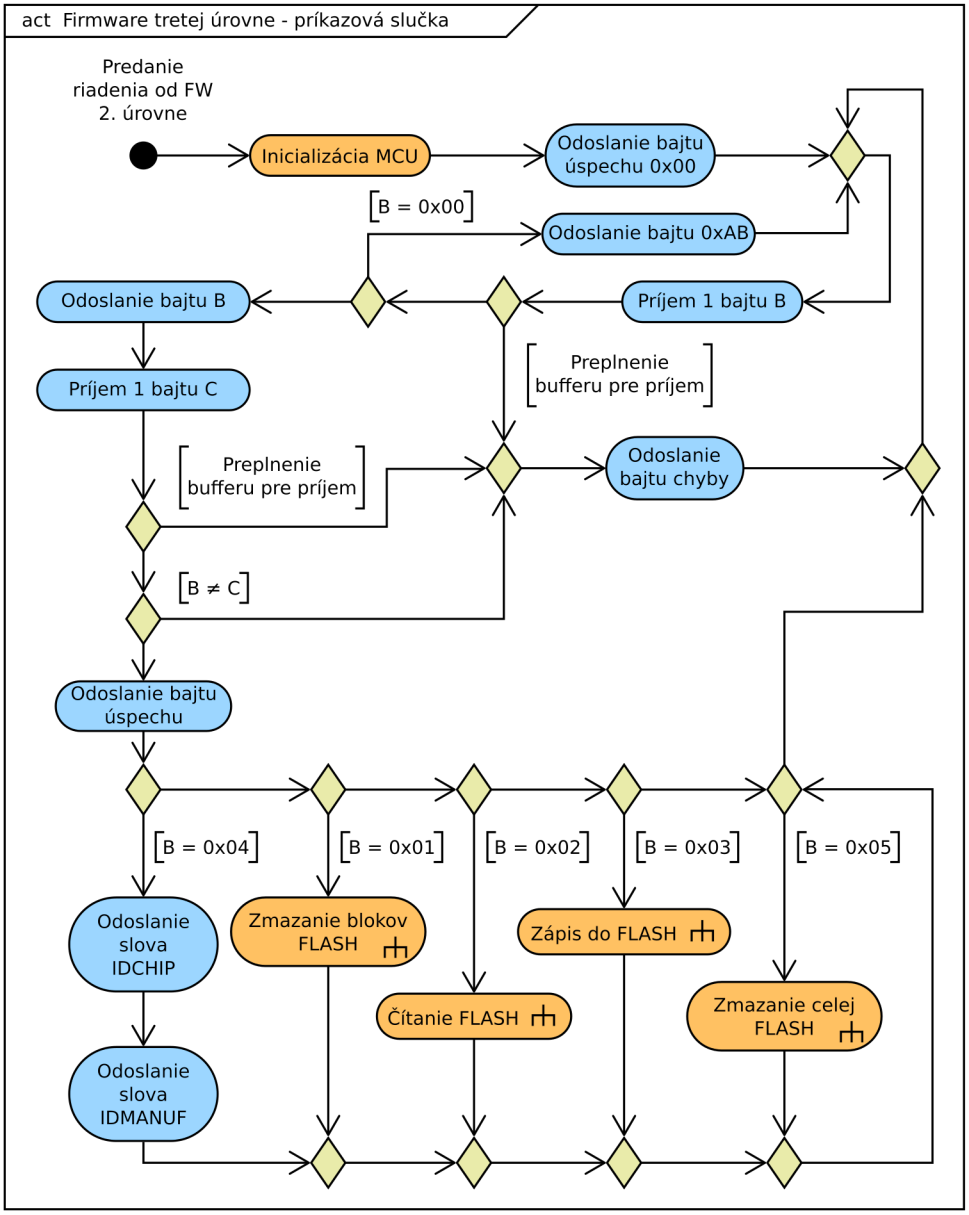
Obr. C.1: Protokol bezpečného prijatia slova



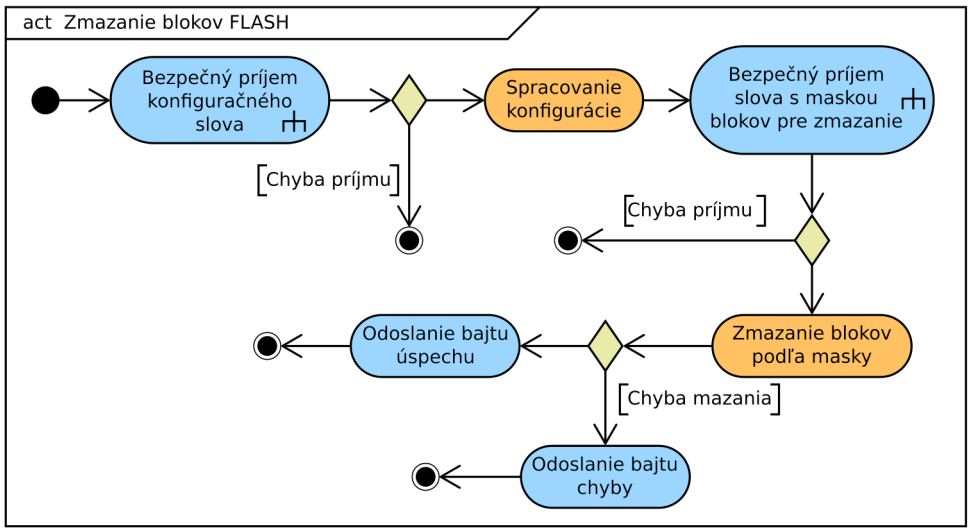
Obr. C.2: Protokol bezpečného prijatia dvojslova



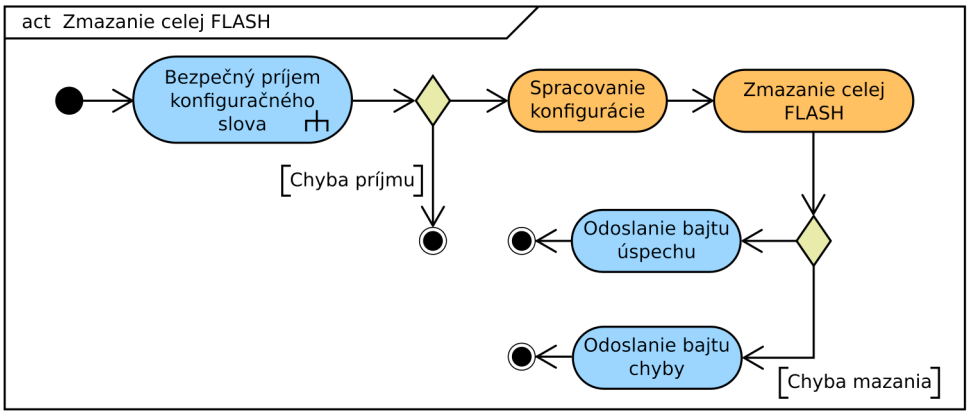
Obr. C.3: Protokol zavedenia firmware tretej úrovne do MCU po resete



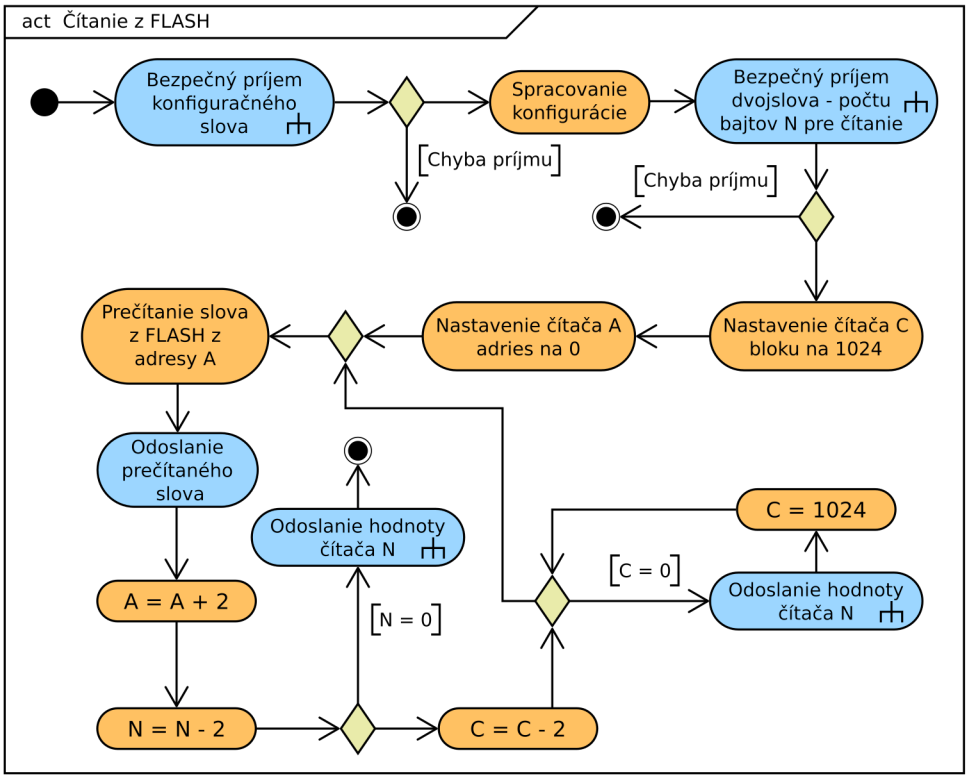
Obr. C.4: Protokol príkazovej slučky



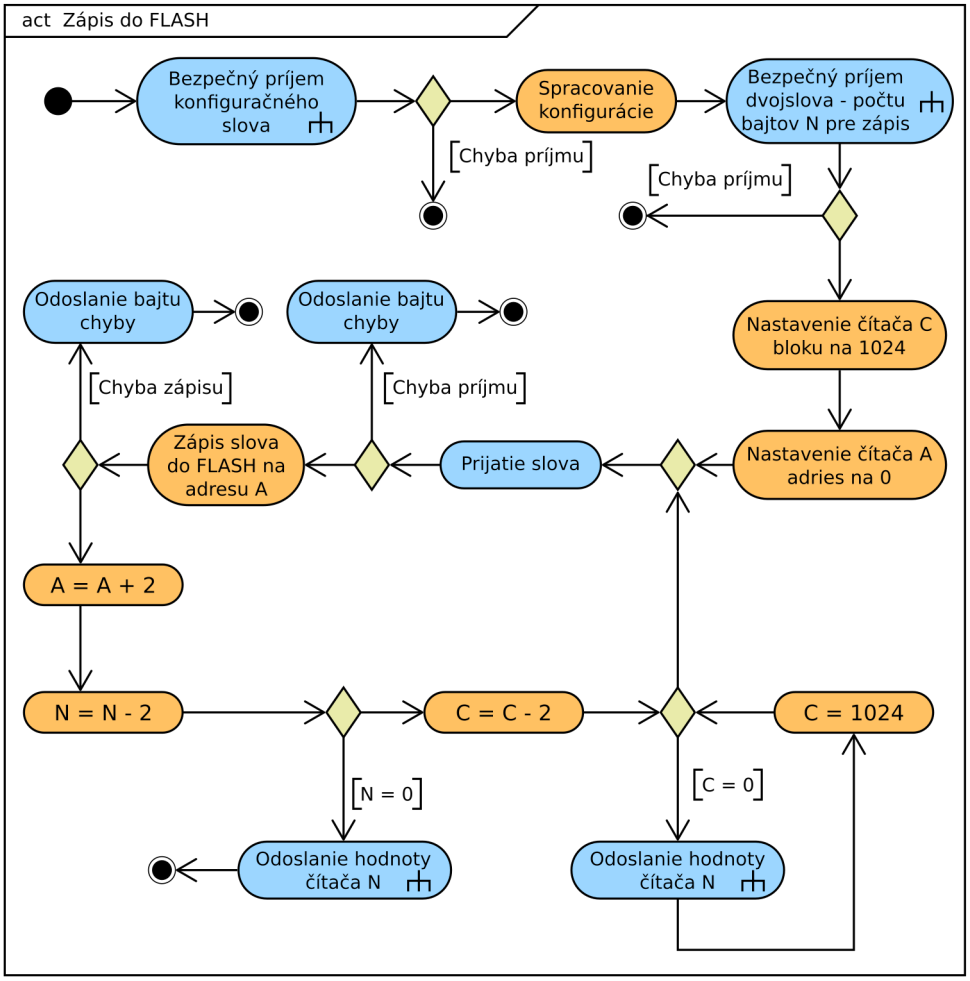
Obr. C.5: Protokol mazania blokov programovej FLASH



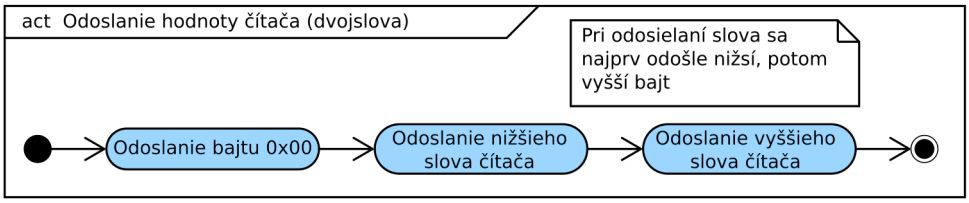
Obr. C.6: Protokol mazania celej programovej FLASH



Obr. C.7: Protokol čítania z FLASH



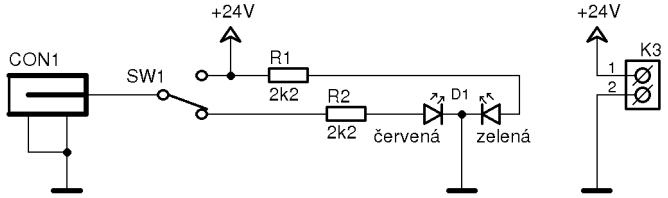
Obr. C.8: Protokol zápisu do FLASH



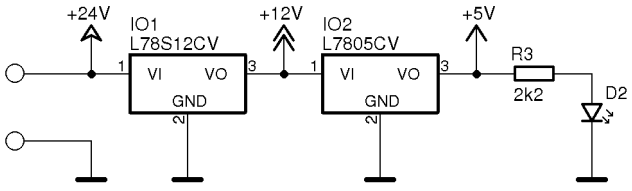
Obr. C.9: Protokol odosielania obsahu 4-bajtového čítača pri zápise a čítaní

Dodatok **D**

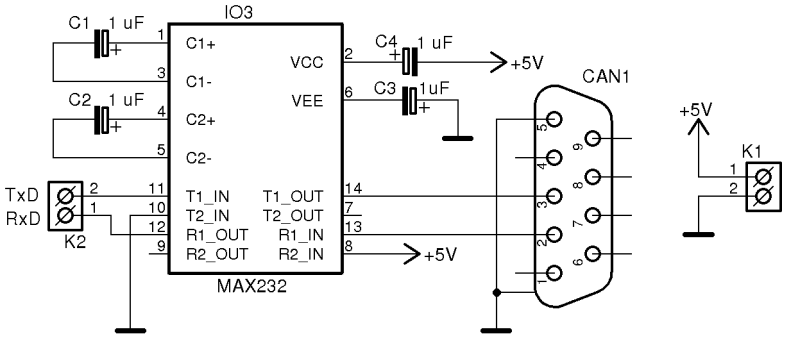
Schémy podporných obvodov



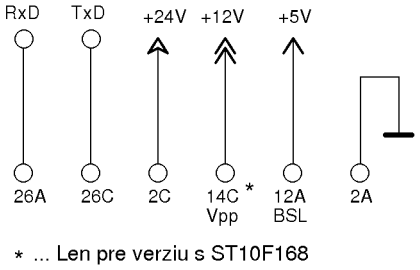
Obr. D.1: Schéma spínacieho prípravku napájacieho napätia



Obr. D.2: Schéma zapojenia stabilizátorov napájacieho napätia



Obr. D.3: Schéma prevodníka medzi RS-232 a TLL úrovňami signálov



Obr. D.4: Schéma pripojenia na kontakty vývojovej dosky