ZÁPADOČESKÁ UNIVERSITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

Katedra aplikované elektroniky a telekomunikací

DIPLOMOVÁ PRÁCE

Rozhraní pro komunikaci s integrovaným bootloaderem v mikrokontroléru STM32

Autor: Bc. Kryštof Przechowski 2018

Vedoucí práce: Ing. Petr Krist Ph.D.

Konzultant: Ing. Luboš Koudelka

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2017/2018**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Kryštof PRZECHOWSKI**
Osobní číslo:	**E15N0006P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a aplikovaná informatika**
Název tématu:	**Rozhraní pro komunikaci s integrovaným bootloaderem v mikrokontroléru STM32**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

Navrhněte zařízení realizující rozhraní mezi integrovaným bootloaderem v cílovém mikrokontroléru STM32 a uživatelskou aplikací na osobním počítači PC. S počítačem bude zařízení připojeno přes USB rozhraní a s cílovým mikrokontrolérem prostřednictvím odpovídajícího komunikačního rozhraní bootloaderu - USART, I2C/SPI, CAN. Jednoduchá aplikace na PC bude schopna vizualizovat připojený cílový mikrokontrolér, nahrát do něho binární kód, vyčíst obsah paměti a podporovat všechny příkazy implementované bootloaderem.

1. Seznamte se s vývojovými prostředky pro mikrokontroléry STM32 a s funkcí integrovaného bootloaderu mikrokontrolérů STM32.
2. Navrhněte a implementujte rozhraní pro komunikaci s bootloaderem za použití druhého mikrokontroléru STM32 (bootloader Master).
3. Navrhněte ovládání tohoto rozhraní prostřednictvím PC.
4. Zamyslete se nad možnostmi zlepšení integrovaného bootloaderu, případně nad implementací bootloaderu vlastního z hlediska optimalizace rychlosti flashování, zabezpečení, apod.

Rozsah grafických prací:	**podle doporučení vedoucího**
Rozsah kvalifikační práce:	**40 - 60 stran**
Forma zpracování diplomové práce:	**tištěná/elektronická**

Seznam odborné literatury:

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

Vedoucí diplomové práce:	**Ing. Petr Krist, Ph.D.** Katedra aplikované elektroniky a telekomunikací
Datum zadání diplomové práce:	**10. října 2017**
Termín odevzdání diplomové práce:	**24. května 2018**

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

L.S.

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 10. října 2017

Abstrakt

Diplomová práce se zabývá integrovaným bootloaderem v mikrokontrolérech STM32, popisuje fungování bootloaderu, zapojení a možnosti použití. Je zde popsána sériová komunikace s bootloaderem a komunikace s PC. Pro komunikaci mezi PC a bootloaderem byl vytvořen firmware. Pro PC byl vytvořen software, který slouží koncovému uživateli k ovládání a zadávání dat do bootloaderu.

Klíčová slova

Integrovaný bootloader, STM32F446RE, sériová komunikace USART, sériová komunikace SPI, sériová komunikace I2C, sériová komunikace CAN, PC aplikace

Abstract

The master thesis deals with an integrated bootloader in microcontrollers STM32, it describes how the bootloader works, its connection and how to use it. Serial communication with the bootloader and the PC is introduced. The firmware for communication between the PC and the bootloader was created. The PC software for the end user was also created which allows control and data entrance to the bootloader.

Key words

Integrated bootloader, STM32F446RE, serial communication USART, serial communication SPI, serial communication I2C, serial communication CAN, PC application

Prohlášení

Prohlašuji, že mnou vypracovaná diplomová práce je vypracována samostatně, s použitím odborné literatury, která je uvedena v seznamu.

Dále prohlašuji, že mnou použitý software je legální.

……..

Podpis

Obsah

Seznam symbolů a zkratek

ACK.....Acknowledgement
CAN..... Controller Area Bus
CRC .....Cyclic Redundancy Check
DMA.....Direct Memory Access
GPIO..... General Purpose Input / Output
HAL .....Hardware Abstraction Layer
I2C ..... Inter – Integrated Circuit
ID.....Identification
LSB.....Least Significant Bit
MISO .....Master In Slave Out
MOSI .....Master Out Slave In
MSB..... Most Significant Bit
NACK.....Not Acknowledgement
PC .....Personal Computer
RTC ..... Real Time Counter
SPI ..... .Serial Peripheral Interface
SRAM.....Static Random Access Memory
USB .....Universal Serial Bus
USART ..... Universal Synchronous / Asynchronous Receiver and Transmitter
XOR..... Exclusive OR

1 Úvod

Diplomová práce se zabývá vývojem firmware pro mikrokontrolér STM32F446RE umístěným na Nucleo 64 kitu, který by byl schopný komunikovat s integrovaným bootloaderem v mikrokontrolérech STM32 po sériové komunikaci USART, SPI, I2C a CAN. Dále by pak měl být vyvinut software pro PC. Software by měl být schopný komunikovat s Nucleo kitem a slouží pro ovládání bootloaderu koncovým uživatelem.

V úvodní části jsou uvedeny vývojové prostředky pro mikrokontroléry STM32 a je zde vysvětleno, k čemu slouží. Poté je zde stručně představen mikrokontrolér, pro který byl vyvíjen firmware a vysvětleny funkce integrovaného bootloaderu pro mikrokontroléry STM32.

Další část je věnována firmware. Firmware by měl implementovat všechny příkazy pro bootloader sériové komunikace USART, SPI, I2C a CAN. Dále zde má být vytvořena komunikace s PC po USART, kterou má být možné řídit příkazy pro bootloader a zadávat nebo přijímat z bootloaderu data.

Poslední část se zabývá softwarem. Software by měl komunikovat s Nucleo kitem přes COM port. Měla by být vytvořena okenní aplikace, kterou by uživatel volil jednotlivé příkazy pro bootloader. Bylo by možné zadávat do bootloaderu data a naopak vizualizovat data přijatá z bootloaderu. V této části je uveden manuál pro uživatelskou aplikaci.

2 Vývojové prostředky pro mikrokontroléry STM32

Vývojových prostředků pro mikrokontroléry STM32 je celá řada. Mezi vývojová prostředí určená pro psaní, ladění a překlad firmware patří například AVR-GCC, Keil uVision4 a novější verze Keil uVision5, nebo System Workbench for STM32. Dále také Attolic TrueSTUDIO vyvíjené firmou Attolic zakoupené firmou ST-Microelectronic. Pro psaní firmware bylo zvoleno prostředí System Workbench for STM32. Pro konfiguraci periferií byl použít STM32 CubeMX. Testování stavu paměti a stavových registrů bylo prováděno pomocí STM32 – ST LINK Utility.

Odkazy ke stažení jednotlivých vývojových prostředků:

GCC: *http://blog.zakkemble.co.uk/avr-gcc-builds/*
Keil uVision: *https://www.keil.com/download/product/*
System Workbench for STM32: *http: //www.st.com/en/development-tools/sw4stm32.html#getsoftware-scroll*
Attolic TrueSTUDIO: *https://atollic.com/resources/download/*
STM32 CubeMX : *http://www.st.com/en/development-tools/stm32cubemx.html*
STM32 – ST LINK Utility: *http://www.st.com/en/development-tools/stsw-link004.html*

2.1 System Workbench for STM32

Systém Workbench for STM32 je prostředí pro vývoj firmware. Je zdarma, ale pro jeho stažení je nutné se zaregistrovat. Na vývoji se podílí přímo firma ST-Microelectronics vyrábějící i mikrokontroléry. Z tohoto důvodu je vhodné použít pro vývoj firmware pro mikrokontroléry ST-Microelectronics právě toto vývojové prostředí. Dalším plusem je, že vývojové prostředí obsahuje v základní instalaci i překladač pro mikrokontroléry od ST-Microelectronic, tudíž pro práci s těmito mikrokontroléry není třeba nic dalšího instalovat. Další výhodou na rozdíl od Keilu je neomezená velikost kódu, který je možné přeložit. Při tvorbě nového projektu je možné vybrat druh mikrokontroléru, pro který se vytvoří startup projekt, nebo použít startup projekt vygenerovaný CubeMX. [1]

2.2 CubeMX

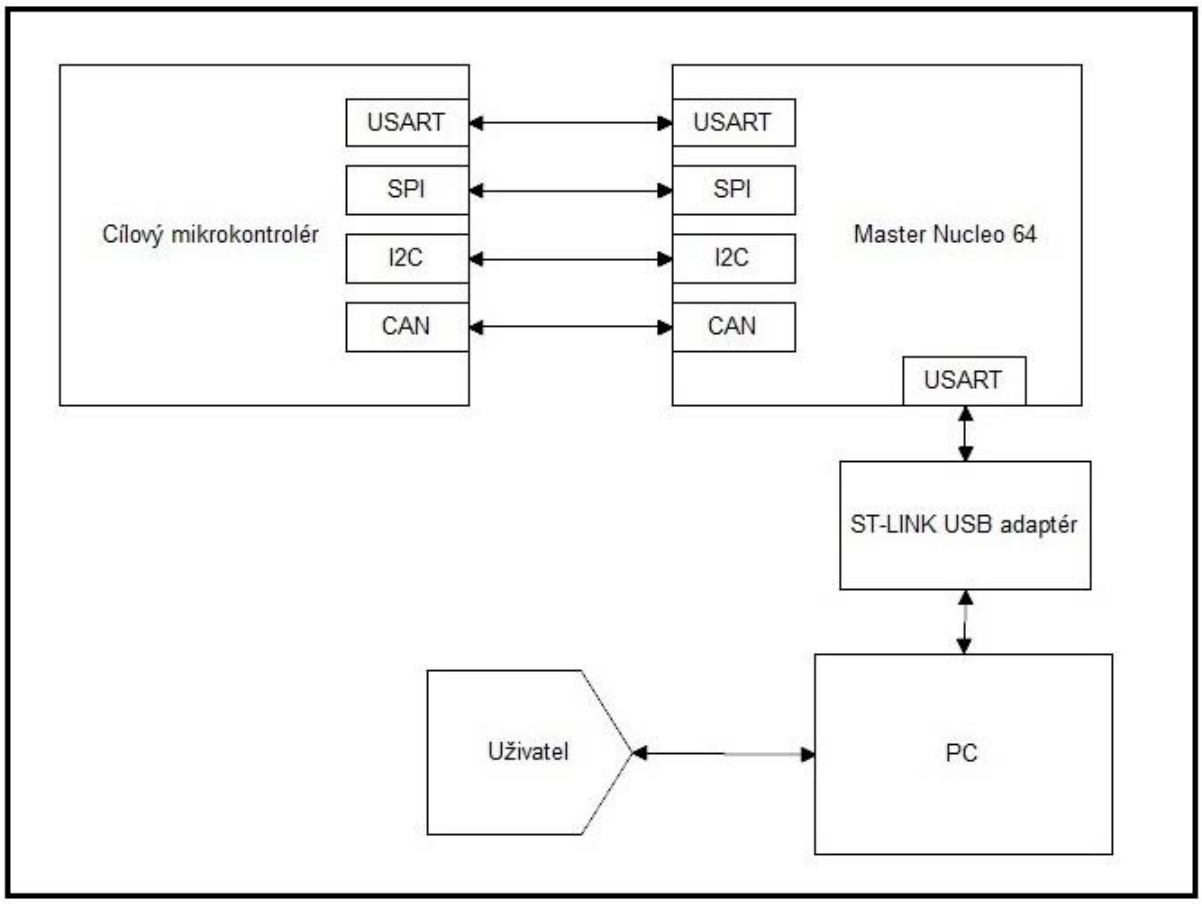
CubeMX je grafický nástroj od firmy ST-Microelectronics. Slouží k nastavení a inicializaci komponent: periferií, systémových hodin, timerů atd. pro jednotlivé 32 bitové mikrokontroléry od ST-Microelectronics. Podle nastavení se vygeneruje startup projekt, v tomto projektu jsou inicializovány komponenty a naimportovány HAL knihovny k těmto komponentám. Projekt je možné importovat do různých vývojových prostředí, např. i do Keil uVision5. [2]

2.3 STM32 – ST LINK Utility

STM – ST Link Utility je nástroj volně dostupný na stránkách společnosti ST-Microelectronic. Program je určen pro mikrokontroléry řady STM32 propojených s počítačem přes ST Link. Pomocí tohoto programu je možné otevírat soubory formátu .bin, .hex, .srec a mezi sebou je převádět. Tyto soubory jsou v zásadě výstupem překladače, linkeru a konverzního programu pro výstup. Soubory je možné do paměti mikrokontroléru nahrát. Dále je možné z paměti mikrokontroléru číst, mazat tuto paměť, nebo nastavit její ochranu (read/write). Paměť se zobrazuje v tabulce spolu s adresou, a jestli přepíšeme hodnotu tabulky, přepíše se paměť mikrokontroléru na dané adrese. Je možné zrušit ochranu paměti, pokud je paměť chráněna před čtením provede se nejprve mazání celé paměti. Tento program poskytuje další funkce, jako například číst registry jádra, ale pro účely této práce nebyly tyto funkce zapotřebí.

3 Popis komponent

Pro práci s vestavěným bootloaderem v mikrokontrolérech STM32 je nutné vytvořit zařízení, které by pomocí některé komunikace (např. USART,SPI,I2C…) řídilo pomocí sekvence bytů vestavěný bootloader cílového mikrokontroléru.



Obr. 1: Základní blokové schéma jednotlivých komponent

Celé zařízení se skládá z dvou Nucleo 64 kitů. Cílové Nucleo slouží pro testování. Master Nucleo obsahuje firmware, tento firmware obsluhuje obsluhu potřebných periferií, příkazy, které aktivují podporované funkce bootloaderu a s tím spojenou logiku. Dále je využíván ST-LINK USB adaptér, který je obsažen na Nucleo kitu. Adaptér je nutný, jelikož komunikace s PC je v mikrokontroléru implementovaná pomoci USART. Další pro chod

důležitá část je uživatelské PC, na kterém běží software. Software je určen především k uživatelské obsluze. Uživatel vybírá komunikaci, data pro cílové Nucleo a příkazy. Nemusí jít vždy pouze o Nucleo, ale cílem může být i jiné zařízení, na kterém je použit stejný mikrokontrolér, je možné ho přepnout do bootovacího režimu a má vyvedenou některou z podporovaných komunikací.

3.1 Mikrokontrolér

Firma ST-Microelectronic poskytla dva vývojové kity STM32 Nucleo-64. Jedno nucleo slouží jako Master, druhé jako Slave. Master řídí komunikaci s bootloaderem cílového mikrokontroléru. Byl použit mikrokontrolér STM32F446RE. Tento mikrokontrolér obsahuje kromě základních periferií jako je například DMA, RTC, časovače atd. i komunikační sběrnice I2C, SPI, USART a CAN. Sběrnice I2C, SPI a USART jsou na čipu implementovány víckrát, je možné použít jeden typ sběrnice víckrát. CAN je implementován pouze jako CAN1 Master a CAN2 Slave.

3.1.1 Integrovaný bootloader mikrokontrolérů STM32

Integrovaný bootloader mikrokontrolérů STM32 byl vyvinut například proto, aby bylo možné šetřit vývody na čipu a vývody ze zařízení. Jako cesta, kterou lze do Flash paměti zapisovat, číst z ní nebo jí mazat, může sloužit jedna z již použitých komunikací (např. konektor pro připojení čidla přes I2C). Vyvíjená aplikace by měla podporovat komunikace I2C, SPI, USART a CAN. Pro aktivaci bootovacího režimu, při kterém je možné do paměti takto zapisovat, je nutné nastavit určité piny a restartovat zařízení. Blíže bude tato problematika vysvětlena v následujícím textu. Komunikaci po sériové sběrnici začíná Master nucleo a s bootloaderem cílového zařízení se specifikuje pomocí příkazů, viz dále.

Kódová oblast začíná na adrese 0x0000 0000 díky své pevné mapě paměti. Tato oblast je přístupná po sběrnici ICode/Dcode, zatímco datová oblast (SRAM) začínající na adrese 0x2000 0000 je přístupná po systémové sběrnici. Mikrokontrolér STM32 s jádrem Cortex – M4 načte reset vektor na sběrnici ICode, to znamená, že spouštěcí oblast je dostupná pouze

v kódové oblasti (Flash paměti). Mikrokontroléry řady STM32 implementují tři různé režimy umožňující spouštět kód z různých sektorů paměti (například SRAM). Tyto režimy se volí piny BOOT[1,0]. [3]

BOOT1	BOOT0	Bootovací režim	Spouštěcí oblast
x	0	Hlavní Flash paměť	Spouštěcí oblastí je hlavní Flash paměť
0	1	Systémová paměť	Spouštěcí oblastí je systémová paměť
1	1	Vestavěná SRAM paměť	Spouštěcí oblastí je vestavěná SRAM paměť

obr. 2: Tabulka bootovacích režimů [3]

Hodnoty BOOT pinů jsou načteny se čtvrtou náběžnou hranou hodin po resetu. Spouštěcí oblast je zvolena podle nastavení těchto BOOT pinů. BOOT0 má samostatný pin, zatímco BOOT1 a GPIO sdílí jeden pin. Po načtení BOOT1 může být pin používán v režimu GPIO. V systémové paměti je uložen kód bootloaderu který umožňuje režim vestavěného bootloaderu. Režim vestavěného bootloaderu se používá k přeprogramování Flash paměti pomocí sériových rozhraní:

- USART
- CAN2
- I2C
- SPI
- USB (Device Firmware Upgrade)

Systémový bootovací režim lze opustit příkazem *Go command*, který podporují všechny komunikace nebo změnou nastavení BOOT pinů a systém resetem. Příkaz *Go command* se liší v závislosti na zvolené komunikaci a je blíže vysvětlen v kapitole 5.1.2.6, 5.2.2.6, 5.3.2.6 a 5.4.2.6. [3][4]

3.1.2 Vestavěná Flash paměť mikrokontroléru STM32F446xx

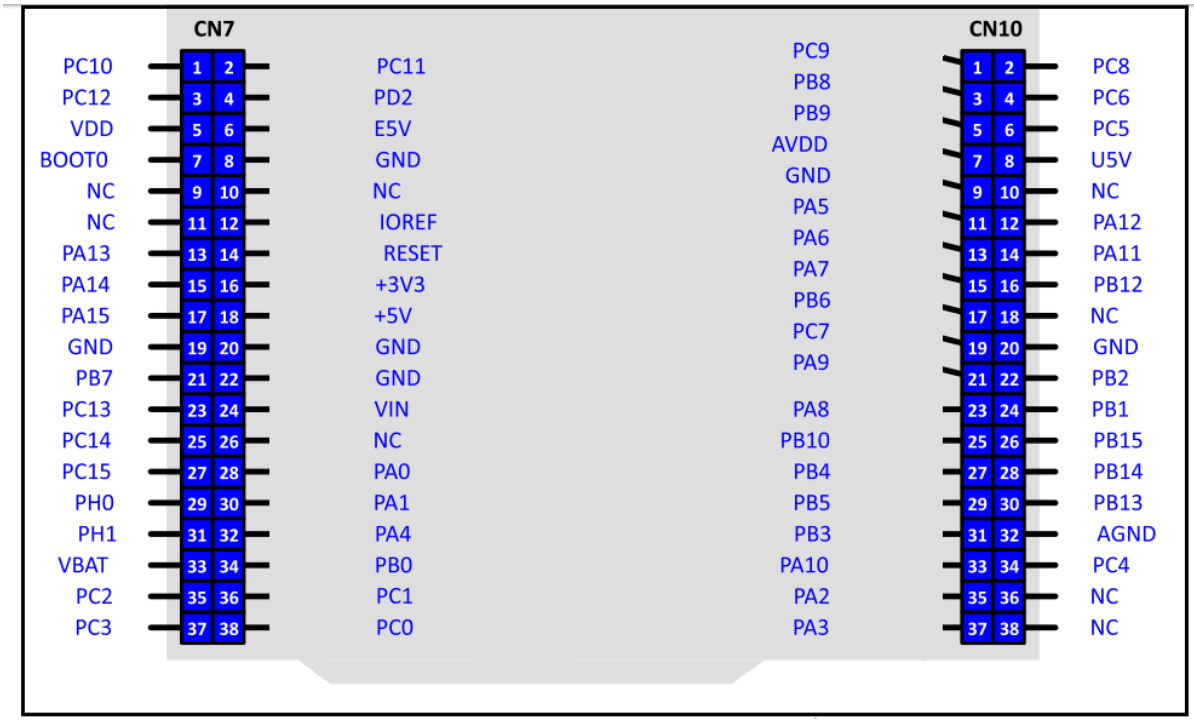
Block	Name	Block base addresses	Size
Main memory	Sector 0	0x0800 0000 - 0x0800 3FFF	16 Kbytes
	Sector 1	0x0800 4000 - 0x0800 7FFF	16 Kbytes
	Sector 2	0x0800 8000 - 0x0800 BFFF	16 Kbytes
	Sector 3	0x0800 C000 - 0x0800 FFFF	16 Kbytes
	Sector 4	0x0801 0000 - 0x0801 FFFF	64 Kbytes
	Sector 5	0x0802 0000 - 0x0803 FFFF	128 Kbytes
	Sector 6	0x0804 0000 - 0x0805 FFFF	128 Kbytes
	Sector 7	0x0806 0000 - 0x0807 FFFF	128 Kbytes
System memory		0x1FFF 0000 - 0x1FFF 77FF	30 Kbytes
OTP area		0x1FFF 7800 - 0x1FFF 7A0F	528 bytes
Option bytes		0x1FFF C000 - 0x1FFF C00F	16 bytes

obr. 3: Organizace vestavěné Flash paměti [3]

Blok Main memory je rozdělen do osmi sektorů, které slouží pro uložení přeloženého kódu z některého vývojového prostředí. Do sektorů je možné zapisovat a číst z nich. Dále podporují Mass Erase, který smaže data všech sektorů, nebo Sector Erase, který vymaže pouze vybrané sektory. Ze System memory bloku se program spustí v bootovacím režimu. Je zde uložen kód potřebný pro správnou činnost bootovacího režimu. Blok OTP area slouží pro jednorázový zápis (vhodné pro uložení například čísla produktu). Podle bloku Option bytes se nastavuje ochrana paměti proti čtení/zápisu, úroveň BOR, softwarový/hardwarový watchdog a reset, když je zařízení v Standby nebo Stop módu. Tento blok paměti lze přepisovat. [3]

4 Propojovací schéma

Piny mikrokontroléru jsou na vývojovém kitu Nucleo 64 vyvedeny na konektory, které lze připojit na samičí Arduino propojku. Takto lze propojit mezi sebou další Nucleo kity nebo Nucleo kit propojit s nepájivým propojovacím polem. [5]



obr. 4: Prodlužovací konektory NUCLEO-F446RE [5]

Master

USART1 : - PA9 USART1_TX
- PA10 USART1_RX

SPI2 : - PB10 SCK
- PC1 MOSI
- PC2 MISO

I2C1 : - PB7 SDA
- PB6 SCL

CAN1 : - PA11 CAN1_RX
- PA12 CAN1_TX

Slave

USART1 : - PA9 - USART1_TX
- PA10 - USART1_RX

SPI2 : - PA5 - SCK
- PA7 - MOSI
- PA6 - MISO
- PA4 - NSS

I2C1 : - PB9 - SDA
- PB6 - SCL

CAN1 : - PB5 - CAN1_RX
- PB13 - CAN1_TX

Komunikace PC

USART1 : - PA2 - USART2_TX
- PA3 - USART2_RX

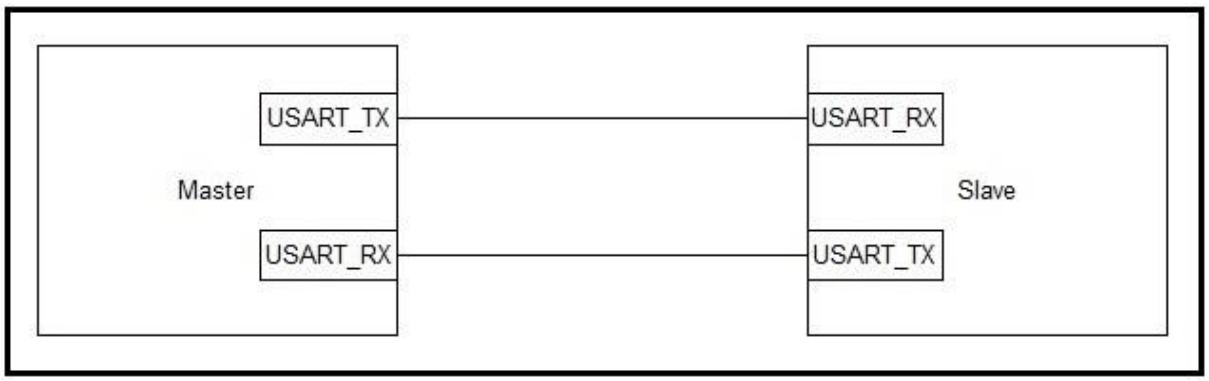
Pro aktivaci bootovacího režimu je nutné odpojit SL-LINK a připojit na pin BOOT0 napájení (pin VDD).

5 Firmware

Firmware pro mikrokontrolér STM32F446RE spravuje jednotlivé periferie, jakými jsou například obvody pro nastavení systémových hodin pro vnitřní sběrnice a periferie, nastavení vývodů komunikačních sběrnic, nastavení komunikačních periferií, nastavení DMA nebo nastavení přerušení. Veškerá tato nastavení byla provedena v CubeMX a toto nastavení bylo vyexportováno jako startup project pro Systém Workbench for STM. Toto nastavení se dá později i ručně měnit, jelikož vše je přístupné (není šifrované) a dá se přepsat. Nastavení se vyexportuje pomocí nastavených inicializačních funkcí HAL knihoven. Pro všechny nastavené periferie jsou dostupné HAL knihovny, usnadňující práci s těmito periferiemi.

5.1 Komunikace USART

Komunikace USART může fungovat v režimu synchronním a asynchronním. Komunikace probíhá mezi dvěma stanicemi, v synchronním režimu je jedna stanice jako Master a druhá jako Slave. Obě stanice jsou mezi sebou propojeny dvěma datovými vodiči v asynchronním režimu, v synchronním režimu je použit ještě jeden pro hodiny SCLK. Ve stanici je datovým výstupem USART_TX a datovým vstupem USART_RX. Datové vodiče mezi sebou propojují USART_TX a USART_RX. V tomto projektu je použit asynchronní režim, proto se dále budeme zabývat pouze tímto režimem. [3]



obr. 5: Propojení Master a Slave stanice u komunikace USART

Pro komunikaci USART je zapotřebí specifikovat přenosovou rychlost, počet přenášených bitů, vybrat paritu a určit počet stop bitů. V dokumentaci pro bootloader (dostupné v příloze: *DVD/dokumentace/AN2606_STM32_system_bootloader*) se udává, že přenosovou rychlost je možné volit v rozmezí 1200 bps až 115200 bps. Přenosová rychlost byla zvolena v tomto intervalu, a to 38400 bps. Počet přenášených bitů je 8. Podle dokumentace by bootloader měl pracovat se sudou paritou, ale s takto nastavenou paritou nebylo možné komunikaci navázat. Nakonec nebyla použita parita žádná. Nastaven byl jeden stop bit a oversampling byl nastaven na 16. Oversampling znamená, kolik vzorků hladiny signálu je vzato pro určení hodnoty jednoho bitu. [5] [6]

5.1.1 HAL knihovny pro komunikaci USART

Použité funkce HAL pro komunikaci USART (v dokumentaci [7] v sekci HAL UART Generic Driver):

- *HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)*
- *HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)*
- *HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)*

huart - Je ukazatel na strukturu *UART_HandleTypeDef*, ve které jsou uloženy konfigurační informace pro UART.

pData – Je ukazatel na osmibitové pole dat určených pro odeslání/příjem.

Size – Požadovaný počet dat pro příjem/odeslání.

Timeout – Maximální čas v milisekundách na provedení funkce.

Všechny tyto funkce vrací hodnotu typu *HAL_StatusTypeDef*, tato hodnota informuje o stavu proběhlé funkce. Například pokud se nepodaří odeslat potřebný počet dat v časovém intervalu, funkce vrátí *HAL_TIMEOUT*, dají se tím indikovat chyby. [7]

Aby bylo možné komunikovat s bootloaderem přes komunikaci USART, je vedle správné konfigurace USART nutná připojovací sekvence, po které zařízení spadne do smyčky, ve které čeká na příkazy určené pro bootloader. [7]

5.1.2 Příkazy komunikace USART

Komunikace je opatřena kontrolními mechanismy. Jedním z nich je kontrolní součet. Pro blok dat odeslaný hostem (Master v této komunikaci) je proveden XOR, XOR je proveden pro všechny byty, takto získaná hodnota je odeslána na konci bloku v jednom bytu. Příjemce datového bloku i s bytem kontrolního součtu provede XOR těchto bytů, takto získaná výsledná hodnota musí být 0x00. Po odeslání paketu hostem příjemce zasílá zprávu, která sděluje příjem tohoto paketu (ACK) nebo vyřazení (NACK): [6]

- ACK = 0x79
- NACK = 0x1F

Po obdržení NACK příkaz končí a vrací hodnotu indikující chybu, pokud příkaz proběhne až do konce, aniž by host obdržel NACK, vrátí hodnotu indikující vykonání tohoto příkazu správně. Zdrojový kód příkazů komunikace USART je dostupný v příloze: *DVD/firmware/BOOT_C/BOOT_C/Inc/usart*.

5.1.2.1 Connect

Je-li mikrokontrolér STM32 v systémovém bootovacím režimu, kód bootloaderu skenuje pin USART_RX a čeká na datový rámec 0x7F. To znamená jeden start bit, 0x7F data bitů, žádná parita a jeden stop bit. Dobu trvání datového rámce počítá Systick časovač. Hodnota vypočtená tímto časovačem je pak použita pro určení přenosové rychlosti vzhledem k systémovým hodinám. Kód poté inicializuje odpovídající sériové rozhraní USART s přenosovou rychlostí vypočtenou z datového rámce 0x7F. Do hostitele je odeslán potvrzovací byt *(0x79)*, tím se signalizuje, že je STM32 připraveno přijímat další příkazy. [6]

Struktura příkazu:

1. Odeslání bytu 0x7F.
2. Příjem ACK nebo NACK, pokud není přijato ani jedno, opětovně se odesílá byt 0x7F a přijímá ACK nebo NACK znovu, dokud není přijat ACK nebo NACK.

5.1.2.2 Get command

Tento příkaz získá z mikrokontroléru STM32 verzi bootloaderu a příkazy, které tento mikrokontrolér podporuje v dané komunikaci. Verze je přijata v jednom bytu, např. 0x10 značí verzi bootloaderu 1.0. Druhy příkazů jsou přijaty v poli bytů. Jeden byt z pole specifikuje svou binární hodnotou druh příkazu. SMT32f446RE s verzí bootloaderu V9.0 podporuje pro komunikaci USART příkazy: [6]

- 0x00 Get
- 0x01 Get Version
- 0x02 Get ID
- 0x11 Read Memory
- 0x21 Go
- 0x31 Write Memory
- 0x43 nebo 0x44 Erase nebo Extended Erase
- 0x63 Write Protect
- 0x73 Write Unprotect
- 0x82 Readout Protect
- 0x92 Readout Unprotect

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x00 (určuje druh příkazu), druhý 0xFF (XOR prvního bytu).
2. Čekání na ACK nebo NACK.
3. Příjem jednoho bytu určujícího počet bytů *N*, které budou následovat.
4. Příjem $N + 1$ bytů, první byt představuje verzi bootloaderu, ostatní byty představují příkazy podporované danou komunikací.
5. Čekání na ACK nebo NACK.

5.1.2.3 Get Version command

Tento příkaz získá z mikrokontroléru STM32 verzi mikrokontroléru, podobně jako příkaz *Get Command*. [6]

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x01 (určuje druh příkazu), druhý 0xFE (XOR prvního bytu).
2. Čekání na ACK nebo NACK
3. Příjem třech bytů, první reprezentuje verzi bootloaderu.
4. Čekáni na ACK nebo NACK

5.1.2.4 Get ID command

Příkaz získá z mikrokontroléru STM32 ID produktu. ID čipu je získáno z dvou bytů, první MSB a druhý LSB. [6]

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x02 (určuje druh příkazu), druhý 0xFD (XOR prvního bytu).
2. Čekání na ACK nebo NACK.
3. Příjem bytu určujícího počet následujících bytů *N*.
4. Příjem *N+1* bytů s ID daného čipu.
5. Čekání na ACK nebo NACK.

5.1.2.5 Read Memory command

Tímto příkazem lze číst z libovolné platné adresy paměti. Po příjmu požadavku na příkaz *Read Memory command* bootloader odešle hostovi ACK a bootloader čeká na čtyři byty adresy a byt s XORem těchto bytů. Je-li adresa platná a kontrolní součet správně, je

odeslán ACK hostovi. Poté bootloader čeká na byt určující počet bytů ke čtení z paměti a XOR tohoto bytu. Počet bytů *N* je v intervalu $0 < N < 256$. Je-li kontrolní součet správně, a jsou-li data z paměti připravena, je odeslán ACK hostovi. Po odeslání potvrzení bootloader pošle hostovi *N* bytů začínajících od hostem odeslané adresy. [6]

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x11 (určuje druh příkazu), druhý 0xEE (XOR prvního bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání pěti bytů: v prvních čtyřech bytech je adresa paměti, z které bude čteno (první byt MSB čtvrtý LSB), a v pátém XOR těchto bytů.
4. Čekání na ACK nebo NACK.
5. Odeslání dvou bytů: první byt určuje *N* bytů ke čtení od zadané adresy a druhý byt je XOR prvního.
6. Čekání na ACK nebo NACK.
7. Příjem *N* bytů dat začínajících na předem zadané adrese.

5.1.2.6 Go command

Příkaz *Go command* slouží ke spuštění kódu od zadané adresy. Po příjmu požadavku na příkaz *Go command* bootloader odešle ACK. Poté čeká na příjem adresy stejně jako v příkaze *Read Memory command*. Pokud je adresa platná, a kontrolní součet správně, kód bootloaderu resetuje registry na základní hodnotu, inicializuje hlavní ukazatel zásobníku a skočí na přijatou adresu + 4. [6]

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x21 (určuje druh příkazu), druhý 0xDE (XOR prvního bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání pěti bytů: v prvních čtyřech bytech je adresa paměti, z které bude čteno (první byt MSB čtvrtý LSB) a v pátém XOR těchto bytů.

4. Čekání na ACK nebo NACK.

5.1.2.7 Write Memory command

Příkazem *Write Memory command* se zapisují data na libovolnou platnou adresu RAM, Flash nebo do bloku Option byte. Obdrží-li bootloader požadavek na příkaz *Write Memory command,* odešle hostovi ACK. Poté čeká na příjem adresy stejně jako v příkaze *Read Memory command*. Pokud je adresa platná, a kontrolní součet správně, odešle hostovi ACK. Host do bootloaderu po obdržení ACK odešle byt, ve kterém uvádí požadovaný počet data bytů pro zápis (*N - 1*), odešle *N - 1* data bytů a XOR všech bytů. Poté přeprogramuje paměť zadanými daty začínajících na zadané adrese. Před zápisem dat do Flash paměti je nutné blok, do kterého chceme zapsat, nejprve vymazat. Příkazem *Write Memmory command* nelze bit paměti z nuly přepsat na jedničku, pouze z jedničky na nulu. Při zápisu do oblasti option byte zařízení provede systém reset a zkonfiguruje se dle nového nastavení. Pokud zápis proběhne správně, bootloader odešle hostu ACK. [6]

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x31 (určuje druh příkazu), druhý 0xCE (XOR prvního bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání pěti bytů: v prvních čtyřech bytech je adresa paměti, z které bude čteno (první byt MSB čtvrtý LSB), a v pátém XOR těchto bytů.
4. Čekání na ACK nebo NACK.
5. Odeslání *N + 2* bytů: v prvním bytu je informace, kolik data bytů se bude do paměti zapisovat *N - 1*, počet data bytů, které chceme zapsat *N,* a byt s XORem těchto bytů.
6. Čekání na ACK nebo NACK.

5.1.2.8 Extended Erase Memory command

Tímto příkazem lze vymazat bloky Flash paměti. Chceme-li do Flash paměti zapisovat, musíme nejprve vymazat tento blok paměti. Obdrží-li bootloader požadavek na příkaz *Erase Memory command*, odešle hostovi ACK. Poté čeká na příjem dvou bytů. Jestliže tyto dva byty jsou hodnoty 0xFF, signalizují *Mass Erase*, to znamená, že dojde k vymazání všech bloků Flash paměti a za těmito byty následuje XOR, v tomto případě 0x00. Bootloader provede *Mass Erase*, a pokud operace proběhne správně, bootloader odešle hostovi ACK. Nejsou-li první dva byty hodnoty 0xFF, je první byt 0x00 a v druhém je požadovaný počet sektorů (*N - 1*) k vymazání. Za těmito byty následují dvojice bytů, určujících, o kolikátý sektor se jedná. Dále následuje tolik dvojic bytů, jako je požadovaný počet sektorů k vymazání. Sektory můžou být řazeny libovolně, jsou určeny pouze dvojicí bytů. Na konci následuje XOR všech bytů. Poté se provede *Erase Sector*, a pokud se operace provede správně, bootloader odešle hostovi ACK. Tato operace, a zejména *Mass Erase*, může trvat několik sekund. [6]

Struktura příkazu Mass Erase:

1. Odeslání dvou bytů: první byt 0x44 (určuje druh příkazu), druhý 0xBB (XOR prvního bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání třech bytů: první dva byty mají hodnotu 0xFF, a poslední 0x00.
4. Čekání na ACK nebo NACK.

Struktura příkazu Erase Sector:

1. Odeslání dvou bytů: první byt 0x44 (určuje druh příkazu), druhý 0xBB (XOR prvního bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání *(N*2) + 3* bytů: první byt je 0x00, druhý byt určuje počet sektorů ke smazání *N - 1*, následující páry bytů určují, jaké sektory mají být smazány, těchto bytů je *N*2*, a na konci následuje byt s XORem předešlých bytů.
4. Čekání na ACK nebo NACK.

5.1.2.9 Write Protect command

Příkaz *Write Protect command* slouží k nastavení ochrany sektorů před zápisem Flash paměti. Obdrží-li bootloader tento příkaz, odešle hostovi ACK. Poté bootloader čeká na byt udávající, kolika sektorům má být nastavena ochrana (*N-1*). Následující byty specifikují, o jaký sektor se jedná, podobně jako u *Erase Sector*. Po odeslání všech bytů se provede ochrana zvolených sektorů, a proběhne-li vše správně, bootloader odešle hostovi ACK a provede systém reset. [6]

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x63 (určuje druh příkazu), druhý 0x9C (XOR prvního bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání *N + 2* bytů: první byt určuje, kolik sektorů má být chráněno před zápisem *N – 1*, následující byty specifikují sektory, které mají být chráněny, těchto bytů je *N*, a na konci následuje byt s XORem předešlých bytů.
4. Čekání na ACK nebo NACK.

5.1.2.10 Write Unprotect command

Příkazem *Write Unprotect command* se zruší ochrana před zápisem všech sektorů paměti Flash. Obdrží-li bootloader příkaz *Write Unprotect command*, odešle hostovi ACK. Po odeslání ACK bootloader zruší ochranu paměti před zápisem a odešle hostovi ACK. Proběhne-li vše správně, na konci *Write Unprotect command* odešle hostovi další ACK a provede systém reset. [6]

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x73 (určuje druh příkazu), druhý 0x8C (XOR prvního bytu).

2. Čekání na 2 * ACK nebo NACK.

5.1.2.11 Readout Protect command

Příkaz *Readout Protect command* aktivuje ochranu paměti Flash před čtením. Obdrží-li bootloader příkaz *Readout Protect command,* odešle hostovi ACK. Po odeslání ACK bootloader nastaví ochranu paměti pře čtením a odešle hostovi ACK. Proběhne-li vše správně, na konci *Readout Protect command* odešle hostovi další ACK a provede systém reset. [6]

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x82 (určuje druh příkazu), druhý 0x7D (XOR prvního bytu).
2. Čekání na 2 * ACK nebo NACK.

5.1.2.12 Readout Unprotect command

Příkazem *Readout Unprotect command* se zruší ochrana před čtením paměti Flash. Obdrží-li bootloader příkaz *Readout Unprotect command,* odešle hostovi ACK. Po odeslání ACK provede bootloader nejprve *Mass Erase,* a teprve potom zruší ochranu paměti před čtením a odešle hostovi ACK. Proběhne-li vše správně, na konci *Readout Unprotect command* odešle hostovi další ACK a provede systém reset. Jelikož vždy musí proběhnout *Mass Erase,* aby bylo zaručeno, že chráněný firmware nikdo ze zařízení nezíská, trvá tento příkaz několik sekund. [6]

Struktura příkazu:

3. Odeslání dvou bytů: první byt 0x92 (určuje druh příkazu), druhý 0x6D (XOR prvního bytu).
4. Čekání na 2 * ACK nebo NACK.

5.2 Komunikace SPI

SPI je synchronní sériová komunikace. Komunikace probíhá mezi jednotlivými stanicemi tak, že jedna stanice je Master. Master stanice je zdrojem hodin pro Slave stanice. Rozhraní SPI podporuje i Multimaster režim. Stanice jsou mezi sebou propojeny třemi datovými vodiči. Vodiče propojují piny SCK, MOSI a MISO. SCK slouží pro přenos hodinového signálu. MOSI slouží jako datový výstup pro Master stanici a datový vstup pro stanici Slave. MISO slouží jako datový vstup pro Master stanici a datový výstup pro stanici Slave. Tuto komunikaci lze nastavit v half-duplex, full-duplex nebo single synchronním režimu. V režimu half-duplex lze buď vysílat, nebo přijímat, ve full-duplex režimu lze zároveň vysílat i přijímat, a v single režimu lze pouze vysílat nebo pouze přijímat. V režimu half-duplex a v single režimu je pin MISO u Mastera a MOSI u Slave volný. [3]

obr. 6: Propojení Master a Slave stanice u komunikace SPI

Pro komunikaci SPI je zapotřebí nastavit stanici jako Master ve full-duplex režimu. Počet přenášených bytů je nutné nastavit na 8 tak, že první přenášený byt bude MSB s přenosovou rychlostí 8 Mbit/s. [5] [8]

5.2.1 HAL knihovny pro komunikaci SPI

Použité funkce HAL pro komunikaci SPI (v dokumentaci [7] v sekci HAL SPI Generic Driver):

- *HAL_StatusTypeDef HAL_SPI_Init (SPI _HandleTypeDef * hspi)*

- *HAL_StatusTypeDef HAL_SPI_Transmit (SPI _HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)*
- *HAL_StatusTypeDef HAL_SPI_Receive (SPI _HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)*
- *HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)*

hspi - Je ukazatel na strukturu *SPI_HandleTypeDef*, ve které jsou uloženy konfigurační informace pro SPI.

pData – Je ukazatel na osmibitové pole dat určených pro odeslání/příjem.

pRxData – Je ukazatel na osmibitové pole dat určených pro příjem.

pTxData – Je ukazatel na osmibitové pole dat určených pro odeslání.

Size – Požadovaný počet dat pro příjem/odeslání.

Timeout – Maximální čas v milisekundách na provedení funkce.

Všechny tyto funkce vrací hodnotu typu *HAL_StatusTypeDef*, tato hodnota informuje o stavu proběhlé funkce. Například pokud se nepodaří odeslat potřebný počet dat v časovém intervalu, funkce vrátí *HAL_TIMEOUT*, dají se tím indikovat chyby. [7]

Aby bylo možné komunikovat s bootloaderem přes komunikaci SPI, je vedle správné konfigurace SPI nutná připojovací sekvence, po které zařízení spadne do smyčky, ve které čeká na příkazy určené pro bootloader. [7]

5.2.2 Příkazy komunikace SPI

Kontrolní mechanizmy pro SPI jsou obdobné jako pro komunikaci USART v kapitole 5.1.2 s tím rozdílem, že po obdržení ACK nebo NACK host do bootloaderu odesílá potvrzení o přijetí ACK. Příkazy pro komunikaci SPI jsou obdobné jako pro komunikaci USART v kapitole 5.1.2, proto bude u jednotlivých příkazů uvedena pouze struktura příkazu. Pouze

příkaz *Connect* se liší, a proto bude uveden celý. Zdrojový kód příkazů komunikace SPI je dostupný v příloze: *DVD/firmware/BOOT_C/BOOT_C/Inc/spi*. [8]

5.2.2.1 Connect

Je-li mikrokontrolér STM32 v systémovém bootovacím režimu, kód bootloaderu skenuje pin SPI_MOSI a čeká na synchronizační byte 0x5A. Obdrží-li bootloader synchronizační byt, synchronizuje se na danou přenosovou rychlost, a proběhne-li vše bez chyby, začne bootloader přijímat další příkazy. Chceme-li z bootloaderu obdržet data, musíme nejprve bootloaderu odeslat prázdný byt pro příjem dat 0x00. [8]

Struktura příkazu:

1. Odeslání dvou bytů: prvního synchronizačního bytu 0x5A a druhého vloženého bytu 0x00.
2. Příjem ACK nebo NACK, pokud není přijato ani jedno, opětovně se odesílá byt 0x7F a přijímá ACK nebo NACK znovu, dokud není přijat ACK nebo NACK.
3. Odeslání ACK do bootloaderu.

5.2.2.2 Get command

Mikrokontrolér SMT32f446RE podporuje pro komunikaci SPI příkazy: [8]

- 0x00 Get
- 0x01 Get Version
- 0x02 Get ID
- 0x11 Read Memory
- 0x21 Go
- 0x31 Write Memory
- 0x44 Erase

- 0x63 Write Protect
- 0x73 Write Unprotect
- 0x82 Readout Protect
- 0x92 Readout Unprotect

Struktura příkazu:

1. Odeslání třech bytů: první byt 0x5A (uvozuje začátek rámce), druhý byt 0x00 (určuje druh příkazu), třetí 0xFF (XOR druhého bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání ACK do bootloaderu.
4. Odeslání prázdného bytu 0x00.
5. Příjem jednoho bytu, který určuje počet bytů *N*, které budou následovat, a zároveň odeslání prázdného bytu 0x00.
6. Příjem $N + 1$ bytů, první byt představuje verzi bootloaderu, ostatní byty představují podporované příkazy danou komunikací.
7. Čekání na ACK nebo NACK.
8. Odeslání ACK do bootloaderu.

5.2.2.3 Get Version command

Struktura příkazu:

1. Odeslání třech bytů: první byt 0x5A, druhý byt 0x01 (určuje druh příkazu), třetí 0xFE (XOR druhého bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání ACK do bootloaderu.
4. Odeslání prázdného bytu 0x00.
5. Příjem jednoho bytu reprezentujícího verzi bootloaderu.
6. Čekáni na ACK nebo NACK.
7. Odeslání ACK do bootloaderu.

5.2.2.4 Get ID command

Struktura příkazu:

1. Odeslání třech bytů: první byt 0x5A, druhý byt 0x02 (určuje druh příkazu), třetí 0xFD (XOR druhého bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání ACK do bootloaderu.
4. Odeslání prázdného bytu 0x00.
5. Příjem bytu určujícího počet následujících bytů *N* a zároveň odeslání prázdného bytu 0x00.
6. Příjem *N+1* bytů s ID daného čipu.
7. Čekání na ACK nebo NACK.
8. Odeslání ACK do bootloaderu.

5.2.2.5 Read Memory command

Struktura příkazu:

1. Odeslání třech bytů: první byt 0x5A, druhý byt 0x11 (určuje druh příkazu), třetí 0xEE (XOR druhého bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání ACK do bootloaderu.
4. Odeslání pěti bytů: v prvních čtyřech bytech je adresa paměti, z které bude čteno (první byt MSB čtvrtý LSB), a v pátém XOR těchto bytů.
5. Čekání na ACK nebo NACK.
6. Odeslání ACK do bootloaderu.
7. Odeslání dvou bytů: první byt určuje *N* bytů ke čtení od zadané adresy a druhý byt je XOR prvního.
8. Čekání na ACK nebo NACK.
9. Odeslání ACK do bootloaderu.
10. Odeslání prázdného bytu 0x00.

11. Příjem *N* bytů dat začínajících na předem zadané adrese.

5.2.2.6 Go command

Struktura příkazu:

1. Odeslání třech bytů: první byt 0x5A, druhý byt 0x21 (určuje druh příkazu), třetí 0xDE (XOR druhého bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání ACK do bootloaderu.
4. Odeslání pěti bytů: v prvních čtyřech bytech je adresa paměti, z které bude čteno (první byt MSB čtvrtý LSB) a v pátém XOR těchto bytů.
5. Čekání na ACK nebo NACK.
6. Odeslání ACK do bootloaderu.

5.2.2.7 Write Memory command

Struktura příkazu:

1. Odeslání třech bytů: první byt 0x5A, druhý byt 0x31 (určuje druh příkazu), třetí 0xCE (XOR druhého bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání ACK do bootloaderu.
4. Odeslání pěti bytů: v prvních čtyřech bytech je adresa paměti, z které bude čteno (první byt MSB čtvrtý LSB), a v pátém XOR těchto bytů.
5. Čekání na ACK nebo NACK.
6. Odeslání ACK do bootloaderu.
7. Odeslání $N + 2$ bytů: v prvním bytu je informace, kolik data bytů se bude do paměti zapisovat $N - 1$, počet data bytů, které chceme zapsat N, a byt s XORem těchto bytů.
8. Čekání na ACK nebo NACK.

9. Odeslání ACK do bootloaderu.

5.2.2.8 Erase Memory command

Struktura příkazu Mass Erase:

1. Odeslání třech bytů: první byt 0x5A, druhý byt 0x44 (určuje druh příkazu), třetí 0xBB (XOR druhého bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání ACK do bootloaderu.
4. Odeslání třech bytů: první dva byty mají hodnotu 0xFF a poslední 0x00.
5. Čekání na ACK nebo NACK.
6. Odeslání ACK do bootloaderu.

Struktura příkazu Erase Sector:

1. Odeslání třech bytů: první byt 0x5A, druhý byt 0x44 (určuje druh příkazu), třetí 0xBB (XOR druhého bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání ACK do bootloaderu.
4. Odeslání třech bytů: první byt je 0x00, druhý byt určuje počet sektorů ke smazání *N – 1* a třetí byt je stejný jako druhý
5. Čekání na ACK nebo NACK.
6. Odeslání ACK do bootloaderu.
7. Odeslání *(N*2) + 1* bytů: tyto byty určují, jaké sektory mají být smazány, těchto bytů je *N*2*, a na konci následuje byt s XORem předešlých bytů.
8. Čekání na ACK nebo NACK.
9. Odeslání ACK do bootloaderu.

5.2.2.9 Write Protect command

Struktura příkazu:

1. Odeslání třech bytů: první byt 0x5A, druhý byt 0x63 (určuje druh příkazu), třetí 0x9C (XOR druhého bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání ACK do bootloaderu.
4. Odeslání dvou bytů: první byt určuje, kolik sektorů má být chráněno před zápisem *N – 1*, a druhý byt je bitovou negací prvního bytu.
5. Čekání na ACK nebo NACK.
6. Odeslání ACK do bootloaderu.
7. Odeslání *N + 1* bytů: tyto byty specifikují sektory, které mají být chráněny, těchto bytů je *N*, a na konci následuje byt s XORem předešlých bytů.
8. Čekání na ACK nebo NACK.
9. Odeslání ACK do bootloaderu.

5.2.2.10 Write Unprotect command

Struktura příkazu:

1. Odeslání třech bytů: první byt 0x5A, druhý byt 0x73 (určuje druh příkazu), třetí 0x8C (XOR druhého bytu).
2. Čekání na 2 * ACK nebo NACK.
3. Odeslání ACK do bootloaderu.

5.2.2.11 Readout Protect command

Struktura příkazu:

1. Odeslání třech bytů: první byt 0x5A, druhý byt 0x82 (určuje druh příkazu), třetí 0x7D (XOR druhého bytu).
2. Čekání na 2 * ACK nebo NACK.
3. Odeslání ACK do bootloaderu.

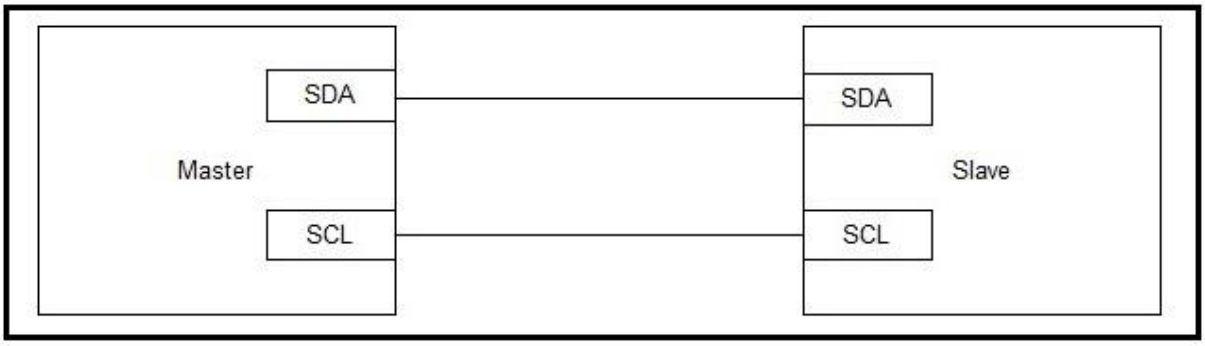
5.2.2.12 Readout Unprotect command

Struktura příkazu:

1. Odeslání třech bytů: první byt 0x5A, druhý byt 0x92 (určuje druh příkazu), třetí 0x6D (XOR druhého bytu).
2. Čekání na 2 * ACK nebo NACK.
3. Odeslání ACK do bootloaderu.

5.3 Komunikace I2C

I2C je synchronní sériová komunikace. Touto komunikační sběrnicí lze propojit více stanic. Jedna stanice je vždy Master a zbylé stanice jsou Slave. Možný je také Multimaster režim, kdy si stanice mezi sebou předávají informaci, jaká stanice má být Master, ale vždy je možné, aby byla v danou chvíli pouze jedna stanice Master. Na propojení stanic jsou třeba pouze dva vodiče. Jeden vodič SCK slouží pro přenos hodinového signálu, který generuje master stanice. Druhý vodič SDA slouží pro přenos dat, přenos dat iniciuje vždy Master stanice. K těmto vodičům jsou připojeny pull-up rezistory, na obrázku 7 nejsou vyznačeny, jelikož jsou integrovány přímo na čipu, není nutné je přidávat externě. Stanice mají nastavenou adresu, podle této adresy stanice pozná, zda jí patří přenášená data. Data jsou přenášena po jednom bytu, na konci tohoto bytu příjemce dat generuje ACK. Přenosová rychlost je 100 kHz pro Standard Speed režim a 400 kHz pro Fast Speed režim. [3]



obr. 7: Propojení Master a Slave stanice u komunikace I2C

Pro komunikaci I2C je zapotřebí specifikovat přenosovou rychlost a délku adresy. Pro komunikaci byl dle dokumentace zvolen Fast Speed režim, to znamená přenosovou rychlost 400 kHz s délkou adresy 7 bitů. Délka adresy Slave (bootloaderu) je 7 bitů. Fyzicky se přenáší 8 bitů, ale poslední LSB určuje, zda se jedná o zápis či o čtení. [5]

5.3.1 HAL knihovny pro komunikaci I2C

Použité funkce HAL pro komunikaci I2C (v dokumentaci [7] v sekci HAL I2C Generic Driver):

- *HAL_StatusTypeDef HAL_I2C_Init (I2C_HandleTypeDef * hi2c)*
- *HAL_StatusTypeDef HAL_I2C_Master_Transmit (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)*
- *HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)*

hi2c - Je ukazatel na strukturu *I2C_HandleTypeDef*, ve které jsou uloženy konfigurační informace pro I2C.

DevAddress - Je adresa stanice, která má odesílat/přijímat

pData – Je ukazatel na osmibitové pole dat určených pro odeslání/příjem.

Size – Požadovaný počet dat pro příjem/odeslání.

Timeout – Maximální čas v milisekundách na provedení funkce.

Funkci *HAL_I2C_Master_Transmit* bylo nutné doplnit zpožďovacím cyklem mezi odesílanými byty. Všechny tyto funkce vrací hodnotu typu *HAL_StatusTypeDef*, tato hodnota informuje o stavu proběhlé funkce. Například pokud se nepodaří odeslat potřebný počet dat v časovém intervalu, funkce vrátí *HAL_TIMEOUT*, dají se tím indikovat chyby. [7]

Aby bylo možné komunikovat s bootloaderem přes komunikaci I2C, je vedle správné konfigurace I2C nutná připojovací sekvence, po které zařízení spadne do smyčky, ve které čeká na příkazy určené pro bootloader. [7]

5.3.2 Příkazy komunikace I2C

Kontrolní mechanizmy pro I2C jsou obdobné jako pro komunikaci USART v kapitole 5.1.2. Příkazy pro komunikaci I2C jsou obdobné, jako pro komunikaci USART v kapitole 5.1.2, proto bude u jednotlivých příkazů uvedena pouze struktura příkazu. Tato komunikace podporuje No-Stretch příkazy, jsou to zpravidla déle trvající příkazy. Tyto příkazy jsou obdobné jako standartní s tím rozdílem, že neblokují sběrnici. Na konci No-Stretch příkazu na dotaz hosta vrací bootloader stav Busy (0x76), dokud se příkaz nedostal na konec nebo dokud nedošlo k chybě. Příkaz *Connect* se liší, a proto bude uveden celý. Zdrojový kód příkazů komunikace I2C je dostupný v příloze: *DVD/firmware/BOOT_C/BOOT_C/Inc/i2c*. [9]

5.3.2.1 Connect

Je-li mikrokontrolér STM32 v systémovém bootovacím režimu, kód bootloaderu skenuje pin SDA a čeká na byt svojí adresy 0b0111100x. Obdrží-li bootloader tuto adresu, začne přijímat další příkazy. Při požadavku na příjem nebo odeslání je zadána tato adresa, proto není nutné zavádět pro připojení přihlašovací příkaz. [9]

5.3.2.2 Get command

Mikrokontrolér SMT32f446RE podporuje pro komunikaci I2C příkazy: [8]

- 0x00 Get
- 0x01 Get Version
- 0x02 Get ID
- 0x11 Read Memory
- 0x21 Go
- 0x31 Write Memory
- 0x32 No-Stretch Write Memory
- 0x44 Erase
- 0x45 No-Stretch Erase
- 0x63 Write Protect
- 0x64 No-Stretch Write Protect
- 0x73 Write Unprotect
- 0x74 No-Stretch Write Unprotect
- 0x82 Readout Protect
- 0x83 No-Stretch Readout Protect
- 0x92 Readout Unprotect
- 0x93 No-Stretch Readout Unprotect

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x00, druhý 0xFF (XOR prvního bytu).
2. Čekání na ACK nebo NACK.
3. Příjem 20 bytů, první byt představuje verzi bootloaderu, ostatní byty představují podporované příkazy danou komunikací.
4. Čekání na ACK nebo NACK.

5.3.2.3 Get Version command

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x01, druhý 0xFE (XOR prvního bytu).
2. Čekání na ACK nebo NACK
3. Příjem jednoho bytu reprezentujícího verzi bootloaderu.
4. Čekáni na ACK nebo NACK

5.3.2.4 Get ID command

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x02, druhý 0xFD (XOR prvního bytu).
2. Čekání na ACK nebo NACK.
3. Příjem třech bytů s ID daného čipu.
4. Čekání na ACK nebo NACK.

5.3.2.5 Read Memory command

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x11, druhý 0xEE (XOR prvního bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání pěti bytů: v prvních čtyřech bytech je adresa paměti, z které bude čteno (první byt MSB, čtvrtý LSB), a v pátém XOR těchto bytů.
4. Čekání na ACK nebo NACK.
5. Odeslání dvou bytů: první byt určuje *N* bytů ke čtení od zadané adresy a druhý byt je XOR prvního.
6. Čekání na ACK nebo NACK.
7. Příjem *N* bytů dat začínajících na předem zadané adrese.

5.3.2.6 Go command

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x21 (určuje druh příkazu), druhý 0xDE (XOR prvního bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání pěti bytů: v prvních čtyřech bytech je adresa paměti, z které bude čteno (první byt MSB, čtvrtý LSB), a v pátém XOR těchto bytů.
4. Čekání na ACK nebo NACK.

5.3.2.7 No-Stretch Write Memory command

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x32, druhý 0xCD (XOR prvního bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání pěti bytů: v prvních čtyřech bytech je adresa paměti, z které bude čteno (první byt MSB, čtvrtý LSB), a v pátém XOR těchto bytů.
4. Čekání na ACK nebo NACK.
5. Odeslání $N + 2$ bytů: v prvním bytu je informace, kolik data bytů se bude do paměti zapisovat $N - 1$, počet data bytů, které chceme zapsat N, a byt s XORem těchto bytů.
6. Čekání na ACK nebo NACK.

5.3.2.8 No-Stretch Erase Memory command

Struktura příkazu Mass Erase:

1. Odeslání dvou bytů: první byt 0x45, druhý 0xBA (XOR prvního bytu).
2. Čekání na ACK nebo NACK.

3. Odeslání třech bytů: první dva byty mají hodnotu 0xFF a poslední 0x00.
4. Čekání na ACK nebo NACK.

Struktura příkazu Erase Sector:

1. Odeslání dvou bytů: první byt 0x45, druhý 0xBA (XOR prvního bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání třech bytů: první byt je 0x00, druhý byt určuje počet sektorů ke smazání *N – 1* a třetí byt je stejný jako druhý
4. Čekání na ACK nebo NACK.
5. Odeslání *(N*2) + 1* bytů: tyto byty určují, jaké sektory mají být smazány, těchto bytů je *N*2,* a na konci následuje byt s XORem předešlých bytů.
6. Čekání na ACK nebo NACK.

5.3.2.9 No-Stretch Write Protect command

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x64, druhý 0x9B (XOR prvního bytu).
2. Čekání na ACK nebo NACK.
3. Odeslání dvou bytů: první byt určuje, kolik sektorů má být chráněno před zápisem *N – 1*, a druhý byt je bitovou negací prvního bytu.
4. Čekání na ACK nebo NACK.
5. Odeslání *N + 1* bytů: tyto byty specifikují sektory, které mají být chráněny, těchto bytů je *N* a na konci následuje byt s XORem předešlých bytů.
6. Čekání na ACK nebo NACK.

5.3.2.10 No-Stretch Write Unprotect command

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x74, druhý 0x8B (XOR prvního bytu).

2. Čekání na 2 * ACK nebo NACK.

5.3.2.11 Readout Protect command

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x83, druhý 0x7C (XOR prvního bytu).
2. Čekání na 2 * ACK nebo NACK.

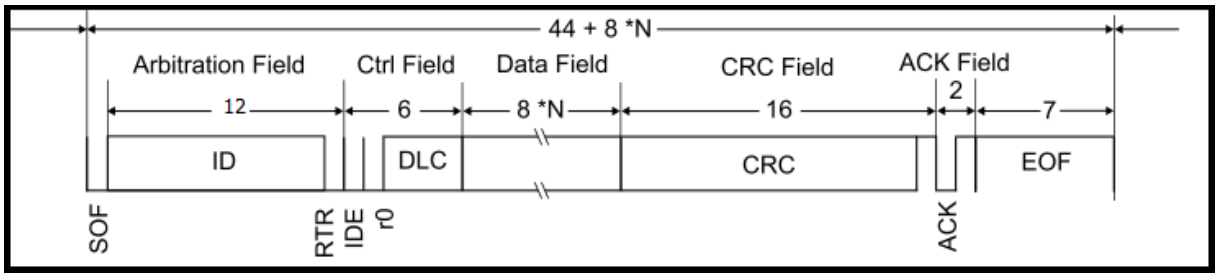
5.3.2.12 Readout Unprotect command

Struktura příkazu:

1. Odeslání dvou bytů: první byt 0x93, druhý 0x6C (XOR prvního bytu).
2. Čekání na 2 * ACK nebo NACK.

5.4 Komunikace CAN

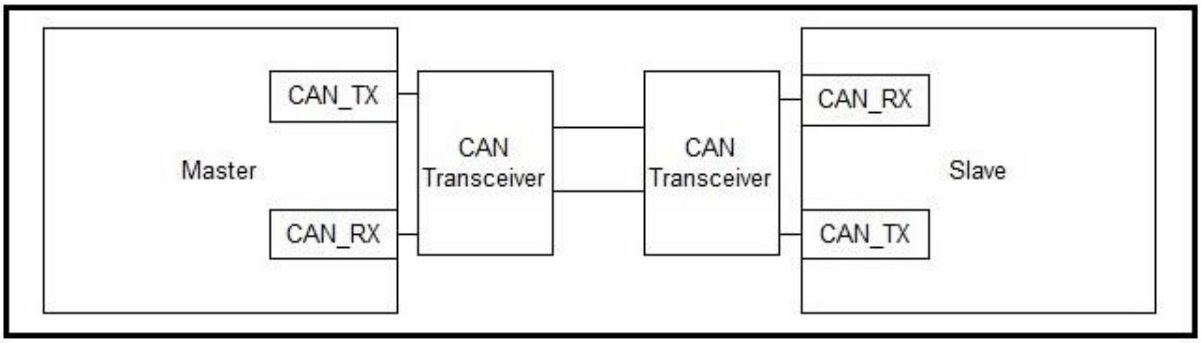
CAN je asynchronní sériová komunikace. Sběrnice, ke které jsou stanice připojeny, se skládá ze dvou vodičů. Jednotlivé stanice jsou na sběrnici připojeny přes CAN Transcievery. Doba trvání jednoho bytu se nastaví pomocí třech segmentů. Segmenty se skládají z časových kvant (odvozených z hodin sběrnice). V prvním segmentu se očekává změna bytu, segment má fixní délku jednoho časového kvanta. Druhý segment má nastavitelnou délku jednoho až šestnácti časových kvant a třetí segment má délku jednoho až osmi časových kvant. Trvání jednoho bytu odpovídá přenosové rychlosti. Jednotlivé stanice si informaci posílají po sběrnici pomocí třech typů CAN rámců. Data Frame, ve kterém se předávají data, může být se standartním identifikátorem nebo rozšířeným identifikátorem. Remote Frame slouží jako požadavek na přenos a Error Frame je chybový rámec informující o chybě v přenosu. Dále bude pro pochopení komunikace blíže uveden Data Frame se standartním identifikátorem. [3]



obr. 8: Data Frame [3]

SOF bit uvozuje začátek zprávy, jeho hodnota je dominantní 0. V ID je 11 bitový identifikátor. Bit RTR rozhoduje, zda se jedná o Data Frame nebo Remote Frame, pro data frame je tento bit v 0. Bit ID rozhoduje, zda je Data Frame se standartním identifikátorem nebo s rozšířeným, pro standartní identifikátor je tento bit v 0. Hodnota bitů DLC rozhoduje, kolik je přenášeno data bytů. Dále následuje pole až osmi data bytů dle hodnoty DLC. V CRC se přenáší kontrolní součet. Následuje ACK, první bit 0 a druhý 1. Rámec je ukončen sedmi recesivními bity 1. [3]

Pro komunikaci s bootloaderem je nastavená přenosová rychlost 125 kbps. Při komunikaci host odesílá v ideálním případě pouze datové rámce s 11 bitovým identifikátorem, při poruše i rámec chybový. Obdobné datové rámce také přijímá. [5]



obr. 9: Propojení Master a Slave stanice u komunikace CAN

5.4.1 HAL knihovny a vytvořené funkce pro komunikaci CAN

Použité funkce HAL pro komunikaci CAN (v dokumentaci [7] v sekci HAL CAN Generic Driver) a dodatečně vytvořené funkce:

- *HAL_StatusTypeDef HAL_CAN_Init (CAN_HandleTypeDef * hcan)*
- *HAL_StatusTypeDef HAL_CAN_ConfigFilter (CAN_HandleTypeDef * hcan, CAN_FilterConfTypeDef * sFilterConfig)*
- *HAL_StatusTypeDef HAL_CAN_Transmit (CAN_HandleTypeDef * hcan, uint32_t Timeout)*
- *HAL_StatusTypeDef HAL_CAN_Receive (CAN_HandleTypeDef * hcan, uint8_t FIFONumber, uint32_t Timeout)*
- *CAN_Filter(CAN_HandleTypeDef *hcan, uint32_t identifier)*
- *CAN_TX(CAN_HandleTypeDef *hcan, CAN_data_str *CANdata)*
- *CAN_RX(CAN_HandleTypeDef *hcan, CAN_data_str *CANdatar)*

hcan - Je ukazatel na strukturu *CAN_HandleTypeDef*, ve které jsou uloženy konfigurační informace pro CAN.

sFilterConfig - Je ukazatel na strukturu *CAN_FilterConfTypeDef*, ve které jsou uloženy konfigurační informace pro filtr.

FIFONumber – Určuje paměť FIFO, do které se mají přijatá data uložit.

Timeout – Maximální čas v milisekundách na provedení funkce.

identifier – Jedenáctibitový identifikátor.

CANdata - Je ukazatel na osmibitové pole dat určených pro odeslání.

CANdatar - Je ukazatel na osmibitové pole dat určených pro příjem.

Všechny funkce HAL vrací hodnotu typu *HAL_StatusTypeDef*, tato hodnota informuje o stavu proběhlé funkce. Například pokud se nepodaří odeslat potřebný počet dat v časovém intervalu, funkce vrátí *HAL_TIMEOUT*, dají se tím indikovat chyby. [7]

Pro nastavení filtru byla vytvořena funkce *CAN_Filter*, která nastaví *sFilterConfig* a aktivuje filtr funkcí *HAL_CAN_ConfigFilter* dle zadaného identifikátoru. [7]

Pro odeslání dat byla vytvořena funkce *CAN_TX*, která nastaví strukturu *pTxMsg* specifikující přenos a aktivuje přenos funkcí *HAL_CAN_Transmit*. [7]

Pro příjem dat byla vytvořena funkce *CAN_RX*, která aktivuje příjem funkcí *HAL_CAN_Receive* a nastaví pole dat *CANdatar* podle struktury *pRxMsg*. [7]

Zdrojový kód vytvořených funkcí je dostupný v příloze: *DVD/firmware/BOOT_C/BOOT_C/Inc/can*.

Aby bylo možné komunikovat s bootloaderem přes komunikaci CAN, je vedle správné konfigurace CAN nutná připojovací sekvence, po které zařízení spadne do smyčky, ve které čeká na příkazy určené pro bootloader. [7]

5.4.2 Příkazy komunikace CAN

Každému příkazu odpovídá jiné ID. Datové rámce odesílané a přijímané v rámci příkazu mají vždy odpovídající ID příkazu. Proto se u každého příkazu nastavuje filtr na ID daného příkazu.

Komunikaci s bootloaderem po sběrnici CAN není nutné opatřovat kontrolním součtem, jelikož kontrolní součet je součástí každého z odesílaných rámců. Na konci paketů bootloader odesílá jednobytový datový rámec, ve kterém se posílá ACK = 0x79, jsou-li přijatá data platná, nebo NACK = 0x1F, pokud přijatá data platná nejsou nebo se mu data přijmout nepovede. Příkazy pro komunikaci s bootloaderem po sběrnici CAN se od příkazů vzorové komunikace s USART v kapitole 5.1.2 v mnoha ohledech liší, proto budou příkazy uvedeny celé. Zdrojový kód příkazů komunikace CAN je dostupný v příloze: *DVD/firmware/BOOT_C/BOOT_C/Inc/can*. [10]

5.4.2.1 Connect

Je-li mikrokontrolér STM32 v systémovém bootovacím režimu, kód bootloaderu skenuje pin CAN_RX a čeká na datový rámec s identifikátorem 0x79. Podle přijatého rámce se synchronizují hodiny CAN periferie bootloaderu. Do hostitele je odeslán potvrzovací datový rámec s identifikátorem 0x79 a jedním data bytem 0x79. Tím se signalizuje, že je STM32 připraven přijímat další příkazy komunikací CAN. [10]

Struktura příkazu:

1. Odeslání datového rámce s identifikátorem 0x79 a žádným data bytem.
2. Příjem ACK nebo NACK, pokud není přijato ani jedno, opětovně se odesílá datový rámec s identifikátorem 0x79 a přijímá ACK nebo NACK znovu, dokud není přijat ACK nebo NACK.

5.4.2.2 Get command

Tímto příkazem se získá verze bootloaderu a seznam podporovaných příkazů. Verze je přijata v jednom datovém rámci, s jedním data bytem. Podporované příkazy jsou odesílány postupně, pro každý je určen jeden datový rámec, s jedním data bytem. SMT32f446RE s verzí bootloaderu V9.0 podporuje pro komunikaci CAN příkazy: [10]

- 0x00 Get
- 0x01 Get Version
- 0x02 Get ID
- 0x03 Speed
- 0x11 Read Memory
- 0x21 Go
- 0x31 Write Memory
- 0x43 Erase
- 0x63 Write Protect

- 0x73 Write Unprotect
- 0x82 Readout Protec
- 0x92 Readout Unprotect

Struktura příkazu:

1. Odeslání datového rámce s identifikátorem 0x00 a bez data bytů.
2. Čekání na ACK nebo NACK.
3. Příjem datového rámce s jedním data bytem určujícím počet rámců *N*, které budou následovat.
4. Příjem *N + 1* datových rámců s jedním data bytem, první datový rámec představuje verzi bootloaderu, ostatní datové rámce představují příkazy podporované danou komunikací.
5. Čekání na ACK nebo NACK.

5.4.2.3 Get Version command

Tímto příkazem se z mikrokontroléru získá verze bootloaderu. Verze bootloaderu se dá získat i příkazem *Get command.* [10]

Struktura příkazu:

1. Odeslání datového rámce s identifikátorem 0x01 a bez data bytů.
2. Čekání na ACK nebo NACK
3. Příjem jednoho datového rámce s jedním data bytem reprezentujícím verzi bootloaderu.
4. Čekáni na ACK nebo NACK

5.4.2.4 Get ID command

Tímto příkazem se z mikrokontroléru získá ID čipu. ID čipu je získáno z dvou bytů jednoho datového rámce, první MSB a druhý LSB. [10]

Struktura příkazu:

1. Odeslání datového rámce s identifikátorem 0x02 a bez data bytů.
2. Čekání na ACK nebo NACK.
3. Příjem jednoho datového rámce s dvěma data byty (reprezentují ID).
4. Čekání na ACK nebo NACK.

5.4.2.5 Read Memory command

Tímto příkazem lze číst z libovolné platné adresy paměti. Po příjmu požadavku na příkaz *Read Memory command* bootloader odešle hostovi ACK a bootloader čeká na jeden datový rámec obsahující čtyři byty adresy a pátý byt, ve kterém je počet bytů, které mají být od zadané adresy přečteny. Počet bytů N je v intervalu $0 < N < 256$. Je-li adresa platná je odeslán ACK hostovi. Po odeslání potvrzení bootloader pošle hostovi N bytů začínajících od hostem odeslané adresy. Tyto byty jsou odesílány hostovi v datových rámcích až o osmi bytech. [10]

Struktura příkazu:

1. Odeslání datového rámce s identifikátorem 0x11 a bez data bytů.
2. Čekání na ACK nebo NACK.
3. Odeslání jednoho datového rámce s pěti byty.
4. Čekání na ACK nebo NACK.
5. Příjem $N/8$ datových rámců s osmi byty a jedním datovým rámcem se zbytkem, pokud nějaký zbytek je.

5.4.2.6 Go command

Příkaz *Go command* slouží ke spuštění kódu od zadané adresy. Po příjmu požadavku na příkaz *Go command* bootloader odešle ACK. Poté čeká na příjem datového rámce se

čtyřmi byty s adresou. Pokud je adresa platná, kód bootloaderu resetuje registry na základní hodnotu, inicializuje hlavní ukazatel zásobníku a skočí na přijatou adresu + 4. [10]

Struktura příkazu:

1. Odeslání datového rámce s identifikátorem 0x21 a bez data bytů.
2. Čekání na ACK nebo NACK.
3. Odeslání jednoho datového rámce se čtyřmi byty.
4. Čekání na ACK nebo NACK.

5.4.2.7 Write Memory command

Příkazem *Write Memory command* se zapisují data na libovolnou platnou adresu RAM, Flash nebo do bloku Option byte. Obdrží-li bootloader požadavek na příkaz *Write Memory command,* odešle hostovi ACK. Poté čeká na příjem adresy a počtu data bytů, které se do paměti budou zapisovat, stejně jako v příkaze *Read Memory command*. Pokud je adresa platná, odešle hostovi ACK. Poté se začnou odesílat data pro zápis do paměti jako datové rámce s až osmi data byty. Před zápisem dat do Flash paměti je nutné blok, do kterého chceme zapsat, nejprve vymazat. Příkazem *Write Memmory command* nelze bit paměti z nuly přepsat na jedničku, pouze z jedničky na nulu. Při zápisu do oblasti option byte zařízení provede systém reset a zkonfiguruje se dle nového nastavení. Pokud zápis proběhne správně, bootloader odešle hostu ACK. [10]

Struktura příkazu:

1. Odeslání datového rámce s identifikátorem 0x31 a bez data bytů.
2. Čekání na ACK nebo NACK.
3. Odeslání jednoho datového rámce s pěti byty.
4. Čekání na ACK nebo NACK.
5. Odeslání *N/8* datových rámců s osmi byty a jedním datovým rámcem se zbytkem, pokud nějaký zbytek je.
6. Čekání na ACK nebo NACK.

5.4.2.8 Erase Memory command

Tímto příkazem lze vymazat bloky Flash paměti. Chceme-li do Flash paměti zapisovat, musíme nejprve vymazat tento blok paměti. Obdrží-li bootloader požadavek na příkaz *Erase Memory command,* odešle hostovi ACK. Poté čeká na příjem datového rámce. Jestliže datový rámec má jeden byt hodnoty 0xFF, signalizuje to *Mass Erase*, to znamená, že dojde k vymazání všech bloků Flash paměti. Bootloader provede *Mass Erase* a pokud operace proběhne správně, bootloader odešle hostovi 2*ACK. Není-li datový rámec s jedním data bytem hodnoty 0xFF, provádí se *Erase Sector*. Bootloader čeká na datový rámec s jedním data bytem. Hodnota tohoto data bytu určuje počet sektorů k erase. Po přijetí tohoto datového rámce bootloader odešle ACK. Poté se odešle jeden datový rámec, který určí, jaké sektory budou vymazány. Počet odeslaných bytů odpovídá počtu sektorů a data byty tohoto rámce určují, o jaké sektory se bude jednat. Sektory můžou být řazeny libovolně, jsou určeny pouze hodnotou data bytů. Poté se provede *Erase Sector* a pokud se operace provede správně, bootloader odešle hostovi 2*ACK. Tato operace a zejména *Mass Erase* může trvat několik sekund. [10]

Struktura příkazu Mass Erase:

1. Odeslání datového rámce s identifikátorem 0x43 a bez data bytů.
2. Čekání na ACK nebo NACK.
3. Odeslání datového rámce s jedním data bytem hodnoty 0xFF.
4. Čekání na 2*ACK nebo NACK.

Struktura příkazu Erase Sector:

1. Odeslání datového rámce s identifikátorem 0x43 a bez data bytů.
2. Čekání na ACK nebo NACK.
3. Odeslání jednoho datového rámce s jedním data bytem určujícím počet sektorů *N - 1* k smazání.
4. Čekání na ACK nebo NACK.

5. Odeslání jednoho datového rámce s počtem data bytů odpovídajících požadovanému počtu sektorů *N* a s hodnotou určující o jaké sektory se jedná.
6. Čekání na 2*ACK nebo NACK.

5.4.2.9 Write Protect command

Příkaz *Write Protect command* slouží k nastavení ochrany sektorů před zápisem Flash paměti. Obdrží-li bootloader tento příkaz, odešle hostovi ACK. Poté bootloader čeká na datový rámec s jedním data bytem. Hodnota tohoto data bytu určuje počet sektorů k ochraně paměti. Po přijetí tohoto datového rámce bootloader odešle ACK. Poté bootloader čeká na datový rámec určující, o jaké sektory se jedná. Počet odeslaných bytů odpovídá počtu sektorů a data byty tohoto rámce určují, o jaké sektory se bude jednat podobně jako u *Erase Sector*. Poté se provede ochrana zvolených sektorů a proběhne-li vše správně bootloader odešle hostovi 2*ACK a provede systém reset. [10]

Struktura příkazu:

1. Odeslání datového rámce s identifikátorem 0x63 a bez data bytů.
2. Čekání na ACK nebo NACK.
3. Odeslání jednoho datového rámce s jedním data bytem určujícím počet sektorů *N - 1* k ochraně před zápisem.
4. Čekání na ACK nebo NACK.
5. Odeslání jednoho datového rámce s počtem data bytů odpovídajících požadovanému počtu sektorů *N* a s hodnotou určující, o jaké sektory se jedná.
6. Čekání na 2*ACK nebo NACK.

5.4.2.10 Write Unprotect command

Příkaz funguje principiálně podobně jako příkaz *Write Unprotect command* v kapitole 5.1.2.10.

Struktura příkazu:

5. Odeslání datového rámce s identifikátorem 0x73 a bez data bytů.
6. Čekání na 2 * ACK nebo NACK.

5.4.2.11 Readout Protect command

Příkaz funguje principiálně podobně jako příkaz *Readout Protect command* v kapitole 5.1.2.11.

Struktura příkazu:

3. Odeslání datového rámce s identifikátorem 0x82 a bez data bytů.
4. Čekání na 2 * ACK nebo NACK.

5.4.2.12 Readout Unprotect command

Příkaz funguje principiálně podobně jako příkaz *Readout Unprotect command* v kapitole 5.1.2.12.

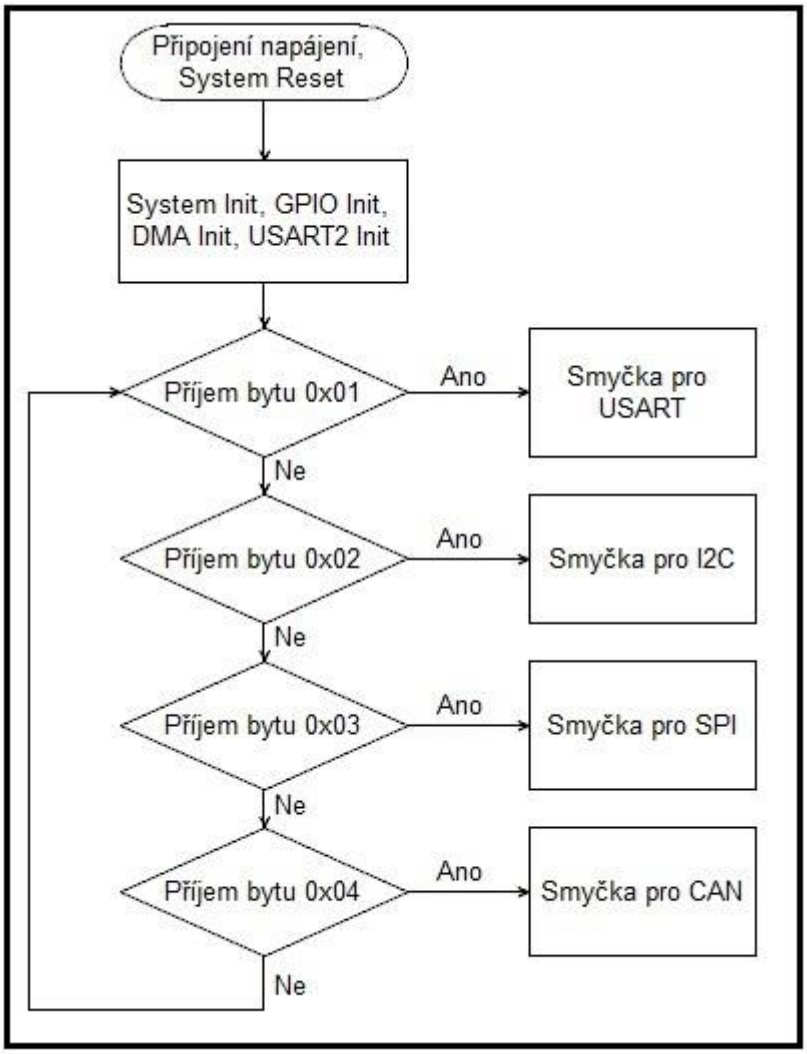
Struktura příkazu:

7. Odeslání datového rámce s identifikátorem 0x92 a bez data bytů.
8. Čekání na 2 * ACK nebo NACK.

5.5 Komunikace s PC

V této kapitole je popsán firmware Master Nuclea s komunikací tohoto Nuclea s PC. Pro přenos informací mezi aplikací na počítači a Master Nucleem je použita sériová komunikace USART. Přijatá data po komunikaci USART jsou do paměti uložena s využitím DMA, které nezatěžuje procesor. Proto je možné současně s přijímáním dat zadávat příkazy pro bootloader. Data přijatá z počítače slouží k rozpoznání příkazu, který má být vykonán, a

mezi nimi jsou i data, která tento příkaz potřebuje, aby byl vykonán podle požadovaného zadání. Pro tento účel byla speciálně vytvořena komunikace.



obr. 10: Výběr komunikace Master Nucleo kitu

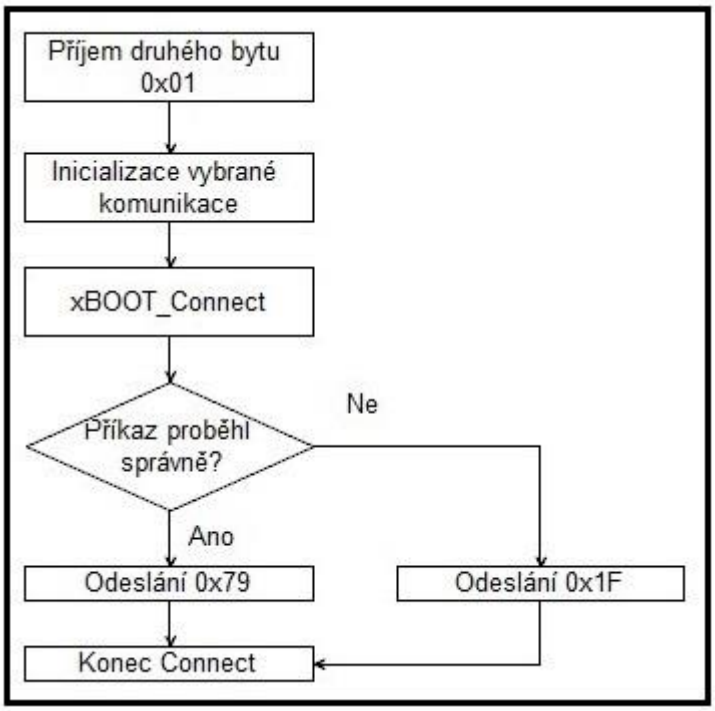
Po připojení napájení nebo System Resetu se inicializují periferie jako například DMA, USART2 (určený pro komunikaci s PC), Clock, GPIOS a další. Poté se nastaví příjem třech bytů. První detekuje požadavek na příkaz (0x5A), druhý volí typ příkazu (0x01 až 0x09, 0x10) a třetí typ komunikace s bootloaderem (0x01 až 0x04). Poté ve smyčce Master čeká, až přijme z PC tři byty. První byt musí být vždy 0x5A, druhý musí být 0x01 (příkaz *Connect*) a třetí libovolná zvolená komunikace (0x01 až 0x04). Po příjmu těchto bytů Master spadne do

smyčky vybrané komunikace. Druh komunikace již za běhu programu nelze měnit, pro změnu komunikace je třeba System Reset. Ve smyčce dané komunikace se čeká na příkazy.

5.5.1 Příkazy komunikace s PC

Příkazy pro komunikaci s PC zpracovávají data přijatá z počítače a podle nich volí a nastavují příkazy pro bootloader. Příkazy pro bootloader vrací do proměnné *retval* hodnotu 0x79, pokud proběhnou správně, a hodnotu 0x1F, pokud nastane chyba. Pro výběr jednoho z příkazů je zapotřebí obdržet z počítače tři byty. Po každém vykonaném příkazu je nutné vymazat pole dat, do kterého se tyto tři byty ukládají a nastavit opětovný příjem těchto třech bytů. Zdrojový kód příkazů komunikace s PC je dostupný v příloze: *DVD/firmware/BOOT_C/BOOT_C/Inc/main*. [10]

5.5.1.1 Connect

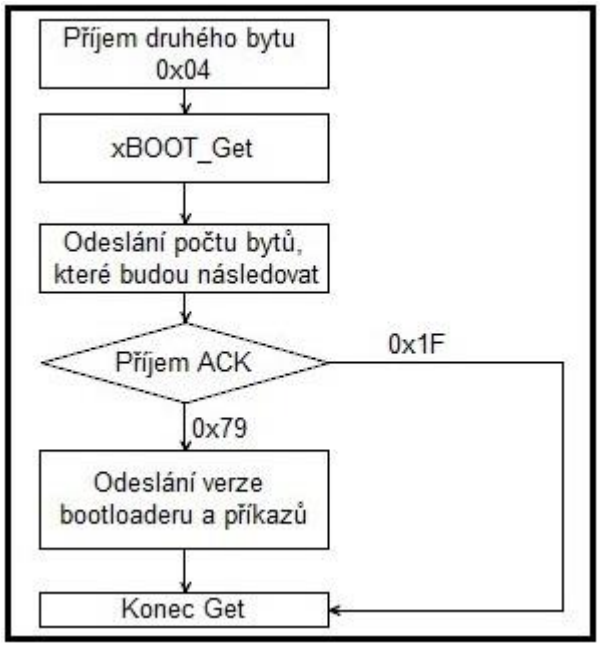


obr. 11: Příkaz Connect

Tento příkaz slouží pro zpracování požadavku na připojení jedné z vybraných komunikací. Podle přijatého druhého bytu se rozpozná, že se jedná o příkaz *Connect*.
Dle třetího bytu se vybere a inicializuje komunikace (USART1, I2C1, SPI2, CAN1). Provede

se příkaz *BOOT_Connect* náležící zvolené komunikaci a proběhne-li správně, odešle se potvrzení 0x79. Při chybě se odešle 0x1F.

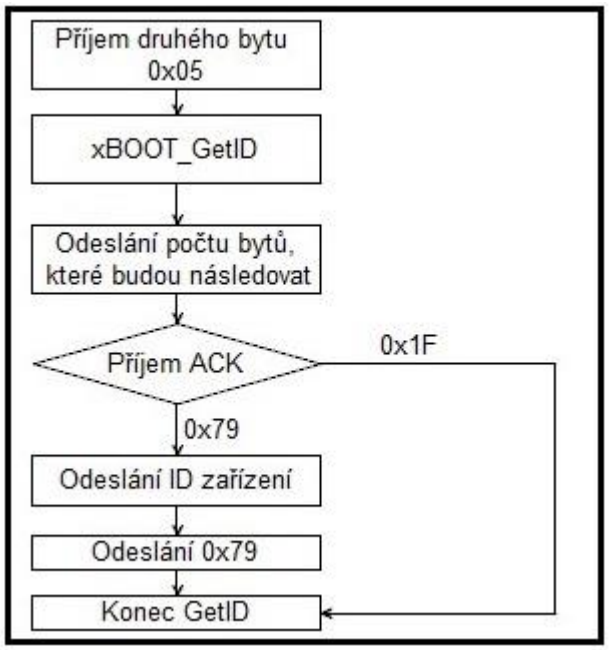
5.5.1.2 Get



obr. 12: Příkaz Get

Tento příkaz slouží ke zpracování požadavku na získání verze a podporovaných příkazů bootloaderu a odeslání těchto dat do počítače. Podle přijatého druhého bytu se rozpozná, že se jedná o příkaz *Get*. Provede se příkaz *BOOT_Get* náležící zvolené komunikaci, tím se získá verze a podporované příkazy bootloaderu. Odešle se byt s hodnotou *N*, který udává, v kolika bytech bude verze a podporované příkazy bootloaderu. Dále je zde čekání na potvrzení převzetí. Poté dojde k odeslání *N+1* bytů, ve kterých je verze bootloaderu, podporované příkazy a potvrzení 0x79.

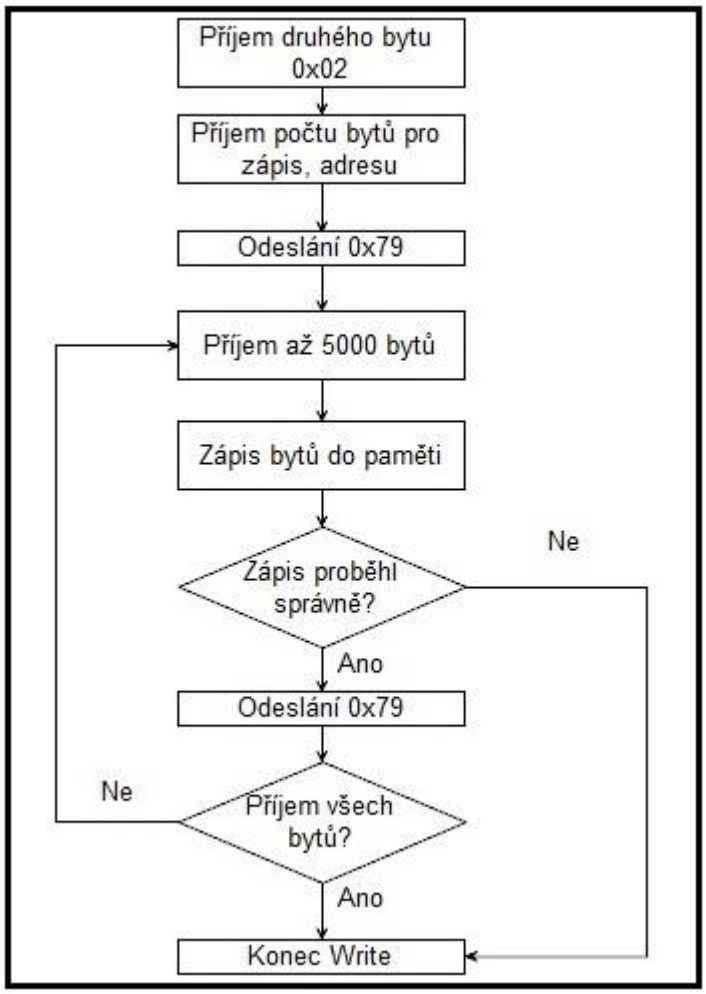
5.5.1.3 GetID



obr. 13: Příkaz GetID

Tento příkaz slouží k získání ID mikrokontroléru. Podle přijatého druhého bytu se rozpozná, že se jedná o příkaz *GetID*. Provedením příkazu *BOOT_GetID* jednou z komunikací se získá ID mikrokontroléru. Odešle se byt s hodnotou *N*, který udává počet bytů příslušících ID mikrokontroléru. Dále je zde čekání na potvrzení převzetí. Poté se odešle *N+1* bytů s ID a potvrzení 0x79.

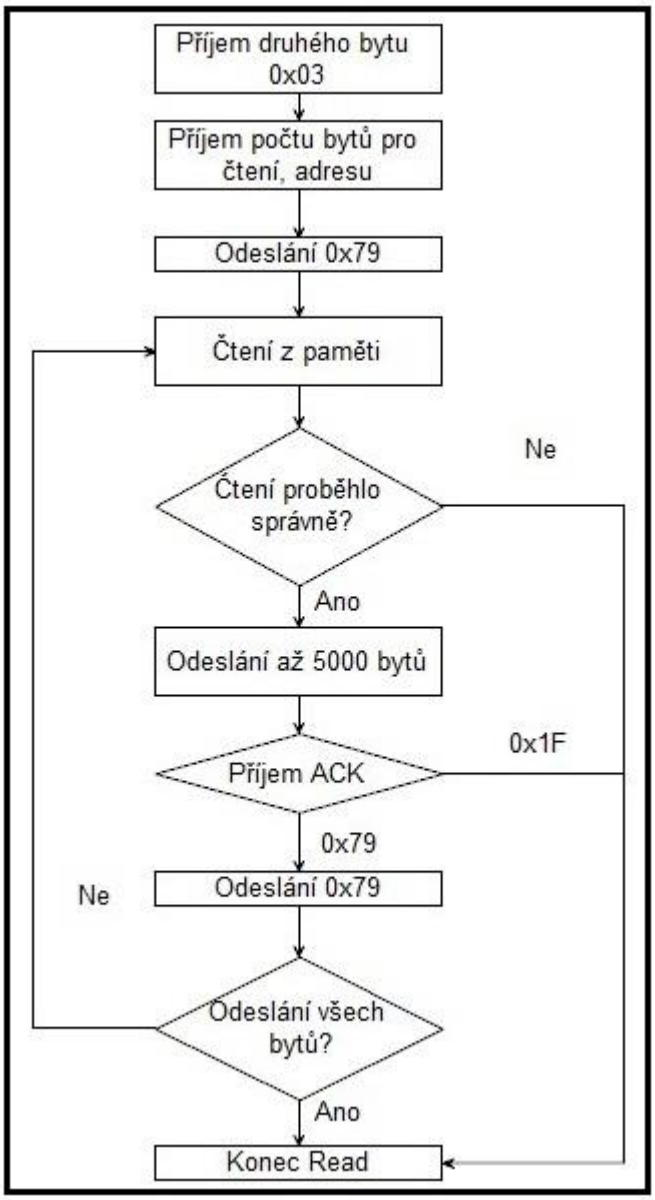
5.5.1.4 Write



obr. 14: Příkaz Write

Tímto příkazem se zapisují data přijatá z počítače do paměti. Podle přijatého druhého bytu se rozpozná, že se jedná o příkaz *Write*. Poté se přijme sedm bytů, v prvních třech bytech je počet bytů pro zápis do paměti a v dalších čtyřech je adresa, od které se zapisuje. Po přijetí těchto bytů se odešle 0x79, tak počítač pozná, že může posílat data. Data jsou přijímána z počítače až po 5 kB a do bootloaderu odesílána po 250 B. Po zápisu všech přijatých bytů je odeslán do počítače byt 0x79. Nejsou-li přijata z počítače všechna data, příjem se opakuje, dokud do paměti nejsou zapsána všechna data.

5.5.1.5 Read

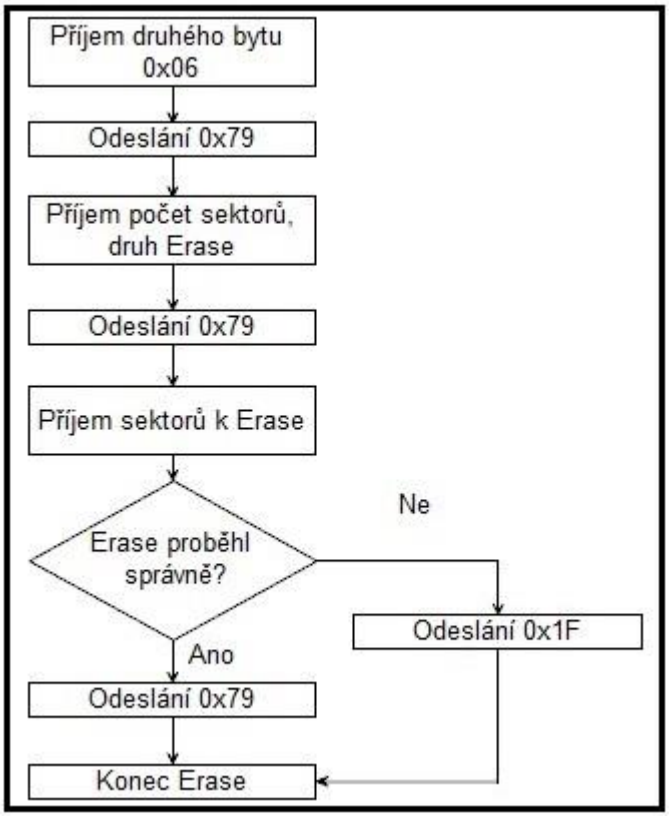


obr. 15: Příkaz Read

Tímto příkazem se čtou data z paměti a odesílají do počítače. Podle přijatého druhého bytu se rozpozná, že se jedná o příkaz *Read*. Poté se přijme sedm bytů, v prvních třech bytech je počet bytů pro čtení z paměti a v dalších čtyřech je adresa, od které se čte. Po přijetí těchto bytů se odešle 0x79 a začnou se číst data z paměti. Najednou se přečte až 5 kB po 250 B. Přečtené byty se poté odesílají do počítače. Po odeslání se čeká na příjem bytu 0x79, po přijetí

tohoto bytu se odešle byt 0x79 zpět. Nejsou-li do počítače odeslána všechna požadovaná data, čtení se opakuje, dokud do počítače nejsou odeslána všechna požadovaná data.

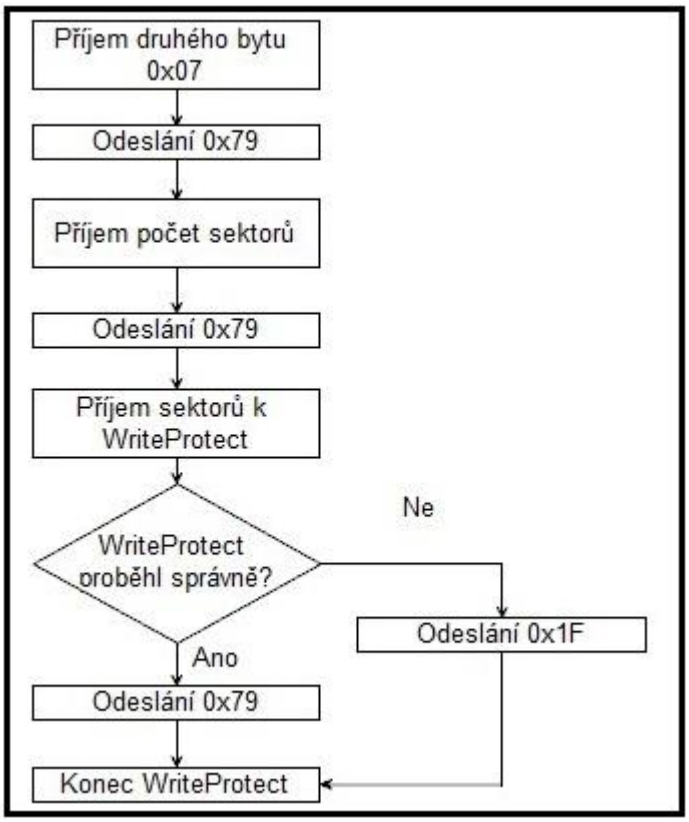
5.5.1.6 Erase



obr. 16: Příkaz Erase

Tímto příkazem se vyvolá *Erase sector* nebo *Mass Erase*. Podle přijatého druhého bytu se rozpozná, že se jedná o příkaz *Erase*. Poté se odešle byt 0x79. Po odeslání se přijímají dva byty, v prvním se odesílá počet sektorů k erase a ve druhém informace, zda se jedná o *Mass Erase* nebo *Sector Erase*. Po přijetí těchto bytů se odešle byt 0x79. Poté se přijímají sektory k mazání. Po přijetí sektorů se vykoná příkaz *BOOT_Erase* zvolenou komunikací s bootloaderem. Proběhne-li příkaz správně, odešle se byt 0x79, dojde-li k chybě, odešle se byt 0x1F.

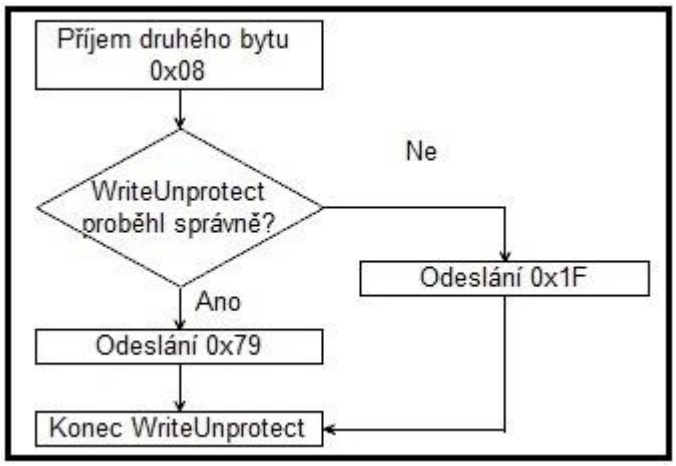
5.5.1.7 WriteProtect



obr. 17: Příkaz WriteProtect

Tímto příkazem se aktivuje ochrana paměti před zápisem. Podle přijatého druhého bytu se rozpozná, že se jedná o příkaz *WriteProtect.* Poté se odešle byt 0x79. Po odeslání se přijímá byt s počtem sektorů k ochraně před zápisem. Po přijetí tohoto bytu se odešle byt 0x79. Poté se přijímají sektory k ochraně před zápisem. Po přijetí těchto sektorů se vykoná příkaz *BOOT_WtiteProtect.* Proběhne-li příkaz správně, odešle se byt 0x79, pokud ne, odešle se byt 0x1F.

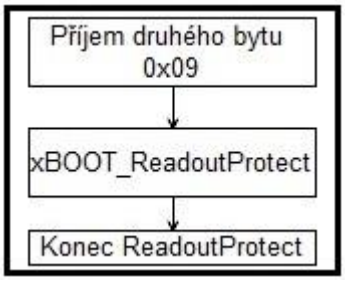
5.5.1.8 WriteUnprotect



obr. 18: Příkaz WriteUnprotect

Tento příkaz vyvolá zrušení ochrany před zápisem. Podle přijatého druhého bytu se rozpozná, že se jedná o příkaz *WriteUnprotect.* Provede se příkaz *BOOT_WriteUnprotect*. Vykoná-li se tento příkaz správně, odešle se byt 0x79, dojde-li k chybě, odešle se byt 0x1F.

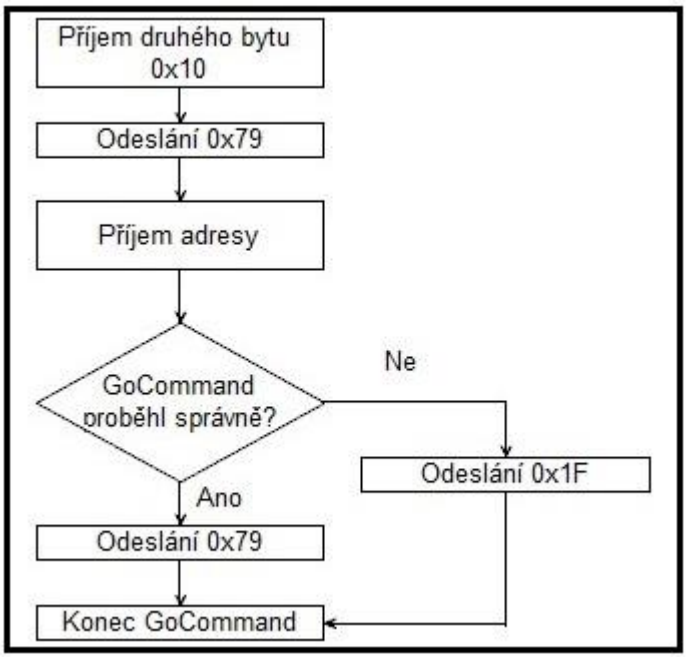
5.5.1.9 ReadoutProtect



obr. 19: Příkaz ReadoutUnprotect

Tímto příkazem se aktivuje ochrana před čtením. Podle přijatého druhého bytu se rozpozná, že se jedná o příkaz *ReadoutProtect.* Provede se příkaz *BOOT_ReadoutProtect*. Po vykonání tohoto příkazu není možné se zařízením pomocí bootloaderu komunikovat. Odstranění této ochrany je možné například přes J-TAG pomocí ST-LINK Utility.

5.5.1.10 GoCommand



obr. 20: Příkaz GoCommand

Tímto příkazem se přejde na adresu v paměti zaslanou počítačem. Podle přijatého druhého bytu se rozpozná, že se jedná o příkaz *GoCommand*. Poté se odešle byt 0x79. Po odeslání tohoto bytu se přijmou čtyři byty s adresou a vykoná se příkaz *BOOT_GoCommand*. Vykoná-li se tento příkaz správně, odešle se byt 0x79, pokud ne odešle se byt 0x1F.

6 Software pro PC

Software je vytvořený pro operační systém Windows 10 ve Visual Studiu 2015. Je napsán v programovacím jazyce C# a pro jeho vytvoření byl použit .NET Framework. Software slouží pro koncového uživatele. Je vytvořen pro jednoduchou práci s bootloaderem a vizualizaci výstupů bootloaderu.

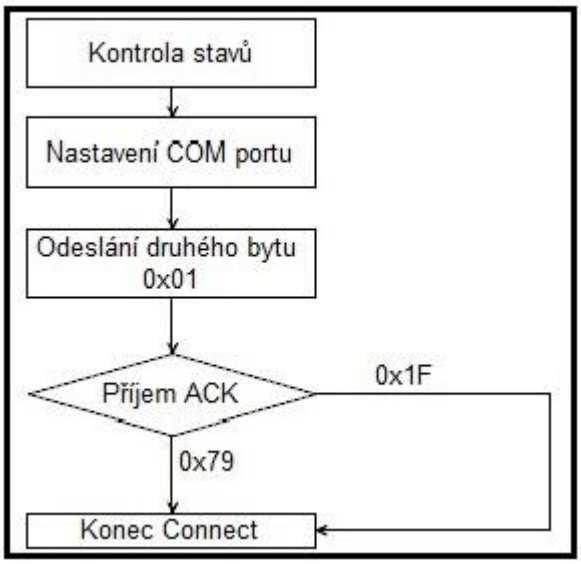
6.1 Komunikace s Master Nucleo kitem

Ke komunikaci s Master Nucleo kitem byl použit sériový COM port. Pro práci se sériovým portem byla využita třída *SerialPort* nacházející se v knihovně *Systém.IO.Ports*. Pro navázání komunikace je nutné zadat název daného COM portu, přenosovou rychlost, paritu a počet přenášených bytů. Název COM portu se zařízení od zařízení a počítač od počítače může měnit, proto je vyhledáván automaticky pomocí specifického názvu. Zbytek je zadán fixně. Přenosová rychlost je nastavena na 115200 bps, bez parity, s jedním stop bytem a osmi data byty.

6.1.1 Příkazy komunikace s Master Nucleo kitem

Příkazy komunikace s Nucleo kitem přejímají data zadaná uživatelem, tato data zpracovávají a odesílají Nucleo kitu v přesně nadefinované sekvenci. Tyto příkazy jsou opatřeny ochranami proti špatnému pořadí příkazů nebo nevhodným zadaným datům. Zda se v příkazu detekuje chyba, příkaz končí a vrací hodnotu 1, pokud proběhne správně, vrací hodnotu 0. Každý příkaz nejprve odesílá Nucleo kitu tři byty, první je vždy 0x5A, druhý specifikuje daný příkaz a třetí byt druh komunikace. Dále u jednotlivých příkazů bude vždy uveden pouze druhý byt specifikující daný příkaz, ale ve skutečnosti budou odesílány vždy na začátku tři byty. Příkazy se vybírají tlačítky nebo v menu. Zdrojový kód příkazů komunikace s PC je dostupný v příloze: *DVD/software/DP_C#/BOOT/BOOT/ComandBOOT*.

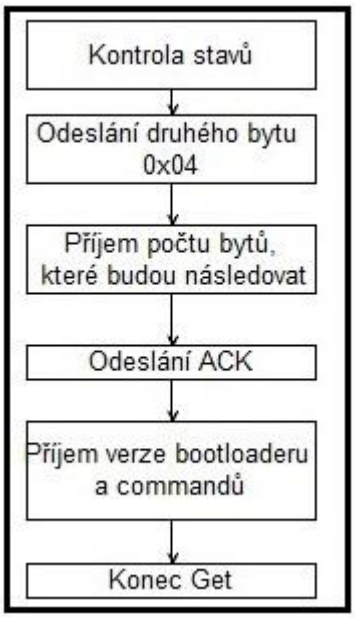
6.1.1.1 Connect



obr. 21: Příkaz Connect

Tímto příkazem se uživatel připojí vybranou komunikací k bootloaderu. Po spuštění tohoto příkazu se zkontrolují nejprve stavy, zda neproběhl před tímto příkazem příkaz *GoCommand* nebo příkaz *Connect*. Jestli ano, příkaz skončí s hlášením. Dále se zkontroluje, zda není COM port již nastaven, jestli ano, nenastavuje se znovu, pokud ne, nastaví se. Dále se odešlou tří byty, druhý 0x01 určuje, zda se jedná o *Connect*. Poté se čeká na příjem ACK. Přijatá hodnota 0x1F detekuje chybu.

6.1.1.2 Get



obr. 22: Příkaz Get

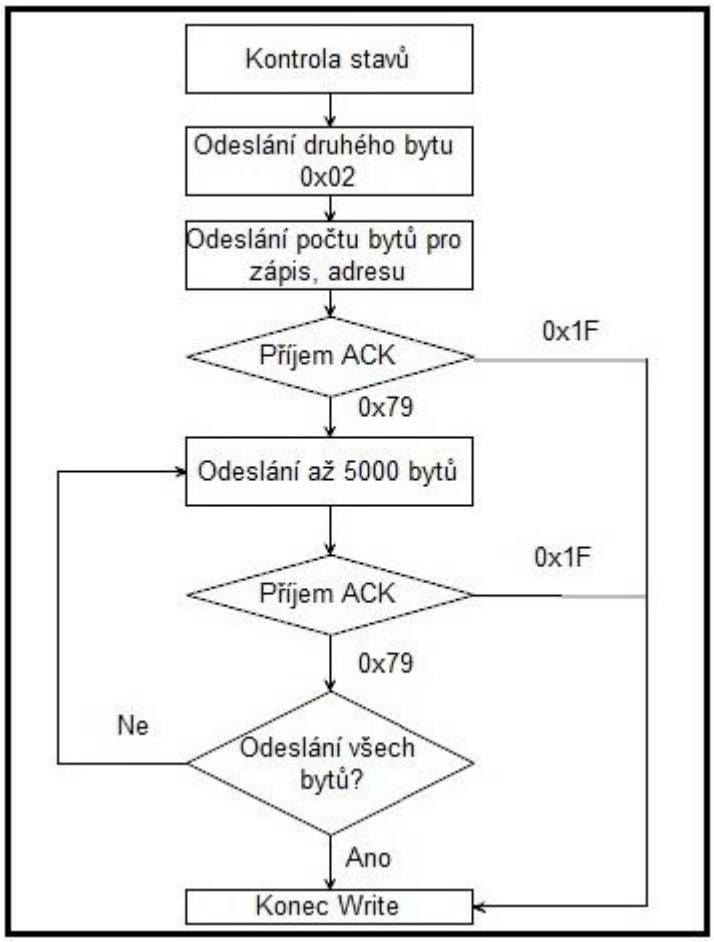
Tímto příkazem se získá verze bootloaderu připojeného zařízení a příkazy, které podporuje. Po spuštění tohoto příkazu se zkontroluje stav, zda již proběhl příkaz *Connect*. Jestli ano, odešlou se tři byty, druhým 0x04 je určen tento příkaz. Poté se přijme byt, v kterém je udáno, v kolika bytech *N* bude přijata verze bootloaderu a podporované příkazy. Poté se čeká na příjem ACK. Po přijetí ACK se začnou přijímat byty *N+1* s verzí bootloaderu a podporovanými příkazy.

6.1.1.3 GetID

obr. 23: Příkaz GetID

Tento příkaz získá ID připojeného zařízení. Po spuštění tohoto příkazu se zkontroluje stav, zda již proběhl příkaz *Connect*. Jestli ano odešlou se tři byty, druhý 0x05 určuje příkaz *GetID*. Poté se přijme byt, v kterém je udáno, v kolika bytech *N* bude přijato ID zařízení. Poté se čeká na příjem ACK. Po přijetí ACK se začnou přijímat byty *N+1* s ID zařízení.

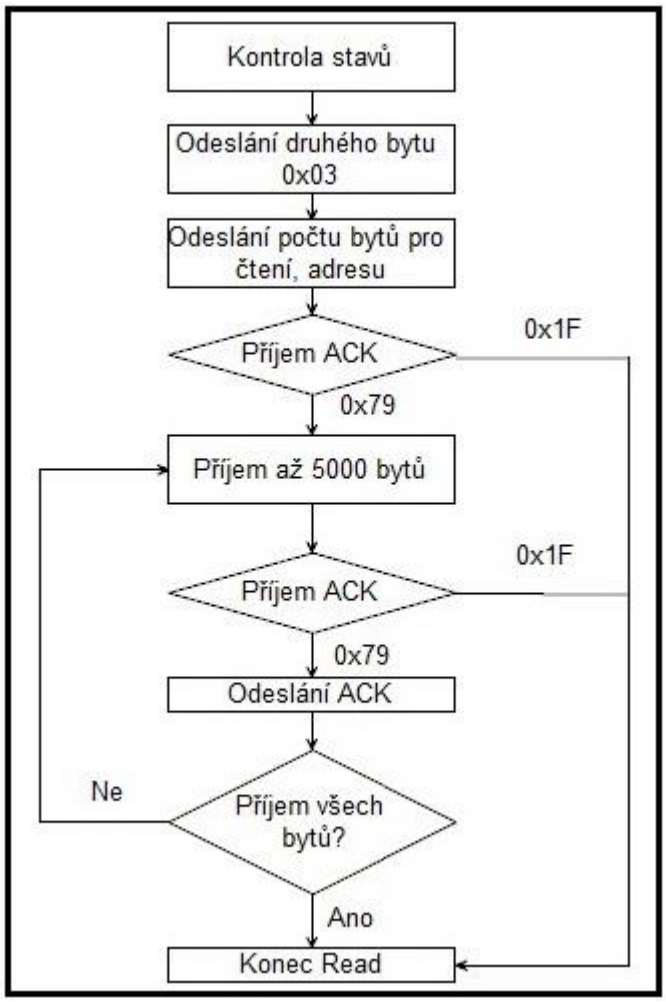
6.1.1.4 Write



obr. 24: Příkaz Write

Tento příkaz do cílového zařízení zapíše na uživatelem stanovenou adresu v paměti zvolený binární kód. Po aktivování tohoto příkazu se zkontroluje stav, jestli proběhl již příkaz *Connect,* zda proběhla ochrana paměti či jestli není počet bytů pro zápis rovný nule. Pokud neproběhl příkaz *Connect* nebo pokud je paměť chráněna nebo když je počet bytů pro zápis rovný nule, příkaz skončí. Poté jsou odeslány tři byty, druhý byt 0x02. Po odeslání těchto bytů je počet bytů pro zápis rozložen do třech bytů a adresa do čtyř bytů a tyto byty odeslány, celkem tedy sedm bytů. Poté se čeká na příjem ACK. Po příjmu ACK se odešle až 5000 bytů. Po odeslání těchto bytů se čeká na příjem ACK. Jsou-li odeslány všechny byty, příkaz končí, pokud ne, odesílání bytů a příjem ACK se opakuje, dokud nejsou odeslány všechny požadované byty.

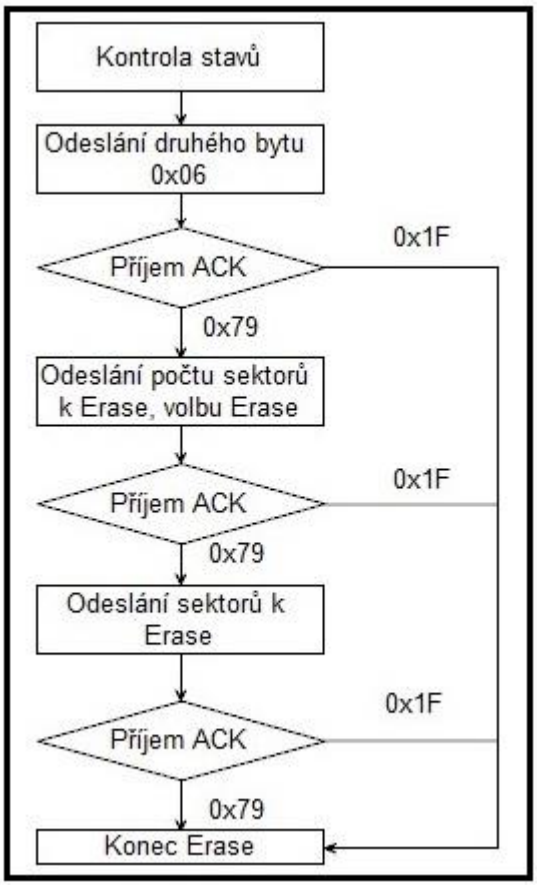
6.1.1.5 Read



obr. 25: Příkaz Read

Tento příkaz z paměti cílového zařízení přečte z uživatelem stanovené adresy zvolený počet bytů. Po aktivování tohoto příkazu se zkontroluje stav, jestli proběhl již příkaz *Connect,* zda proběhla ochrana paměti či jestli není počet bytů pro čtení rovný nule. Pokud neproběhl příkaz *Connect* nebo pokud je paměť chráněna nebo když je počet bytů pro čtení rovný nule, příkaz skončí. Poté jsou odeslány tři byty, druhý byt 0x03. Po odeslání těchto bytů je počet bytů pro čtení a adresa rozložena podobně jako v příkazu *Write* do sedmi bytů a odeslána. Po odeslání se čeká na ACK. Po přijetí se přijímají přečtené data byty a ACK. Po přijetí ACK se odešle ACK. Jsou-li přijaty všechny byty, příkaz končí, pokud ne, příjem bytů a ACK se opakuje, dokud nejsou přijaty všechny požadované byty.

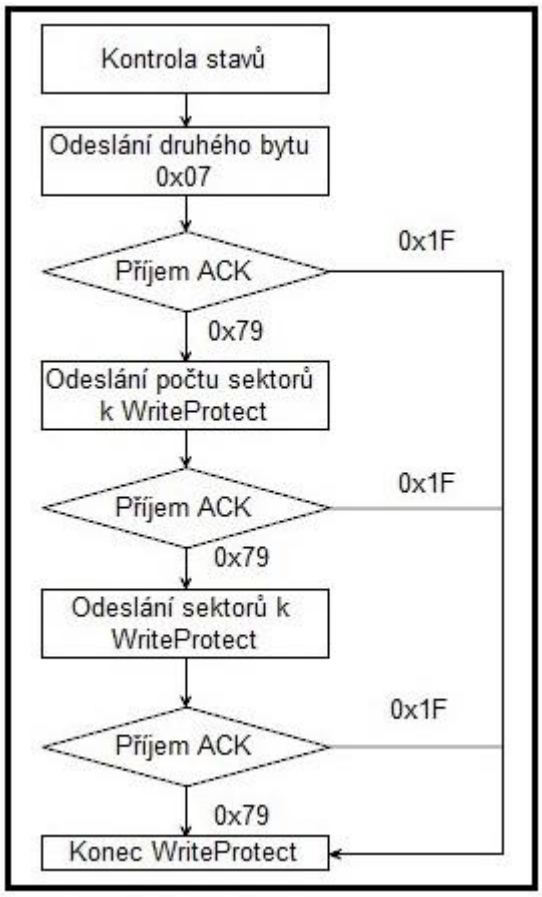
6.1.1.6 Erase



obr. 26: Příkaz Erase

Tento příkaz maže paměť cílového mikrokontroléru, maže buď celou paměť, nebo vybrané sektory. Po aktivování tohoto příkazu se zkontroluje stav, jestli proběhl již příkaz *Connect,* zda proběhla ochrana paměti či jestli není počet sektorů k mazání rovný nule. Pokud neproběhl příkaz *Connect* nebo pokud je paměť chráněna nebo když je počet sektorů k mazání rovný nule, příkaz skončí. Poté jsou odeslány tři byty, druhý byt je 0x06. Poté se čeká na příjem ACK. Po přijetí ACK jsou odeslány dva byty, první byt určuje počet sektorů k mazání a druhý, zda se jedná o *Erase Sector* nebo *Mass Erase*. Poté se čeká na přijetá ACK. Po přijetí ACK jsou odeslány sektory, kterých se mazání týká. Poté je přijat ACK a příkaz končí.

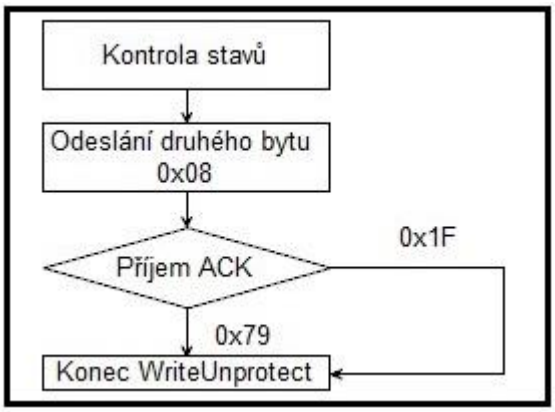
6.1.1.7 WriteProtect



obr. 27: Příkaz WriteProtect

Příkaz *WriteProtect* nastaví u cílového mikrokontroléru ochranu paměti před zápisem vybraným sektorům. Po aktivování tohoto příkazu se zkontroluje stav, jestli proběhl již příkaz *Connect,* zda proběhla ochrana paměti či jestli není počet sektorů k ochraně paměti před zápisem rovný nule. Pokud neproběhl příkaz *Connect* nebo pokud je paměť chráněna nebo když je počet sektorů k ochraně paměti před zápisem rovný nule, příkaz skončí. Poté jsou odeslány tři byty, druhý byt je 0x07. Poté se čeká na příjem ACK. Po přijetí ACK se odešle počet sektorů k ochraně paměti před zápisem. Poté se čeká na ACK. Po přijetí ACK se odesílají sektory k ochraně paměti před zápisem. Poté je přijat ACK a příkaz končí.

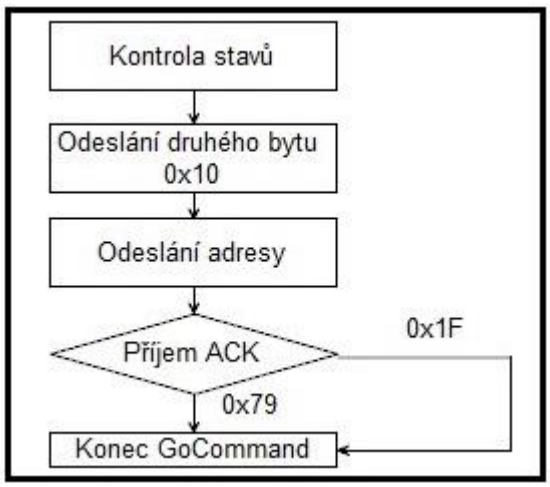
6.1.1.8 WriteUnprotect



obr. 28: Příkaz WriteUnprotect

Tento příkaz zruší ochranu paměti před zápisem všech sektorů cílového mikrokontroléru. Po aktivování tohoto příkazu se kontroluje stav, zda proběhl již příkaz *Connect*, pokud ne, příkaz končí. Poté jsou přijaty tři byty, druhý 0x08 určuje, že se jedná o tento příkaz. Poté je přijat ACK a příkaz končí.

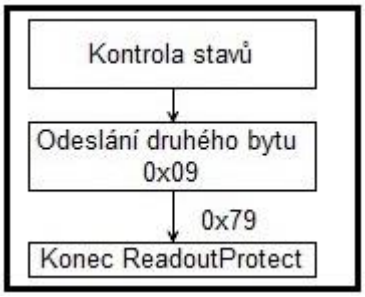
6.1.1.9 GoCommand



obr. 29: Příkaz GoCommand

Tímto příkazem se skočí na zvolenou adresu paměti cílového mikrokontroléru. Po aktivování tohoto příkazu se kontroluje stav, zda proběhl již příkaz *Connect*, pokud ne, příkaz končí. Poté jsou odeslány tři byty, druhý byt je 0x10. Po odeslání těchto bytů se odešle adresa, na kterou se má skočit. Adresa je odeslána ve čtyřech bytech. Poté se čeká na příjem ACK. Po přijetí ACK příkaz končí.

6.1.1.10 ReadoutProtect

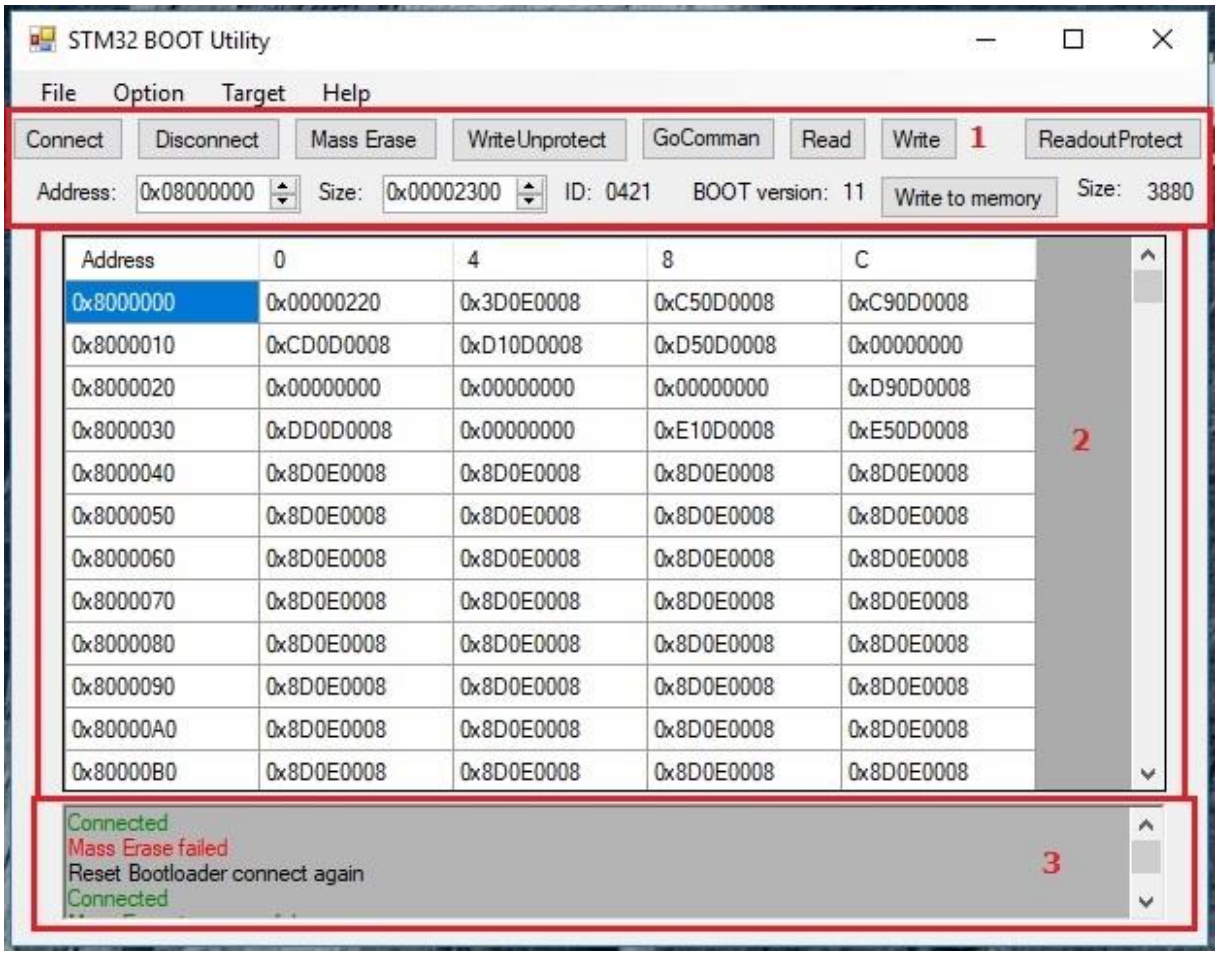


obr. 30: Příkaz ReadoutProtect

Tento příkaz nastaví ochranu celé paměti před čtením. Po aktivování tohoto příkazu se kontroluje stav, zda proběhl již příkaz *Connect*, pokud ne, příkaz končí. Poté jsou odeslány tři byty, druhý byt 0x09. Po odeslání těchto bytů příkaz končí.

6.2 Uživatelské prostředí

Uživatelská aplikace je klasickou okenní aplikací. Snahou bylo, aby byl graficky a uživatelsky podobný ST-LINK Utilitě. Uživatel vybírá jednotlivé příkazy v menu nebo pomocí tlačítek. Data, jako je například adresa, se zadávají do číselného pole. V této kapitole budou popsány a vysvětleny možnosti uživatelského prostředí. Nebude chybět popis tlačítek, číselných polích, zaškrtávacích polí, atd. Zdrojový kód aplikace je dostupný v příloze: *DVD/software/DP_C#/BOOT/BOOT*.



obr. 31: Hlavní okno PC aplikace

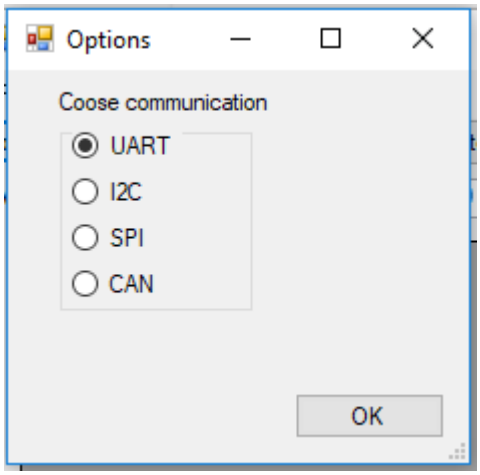
6.2.1 Tlačítka hlavního okna

Tlačítka hlavního okna, číselná pole a textová pole vizualizující ID a verzi bootloaderu cílového zařízení jsou umístěna na obrázku 31 v červeném obdélníku 1. V hlavním okně jsou tato tlačítka:

Connect

Po aktivování tohoto tlačítka se zobrazí okno (na obrázku 32) s výběrem jedné ze čtyř komunikací s bootloaderem cílového zařízení a Nucleem. Křížkem se dá výběr zrušit, neproběhne žádná akce. Tlačítkem OK (na obrázku 32) se provede připojení k bootloaderu zvolenou komunikací příkazem *Connect,* zruší se ochrana před zápisem celé paměti příkazem

WriteUnprotect a získá se ID zařízení a verze bootloaderu příkazem *GetID* a *Get*. Poté se přečte pole bytů z paměti začínajícího na adrese zadané číselným polem uvozeným textem *„Address:“* s velikostí zadanou číselným polem uvozeným textem *„Size:“*. Poté se aktualizuje tabulka (kapitola 6.2.2) přečtenými daty z paměti a aktualizují se textová pole s ID a verzí bootloaderu. Nakonec se nastaví stav informující o připojení k bootloaderu.



obr. 32: Okno výběru komunikace

Disconnect

Toto tlačítko nastaví veškeré stavy do počátečního nastavení, zrušení připojení nuclea s bootloaderem cílového zařízení je možné jednoduše odpojením a opětovným zapojením nuclea a PC.

Mass Erase

Aktivací tlačítka Mass Erase aktivujeme celkové vymazání flash paměti cílového zařízení příkazem *Erase*.

WriteUnprotect

Tlačítko WriteUnprotect zruší ochranu flash paměti před zápisem příkazem *WriteUnprotect*. Dále se zruší stav indikující ochranu paměti před zápisem.

GoCommand

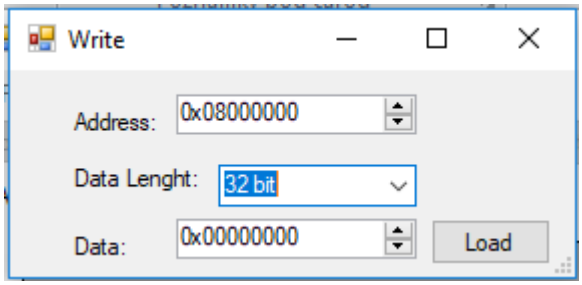
Po aktivaci tohoto tlačítka se začne provádět kód od předem zadané adresy, tato adresa je zadaná číselným polem uvozeným textem *„Address:“* podobně jako pro čtení paměti u tlačítka Connect. Pro skok na danou adresu slouží příkaz *GoCommand*. Po úspěšném příkazu *GoCommand* je bootloader cílového zařízení odpojen a pro opětovné připojení je nutné odpojit a znovu připojit Nucleo a znovu navázat komunikaci tlačítkem Connect.

Read

Aktivací tlačítka Read se aktivuje příkaz *Read*. Tím se přečte pole bytů z paměti začínajícího na adrese zadané číselným polem uvozeným textem *„Address:“* s velikostí zadanou číselným polem uvozeným textem *„Size:“* stejně jako u tlačítka Connect. Nakonec se aktualizuje tabulka (kapitola 6.2.2) nově získanými daty z paměti.

Write

Tímto tlačítkem se se zapisuje do paměti cílového zařízení 8, 16, 24 nebo 32 bitů dle volby. Po aktivaci tohoto tlačítka se zobrazí okno (obrázek 33) s výběrem adresy, na kterou chceme zapisovat, výběrem počtu bitů které chceme zapsat a zadáním těchto bitů do číselného pole. Křížkem můžeme okno zavřít. Tlačítkem Load aktivujeme příkaz *Write* a zapíše se na vybranou adresu zadaný počet bitů. Pokud je nastavená ochrana paměti, příkaz *Write* neproběhne. Nakonec se vymaže tabulka (kapitola 6.2.2) s přečtenou pamětí, jelikož data v ní by již nebyla aktuální.



obr. 33: Okno pro zadávání dat pro zápis do paměti

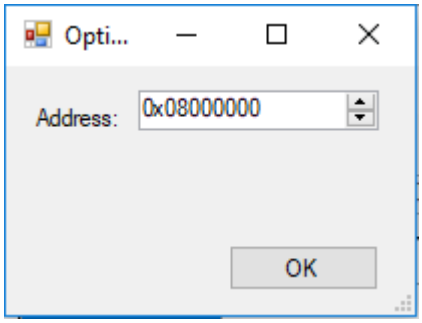
ReadoutProtect

Toto tlačítko nastavuje ochranu paměti úrovně 1 příkazem *ReadoutProtect*. Ochrana úrovně 1 znamená, že z paměti se již nedá číst a s bootloaderem cílového zařízení již není

možné komunikovat. Tato ochrana nejde pomocí bootloaderu odstranit a pro odstranění je nutné použít například nástroj ST-LINK Utility.

Write to memory

Toto tlačítko slouží pro zápis binárního souboru do paměti cílového zařízení. Po aktivování tohoto tlačítka se zobrazí okno, ve kterém se zadá cesta k binárnímu souboru. Po zadání cesty k binárnímu souboru se zobrazí okno (obrázek 34) pro zadání adresy, od které má být proveden zápis. Stiskem tlačítka OK se aktivuje příkaz *Write*, tím se provede zápis binárního souboru do paměti Flash od zadané adresy. Pokud neproběhlo připojení bootloaderu příkazem *Connect* nebo pokud je počet bytů pro zápis rovný nule nebo pokud je paměť chráněna před zápisem, zápis do paměti se neuskuteční. Pokud je nastavená ochrana paměti příkaz *Write* neproběhne. Nakonec se vymaže tabulka (kapitola 6.2.2) s přečtenou pamětí, jelikož data v ní by již nebyla aktuální.



obr. 34: Okno pro zadávání adresy pro zápis do paměti

6.2.2 Tabulka

Tabulka je umístěna v hlavním okně na obrázku 31 v červeném obdélníku 2. Tabulka začíná na adrese, od které proběhl příkaz *Read*, a její velikost odpovídá počtu přečtených bytů. Tabulka obsahuje pět sloupců. V prvním sloupci je adresa uvozující začátek dat dané řádky. V druhém až třetím sloupci jsou 32 bitová data v hexadecimální soustavě. Data začínají na adrese odpovídající adrese v prvním sloupci daného řádku. U každého sloupce si je nutné k této adrese přičíst hexadecimální hodnotu, tato hexadecimální hodnota je pro každý sloupec jiná a odpovídá názvu sloupců.

6.2.3 Status okno

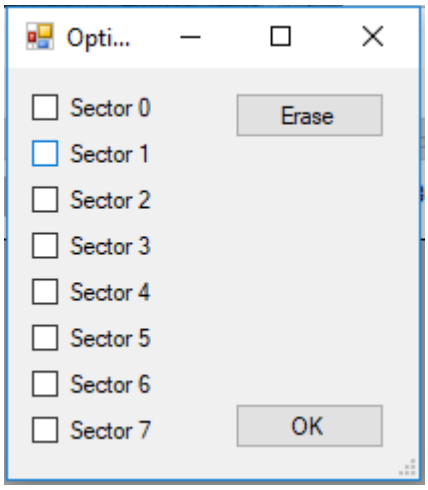
V Status okně jsou hlášky spojené s průběhem příkazů. Proběhne-li některá akce správně, v Status okně se zobrazí hláška napsaná zeleným písmem s informací, co proběhlo úspěšně. Poté jsou zde hlášky černým písmem, které uživateli poskytují doplňující informace. Posledním typem hlášek jsou červené, ty informují uživatele o chybě.

6.2.4 Menu

Menu je umístěno v hlavním okně (obrázek 25) u horního okraje. První dvě položky z menu File a Option nemají prozatím žádné praktické využití a jsou zde pro případné další rozšíření funkcionality aplikace. Položka Help zobrazí nápovědu s piny Nuclea, které jsou důležité pro funkci bootloaderu. V položce Target jsou na výběr další čtyři možnosti pro práci s bootloaderem a to Connect, Disconnect, Erase Sectors a Write Protect memory. Connect a Disconnect fungují stejně jako stejnojmenná tlačítka uvedená v kapitole 6.2.1, zbylé dvě možnosti budou vysvětleny zvlášť.

Erase Sectros

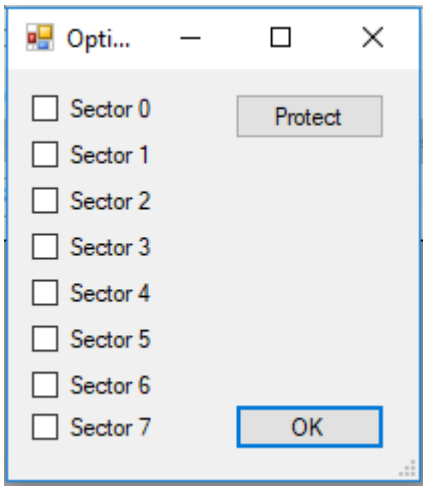
Tato položka slouží k výběru sektorů a následně k jejich smazání. Po zvolení Erase Sector vyskočí okno (obrázek 35) s výběrem sektorů k mazání a tlačítko Erase pro aktivaci *Erase*, tím se smažou vybrané sektory. Pokud neproběhlo připojení bootloaderu příkazem *Connect* nebo pokud je paměť chráněna před zápisem, mazání paměti se neuskuteční. Dále je zde tlačítko OK pro zrušení výběru a zavření okna.



obr. 35: Okno pro volbu sektorů k mazání

Write Protect memory

Položka slouží k výběru sektorů a následně k nastavení jejich ochrany před zápisem. Po zvolení vyskočí okno (obrázek 36) s výběrem sektorů k ochraně před zápisem a tlačítko Protect. Po aktivaci tlačítka Protect se nejprve provede příkaz *Unprotect*, který zruší případnou předešlou ochranu sektorů před zápisem, a teprve až poté se provede příkaz *Write Protect*, kterým se nastaví ochrana před zápisem u vybraných sektorů. Pokud neproběhlo připojení bootloaderu příkazem *Connect*, ochrana vybraných sektorů se neuskuteční. Dále je zde tlačítko OK pro zrušení výběru a zavření okna.



obr. 36: Okno pro volbu sektorů k ochraně před zápisem

7 Závěr

Vývojových prostředků pro mikrokontroléry STM32 byla uvedena celá řada. Blíže byly uvedeny ty, které pro vytvoření diplomové práce byly opravdu použity. Byl popsán mikrokontrolér, pro který byl psán firmware se zaměřením především na jeho Flash paměť. Dále byl obecně popsán integrovaný bootloader v mikrokontrolérech STM32.

Ve firmware se podařilo integrovat většinu příkazů pro integrovaný bootloader přes sériovou komunikaci USART, SPI, I2C a CAN. Byla vytvořena komunikace s PC po sériové komunikaci USART s použitím ST-LINK. Pro zpracování dat z PC a následnému využití těchto dat příkazy pro integrovaný bootloader slouží vytvořený algoritmus. Jediný příkaz, který se nepovedlo implementovat, byl *Readout Unprotected*, který slouží k odstranění ochrany úrovně 1. Tento příkaz nefunguje správně, proto nemohl být ani implementován.

Byl vytvořen software v podobě okenní aplikace. Aplikace je schopná komunikovat s Nucleo kitem přes COM port. Pro výběr jednotlivých příkazů pro bootloader slouží tlačítka a menu. Pro navázání spojení s Nucleo kitem není nutné zadávat COM port, příslušný COM port se najde automaticky. Data se zadávají do číselných polí a k vizualizaci dat z bootloaderu slouží textové pole a tabulka. Informace o stavu vykonávaných akcí je zobrazena ve status okně.

Hlavní cíl diplomové práce, vytvořit kompletní software pro práci s bootloaderem, se podařilo splnit. Do cílového mikrokontroléru STM32 pomocí integrovaného bootloaderu je možné pomocí jedné ze sériových komunikací USART, SPI, I2C nebo CAN nahrát binární kód do Flash paměti na libovolnou adresu přístupnou pro zápis. Je možné vyčíst data z paměti Flash z libovolné adresy přístupné pro čtení, je možné paměť mazat buď celou, nebo po sektorech. Celou paměť nebo po sektorech je možné chránit před zápisem, nebo nad celou pamětí ochranu před zápisem zrušit. Z cílového mikrokontroléru je možné vyčíst ID a verzi bootloaderu a je možné nastavit ochranu úrovně 1.

Současná práce skýtá možná budoucí vylepšení, a to pro software za použití stávajících příkazů. Jedním z nich je možnost přepisovat paměť přímo v tabulce jak je tomu například v ST-LINK Utilitě. Nebo přečtenou paměť cílového mikrokontroléru uložit do binárního souboru.

Seznam použité literatury

[1] OpenSTM32 Community. *About System Workbench for STM32* [online]. [cit. 2018-03-02]. Dostupné z:
http://www.openstm32.org/About%2BSystem%2BWorkbench%2Bfor%2BSTM32

[2] UM1718 User manual. *STM32CubeMX for STM32 configuration and initialization C code generation* [online]. c2018 [cit. 2018-03-02]. Dostupné z:
http://www.st.com/content/ccc/resource/technical/document/user_manual/10/c5/1a/43/3a/70/43/7d/DM00104712.pdf/files/DM00104712.pdf/jcr:content/translations/en.DM00104712.pdf

[3] RM0390 Reference manual. *STM32F446xx advanced Arm®-based 32-bit MCUs* [online]. c2018 [cit. 2018-03-05]. Dostupné z:
http://www.st.com/content/ccc/resource/technical/document/reference_manual/4d/ed/bc/89/b5/70/40/dc/DM00135183.pdf/files/DM00135183.pdf/jcr:content/translations/en.DM00135183.pdf

[4] AN2606 Application note. *STM32 microcontroller system memory boot mode* [online]. c2018 [cit. 2018-03-05]. Dostupné z:
http://www.st.com/content/ccc/resource/technical/document/reference_manual/4d/ed/bc/89/b5/70/40/dc/DM00135183.pdf/files/DM00135183.pdf/jcr:content/translations/en.DM00135183.pdf

[5] UM1724 User manual. *STM32 Nucleo-64 boards* [online]. c2015 [cit. 2018-03-07]. Dostupné z:

http://www.st.com/content/ccc/resource/technical/document/user_manual/98/2e/fa/4b/e0/82/43/b7/DM00105823.pdf/files/DM00105823.pdf/jcr:content/translations/en.DM00105823.pdf

[6] AN3155 Application note. *USART protocol used in the STM32 bootloader* [online]. c2016 [cit. 2018-03-22]. Dostupné z:
http://www.st.com/content/ccc/resource/technical/document/application_note/51/5f/03/1e/bd/9b/45/be/CD00264342.pdf/files/CD00264342.pdf/jcr:content/translations/en.CD00264342.pdf

[7] UM1725 User Manual. *Description of STM32F4 HAL and LL drivers* [online]. c2017 [cit. 2018-03-22]. Dostupné z:

http://www.st.com/content/ccc/resource/technical/document/user_manual/2f/71/ba/b8/75/54/47/cf/DM00105879.pdf/files/DM00105879.pdf/jcr:content/translations/en.DM00105879.pdf

[8] AN4286 Application note. *SPI protocol used in the STM32 bootloader* [online]. c2017 [cit. 2018-04-03]. Dostupné z:
http://www.st.com/content/ccc/resource/technical/document/application_note/7a/8a/0a/8f/8f/38/47/c0/DM00081379.pdf/files/DM00081379.pdf/jcr:content/translations/en.DM00081379.pdf

[9] AN4221 Application note. *I2C protocol used in the STM32 bootloader* [online]. c2017 [cit. 2018-04-05]. Dostupné z:
http://www.st.com/content/ccc/resource/technical/document/application_note/4c/68/fe/72/a8/cd/47/83/DM00072315.pdf/files/DM00072315.pdf/jcr:content/translations/en.DM00072315.pdf

[10] AN3154 Application note. *CAN protocol used in the STM32 bootloader* [online]. c2016 [cit. Dostupné z:
http://www.st.com/content/ccc/resource/technical/document/application_note/56/94/0c/7d/63/f6/4d/96/CD00264321.pdf/files/CD00264321.pdf/jcr:content/translations/en.CD00264321.pdf

8 Přílohy na DVD

A) Technická dokumentace.

B) Firmware mikrokontroléru STM32F446RE.

C) Doplňující zdrojové kódy pro vývoj firmware.

D) Software.