

# **System pro programování a testování zařízení s procesory ARM**

Tomáš Juřena

---

Bakalářská práce  
2014



Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky  
akademický rok: 2013/2014

## ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Tomáš Juřena**  
Osobní číslo: **A11116**  
Studijní program: **B3902 Inženýrská informatika**  
Studijní obor: **Informační a řídicí technologie**  
Forma studia: **prezenční**

Téma práce: **Systém pro programování a testování zařízení s procesory ARM**

Zásady pro vypracování:

1. Prostudujte protokol IAP (In Application Programming) procesorů ARM Cortex M3 a M4.
2. Implementujte protokol IAP v jazyce C na procesoru ARM Cortex M3 nebo M4.
3. Realizujte prototyp zařízení pro programování procesorů ARM Cortex M.
4. Navrhněte postup pro testování nově naprogramovaných zařízení, které mají síťové rozhraní Ethernet.
5. Implementujte aplikaci pro testování takových zařízení.

Rozsah bakalářské práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. YIU, J. **The Definitive Guide to the ARM Cortex-M3**. Elsevier, 2007. ISBN 978-0-7506-8534-4.
2. SLOSS, A., D. SYMES a Ch. WRIGHT. **ARM System Developers Guide**. Elsevier, 2004. ISBN 1-55860-874-5.
3. VALVANO, J. W. **Embedded Systems: Real-Time Interfacing to Arm Cortex(TM)-M Microcontrollers**. CreateSpace Independent Publishing Platform, 2011. ISBN 978-1463590154.
4. VALVANO, J. W. **Embedded Systems: Introduction to ARM Cortex-M Microcontrollers**. CreateSpace Independent Publishing Platform, 2013. ISBN 978-1477508992.
5. VALVANO, J. W. **Embedded Systems: Real-Time Operating Systems for the ARM Cortex-M Microcontrollers**. CreateSpace Independent Publishing Platform, 2012. ISBN 978-1466468863.
6. SEAL, D. **ARM Architecture Reference Manual**. Addison-Wesley, 2001. ISBN 978-0201737196.
7. SLOSS, A., D. SYMES a C. WRIGHT. **ARM System Developers Guide**. Morgan Kaufmann, 2004. ISBN 978-1558608740.

Vedoucí bakalářské práce:

**Ing. Tomáš Dulík, Ph.D.**

Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce:

**28. února 2014**

Termín odevzdání bakalářské práce:

**13. června 2014**

Ve Zlíně dne 28. února 2014

prof. Ing. Vladimír Vašek, CSc.  
*děkan*



prof. Ing. Vladimír Vašek, CSc.  
*ředitel ústavu*

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....  
podpis diplomanta

## **ABSTRAKT**

Prezentovaný systém je určen pro hromadnou výrobu, kde zajišťuje programování zařízení s mikroprocesorem ARM Cortex M firmy STMicroelectronics pomocí protokolu IAP a následné testování naprogramovaných zařízení skrze jejich rozhraní Ethernet. Pro interakci s obsluhou programátor obsahuje LCD display a skupinu tlačítek. Díky tomu je možné cílové zařízení naprogramovat pomocí některého z více firmwarů uložených v paměti a případně měnit některé konstanty, jako např. MAC adresy programovaných zařízení. Další částí práce je testování zařízení s portem RJ-45 v teplotní komoře, kdy je zařízení vystavováno prudkým teplotním výkyvům pro odhalení hardwarových vad. Testování je realizováno pomocí posílání dat přes zařízení a měření ztrátivosti paketů v závislosti na různých teplotách v komoře. Systém graficky vykreslí průběh teploty a počet ztracených paketů v čase.

Klíčová slova: ARM, IAP, Cortex-M, testování, Ethernet

## **ABSTRACT**

Presented system aims to mass production where it is used to program devices with ARM Cortex M CPU by IAP protocol and to test the produced devices with Ethernet interface. For interaction with user, the programmer contains LCD display and group of buttons. Thanks to this solution, it is possible to program device with one of the firmware images stored in the internal memory of programmer and it is also possible to change some constants such as MAC address. Other part of this thesis describes a system for testing devices with RJ-45 connectors in temperature test chamber. The device is exposed to fast temperature changes to discover hardware errors. Testing is realized by sending packets through the tested device and measuring packets loss during different temperatures in chamber. System displays results of testing as graphs of packets loss over time.

Keywords: ARM, IAP, Cortex-M, testing, Ethernet

Rád bych poděkoval své rodině za podporu a pomoc při studiu, ale hlavně vedoucímu práce Ing. Tomáši Dulíkovi Ph.D, za ochotu a cenné připomínky při realizaci.

*„Vzdáte-li se kouření, pití a milování, nebudete ve skutečnosti žít déle, ale jen vám to tak bude připadat.“ - Sigmund Freud*

# OBSAH

|   |           |
|---|-----------|
| <b>ÚVOD</b> .....   | <b>9</b>  |
| <b>I TEORETICKÁ ČÁST</b> .....                            | <b>10</b> |
| <b>1 PROGRAMOVÁNÍ MIKROPROCESORŮ</b> .....                | <b>11</b> |
| 1.1 SOFTWARE EMBEDDED SYSTÉMŮ, FIRMWARE A BOOTLOADER..... | 11        |
| 1.1.1 Software embedded systémů.....                      | 11        |
| 1.1.2 Firmware.....                                       | 11        |
| 1.1.3 Bootloader.....                                     | 12        |
| 1.2 IN APPLICATION PROGRAMMING.....                       | 12        |
| 1.3 JOINT TEST ACTION GROUP - JTAG.....                   | 12        |
| 1.4 STRUKTURA PAMĚTI.....                                 | 13        |
| <b>2 TESTOVÁNÍ ZAŘÍZENÍ</b> .....                         | <b>14</b> |
| 2.1 ETHERNET.....   | 14        |
| 2.2 TESTOVACÍ KOMORA.....                                 | 14        |
| <b>II PRAKTICKÁ ČÁST</b> .....                            | <b>15</b> |
| <b>3 SYSTÉM PRO PROGRAMOVÁNÍ</b> .....                    | <b>16</b> |
| 3.1 SCHÉMA PLOŠNÉHO SPOJE PROGRAMÁTORU.....               | 16        |
| 3.2 OVLÁDÁNÍ A MENU.....                                  | 16        |
| 3.2.1 Hlavní menu.....                                    | 17        |
| 3.2.2 Nahrát FW.....                                      | 18        |
| 3.2.3 Cmds list.....                                      | 18        |
| 3.3 POPIS KOMPONENT NA PLOŠNÉM SPOJL.....                 | 19        |
| 3.4 INTERNÍ BOOTLOADER.....                               | 19        |
| 3.4.1 Inicializace bootloADERu.....                       | 20        |
| 3.4.2 Podporované příkazy.....                            | 20        |
| 3.5 PŘÍPRAVA PŘED PROGRAMOVÁNÍM.....                      | 21        |
| 3.6 NAHRÁVACÍ SMYČKA.....                                 | 21        |
| 3.7 SIMULACE EEPROM.....                                  | 23        |
| <b>4 NÁVRH TESTOVACÍ APLIKACE</b> .....                   | <b>25</b> |
| 4.1 ZAPOJENÍ PŘI TESTOVÁNÍ.....                           | 25        |
| 4.2 REÁLNÉ ZAPOJENÍ.....                                  | 25        |
| 4.3 PRINCIP TESTOVÁNÍ.....                                | 26        |
| 4.4 TESTOVACÍ SMYČKA.....                                 | 27        |
| <b>5 POUŽITÉ TECHNOLOGIE</b> .....                        | <b>28</b> |
| 5.1 RRDTOOL.....  | 28        |
| 5.2 SNMP.....   | 28        |
| 5.3 LAMP.....   | 28        |

|          |  |           |
|----------|--|-----------|
| 5.3.1    | MySQL.....                                     | 28        |
| 5.3.2    | PHP.....                                       | 29        |
| 5.4      | BASH.....                                      | 29        |
| 5.5      | VLAN.....                                      | 29        |
| <b>6</b> | <b>POUŽITÉ VÝVOJOVÉ NÁSTROJE.....</b>          | <b>30</b> |
| 6.1      | TASKING.....                                   | 30        |
| 6.2      | PHPSORM.....                                   | 30        |
| <b>7</b> | <b>IMPLEMENTACE.....</b>                       | <b>32</b> |
| 7.1      | IMPLEMENTACE PROGRAMÁTORU.....                 | 32        |
| 7.2      | IMPLEMENTACE SYSTÉMU PRO TESTOVÁNÍ.....        | 34        |
| 7.2.1    | Konfigurace switchů.....                       | 34        |
| 7.2.2    | Databáze.....                                  | 35        |
| 7.2.3    | Běh testovacího skriptu.....                   | 35        |
| 7.2.4    | Použité identifikátory SNMP.....               | 36        |
| 7.2.5    | Prohlížení souhrnných výsledků.....            | 36        |
| 7.2.6    | Detailní prohlížení výsledků.....              | 37        |
|          | <b>ZÁVĚR.....</b>                              | <b>38</b> |
|          | <b>ZÁVĚR V ANGLIČTINĚ.....</b>                 | <b>39</b> |
|          | <b>SEZNAM POUŽITÉ LITERATURY.....</b>          | <b>40</b> |
|          | <b>SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....</b> | <b>41</b> |
|          | <b>SEZNAM OBRÁZKŮ.....</b>                     | <b>42</b> |
|          | <b>SEZNAM TABULEK.....</b>                     | <b>43</b> |
|          | <b>SEZNAM PŘÍLOH.....</b>                      | <b>44</b> |

## ÚVOD

Impulesem pro napsání této práce byl požadavek na vytvoření systému, který by byl schopen programovat v hromadné výrobě síťová zařízení firmwarem uloženým v zařízení a poté otestovat správné hardwarové zapojení. Jelikož se jedná o poměrně specifické zadání, nebylo možné využít již hotových řešení. Systém cílí na mikroprocesory firmy STMicroelectronics jejichž interní bootloader lze snadno použít pro potřeby práce.

Druhou částí zadání je implementovat testovací sekvenci, která by byla schopna odhalit hardwarové chyby zapojení testovaného zařízení. Jelikož se jedná o síťové zařízení typu switch, který má spojeny porty RJ-45 do páru, je možné přes tyto páry posílat a poté počítat pakety. Testování probíhá v teplotní komoře, kvůli odhalení chyb zapojení, které by se mohly objevit pouze v jisté teplotě vlivem tepelné dilatace (roztahování a smršťování) desky plošných spojů a na ní osazených komponent.

V praktické části práce nejprve popisují systém pro programování mikroprocesorů. Zde je vysvětlena jeho architektura a způsob programování cílových zařízení. Poté popisují systém pro testování. V této kapitole je naznačeno, jaké podmínky musí být splněny, aby testování mohlo proběhnout a čeho se využívá při testování. Za těmito kapitolami následuje krátký přehled použitých technologií a nástrojů potřebných k vývoji. Nakonec je popsán postup implementace jednotlivých systémů.

## **I. TEORETICKÁ ČÁST**

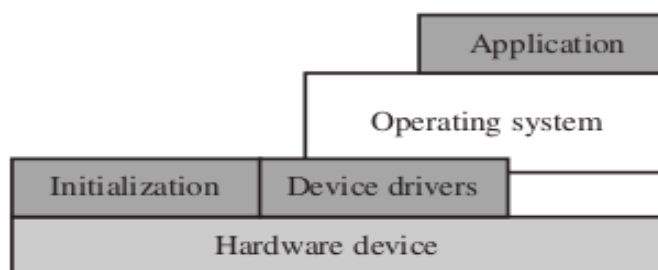
## 1 PROGRAMOVÁNÍ MIKROPROCESORŮ

V současné době není na trhu k dostání levné zařízení pro hromadné programování mikroprocesorů, které by si firmware pamatovalo ve své vnitřní paměti a mělo interaktivní ovládání pro snadnou obsluhu. Toto bylo jedním z důvodů, proč tato práce vznikla.

### 1.1 Software embedded systémů, Firmware a Bootloader

#### 1.1.1 Software embedded systémů

Embedded systémy potřebují software, aby mohly pracovat. Na obrázku Obr. 1 jsou zobrazeny čtyři typické softwarové komponenty, které jsou potřeba k ovládní embedded systémů. Každá vrstva používá vyšší stupeň abstrakce, oproti vrstvě pod ní, k ovládní systému. Inicializační kód je první provedený kód na desce a je specifický pro každé zařízení nebo skupinu zařízení.



Obr. 1 - Software spouštěný embedded systémy [1]

Operační systém poskytuje infrastrukturu pro kontrolu aplikací a správu hardwarových prostředků. Mnoho embedded systémů nepožaduje plný operační systém, ale jednoduchý plánovač úkolů, který je buď řízený událostmi nebo reakcí na zachycenou změnu stavu pinů. Ovladače zařízení (device drivers) jsou třetí komponentou na obrázku Obr. 1. Ty poskytují softwarové rozhraní k perifériím hardwaru zařízení. Posledním blokem je aplikace vykonávající požadovanou činnost. Takových aplikací může být více na stejném zařízení, všechny jsou ale řízeny operačním systémem. Jednotlivé softwarové komponenty mohou běžet z paměti RAM nebo ROM. Kód uložený v paměti ROM je neměnný a nazývá se firmware.[1]

#### 1.1.2 Firmware

Firmware je nízkoúrovňový software, který poskytuje rozhraní mezi hardwarem a aplikací/operačním systémem. Je uložen v ROM paměti a je spouštěn, když je zařízení připojeno ke zdroji. Firmware zůstává aktivní i poté, kdy je systém ziniclizován, a zajišť-

tuje základní systémové operace. Volba, jaký firmware použít pro konkrétní řešení, je odvozena podle zadání, kdy se může jednat o načtení a spuštění sofistikovaného operačního systému až po přenechání kontroly pro jiný mikroprocesor. Důsledkem toho jsou požadavky na firmware různé pro každou implementaci. Například malá aplikace může požadovat minimální podporu od firmwaru. Jedním z hlavních účelů firmwaru je poskytnutí stabilního mechanismu k načtení a nabootování aplikace/operačního systému.[1]

### 1.1.3 Bootloader

Bootloader je malá aplikace, která instaluje operační systém nebo aplikaci do cílového zařízení. Je volán pouze při spuštění aplikace nebo operačního systému, a je často integrován do firmwaru procesoru. Tento kód uvádí procesor ze stavu po restartu do stavu, kdy na něm může běžet aplikace nebo operační systém. Většinou se konfiguruje řadič paměti, cache procesoru a inicializují se některá zařízení nebo periférie.[1]

## 1.2 In Application Programming

IAP (In Application Programming) slouží k programování mikropočítačů pomocí různých periférií mikroprocesoru, kdy není nutné využívat primárního bootloADERU zařízení ani rozhraní JTAG nebo SWD, jehož konektory na produkčních deskách plošných spojů nebývají osazeny nebo nejsou snadno přístupné. Implementace IAP je často uložena v části interní ROM/flash paměti a je podle potřeby volána. Tohoto protokolu může být využíváno také při upgradování firmwaru zařízení.

Implementované IAP spouští operace zápisu a mazání v paměti přímo jako koncová aplikace uživatele.

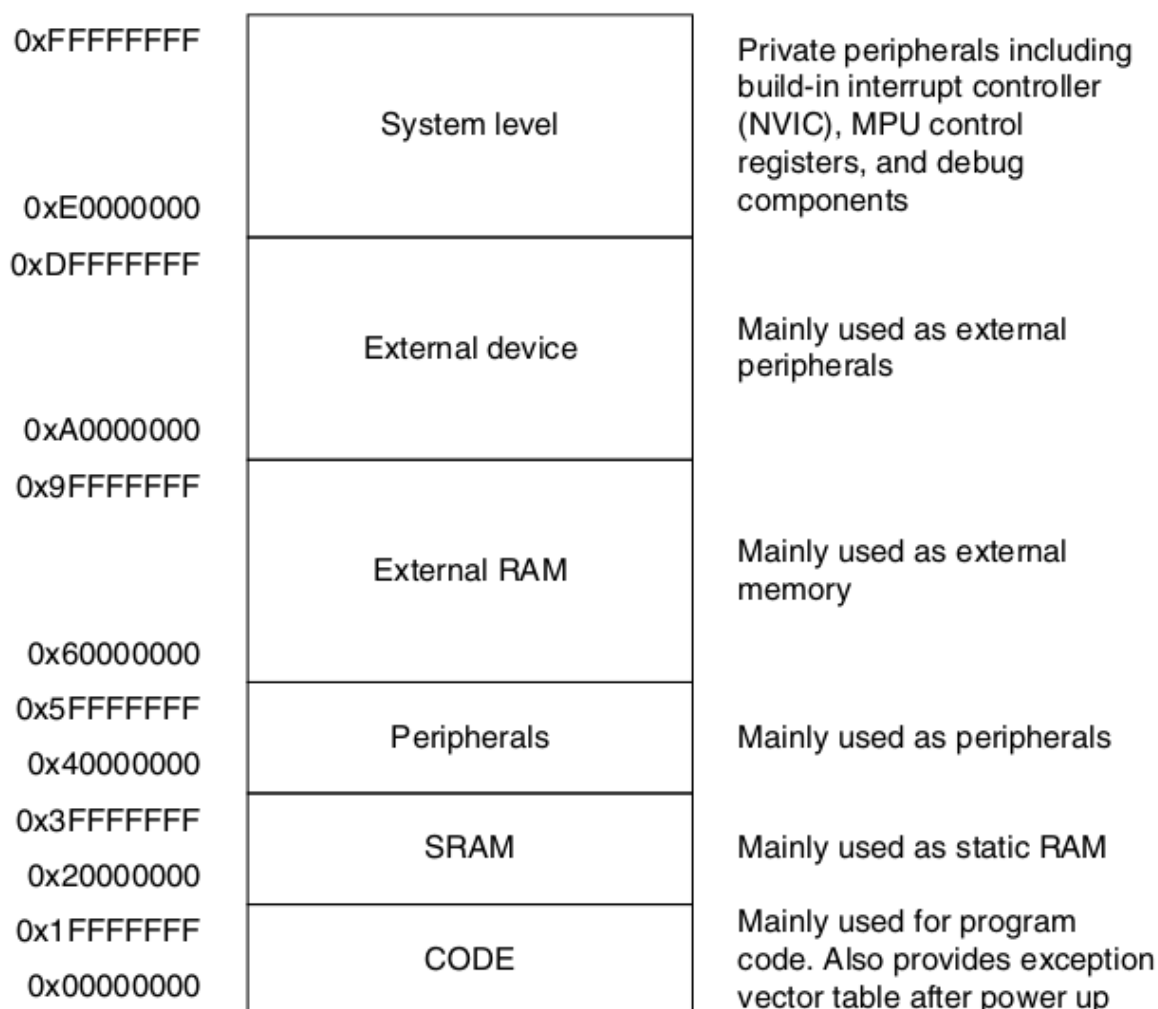
## 1.3 Joint Test Action Group - JTAG

Joint Test Action Group je standard definovaný normou IEEE 1149.1, tzv. Standard Test Access Port (TAP). Jedná se o architekturu Boundary-Scan pro testování plošných spojů, programování FLASH paměti apod. JTAG je možné použít kromě primárního účelu, kterým je testování plošných spojů a interní funkce obvodů, také k programování flash paměti, procesorů, FPGA, CPLD a dalších. K tomuto bylo vytvořeno několik standardů, např. IEEE 1532, JEDEC STAPL, nebo nestandardizovaný, ale hodně používaný Serial Vector Format.[2]

## 1.4 Struktura paměti

Mikroprocesory Cortex-M3 mají předdefinovanou strukturu paměti. Díky tomuto mohou být vestavěné periferie, jako jsou řadič přerušení nebo ladící nástroje, lehce přístupné pomocí jednoduchých instrukcí pro správu paměti. Většina systémových funkcí je tedy přístupná pomocí programu napsaného v C. Předdefinované rozložení paměti také dovoluje použít procesory Cortex-M3 použít v system-on-a-chip (SoC) návrzích díky své rychlosti a jednoduchosti integrace. Maximální velikost 4GB může být rozdělena do úseků jak je zobrazeno na obrázku Obr. 2.

Návrh Cortex-M3 má interní sběrniceovou strukturu optimalizovanou pro toto použití paměti. Navíc návrh dovoluje použít tyto úseky jinak. Například datová paměť může být umístěna v úseku kódu, ale aplikace může být spuštěna z externí RAM.



Obr. 2 - Struktura paměti Cortex-M3 [3]

Úsek „system level“ obsahuje řadič přerušení a ladící nástroje. Tyto zařízení mají fixované adresování, díky čemuž je možné snadněji portovat aplikace mezi různými Cortex-M3 čipy.[3]

## 2 TESTOVÁNÍ ZAŘÍZENÍ

Testování zařízení se síťovým rozhraním Ethernet probíhá v teplotní komoře, kdy se systém snaží odhalit hardwarové chyby výroby. Systém je vytvořen pro testování specifického síťového zařízení (POE panel) podobného switchi, a pro správnou funkčnost navržený systém potřebuje určitý způsob propojení portů, aby mohlo testování proběhnout.

### 2.1 Ethernet

Ethernet je název souhrnu technologií pro lokální počítačové sítě (LAN) z větší části standardizovaných jako IEEE 802.3, které používají kabely s kroucenou dvoulinkou, optické kabely (a dříve i koaxiální kabely) pro komunikaci přenosovými rychlostmi od 10 Mbit/s po 10 Gbit/s. Síť Ethernet realizují fyzickou a linkovou vrstvu referenčního modelu OSI, takže je možné po nich provozovat jeden nebo více protokolů síťové vrstvy, například AppleTalk, DECnet, IPX/SPX a především protokoly IP a IPv6, které se používají pro služby sítě Internet.

Ještě před rokem 2000 se Ethernet stal dominantní technologií pro drátové nebo kabelové lokální sítě a prakticky synonymem pro lokální síť (LAN). Používá se nejen pro propojování počítačů, ale i pro datová úložiště, zařízení spotřební elektroniky jako jsou televizní přijímače a herní konzole a také jako drátové rozhraní pro přístupové body WiFi a zařízení pro přístup k Internetu. Pokud zařízení deklaruje, že má připojení na LAN, v naprosté většině případů to znamená, že je vybaveno konektorem 8P8C (RJ-45) pro síť Ethernet s rychlostí 100 nebo 1000 Mbit/s.[4]

### 2.2 Testovací komora

Testovací komory mají širokou škálu použitelnosti a jsou využívány pro velké množství testů například při vývoji nového zařízení, testování finálního výrobku nebo kdekoli, kde je potřeba zjistit vliv prostředí. Komory je možné použít při zkoumání vlivu velkého množství faktorů jako jsou například:

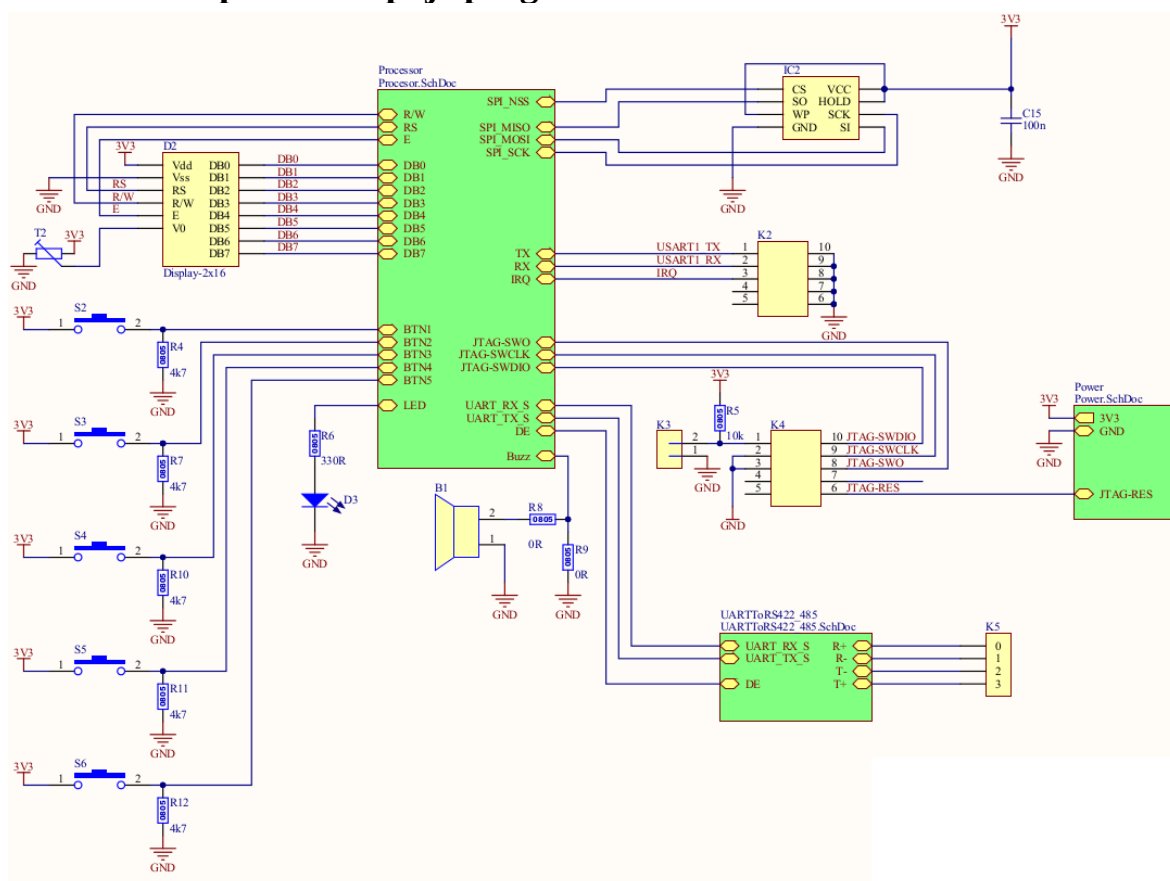
- Životnost,
- Teplota,
- Vítr a prach,
- atd..

## **II. PRAKTICKÁ ČÁST**

### 3 SYSTÉM PRO PROGRAMOVÁNÍ

Systém pro programování (dále jen programátor), je plošný spoj, který komunikuje s cílovým zařízením po periférii RS232 a využívá interního bootloaderu mikroprocesoru k tomu, aby do cílového zařízení byl přenesen požadovaný binární soubor. Zvolený soubor může být po výběru editován (např. změna konstanty).

#### 3.1 Schéma plošného spoje programátoru

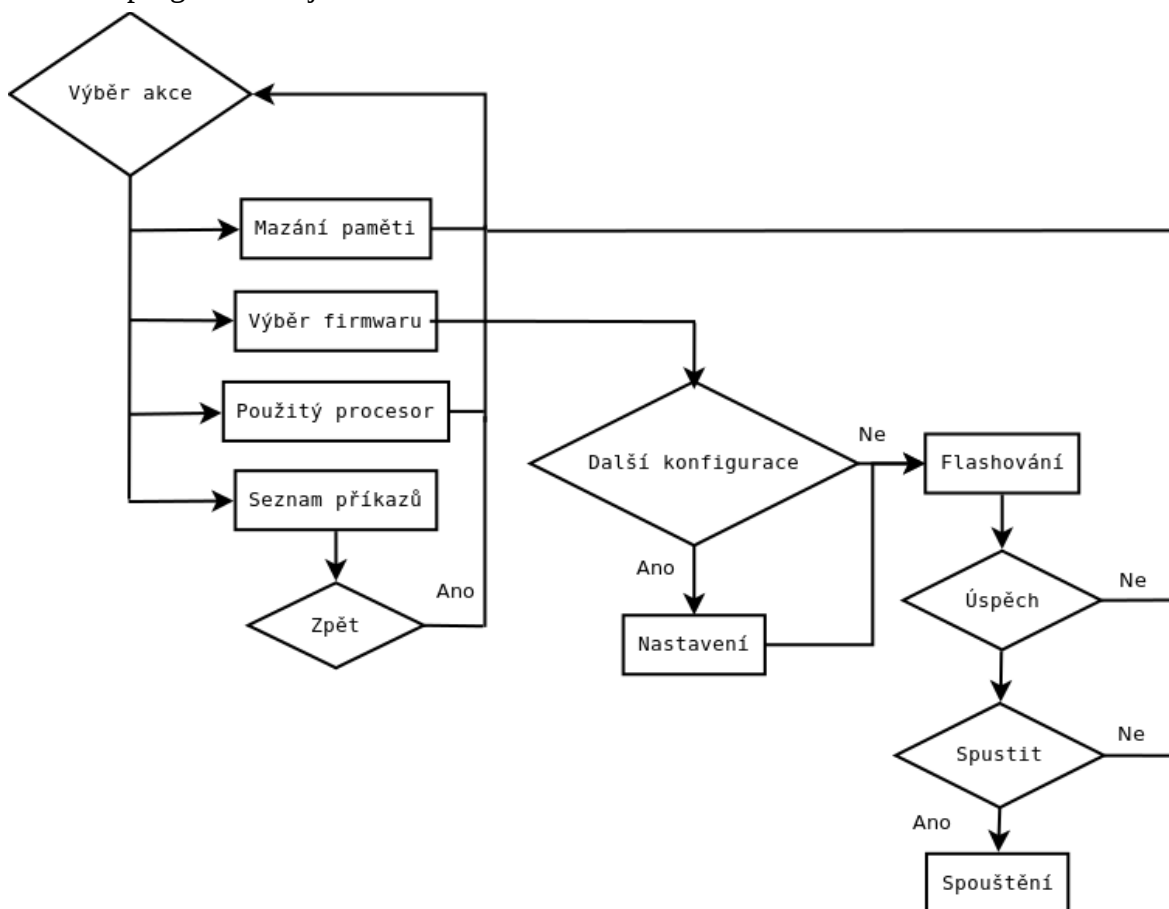


Obr. 3 - Zjednodušené plošného spoje

#### 3.2 Ovládání a menu

Pro ovládání programátoru je využito pět tlačítek, přičemž čtyři jsou využity jako směrové šipky pro pohyb v menu a jedno tlačítko je potvrzovací. Pro případy, kdy by programátor skončil kvůli nějaké chybě v nekonečné smyčce, je zde ještě šesté tlačítko pro reset zařízení. Rozhraní je realizováno pomocí znakového LCD displaye, který zobrazuje aktuální vybranou položku v menu nebo informace o činnosti. Kromě samotného programování zařízení je možné číst informace o použitém bootloadru nebo o typu použitého mikroprocesoru, takže menu je obsáhlejší a dostupné informace komplexnější. Menu je vytvořeno spíše pro demonstrační účely programátoru. Při reálné výrobě bude pravdě-

podobně obsahovat pouze seznam binárních souborů, které v něm budou uloženy. Schéma ovládání programátoru je uvedeno níže na obrázku Obr. 4.



Obr. 4 - Hierarchie menu

Interakce uživatele a programátoru je tedy realizována pomocí LCD displeje, na který se zobrazuje jednak menu, ale také aktuální činnost. Po restartu zařízení jsou vypisovány informace o snaze programátoru navázat spojení s cílovým zařízením. Po navázání se získávají informace o použitém mikroprocesoru a stahuje se seznam dostupných příkazů, které interní bootloader cílového zařízení umí zpracovat. Jakmile jsou tyto činnosti dokončeny, je uživateli zobrazeno menu. Toto menu se skládá z více úrovní, přesněji popsanych níže, mezi kterými je možno přecházet stiskem tlačítka potvrdit na požadované položce.

### 3.2.1 Hlavní menu

Po úspěšném spuštění programátoru je uživateli zobrazeno hlavní menu. V hlavním menu jsou zobrazeny čtyři položky:

- Nahrát FW – vybráním této položky se zobrazí seznam dostupných binárních souborů, které je možné nahrát do zařízení.

- Smazat FW – pokusí se smazat paměť druhého zařízení. Tato položka nemá žádné podmenu.
- Cmds list – vypíše seznam dostupných příkazů podporovaných v druhém zařízení.
- Detect MCU – zobrazí informaci o tom, jaký je použit mikroprocesor v připojeném zařízení. Tato položka nemá žádné podmenu.

### 3.2.2 Nahrát FW

Po zvolení této položky je vypsán seznam všech dostupných binárních souborů. Tento seznam lze cyklicky dokola procházet a návrat o úroveň výš je možné zvolením volby *zpět* na poslední položce seznamu. Pokud obsluha vybere soubor, který chce nahrát, a není třeba upravovat žádnou konstantu v tomto souboru, přejde se rovnou k nahrávání. Jestliže je ale možné soubor před nahráním upravit, zobrazí se na displayi buď hodnota poslední úspěšně nahrané konstanty (např. poslední nahraná MAC adresa nebo sériové číslo), anebo hodnota výchozí (např. adresa ke skoku programu).

### 3.2.3 Cmds list

Toto podmenu obsahuje seznam příkazů, které je možné využít. Jedná se opět o cyklický seznam, ze kterého se dá vrátit zpět stejně jako v případě podmenu *Nahrát FW*.

### 3.3 Popis komponent na plošném spoji



Obr. 5 - Vyrobený plošný spoj programátoru

Rozložení komponent na vyrobeném plošném spoji:

1. Napájecí konektor
2. Port JTAG
3. Port RS232
4. LCD display
5. Směrová tlačítka
6. Potvrzovací tlačítko
7. Resetovací tlačítko

### 3.4 Interní bootloader

Programátor se zaměřuje na mikroprocesory společnosti STMicroelectronics, proto budu dále popisovat příkazy a postup inicializace interního bootloaderu právě od této firmy.

### 3.4.1 Inicializace bootloaderu

Aby bylo možné vůbec využívat bootloader, je nutné, aby jej programované zařízení inicializovalo. Tohoto je dosaženo přivedením napětí na pin BOOT0 a resetováním zařízení. Bootloader se poté sám inicializuje vstupem do systémové paměti a vykonání programu, který je v ní uložen. Přesný postup inicializace pro jednotlivé periferie a mikroprocesory je uveden v Application note AN2606 (k nalezení na <http://www.st.com>), kde jsou uvedeny všechny potřebné detaily.

Jakmile je bootloader zinicializován je možné odesílat jednotlivé příkazy pro práci s flash pamětí mikroprocesoru i mikroprocesorem samotným. Seznam používaných příkazů je uveden v tabulce Tab. 1.

### 3.4.2 Podporované příkazy

Seznam podporovaných příkazů se liší podle zařízení a použité verze bootloaderu, ale mělo by se vždy jednat o kombinaci níže uvedených.

Tab. 1 - Seznam příkazů podporovaných interním bootloaderem [5]

| <b>Příkaz</b>                     | <b>Popis</b>  |
|-----------------------------------|---|
| Get                               | Získá verzi a povolení příkazy v aktuální verzi bootloaderu.                    |
| Get version and protection status | Získá verzi bootloaderu a stupeň ochrany paměti                                 |
| Get ID                            | Získá ID použitého mikroprocesoru   |
| Read memory                       | Přečte až 256 bytů paměti na požadované adrese                                  |
| Go                                | Spustí uživatelský kód na zadané adrese ve Flash nebo SRAM                      |
| Write memory                      | Zapíše až 256 bytů na požadovanou adresu  |
| Erase                             | Smaže od jedné po všechny stránky paměti  |
| Extended erase                    | Smaže od jedné po všechny stránky paměti, využívá přitom dvoubytový adresní mód |
| Write protect                     | Zakáže zápis do paměti  |
| Write unprotect                   | Povolí zápis do paměti  |
| Read protect                      | Zakáže čtení z paměti   |
| Read unprotect                    | Povolí čtení z paměti   |

Každý z těchto příkazů má návratovou hodnotu ACK nebo NACK, kde ACK znamená úspěšné provedení příkazu nebo jeho části. Hodnota NACK je odeslána pokud se příkaz nepodaří provést, ale i v případě, že se programátor snaží vykonat příkaz, který již jednou požadoval a stav bootloaderu se od té doby nezměnil. Příkladem může být odstranění ochrany paměti. Pokud by se o něj požádalo dvakrát za sebou a již první příkaz se úspěšně provedl, při druhém volání se vrátí NACK, jelikož již není možné ochranu odstranit znovu.

Některé příkazy (odstranění ochrany paměti, zápis na option byte adresu) vyvolávají softwarový restart zařízení, na tuto vlastnost je potřeba brát ohled a upravit vybrané příkazy tak, aby zachovaly bootloader ve stejném stavu, při jakém byly volány. Jedná se tedy o rozšíření daných příkazů o opětovnou inicializaci bootloaderu, který se restartem deinitializoval.

### 3.5 Příprava před programováním

Před tím, než je možné s programátorem programovat zařízení, musí obsluha přidat do firmwaru programátoru binární soubor převedený do formy bytového pole. Programátor má pro výpis dostupných binárních souborů vytvořeno vnitřní pole, ve kterém každý prvek popisuje jeden soubor. Jedná se o pole struktur, kde každá struktura má uloženy informace o názvu souboru, velikosti nebo adrese, na kterou se má nahrávat. Do tohoto pole je nutné vyplnit žádané údaje, aby byl soubor pro programátor viditelný. Teprve poté se zobrazí mezi ostatními v seznamu. Jakmile je toto splněno musí obsluha kód zkompilovat a nahrát do programátoru pomocí portu JTAG. Po nahrání nového softwaru do programátoru, je nutné propojit obě zařízení. Na desce programátoru se jedná o port RS232, přičemž první pin se připojuje na Tx pin cílového zařízení a druhý pin na Rx pin. Pro sjednocení napěťových úrovní je třeba spojit ještě zemnicí vodiče. Jestliže jsou výše uvedené pokyny úspěšně provedeny, je možné začít s programátorem pracovat. Práce s programátorem je velmi jednoduchá a intuitivní díky LCD displayi. Nutnou podmínkou je dále vstup do systémové paměti programovaného zařízení, aby mohl být inicializován bootloader. Dále popisovaný postup se může lišit podle použitého procesoru, ale pro mikroprocesory od společnosti od STMicroelectronic by mělo být dostatečné přivést napětí na pin BOOT0, přesný postup je uveden v dokumentaci mikroprocesoru. Pokud bylo zařízení již před tímto bodem zapnuto, je nutné jej restartovat, aby zařízení vstoupilo do systémové paměti a bootloader se zinicializoval. V tento moment již stačí zapnout/restartovat programátor. Zařízení by měla začít komunikovat a mělo by být zobrazeno hlavní menu. Jestliže tomu tak není, stala se chyba při inicializaci komunikace. Pokud jsou správně zapojeny kabely a na pinu BOOT0 je přivedeno dostatečné napětí, tak chyba pravděpodobně vznikla tím, že programátor odeslal synchronizační byte příliš brzy. V takovémto případě stačí stačí resetovat programované zařízení, asi vteřinu počkat a poté resetovat programátor.

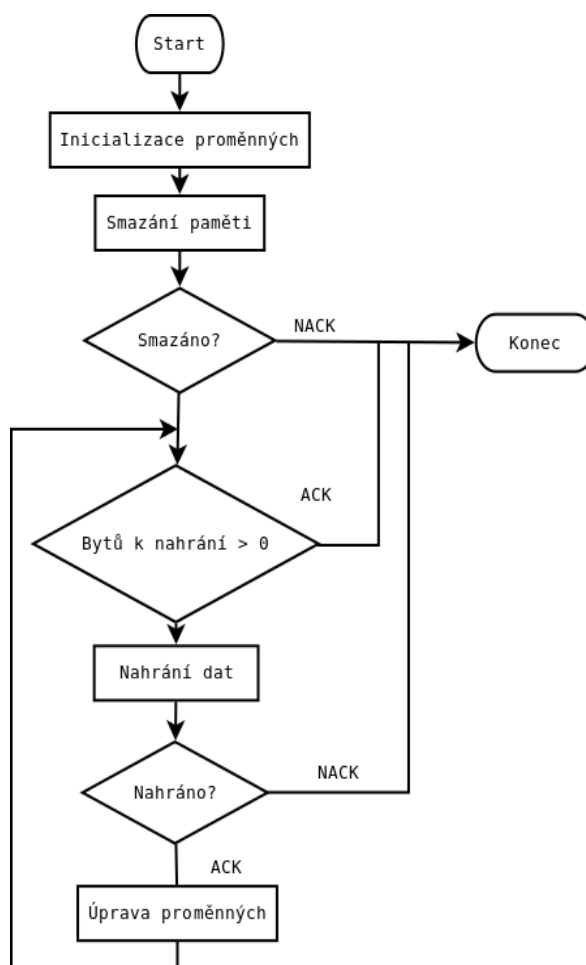
### 3.6 Nahrávací smyčka

Jestliže je cílové zařízení úspěšně zinicilizováno a obsluha zvolí možnost naprogramování některého z binárních souborů uloženého v paměti, programátor se poté snaží tento zvolený soubor přenést. Postupuje se podle smyčky na obrázku Obr. 6.

Zařízení je potřeba před každým nahráváním vymazat, jelikož většinou obsahují paměti typu flash, které nedovolují měnit jednotlivé bity, ale mohou se mazat pouze celé sektory. Pokud by zařízení nebylo smazáno, potom by proces nahrávání skončil chybou.

Zobrazený diagram je jednodušší kvůli přehlednosti, nicméně vystihuje činnost, kterou programátor vykonává. Výstupem z této sekvence je hodnota ACK nebo NACK, na kterou reaguje hlavní smyčka programu oznámením obsluze, jestli se činnost podařila.

Jelikož je plánováno využít programátor pro nahrávání softwaru do zařízení v hromadné výrobě, je často nutné si pamatovat určitou konfiguraci, která se mění od každého vyrobeného zařízení. Při vývoji se na tuto skutečnost myslelo, proto je možné načítat poslední zvolenou konfiguraci z flash paměti programátoru a stejně tak do ní i zapisovat. K tomuto účelu byla napsána knihovna, která simuluje chování EEPROM, aby se co nejvíce šetřila interní flash paměť programátoru.



Obr. 6 - Nahrávací sekvence

### 3.7 Simulace EEPROM

Pro uchovávání dat v programátoru je využita interní paměť typu flash. Tento typ paměti neumožňuje přepisovat jednotlivé paměťové buňky libovolně z jakéhokoli stavu a lze mazat pouze celé sektory paměti. Jelikož se flash paměti mazáním snižuje její životnost, je vhodné optimalizovat práci s ní. Bylo by zbytečné při každém zapsání do paměti smazat celý, například 16 kilobytový, sektor, když program potřebuje změnit jeden byte nebo dokonce bit. Pro tento účel jsem napsal knihovnu, která požadavky na zápis do paměti řeší tak, že když je potřeba zapsat data, vloží je na první volnou pozici v paměti. Případně, pokud je technicky možné přepsat jenom buňku, která aktuálně uchovává data, zapsat je do této buňky a přepočítat CRC. Tímto přístupem se životnost paměti dramaticky zvýší a sníží se náchylnost k chybám, pokud by v průběhu mazání byl programátor odpojen od zdroje napájení, což by mělo za následek ztrátu dat uložených v RAM.

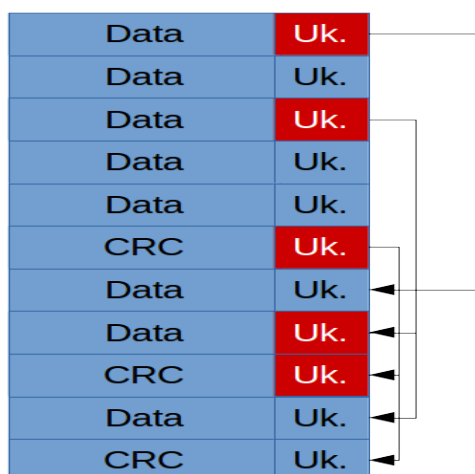
Simulace je řešena tak, že jsou vyhrazeny dva sektory interní paměti, kdy jsou do obou sektorů ukládána stejná data, přičemž druhý ze sektorů slouží pro zálohu a obnovu poškozených dat. Data o šířce 16 bitů jsou ukládána do buněk paměti a jejich správnost je ověřována kontrolním CRC součtem. Každá obsazená datová buňka má vyhrazenou další buňku, která drží informaci o tom, jestli jsou data v buňce validní. Při změně obsahu buňky je poté přepsána hodnota v této validační buňce na hodnotu adresy nových validních dat. Tento způsob odkazování je inspirovaný lineárním seznamem. Po zápisu je spočítáno CRC sektoru a rovněž zapsáno.

Při čtení dat z paměti se knihovna snaží přečíst data z prvního sektoru a spočítat CRC. Pokud CRC nesouhlasí, pokusí se knihovna o načtení ze záložního sektoru. Pokud jsou data poškozena i zde knihovna smaže oba dva sektory a nahraje výchozí data. Při správném přečtení jsou data uložena na adresu předanou v argumentu funkce.

| Address    | 0    | 2    | 4    | 6    | 8    | A    | C    | E    |
|------------|------|------|------|------|------|------|------|------|
| 0x08008000 | 000C | 0000 | 007B | 0000 | 0000 | 0000 | FFFF | 00FF |
| 0x08008010 | 001C | FFFF | 0020 | FFFF | FFFF | FFFF | 34FF | 0024 |
| 0x08008020 | 0002 | FFFF | 05FF | FFFF | FFFF | FFFF | FFFF | FFFF |
| 0x08008030 | FFFF | FFFF | FFFF | FFFF | FFFF | FFFF | FFFF | FFFF |

Obr. 7 - Paměť mikroprocesoru v aplikaci ST-Link Utility

Na obrázku Obr. 8 je zobrazen princip, jak jsou data v paměti provázána. Obrázek Obr. 7 zobrazuje výstup z programu ST-Link Utility, kterým lze přečíst obsah v paměti mikroprocesoru.



**Uk.** Ukazatel na další data

Obr. 8 - Princip uložení dat v paměti

```

/**
 * @brief Vytažení poslední konfigurace z paměti
 *
 * @param config: ukazatel na místo paměti, kam se mají data uložit
 *
 * @param size: velikost dat, která se mají načíst
 *
 * @retval vrací PERSISTENTDATAPASSED pokud se načtení povede, PERSISTENTDATAFAILED pokud ne, PERSISTENTDATACRC_MISMATCH
 */
TERROR PersistentData_get(uint16_t * config,uint16_t size)
{
    uint16_t count = 0,data,validity, crc, crc_computed, lastCRCAddress;
    FLASH_Unlock();
    FLASH_ClearFlag(FLASH_FLAG_EOP | FLASH_FLAG_OPERR | FLASH_FLAG_WRPERR | FLASH_FLAG_PGAERR | FLASH_FLAG_PGPER |FLASH_

    crc = PersistentData_crc16_get(FLASH_ADDR[aktivniSektor],&validity,&lastCRCAddress);
    crc_computed = PersistentData_crc16_compute(FLASH_ADDR[aktivniSektor],0);
    if(crc != crc_computed){/*test jestli poslední zapsané CRC souhlasí s nově spočítaným,
                             pokud ne, pokusí se spočítat CRC druhého sektoru. jestli nesouhlasí ani to,
                             vrátí PERSISTENTDATACRC_MISMATCH, jinak vytvoří kopii sektoru kde je CRC v pořádku*/
        aktivniSektor = PersistentData_SwitchSector();
        crc = PersistentData_crc16_get(FLASH_ADDR[aktivniSektor],&validity,&lastCRCAddress);
        crc_computed = PersistentData_crc16_compute(FLASH_ADDR[aktivniSektor],0);
        if(crc == crc_computed){//kopíruje sektor
            if(PersistentData_CopySector(FLASH_ADDR[ADDR1],FLASH_ADDR[ADDR2], lastCRCAddress) != PERSISTENTDATAPASSED)
                return PERSISTENTDATAFAILED;
        }
        else
            return PERSISTENTDATACRC_MISMATCH;
    }
    while(count < size){
        PersistentData_ReadRawData(&validity,2,FLASH_ADDR[aktivniSektor] + size + count + 4);
        PersistentData_ReadRawData(&data,2,FLASH_ADDR[aktivniSektor] + count + 4);
        while(validity != 0xFFFF){
            PersistentData_ReadRawData(&data,2,FLASH_ADDR[aktivniSektor] + validity);
            PersistentData_ReadRawData(&validity,2,FLASH_ADDR[aktivniSektor] + validity + 2);
        }
        (*config) = data;
        config++;
    }
}

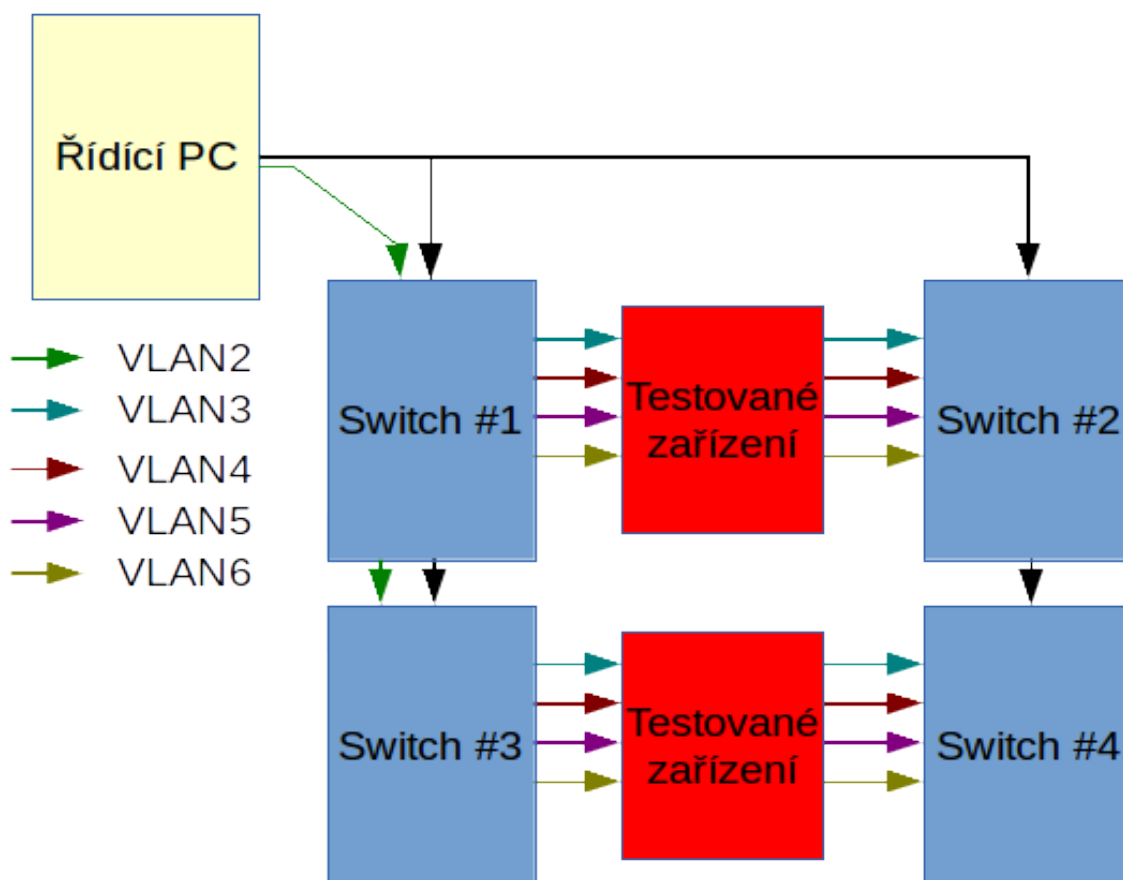
```

Obr. 9 - Kód funkce na získání dat z interní paměti

## 4 NÁVRH TESTOVACÍ APLIKACE

Jak jsem již výše zmiňoval, pro testování je využito testovací komory, ve které se cyklicky střídá teplota a sleduje se její vliv na chování testovaného zařízení. Testované zařízení je podobné switchi s tím rozdílem, že má spojeny jednotlivé porty do párů. Při testování se hledají chyby právě na těchto párech portů.

### 4.1 Zapojení při testování



Obr. 10 - Blokové schéma zapojení testovacího systému

Ze schématu je patrné, že celý proces testování řídí jeden počítač, ke kterému jsou pomocí UTP kabelu připojeny switche. Switche 1 a 3 slouží jako tzv. transmitters, které přijímají data odesílané řídicím počítačem a dále je poté přeposílají přes testované zařízení ke switchům 2 a 4, které slouží jako tzv. receivers. Počet switchů není omezen, stačí je pouze správně nakonfigurovat a přidat jejich adresy do testovacího skriptu, takže lze relativně snadno rozšiřovat testovací systém o další switche a otestovat tak více zařízení současně.

### 4.2 Reálné zapojení

Na obrázku Obr. 11 je fotka fyzického zapojení, kterého jsem při vývoji využíval. Je zde vidět jak jsou propojeny jednotlivé porty testovaného zařízení s porty switchů. Číselné popisy označují:

1. Transmitter
2. Receiver
3. Testovaný modul



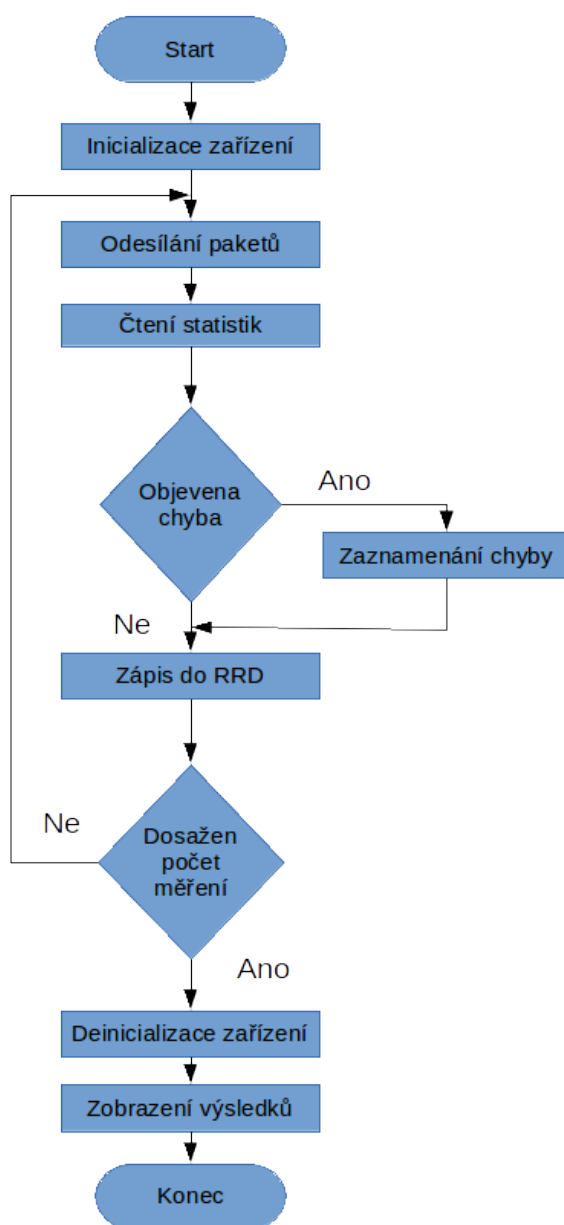
*Obr. 11 - Reálné zapojení testovacího systému*

Řídící počítač je připojen do transmitteru do portu 8. Přes tento port je přenášena veškerá komunikace, ať už pro získávání údajů o portech, nebo při přenosu broadcastu. Na prvním transmitter switchi připojeným k řídicímu počítači jsou vyhrazeny ještě porty 10 a 11 pro přenos komunikace. Přes port 10 je přenášena komunikace a broadcast na další transmitter switche a na portu 11 je přenášena komunikace na receiver switche.

### **4.3 Princip testování**

Principem testování je jednoduché odesílání paketů na všesměrovou adresu virtuální LAN (VLAN2 na obrázku Obr. 10) řídicího počítače, odkud jsou pakety přeposílány na switche (transmittery). Ty poté opět přepošlou pakety, tentokrát již přes samotná testovaná zařízení. Počet přijatých paketů na receiver switchích se srovná s počtem odeslaných paketů. Jestliže je odchylka mezi počtem přijatých a odeslaných paketů, zaznamená se tento rozdíl jako chyba v dané teplotě.

#### 4.4 Testovací smyčka



Obr. 12 - Zobecněné schéma testovací smyčky

## 5 POUŽITÉ TECHNOLOGIE

### 5.1 RRDtool

Nástroj RRDtool jsem využil při sběru dat pro vykreslení grafů testování, jak v průběhu tak po skončení testování a prohlížení výsledků.

RRDtool je průmyslový standard s otevřeným kódem, který má vysokou výkonost při logování a grafickém znázornění dat v čase. RRDtool může být lehce integrován do projektů psaných v shellu, Perlu, Pythonu, Ruby, Tcl, Lua, PHP a dalších. Pro záznam dat je použita databáze round-robin (odtud zkratka RRD – round-robin database), která uchovává data po přesných krocích.[6][7]

### 5.2 SNMP

K získávání informací o switchích jsem využil protokolu SNMP, který je implementován v PHP od verze 5.4, takže nebylo nutné psát další knihovnu, která by byla schopná informace ze switchů číst. Pro získávání informací jsou využívány OID identifikátory, což je sekvence čísel oddělených tečkou. Tečky rozdělují jednotlivé úrovně protokolu kategorií pro snadnější orientaci.

SNMP (Simple Network Management Protocol) je součástí sady internetových protokolů. Slouží potřebám správy sítí. Umožňuje průběžný sběr nejrůznějších dat pro potřeby správy sítě, a jejich následné vyhodnocování. Na tomto protokolu je dnes založena většina prostředků a nástrojů pro správu sítě.[8]

### 5.3 LAMP

Tato zkratka označuje sadu svobodného softwaru použitého při tvorbě dynamického webu. Jedná se o distribuci operačního systému Linux, kde je nainstalován webový server Apache, který poskytuje také databázový systém (např. MySQL) a skriptovací jazyk (PHP, Python, Perl).

#### 5.3.1 MySQL

MySQL je s více než 100 miliony celosvětově distribuovanými kopiemi nejpopulárnějším svobodným databázovým softwarem. Díky vysoké rychlosti, spolehlivosti a snadnosti použití se MySQL stalo oblíbenou volbou mnoha společností díky odstranění hlavních problé-

mů s údržbou a administrací moderních online aplikací.[9]

### 5.3.2 PHP

PHP jsem využil ke spojení se switchi pomocí SNMP, pro připojení k databázovému serveru a zpracování výsledků měření.

Zkratka PHP je rekurzivní akronym pro *PHP: Hypertextový Preprocesor*. PHP je velmi rozšířený a pro všeobecné užívání určený open source skriptovací jazyk, který je vhodný zvláště pro programování na webu a může být vnořen do HTML.[10]

### 5.4 BASH

Linux využívá pro práci s příkazovou řádkou různé shelly. Já jsem pro spuštění testování použil shell bash, se který mám nastavený jako výchozí, takže jsem nemusel po otestování příkazu řešit kompatibilitu s jinými shelly. Jedná se o Unixový procesor, který vykonává příkazy postupně, jak jsou uvedeny za sebou ve zdrojovém skriptu.

### 5.5 VLAN

Virtuální LAN je logicky nezávislá síť v rámci jednoho nebo několika zařízení. Virtuální síť lze definovat jako domény všesměrového vysílání (stejně jako LAN) s cílem učinit logickou organizaci sítě nezávislou na fyzické vrstvě, čímž lze usnadnit správu sítě, zvýšit její výkon a podpořit bezpečnost. Obvykle bývá realizována na zařízeních zvaných přepínač, jehož porty se rozdělí na několik logicky samostatných částí. Jde o dělení sítě už na úrovni 2. vrstvy ISO/OSI, v porovnání s podsítěmi na 3. vrstvě.[11]

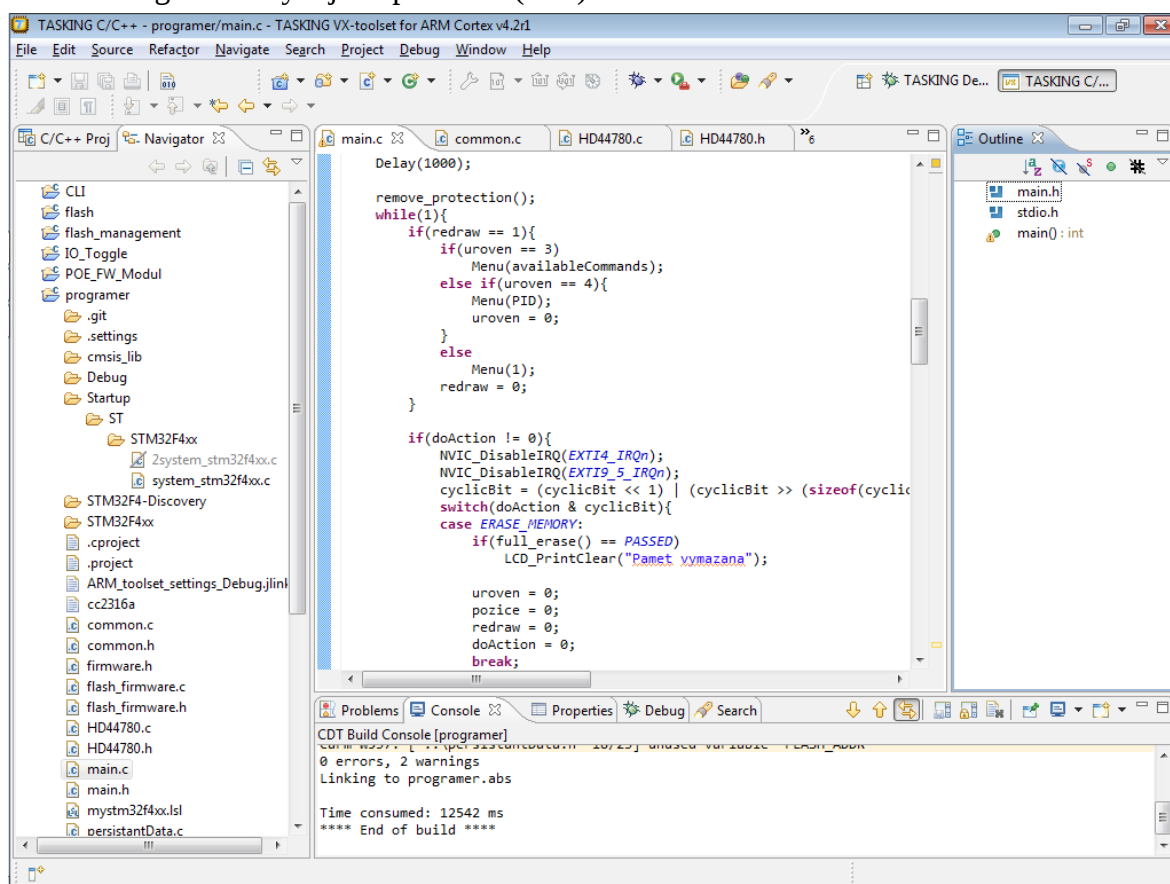
## 6 POUŽITÉ VÝVOJOVÉ NÁSTROJE

Jelikož jsem při realizaci tvořil dvě aplikace, kdy jedna byla psána v C pro mikroprocesor a druhá v PHP, která běží na operačním systému Linux. Použil jsem hlavně dva nástroje, které mi práci ulehčili díky své propracovanosti.

### 6.1 Tasking

Tasking od společnosti Altium je prostředí pro vývoj a ladění embedded aplikací. Poskytuje nástroje jako jsou:

- Kompilátor
- Pokročilé (vícejádrové) linkery a pomocné utility
- Debugery
- Integrované vývojové prostředí (IDE)



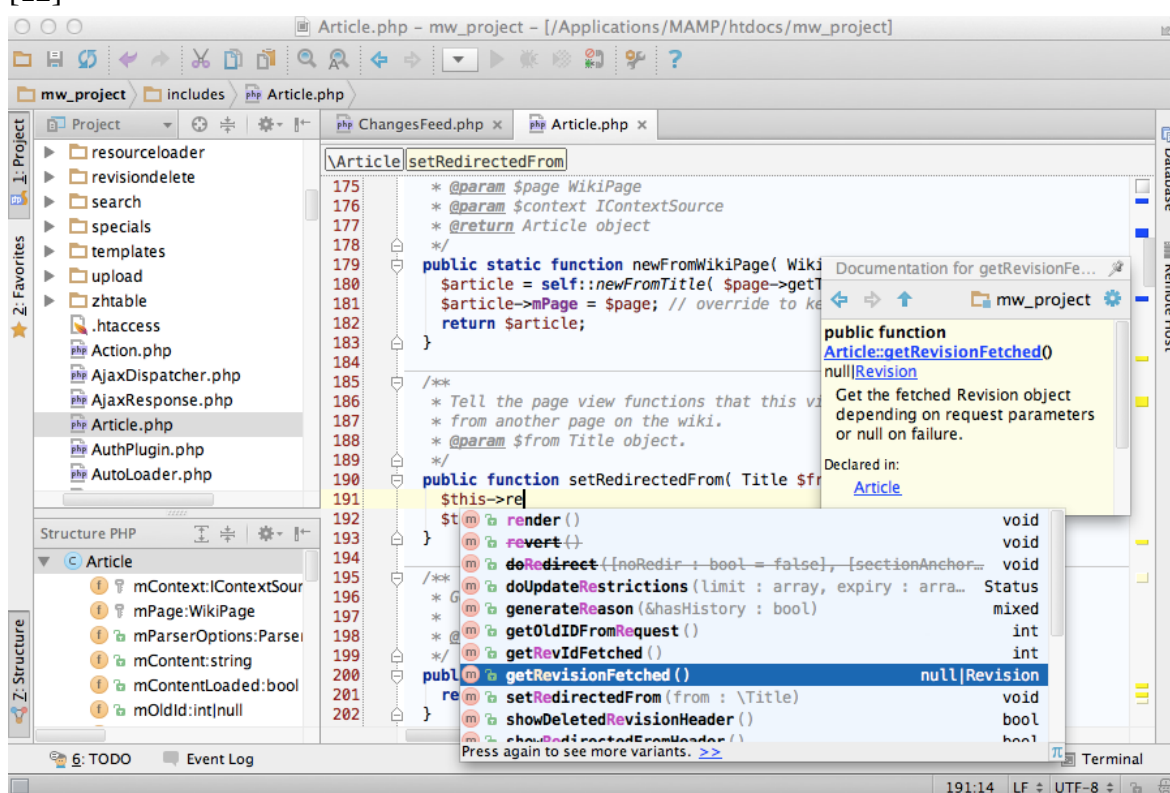
Obr. 13 - Prostředí Tasking

### 6.2 PHPStorm

Jedná se o nástroj pro vývoj webových aplikací s množstvím velmi kvalitních funkcí jako jsou:

- Podpora aktuální verze jazyka PHP
- Doplnění kódu
- Refaktorizace
- Formátování kódu
- Podpora frameworků

[12]



Obr. 14 - PhpStorm [12]

## 7 IMPLEMENTACE

Při samotné implementaci jsem nejprve jsem nejprve vyvíjel systém pro programování, protože na ten byly kladeny větší nároky.

### 7.1 Implementace programátoru

Při vývoji systému pro programování bylo nejprve nutné nastudovat API k bootloaderu a poté jej využít k samotnému programování zařízení. Postupoval jsem tedy tak, že jsem podle vývojových diagramů napsal jednotlivé příkazy, které bylo třeba pro programování použít. Jelikož jsem se rozhodl pro komunikaci využívat sériovou linku RS232, musel jsem ji zprovoznit. To se, díky ukázkovým příkladům dostupným na webových stránkách výrobce, ukázalo jako značně triviální. Bylo především nutné správně nakonfigurovat sériovou linku, aby bootloader správně detekoval synchronizační byte a správně dopočítal baud rate. Hodnota použité baud rate by se podle dokumentace měla pohybovat mezi 1200 a 115 200 b/s. Při inicializaci komunikace bootloader požaduje, aby komunikace probíhala s délkou slova 8 bitů, byla zajištěna sudou paritou a generovala 1 stop bit.

Nahrávání binárního kódu do zařízení je proces, při kterém jsou na cílové zařízení posílány byty přes komunikační rozhraní. Předtím, než je zařízení schopné přijímat jakákoli data od programátoru je nutné, aby zařízení mělo inicializovaný bootloader a mělo odstraněnu ochranu proti zápisu do paměti. Jelikož ne vždy má cílové zařízení prázdnou paměť, tak je před samotným zápisem paměť vymazána. Pro jednoduchost jsem zvolil možnost smazání všech sektorů a tím celé paměti. Proces mazání paměti je časově náročný a jelikož programované zařízení odesílá potvrzení o přijetí příkazu okamžitě, je nutné čekat nějakou dobu než se mazání dokončí. Pokud by se začali data odesílat před dokončením, mělo by to za výsledek vrácení NACK a programátor by oznámil chybu nahrávání.

Po smazání paměti se začínají odesílat byty, které bootloader zpracovává a zapisuje do paměti na adresu, která je bootloaderu předávána společně s daty k zápisu. Bootloader vždy po přijetí sekvence bytů posílá zpět do zařízení odpověď, jestli se mu podařilo data zapsat. V případě, že by se mu to nepodařilo, celý proces končí s chybou. Jestliže se během procesu nahrávání nevyskytne žádná chyba, je obsluha nabídnuta možnost nahrání kódu spustit. Programátor se poté snaží vykonat `Go` příkaz, který se pokusí nahrání soubor spustit, a obsluha může zkontrolovat správnost nahrání. O výsledku spuštění je obsluha opět informována výpisem na display.

Při vývoji programátoru jsem se často setkával se situací, kdy se jednou příkaz podařilo provést a po restartu a zavolání stejného příkazu byla vrácena chyba. Tato situace se mi stávala například tehdy, když byl kód v programátoru pozastaven pomocí debuggeru a breakpointu, kdy programátor fungoval bez problému, ale po spuštění bez pozastavení kód nefungoval. Toto chování bylo způsobeno tím, že kód bootladeru je často náchylný dobu prodlevy mezi jednotlivými příkazy nebo jejich částmi. Každý příkaz má jistou časovou náročnost a i přesto, že jsou dostupné návratové hodnoty, není dobré se na ně spoléhat, jelikož je bootloader může vyslat dříve než je příkaz skutečně proveden případně je po vykonání příkazu mikroprocesor restartován a opětovná inicializace zabere jistou dávku času. Situace se komplikuje tím, že každé zařízení má časy potřebné k vykonání příkazu trochu jiné, proto jsem musel nastavit časové prodlevy tak, aby bylo možné programátor použít na co největší škále zařízení.

```
SystemInit();
SystemCoreClockUpdate();
if(SysTick_Config(SystemCoreClock / 1000)){
    while(1);
}
LCD_Init();
LCD_Clear();
LCD_Print("Programator 1.0");
RCC_Configuration();
GPIO_Configuration();
USART2_Configuration();
Led_Init();
Led_On();
EXTIline4_Config();
EXTIline5_Config();
Delay(1500);
if(Init_Bootloader() == FAILED){
    LCD_PrintClear("Init error");
    while(1);
}
LCD_PrintClear("Ziskavam seznam prikazu");
LCD_MoveDisplay(0, 7);
GetCommand(&availableCommands);
GetIdCommand(&PID);
LCD_PrintClear("Ziskavani dokonceno");
LCD_MoveDisplay(0, 3);
Delay(1000);

remove_protection();
while(1){
    if(redraw == 1){
        if(uroven == 3)
            Menu(availableCommands);
        else if(uroven == 4){
            Menu(PID);
            uroven = 0;
        }
        else
            Menu(1);
        redraw = 0;
    }
}
```

Obr. 15 - Ukázka kódu programátoru – inicializace zařízení

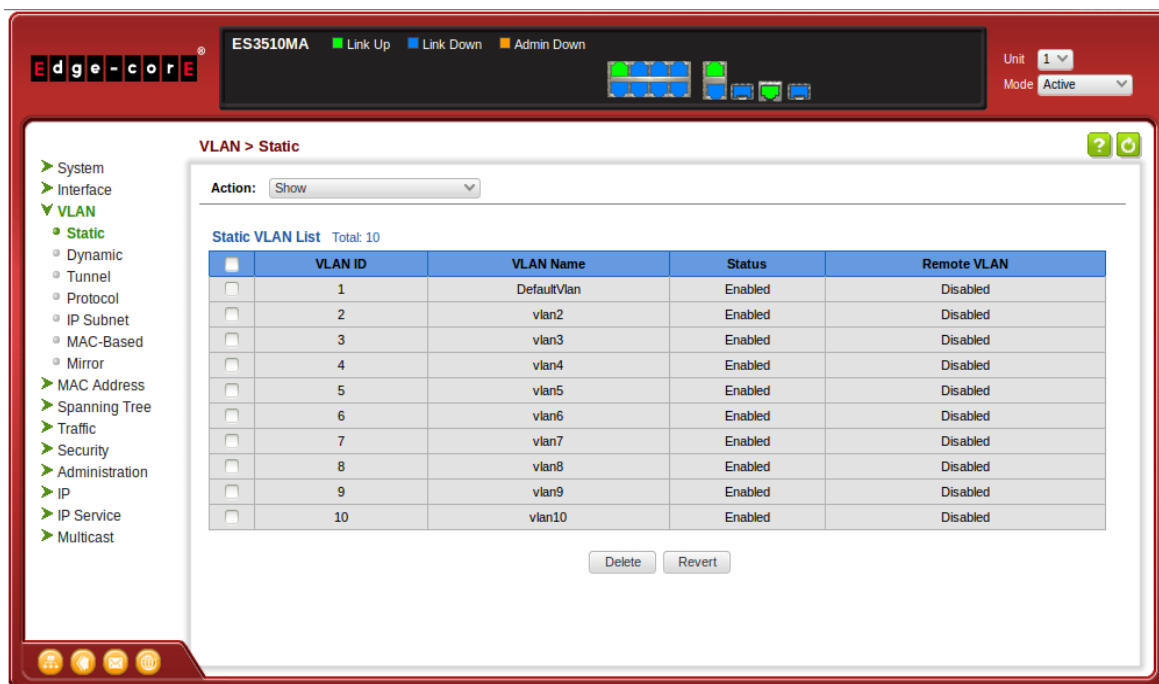
## 7.2 Implementace systému pro testování

Systém pro testování se dá rozdělit na část testování a část zobrazování výsledků, které jsou jednak zobrazovány průběžně v podobě grafů a také po skončení testování, kdy je lze procházet a vyhodnocovat.

### 7.2.1 Konfigurace switchů

Pro správnou funkci switchů při testování je bylo nutné nakonfigurovat, aby pakety správně rozdělily na jednotlivé porty. Pro testování jsem využil dva switche od společnosti Edge-core s označením ES3510MA. K nastavení switchů jsem využil webové rozhraní, které switche mají nahráno v sobě. Než jsem se ale byl schopen do tohoto rozhraní přihlásit, bylo nutné pomocí sériového portu a CLI rozhraní nastavit switchům IP adresy. Díky dokumentaci to nebyl složitý úkol, šlo pouze o spuštění příkazu *interface vlan 1* a ip address 192.168.1.5 255.255.255.0. Větším problémem bylo se se zařízením prvně spojit pomocí sériového portu, jelikož switch ze začátku neodpovídal.

Poté, co jsem se byl schopen přihlásit na webové rozhraní, jsem nakonfiguroval porty switchů tak, aby přijímaný broadcast na jednom portu přeposílal jednak dál na případný další switch přes port 10 a jednak na porty 1 až 7. Pro každý tento port jsem vytvořil vlastní virtuální LAN, aby se broadcast pakety nepřeposílaly ještě v rámci switchu. Jeden port je také vyhrazen pro řídicí přenos (port 11). Pomocí tohoto portu řídicí počítač komunikuje se switchi a dotazuje se jich na počty přijatých a odeslaných paketů. Pro správnou funkčnost bylo potřeba zakázat různé služby, které měl switch od výrobce povoleny, jelikož by tyto služby posílaly na porty pakety, které by se mohly přičítat ke sledovaným statistikám, což by testování většinou vyhodnotilo jako chybu, i když by vznikla uměle. Jednalo se například o službu spanning tree algorithm, která slouží k detekci smyček v síti.



Obr. 16 - Webové rozhraní switche s konfigurací VLAN

## 7.2.2 Databáze

Při testování je nutné ukládat informace o vzniklých chybách. K tomuto účelu byla vybrána databáze kvůli jednoduché správě dat. Jsou v ní vytvořeny pouze dvě tabulky. Jedna uchovává informace o spojení switchů přes testované zařízení, druhá je určena pro evidenci chyb, které se při testování vyskytnou.

## 7.2.3 Běh testovacího skriptu

Na obrázku Obr. 12 bylo naznačeno, jak funguje testovací smyčka. Zde bych rád blíže popsal skutečný průběh. Testování probíhá na operačním systému Linux, kdy je spuštěno z terminálu pomocí BASH skriptu. Po zavolání skriptu se nakonfiguruje virtuální síť (VLAN2), používaná k odesílání paketů na broadcast adresu. Po inicializaci se otevírá nové okno prohlížeče Mozilla Firefox, aby bylo možné aktuální výsledky sledovat v reálném čase. K odesílání paketů je využit příkaz ping. Příkaz odesílá po určenou dobu pakety o dané velikosti a po dokončení vypíše souhrnné statistiky. Jelikož v sobě nesou informaci o celkovém počtu odeslaných paketů za dobu běhu, jsou poté použity jako vstup do PHP skriptu. PHP skript se poté zeptá pomocí SNMP switchů, kolik paketů přijaly a odeslaly na jednotlivých portech. Tyto hodnoty jsou poté porovnány mezi sebou a se skutečným počtem odeslaných paketů. Pokud počty nejsou shodné, je tento stav vyhodnocen jako chyba a zapsán do databáze. Tímto způsobem jsou vyhodnoceny všechny porty.

Jakmile je dokončeno vyhodnocování posledního portu, spočítá se celkový počet ztracených paketů a výsledek se zapíše do RRD databáze. Proces se poté opakuje. Počet opakování je dán délkou trvání testu. Doba jednoho cyklu trvá 8 sekund, takže za 24 hodin je sesbíráno velké množství dat v různých teplotách. Po skončení testování se opět otevře nový panel internetového prohlížeče a obsluha si může prohlédnout výsledky testování v grafech vykreslených pomocí nástroje RRDtool, nebo v tabulce pod nimi.

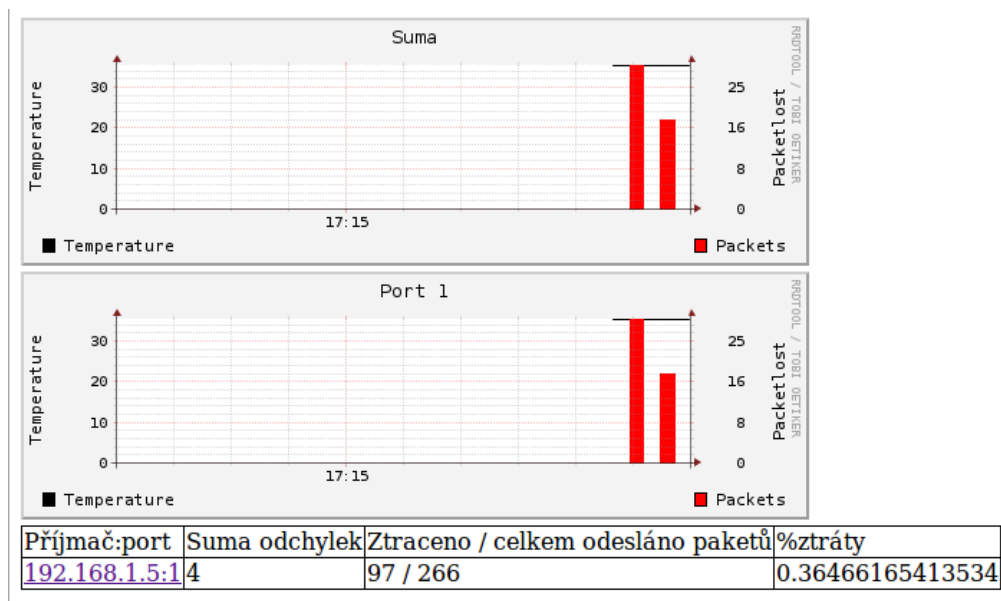
#### 7.2.4 Použité identifikátory SNMP

Jak již bylo zmíněno, je třeba v průběhu testování znát aktuální hodnoty přijatých a odeslaných paketů na switchích. K tomuto zjišťování stavu je využito SNMP, které využívá pro získávání informací OID identifikátory. Na stránkách [tools.cisco.com](http://tools.cisco.com) v nástroji SNMP Object Navigatoru jsem vyhledal, jaké identifikátory OID by mohly být použity pro získání těchto informací. Jelikož jsem věděl, že se při testování budou posílat broadcast pakety, tak jsem zvolil identifikátory *ifInBroadcastPkts* (OID: 1.3.6.1.2.1.31.1.1.1.3) a *ifOutBroadcastPkts* (OID: 1.3.6.1.2.1.31.1.1.1.5), které informace o přijatých a odeslaných broadcast paketech poskytují.

Druhým problémem bylo zjistit, jestli switche mají tyto identifikátory implementovány. Pro zjištění jsem použil nástroj *snmpwalk*, který je často součástí Linuxových distribucí. Výstupem tohoto programu je seznam dostupných OID identifikátorů. V tomto seznamu jsem již jenom vyhledal mnou potřebné identifikátory. Vzhledem k tomu, že všechny ostatní funkce switche, které by mohly broadcast odesílat jsem vypnul, považuji toto řešení za dostatečně správné.

#### 7.2.5 Prohlížení souhrnných výsledků

Prohlížení výsledků je možné v prohlížeči webových stránek. Na základní obrazovce jsou vykresleny výsledky za posledních pět minut testování pro každý testovaný port. Pod grafy je vykreslena tabulka se souhrnnými statistikami a s odkazy na detailní statistiky jednotlivých portů.



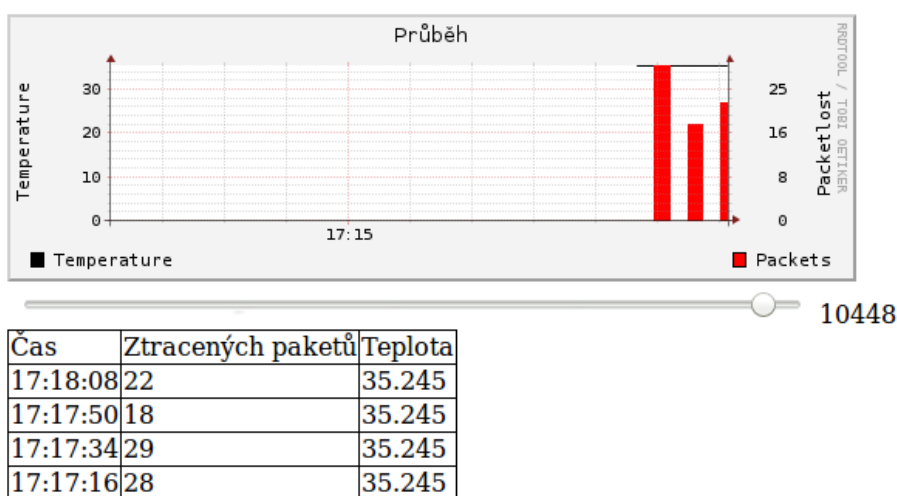
Obr. 17 - Výsledky testování pro jeden zapojený port - přehled

### 7.2.6 Detailní prohlížení výsledků

System umožňuje také detailní náhled výsledků v čase. K těmto výsledkům se dá snadno dostat kliknutím na odkaz v prvním sloupci tabulky. Obsluze se poté zobrazí podobný obrázek jako je uveden níže.

Jelikož jsou všechny data uloženy v databázi RRD, je možné posuvníkem pod grafem vyhledat žádaný okamžik testování. Stejná data jsou ale i v tabulce pod grafem, jelikož odečet z grafu by nebyl jednak přesný ani pohodlný.

V tabulce je uvedena informace o tom, jaká byla v době odchyčení chyby v komoře teplota a kolik paketů se ztratilo.



Obr. 18 - Výsledky testování - detail

## ZÁVĚR

Výstupem této bakalářské práce měl být systém pro programování mikroprocesorů. Byl tedy vyroben plošný spoj, který pomocí interního bootloaderu programovaných zařízení a komunikace po RS232, dokáže přenést binární soubor uložený v interní paměti do druhého zařízení. Specifické je, že přenášený soubor je možné ještě před přenosem upravit, například kvůli změně MAC adresy cílového zařízení nebo sériového čísla. Ovládání je intuitivní díky LCD displayi, takže obsluha vždy ví, co se aktuálně vykonává za činnost.

Další částí mé práce bylo vytvoření systému pro testování zařízení s Ethernetovým rozhraním. V případě tohoto bodu jsem vytvořil jednoduchý skript, který po zavolání provede celé testování sám a obsluha poté pouze zhodnotí výsledky. Tento způsob není příliš elegantní, ale vzhledem k neurčitosti zadání (nebyly zadány žádné limity ztrátovosti) se tato cesta jevila jako vhodná. Při dalším rozvoji bych ale chtěl výsledky lépe vyhodnocovat a interpretovat.

Na systému pro programování mám v plánu dále pracovat, jelikož se to ukázalo jako velmi zajímavé zadání a s panem doktorem Dulíkem máme ještě nápady, jak tento systém dále rozšířit jako je například odstranění potřeby mít binární soubor nahraný přímo v interní paměti. Zajisté bude potřeba zdrojové kódy pročistit a optimalizovat, ale myslím, že pro budoucí rozvoj nejsou ve špatném stavu a úpravy tím pádem nebudou tak složité.

## ZÁVĚR V ANGLIČTINĚ

This thesis describes two systems for programming and testing microprocessor boards. The programming system was made as a printed circuit board which uses an internal bootloader of programmed devices and the RS232 interface for communication to transfer binary files located in internal memory of programmer to other device. The programmer allows editing the binary file before transfer, for example to change MAC address or serial number. Interaction with the device is intuitive thanks to the LCD display.

Second part of this thesis describes the system for testing devices with Ethernet interface. To solve this problem I wrote a simple script which does the whole testing by itself and user just evaluates the results. This solution is not very elegant but because of vague specification of requirements (e.g., no limits for packets loss were given), this solution is good enough. I would like to evaluate and interpret the results to better in future development.

In future, I would like to do more work also on the system for programming microprocessors, because it is an interesting topic and I have some ideas to improve it such as remove requirement for having binary file stored directly in internal memory. Of course, it will need to clean and optimize the code, but the changes will not be so hard.

## SEZNAM POUŽITÉ LITERATURY

- [1] Sloss, A., D.Symes a Ch.Wright ARM System Developer's Guide. . San Francisco: Elsevier, 2004. ISBN: 1-55860-874-5
- [2] Joint Test Action Group. In Wikipedia : the free encyclopedia [online]. 2001-2014 [cit. 2014-05-23]. Dostupné z WWW: <[http://cs.wikipedia.org/wiki/Joint\\_Test\\_Action\\_Group](http://cs.wikipedia.org/wiki/Joint_Test_Action_Group)>
- [3] Yiu, J. The Definitive Guide to the ARM Cortex-M3. . Burlington: Elsevier, 2007. ISBN: 978-0-7506-8534-4
- [4] Ethernet. In Wikipedia : the free encyclopedia [online]. 2001-2014 [cit. 2014-05-23]. Dostupné z WWW: <<http://cs.wikipedia.org/wiki/Ethernet>>
- [5] STM32 microcontroller system memory boot mode. STMicroelectronics [online]. 2014 [cit. 2014-05-25]. Dostupné z WWW: <[http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application\\_note/CD00167594.pdf](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note/CD00167594.pdf)>
- [6] Tobias Oetiker About RRDtool [online]. 2013 [cit. 2014-05-27]. Dostupné z WWW: <<http://oss.oetiker.ch/rrdtool/>>
- [7] RRDtool. In Wikipedia : the free encyclopedia [online]. 2001-2014 [cit. 2014-05-24]. Dostupné z WWW: <<http://en.wikipedia.org/wiki/RRDtool>>
- [8] Simple Network Management Protocol. In Wikipedia : the free encyclopedia [online]. 2001-2014 [cit. 2014-05-27]. Dostupné z WWW: <[http://cs.wikipedia.org/wiki/Simple\\_Network\\_Management\\_Protocol](http://cs.wikipedia.org/wiki/Simple_Network_Management_Protocol)>
- [9] About MySQL Oracle [online]. 2014 [cit. 2014-05-27]. Dostupné z WWW: <<http://www.mysql.com/about/>>
- [10] Leiss, O., Schmidt, J. PHP v praxi. . Havlíčkův Brod: Grada publishing, 2010. ISBN: 978-80-247-3060-8
- [11] VLAN. In Wikipedia : the free encyclopedia [online]. 2001-2014 [cit. 2014-05-28]. Dostupné z WWW: <<http://cs.wikipedia.org/wiki/VLAN>>
- [12] PhpStorm Features — PHP Code Editor JetBrains [online]. 2014 [cit. 2014-05-28]. Dostupné z WWW: <<http://www.jetbrains.com/phpstorm/features/>>

**SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK**

|      |                                    |
|------|------------------------------------|
| ACK  | Acknowledge                        |
| NACK | Not Acknowledge                    |
| PHP  | PHP: Hypertextový Preprocesor      |
| SNMP | Simple Network Management Protocol |
| BASH | Bourne Again Shell                 |
| RRD  | Round-robin Database               |
| MAC  | Media Access Control               |
| IAP  | In-application Programing          |
| JTAG | Joint Test Action Group            |
| CLI  | Command-line Interface             |
| VLAN | Virtuální LAN                      |
| ROM  | Read-only Memory                   |
| RAM  | Random Access Memory               |
| CRC  | Cyclic redundancy check            |

**SEZNAM OBRÁZKŮ**

|   |    |
|---|----|
| Obr. 1 - Software spouštěný embedded systémy [1].....               | 11 |
| Obr. 2 - Struktura paměti Cortex-M3 [3].....                        | 13 |
| Obr. 3 - Zjednodušené plošného spoje.....                           | 16 |
| Obr. 4 - Hierarchie menu.....                                       | 17 |
| Obr. 5 - Vyrobený plošný spoj programátoru.....                     | 19 |
| Obr. 6 - Nahrávací sekvence.....                                    | 22 |
| Obr. 7 - Paměť mikroprocesoru v aplikaci ST-Link Utility.....       | 23 |
| Obr. 8 - Princip uložení dat v paměti.....                          | 24 |
| Obr. 9 - Kód funkce na získání dat z interní paměti.....            | 24 |
| Obr. 10 - Blokové schéma zapojení testovacího systému.....          | 25 |
| Obr. 11 - Reálné zapojení testovacího systému.....                  | 26 |
| Obr. 12 - Zobecněné schéma testovací smyčky.....                    | 27 |
| Obr. 13 - Prostředí Tasking.....                                    | 30 |
| Obr. 14 - PhpStorm [12].....  | 31 |
| Obr. 15 - Ukázka kódu programátoru – inicializace zařízení.....     | 33 |
| Obr. 16 - Webové rozhraní switche s konfigurací VLAN.....           | 35 |
| Obr. 17 - Výsledky testování pro jeden zapojený port - přehled..... | 37 |
| Obr. 18 - Výsledky testování - detail.....                          | 37 |

## SEZNAM TABULEK

|  |    |
|--|----|
| Tab. 1 - Seznam příkazů podporovaných interním bootloaderem [5]..... | 20 |
|--|----|

## SEZNAM PŘÍLOH

Příloha P 1: Použitý software

Příloha P 2: Zdrojové kódy systému pro programování

Příloha P 3: Zdrojové kódy systému pro testování

## **PŘÍLOHA P 1: POUŽITÝ SOFTWARE**

|                    |                   |
|--------------------|-------------------|
| Tasking            | Vývoj aplikace    |
| LibreOffice Writer | Sazba práce       |
| LibreOffice Draw   | Tvorba diagramů   |
| PHPStorm           | Vývoj aplikace    |
| dia-gnome          | Tvorba diagramů   |
| PhpMyAdmin         | Správa databáze   |
| MySQL              | Databázový systém |
| snmpwalk           | Vývoj aplikace    |