ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA APLIKOVANÉ ELEKTRONIKY A TELEKOMUNIKACÍ

DIPLOMOVÁ PRÁCE

Dálkově ovládaná laboratoř na platformě STR912FAW44

Jaroslav Malán **2015**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2014/2015**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jaroslav MALÁN**
Osobní číslo:	**E13N0106P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a aplikovaná informatika**
Název tématu:	**Dálkově ovládaná laboratoř na platformě STR912FAW44**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

1. Seznamte se s vlastnostmi mikrokontroléru STR912FAW44 s jádrem ARM9, jeho dostupnými vývojovými prostředky a strukturou hardwarového modulu komunikační gatewaye JH10.
2. Seznamte se vlastnostmi standardu Ethernet IEEE 802.3, protokolové sady TCP/IP (ARP, IPv4, ICMP, UDP, TCP) a protokolu HTTP.
3. Pro gateway JH10 navrhněte a realizujte rozšiřující submodul obsahující interfaceové obvody pro maximum unifikovaných vstupních a výstupních logických a analogových signálů.
4. Na hardwarovém modulu - komunikační gatewaye JH10 s mikrokontrolérem STR912FAW44 - implementujte aplikaci WWW serveru realizující obecně konfigurovatelné ovládací algoritmy operující nad subsystémem výše zmíněných vstupních a výstupních logických a analogových signálů. Vyspecifikujte a popište metodiku uživatelské konfigurace obecné úlohy.
5. Návrh ověřte konkrétní implementací 3 netriviálních úloh z oblasti číslicových systémů a 2 úloh z oblasti analogových elektronických systémů.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah pracovní zprávy: **30 - 40 stran**

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

Seznam odborné literatury:

1. **STMicroelectronics www.st.com/mcu - STR91xFA ARM9 Based Microcontroller Family, UM0216 Reference Manual, STR91xFA Data Sheet**
2. **IEEE 802.3 Standard - Part 3 - Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications**
3. **Dostálek, L., Kabelová, A.: Velký průvodce protokoly TCP/IP a systémem DNS. Computer Press, 2000. ISBN 80-7226-323-4**
4. **www.ietf.org/rfc.html - RFC768 User Datagram Protocol, RFC791 Internet Protocol, RFC792 Internet Control Message Protocol, RFC826 Address Resolution Protocol, RFC 962 Transport Control Protocol**

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

Datum zadání diplomové práce: **15. října 2014**

Termín odevzdání diplomové práce: **11. května 2015**

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

L.S.

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 15. října 2014

Abstrakt

Předkládaná diplomová práce je zaměřena na vývoj softwaru webového serveru na platformě STR912FAW44 a hardwarový návrh rozšiřujícího submodulu pro desku komunikačního modulu JH10 s mikrokontrolérem typu ARM a ethernetovým rozhraním. Cílem práce je realizace systému vzdáleně ovládané laboratoře pomocí dynamického webového serveru.

Pro aplikaci webového serveru bylo použito TCP/IP stacku LwIP. Nad tímto stackem byla implementována aplikace HTTP webového serveru rozšířená o dynamické generování obsahu. Aplikace HTTP webového serveru slouží jako uživatelské rozhraní pro obsluhu navrženého submodulu. Navržený a vyrobený submodul plní funkci standardních analogových a digitálních vstupů i výstupů.

Dílčím výsledkem práce je plně funkční submodul obsahující základní digitální a analogové vstupy i výstupy, které jsou připojeny pomocí externí datové sběrnice a integrovaného AD převodníku ke komunikačnímu modulu s mikrokontrolérem STR912. Vstupní a výstupní obvody submodulu jsou řízeny pomocí plně funkčního dynamického HTTP webového serveru. Webový server obsahuje zabezpečení přístupu protokolem HTTP a šifrováním Basic64.

Práce poskytuje podrobný návod implementace LwIP stacku včetně webového serveru na libovolnou platformu. Pro příliš velký rozsah kódu, který není podporován základními verzemi komerčních vývojových nástrojů, je v práci popsána také instalace vývojového prostředí Eclipse a jeho součástí pro bezplatný vývoj aplikací na platformě ARM.

Klíčová slova

ARM, STR9, STR912, STR91X, TCP/IP, LwIp, stack, webový server, dynamický server, HTTP, CGI, SSI, Basic64

Abstract

The final thesis is focused on webserver software development for the STR912FAW44 platform and expansion submodule hardware design for communication module JH10 with ARM microcontroller containing ethernet interface. Purpose of this thesis is implementation of remotely controlled laboratory system through a dynamic webserver.

TCP/IP stack called LwIP was used for the webserver application. The application of dynamic content HTTP webserver was founded on basis of the LwIP stack. The application of HTTP webserver is ment to be used as a user interface for control of the designed submodule. The submodule which was practicaly build contains standard analog and digital outputs and inputs.

This thesis resulted in fully functional sumbodule connected through the external data bus to the comunication module containing STR912 microcontroller. Input and output circuits of submodule are controlled by fully functional dynamic HTTP webserver. The webserver cointains access security mechanism developed using the HTTP protocol and the Basic64 encryption.

This thesis provides detailed manual of LwIP stack implementation with the webserver for any platform. The final thesis also describes the installation of Eclipse development environment with all necessary external software for an ARM platform development. The design of the digital and analog input and output circuits connected through the external data bus is also described in the text.

Key words

ARM, STR9, STR912, STR91X, TCP/IP, LwIp, stack, webserver, dynamic server, HTTP, CGI, SSI, Basic64

Prohlášení

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

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

.....

podpis

V Plzni dne 7.5.2015 Jaroslav Malán

Poděkování

Tímto bych rád poděkoval vedoucímu diplomové práce Ing. Petrovi Kristovi, Ph.D. za cenné profesionální rady, připomínky a metodické vedení diplomové práce. Dále bych rád touto formou poděkoval celé mé rodině, která mi byla v průběhu celého mého studia nepostradatelnou oporou.

Obsah

Seznam symbolů a zkratek

ARM ..... Advanced RISC Machines

HTTP ..... Hypertext Transfer Protocol

RISC..... Reduced Instruction Set Computing

GPIO ..... General Purpose Input/Output

EMI ..... External Memory Interface

AHB ..... Advanced High-performance Bus

CS..... Chip Select

OE ..... Output Enable

CGI..... Common Gateway Interface

SSI..... Server Side Includes

DAC ..... Digital-to-Analog Converter

ADC ..... Analog-to-Digital Converter

TTL ..... Transistor-Transistor Logic

IDE..... Integrated Development Environment

PHP ..... PHP: Hypertext Preprocessor

JRE..... Java Runtime Environment

JDK ..... Java Development Kit

GCC ..... GNU Compiler Collection

CDT ..... C/C++ Development Tooling

ROM ..... Read Only Memory

RAM ..... Random Access Memory

API..... Application Programming Interface

TCP/IP ..... Transmission Control Protocol/Internet Protocol

MAC ..... Media Access Control

ARP..... Address Resolution Protocol

DMA ..... Direct Memory Access

RTC..... Real Time Clock

HTML ..... Hypertext Markup Language

PWM..... Pulse Width Modulation

CSV..... Comma Separated Values

Úvod

Předkládaná diplomová práce se zabývá realizací vzdáleně ovládané laboratoře za využití ethernetového rozhraní, které je součástí desky s ARM mikrokontrolérem STR912. Vyvíjená laboratoř je schopna poskytnout informace o aktuálním stavu všech analogových a digitálních vstupů i výstupů. Výsledný systém je schopen poskytovat data skrze webové rozhraní a také řídit všechny analogové i digitální výstupy pro aplikace v rozličných úlohách. Webové rozhraní generuje dynamicky proměnný obsah a poskytuje základní zabezpečení přístupu pomocí přihlašovacích údajů šifrovaných algoritmem Base64, které jsou předány pomocí HTTP hlavičky.

V první části práce je popsán princip návrhu harwarového submodulu, který je navržen je pro rozšíření funkcí zapůjčeného komunikačního gatewaye J10 s mikrokontrolérem SRT912. Ke komunikačnímu modulu je navržený submodul připojen za použití konektorů, které tvoří rastr o vzdálenosti pinů 2,54 mm. Submodul obsahuje šestnáct digitálních vstupů, šestnáct digitálních výstupů, osm analogových vstupů a dva analogové výstupy. Většina obvodů navrhovaného submodulu je připojena k mikrokontroléru STR912 pomocí externího paměťového rozhraní EMI bez přidání další nežádoucí softwarové emulace.

V druhé části této práce je popsán vývoj softwaru pro navržený hardware. Především je podrobně rozebrána obecná implementace LwIP stacku nejen pro použitý komunikační gateway. Následně je v práci podrobně popsána implementace HTTP serveru. Jsou popsány úkony pro vytvoření HTTP severu schopného generovat dynamický webový obsah a přijímat příkazy z webového rozhraní klienta za použití standardních komunikačních protokolů CGI a SSI. Na závěr je také popsáno, jakým způsobem byly rozšířeny zdrojové kódy webového serveru o zabezpečení přístupu pomocí HTTP protokolu s šifrováním Base64. Popis implementace LwIP stacku je velmi podrobný a je napsán tak, aby i méně zkušenému čtenáři usnadnil implementaci široce uplatnitelného LwIP stacku. LwIP stack je vhodný pro všechny moderní mikrokontroléry a po přečtení práce by měl být čtenář obeznámen se všemi potřebnými kroky pro implementaci na libovolný mikrokontrolér. Součástí práce je návrh hardwarového submodulu zpracovaný v prostředí Eagle a kompletní zdrojové kódy vzdáleně ovládané laboratoře, které mohou sloužit jako studijní materiál pro doplnění informací obsažených v textu.

1 Hardwarový modul komunikačního gatewaye JH10

1.1 STR912FW44

Mikrokontrolér pod označením STR912FW44 je založen na architektuře ARM966E-S. Jedná o 32-bitový RISC jednočipový počítač, který byl navržen především pro real-time aplikace. Miktokontrolér STR912FW44 je harvardské architektury s oddělenou datovou a programovou pamětí a 5-ti úrovňovou pipeline. Maximální pracovní frekvencí mikrokontroléru je 96MHz. Mikrokontrolér neobsahuje cache paměť, ale obsahuje pouze tightly coupled memories, které na rozdíl od cache paměti poskytují deterministickou přístupovou dobu. Takovéto řešení je vhodné pro pro mikrokontroléry nasazené v real-time aplikacích.

Ačkoliv jsou v současné době na trhu dostupné modernější a výkonnější alternativy například v podobě ARM rodiny Cortex, STR912FW44 stále poskytuje dostatek výkonu pro aplikace obsahující implementaci TCP/IP stacku s nástavbou HTTP webového serveru, který slouží jako komunikační rozhranní pro obsluhu celého embedded systému.

1.2 Gateway JH10

Diplomová práce je postavena na hardwaru komunikačního gatewaye JH10. Veškerá dokumentace ke komunikačnímu modulu je dostupná na datovém nosiči této práce ve složce s názvem "Dokumentace Gateway JH10". Tento komunikační modul byl vyvinut na základě diplomové práce s názvem Hardware pro komunikační gateway s STR912FW44. Autorem této práce je Jakub Hájek, který práci úspěšně dokončil a na základě jeho návrhu byly vyrobeny komunikační moduly se záměrem použití ve výuce. Tento záměr však nebyl realizován pro rozšíření modernějších a finančně dostupných vývojových desek s výkonnějším mikrokontrolérem typu ARM Cortex. Z tohoto důvodu nenalezlo mnoho z vyrobených kusů komunikačního modulu praktické uplatnění, ačkoliv vykazují znaky velmi dobře navrženého a fungujícího hardwaru.

2 Návrh hardwaru submodulu pro gateway JH10

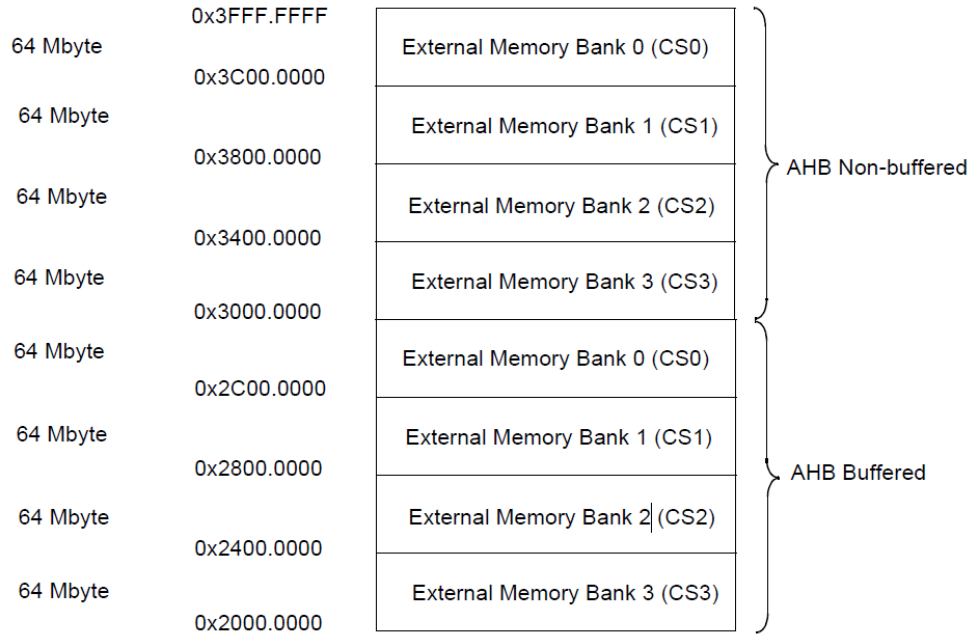
Prvním cílem této práce bylo navrhnout rozšiřující submodul pro poskytnutý gateway JH10. Bylo požadováno, aby submodul obsahoval interfacové obvody pro maximum

unifikovaných vstupních a výstupních signálů. Po konzultaci s vedoucím práce bylo rozhodnuto, že submodul bude obsahovat celkem šestnáct digitálních výstupů, šestnáct digitálních vstupů, osm analogových vstupů a dva analogové výstupy.

Rozhraní pro submodul, které bylo dostupné na dodaném modulu JH10, však obsahuje nedostatečný počet vývodů pro zapojení všech interfacových obvodů ve standardním GPIO módu. Ani zapojení externích DA převodníků pomocí GPIO není z hlediska dynamiky systému ideálním řešením. Proto bylo nutné využít sběrnice EMI, která je vyvedena na výstupní piny gatewaye. Tato sběrnice i přes velmi omezený počet fyzických signálů na rozhraní gateway - submodul umožnila připojit všechny potřebné interfacové obvody.

2.1 External memory interface

External memory interface je konkrétní blok rozhraní použitý v mikrokontroléru STR912. Jedná se o rozhranní určené primárně pro připojení externích paměťových zařízení k AHB sběrnici mikrokontroléru. Tento typ rozhraní se vyskytuje i v mnoha jiných pokročilých mikrokontrolérech v rozličných modifikacích pod různými názvy. Rozhraní podporuje 4 externí paměťové banky připojené ke sběrnici AHB s využitím bufferu, nebo bez bufferu viz obrázek č. 1. Pokud je sběrnice EMI konfigurována pro banky s přístupem ke sběrnici za využití bufferu, nedochází k pozastavení jádra, které by v módu bez bufferu muselo čekat na vyřízení požadavku na sběrnici EMI. Avšak k pozastavení jádra by došlo i v módu s využitím bufferu, pokud by byl buffer EMI zcela zaplněn. Pro realizaci aplikace vzdáleně ovládané laboratoře, kdy jsou pomocí externí paměťové sběrnice přenášena data pro výstupy submodulu a čtena data ze vstupů submodulu, jsem použil výhradně přístup na sběrnici bez použití bufferu. Tím jsem dosáhl bezodkladného zápisu i čtení dat ze sběrnice, čímž je zajištěna větší kontrola především nad správným časováním výstupů. Okamžitý zápis dat na sběrnici vyvolaný miktrokontrolérem je nutný například při realizaci pulsní šířkové modulace pomocí digitálních výstupů submodulu [1].



Obr. č. 1: Paměťové banky EMI (převzato z [2])

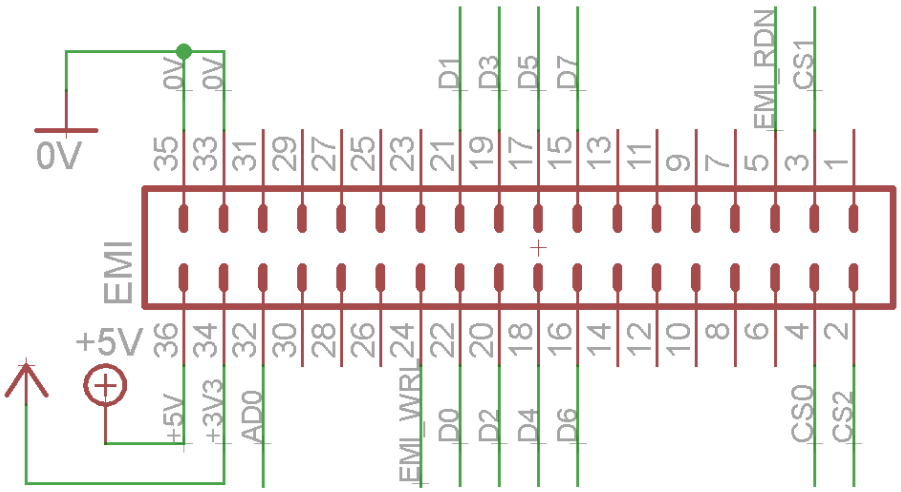
EMI sběrnice poskytuje více možností konfigurace. První z nich je možnost provozování sběrnice v multiplexovaném módu. V tomto módu je možné dosáhnout až 24-bitové adresy a přenosu 16-bitových dat. To je realizováno pomocí až 24 adresních vodičů, z nichž je až 16 vodičů multiplexováno pro účel přenosu dat pomocí jednoho řídícího signálu. V multiplexovaném režimu je možno volit z 8-bitových dat a 16-bitové adresy, 16-bitových dat a 20-bitové adresy, nebo 16-bitových dat a 24-bitové adresy [1].

Druhou možností konfigurace EMI sběrnice je funkce v nemultiplexovaném režimu. V tomto režimu je možné dosáhnout maximálně 8-bitových dat a 16-bitové adresy. Žádný z datových a adresních vodičů nemění svoji funkci v závislosti na čase. Tento mód, při kterém je sběrnice nastavena v nemultiplexovaném módu s 8-bitovou velikostí datové sběrnice byl zvolen pro komunikaci mezi dodaným gatewayem JH10 a navrhovaným submodulem. Důvodem pro tuto volbu byla skutečnost, že nemultiplexovaný mód je snadněji konfigurovatelný a méně náročný na obvodové řešení submodulu. Díky nemultiplexovanému módu není nutné realizovat latch obvody pro multiplex datových a adresních signálů v jednom zápisovém nebo čtecím cyklu. Ačkoliv je datová sběrnice v tomto režimu pouze 8 bitová, lze v programu v jazyce C požadovat zápis jak 16-bitového tak 32 bitového záznamu do paměti. To samé platí i pro čtení. Řadič EMI v 8-bitovém nemultiplexovaném režimu rozdělí 16-bitová data na dvě 8-bitové části. Případně 32-bitová data rozdělí na 4 části a zápis provede na úrovni HW zcela bez nutnosti explicitního vyjádření požadavku v kódu.

Samozřejmě je nutné podotknout, že čas potřebný pro zápis nebo čtení dat o větší délce než je šířka datové sběrnice odpovídá násobkům osmi dle bitové délky paralelně zapisovaných dat [1].

2.1.1 Použité signály EMI

Komunikační gateway JH10 obsahuje EMI konektor, který zahrnuje všechny důležité signály pro rozličné nastavení EMI sběrnice. Pro účely nemultiplexovaného módu s 8-bitovými daty a 16-bitovou adresou byl však využit jen potřebný počet vodičů a ostatní zůstaly nevyvedeny do obvodů submodulu. Na obrázku č. 2 je zobrazen EMI konektor s vyznačenými signály, které jsou vedeny do submodulu.



Obr. č. 2: Použité vodiče na patici pinů gatewaye JH10

Využité EMI signály:

- D0 - D7 - signály datové části sběrnice
- AD0 - nejméně významný bit (LSB) adresní části sběrnice
- EMI_WRL - signál pro zápis na sběrnici
- EMI_RDN - signál pro čtení ze sběrnice
- CS0 - CS2 - výběrové signály (chip select)

Dle výše vypsaného seznamu využitých signálů je vidět, že aplikace využívá třech výběrových signálů (chip select) z externí datové sběrnice. Každý z výběrový signál odpovídá jedné z celkem 4 dostupných paměťových bank externí datové sběrnice. Celkem byly obvody připojené na sběrnici rozděleny do tří paměťových bank dle své povahy. Vzhledem ke třem

použitým paměťovým bankám bylo nutné použít tři vodiče pro výběrové signály. CS0 je využit pro digitální vstupy, CS1 pro digitální výstupy a CS2 pro analogové výstupy, které jsou generovány za použití dvou D/A převodníků. Vzhledem k tomu, že datová šířka externí datové sběrnice byla zvolena 8 bitů, avšak například digitální výstup obsahuje celkem 16 bitů dat reprezentovaných dvojicí 8-bitových registrů, bylo nutné navrhnout dekódovací logiku. Dekódovací logika má za účel především pomocí kombinace adresy a základního výběrového signálu z EMI sběrnice generovat vlastní výběrový signál, který jednoznačně určí, do kterého výstupního registru mají být data zapsána(v případě digitálních výstupů). Dle požadavků integrovaných obvodů, tvořících vstupy a výstupy submodulu, jsou v některých případech součástí dekódovací logiky také zápisové a čtecí signály EMI. Vzhledem k tomu, že na každou externí paměťovou banku EMI sběrnice jsou připojeny přesně dva čipy, je zcela dostatečné použití pouze jednoho bitu adresy EMI. Ostatní adresní vodiče nejsou do submodulu vyvedeny. Proto se dá očekávat, že všechny požadavky s lichými adresami v jedné datové bance budou mít v případě výstupních registrů za následek zápis do jednoho registru. Oproti tomu všechny požadavky se sudými adresami budou mít za výsledek zápis do druhého registru. Tato situace je stejná i pro dvojici vstupních budičů, které jsou použity jako digitální vstupy.

2.1.2 Dekódovací logika EMI

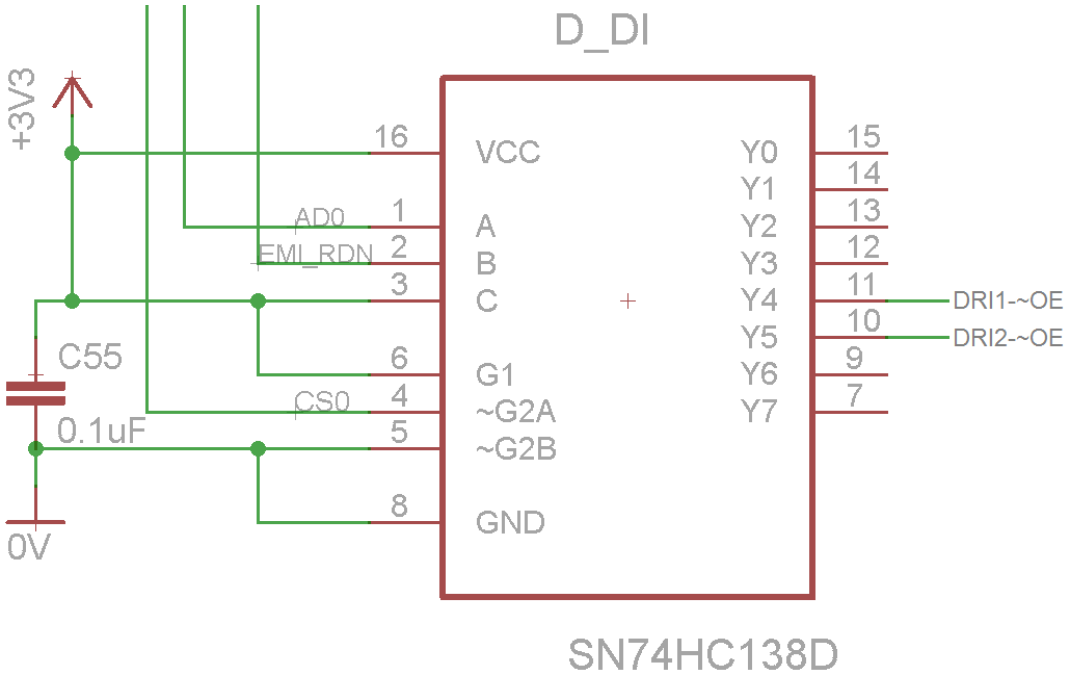
Pro realizaci dekódovací logiky byl zvolen integrovaný obvod SN74HC138. Ten je určen pro vysokorychlostní aplikace především jako paměťový dekodér, čímž přesně splňuje požadavky na aplikaci v navrhovaném systému. Napájecí napětí je možno volit v rozmezí 2V – 6 V.

Cílem pro sestavení dekódovací logiky bylo pomocí jedinečné kombinace výběrového signálu EMI, jednoho bitu datové sběrnice a zápisového/čtecího signálu generovat potřebné signály pro výběr a aktivaci pouze jednoho integrovaného obvodu z celkových 6 integrovaných obvodů - dvou výstupních registrů, dvou vstupních budičů a dvou D/A převodníků. Každý typ integrovaného obvodu připojený na sběrnici EMI má však specifické požadavky na řídicí signály. Tuto situaci bylo nutné při návrhu dekódovací logiky zohlednit a současně použít co nejmenší počet dekodérů v podobě integrovaného obvodu SN74HC138. V tomto případě jeden dekodér nebyl dostatečný pro všechny řídicí signály především pro omezený počet vstupních signálů, avšak dva dekodéry poskytly dostatek vstupních a výstupních signálů.

První dekodér byl použit pouze pro obvody plnící funkci digitálních vstupů. Chip select 0 byl přiveden do ~G2A - negovaného OE pinu dekodéru. Adresní bit byl přiveden na vstup A a čtecí signál EMI_RDN byl přiveden na logický vstup B. Logický vstup C, G1 jsou trvale připojeny na 3.3V tedy H úroveň signálu. A poslední OE pin ~G2B je trvale uzemněn, tedy stav L. Výstupní signál dekodéru Y4 je přiveden na negovaný OE vstup vstupního budiče DRI1 a při vhodné kombinaci CS0, AD0 a čtecího pulsu EMI_RDN dojde k aktivaci výstupů budiče, čímž budič přenese své vstupy na sběrnici a je umožněno jejich čtení a zpracování pomocí řadiče EMI. Principielně stejná logika přivede na sběrnici data z druhého budiče, pouze však s rozdílným stavem AD0 pinu na vstupu dekodéru. Prakticky byly v programu použity adresy 0x3C000000 pro první budič a 0x3FFFFFFE pro druhý budič viz obr. č. 1. Adresy mohou být však odlišné, jen je nutné zajistit rozdílnou úroveň signálu AD0 a musí být zohledněno, jaká délka dat je čtena.

Tab.č. 1: Funkce převodu vstupních signálů na výstupní v prvním dekodéru

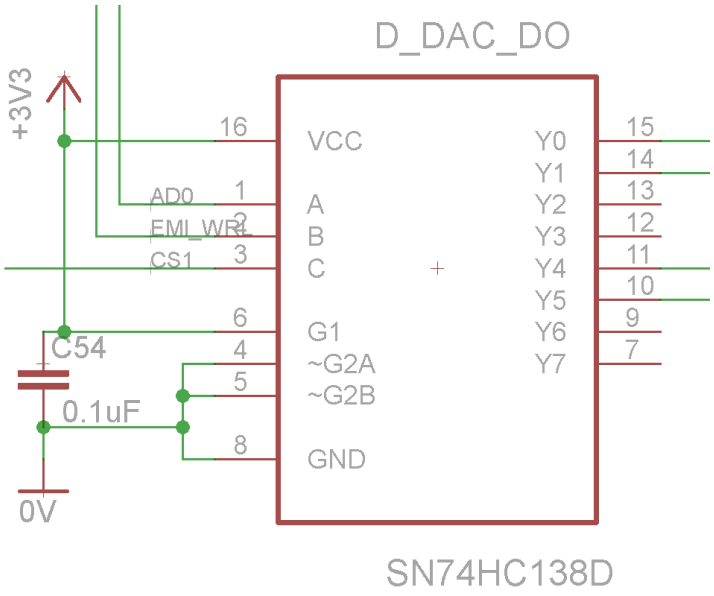
~G2A(CS0)	A(AD0)	B(EMI_RDN)	Y4	Y5
H	X	X	H	H
L	L	L	L	H
L	H	L	H	L
X	X	H	H	H



Obr.č. 3: Zapojení dekodéru SN74HC138D pro digitální vstupy

Druhý dekodér byl použit pro dva typy periferií připojených na sběrnici. Prvními z nich jsou digitální výstupy reprezentované dvojicí 8-bitových registrů a druhými jsou dva DA převodníky v 8-bitovém režimu. Všechny output enable piny dekodéru(G1, ~G2A, ~G2B) jsou staticky připojeny dle své povahy na odpovídající úroveň napětí pro nepřetržité sepnutí převodu vstupů dekodéru na jeho výstupy. Díky tomu jsou výstupy pouze kombinací tří vstupních signálů AD0, EMI_WRL a CS1 připojených na vstupy dekodéru A, B, C. Ačkoliv by při prvním pohledu mohlo čtenáře napadnout, že v zapojení chybí poslední chip select 3 odpovídající datové bance připojené k DA převodníkům, pro funkci dekódování jedinečného přístupu k periferiím na sběrnici není potřeba, aby byl součástí druhého dekodéru. Důvodem je, že pokud je sepnut chip select 3 pro zápis dat do jednoho ze dvou DAC, tak je z pohledu EMI řadiče zřejmé, že CS1 musí být bezpodmínečně neaktivní. Z tohoto důvodu nemohou výstupy Y0, Y1 druhého dekodéru nabýt stavu pro sepnutí registrů plnících funkci digitálních výstupů. Samotný CS3 je tedy přiveden přímo do CS pinu každého z dvojice DA převodníků. Výstupy Y4 a Y5 z dekodéru jsou přivedeny do pinu WR odpovídajícího DA převodníku. Kombinací nízkých úrovní signálu na pinu CS i WR dojde k zápisu dat ze sběrnice do vnitřních registů DA převodníku a vzhledem k zapojení zbytku jeho řídicích signálů dojde k okamžitému převodu zapsaných diskrétních dat na výstupní analogový signál. Oproti tomu výstupy Y0, Y1 jsou zapojeny jako hodinový vstup do registrů plnících funkci digitálních výstupů. Po jejich sepnutí do L stavu a následném přechodu zpět do H jsou data stále po

krátký okamžik na sběrnici a jsou tedy přenesena do výstupních registrů. Více o převodní funkci druhého dekodéru viz tabulka č. 2.



Obr. č. 4: Zapojení dekodéru SN74HC138D pro digitální výstupy a DA převodníky

Tab.č. 2: Funkce převodu vstupních signálů na výstupní ve druhém dekodéru

A(AD0)	B(EMI_WR)	C(CS1)	Y0	Y1	Y4	Y5
L	L	L	L	H	H	H
H	L	L	H	L	H	H
X	H	X	H	H	H	H
L	L	H	H	H	L	H
H	L	H	H	H	H	L

2.2 Návrh submodulu

Navazující text popisuje navržené schéma zapojení včetně výše popsaného zapojení dekódovací logiky, které jsou obsaženy v příloze A1 i A2. Sjednocení příloh A1 a A2 tvoří kompletní schéma zapojení navrženého submodulu, které bylo rozděleno na dvě části pro tištěnou verzi práce. Kompletní celistvé schéma je obsaženo na datovém nosiči ve složce /EAGLE_HW/dp_submodul/ , která obsahuje soubory pro Eagle návrhový software verze

6.5.0 . Pro její použití je však nutné do návrhového softwaru Eagle importovat knihovny, které jsou také dostupné na datovém nosiči ve složce /EAGLE_HW/lbr/.

2.2.1 Digitální výstupy

Pro navrhovaný sumbodul bylo určeno, že bude obsahovat 16 binárních výstupů. Vzhledem k tomu, že data jsou na výstup předávána pomocí sběrnice EMI, bylo pro zápis výstupních dat nutné zvolit výstupní registry.

Pro tento účel byly zvoleny dva integrované obvody MM74HC373WM. Každý z nich plní funkci třístavového osmibitového bistabilního D klopného obvodu. Dle specifikace jsou navrženy pro použití v rychlých sběrnicových rozhraních, čímž ideálně splňují požadavky na navrhované digitální obvody. Další výhodou těchto integrovaných obvodů je jejich schopnost budit na výstupu napětí odpovídající pětivoltové úrovni TTL logiky. Při aplikaci těchto součástek v obvodu je nutno brát v potaz, že se tyto integrované obvody překlápí na náběžnou hranu hodinového vstupu. Vzhledem k tomu, že použitý dekodér EMI řídicích signálů v podobě dříve popsaného integrovaného obvodu SN74HC138 podává na aktivním výstupu nízkou úroveň napětí, dojde k zapsání do registru až při přechodu výstupu dekodéru na vyšší úroveň napětí. Schéma zapojení je zobrazeno v příloze A1 a výstupní obvody jsou reprezentovány zapojením integrovaných obvodů REG1 a REG2.

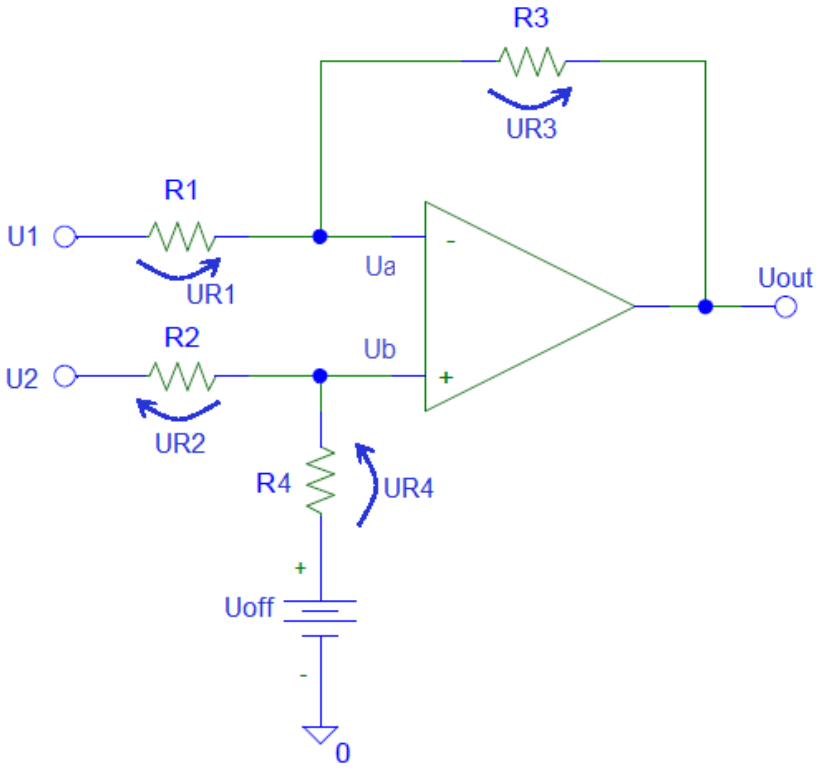
Vzhledem k tomu, že použitý registr obsahuje OE pin a CLK pin, jsem se rozhodl připojit OE pin k jednomu pinu nevyužitého MC rozhraní gatewaye JH10. Tím je umožněno aktivovat či deaktivovat výstup registru 74HC374 skrze standardní GPIO rozhraní použitého mikrokontroléru. Po aktivaci OE pinu je možné zapsat data do registru pomocí náběžné hrany signálu přivedeného z dekodéru do CLK pinu registru.

2.2.2 Analogové vstupy

Pro analogové vstupy je využito v mikrokontroléru integrovaného AD převodníku. Tento převodník nabízí převod analogového signálu do digitálního formátu o délce 10 bitů. A celkový počet analogových vstupů je osm. Pro maximální využití bylo stanoveno, že má být využito všech osmi analogových vstupů. Vstupní napětí AD převodníku je v rozmezí 0 až 3.3V. Oproti tomu napětí na vstupech vyvíjeného submodulu bylo stanoveno na rozmezí -10 až 10V. Pro praktické řešení tohoto úkolu bylo nutné zvolit vhodné integrované obvody

operačních zesilovačů. Po konzultaci s vedoucím práce byly vybrány integrované obvody TL084AC. Jedná se o integrované obvody vysokorychlostních operačních zesilovačů s J-FET vstupy, které jsou umístěny čtyři v jednom pouzdře.

V první části návrhu analogových obvodů bylo potřeba zeslabit vstupní signál v rozmezí -10V až -10V na napětí 0V až 3.3V. Pro řešení tohoto problému bylo zvoleno zapojení vycházející ze zapojení standardního diferenčního zesilovače. Avšak toto zapojení bylo nutné upravit, aby výstupem bylo napětí pouze jedné polarity, které vyžaduje použitý AD převodník. Výsledkem je zapojení zobrazené na obrázku č. 5.



Obr. č. 5: Zapojení vstupního diferenčního zesilovače s ofsetem se zvolenou orientací napětí pro analýzu obvodu

Pro správnou funkci vstupního zesilovače bylo nutné obvod analyzovat a najít vhodné napětí Uoff, které při $U_1 = U_2$ zajistí, že výstup bude roven půlce referenčního napětí. Referenční napětí odpovídá 3.3V.

Nejdříve je nutné získat rovnici převodu rozdílu vstupních napětí na výstupní napětí. K tomu využijeme mimo jiné princip superpozice a předpoklad, že $R_1 = R_2$ a $R_3 = R_4$. Postup je následující:

1) Předpoklad: $U_2 = 0$ $U_{off} = 0$

$$U_b = U_a = U_{off} - U_{R4} = U_{off} - \frac{U_{off}}{R_2 + R_4} \cdot R_4 = 0 - 0 = 0$$

$$U_{out(a)} = U_a - U_{R3} = 0 - \frac{U_1 - U_a}{R_1} \cdot R_3 = -\frac{U_1 - 0}{R_1} \cdot R_3 = -U_1 \cdot \frac{R_3}{R_1}$$

2) Předpoklad: $U_1 = 0$ $U_{off} = 0$

$$U_b = U_a = U_{off} - U_{R4} = U_{off} - \frac{U_{off} - U_2}{R_4 + R_2} \cdot R_4 = \frac{U_2}{R_4 + R_2} \cdot R_4$$

$$U_{out(b)} = U_a - U_{R3} = U_a - \frac{U_1 - U_a}{R_1} \cdot R_3 = \frac{U_a \cdot (R_1 + R_3)}{R_1}$$

$$U_{out(b)} = \frac{U_2}{R_4 + R_2} \cdot R_4 \cdot \frac{R_1 + R_3}{R_1} = U_2 \cdot \frac{R_4}{R_1} = U_2 \cdot \frac{R_3}{R_1}$$

3) Předpoklad: $U_1 = 0$ $U_2 = 0$

$$U_b = U_a = U_{off} - U_{R4} = U_{off} - \frac{U_{off}}{R_2 + R_4} \cdot R_4 = \frac{U_{off} \cdot R_2}{R_2 + R_4}$$

$$U_{out(c)} = U_a - U_{R3} = U_a - \frac{U_1 - U_a}{R_1} \cdot R_3 = \frac{U_a \cdot (R_1 + R_3)}{R_1} = \frac{\frac{U_{off} \cdot R_2}{R_2 + R_4} \cdot (R_1 + R_3)}{R_1}$$

$$U_{out(c)} = U_{off}$$

4) Z principu superpozice:

$$U_{out} = U_{out(a)} + U_{out(b)} + U_{out(c)}$$

$$U_{out} = -U_1 \cdot \frac{R_3}{R_1} + U_2 \cdot \frac{R_3}{R_1} + U_{off}$$

$$U_{out} = (U_2 - U_1) \cdot \frac{R_3}{R_1} + U_{off}$$

Po získání předchozí rovnice je možné přistoupit k dalšímu kroku, ve kterém získáme požadované napětí V_{off}, které zajistí, že při nulovém rozdílu vstupních napětí bude na výstupu napětí $V_{ref}/2$. Napětí, které se nachází v polovině měřícího rozsahu AD převodníku. V tomto případě je interval zvolen na rozmezí 0 až 3.3V.

1) Předpoklad: $U_{out} = U_{ref}/2$ $U_2 - U_1 = 0$

$$U_{out} = (U_2 - U_1) \cdot \frac{R_3}{R_1} + U_{off}$$

2) Dosadíme do rovnice výstupního napětí a získáme:

$$\frac{U_{ref}}{2} = U_{off}$$

Tímto jsme dospěli k závěru, že $U_{off} = U_{ref}/2$. Nyní je nutno vypočítat poměr rezistorů R_1 a R_3 a podle něho zvolit reálné hodnoty rezistorů. Pro tento výpočet volíme předpoklad, že výstupní napětí je rovno referenčnímu a rozdíl vstupních napětí je roven deseti voltům.

1) Vycházíme ze získané rovnice:

$$U_{out} = (U_2 - U_1) \cdot \frac{R_3}{R_1} + U_{off}$$

2) Předpoklad: $U_{out} = U_{ref}$ $U_2 - U_1 = 10$ $U_{off} = U_{ref}/2$

$$U_{ref} = 10 \cdot \frac{R_3}{R_1} + \frac{U_{ref}}{2}$$

$$U_{ref} = \frac{20 \cdot R_3}{R_1}$$

3) Dosadíme do rovnice: $U_{ref} = 3{,}3$

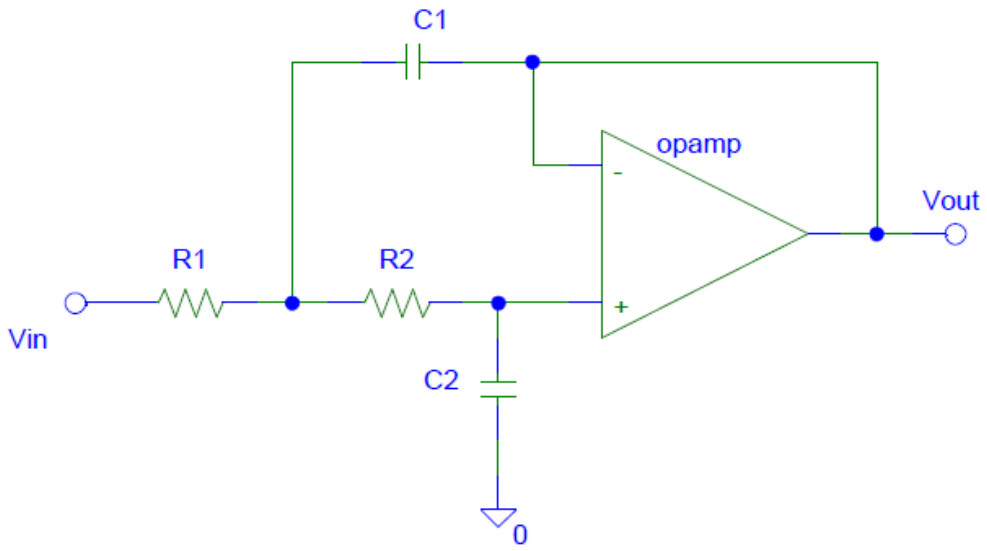
$$3{,}3 = \frac{20 \cdot R_3}{R_1}$$

$$\frac{R_3}{R_1} = \frac{33}{200}$$

Výpočty se došlo k závěru, že poměr rezistorů $R_3 : R_1 = 33 : 200$. Proto byly reálné hodnoty rezistorů zvoleny $R_3 = 33k\Omega$ a $R_1 = 220k\Omega$.

Pro obvodovou realizaci napětí $V_{ref}/2$ byl použit sledovač napětí, na jehož kladný vstup je přivedeno napětí ze symetrického děliče napětí, který je připojen na napětí 3.3V - viz příloha A2 - integrovaný obvod AMP1 - výstup 3OUT .

Pro potlačení vlivu aliasingu při vzorkování vstupního analogového signálu AD převodníkem je nutné za první část analogového obvodu, kde dochází k zeslabení a posunu napětí, zařadit antialiasing filtr. Pro realizaci antialiasing filtru bylo zvoleno zapojení Sallen-Key dolní propusti. Jedná se o aktivní filtr druhého řádu a jako takový poskytuje v nepropustném pásmu útlum 40dB na dekádu. Je však zřejmé, že zlomová frekvence antialiasin filtru musí odpovídat vzorkovací frekvenci AD převodníku. Z toho důvodu je patrné, že hodnoty pasivních součástek v Sallen-Key filtru je nutné vždy upravit, dle vzorkovací frekvence dané měřící úlohy. Pro realizaci samotného submodulu bylo zvoleno, že výchozí hodnoty pasivních součástek antialiasing filtru budou odpovídat zlomové frekvenci filtru 22kHz. Tato frekvence je hojně užívána pro vzorkování při vytváření zvukových záznamů pro frekvenční rozsah lidského sluchu 20Hz až 20kHz i s přidanou rezervou 2kHz. Tím se aplikací Nyquistova teorému dostaneme na hodnotu frekvence vzorkování 44kHz.



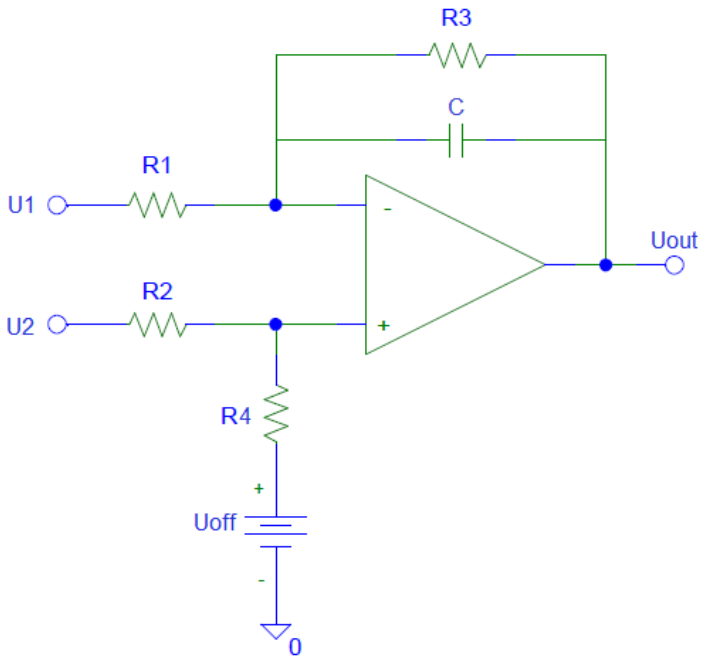
Obr. č. 6: Zapojení Sallen-Key antialiasing filtru druhého řádu

V příloze A2 jsou antialiasing filtry realizovány zapojením operačních zesilovačů ANT1 a ANT2. Z požadované zlomové frekvence 22kHz byly pro vzor obvodového řešení zobrazeného na obrázku č. 6 dopočítány hodnoty pasivních součástek dle vztahu:

$$f_c = \frac{1}{2 \cdot \pi \cdot \sqrt{R_1 \cdot R_2 \cdot C_1 \cdot C_2}}$$

Dle tohoto vztahu je tedy možné vypočítat zlomovou frekvenci dolní propusti pro vlastní úlohy s rozdílnou frekvencí vzorkování. Pro další zvýšení strmosti útlumu v nepropustném

pásmu filtru bylo také využito předchozího zapojení diferenčního operačního doplněného o kondenzátor ve zpětné vazbě viz obrázek č. 7. Ve schématu v příloze A2 jsou tyto obvody realizovány operačními zesilovači AMP_IN1 a AMP_IN2.



Obr. č. 7: Zapojení vstupního diferenčního zesilovače s ofsetem a kondenzátorem ve zpětné vazbě

K rezistoru ve zpětné vazbě byl paralelně přidán kondenzátor. Touto paralelní kombinací bylo dosaženo aktivní dolní propusti, která přidává další útlum 20dB na dekádu dle vztahu pro požadovanou zlomovou frekvenci 22 kHz. Rezistor R_3 byl již určen na 33 kΩ a kondenzátor C byl dopočítán pro požadovanou frekvenci 22 kHz. Potřebný kondenzátor C má kapacitu 220pF.

$$f_C = \frac{1}{2 \cdot \pi \cdot R_3 \cdot C} = \frac{1}{2 \cdot \pi \cdot 33000 \cdot 220 \cdot 10^{-12}} \doteq 21922 Hz$$

Na závěr bylo také nutné přidat ochranu proti připojení nevhodného napětí na vstupy diferenčního zesilovače, čímž by na vstupy integrovaného AD převodníku mohlo přijít napětí neodpovídající jeho povolenému vstupnímu rozsahu 0V až 3,3V. Tento problém byl řešen tak, že již zpracovaný analogový signál, který vstupuje přímo do AD převodníku, je připojen k napětí 3,3V pomocí Schottkyho diody v závěrném směru a rezistoru pro omezení proudu o hodnotě 100Ω v sériovém zapojení. Podobná kombinace Shottkyho diody a rezistoru je připojena i k zemnímu potenciálu, čímž chráníme vstup AD převodníku před záporným

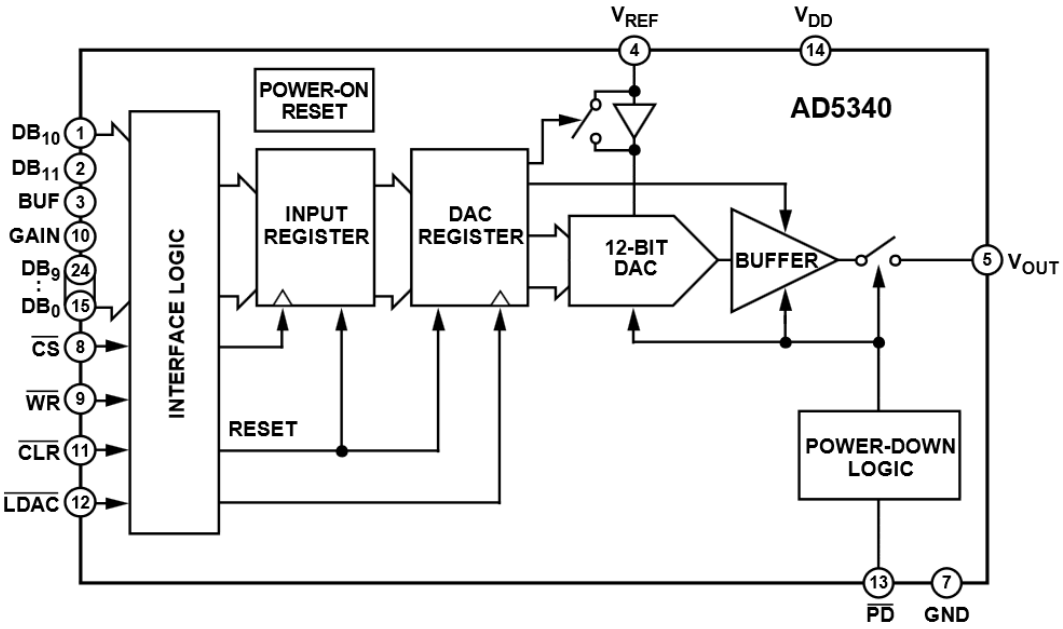
vstupním napětím větším než je napětí v propustném směru Schottkyho diody. Pro tyto účely byly použity Shottkyho diody v integrovaném obvodu BAT54C a BAT54A, které zahrnují vždy dvě diody v potřebném zapojení v jednom pouzdře. Zapojení ochranných obvodů s diodami se nachází v příloze A2, kde jsou použity diody D1 až D8, které jsou připojeny na již zpracovaný signál vstupující přímo do patice pinů s názvem ADC.

2.2.3 Digitální vstupy

Vzhledem k tomu, že počet digitálních vstupů byl stanoven na 16 a data jsou čtena skrze externí sběrnici EMI, bylo nutné vybrat vhodné budiče na sběrnici. Pro tento účel byly zvoleny dva budiče typu 74HCT244D. Použitá verze T je kompatibilní s TTL logickými obvody, které budou pravděpodobně použity s navrhovaným submodulem v jeho zamýšleném praktickém použití. Jedná se o osmibitové budiče s třístavovým výstupem. Každý budič obsahuje dvojici negovaných OE pinů pro případné použití dvou nezávislých čtyřbitových budičů. Avšak v této aplikaci je potřeba dvou osmibitových budičů, proto jsou na jednom budiči oba OE piny paralelně připojeny k řídicímu signálu navržené logiky. Tento signál z navržené logiky dekodéru při nízkém úrovni napětí způsobí, že jsou osmibitová vstupní data budiče přivedena na sběrnici v požadovaný okamžik. Schéma zapojení je zobrazeno v příloze A1 a vstupní obvody jsou reprezentovány zapojením integrovaných obvodů DRI1 a DRI2.

2.2.4 Analogové výstupy

Dalším požadavkem na funkce submodulu bylo generování analogového signálu v rozmezí -10 V až +10 V. Pro tento účel byla zvolena dvojice DA převodníků AD5340BRUZ od společnosti Analog Devices. Jedná se o 12-bitový DA převodník, jenž může být napájen napětím v rozmezí 2.5 V až 5.5 V se spotřebou pouze 115 µA. V power down módu je spotřeba redukována dokonce na 80nA. V průběhu vývoje bylo rozhodnuto, že bude použita EMI sběrnice v osmibitovém nemultiplexovaném režimu. Z tohoto důvodu jsou oba DA převodníky použity pouze v osmibitovém režimu. Výsledkem je, že jsou na datovou část sběrnice EMI připojeny pouze DAC vstupy s největší váhou. Zbylé čtyři datové DAC vstupy s nejnižší váhou jsou uzemněny. Tímto zapojením bylo dosaženo plnohodnotných osmibitových DA převodníků za použití již nakoupených DA převodníků.



Obr. č. 8: Vnitřní blokové schéma DA převodníku AD5340BRUZ (převzato z [3])

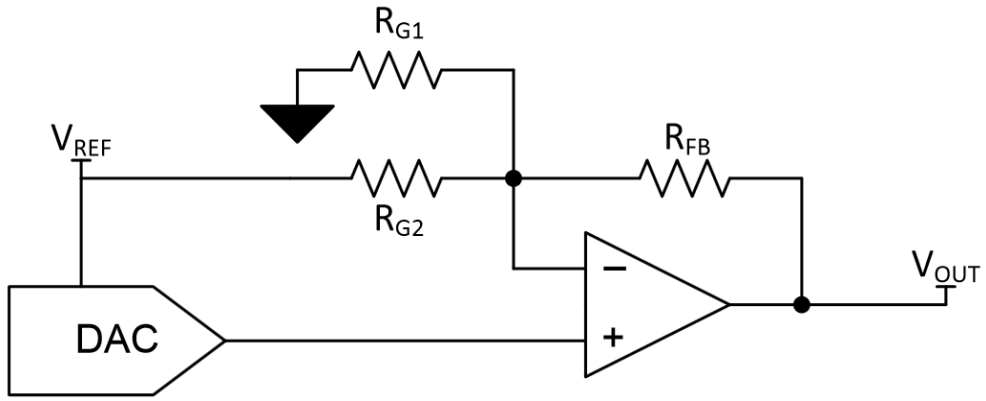
Tab.č. 3: Vstupy DA převodníku AD5340BRUZ

$DB_{0\text{-}10}$	Paralelní datové vstupy
DB_{11}	Bit paralelních datových vstupů s nejvyšší váhou
BUF	Pin pro nastavení bufferu na výstupu DA převodníku
V_{REF}	Vstup pro referenční napětí
V_{OUT}	Výstup DA převodníku - rail-to-rail
/CS	Negovaný vstup pro výběrový signál
/WR	Negovaný vstup pro zápis dat do vstupních registrů DA převodníku
GAIN	Vstup, který přepíná rozsah výstupního napětí mezi 0 až V_{REF}, nebo 0 až 2x V_{REF}
/CLR	Negovaný vstup pro reset vstupních registrů DA převodníku
/LDAC	Negovaný vstup pro zápis do registrů DA převodníku z jeho vstupních registrů
PD	Negovaný power down vstup

Pro aplikaci DA převodníků na submodulu není nutné všechny vstupy DA převodníku plně ovládat pomocí mikrokontroléru. Proto jsou některé vstupy trvale připojeny na zemní, nebo napájecí potenciál. Kompletní schéma zapojení DA převodníků je zobrazeno v příloze A1 a převodníky jsou reprezentovány bloky integrovaných obvodů s názvy DAC1 a DAC2. Na napájecí napětí 3,3 V je trvale připojen vstup BUF, což má za následek, že výstup DA

převodníku je generován za použití výstupního operačního zesilovače, který zajistí co nejvyšší nezávislost na zátěži připojené přímo k výstupu DA převodníku. Vstup V_{REF} je připojen k referenčnímu napětí, které odpovídá napětí 2,5 V. Tohoto napětí je dosaženo vhodným zapojením napěťové reference TL431AIDBZT od společnosti Texas Instruments. Pin GAIN je trvale uzemněn z důvodu, že napájecí napětí DA převodníku nedosahuje dvojnásobku V_{REF} napětí, proto je takto nutné nastavit výstupní rozsah napětí na 0V až V_{REF}. Vstup CLR, který slouží pro reset registrů DAC na výchozí stav, je trvale připojen na napájecí napětí a to z důvodu, že pro reset registrů DA převodníků stačí zapsat na paralelní datové vstupy DA převodníků požadovaný stav výstupu. Z toho důvodu není bezpodmínečně nutné CLR pin spínat. Negovaný vstup LDAC, který slouží pro přesun dat ze vstupních registrů DAC do registrů pro převod na analogový signál je trvale připojen k zemnímu potenciálu. Tímto zapojením LDAC vstupu je dosaženo toho, že obsah vstupních registrů DA převodníku je v rámci možností vnitřních obvodů DAC okamžitě přesunut do registru DAC pro převod na analogový signál. U tohoto řešení vyvstává otázka, zda nebudou vznikat hazardní stavy v obvodech registrů DA převodníku. Avšak dle dokumentace k použitým převodníkům je toto řešení možné a proto se předpokládá, že integrovaný obvod DAC je navržen tak, aby hazardní stavy nevznikaly. Posledním řídicím vstupem převodníku je negovaný PD vstup. Ten byl vzhledem k možnosti snížení odběru proudu DA převodníky připojen k nevyužitému rozhraní motor control gatewaye JH10. Díky tomu je možné pomocí standardního GPIO nastavení motor control pinů připojených k PD vstupu převodníku aktivovat nebo deaktivovat DA převodník a mít tak možnost snížení odběru submodulu, pokud není potřeba generování analogového signálu.

Jak bylo zmíněno v předchozím odstavci výstupní napětí DA převodníků je v rozsahu 0 V až V_{REF}. Zmíněné referenční napětí odpovídá napětí 2,5 V. Požadovaným výstupem však je napětí -10 V až 10 V. Výstupní napětí z obou DA převodníků je tedy nutné zesílit a posunout o požadovaný offset. Pro tento účel bylo zvoleno zapojení operačního zesilovače zobrazené na obrázku č. 9.



Obr. č. 9: Zapojení operačního zesilovače pro převod unipolárního signálu na bipolární (převzato z [4])

Toto zapojení považuji za velmi užitečné, pro jeho jednoduchost a velkou praktičnost. Za jeho pomoci je možné převést signál o jedné polaritě na signál bipolární. Funkce tohoto zapojení spočívá v kombinaci vlastností invertujícího a neinvertujícího zapojení operačního zesilovače. Pokud je výstupní napětí DAC rovno 0 V, tak je negovaný vstup operačního zesilovače také roven 0 V. Tím neteče žádný proud rezistorem R_{G1}. To způsobí, že se obvod chová jako invertující zesilovač se zesílením R_{FB}/ R_{G2}. Když je však na výstupu DAC napětí rovno maximálnímu možnému výstupnímu napětí $V_{REF,}$ tak nepoteče žádný proud rezistorem R_{G2}, proto se obvod začne chovat jako neinvertující operační zesilovač se zesílením $1+ R_{FB}/ R_{G1}$.

Přenosová funkce pro obvod zobrazený na obrázku č. 9 [4]:

$$V_{OUT} = \left(1 + \frac{R_{FB}}{R_{G2}} + \frac{R_{FB}}{R_{G1}}\right) V_{DAC} - \frac{R_{FB}}{R_{G2}} V_{REF}$$

Pro výpočet konkrétních hodnot rezistorů pro schéma zobrazené na obrázku č. 9 nejdříve dosadíme požadovaný krajní stav V_{OUT} = -10 V , V_{DAC} = 0 V a V_{REF} = 2.5 V do přenosové rovnice. Úpravou rovnice získáme poměr R_{FB} a $R_{G2.}$ Následně do přenosové funkce dosadíme druhý krajní stav V_{OUT} = 10 V , V_{DAC} = V_{REF} a V_{REF} = 2.5 V. Úpravou rovnice získáme poměr R_{FB} a R_{G1}. Získaná dvojice poměrů odporů z rovnic obou krajních stavů tvoří soustavu dvou rovnic o třech neznámých R_{G1}, R_{G2} a R_{FB}. Jeden odpor například R_{G2} si zvolíme a zbylé dva rezistory ze soustavy rovnic dopočítáme. Zvolená hodnota rezistoru by se měla pohybovat v řádech desítek kΩ. Takto zvolený odpor by měl zajistit dostatečně malé odpory pro nízký výstupní šum a zároveň dostatečně velký odpor pro minimalizaci ztrát na rezistorech. Pro

předpokládané krajní stavy obvodu byly tímto postupem dopočítány hodnoty odporů R_{G1} = 33 kΩ, R_{G2} = 22 kΩ a R_{FB} = 100 kΩ. Praktickou realizací obvodu s konkrétními hodnotami rezistorů, která je zobrazena v příloze A1 za použití integrovaného obvodu AMP1 (1. a 2. výstup), dostaneme zesílený symetrický bipolární výstup z výstupního signálu DA převodníku o jedné polaritě. Pro případnou jemnou ruční korekci rozdílu referenčního napětí DA převodníku a referenčního napětí přivedeného na rezistor R_{G2} použitého zapojení, je referenční napětí přivedené na rezistor R_{G2} získáno pomocí děliče napětí obsahujícího trimr pro plynulou změnu požadovaného referenčního napětí viz příloha A1 [4].

3 Software pro laboratoř STR912

V této části práce je podrobně popsán postup vývoje softwaru pro vzdáleně ovládanou laboratoř, která je tvořena komunikačním modulem JH10 a výše popsaným a prakticky realizovaným prototypem submodulu s digitálními a analogovými vstupy i výstupy.

3.1 Vývojové prostředí

K zapůjčenému modulu JH10 byl zapůjčen také programátor Rlink, který podporuje programování na desce komunikačního modulu obsaženého mikrokontroléru STR912. Součástí balení programátoru Rlink je také vývojové prostředí Ride od společnosti Raisonance. Na dodaném CD bylo obsaženo prostředí Ride6, avšak ze stránek společnosti Keolabs (dříve Raisonance) bylo možné stáhnout novější verzi Ride7. Po instalaci vývojového prostředí Ride je také nutné nainstalovat plugin Rkit ARM Toolset, který v kombinaci s vývojovým prostředím Ride umožní kompilovat a debugovat zdrojové kódy vyvinuté pro široké spektrum mikrokontrolérů architektury ARM. Mimo jiné obsahuje zmíněný Rkit také všechny důležité soubory pro kompilaci programů vyvíjených pro mikrokontrolér STR912, jako jsou například linker script a startup soubory v jazyce assembler. V neposlední řadě obsahuje také velmi užitečné knihovny pro platformu STR91x včetně jednoduchých příkladů základních aplikací, jako je například inicializace timeru včetně správné inicializace řadiče přerušení pro jednotlivé periferie na čipu mikrokontroléru.

Instalace tohoto vývojového prostředí včetně Rkitu pro mikrokontroléry ARM byla velmi jednoduchá a umožnila rychlé zahájení vývoje první aplikace pro platformu STR912. Bohužel

po počáteční spokojenosti s vývojovým prostředím vyšlo najevo, že kód je omezen pouze na 64kB výstupního kódu. Programy, které tuto hranici přesáhly, nebylo možné nahrát do paměti mikrokontroléru a tím pádem ani debuggovat. Licence na software k zakoupenému programátoru Raisonance Rlink byla pouze časově omezená a již nebylo možné ji bezplatně prodloužit. Nová aktivace programátoru pro neomezené nahrávání softwaru do mikrokontroléru a jeho debuggování byla v době psaní této práce stanovena na 630 Euro. Vzhledem k takto vysoké částce bylo nutné najít jiný způsob, kterým bude možné zkompilované zdrojové kódy nahrát do mikrokontroléru a případně i vyvinutý software debuggovat. Tato situace byla konzultována s vedoucím práce a bylo navrženo použití alternativního řešení, které obsahuje bezplatné vývojové prostředí, překladač jazyka C i podporu debuggování vyvíjeného softwaru.

Dříve než bude podrobně popsána instalace a konfigurace všech potřebných nástrojů pro bezplatný vývoj softwaru pro platformu ARM, je nutné připomenout, že vzhledem k rychlému vývoji softwaru a vydávání jeho nových verzí je možné, že tento návod nemusí pokrýt všechny kroky v novějších verzích. Toto je nutné brát na zřetel a v případně problémů je doporučeno použít starší verze softwaru, které byly použity v době psaní této práce.

3.1.1 Instalace vývojového prostředí Eclipse

Prvním krokem je instalace vývojového prostředí. Tímto prostředím je IDE Eclipse. Jedná se o open source vývojovou platformu, která je pro mnohé vývojáře známá především pro možnosti vývoje aplikací v jazyce Java. Vývojové prostředí Eclipse je samo o sobě vyvinuto v jazyce Java. Tato skutečnost je často kritizována vývojáři, kteří tvrdí, že Java má za následek velké hardwarové nároky. Dle mého názoru jsou s postupem času tyto názory bezpředmětné, vzhledem k rychlému pokroku ve zvyšování výkonu výpočetní techniky. Také jsem přesvědčen, že každý alespoň průměrný počítač zajistí bezproblémový chod vývojového prostředí Eclipse. Pokud se podíváme na software vyvinutý v jazyce Java z jiného úhlu, dojdeme k závěru, že vyšší hardvarové nároky jsou jen malou daní za skutečně bohaté knihovny jazyka Java, díky nimž je možné velmi rychle a efektivně vyvíjet desktopové aplikace s pokročilými grafickými i výpočetními funkcemi bez nutnosti programování triviálních algoritmů. Z této skutečnosti prostředí Eclipse těží a jeho hlavní filosofií je možnost instalace pluginů, které mohou zásadním způsobem rozšiřovat funkce vývojového prostředí. Jako příklad lze uvést hojně užívaný plugin, který umožní vývoj aplikací v PHP skriptovacím jazyce. Mimo jiné se toto vývojové prostředí také stalo nativním vývojovým

prostředím pro vývoj aplikací na mobilní platformu Android. Z těchto skutečností je zřejmé, že vzhledem k bohatství pluginů vývojového prostředí Eclipse existuje plugin, který přidá podporu pro vývoj aplikací v jazyce C. Pro kompletní funkci vývoje a debugování na ARM platformě je však celý postup instalace velmi zdlouhavý a tato práce neposkytuje dostatek prostoru pro jeho důkladný popis. Proto jsem se rozhodl nejdříve pouze nastínit základní části instalace pro vývoj ARM aplikací v prostředí Eclipse. Samotný datový nosič diplomové práce obsahuje prostředí Eclipse, které již obsahuje některé z potřebných pluginů a pro jeho spuštění není potřeba instalace. Složka s prostředím Eclipse je umístěna na datovém nosiči pod názvem Eclipse_ARM. Tímto způsobem je možné celý popis výrazně zkrátit [5].

Pro ty, kteří nemají zkušenosti s tímto vývojovým prostředím je také vhodné vysvětlit princip pojmenovávání jednotlivých verzí. Prostředí Eclipse nepoužívá standardního číslování verzí. Namísto toho je každá verze pojmenována po vybraném kosmickém tělesu. Určení, zda je jedna verze novější než druhá, lze poznat jednoduše podle pořadí počátečního písmena v abecedě.Před instalací prostředí Eclipse je však nutné stáhnout a nainstalovat Java Runtime Enviroment, které obsahuje virtuální stoj Javy, který umožní spustit program napsaný v jazyce Java - vývojové prostředí Eclipse. Ačkoliv je pravděpodobné, že na mnohých počítačích je toto běhové prostředí již přítomno, není možné tento krok opomenout. Stažení Java JRE je možné v době psaní této práce z webových stránek společnosti Oracle uvedené v seznamu použité literatury [11]. Lze také namísto Java JRE stáhnout Java JDK ze stejných webových stránek. Java Development Kit obsahuje všechny části Java Runtime Enviroment.

Kroky pro vlastní instalaci vývojových nástrojů pro vývoj v prostředí Eclipse:

1. Stažení a instalace prostředí Eclipse (verze Luna prakticky ověřena)
2. Eclipse CDT plugin pro vývoj v jazyce C
3. Instalace OpenOCD (open-source debugovací rozhraní pro komunikaci s JTAG portem programátoru)
4. Instalace USB ovladače programátoru (RLink) pro použitý operační systém
5. GDB server plugin pro prostředí Eclipse
6. Instalace CodeSourcery či jiného GCC (GNU Compiler Collection)
7. Instalace makefile programu, který optimalizuje funkce GCC.
8. V projektu v prostředí Eclipse je nutné konfigurovat GCC pro načtení linker scriptu, který odpovídá použité platformě
9. V projektu v prostředí Eclipse je nutné konfigurovat startup soubor (standardně v

jazyce assembler), který odpovídá použité platformě

10. V prostředí Eclipse je nutné konfigurovat nainstalovaný plugin GDB serveru
11. V prostředí Eclipse je nutné konfigurovat integraci OpenOCD pro snadné debugování z prostředí Eclipsu
12. Jako poslední rozšíření funkcí debugování o možnost vizualizace interních registrů je možné nainstalovat plugin EmbSysRegisters (instalace tohoto pluginu nebyla prakticky ověřena v této práci)

Jak již bylo zmíněno, vývojové prostředí Eclipse nepotřebuje instalaci, proto je možné pouhým kopírováním souborů přenést celý adresář obsahující všechny podstatné části včetně všech doinstalovaných pluginů vývojového prostředí. Součástí datového nosiče, který je přílohou této práce, je prostředí Eclipse ve verzi Luna, které obsahuje již nainstalovaný CDT plugin. Součástí této verze prostředí Eclipse je také kompletní konfigurace, která odkazuje na další podpůrný software, který byl vyjmenován v základních krocích instalce nástrojů. Je nutné podotknout, že tato práce byla zpracována za použití operačního systému Windows 7 x64. Proto je možné, že v jiném operačním systému bude nutné provést některé úpravy.

Zkrácený postup pro instalaci vývojového prostředí za použití souborů na datovém nosiči diplomové práce:

1. Přenesení vývojového prostředí Eclipse z datového nosiče (složka Eclipse_ARM) do libovolné složky operačního systému Windows 7
2. Přenesení složky STR91x z datového nosiče diplomové práce, která tvoří kompletní pracovní adresář prostředí Eclipse tzv. workspace. Ten slouží pro ukládání všech projektů i jejich dat v prostředí Eclipse. Tato složka obsahuje mimo jiné i binární soubory OpenOCD pro Windows 7 x64.
3. Přidání cesty do proměnné PATH systému Windows - [cesta k workspace] \STR91x\openocd-0.8.0\bin-x64
4. Instalace CodeSourcery, které obsahuje kolekci binárních řádkových utilit (překladače, linker, konvertor cílového kódu, atd.) použitelných pro ARM platformu. Instalační soubor je obsažen na datovém nosiči této práce ve složce CodeSourcery.
5. Přidáni cesty do proměnné PATH systému Windows - [cesta k instalaci CodeSourcery]\CodeSourcery\Sourcery G++ Lite\bin
6. Na závěr je nutné přidat do složky ...\Sourcery G++ Lite\bin soubory, které jsou na datovém nosiči ve složce Makefile. Tím se automaticky stane soubor make.exe

součástí PATH proměnné Windows a jako takový je snadno spustitelný řádkovým příkazem bez udání cesty v prostředí Windows.

V této fázi je již možné spustit prostředí Eclipse pomocí souboru eclipse.exe, který je obsažen v přenesené verzi IDE Eclipse. Po spuštění je nutné vybrat složku, ve které je umístěn pracovní adresář workspace, který byl také překopírován do libovolného adresáře na pevném disku v kroku 2. Tímto postupem bylo dosaženo toho, že bude automaticky načten projekt LaboratorSTR912 i projekt Template, který slouží jako vzorový projekt pro vývoj libovolné aplikace na platformě STR912. V této fázi jsou již všechny základní kroky instalace dokončeny. Nyní je však nutné v prostředí Eclipse vlastnoručně editovat všechny cesty k adresářům, které se pravděpodobně liší vzhledem libovolnému umístění všech dříve provedených instalací a překopírovaných adresářů.

Kontrola a úprava cest k externím souborům v prostředí Eclipse:

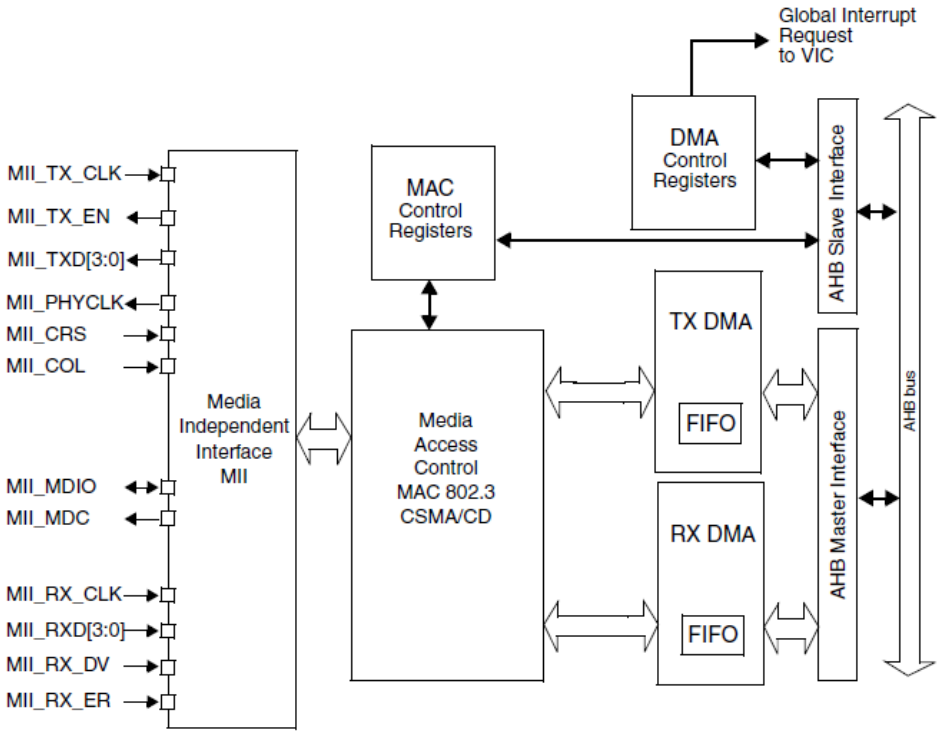
1. Pravým tlačítkem myši kliknout na aktuální projekt v prostředí Eclipse a vybrat položku Properties. Vybrat záložku C/C++ Build a poté další záložku Enviroment. Zde je nutné zkontrolovat především správnost cest pod položkou PATH.
2. Ve stejné záložce C/C++ Build vybrat podzáložku Settings. Zde vybrat další záložku Tool Settings a pro každý řádek v zobrazeném seznamu kompilerů, likerů a assemblerů zkontrolovat, že se cesty shodují s aktuálním umístěním na vlastním pevném disku a případně editovat.
3. Dalším krokem je zajistit správnou cestu k OpenOCD binárním souborům, které jsou součástí přeneseného workspace. Toho docílíme nalezení External Tools ikonky a vybráním podnabídky External Tools Configuration. Zde vybereme záložku OpenOCD a zkontrolujeme, zda všechny záložky obsahují aktuální cesty.
4. Dalším krokem je nastavení aktuálních cest pro GDB server plugin. Toho docílíme výběrem záložky Run v hlavní nabídce Eclipse. Zde otevřeme záložku Debug Configurations. Pod položkou GDB Hardware Debugging vybere název konfigurace dle projektu (LaboratorSTR912 OpenOCD) a znova zkontrolujeme, zda všechny záložky v této sekci obsahují aktuální cesty.
5. Posledním krokem je instalace specifického ovladače RLink (pokud je použit RLink programátor). Soubor ovladače je k nalezení ve složce OpenOCD, která je obsažena ve workspace - STR91x\openocd-0.8.0\drivers.

Po dokončení těchto kroků by již mělo být možné provést Build projektu vzdáleně ovládané laboratoře. Poté pomocí ikonky External Tools OpenOCD spustíme komunikaci s připojeným programátorem RLink a na závěr zvolíme pod ikonkou debug LaboratorSTR912 OpenOCD, čímž dojde k nahrání programu z vygenerovaného elf souboru do flash paměti mikrokontroléru STR912. Tím dojde ke spuštění kódu a spuštění aplikace v režimu debugování.

Výše zmíněný postup je tedy výrazně zjednodušený, avšak dá se očekávat, že se mohou vyskytnout různé komplikace při instalaci. Očekává se, že každý, kdo tento návod použije, je alespoň zkušený uživatel, který je schopen hledat řešení všech případných problémů. Například použité binární soubory OpenOCD jsou určeny výhradně pro 64-bitový operační systém Windows.

3.2 STR912 Ethernet - MAC/DMA řadič

Použitý mikrokontrolér STR912 obsahuje periférii nazvanou ENET, která je tvořena MAC 802.3 řadičem společně s MMI (Media Independent Interface) rozhranním a specifickým DMA řadičem, který je vyhrazen pouze pro účely ethernetového rozhraní.



Obr. č. 10: Blokové schéma ENET periférie STR912 (převzato z [2])

Z hlediska vývoje aplikace pro zvolený komunikační modul JH10 je potřeba se zaměřit především na správnou obsluhu DMA řadiče. Více podrobností o použitém ethernetovém transcieveru STE100P a jeho zapojení je možné získat z již zmíněné diplomové práce Hardware pro komunikační gateway s STR912FW44 napsané Jakubem Hájkem. Na obrázku č. 10 je možno vidět TX DMA a RX DMA bloky, které obsahují FIFO buffery přijatých a odesílaných dat. Celé tyto bloky jsou řízeny vyhrazenými RX a TX konfiguračními registry.

Ve vyvíjené aplikaci bylo využito dostupné knihovny k ENET periférii, díky kterým byla výrazně usnadněna celá inicializace a obsluha ethernetového rozhraní. Zdrojové kódy knihovny pro obsluhu ethernetového rozhraní jsou obsaženy na datovém nosiči diplomové práce ve složce \STR91x\LaboratorSTR912\LibSrc\, kde se nachází soubor 91x_enet.c. Hlavičkový souboru 91x_enet.h se nachází ve složce \STR91x\LaboratorSTR912\LibInc. Tato knihovna umožnila prosté volání funkcí pro inicializace rozhraní, čtení, vysílání dat i jiných operací. Všechna použitá potřebná volání funkcí ENET knihovny pro inicializaci i správu datového toku jsou součástí fragmentů zdrojových kódů v kapitole Rozhraní LwIP-STR912, která je součástí následující kapitoly LwIP stack.

3.3 LwIP stack

Pro započetí vývoje webserveru postaveného na platformě STR912 bylo nutné zvolit TPC/IP stack, který implementuje všechny potřebné přenosové protokoly a na jehož základech je možné postavit samotný webový server. Pro tento účel byl vybrán populární LwIP stack. Jedná se volně dostupný open source TCP/IP stack, který byl navržen primárně pro embedded systémy. LwIP byl prvotně vyvinut Adamem Dunkelsem v nezávislé neziskové organizaci Swedish Institute of Computer Science. LwIP stack je populární mezi výrobci embedded systémů především pro své velmi nízké nároky na velikost volné paměti, kdy implementace stacku může zabírat dle obecných informací pouze několik desítek kilobytů paměti RAM a přibližně 40 kilobytů programové paměti ROM. Velikost interní paměti na použitém mikrokontroléru STR912FAW44 je 96 kB [6].

3.3.1 Implementace LwIP stacku

Prvním krokem k úspěšné implementaci je získání zdrojových kódů LwIP stacku. V této práci jsem použil verzi 1.4.1, která by měla poskytnout stabilní mnoha let vývoje ověřenou aplikaci TCP/IP stacku. Ve verzi 1.4.1 došlo oproti starším verzím k několika změnám a díky

tomu nemusí níže popsaný postup implementace pokrýt všechny potřebné úkony při použití starší verze. Při použití starší verze je potřeba provést některé úpravy pro správnou funkci stacku, avšak pokud není z nějakého konkrétního důvodu nutné použít starší verzi, je vždy vhodné použít verzi nejnovější. V aktuální verzi je vždy nejvyšší pravděpodobnost, že výsledný program nebude vykazovat chybné funkce vzhledem k tomu, že celý LwIP stack je velmi komplexní, díky čemuž se dá očekávat, že vždy bude existovat nějaká část kódu, která za konkrétní situace nebude vykonávat svoji funkčnost správně. Tuto skutečnost je možné pozorovat i z četného množství odstraněných chyb v jednotlivých verzích LwIP stacku.

Uvnitř stažené složky LwIP stacku (na datovém nosiči složka lwIP_1_4_1) se nachází složka **apps**. Tato složka obsahuje aplikace, které lze aplikovat na implementovaný LwIP stack. Mimo jiné se v této složce nachází i aplikace http webového serveru, jehož implementace bude popsána později. Další obsaženou složkou je složka **doc.** V této složce se nachází informace, které je vhodné nastudovat pro hlubší pochopení funkce samotného stacku. Další složkou obsaženou ve staženém balíku LwIP stacku je složka **test**. Tato složka je zřejmě jen vzorovou implementací stacku na nějakou konkrétní platformu a vyjma informační funkce je pro další práci zbytečná. Poslední, avšak tou nejdůležitější složkou v celém balíku je složka **src**, která obsahuje všechny zdrojové kódy LwIP stacku. Vybrané části této složky je nutné přenést do vývojového prostředí.

LwIP stack je rozdělen do tří rozdílných programovacích rozhraní [7]:

Raw API - Jedná se o nativní rozhraní, které je určeno pro implementaci do embedded systému, který neobsahuje operační systém. Toto aplikační rozhranní poskytuje nejvyšší výkon i nejnižší velikost výsledného kódu. Avšak také se jeví jako neobtížněji implementovatelné. Funkce tohoto rozhraní je podmíněna registrací callback funkcí. Standardně jsou tyto funkce volány při události vyvolané řadičem přerušení mikrokontroléru.

Netconn API - Jedná se o rozhraní vytvořené pro vyšší úrovně softwaru. Jako takové vyžaduje obsluhu real-time operačního systému. Toto rozhraní také umožňuje vícevláknové operace.

BSD Socket API - Jedná se o rozhraní, které je postaveno nad Netconn API a tím pádem je také podmíněno přítomností real-time operačního systému.

Pro další popis implementace LwIP stacku je použito výhradně rozhraní Raw API. Tento způsob implementace bez použití operačního systému je nazýván také jako bare metal implementace a ve zdrojových kódech LwIP stacku je tato implementace označena NO_SYS identifikátorem.

3.3.2 Funkce obsažené v Raw API

Níže se nachází tabulka, která tvoří výčet všech důležitých funkcí, které jsou součástí Raw API rozhraní pro TCP protokol. Všechny tyto funkce jsou deklarovány v hlavičkovém souboru tcp.h a většina z těchto funkcí je implementována v souboru tcp.c. Také je vhodné zmínit, že podobné funkce pro protokol typu UDP jsou také deklarovány v LwIP stacku v hlavičkovém souboru udp.h a implementovány v souboru udp.c. Pomocí funkcí vypsaných v tabulce č. 4 lze obsluhovat spojení na úrovni TCP protokolu. Také je možné obecně použít tyto funkce jako takzvané callback funkce, jejichž registrací docílíme požadované reakce programu na specifikovanou událost vyvolanou například řadičem přerušení (například přijetí dat).

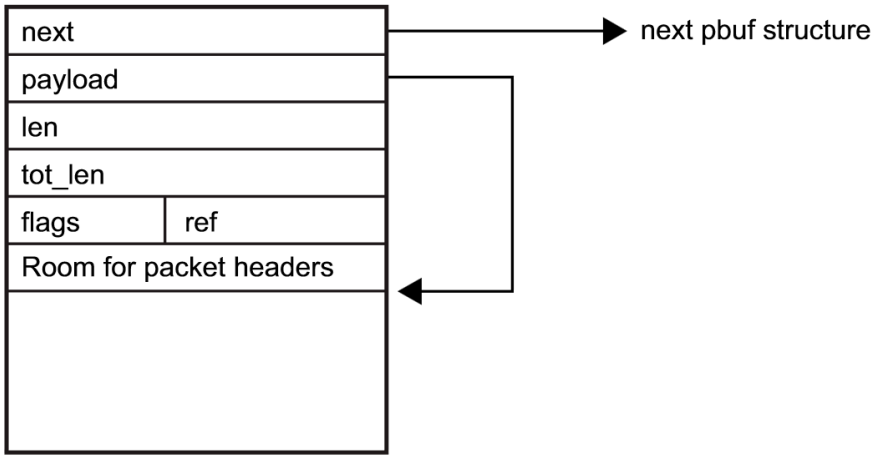
Tab.č. 4: Funkce Raw API pro TCP protokol [7]

	API funkce	Popis
Navázání TCP spojení	tcp_new	Vytvoří nový TCP PCB (protocol control block).
	tcp_bind	Připojí TCP PCB k lokální IP adrese a portu.
	tcp_listen	Spustí proces naslouchání na TCP PCB.
	tcp_accept	Přiřadí callback funkci, která bude volána ve chvíli, kdy přijde nové TCP spojení.
	tcp_accepted	Informuje LwIP stack, že nově příchozí spojení bylo přijato.
	tcp_connect	Připojí ke vzdálenému TCP hostiteli.
Odesílání TCP dat	tcp_write	Přidá data do fronty pro odeslání.
	tcp_sent	Přiřadí callback funkci, která bude volána ve chvíli, kdy je přijetí odeslaných dat potvrzeno vzdáleným zařízením.
	tcp_output	Vyvolá nucené odeslání dat ve frontě.
Přijímání TCP dat	tcp_recv	Přiřadí callback funkci, která bude volána při přijetí nových dat.
	tcp_recved	Musí být volána když aplikace zpracuje příchozí datový paket.
TCP průzkum	tcp_poll	Přiřadí callback funkci, která bude volána v pravidelných intervalech. V aplikaci tato funkce může sloužit pro pravidelnou kontrolu, zda nejsou připravená data k odeslání nebo zda

		nejsou dostupná spojení čekající na ukončení.
Uzavírání a rušení spojení	tcp_close	Uzavře spojení se vzdáleným zařízením.
	tcp_err	Přiřadí callback funkci, která je volána v případě, že obsluha spojení byla zrušena z důvodu chyby (např. nedostatek volné paměti).
	tcp_abort	Zruší TCP spojení

3.3.3 Struktura bufferů paketů LwIP

Pro přijetí a odesílání paketů je ve stacku LwIP vytvořena struktura nazvaná pbuf, která je deklarována v hlavičkovém souboru pbuf.h. Tato struktura může být zřetězená a tím se mohou data rozkládat po více pbuf strukturách.



Obr. č. 11: Struktura pbuf LwIP stacku (převzato z [7])

Proměnné struktury pbuf [7]:

next: pointer na další pbuf strukturu

payload: pointer ukazující na datovou část samotného paketu

len: délka datového obsahu jedné samotné pbuf struktury

tot_len: celkový součet všech délek jednotlivých pbuf struktur v řetězci

ref: čtyřbitová hodnota, která udává počet referencí(pointerů), které ukazují na danou pbuf strukturu. Pokud je toto číslo rovno nule, je pbuf struktura uvolněna z paměti

flags: čtyřbitová hodnota, která indikuje typ pbuf struktury v závislosti na typu alokace paměti - viz níže

Typy pbuf struktur v závislosti na allokaci paměti [7]:
PBUF_POOL: předem vyhrazený a definovaný blok v paměti (velikost je definována v konfiguračním souboru lwipopts.h), který je staticky alokován během překladu programu. LwIP stack následně sám za běhu programu allokuje v tomto vyhrazeném paměťovém bloku paměť pro packet buffery pomocí své vlastní zabezpečené funkce malloc. Jedná se o vlastní implemetaci funkce hromady LwIP stackem.
PBUF_RAM: dynamické alokování paměti z hromady dostupné na daném miktrokontroléru
PBUF_ROM: nedochází k alokování paměti pro data paketů. Namísto toho ukazuje payload pointer přímo do ROM paměti. Z toho vyplývá, že data musí být pouze konstantní

V této práci je využito pbuf struktur pouze typu PBUF_POOL, které umožňují odesílání dynamických dat v paketech a současně se jedná o staticky alokovaný blok paměti, čímž se dá předpokládat nevyšší spolehlivost výsledného programu. Také je vhodné zmínit, že je důrazně doporučeno použít PBUF_POOL pro přijímaná data, protože alokace typu PBUF_RAM generuje na rozdíl od PBUF_POOL výraznou časovou prodlevu závislou mimo jiné na velikosti příchozích dat. Také může dojít k fragmentaci dat v paměti. Na druhou stranu u odesílaných dat je možné spolehlivě použít všechny typy alokace pbuf struktur dle uvážení vývojáře [7].

3.3.4 Rozhraní LwIP - STR912

V této části práce se předpokládá, že jsou již staženy LwIP zdrojové kódy a že složka src zdojových kódů je importována do projektu ve vývojovém prostředí. Nyní je nutné vytvořit programové propojení mezi konkrétním hardwarem s ethernetovým rozhraním a mezi TCP/IP stackem. K tomu slouží soubor s názvem ethernetif.c, který se nachází v importovaných zdrojových kódech LwIP v umístění src/netif/. Soubor ethernetif.c je souborem, který definuje základní funkce, které jsou volány z vyšších částí LwIP stacku. Obsah těchto funkcí je specifický pro daný hardware a proto se očekává, že vývojář tyto funkce bude editovat a doplní obsah dle požadavků daného hardwaru.

První z funkcí, kterou je nutno editovat, je funkce **low_level_init**. Tato funkce je standardně volána při inicializaci LwIP stacku a má za úkol konfigurovat MAC adresu pro LwIP stack i maximální velikost přenosové jednotky. Následně provádí kompletní inicializaci Ethernetového rozhraní a na závěr také konfiguruje řadič přerušení pro vyvolání potřebné funkce při přijetí dat Ethernetovým rozhraním, čímž je odeslání hardwarem přijatých dat do

paket bufferů (pbuf) LwIP stacku. Odeslání dat ethernotovým rozhraním je vždy vyvoláno jako reakce na přijetí dat nebo periodickou správou LwIP stacku z nekonečné smyčky funkce main funkcí sys_check_timeouts(), která bude popsána později. Celá funkce inicializace ethernetového rozhraní pro platformu STR912 je vypsána níže. Parametr funkce, typu netif, je sktruktura, která reprezentuje vytvořené ethernetové rozhraní a jeho doposud nastavené vlastnosti.

```
static void low_level_init(struct netif *netif)
{
  /* nastaví délku MAC adresy */
  netif->hwaddr_len = ETHARP_HWADDR_LEN;

  /* nastaví požadovanou MAC adresu */
  netif->hwaddr[0] = 0x00;
  netif->hwaddr[1] = 0x02;
  netif->hwaddr[2] = 0x04;
  netif->hwaddr[3] = 0x08;
  netif->hwaddr[4] = 0x0A;
  netif->hwaddr[5] = 0x0D;

  /* nastaví maximální délku přenosové jednotky */
  netif->mtu = 1500;

  /* rozšířená nastavení */
  netif->flags = NETIF_FLAG_BROADCAST | NETIF_FLAG_ETHARP | NETIF_FLAG_LINK_UP;

  /*hardwarově závislý kód - STR912*/
  ENET_InitClocks();
  ENET_Init(PHY_HALFDUPLEX_100M);
  ENET_Start ();

  ENET_DMA->ISR = 0x80; // RX_DONE interrupt
  ENET_DMA->IER = 0x80;

  VIC_Config(ENET_ITLine,  VIC_IRQ, 4);
  VIC_ITCmd(ENET_ITLine, ENABLE);
}
```

Další funkcí obsaženou v souboru ethernetif.c, kterou je nutné editovat pro konkrétní hardware je funkce **low_level_output**. Již z názvu je zřejmé, že tato funkce je částí kódu, který slouží pro odeslání dat z LwIP stacku za použití konkrétního hardwaru Ethernetového rozhraní.

```
static err_t low_level_output(struct netif *netif, struct pbuf *p)
{
  struct pbuf *q;

/* Odsazení dle konfiguračního souboru */
#if ETH_PAD_SIZE
```

```
  pbuf_header(p, -ETH_PAD_SIZE); /*Funkce pro posunutí pointeru payload v pbuf.
Ten ukazuje na adresu, kde začíná hlavní obsah datové části a současně končí data
hlavičky. Tato dělící adresa může být dle potřeby upravena editací souboru
lwipopts.h*/
#endif

  for(q = p; q != NULL; q = q->next) {
    /* Odešle data z pbuf do ethernetového rozhraní. Vždy jeden pbuf v danou
       chvíli. */
    ENET_TxPkt(q->payload, q->len);
    while(ENET_DMA->TXSTR & DMA_TX_START_FETCH){ //čeká dokud DMA nenačte data
        ;
    }
  }

#if ETH_PAD_SIZE
  pbuf_header(p, ETH_PAD_SIZE); /* přidá odsazení dle konf. souboru */
#endif

  LINK_STATS_INC(link.xmit); /* Funkce vytvoří v LwIP statistikách záznam o
                                provedené akci (xmit = odeslání).*/
  return ERR_OK;
}
```

Poslední funkcí, která zajišťuje napojení LwIP stacku na konkrétní hardware musí být logicky funkce pro přijetí dat z Ethernetového rozhraní do stacku. Tato fukce se nazývá **low_level_input**. Zbylé funkce v souboru ethernetif.c již není potřeba nijak editovat, vzhledem k tomu, že jsou nezávislé na hardwaru.

```
static struct pbuf * low_level_input(struct netif *netif)
{
  struct pbuf *p, *q;
  u16_t len, l;
  l=0;

  /* Získáme informaci o délce přijatých dat a uložíme ji do proměnné "len"*/
  len  = ENET_HandleRxPkt(s_rxBuff);

  if(len)
  {
  #if ETH_PAD_SIZE
    len += ETH_PAD_SIZE; /* Ethernet odsazení dle konfiguračního souboru */
  #endif

    /* Alokace pbuf řetězce nebo pbuf struktur z paměti PBUF_POOL. */
    p = pbuf_alloc(PBUF_RAW, len, PBUF_POOL);

    if (p != NULL) {
    #if ETH_PAD_SIZE
      pbuf_header(p, -ETH_PAD_SIZE); /* přidá odsazení dle konf. souboru */
    #endif

     /* Procházíme alokovaný řetězec pbuf, dokud nedojdeme na konec a tím pádem
dokud není celý přijatý packet umístěn do pbuf struktur */
      for(q = p; q != NULL; q = q->next)
```

```
      {
        memcpy((u8_t*)q->payload, &s_rxBuff[l], q->len);
        l = l + q->len;
      }

    #if ETH_PAD_SIZE
      pbuf_header(p, ETH_PAD_SIZE); /* přidá odsazení dle konf. souboru */
    #endif

      LINK_STATS_INC(link.recv);
    } else {
      LINK_STATS_INC(link.memerr);
      LINK_STATS_INC(link.drop);
    }
  }
  return p;
}
```

Posledním krokem pro kompletní propojení rozhranní LwIP stacku s hardwarem desky s mikrokontrolérem STR912 je napsání interrupt handleru pro správné zpracování ethernetem přijatých dat. Vzhledem k tomu, že výše popsaná funkce low_level_init obsahuje kompletní konfiguraci ethernetového rozhraní včetně inicializace řadiče přerušení pro vyvolání interrupt handleru při přijetí dat ethernetem, stačí napsat krátký kód. Tento kód zajistí kompletní přečtení přijatých dat z bufferu ethernetu(v tomto případě DMA) a jejich odeslání do pbuf struktur LwIP stacku pro další zpracování.

```
void ENET_IRQHandler(void)
{
  /* Zpracuje všechny přijaté pakety */
  while(ENET_RxPacketGetSize() != 0)
    /* Přečte přijatý paket bufferu ethernetu a pošle ho ke zpracování stacku */
    ethernetif_input(&netifSTR);

  /* Nastaví vlajku interruptu na neaktivní hodnotu */
  ENET_DMA->ISR &= ~0x80; //RX DONE
}
```

3.3.5 LwIP konfigurační soubor

V této části práce je již LwIP stack plně implementován včetně funkcí na rozhraní LwIP - STR912, které plní funkci ovladače hardwaru. Nyní je nutné správně konfigurovat samotný LwIP stack. Pro tento účel slouží soubor **lwipopts.h**. Jedná se o header soubor obsahující pouze direktivy pro C Preprocessor. Pomocí těchto definic preproccesor zahrne nebo vyjme části zdrojového kódu LwIP stacku pro další překlad kompilerem. Díky tomu je například možné jednoduchým zápisem "#define LWIP_UDP 0 " v souboru lwipopts.h nastavit, že v naší aplikaci nebude vůbec využito UDP přenosového protokolu. Díky tomu preprocessor

jazyka C vyjme všechny části kódu, které by byly nevyužité a tím dostane překladač k překladu znatelně menší soubory. Tento princip konfigurace LwIP stacku se mi jeví jako velmi efektivní a praktický.

Pokud se však pokusíme výše zmíněný soubor najít v importovaných zdrojových kódech LwIP stacku zjistíme, že soubor není ve složce LwIP stacku obsažen. Z toho důvodu existují dva způsoby, jak potřebný soubor získat. První možností je vyhledat projekt, který se svým zaměřením i platformou nejvíce podobá našemu projektu a v získaném souboru případně provést editaci. Tím zvýšíme šanci na rychlé a funkční nastavení stacku. Druhou možností je vytvoření vlastního souboru lwipopts.h. Tento soubor je vhodné umístit do složky LwIP stacku src/include/. Tím se předpokládá, že soubor bude součástí projektem includovaných header souborů. Nyní, když je již vytvořen prázdný soubor, je nutné získat názvy definic, které LwIP stack očekává pro zpracování preprocessorem. Pro tento účel nám poslouží soubor **opt.h**, který se nachází ve složce zdrojových kódu LwIP stacku src/include/lwip/. Tento soubor plní funkci defaultního nastavení LwIP stacku. To znamená, že pokud například nastavení "#define LWIP_UDP 0 " není obsaženo v konfiguračním souboru lwipopts.h, tak se použije nastavení "#define LWIP_UDP 1 ", které je obsaženo v souboru opt.h. Proto můžeme při přečtení souboru opt.h získat všechna nastavení, jejichž názvy zkopírujeme do prázdného souboru lwipopts.h a zeditujeme jejich hodnoty. Soubor lwipiopts.h má vždy přednost a každá definice, kterou obsahuje, nahradí defaultní nastavení ze souboru opt.h. Toto řešení je velmi elegantní a umožní snadné změny nastavení LwIP stacku. Doporučuji tedy této funkce využívat a v žádném případě needitovat přímo soubor opt.h.

3.3.6 Inicializace LwIP stacku

V této části práce je již celý LwIP stack implementovaný na daný hardware a konfigurovaný lwiopts.h souborem. Nyní zbývá poslední krok, kterým je inicializace samotného stacku. Ta je popsána na zdrojovém kódu níže.

```
/*Staticky definovaná struktura síťového rozhraní*/
struct netif netifSTR;

/* Nastavení IP, gateway i netmask adresy */
struct ip_addr ipaddr, netmask, gw;

IP4_ADDR(&gw, (u8)192, (u8)168, (u8)1, (u8)1);
IP4_ADDR(&ipaddr, (u8)192, (u8)168, (u8)1, (u8)200);
IP4_ADDR(&netmask, (u8)255, (u8)255, (u8)255, (u8)0);
/* LwIP jednovláknová inicializace LwIP stacku */
```

```
lwip_init();

/* Inicializace síťového rozhraní */
netif_add(&netifSTR, &ipaddr, &netmask, &gw, NULL, ethernetif_init,
       ethernet_input);
netif_set_default(&netifSTR);
netif_set_up(&netifSTR);
```

Pro správnou funkci stacku je také nutné v pravidelných intervalech kontrolovat stav všech dosavadních spojení a ta, která jsou neaktivní delší dobu, uzavírat. Nejen tuto situaci, ale i mnohé další úkony sloužící pro správu všech spojení lze elegantně řešit jedinou funkcí sys_check_timeouts. Tuto funkci lze umístit přímo do nekonečné smyčky hlavního programu. Není nutné ani řešit četnost jejího volání. Tuto situaci řeší funkce sama. Protože při každém zavolání zkontroluje kolik milisekund uběhlo od posledního úspěšného volání void sys_check_timeouts(void). Pokud rozhodne, že ještě neuběhl dostatečně dlouhý časový interval od poslední údržby spojení, neprovede žádnou další akci. V opačném případě sama vyvolá všechny potřebné funkce pro správu a čištění stacku. V těchto funkcích jsou mimo jiné zahrnuty funkce ARP protokolu. Pro tuto funkci je samozřejmě podmínkou, aby funkce měla přístup k aktuálnímu času v milisekundách. To je nutné zajistit pomocí editace funkce u32_t sys_now(void), která se nachází v souboru src/core/sys.c. Jediné co tato funkce musí vracet je čas v milisekundách. Proto je vhodné například inicializovat RTC periferii mikrokontroléru a do funkce sys_now dát jako návratovou hodnotu aktuální čas v milisekundách. Toto řešení je zcela dostatečné a celý stack je tímto krokem správně nastaven a měl by být plně funkční. Tímto krokem můžeme přistoupit k dalšímu kroku a to je vyvinutí aplikace za využití LwIP stacku.

3.4 Http webserver postavený na LwIP stacku

Součástí staženého archivu obsahujícího zdrojové kódy LwIP stacku je také složka s názvem apps. Tato složka obsahuje poměrně velké množství rozličných vzorových aplikací, které jsou nejčastěji vyvíjené za použití LwIP stacku. Součástí této složky je také aplikace httpserver_raw. Důležité je nezaměnit tuto složku se složkou httpserver, která je vhodná pouze pro embedded systémy s real-time operačním systémem. Ze složky vybereme jen určité soubory a přeneseme je do projektu. Těmito soubory jsou: fs.c, fs.h, fsdata.h, http.c, http.h a httpd_structs.h. Zbyl nám jediný nevybraný soubor fsdata.c a dvě složky fs a makefsdata. Soubor fsdata.c je soubor obsahující všechny webové stránky v jazyce HTML i použité obrázky v hexadecimálním formátu struktur jazyka C.

3.4.1 Souborový systém stránek webového serveru

V této fázi je nejdříve nutné vytvořit své vlastní HTML soubory. Jako výchozí soubor každého webu je soubor s názvem index.html. Přípona se však může lišit a může být také typu htm nebo shtml. Pro první web je však vhodné pojmenovat první stánku index.html. Jakmile je vytvořena první webová stránka s vhodným názvem i příponou, je nutné ji konvertovat do jediného souboru fsdata.c. Pokud je souborů v jazyce HTML více, jsou vždy všechny zkonvertovány do jediného souboru fsdata.c. V adresáři httpserver_raw se nachází adresář fs. Tento adresář představuje kořenový adresář pro vyvíjený webserver. Zkopírujeme tedy do něj soubor index.html i případné další html soubory. Také je možné uvnitř kořenového adresáře vytvořit složku pro obrázky, na než se odkazují HTML soubory. Tím máme připravený kořenový adresář se všemi potřebnými soubory HTLM stránek. Konverzi do souboru fsdata.c můžeme provést dvěma způsoby. První možností je vlastní kompilace souboru se zdrojovým kódem makefsdata.c, které jsou obsaženy ve složce apps/httpserver_raw/makefsdata. Tyto zdrojové kódy obsahují algoritmy pro potřebnou konverzi. Druhou možností je použití souboru makefsdata bez přípony, který se nachází ve stejné složce. Tento soubor obsahuje Perl script, který je možné bez potřeby dalších instalací spustit například v operačním systému Ubuntu nebo v jakémkoliv jiné verzi operačního systému Linux. Pro další postup si přeneseme tento soubor obsahující skript do stejné složky, ve které je umístěn kořenový adresář webu fs. Pro spuštění skriptu jsem nainstaloval Oracle VM VirtualBox, do něhož jsem nainstaloval nejnovější kopii operačního systému Ubuntu. Tím jsem docílil toho, že na vývoj stačí jeden počítač a není nutno provádět bootování při každém generování nového fsdata.c souboru. V operačním systému Ubuntu je nutno spustit terminál a v příkazové řádce nastavit adresář, ve kterém je umístěn kořenový adresář i skript makefsdata. Následujícím příkazem spustíme samotnou konverzi.

perl makefsdata

Tímto příkazem se vytvoří v adresáři soubor fsdata.c. Který následně vložíme do projektu webového serveru ve vývojovém prostředí. Tento skript automaticky generuje hlavičku HTTP přenosu do dat samotného souboru výstupního souboru. Vygenerovaná hlavička obsahuje řádku odpovědi HTTP/1.0 200 OK, informaci o HTTP severu, informaci o typu obsahu souboru a délku obsahu. Avšak pokud je naším dalším cílem vytvořit dynamický obsah webových stránek, tak hlavičku musí generovat přímo webserver pro její proměnlivé

vlastnosti, jako je například informace o délce obsahu. Avšak nejsnazším řešením je duplicitní hlavičku obsaženou v souboru fsdata.c ve finálním souboru ručně umazat, do té doby nijak nenaruší funkci, protože je automaticky považována za textový obsah HTML souboru. Druhou možností je editovat Perl skript nebo použít alternativní metodu kompilace zdrojového kódu makefsdata.c. Finální soubor fsdata.c přeneseme do projektu webserveru do složky httpserver_raw ve vývojovém prostředí. Standardně je tento soubor nutné nastavit s vlastností "exclude from build". To je způsobeno tím, že fsdata.c není kompletní zdrojový soubor a není ho možné přeložit samostatně. Tento soubor je přeložen až jako část souboru fs.c, který odkazuje direktivou #include na fsdata.c. Tímto posledním krokem bychom měli být schopni spustit statický webový obsah skrze funkční webserver na embedded zařízení.

3.4.2 Dynamický webový obsah

Pro vývoj webového serveru s dynamickým obsahem je nutné zajistit přenos dat nejen směrem od serveru ke klientu, ale také obráceně. Přenos dat směrem od serveru ke klientu je zřejmý. Je nutné do konkrétní části HTML souborů vložit HTML kód vygenerovat hlavičku a celý výsledný soubor odeslat ke klientu. Pro odesílání dat směrem k serveru existují v HTTP protokolu dva typy dotazovacích metod. První z nich je metoda GET a druhou je metoda POST. Zásadním rozdílem mezi těmito metodami dotazů je to, že pokud odešleme vyplněný obsah HTML formulářů metodou GET, je obsah formulářů předáván formou metaproměnných, které jsou připojeny ve vlastním formátu za adresu webového serveru. Pokud bychom však chtěli odeslat přihlašovací údaje metodou GET, narazíme na závažný bezpečnostní problém, protože kdokoliv může po odeslání požadavku přečíst přihlašovací údaje z adresy, která je zobrazena ve webovém prohlížeči. Pro tento účel je vhodné použít dotazovací metodu POST, která nezahrnuje proměnné formulářů do adresy, čímž neumožní jejich zpětné čtení z prohlížeče. Navzdory výhodám metody POST je v LwIP stacku vhodné použít dotazovací metodu GET a to vzhledem k tomu, že LwIP nemá dokončenou implementaci POST dotazů. Ačkoliv jsou již obsaženy části zdrojových kódů pro dotazy typu POST, jsou hlášený časté problémy s nestabilitou při jejich použití. V této práci považuji za důležitější sestavit spolehlivější webserver než server s vyšší bezpečností. A vzhledem k tomu, že zabezpečené přihlášení webserveru je v další části práce popsáno za využití hlavičky HTTP protokolu nepředpokládá se, že budou metodou GET přenášena citlivá data.

3.4.2.1 Common Gateway Interface

Common Gateway Interface neboli zkráceně CGI je protokol pro přenos dat směrem od klienta k serveru, kdy se očekává vygenerování dynamické stránky a její navrácení klientu (SSI popsané níže). CGI nedefinuje pevně jakou HTTP metodou mají být data předána. Mohou být použity metody PUT, POST nebo GET. Jak již bylo výše napsáno, LwIP stack spolehlivě podporuje pouze metodu GET. Pro odeslání požadavku GET pro CGI je potřeba, aby HTML soubory obsahovaly formulář, který předá všechny své proměnné webserveru [8].

Příklad HTML formuláře:

```
<form action="index.cgi">
        <p style="font-size:15px;">Zadej hodnotu proměnné: </p>
        <input type="text" name="idProm" value="0" style="width:115px"><br>
        <input type="submit" value="Nastav periodu obnovení[s]">
</form>
```

Důležitým prvkem je atribut tagu form s názvem action. Tomuto atributu musí být přiřazen řetězec, který je tvořen názvem stránky, která obsahuje daný formulář, avšak přípona musí být typu cgi. To i navzdory tomu, že html soubor má příponu html nebo shtml. Voláním této koncovky dáme webserveru najevo, že mu předáváme data ve formě řetězců atribut_name=atribut_value ze všech tagů input, které se řadí za sebe. Důležité je zmínit, že LwIP stack povoluje maximální počet předaných proměnných jednoho formuláře na 16. Avšak této hodnoty by vlivem vhodného členění HTML stránky nemělo být prakticky dosaženo.

V této fázi je nutné nastavit LwIP webserver tak, aby podporoval Common Gateway Interface. Nedříve je nutné zapsat novou definici nastavení do konfiguračního souboru lwipopts.h "#define LWIP_HTTPD_CGI 1". Následně je nutné doplnit do kódu pro jaké stránky očekáváme CGI podporu a napsat handler, který předaná data zpracuje, viz níže. Zdrojový kód obsahující kompletní implementaci CGI se nachází na datovém nosiči ve složce \STR91x\LaboratorSTR912\src\dynamic_web.c.

```
// Cgi tabulka handlerů (jeden handler)
tCGI CGI_TAB[1];

// Handler pro zpracování příchozích dat ve formě řetězců
const char * CGI_INDEX_handler(int ind, int NumParam, char *param[], char *val[])
{
      int i = 0;
```

```
        for (i = 0; i < NumParam; i++) {
                // Při nalezení shody s hledaným identifikátorem se řetězec zpracuje
                if (strcmp(param[i], "idProm") == 0) {
                        string = val[i];
                }
        }
        return "/index.shtml";
}

// Funkce inicializace jednoho CGI handleru
initCGIhandlers(void){
    const tCGI indexCGI = {"/index.cgi",CGI_INDEX_handler};
    CGI_TAB[0] = indexCGI;
}
```

3.4.2.2 Server Side Includes

Server Side Includes nebo také zkráceně SSI je jednoduchý interpretační skriptovací jazyk používaný především ve webových aplikacích pro přenos dat směrem od serveru ke klientu. Jeho princip spočívá v jednoduchém vkládání dynamického obsahu do statického HTML souboru jakým je například index.html. Pro vložení dynamického obsahu musí být ve statickém html souboru (následně konvertovaném do formátu souboru fsdata.c) obsažen řetězec sloužící pro jednoznačné určení místa, kam má být vložen dynamický obsah. Jedna HTML stránka však může obsahovat více dynamických dat na různých místech HTML souboru. Proto je nutné, aby každý takovýto řetězec obsahoval identifikátor pro jednoznačné určení, jaká konkrétní data mají být do souboru vložena. Níže je možné vidět řetězec sloužící pro vložení dynamického obsahu s identifikačním řetězcem XYZ [9].

<!--#XYZ-->

Ačkoliv je tento SSI tag vždy obsažen ve zdrojovém kódu výsledné HTML stránky, tak je webovým prohlížečem ignorován a uživateli není zobrazen. Za tento řetězec však LwIP server, se správně nakonfigurovaným SSI, umístí dynamický obsah, který může být tvořen nejen čistým textem, ale může obsahovat i tagy HTML jazyka. Tento dynamicky vložený obsah je již uživateli viditelný a tímto jednoduchým principem můžeme na stránku přidat neomezeně takovýchto SSI tagů s rozdílným identifikačním řetězcem. Je však vhodné identifikátor takového tagu (výše XYZ) omezit na co nejkratší délku z důvodu snížení výpočetních nároků na porovnávání řetězců na straně serveru. Ideální délka by neměla překročit osm znaků. Také je důležité zmínit, že maximální délka dynamického obsahu, který

je vložen za SSI tag je 160 znaků. Při delších řetězcích se začíná server chovat velmi nepředvídatelně. Toto je možné snadno vyřešit vložením dvou tagů s číslovaným identifikátorem a je jen na vývojáři, jakým způsobem napojí dynamický obsah pro požadovanou délku dynamického obsahu [9].

Na závěr je také nutné popsat způsob, jak dosáhnout generování dynamického obsahu v LwIP serveru. Příklad popisuje vypsání dvou stavů digitálních výstupů z proměnné. SSI tagy by odpovídaly řetězcům "<!--#D_OUT0-->" a "<!--#D_OUT1-->". Také je nutné všechny HTML soubory před konverzí do fsdata.c přejmenovat na koncovku shtml, čímže webserver pozná, že má generovat dynamický obsah. Před použitím následujícího zdrojového kódu je také nutné přidat do konfiguračního souboru lwipopts.h LwIP stacku definice pro aktivaci SSI a generování dynamických hlaviček HTTP: "#define LWIP_HTTPD_SSI 1" a "#define LWIP_DYNAMIC HEADERS 1". Zdrojový kód obsahující kompletní implementaci SSI se nachází na datovém nosiči ve složce \STR91x\LaboratorSTR912\src\dynamic_web.c.

```
//Vytvoření rozpoznatelných identifikátorů a jejich přidání do pole
const char TAGCHAR_DO0_VAL[] = "D_OUT0";
const char TAGCHAR_DO1_VAL[] = "D_OUT1";

char const * TAGS[2] ={TAGCHAR_DO0_VAL,TAGCHAR_DO1};

//funkce, která generuje dynamický obsah
u16_t SSI_Handler(int iIndex, char *pcInsert, int iInsertLen){
	if(iIndex == 0){  // nastaveni Obnoveni stranky
		char vystup[10];
		memset(vystup,0x00,10);

		sprintf(vystup,"< "DIG_OUT0=%1d\">",D_OUT_state0);

		strcpy((char*)pcInsert, vystup);
		return strlen(pcInsert);
	 }else if(iIndex == 1){  // nastaveni Obnoveni stranky Dig vstupu
		char vystup[10];
		memset(vystup,0x00,10);

		sprintf(vystup,"< "DIG_OUT1=%1d\">",D_OUT_state1);

		strcpy((char*)pcInsert, vystup);
		return strlen(pcInsert);
	}
	return 0;
}


// Registrace callback funkce generující obsah
http_set_ssi_handler(SSI_Handler, TAGS, 2);
```

3.4.2.3 Basic access authentication

Při vývoji webserveru je dosti pravděpodobné, že bude nutné řešit alespoň základní úroveň zabezpečení. Ve valné většině aplikací se neočekává, že bude obsahovat zabezpečení na vysoké úrovni, které poskytuje například HTTPS protokol. Úroveň zabezpečení by měla odpovídat výši škod, které mohou vzniknout při jejím překonání neoprávněnou osobou. V případě vzdálené laboratoře se očekává, že zabezpečení zamezí nechtěnému přístupu jiné osoby v době, kdy je laboratoř konfigurována pro konkrétní úlohu. Jak již bylo napsáno v předchozích odstavcích, data směrem od klienta k serveru jsou předávána pomocí metody GET připojené za požadovanou adresou. Tato metoda je nevhodná pro předávání přihlašovacích údajů a metoda POST není zcela funkční v aplikaci HTTP serveru pro LwIP stack. Proto jsem se rozhodl do aplikace httpserver_raw přidat podporu přihlašování pomocí dat předaných v hlavičce HTTP protokolu.

Basic access authentication poskytuje základní ověření přístupu. Webový server při každém požadavku GET od klienta zkontroluje, zda přijatý HTTP paket ve své hlavičce obsahuje také atribut "Authorization:". Pokud tento atribut není dostupný, automaticky odešle klientu chybovou zprávu HTTP číslo 401, jejíž hlavička obsahuje řetězec "WWW-Authenticate: Basic realm="Secure Area"". Ve chvíli, kdy webový prohlížeč klienta obdrží tuto HTTP zprávu, vyvolá v uživatelském prostředí prohlížeče dotaz na přihlašovací údaje. Po vyplnění přihlašovacích údajů odešle webový prohlížeč klienta znova HTTP rámec, který však již obsahuje v hlavičce atribut Authorization včetně šifrovaného přihlašovacího jména a hesla. Server porovná přihlašovací údaje a pokud se shodují, tak odešle klientu požadovanou stránku. V opačné případě je opakovaně vrácena chybová HTTP zpráva číslo 401 s požadavkem na zadání správných přihlašovacích údajů.

Příklad komunikace [10]:

1) Požadavek klienta

```
GET /index.shtml HTTP/1.0
```

2) Odpověď serveru

```
HTTP/1.0 401 Authorization Required
Server: HTTPd/1.0
Date: Sat, 27 Nov 2004 10:18:15 GMT
WWW-Authenticate: Basic realm="Secure Area"
Content-Type: text/html
Content-Length: 311

<HTML>...
```

3) Opakovaný požadavek klienta s přihlašovacími údaji

```
GET / index.shtml HTTP/1.0
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

4) Odpověď serveru při správných přihlašovacích údajích

```
HTTP/1.0 200 OK
Server: HTTPd/1.0
Date: Sat, 27 Nov 2004 10:19:07 GMT
Content-Type: text/html
Content-Length: 10476

<HTML>...
```

Jak již bylo zmíněno, přihlašovací údaje jsou odeslány v šifrované podobě. Šifrování probíhá pomocí algoritmu Base64. Šifrovacímu algoritmu je předán jediný řetězec, který zahrnuje přihlašovací jméno i heslo. Ta jsou oddělena dvojtečkou bez přidaných mezer - "login:pass". Pro šifrování a dešifrování je tedy nutné doplnit webový server o zdrojový kód obsahující Base64 algoritmus. Pro tento účel jsem použil open source zdrojový kód, který byl zveřejněn na stránkách www.opensource.apple.com přímo společnosti Apple.

Výhodou tohoto typu zabezpečení je, že dotaz na přihlašovací údaje generuje sám prohlížeč a proto je výsledkem vysoká kompatibilita, kdy přihlašování proběhne zcela správně i na prohlížečích populárních mobilních operačních systémů Android i iOS. Podmínkou však je, aby tato funkce byla implementována v daném prohlížeči, což je podmínka, kterou splňuje drtivá většina všech webových prohlížečů.

Nevýhodou je, že ačkoliv jsou přihlašovací údaje šifrovány pomocí algoritmu Base64, je snadné odeslání těchto dat po síti, odposlouchat a případně upravený požadavek s odposlouchanou šifrou odeslat na server. Druhou možností je odposlouchané přihlašovací údaje dešifrovat a následně je použít pro neoprávněný přístup na server pomocí webového prohlížeče [10].

4 Popis vzdáleně ovládané laboratoře

V této fázi práce byl navržený hardware realizován v podobě funkčního prototypu desky plošných spojů v oboustranném provedením. Deska byla vyrobena společností PagoBoard a byla mnou vlastnoručně osazena za použití popsaných součástek. Výsledkem byl funkční prototyp, který prakticky ověřil funkčnost návrhu. Submodul byl pomocí patice pinů rozteče 2,54mm připojen ke komunikačnímu modulu JH10. Následně byl vytvořen v prostředí

Eclipse projekt LaboratorSTR912. Tento projekt je součástí diplomové práce na datovém nosiči a je umístěn ve složce typu Eclipse workspace s názvem STR91x . Vzhledem k tomu, že je projekt vytvořen v prostředí Eclipse, tak je možné snadno importovat projekt do vlastní instalace prostředí Eclipse. Tato instalace musí samozřejmě obsahovat všechny již dříve popsané pluginy. Projekt obsahuje adresáře LibInc a LibSrc. Tyto adresáře obsahují knihovny, které byly vydány společností STMicroelectronics pro usnadnění práce s použitým mikrokontrolérem STR912. Další adresář je pojmenován LwIP. Tento adresář obsahuje kompletní balík zdrojových kódů použitého stacku LwIP. Součástí tohoto adresáře je také samotný HTTP server. Neodpovídá však verzi, která byla obdržena s LwIP stackem. Zdrojové kódy webserveru jsou doplněny například o HTTP zabezpečení. Posledními adresáři jsou src a inc. Tyto adresáře obsahují soubory zdrojových kódů zahrnující funkce a proměnné použité pro vyvíjenou vzdáleně ovládanou laboratoř viz tabulka č. 5.

Tab. č. 5: Obsah složky src

Název souboru	**Obsah souboru**
base64.c	Funkce algoritmu Base64 od společnosti Apple
dynamic_web.c	Obsahuje funkce HTTP serveru především pro generování dynamického obsahu pomocí SSI a CGI protokolů
init_src.c	Funkce inicializace různých částí systému vzdáleně ovládané laboratoře na platformě STR912
string_function.c	Funkce pro zpracování vstupů v podobě řetězců (pole znaků)
submodul.c	Funkce vstupů a výstupů navrženého submodulu včetně potřebných proměnných

4.1 Funkce webového serveru laboratoře STR912

Výsledkem práce je, že za pomoci SSI a CGI přenosových protokolů webserveru bylo vyvinuto funkční zabezpečené HTTP rozhranní. Toto rozhraní je rozděleno do šesti základních oddílů tvořených jednotlivými HTML webovými stránkami. Všechny tyto stránky jsou tvořeny standardním HTML kódem, který je doplněn o kaskádové styly CSS, které jsou vyjma správného umístění komponentů HTML kódu také vhodné pro kompletní grafické rozvržení. Tím není potřeba pro základní barevný návrh nutné použití rastrových obrázků, které by zabíraly příliš velké množství paměti v mikrokontroléru. Všechny tyto stránky jsou v podobě nekonvertovaných SHTML souborů dostupné na datovém nosiči ve složce

httptoCfile, která je součástí této práce. Funkce jednotlivých souborů s HTML kódem je popsána v tabulce č. 6.

Tab. č. 6: Základní HTML stránky webového serveru

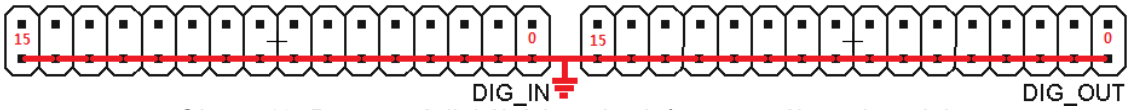
Název HTML souboru	**Funkce HTML souboru**
index.shtml	Zobrazuje informace o aktivních vstupech a výstupech a přiřazených funkcích ovládání vstupů a výstupů.
options.shtml	Umožňuje za běhu měnit IP adresu včetně masky a gateway. Nastavení aktuálního času a data do RTC. Nastavení přihlašovacích údajů HTTP zabezpečení přístupu.
an_input.shtml	Nabízí otevření nezávislé stránky pro sledování aktuálního stavu analogových vstupů an_input_data.shtml. Konfigurace periody obnovení stránky pro sledování aktuálního stavu analogových vstupů.
an_output.shtml	Nabízí otevření nezávislé stránky pro sledování aktuálního stavu analogových výstupů an_output_data.shtml. Konfigurace periody obnovení stránky pro sledování aktuálního stavu analogových výstupů. Nastavení statického analogového výstupu pro jeden ze dvou DA převodníků s výstupním zesílením v rozsahu -10 V až 10 V. Nastavení generování tří typů průběhů: sinus, pila a trojúhelník. Průběhy jsou generovány s časovým rozlišení 1 ms. Lze nastavit periodu až 10 s, amplitudu a napěťový offset. Toto nastavení umožní generovat snadno nastavitelný výstupní signál zcela nezávisle na druhém DA převodníku, který může generovat zcela jiný průběh nebo případně může držet statické výstupní napětí.
dig_input.shtml	Nabízí otevření nezávislé stránky pro sledování aktuálního stavu digitálních vstupů dig_input_data.shtml. Konfigurace periody obnovení stránky pro sledování aktuálního stavu digitálních vstupů.
dig_output.shtml	Nabízí otevření nezávislé stránky pro sledování aktuálního stavu digitálních výstupů dig_output_data.shtml. Konfigurace periody obnovení stránky pro sledování aktuálního stavu digitálních výstupů. Nastavení statického logického výstupu konkrétního pinu. Nastavení PWM výstupu na libovolný výčet digitálních výstupů, avšak pouze s jednou střídou, která je definována délkou

	časového intervalu pro aktivní část a neaktivní část periody PWM v mikrosekundách. Nastavení hodinového signálů v milisekundách pro libovolný počet digitálních výstupů se zcela libovolnou frekvencí pro každý zvolený výstupní pin. Všechny zmíněné funkce digitálních výstupů mohou být spuštěny zcela nezávisle na uživatelem zvolených výstupních pinech bez vzájemného narušení správné funkce.

Vyjma v tabulce zmíněných SHTML stránek obsahuje server také chybovou stránku 404.shtml, která je vypsána v případě, když je požadována stránka, která na serveru neexistuje. Druhou chybovou stránkou je 401.shtml, která je vypsána v případě, že v HTTP hlavičce přijaté serverem od klienta nejsou obsaženy správné přihlašovací údaje. Posledními stránkami webserveru jsou automaticky obnovitelné stránky, které dle konfigurace webserveru zobrazují aktuální stav vstupů a výstupů submodulu. Stránky tohoto typu jsou celkem čtyři. Jejich názvy jsou an_input_data.shtml, an_output_data.shtml, dig_input_data.shtml a dig_output_data.shtml. Jejich automatické obnovení v prohlížeči je konfigurováno na odpovídající stránce, která má téměř totožný název pouze bez konce _data. Obnovení stránky pro sledování vstupů nebo výstupů může být v periodě až nejméně jedné sekundy. Minimální hodnota jedné sekundy je nastavena z důvodu, že vzhledem k celkové velikosti odeslaného paketu HTTP protokolu, obsahujícího webovou stránku, nepovažuji za vhodný kratší interval. Stav digitálních vstupů a výstupů je reprezentován hodnotami "0" a "1" dle logické úrovně napětí. Stav analogových vstupů a výstupů je vypsán ve voltech, které jsou v případě výstupů zadány uživatelem. V případě analogových vstupů je hodnota ve voltech vypočítána pomocí konstanty, která je součástí zdrojových kódů a byla nalezena kombinací výpočtu a praktického ověření.

4.2 Praktické použití vzdáleně ovládané laboratoře

Pro nasazení vyvinutého systému vzdáleně ovládané laboratoře ve vlastní úloze je nejdříve nutné správně zapojit vodiče signálů na příslušné piny patic, které jsou dostupné na submodulu. Na obrázku číslo 12 je možné vidět znázornění zapojení pinů v patici digitálních vstupů a výstupů. Znázorněné rozvržení pinů odpovídá pohledu shora, tak aby popisky patic na submodulu odpovídaly svojí pozicí popiskům na obrázku. Dolní řada obou patic s digitálními vstupy a výstupy je připojena k zemnímu potenciálu submodulu.



Obr. č. 12: Rozvržení digitálních patic pinů navrženého submodulu

Obrázek číslo 13 zobrazuje zapojení pinů v paticích analogových vstupů a výstupů. Patice analogových vstupů obsahuje celkem osm dvojic diferenčních vstupů pro osm multiplexovaných analogových vstupů AD převodníku. Patice také obsahuje dva uzemněné piny pro každý pár diferenčních vstupních pinů pro případ, kdy je vstupující signál vztažen k zemi. Díky těmto pinům je možné například pomocí standardních jumperů dosáhnout požadovaného zapojení. Poslední paticí, která je také na obrázku číslo 13 je patice analogových výstupů, jejichž výstupní signál je generován dvojicí DA převodníků se potřebným zesílením a posunem napětí, který byl již vysvětlen při popisu zapojení analogových výstupů. Spodní dva piny této patice jsou uzemněny.

Obr. č. 13: Rozvržení analogových patic pinů navrženého submodulu

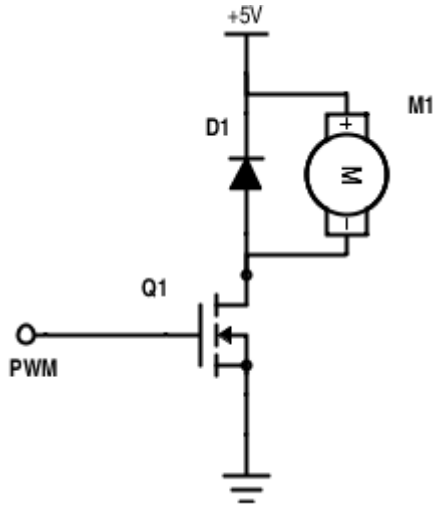
Pokud je komunikační modul připojen k napájecímu napětí pomocí USB rozhraní, nebo případně jiného zdroje napětí přes měnič napětí komunikačního modulu a jsou zapojeny potřebné vstupní a výstupní signály na patice submodulu, je možné navázat komunikaci prostřednictvím webového serveru. Výchozí adresa po připojení napájení modulu je automaticky konfigurována na adresu 192.168.1.180. Výchozí gateway adresa je 192.168.1.1 a výchozí maska je 255.255.255.0. Všechna tato nastavení je možné okamžitě změnit na stránce options.shtml webserveru. Na stejné stránce je též možné okamžitě změnit výchozí přihlašovací údaje. Výchozí jméno je "admin" a heslo také "admin". Také je vhodné na této stránce nastavit aktuální čas a datum.

Po základním nastavení je již možné přistupovat k jednotlivým vstupům a výstupům submodulu pomocí zobrazeného menu, které umožňuje jednoduchou orientaci na webovém rozhraní. Pro rychlou deaktivaci všech změn v nastavení vstupních a výstupních obvodů submodulu je vhodné použít výchozí stránku index.shtml, která nabízí tlačítko reset. Toto

tlačítko je však pouze resetem pro nastavení submodulu a nemá vliv na mikrokontrolér STR921 ani na konfiguraci webového serveru.

4.3 Aplikace vzdáleně ovládané laboratoře na tři úlohy z oblasti číslicových systémů

První z praktických aplikací systému vzdáleně ovládané laboratoře je použití PWM modulace. PWM modulace má široké uplatnění v elektronice a podstatou své funkce dokáže částečně nahradit analogový výstup použitím digitálního výstupu. Jak již bylo zmíněno PWM modulace je součástí funkcí digitálních výstupů a je pro ni vyhrazen jeden konkrétní timer mikrokontroléru v módu output compare. Tím je dosaženo rozlišení PWM až v řádech jednotek mikrosekund. V této úloze byla ověřena funkčnost PWM digitálního výstupu na řízení otáček stejnosměrného motoru.



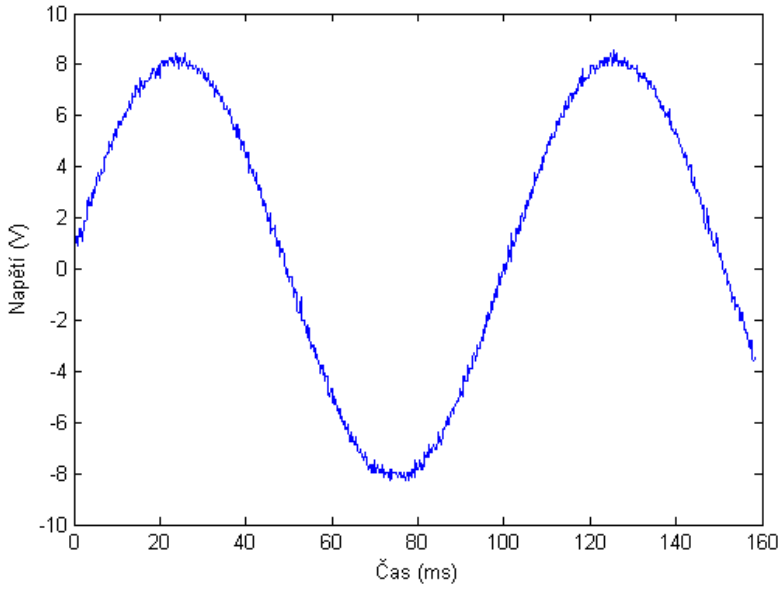
Obr. č. 14: Schéma zapojení DC motoru řízeného PWM

Druhou aplikací je spínání unipolárního krokového motoru se čtyřmi vstupy pomocí hodinových signálů na digitálních výstupech systému vzdáleně ovládané laboratoře. Ve webovém rozhraní byly nastaveny čtyři hodinové signály, které měly stejnou periodu, ale byly vůči sobě posunuty o čtvrt periody. Na navolené výstupní signály byl připojen obvod obsahující budící prvky odpovídající výkonu řízeného krokového motoru. Výsledkem bylo řízení krokového motoru za použití plného kroku.

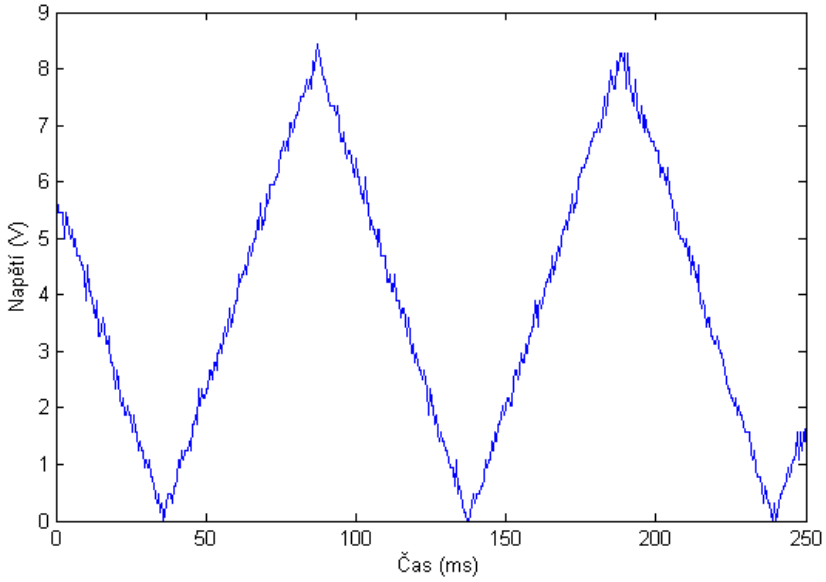
Třetí aplikací pro praktické využití navržené laboratoře v digitálních systémech je použití laboratoře jako testovacího systému. Laboratoř STR912 dokáže generovat až šestnáct nezávislých hodinových signálů a také umožňuje manuálně přepínat své digitální výstupy na požadovaný stav skrze webové rozhranní. Digitální výstupní signály z testovaného systému mohou být přivedeny na vstupní obvody laboratoře. Takto je možné snadno ručně měnit stav vstupních obvodů testovaného zařízení a následně číst, zda se na výstupech testovaného zařízení vyskytuje signál o požadované logické hodnotě. Bohužel však rozhraní webového serveru neumožňuje velmi rychlé čtení stavu digitálních vstupů. Pro tento účel je vhodnější použít samostatného TCP protokolu a vyvinout aplikaci, která bude zaslaná data ukládat do paměti PC.

4.4 Aplikace vzdáleně ovládané laboratoře na dvě úlohy z oblasti analogových systémů

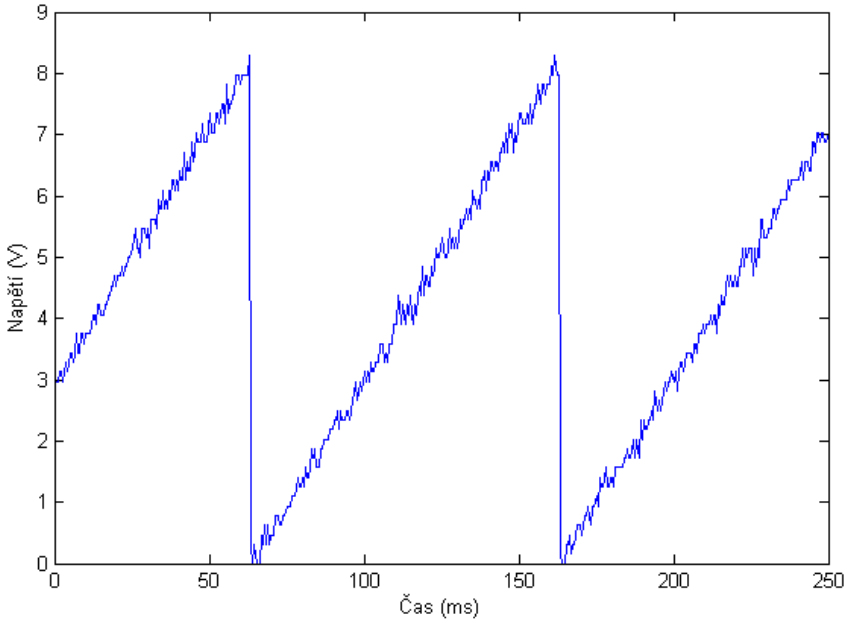
První aplikací analogové části navržené vzdáleně ovládané laboratoře je funkce generátoru signálů. Generátor signálů může nalézt široké uplatnění v každé laboratoři, která je zaměřena na vývoj, realizaci a testování elektronických systémů. Pomocí vyvinuté vzdáleně ovládaní laboratoře je možné generovat signály s průběhy typu sinus, pila a trojúhelník. Časové rozlišení generátoru signálů je jedna milisekunda a je možné nastavit výstupní periodu o délce 2 až 10000 milisekund. Amplitudu až 9000 mV a napěťový offset - 9000 mV až 9000 mV, avšak tyto hodnoty lze snadno upravit dle potřeby ve zdrojových kódech vzdáleně ovládané laboratoře. Ačkoliv bylo zamýšleno dosáhnout výstupního napětí v rozsahu -10 V až 10 V, tak byly na výstupu prakticky změřeny hodnoty cca -9 V až 9 V. Tato situace však může být dle konkrétních požadavků opravena vhodnou změnou poměrů rezistorů výstupního zesilovače DAC signálu, který je popsán v části práce zabývající se návrhem analogových výstupů. Konkrétní realizace zapojení viz příloha A1 - blok operačního zesilovače s názvem AMP1. Průběhy na výstupu generátoru signálu byly změřeny a vyneseny do grafů v obrázkách č. 15, 16 a 17. Z průběhů je patrno, že by návrh mohl být rozšířen o rekonstrukční filtr, který pro principielní jednoduchost návrhu nebyl použit. Přenos dat do DAC submodulu pomocí externí datové sběrnice je vyvolán přerušením od timeru každou jednu milisekundu. Samotná data jsou přenášena prostřednictvím DMA řadiče z dynamicky alokované paměti, která obsahuje průběh, který byl vypočítán okamžitě a pouze jednou po navolení požadovaných parametrů signálu.



Obr. č. 15: Graf zobrazující časový průběh napětí generovaného signálu typu sinus

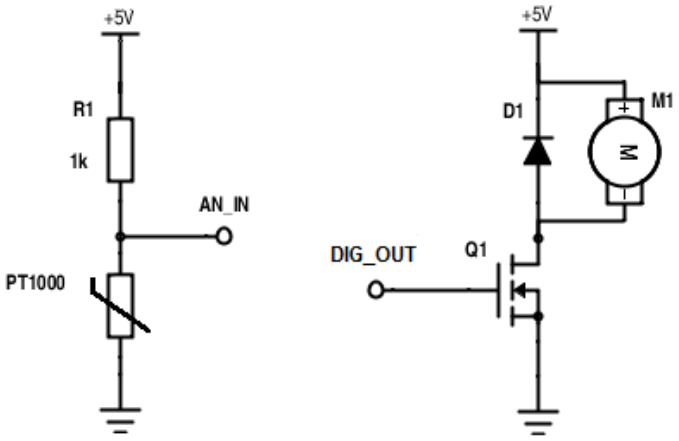


Obr. č. 16: Graf zobrazující časový průběh napětí generovaného signálu typu trojúhelník



Obr. č. 17: Graf zobrazující časový průběh napětí generovaného signálu typu pila

Druhou aplikací z oblasti analogových systémů je dvoustavová regulace. Jedná se o kombinaci analogových vstupů a digitálních výstupů. V nastavení analogových vstupů webového serveru je potřeba nastavit napětí, při jehož překročení, na konkrétním analogovém vstupu, dojde k sepnutí vybraného digitálního výstupního pinu. Tato funkce může sloužit například pro spínání odvětrávání prostředí, ve kterém je pevně určena maximální teplota. Digitální výstup je připojen ke spínacímu prvku odvětrávacího stejnosměrného motoru, kterým může být vhodně zvolený tranzistor. Zdrojem napětí přivedeného na analogový vstup je napětí z děliče napětí, který je tvořen rezistorem a odporovým teplotním čidlem PT1000, viz obrázek č. 18.



Obr. č. 18: Schéma měřícího obvodu teploty pomocí PT1000 a spínání ventilátoru s DC motorem

Závěr

Tato práce byla navržena pro nalezení praktického uplatnění nevyužitých komunikačních modulů JH10. Jednalo se velmi komplexní práci zahrnující návrh hardwaru rozšiřujícího submodulu, který obsahuje digitální i analogové vstupní i výstupní obvody. Deska plošných spojů navrženého submodulu byla vyrobena společností PragoBoard a po jejím ručním osazení bylo prakticky potvrzeno, že návrh hardwaru splnil všechny požadované funkce. Submodul byl připojen na komunikační modul prostřednictvím konektorů definovaných komunikačním modulem. Pro realizaci komunikace mezi komunikačním modulem a navrženým submodulem bylo využito sběrnicového rozhraní External Memory Interface, které je dostupné na mikrokontroléru STR912. Výsledné připojení funkčního navrženého submodulu proběhlo dle očekávání. Komunikace mezi submodulem a komunikačním modulem s mikrokontrolérem STR912 prostřednictvím sběrnice EMI umožnila velmi rychlou výměnu dat bez nutnosti další softwarové emulace. V softwarové části práce byl implementován TCP/IP stack LwIP, který umožnil další nadstavbu v podobě HTTP webového serveru. HTTP server byl upraven pro generování libovolného dynamického obsahu a byl také zabezpečen přihlašovacími údaji v podobě údajů šifrovaných algoritmem Base64. Za využití všech těchto funkcí bylo vyvinuto na serveru uživatelské rozhranní s širokými možnostmi ovládání navrženého hardwaru a výbornou kompatibilitou se všemi dostupnými webovými prohlížeči.

Práci považuji za úspěšnou, protože výsledkem je funkční systém vzdáleně ovládané laboratoře. Ačkoliv byly vytvořeny všechny požadované funkce systému, je nutné podotknout, že především softwarová část práce nabízí mnoho dalších možností pro rozšíření funkcí. Použitá implementace LwIP stacku včetně HTTP serveru je zřejmě navržena hlavně pro jednoduché aplikace. Ačkoliv jsou její možnosti omezené především velikostí dostupné paměti použitého mikrokontroléru, dospěl jsem při vývoji softwaru k závěru, že s velkým množstvím dynamických prvků HTTP serveru vzniká pro vývojáře zdrojový kód, který může být velmi nepřehledný. To je zapříčiněno pojetím dynamických prvků ze strany LwIP stacku. Pokud by bylo nutné vyvinout rozsáhlejší webový server, za použití výkonnějšího mikrokontroléru, doporučil bych vyvinutí vlastní knihovny SSI a GDI přenosových protokolů pro snazší editaci a přidávání dynamickým prvků.

Pro ještě širší uplatnění vyvinuté vzdáleně ovládané laboratoře by bylo vhodné realizovat řešení uložení dat získaných z analogových a digitálních vstupů o velkém objemu. Bohužel komunikační modul neposkytuje dostatečné množství paměti pro ukládání dat získaných vzorkováním na kmitočtu v řádu jednotek či desítek kilohertzů. Tato frekvence vzorkování je nutná například pro vzorkování zvukového signálu. Možným řešením by bylo software vzdáleně ovládané laboratoře rozšířit o jednoduchou komunikaci pomocí TCP protokolu, která by zajistila přenos dat například ve formátu CVS do uživatelského počítače. Uživatelský počítač by tato data ukládal na svůj pevný disk. Tím by byl výrazně snížen objem dat, který je nadbytečně přenášen za použití vyššího protokolu HTTP. Avšak také by bylo nutné vyvinout aplikaci pro počítač, která by spravovala TCP spojení a ukládala by data na pevný disk.

Seznam literatury a informačních zdrojů

[1] AN2647 Application note: Using the STR91xFA external memory interface (EMI). STMICROELECTRONICS. *STMicroelectronics* [online]. 2007 [cit. 2015-05-03]. Dostupné z: http://www.st.com/web/en/resource/technical/document/application_note/CD00174547.pdf

[2] UM0216 Reference manual: STR91xF ARM9 ® -based microcontroller family. STMICROELECTRONICS. *Keil Embedded Development Tools* [online]. 2006 [cit. 2015-05-03]. Dostupné z: http://www.keil.com/dd/docs/datashts/st/str9_ug.pdf

[3] 2.5 V to 5.5 V, 115 μA, Parallel Interface Single Voltage-Output 8-/10-/12-Bit DACs: AD5330/AD5331/AD5340/AD5341. ANALOG DEVICES. *Analog Devices* [online]. 2008 [cit. 2015-05-03]. Dostupné z: http://www.analog.com/media/en/technical-documentation/data-sheets/AD5330_5331_5340_5341.pdf

[4] Bipolar +/ - 10V Analog Output from a Unipolar Voltage Output DAC. DUKE, Kevin. TEXAS INSTRUMENTS. *Texas Instruments* [online]. 2014 [cit. 2015-05-03]. Dostupné z: http://www.ti.com/lit/ug/slau525/slau525.pdf

[5] Eclipse (vývojové prostředí). WIKIMEDIA FOUNDATION, Inc. *Wikipedia* [online]. 2014 [cit. 2015-05-03]. Dostupné z: http://cs.wikipedia.org/wiki/Eclipse_%28v%C3%BDvojov%C3%A9_prost%C5%99ed%C3%AD%29

[6] LwIP. WIKIMEDIA FOUNDATION, Inc. *Wikipedia* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://en.wikipedia.org/wiki/LwIP

[7] AN3966 Application note: LwIP TCP/IP stack demonstrati on for STM32F4x7 microcontrollers. *STMicroelectronics* [online]. 2014 [cit. 2015-05-03]. Dostupné z: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note/DM00036052.pdf

[8] Common Gateway Interface. WIKIMEDIA FOUNDATION, Inc. *Wikipedia* [online]. 2014 [cit. 2015-05-03]. Dostupné z: http://cs.wikipedia.org/wiki/Common_Gateway_Interface

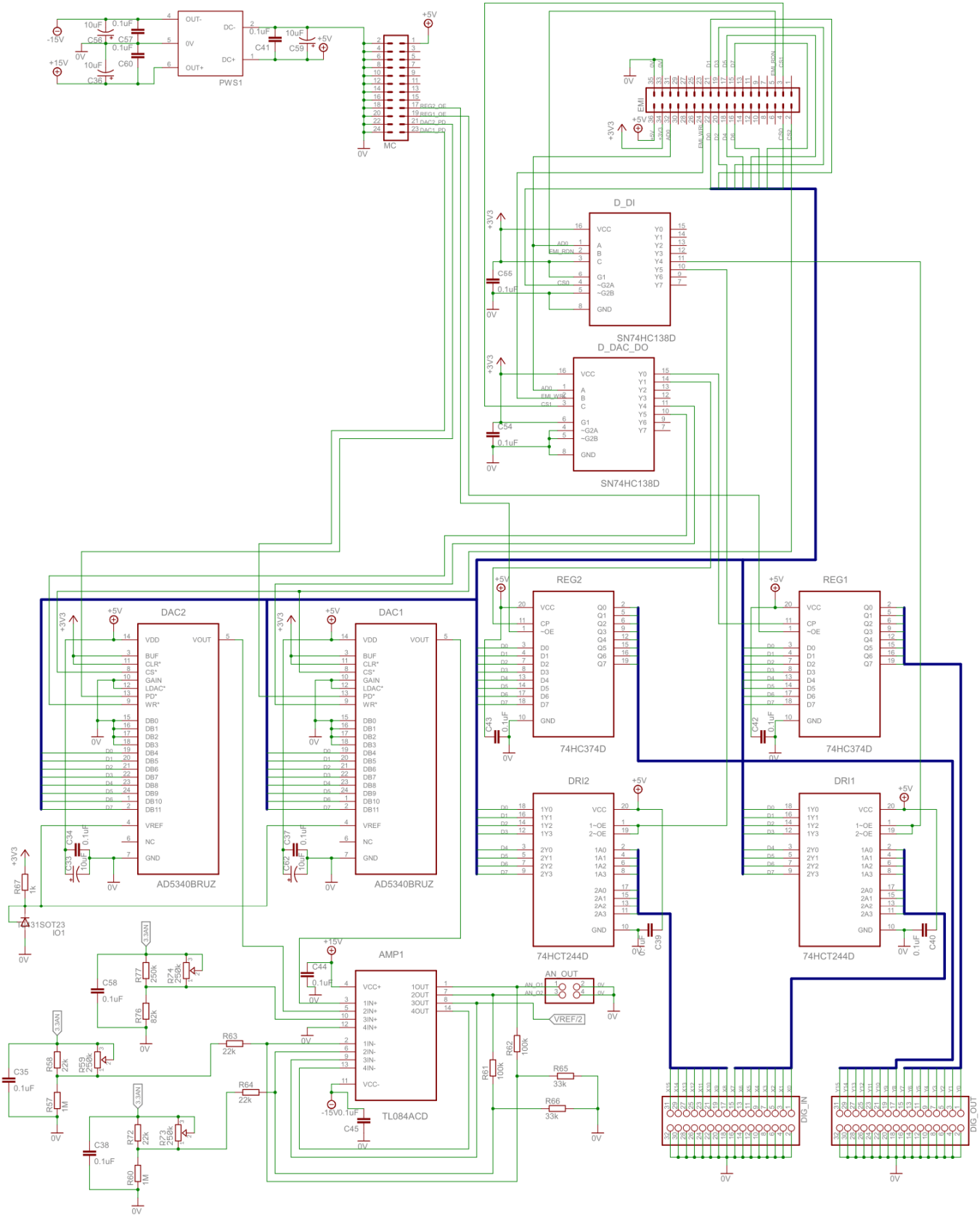
[9] Server Side Includes. WIKIMEDIA FOUNDATION, Inc. *Wikipedia* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://en.wikipedia.org/wiki/Server_Side_Includes

[10] Basic access authentication. WIKIMEDIA FOUNDATION, Inc. *Wikipedia* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://cs.wikipedia.org/wiki/Basic_access_authentication

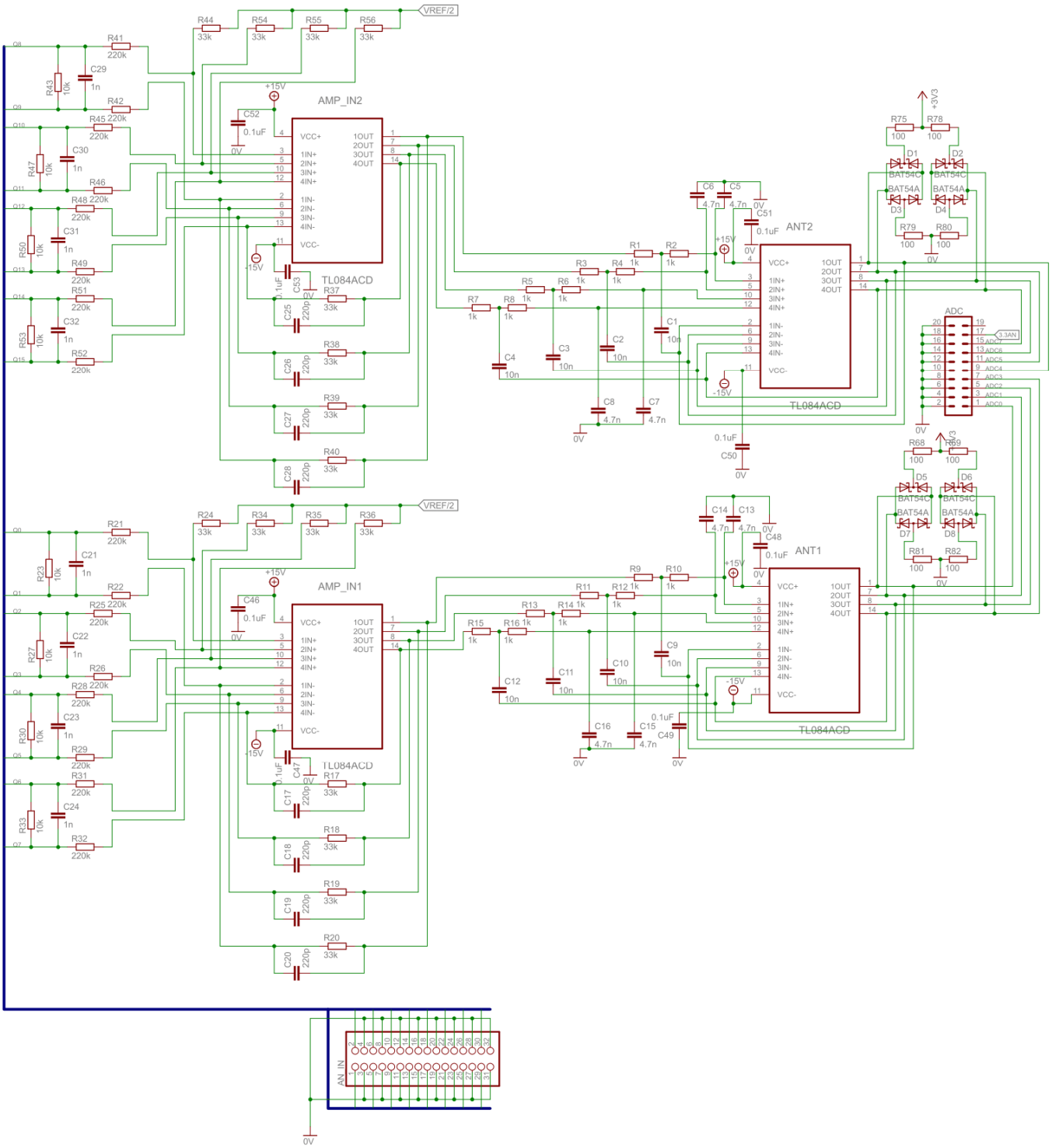
[11] ORACLE. 2015. *Java* [online]. [cit. 2015-05-05]. Dostupné z: http://www.java.com/

Přílohy

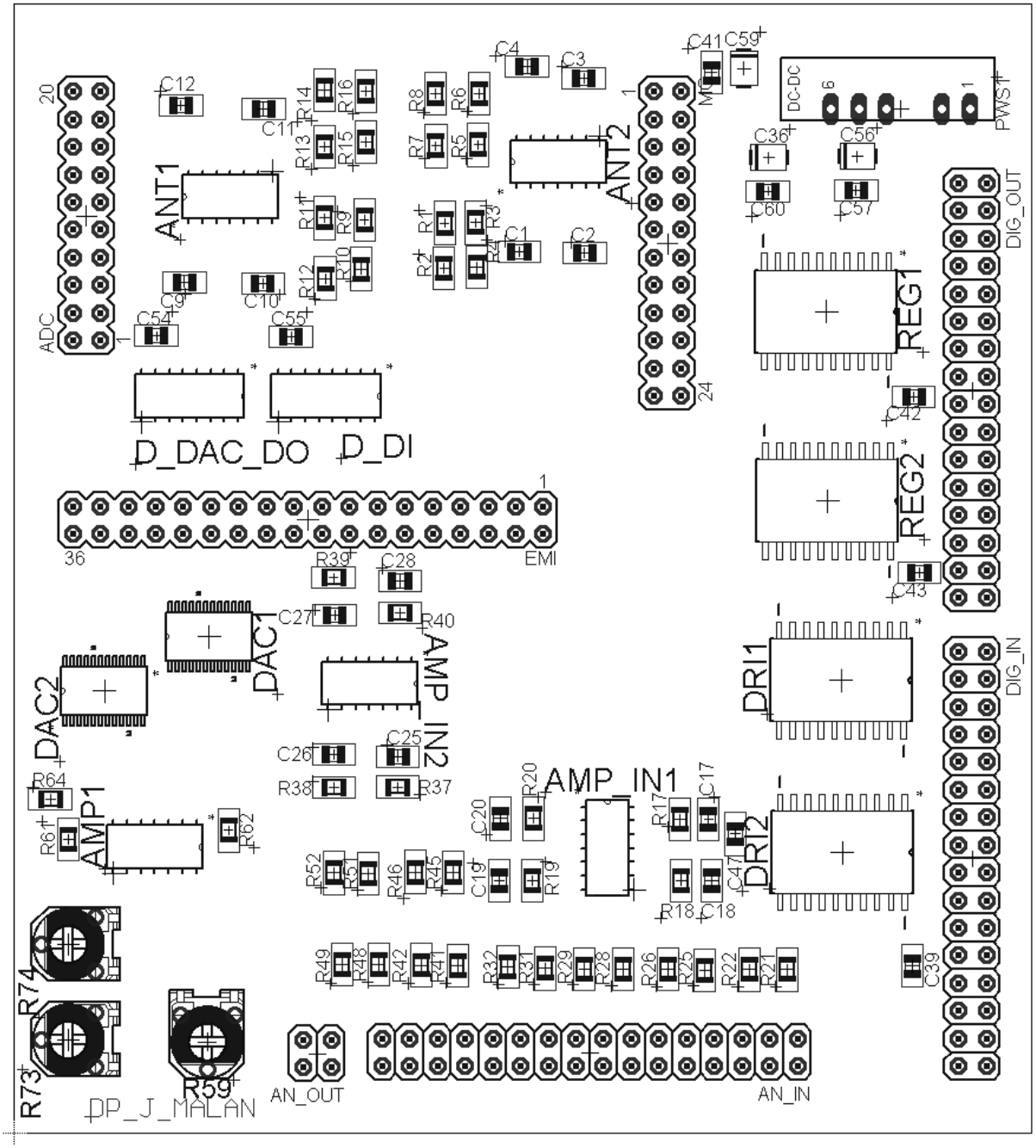
Příloha A1 – Schéma submodulu - Část 1. - Napěťový měnič, digitální vstupy a výstupy, analogové výstupy (D/A převodníky)



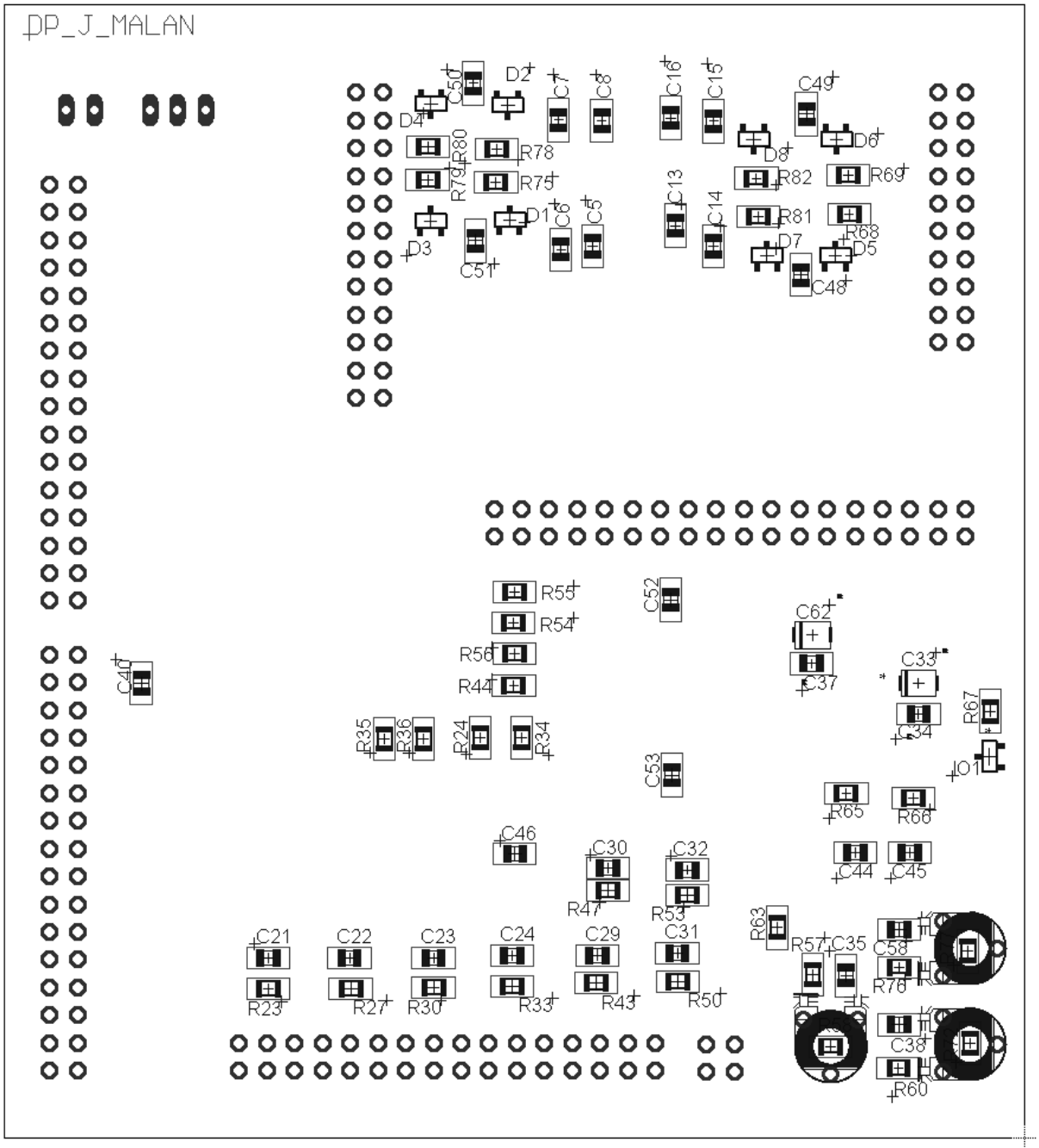
Příloha A2 – Schéma submodulu - Část 2. - Analogové vstupy



Příloha B1 – Rozvržení součástek na desce submodulu - Horní strana



Příloha B2 – Rozvržení součástek na desce submodulu - Spodní strana



Příloha C – Tabulka - seznam použitých součástek

Název	Hodnota	Pouzdro	Popis
ADC		MA10-2	Patice pinů 2,54 mm - typ samice
AMP1	TL084ACD	SOIC127P600X175-14N	Operační zesilovač (4 v pouzdře)
AMP_IN1	TL084ACD	SOIC127P600X175-14N	Operační zesilovač (4 v pouzdře)
AMP_IN2	TL084ACD	SOIC127P600X175-14N	Operační zesilovač (4 v pouzdře)
ANT1	TL084ACD	SOIC127P600X175-14N	Operační zesilovač (4 v pouzdře)
ANT2	TL084ACD	SOIC127P600X175-14N	Operační zesilovač (4 v pouzdře)
AN_IN		2X16	Patice pinů 2,54 mm - typ samec
AN_OUT		2X02	Patice pinů 2,54 mm - typ samec
C1	10n	C0805	Keramický kondenzátor SMD
C2	10n	C0805	Keramický kondenzátor SMD
C3	10n	C0805	Keramický kondenzátor SMD
C4	10n	C0805	Keramický kondenzátor SMD
C5	4.7n	C0805	Keramický kondenzátor SMD
C6	4.7n	C0805	Keramický kondenzátor SMD
C7	4.7n	C0805	Keramický kondenzátor SMD
C8	4.7n	C0805	Keramický kondenzátor SMD
C9	10n	C0805	Keramický kondenzátor SMD
C10	10n	C0805	Keramický kondenzátor SMD
C11	10n	C0805	Keramický kondenzátor SMD
C12	10n	C0805	Keramický kondenzátor SMD
C13	4.7n	C0805	Keramický kondenzátor SMD
C14	4.7n	C0805	Keramický kondenzátor SMD
C15	4.7n	C0805	Keramický kondenzátor SMD
C16	4.7n	C0805	Keramický kondenzátor SMD
C17	220p	C0805	Keramický kondenzátor SMD
C18	220p	C0805	Keramický kondenzátor SMD
C19	220p	C0805	Keramický kondenzátor SMD
C20	220p	C0805	Keramický kondenzátor SMD
C21	1n	C0805	Keramický kondenzátor SMD
C22	1n	C0805	Keramický kondenzátor SMD
C23	1n	C0805	Keramický kondenzátor SMD
C24	1n	C0805	Keramický kondenzátor SMD
C25	220p	C0805	Keramický kondenzátor SMD
C26	220p	C0805	Keramický kondenzátor SMD
C27	220p	C0805	Keramický kondenzátor SMD
C28	220p	C0805	Keramický kondenzátor SMD
C29	1n	C0805	Keramický kondenzátor SMD
C30	1n	C0805	Keramický kondenzátor SMD
C31	1n	C0805	Keramický kondenzátor SMD
C32	1n	C0805	Keramický kondenzátor SMD
C33	10uF	B/3528-21W	Tantalový kondenzátor SMD
C34	0.1uF	C0805	Keramický kondenzátor SMD
C35	0.1uF	C0805	Keramický kondenzátor SMD
C36	10uF	B/3528-21W	Tantalový kondenzátor SMD
C37	0.1uF	C0805	Keramický kondenzátor SMD
C38	0.1uF	C0805	Keramický kondenzátor SMD

C39	0.1uF	C0805	Keramický kondenzátor SMD
C40	0.1uF	C0805	Keramický kondenzátor SMD
C41	0.1uF	C0805	Keramický kondenzátor SMD
C42	0.1uF	C0805	Keramický kondenzátor SMD
C43	0.1uF	C0805	Keramický kondenzátor SMD
C44	0.1uF	C0805	Keramický kondenzátor SMD
C45	0.1uF	C0805	Keramický kondenzátor SMD
C46	0.1uF	C0805	Keramický kondenzátor SMD
C47	0.1uF	C0805	Keramický kondenzátor SMD
C48	0.1uF	C0805	Keramický kondenzátor SMD
C49	0.1uF	C0805	Keramický kondenzátor SMD
C50	0.1uF	C0805	Keramický kondenzátor SMD
C51	0.1uF	C0805	Keramický kondenzátor SMD
C52	0.1uF	C0805	Keramický kondenzátor SMD
C53	0.1uF	C0805	Keramický kondenzátor SMD
C54	0.1uF	C0805	Keramický kondenzátor SMD
C55	0.1uF	C0805	Keramický kondenzátor SMD
C56	10uF	B/3528-21W	Tantalový kondenzátor SMD
C57	0.1uF	C0805	Keramický kondenzátor SMD
C58	0.1uF	C0805	Keramický kondenzátor SMD
C59	10uF	B/3528-21W	Tantalový kondenzátor SMD
C60	0.1uF	C0805	Keramický kondenzátor SMD
C62	10uF	B/3528-21W	Tantalový kondenzátor SMD
D1	BAT54C	SOT23	Schottkyho dioda (2 v pouzdře)
D2	BAT54C	SOT23	Schottkyho dioda (2 v pouzdře)
D3	BAT54A	SOT23	Schottkyho dioda (2 v pouzdře)
D4	BAT54A	SOT23	Schottkyho dioda (2 v pouzdře)
D5	BAT54C	SOT23	Schottkyho dioda (2 v pouzdře)
D6	BAT54C	SOT23	Schottkyho dioda (2 v pouzdře)
D7	BAT54A	SOT23	Schottkyho dioda (2 v pouzdře)
D8	BAT54A	SOT23	Schottkyho dioda (2 v pouzdře)
DAC1	AD5340BRUZ	SOP65P640X120-24N	Integr. obvod D/A převodníku
DAC2	AD5340BRUZ	SOP65P640X120-24N	Integr. obvod D/A převodníku
DIG_IN		2X16	Patice pinů 2,54 mm - typ samec
DIG_OUT		2X16	Patice pinů 2,54 mm - typ samec
DRI1	74HCT244D	SOIC127P1032X265-20	Osmibitový 3-stavový budič
DRI2	74HCT244D	SOIC127P1032X265-20	Osmibitový 3-stavový budič
D_DAC_DO	SN74HC138D	SOIC127P600X175-16N	Dig. dekodér/demultiplexer
D_DI	SN74HC138D	SOIC127P600X175-16N	Dig. dekodér/demultiplexer
EMI		MA18-2	Patice pinů 2,54 mm - typ samice
IO1	TL431SOT23	SOT23	Napěťová reference
MC		MA12-2	Patice pinů 2,54 mm - typ samice
PWS1	TMA0515D	TMA-D	Měnič nap. - výstup -15 V a 15 V
R1	1k	M0805	Rezistor SMD
R2	1k	M0805	Rezistor SMD
R3	1k	M0805	Rezistor SMD
R4	1k	M0805	Rezistor SMD

R5	1k	M0805	Rezistor SMD
R6	1k	M0805	Rezistor SMD
R7	1k	M0805	Rezistor SMD
R8	1k	M0805	Rezistor SMD
R9	1k	M0805	Rezistor SMD
R10	1k	M0805	Rezistor SMD
R11	1k	M0805	Rezistor SMD
R12	1k	M0805	Rezistor SMD
R13	1k	M0805	Rezistor SMD
R14	1k	M0805	Rezistor SMD
R15	1k	M0805	Rezistor SMD
R16	1k	M0805	Rezistor SMD
R17	33k	M0805	Rezistor SMD
R18	33k	M0805	Rezistor SMD
R19	33k	M0805	Rezistor SMD
R20	33k	M0805	Rezistor SMD
R21	220k	M0805	Rezistor SMD
R22	220k	M0805	Rezistor SMD
R23	10k	M0805	Rezistor SMD
R24	33k	M0805	Rezistor SMD
R25	220k	M0805	Rezistor SMD
R26	220k	M0805	Rezistor SMD
R27	10k	M0805	Rezistor SMD
R28	220k	M0805	Rezistor SMD
R29	220k	M0805	Rezistor SMD
R30	10k	M0805	Rezistor SMD
R31	220k	M0805	Rezistor SMD
R32	220k	M0805	Rezistor SMD
R33	10k	M0805	Rezistor SMD
R34	33k	M0805	Rezistor SMD
R35	33k	M0805	Rezistor SMD
R36	33k	M0805	Rezistor SMD
R37	33k	M0805	Rezistor SMD
R38	33k	M0805	Rezistor SMD
R39	33k	M0805	Rezistor SMD
R40	33k	M0805	Rezistor SMD
R41	220k	M0805	Rezistor SMD
R42	220k	M0805	Rezistor SMD
R43	10k	M0805	Rezistor SMD
R44	33k	M0805	Rezistor SMD
R45	220k	M0805	Rezistor SMD
R46	220k	M0805	Rezistor SMD
R47	10k	M0805	Rezistor SMD
R48	220k	M0805	Rezistor SMD
R49	220k	M0805	Rezistor SMD
R50	10k	M0805	Rezistor SMD
R51	220k	M0805	Rezistor SMD
R52	220k	M0805	Rezistor SMD

R53	10k	M0805	Rezistor SMD
R54	33k	M0805	Rezistor SMD
R55	33k	M0805	Rezistor SMD
R56	33k	M0805	Rezistor SMD
R57	1M	M0805	Rezistor SMD
R58	22k	M0805	Rezistor SMD
R59	250k	TRIMMER_3306F	Trimr - nastavitelný rezistor
R60	1M	M0805	Rezistor SMD
R61	100k	M0805	Rezistor SMD
R62	100k	M0805	Rezistor SMD
R63	22k	M0805	Rezistor SMD
R64	22k	M0805	Rezistor SMD
R65	33k	M0805	Rezistor SMD
R66	33k	M0805	Rezistor SMD
R67	1k	M0805	Rezistor SMD
R68	100	M0805	Rezistor SMD
R69	100	M0805	Rezistor SMD
R72	22k	M0805	Rezistor SMD
R73	250k	TRIMMER_3306F	Trimr - nastavitelný rezistor
R74	250k	TRIMMER_3306F	Trimr - nastavitelný rezistor
R75	100	M0805	Rezistor SMD
R76	82k	M0805	Rezistor SMD
R77	250k	M0805	Rezistor SMD
R78	100	M0805	Rezistor SMD
R79	100	M0805	Rezistor SMD
R80	100	M0805	Rezistor SMD
R81	100	M0805	Rezistor SMD
R82	100	M0805	Rezistor SMD
REG1	74HC374D	SOIC127P1032X265-20	Registr 8-bitový
REG2	74HC374D	SOIC127P1032X265-20	Registr 8-bitový