Fakulta elektrotechnická
Katedra elektroniky a informačních technologií

Bakalářská práce

Kompaktní IO modul distribuovaného systému s komunikací MODBUS

Autor práce: Tomáš Viktora
Vedoucí práce: Ing. Kamil Kosturik, Ph.D.

Plzeň 2022

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta elektrotechnická

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Tomáš VIKTORA**
Osobní číslo:	**E18B0038P**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a telekomunikace**
Téma práce:	**Kompaktní IO modul distribuovaného systému s komunikací MODBUS**
Zadávající katedra:	**Katedra elektroniky a informačních technologií**

Zásady pro vypracování

1. Proveďte rozbor možných řešení IO ve stanici a jejich výhody/nevýhody
2. Uveďte základní princip komunikace MODBUS RTU pro sběrnici RS-485
3. Navrhněte kompaktní IO modul pro modulární tlačítkový blok IEC 60947-5-1, zaměřte se na snadnou montáž, výrobní náklady a diagnostikovatelnost
4. Realizujte funkční prototyp modulu jako po HW, tak SW stránce, připravte výrobní dokumentaci
5. Zhodnoťte navržené řešení a případné možnosti rozšíření

Rozsah bakalářské práce: **30 – 40**
Rozsah grafických prací: **dle doporučení vedoucího**
Forma zpracování bakalářské práce: **elektronická**

Seznam doporučené literatury:

Modbus The Manual: Definitive guide on Modbus, Rob Hulsebos, Mijnbestseller.nl (April 3, 2019), ISBN: 9463867643

Vedoucí bakalářské práce: **Ing. Kamil Kosturik, Ph.D.**
Katedra elektroniky a informačních technologií

Datum zadání bakalářské práce: **8. října 2021**
Termín odevzdání bakalářské práce: **26. května 2022**

Západočeská univerzita v Plzni
Fakulta elektrotechnická
L.S.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Doc. Ing. Jiří Hammerbauer, Ph.D.
vedoucí katedry

V Plzni dne 8. října 2021

Abstrakt

Práce za zabývá návrhem a tvorbou periferie se vstupy a výstupy. Řídícím prvkem této periferie je procesor ATmega 328P. Tuto periferii lze ovládat přes sběrnici RS-485 a po komunikačním protokolu Modbus. Práce popisuje postup návrhu při tvorbě tohoto modulu od návrhu schématu, přes návrh plošného spoje až po samotnou výrobu. Druhá část je věnována tvorbě a popisu softwarového řešení, které celý tento modul řídí. Výsledkem je funkční prototyp se kterým lze komunikovat a ovládat ho pomocí protokolu Modbus.

Klíčová slova

IO Modul, MODBUS, RS-485, tlačítkový blok, sériová komunikace, ATMega328P, Atmel

Abstract

Viktora, Tomáš. *Compact IO module of distributed system with MODBUS communication* [*Kompaktní IO modul distribuovaného systému s komunikací MODBUS*]. Pilsen, 2022. Bachelor thesis (in Czech). University of West Bohemia. Faculty of Electrical Engineering. Department of Electronics and Information Technologies. Supervisor: Kamil Kosturik

---

The work deals with the design and creation of module with inputs and outputs. The control element of this module is the processor ATmega 328P. This module can be controlled via the RS-485 bus and via the Modbus communication protocol. The work describes the design process in the creation of this module from the design of the scheme, through the design of the printed circuit board to the production itself. The second part deals with the creation and description of the software solution that controls this module. The result is a functional prototype with which it can be communicated and controlled using the Modbus protocol.

Keywords

IO Modul, MODBUS, RS-485, button block, serial communication, ATMega328P, Atmel

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 270 trestního zákona č. 40/2009 Sb.

Také prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

V Plzni dne 26. května 2022

Tomáš Viktora

.....

Podpis

Poděkování

Tato práce vznikla za podpory firmy FPC s.r.o

Obsah

Seznam obrázků

Seznam tabulek

Seznam symbolů a zkratek

DPS	Deska plošných spojů někdy označováno PCB
PCB	Printed Circuit Board
USB	Universal Serial Bus
PLC	Programmable Logic Controller
CRC	Cyclic Redundancy Check
MCU	Microcontroller, jednočipový počítač
R	Elektrický odpor [Ω]
V	Elektrické napětí [V]
L	Indukčnost [H]
VCC	Napájecí napětí
GND	Zemnící bod
UART	Sériová sběrnice
DC	Stejnosměrné napětí
SMD	Surface Mount Device, součástka pro povrchovou montáž plošných spojů
SSR	Solid-State Relay, polovodičový spínací prvek
UTP	Unshielded Twisted Pair, nestíněná kroucená dvojlinka

1

Úvod

Díky masové výrobě elektroniky vznikla potřeba tuto elektroniku efektivně testovat. Na základě toho vznikají sofistikované testovací stanice. Nedílnou součástí každé takové stanice jsou moduly se vstupy a výstupy. Tyto moduly slouží ke komunikaci mezi ovládacími prvky a řídícím programem.

Současných řešení je na trhu mnoho, bohužel přestávají být v některých projektech výhodná a z tohoto důvodu bylo potřeba vymyslet nové kompaktní a univerzální řešení. Cílem této práce je vymyslet řešení a následně navrhnout a vytvořit funkční prototyp.

Řešení má již od začátku stanovené základní parametry které vycházejí z požadavků zadavatele práce. První požadavkem je komunikace pro protokolu MODBUS, který je v průmyslu velmi rozšířený. Druhým je možnost připojení modulu jedním kabelem s průchozí sběrnicí. Posledním požadavkem je možnost umístění na tlačítkový modul IEC 60947-5-1.

Důležitou stránkou celého řešení je jak hardwarová stránka věci, kdy je potřeba držet se rozměrů které vycházejí z tlačítkového bloku IEC 60947-5-1, tak i softwarová stránka věci, kdy je potřeby vyřešit všechny úskalí komunikačního protokolu.

V neposlední řadě je potřeba rozebrat možnost dalších revizí zařízení, plány se zařízením do reálné aplikace a možnosti rozšíření.

2

Řešení IO v testovacích stanicích

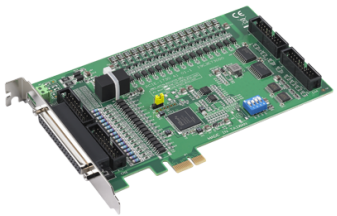
2.1 Princip IO

IO moduly slouží ke komunikaci vstupně/výstupních zařízení s řídícím procesorem. Vstupní zařízení přijímá signál například z tlačítek nebo senzorů a převádí ho na informaci zpracovatelnou procesorem. Naopak výstupní zařízení převádí informaci od procesoru na signál, kterým lze ovládat například spínací relé nebo signalizační diody. Nejčastěji se lze s IO moduly setkat v automatizaci, kde jsou nedílnou součástí každé aplikace.

V testovacích stanicích se vyskytují dvě varianty IO modulů. Tyto dvě varianty závisí na složitosti testovací stanice. V případě jednoduché stanice je IO přímo součástí testeru. U složitějších stanic je potřeba přivést IO do řídícího počítače. Moduly pro počítač se vyrábí v podobě PCI a PCIE karet nebo samostatného zařízení s komunikačním protokolem, nejčastěji ethernetem.

2.2 Současná IO řešení

2.2.1 Advantech PCIE-1730



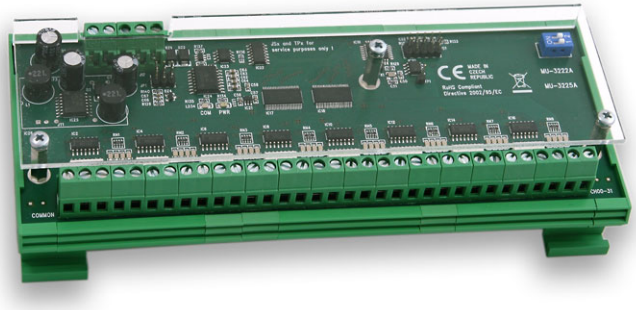
Obr. 2.1: Advantech PCIE-1730 [5]

Běžným řešením v menších stanicích je PCIE karta od firmy Advantech. Tato karta

má 16 vstupů a 16 výstupů. Hlavní výhodou této karty je její snadná připojitelnost a integrace. Logické úrovně vstupů mají hodnotu 0-3V pro logickou 0 a 10-30V pro logickou 1. Výstupem lze spínat napětí 5-40V. Všechny vstupy jsou izolované a mají izolační napětí 2500V. Hlavní nevýhodou je potřeba dalšího PCB, které bude umístěno v rozvaděči a bude sloužit jako interface, do které lze snadno připojit na jedné straně kabel z karty a na druhé vodiče od jednotlivých periferií. Připojovací kabel je poměrně masivní a zabírá nemalá množství místa.

2.2.2 MicroUnit

Toto řešení je v našich stanicích poměrně nové. Vzniklo z potřeby poměrně velkého množství IO a dlouhodobé nedostupnosti karet PCIE-1730. Základem je řídící modul, který je s počítačem propojen pomocí ethernetu. Celé řešení je provozováno na komunikačním protokolu MODBUS. Výrobce nabízí nezměrné množství modulů které lze k řídícímu modulu připojit na sběrnici RS-4855. V našem případě se nejčastěji používají dva typy modulů. A to MU-3251A, který obsahuje 32 výstupů a MU-3222A, který má stejný počet vstupů. Výhodou těchto zařízení je možnost umístění přímo do rozvaděče na DIN lištu, velká variabilita a možnost propojení pomocí ethernetu. Mezi nevýhody patří rozměry modulů, které hlavně v menší rozvaděčích zabírají nemalé množství místa.



Obr. 2.2: Modul Microunit MU-3222AV s 32 izolovaných digitálních vstupních kanálů [4]

3

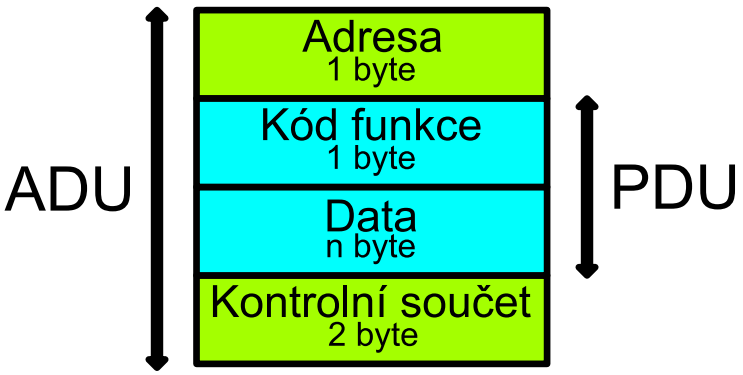
Komunikační protokol MODBUS

3.1 Základní informace

MODBUS je komunikační protokol vyvinutý v roce 1979 původně pro PLC. Stal se nejčastěji používaným protokolem v průmyslových aplikacích. Jedná se o protokol pracující na úrovni aplikační vrstvy ISO/OSI modelu, který využívá komunikaci master-slave. Jako komunikační médium se v dnešní nejčastěji používají sériové linky, například RS-232 nebo RS-485 nebo ethernet. Pro ethernet je potřeba do komunikace začlenit protokol TCP/IP.

3.2 Popis komunikace

Základem datového rámce protokolu je takzvané PDU (Protocol data unit), který je v závislosti na druhu komunikačního média začleněn do různé struktury ADU (Application data unit).



Obr. 3.1: Struktura rámce MODBUS zprávy

Prvním bytem je adresa slave zařízení. Může nabývat hodnot od 0 do 255, přičemž adresy 248-255 jsou vyhrazené. Druhý byte obsahuje kód funkce. Tyto funkce určují jak

a se kterými registry se má pracovat. Kódy jsou popsány v tabulce

Kód funkce	Popis funkce	Typ hodnoty	Typ přístupu
01(0x01)	Čtení DO	Diskrétní	Čtení
01(0x01)	Čtení DO	Diskrétní	Čtení
02(0x02)	Čtení DI	Diskrétní	Čtení
03(0x03)	Čtení AO	16 bitová	Čtení
04(0x04)	Čtení AI	16 bitová	Čtení
05(0x05)	Zápis do jednoho DO	Diskrétní	Zápis
06(0x06)	Zápis do jednoho AO	16 bitová	Zápis
15(0x0F)	Zápis do více DO	Diskrétní	Zápis
16(0x10)	Zápis do více AO	16 bitová	Zápis

Tab. 3.1: Přehled kódu pro čtení a zápis do registrů [3]

Po funkčním kódu následuje datový rámec. Zde jsou obsažena požadovaná data. Přenášená data by neměla překročit 252 bytů. V některých případech se na prvním místě datového paketu vyskytuje informace o počtu bytů. Tato situace je v případě kdy slave odesílá masteru hodnoty registrů. Nebo z masteru přijde požadavek na zápis do více registrů.

Poslední dva byty obsahují kontrolní součet vycházející z normy CRC-16-IBM.

MODBUS narozdíl od mnoho komunikačních protokolů nemá ukončovací znak. Ukončení komunikace je zajištěno ukončovací mezerou o dělce 3,5 znaku.

V případě že si slave zařízení nedokáže poradit s přijatými daty odesílá zpět chybové kódy, takovýto chybový kód má na místě funkčního bytu přijatou fukci zvětšenou o 0x80. Například 0x82 pro chybu při čtení diditální vstupů. Poté následuje příslušný chybový kód.

Chybový kód	Popis chyby
01	Přijatý kód funkce nelze zpracovat
02	Datová adresa uvedená v požadavku není dostupná
03	Hodnota obsažená v datovém poli dotazu je neplatná
04	Vyskytla se neodstranitelná chyba při pokusu zařízení slave o provedení požadovaného úkonu
05	Zařízení slave přijalo požadavek a zpracovává ho, ale trvá to dlouhou dobu. Tato odpověď brání hostiteli vygenerovat chybu časového limitu
06	Zařízení slave je zaneprázdněno zpracováváním příkazu. Master musí zprávu zopakovat později, až bude zařízení slave volné
07	Zařízení slave nemůže vykonat programovou funkci uvedenou v požadavku. Tento kód se vrací při neúspěšném požadavku programu prostřednictvím funkcí s čísly 13 nebo 14. Master si musí od zařízení slave vyžádat diagnostické informace nebo informace o chybě
08	Zařízení slave zjistilo chybu parity při čtení rozšířené paměti. Master může požadavek zopakovat, ale obvykle jsou v takových případech potřeba opravy

Tab. 3.2: Přehled chybových kódů [3]

4

Sériová sběrnice RS-485

RS-485 je definice standartu sloužící k přenosu po sériové lince. Standart byl zveřejněn a je spravován dvěma americkými společnostmi. Telecommunications Industry Association a Electronic Industries Alliance (TIA/EIA). Proto bývá někdy tento standart označována TIA-485 nebo EIA-485. Největší výhodou sběrnice je velká vzdálenost přenosu bez opakovače a vysoká odolnost vůči rušení. Z tohoto důvodu se velmi často používá pro komunikaci v průmyslových aplikacích.

4.1 Popis a parametry RS-485

RS-485 využívá komunikace po dvou vodičích, kde je logická úroveň reprezentována rozdílovými napětími. Tyto vodiče jsou označeny A a B, přičemž A reprezentuje nižší napěťovou úroveň a B vyšší napěťovou úroveň. Jelikož se jedná o rozdílové úrovně, tak i v klidovém stavu je na signálech A a B napětí, typicky $\pm 200mV$. Při provozu bývá rozdílové napětí okolo $\pm 2V$, velikost záleží na délce vedení a velikosti terminačních odporů. Jelikož je toto napětí potřeba aby přijímač mohl pracovat diferenciální, je nutné dodržet dvě důležité podmínky:

- Napěťový rozdíl mezi zemí přijímače a zemí vysílače nesmí být větší než $7V$
- Linka RS-485 musí být pro zajištění všech výhod sběrnice galvanicky oddělena

Na sběrnci lze za běžných okolností připojit až 32 zařízení, při použití opakovačů lze toto číslo zvýšit až na 256. Důležitým prvkem je terminační rezistor, který by měl odpovídat impedanci vedení kroucené dvoulinky, což je hodnota 100 - 120 ohm.

4.2 MODBUS na sběrnici RS-485

Vzhledem ke svým paramterům a odolnosti je tato sběrnice hojně využívána pro komunikační protokol MODBUS.

5

Důvod návrhu kompaktního IO modulu

V současné době se v testerech používají moduly umístěné v rozvaděčích, ke kterým je potřeba při rozsáhlých projektech přivádět obrovské množství vodičů. Toto velké množství vodičů není estetické a zároveň snižuje přehlednost pro případné dohledávání chyb. Na základě poznatků a zkušeností vznikl nápad na vytvoření malých modulů, které budou umístěné na periferii a budou připojené na společnou sběrnici. Tím odpadne velké množství vodičů, jelikož bude stačit pouze jeden kabel, natažený od modulu k modulu.

6

Postup při návrhu DPS a výběr vhodných komponent

6.1 Výběr hlavních komponent

Před samotným kreslením schématu je potřeba vybrat hlavní komponenty celého řešení. Nejprve bylo potřeba vybrat vhodný řídící procesor. V původním návrhu bylo počítáno s moderním MCU ATmega3208 od společnosti Microchip. Bohužel v současné situaci, kdy je velký nedostatek polovodičů, je tento procesor nedostupný. Z tohoto důvodu jsem se rozhodl pro starší a v mnoho aplikacích používané MCU ATmega328P.

Další velmi důležitou částí řešení je napájení. Jelikož bude celé zařízení z vnějšku napájeno $24V$ a vnitřní komponenty pracují na $5V$, je potřeba se rozhodnout jakým způsobem napětí snížit. Existují dva způsoby pro snížení napětí. První je napěťový stabilizátor a druhým je DC/DC měnič. Napěťový stabilizátor je velmi jednoduché a spolehlivé řešení. Jeho největší nevýhodou je, že při takhle velkém snižování napětí se úbytek napětí mění na teplo. Jelikož je při komunikaci počítáno s odběrem okolo 50mA, docházelo by k přeměně téměř 1W na teplo. Toto řešení není z důvodu úspory energie a vyššího tepelného namáhání modulu vhodné. Z výše zmíněných důvodů padlo rozhodnutí použít DC/DC měnič. Toto řešení je poněkud náročnější a při návrhu DPS je potřeba počítat s působením mnoha parazitních vlivů. Velkou výhodou je, že eliminuje nevýhody stabilizátoru napětí. Na trhu je velké množství měničů, já jsem se rozhodl pro dříve několikrát použitý a vyzkoušený měnič MCP16311-E/MS.

Poslední důležitou komponentou je převodník na sériovou linku RS-485. I zde je na trhu nezměrné množství převodníků, pracující na různých napětích, umožňující různé rychlosti přenosu a s různou velikostí pouzdra. Jelikož u tohoto modulu je hlavní parametrem rozměr, rozhodl jsem se pro pouzdro MSOP8. V tomto pouzdře nejlépe vycházel převodník ADM3065EARMZ. Výhodou tohoto převodníku je nízký odběr a vysoké přenosové rychlosti. Největší nevýhodou je cena a jeho trvalá nedostupnost v poslední době. Pro použití tohoto projektu byl zapůjčen z jiného projektu.

Další komponenty jako oddělovací optočleny, pasivní součástky a ovládací prvky, byly vybírány v pouzdrech a s rozložením vývodů tak, aby se i při současné situaci na trhu součástek daly snadno vyhledat náhrady.

6.2 Řešení konektivity

Nedílnou součástí řešení je konektivita. Jelikož modul má být snadno odpojitelný a připojitelný, rozhodl jsem se pro uchycení použít kolíkové lišty. Díky tomu je snadné připravit připojení k tlačítkovému modulu (viz obrázek xx).

Netěžší části bylo vymyslet připojení napájení a komunikačních signálů. Od začátku bylo jasné, že v jednom kabelu musí být umístěno nejméně 6 vodičů. Dva napájecí vodiče, dva pro Modbus komunikaci, jeden vstupní řídící signál a jeden výstupní řídící signál. V prvních fázích se nejvýhodněji jevilo použít konektor RJ-12. Největší výhodou byla jeho snadná výroba a snadné připojení. Nakonec jsem se však rozhodl použít sice větší ale mnohem více rozšířený konektor RJ-45. Zde je výhodou možnost použití běžného síťového UTP kabelu a možnost koupit již vyrobený kabel.

Poslední připojení na desce, které bylo potřeba vyřešit, je konektor pro rozšiřující moduly. Zde byla volby poměrně jednoduchá. Jelikož komunikace s rozšiřujícími moduly poběží po protokolu I2C, padla volba na propojení plochými kabely a k ním odpovídající konektory (viz obrázek xx).

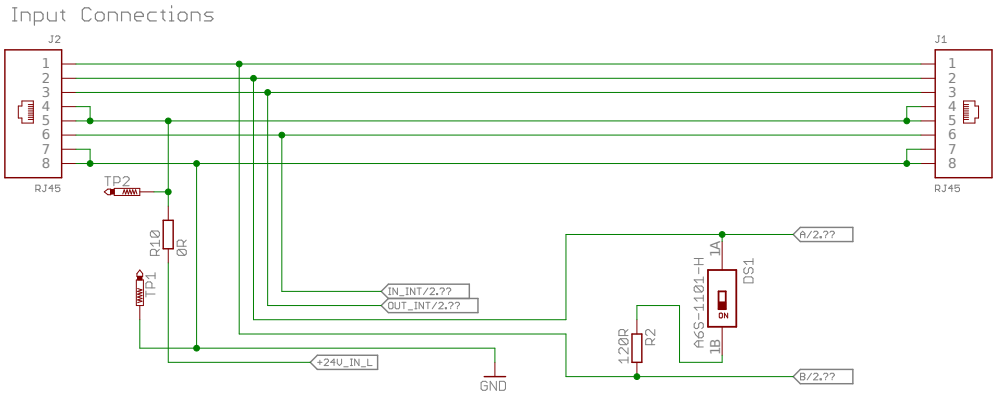
6.3 Schéma zapojení

Pro návrh elektrického schéma zapojení a desky plošného spoje jsem zvolil návrhový software Eagle. Software jsem si vybral, jelikož jsem s ním mnohokrát před tím pracoval a ve firmě FPC, která celou práci zaštiťuje, je dostupná licence. Kreslení schématu je nejprve potřeba rozdělit si do několika bloků pro větší přehlednost. Jednotlivé bloky jsou odděleny a popsány příslušnými nadpisy.

6.3.1 Popis funkce zapojení

První blok první strany schématu obsahuje připojovací konektory, které jsou sběrnicově propojeny a paralelně jsou k nim připojeny další bloky modulu. Tento blok obsahuje 3 důležité části. První jsou RJ-45 kontory, který slouží k připojení modulu, druhou částí je rezistor $R10$, které plní funkci pojistky a je díky němu možné odpojit celý modul od napájení a v neposlední řadě je to DIP switch DS1, který slouží k připojení terminačního odporu posledního modulu na sběrnici.

Druhý blok na této stránce je věnován napájení. Nachází se zde doporučené katalogové schéma pro DC/DC měnič MCP16311 doplněné o ochranou diodu. Zde je potřeba pečlivě prolistovat katalog a vyhledat vzorec pro výpočet indukčnosti výstupní cívky a



Obr. 6.1: Zapojení vstupních signálů

zpětnovazebních rezistorů. Pro výpočet indukčnosti výstupní cívky lze použít jednoduchý vzorec [6.1] nebo katalogovou tabulku [6.1] pro běžně používané hodnoty napětí. Jelikož je na výstupu měniče požadováno napětí $5V$, lze z tabulky snadno vyčíst, že bude potřeba cívky s indukčností $22\mu H$. Kromě indukčnosti je dalším velmi důležitým parametrem cívky maximální zátěžový proud. Jelikož se nepočítá s velkými odběry proudu, byla zvolena cívka s zátěžovým proudem $300mA$. V katalogu se nacházejí také hodnoty zpětnovazebních rezistorů pro nominální hodnoty napětí. V tomto případě bude snadnější použít vzorec[6.2], který dám dovoluje využít rezistory z běžně dostupných řad. V prvé řadě je potřeba vycházet z známých hodnot, kterými jsou V_{OUT} a V_{FB}. V_{OUT} je jak již bylo zmíněno $5V$ a V_{FB} je katalogem daná hodnota $0.8V$. Následně je třeba zvolit vhodnou hodnotu R_{BOT}. Lze zvolit hodnotu například 10Ω, přičemž výsledek bude desetinásobek poměru V_{OUT} a V_{FB} zmenšený o jedna nebo odpor mírně snížit na některou ze standartně dostupných hodnot a spočítat k ní odpovídající hodnotu R_{TOP}. Při zkoumání mi nejlépe vyšlo použít $R_{BOT} = 6.2k$ a $R_{TOP} = 33k$. Při těchto hodnotách by v ideálních podmínkách mělo výstupní napětí nabývat hodnoty $5.058V$. Poslední důležitou součástkou, které je potřeba věnovat při návrhu měniče náležitou pozornost jsou vstupní kondenzátory. Zde je třeba zvolit kvalitní, keramické a napěťově co nejméně závislé kondenzátory. Vzhledem k frekvenci, na které měnič pracuje, je důležité, aby kvalitně kompenzovali napěťové špičky. V případě špatně zvolených kondenzátorů by mohlo dojít ke špatné funkci měniče.[10]

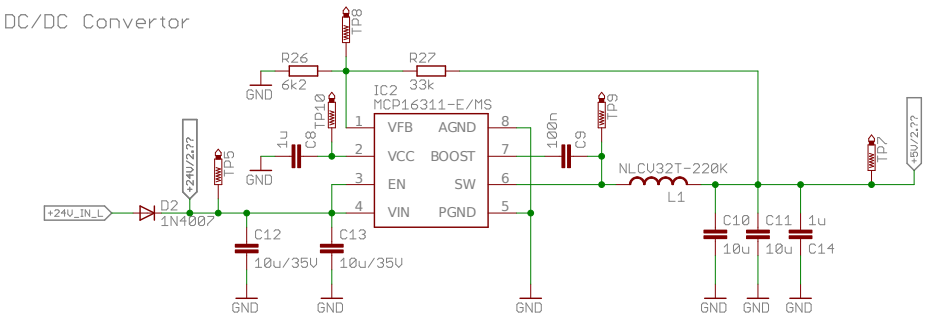
$$K = \frac{V_{OUT}}{L} \qquad [-] \tag{6.1}$$

$$R_{TOP} = R_{BOT} \times (\frac{V_{OUT}}{V_{FB}} - 1) \qquad [\Omega] \tag{6.2}$$

Poslední blok první strany je věnován řešení vstupů a výstupů modulu. Pro vstupy jsou použity optočleny ACPL-244-500ES, jejichž největší výhodou je možnost použít libovolnou logiku a umístění čtyřech optočlenů v jednom pouzdře. Pro omezení proudu do optočlenů jsou použity rezistory s nominální hodnotou $10k$. Pro zajištění kvalitního rozpoznání aktivního vstupu jsou výstupní piny optočlenů, které jsou zavedeny přímo na

V_{OUT}	K	$L_{STANDART}$
$2V$	0.20	$10\mu H$
$3.3V$	0.22	$15\mu H$
$5V$	0.23	$22\mu H$
$12V$	0.21	$56\mu H$
$15V$	0.22	$68\mu H$

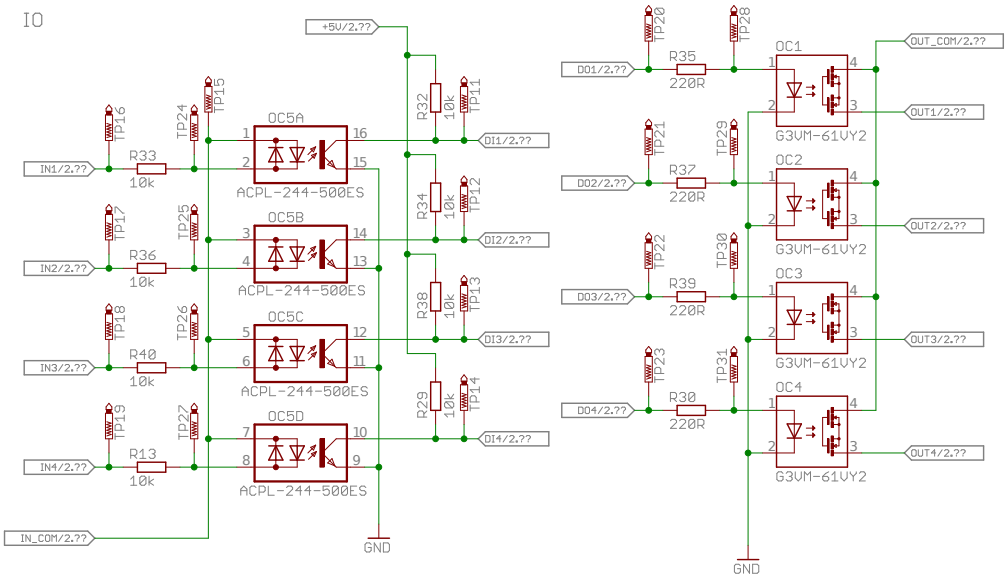
Tab. 6.1: Doporučené hodnoty indukčností pro nominální hodnoty napětí měniče MCP16311 [10]



Obr. 6.2: Zapojení DC/DC měniče

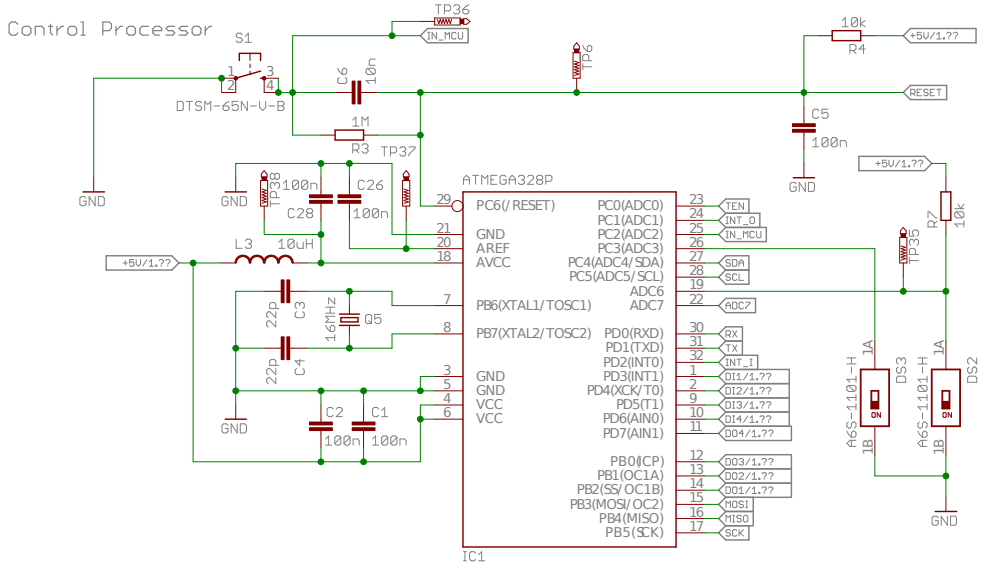
porty procesoru opatřeny pull-up rezistory s nominální hodnotou $10k$. Jelikož u výstupních pinů je potřeba počítat s většími hodnotami proudů, rozhodl jsem se pro použití SSR relé G3VM61VY2, které stejně jako vstupní optočleny umožňuje použití libovolné logiky. Výstupy z procesoru jsou opatřeny rezistory pro omezení proudu s nominální hodnotou $220R$.

První polovina druhé strany je věnována hlavnímu mozku celého zařízení. A to řídícímu MCU a součástkám umožňujícím jeho správný chod. Nejprve je potřeba vyřešit napájení. Na piny VCC a GND je potřeba připojit napájecí napětí a filtrační kondenzátory, které slouží ke kompenzaci vysokofrekvenčních kmitů. Kondenzátory je zde potřeba umístit dva. A to z důvodu, že u některých MCU bývá umístění napájení na dvou různých stranách procesoru a je potřeba zajistit, že filtrační kondenzátor bude co nejblíže pinům MCU. V tomto případě jsou piny těsně vedle sebe, tak lze využít dvou kondenzátoru k umístění každého na jednu stranu PCB. Dále je potřeba umístit filtrační kondenzátor na pin AREF, sloužící jako referenční hodnota pro A/D převodník a LC článek na vstup napájení A/D převodníku pro potlačení šumu. Dalším prvkem je tlačítko pro reset procesoru. Pro správnou funkci je zde umístěn pull-up rezistor, filtrační kondenzátor a RC člen pro správnou reakci tlačítka. Z tlačítka je pro informaci o stisknutí vyveden signál na interrupt pin procesoru. Při návrhu zbyly na procesoru dva nevyužité piny. Pro správnou funkci zařízení by se neměli v zapojení nevyužité piny procesoru vyskytovat. První variantou ošetření je piny připojit přes pull-up rezistor na napájení nebo připojit na zem. Já jsem se rozhodl na tyto piny zapojit přepínače, které mohou být v budoucnu využity pro



Obr. 6.3: Zapojení IO rozhraní

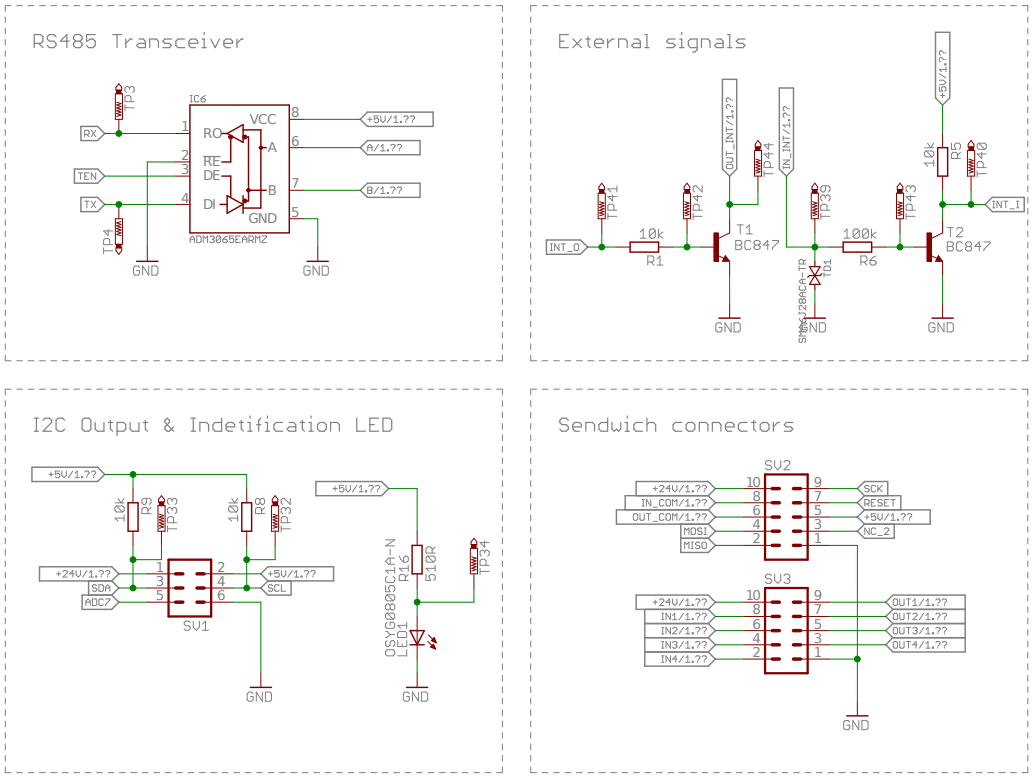
specifické nastavení modulu.



Obr. 6.4: Zapojení MCU

Druhá polovina druhé strany je rozdělena na čtyři bloky. V prvním bloku je převodník na sériovou sběrnici na RS-485. Do převodníku je přiveden UART z procesoru a na výstupu jsou diferenční signály A a B. Převodním má další dva ovládací vstupy. Vstupy jsou označeny $\bar{RE}$, který slouží k zapnutí přijímání a DE, který slouží k zapnutí vysílání. Jelikož je potřeba aby zařízení stále přijímalo, je vstup $\bar{RE}$ připojen na zem. Vstup DE je přiveden na vstup procesoru, aby se dal zapnout pouze v případě, kdy je třeba odesílat data. Druhý blok řeší připojení externích signálů na piny procesoru. Externí signály jsou ovládány dvojicí NPN tranzistorů. Externí vstup je ošetřen TVS diodou pro případ, že by se na vstupním signálu objevilo vyšší než přípustné napětí. Poslední dva bloky obsahují

konektivitu.

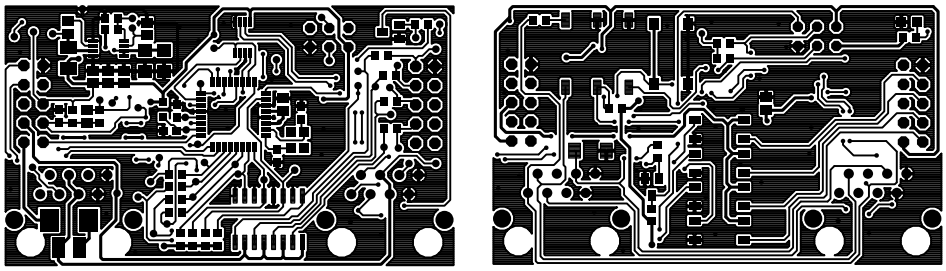


Obr. 6.5: Zapojení RS-485 převodníku, ovládání externích signálů a konektorů

6.4 Návrh DPS

Před návrhem je potřeba si rozmyslet rozměry DPS. Vzhledem k tomu, že plánované umístění modulu je na zadní stranu tlačítkového modulu IEC 60947-5-1, je potřeba rozměry přizpůsobit. Dle výkresu tlačítkového modulu jsem zvolil rozměry $35mm \times 60mm$.

Samotný návrh začneme rozmístěním součástek s pevným umístěním. Nejprve jsem tedy rozmístil připojovací konektory. Následně jsem propojil všechny propojení, které jsou sběrnicové a vedou přes celou desku. Následně jsem doprostřed spodní strany umístil řídící procesor a na levou horní stranu ovládací prvky. Další důležitou části je DC/DC měnič. Zde je potřeba dodržet co nejtěsnější umístění DC/DC měnič. Zde je potřeba dodržet co nejtěsnější umístění součástek okolo měniče. Lze vycházet z doporučeného zapojené v datasheetu, ale při dodržení všech pravidel lze navrhnout vlastní zapojení. I přestože deska bude mít rozlitou zem, pro zajištění co nejkratšího propojení země okolo měniče je potřeba zem zapojit ručně. Pokud by nebyla tyto pravidla dodrženo, mohlo by dojít k nefunkčnosti měniče. Další součástky, pro které je důležité jejich umístění jsou filtrační kondenzátory a oscilátor okolo řídícího procesoru. I zde je důležité umístění co nejblíže k pinům procesoru. Poté rozmístíme ostatní součástky a provedeme propojení. Nakonec vytvoříme ve volných plochách rozlitou měď a zajistíme spojení všech míst se zemí.



(a) Spodní strana vodivé vrstvy (b) Horní strana vodivé vrstvy

Obr. 6.6: Vodivé vrstvy navržené desky plošných spojů

Po navržení desky se v softwaru Eagle provede takzvaný DRC (Design Rule Check). Zde jsou nastaveny parametry výrobce DPS, které musí být dodrženy.

Nakonec je potřeba provést export dat potřebných pro výrobu. Tyto data se označují gerbery. Obsahují všechny informace potřebné pro výrobu desky.

7

Realizace a výroba prototypu

Dříve by bylo výhodnější vyrobit si plošný spoj sám. Při dnešních cenách a kvalitě profesionální výroby, je výhodnější nechat si DPS vyrobit. Na trhu je spoustu výrobců, kteří se tímto odvětvím zabývají.

7.1 Výběr výrobce PCB

Pokud budeme hledat výrobce v České republice je to například firma PragoBoard. Zde je velká výhoda ceny při objednání velkého množství kusů. Pro prototypovou výrobu se více vyplatí vybrat si některého výrobce z Asie. Například firma JLPCB nabízí velmi kvalitní desky za nízké ceny. Vždy je ale potřeba objednat 5 a více kusů. Cena se však pohybuje okolo 5 USD za 5 kusů do rozměru desky 100mm x 100mm. Jelikož mám již s tímto výrobcem bohaté zkušenosti byl nakonec jasnou volbou.

JLPCB má velmi pěkné prostředí pro nahrání dat a vytvoření náhledu. Je potřeba pouze dodržet správné pojmenovaní gerber souborů. Pokud je v podkladech nějaká chyba, kontrolní software na ni upozorní. Pokud je vše v pořádku, zobrazí se náhled desky a uživatel si může vybrat ze spousty možností, jak má finální výrobek vypadat. Například barva sítotisku, zda má být deska v panelu, tloušťka desky a mnoho dalších parametrů. Dodání desky trvá přibližně jeden týden a doprava je zdarma.

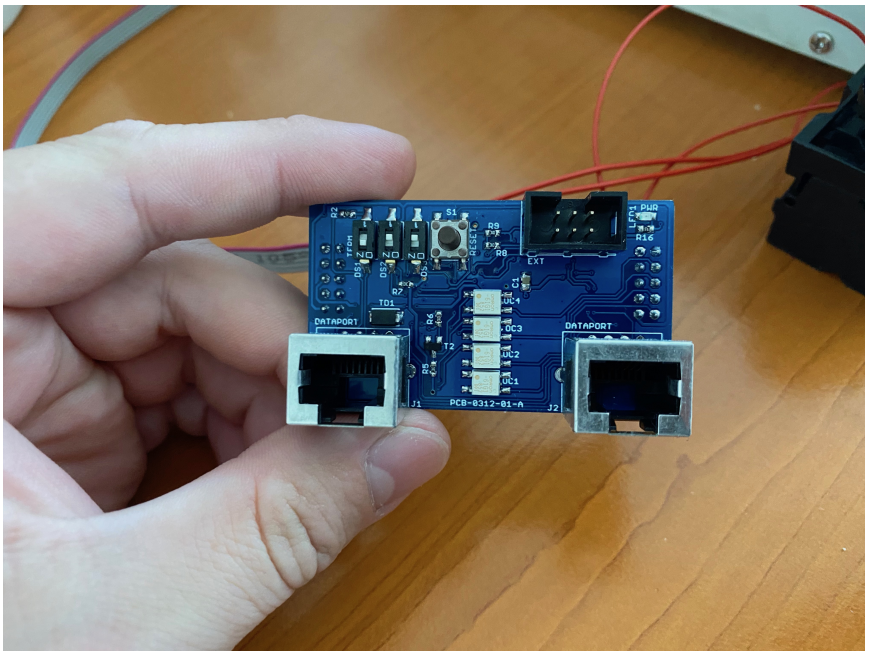
7.2 Nákup komponent

Nákup komponent je v dnešní době velmi náročný úkol. Nedostatek polovodičů na trhu způsobuje velmi dlouhé dodací lhůtu u některých součástek zejména u mikročipů. V mém případě byl největší problém u MCU ATmega328p. Dodací termín tohoto MCU se pohybuje od 50 do 53 týdnů. Výhodou tohoto procesoru je využit ve velkém množství aplikací, zejména na deskách Ardunino. Jelikož nemalé množství těchto desek vlastním, nebyl problém si z jedné desky tento procesor vypůjčit. Dalším velkým problémem byl měnič MCP. Tento měnič je velmi hojně používaný, tudíž velmi těžko dostupný a náhrady se stejným

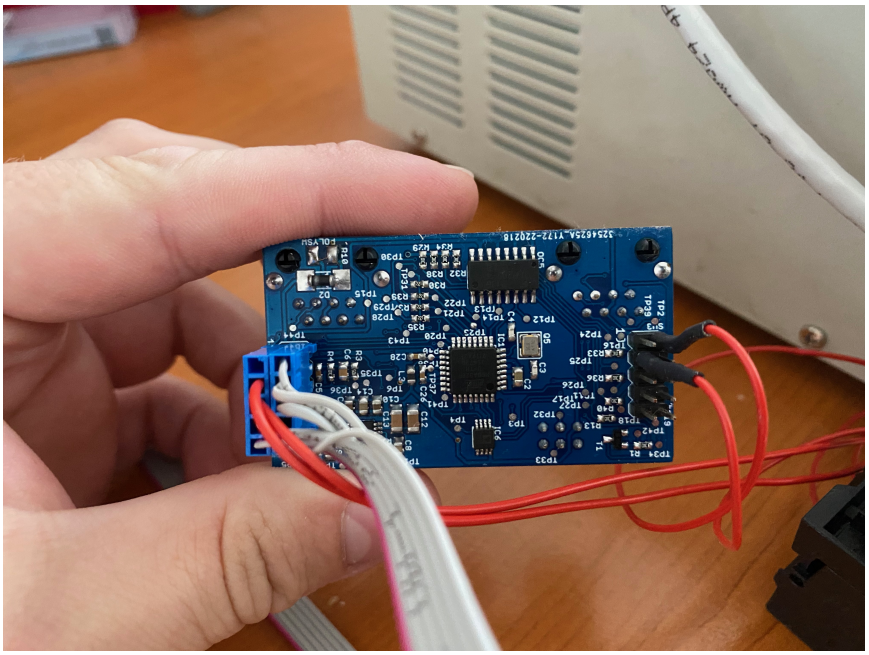
rozložením pinů a stejnými parametry neexistují nebo také nejsou dostupné. Zde bylo velkou výhodou, že tento měnič je hojně používaných ve starších revizích řídících desek používaných firmou FPC a bylo možnou si z jedné z desek tento měnič vypůjčit. Poslední nedostupnou součástkou je RS-485, zde byla chyba použít ADM3065EARMZ, který je také velmi těžko dostupný. Neštěstí se mi zde podařilo najít adekvátní náhradu. Pro oddělovací optočleny a SSR relé jsem taká musel vyhledávat náhrady. Zde je však situace poměrně jednodušší, jelikož je na trhu dostupných spoustu součástek s podobnými parametry a rozložením pinů. Ostatní, zejména pasivní součástky jsou i v dnešní době dobře dostupné a není problém je běžně koupit.

7.3 Osazení komponent

Po doručení všech potřebných komponent a desky plošných spojů bylo možné přistoupit k osazení desky součástkami. Jelikož téměř všechny komponenty jsou SMD součástky je potřeba mít pří pájení dostatek zkušeností. Pro zajištění kvalitního výsledku jsem se rozhodl využít toho, že projekt dělám za podpory firma FPC a desku si nechat osadit lidmi co s tím mají dostatek zkušeností. Výsledkem byla profesionálně osazená deska, u které se dalo snadno přistoupit k oživení. Jelikož nevíme, jestli měnič bude fungovat správně, je součástky, které jsou z měniče napájeny, lepší neosazovat a nejprve ověřit funkčnost měniče. Pokud bychom tento krok vynechali, mohlo by dojít k nevratnému poškození některých polovodičových součástek.



(a) Horní strana modulu



(b) Spodní strana modulu

Obr. 7.1: Vzhled modulu po osazení

8

Oživení prototypu

Pro oživení a následné programování bylo potřeba vyrobit několik kabelů. Prvním byl napájecí a komunikační UTP kabel s nakrimpovaným RJ-45 konektorem na jedné straně a volným konci, které se dají dále připojit na druhé straně. Druhý kabelem je plochý kabel s konektorem, který je na jedné straně možné připojit na kolíkovou lištu a na druhé straně do AVR programátoru. Poslední skupinou jsou dráty s konektory, které jsou zapojeny na IO a je možné díky nim sledovat chování na těchto pinech.

8.1 Kontrola napájení

Před připojením modulu k napájení je potřeba zkontrolovat, jestli mezi napájecími větvemi nebo zemí nejsou nějaké zkraty, které by mohli způsobit poškození součástek. Pokud tato kontrola projde v pořádku je možné připojit zařízení k napájení. Následně provedeme změření výstupního napětí na měniči. Zjistil jsem, že oproti předpokládaným 5.058V je napájecí napětí mírně nižší a to 4,958V, což je po započítání možných výrobních tolerancí součástek a parazitních vlivů velmi přijatelná hodnota. Po tomto testu bylo možné přistoupit a osazení desky polovodičovými součástkami napájenými z této větve. V tomto případě se jednalo a mikročip ATmega328P a RS-485 převodník. Zařízení jsem opět proměřil na zkraty a jelikož vše prošlo v pořádku bylo možné desku připojit k napájecímu napětí.

8.2 Vývojové prostředí

Programovaní procesorů AVR je možné dvěma základními způsoby. První možností je psát v přímo v nízkoúrovňovém programovacím jazyce jakým je Assembler. Programování v tomto jazyce umožňuje přímý přístup k periferiím, tudíž je možné docílit velké optimalizace výsledné aplikace. Pro menší aplikace, které musí zabírat co nejméně místa je velmi výhodné i přes náročnost programovaní tento jazyk použít. Pro rozsáhlejší projekty je však lepší využít některé z vývojových prostředí, ve kterém je možné programovat

v jazyce C. Pro programování mikroprocesorů ATmega se nejvíce hodí Atmel (nyní Microchip) Studio. Toto prostředí je založeno na Visual Studiu od společnosti Microsoft, čímž přebírá spoustu užitečných funkcí, které usnadňují tvorbu aplikace. Dále obsahuje všechny potřebné knihovny, které slouží k ovládání periferií mikrokontroleru a v neposlední ředě má integrované drivery umožňující programovaní.

8.3 Připojení desky k programátoru a první testovací aplikace

Při vytváření nového projektu je nejprve potřeba zvolit správný typ procesoru. Po vytvoření projektu je potřeba provést několik nastavení. Hlavním je nastavit správnou frekvenci oscilátoru, v našem případě 16MHz. Poté připojíme desku k AVR programátoru. V mém případě se jedná o programátor JTAGICE3. Tento test prošel v pořádku a díky tomu bylo možné přistoupit k prvním jednoduchým aplikacím, kterými jsem otestoval správnou funkčnost modulu.

Pro test výstupů jsem použil jednoduchou funkci, která bude blikat LED diodou připojenou na výstup mikročipu. Tuto funkci vytvoříme pomocí jednoduchých příkazů *write* a *_delay_ms*.

```
write(OUT1, 1);
_delay_ms(250);
write(OUT1, 0);
_delay_ms(250);
```

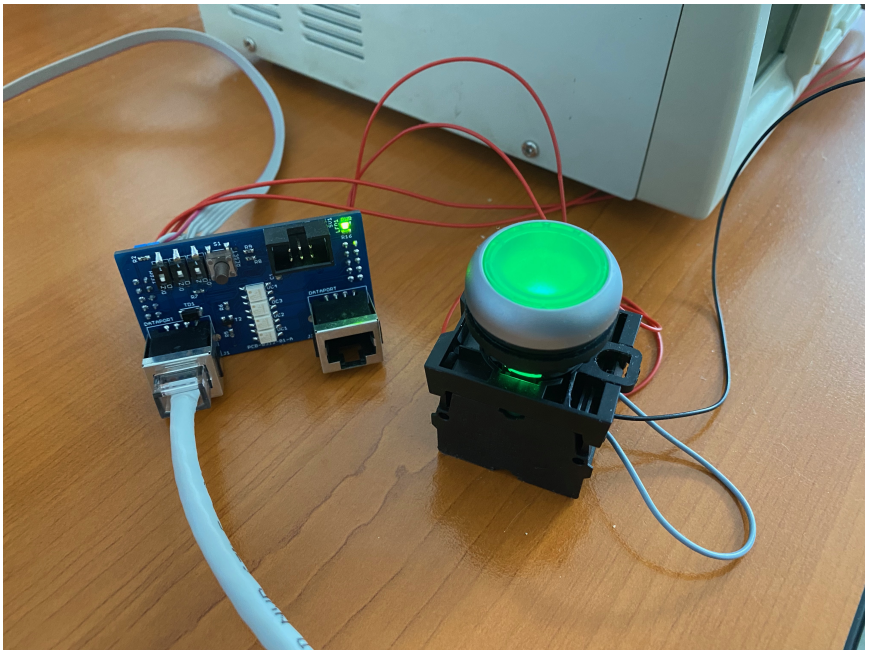
Pro test funkce vstupů jsem použil připojené tlačítko na vstup a již připojenou LED diodu. Tuto funkci vytvoříme pomocí funkce *read* a *write*.

```
if(!read(OUT1))
        write(OUT1, 1);
else
        write(OUT1, 0);
```

Pro test komunikace jsem použil převodní ze sběrnice RS-485 na USB a program Herkules který slouží ke čtení a zápisu hodnot na sériovou sběrnici. Díky tomu bylo možné ověřit jestli převodník na modulu funguje správně a je správně zapojen. Nakonec stačí provést inicializace UART komunikace a použít zápis do registru $UDR0$, který zajistí odeslání bytu na sběrnici.

```
write(TEN, 1);

while(!(UCSR0A &  (1<<UDRE0)));
UDR0 = 'a';
while(!(UCSR0A &  (1<<TXC0)));
```



Obr. 8.1: Ukázka zapojeného modulu při testování funkčnosti

9

Software pro IO modul

Celý projekt je rozložen do několika samostatní souborů a knihoven, tak aby byla zvýšena přehlednost celého programu. V hlavním souboru se nachází pouze metody pro inicializaci komponent a samotná smyčka pro běh programu. Další částí jsou knihovny s deklaracích hlavních proměnných a definicemi, knihovna pro obsluhu pro funkce obsluhující piny procesoru a nakonec soubor s funkcemi pro obsluhu komunikačního protokolu MODBUS.

9.1 Knihovny

První z knihoven se jmenuje *ioctrl.h*. Tato knihovna je převzatá a obsahuje funkce pro obsluhu definice pro snadné nastavení a obsluhu pinů. Pro náš projekt je jsou nejdůležitější 4 funkce z této knihovny. Funkce *setout* a setin nastaví vlastnost daného pinu. Parametrem funkcí je port a číslo pinu. Další dvě důležité funkce, které potřebujeme jsou *read* a *write*. Díky nim lze zapsat na konkrétní pin hodnotu stavu nebo přečíst stav na konkrétním pinu. I když se nejedná o složité definice, lze díky těmto jednoduchým funkcím velmi zpřehlednit výsledný kód. Druhá knihovna je *iodef.h*. Zde se nacházejí definice konkrétních pinů a deklarace některých globálních proměnných. Knihovna *std.h* obsahuje pouze přejmenování proměnných *unsigned char* a *unsigned int* na *uin* a *uchar*. Tyto definice také slouží k vytvoření přehlednějšího a kratšího kódu. Poslední knihovnou je *MODBUS.h* Tyto knihovna obsahuje definice, proměnné a metody pro obsluhu a zpracování MODBUS komunikace.

9.2 Obsluha MODBUS komunikace

V tomto souboru jsou obsaženy všechny důležité metody pro obsluhu komunikace. Jsem zde dvě hlavní metody *recieve_task* a *send_task*. Funkce *recieve_task* slouží ke zpracování přijatých dat. Po deklaraci potřebných proměnných je zde hlavní podmínka, která ověří, jestli je přijatá zpráva pro tento modul. Pokud ne, provede se pouze vyčištění bufferu, kde jsou uložena přijatá data a žádná funkce se nevykoná. Pokud ano ověří se, který z

funkčních příkazů se má vykonat. Jak již bylo zmíněno v protokolu MODBUS je těchto příkazů celkem osm. Všechny funkce mají podobnou strukturu. Nejprve se provede ověření některých přijatých dat tak, aby bylo zajištěno že je možné s daty pracovat. Pokud jakákoli z podmínek nebude splněna, dojde k nastavení chybového kódu, který na konci metody odešle zpět chybovou hlášku. Po splnění všech podmínek se vykoná samotná funkce, která na konci odešle zpět odpověď. Přijímací metoda obsahuje všechny funkční kódy tak, aby bylo zajištěno že modul správně odpoví na kterýkoli z dotazů. Pro funkci modulu jsou v současném řešení použita pouze polovina funkcí sloužící k práci s diskrétními registry. Funkce pro práci s analogovými registry, které jsou v metodě obsaženy, jsou připraveny pro budoucí použití.

Funkce pro práci s přijatými daty jsou v metodě *recieve_task* seřazeny chronologicky podle kódu funkce. První dvě funkce slouží pro odeslání hodnot digitálních vstupů a výstupů z diskrétních registrů, jsou téměř identické. Nejprve se zjišťuje kolik je potřeba přečíst bitů z registrů a na základě této informace se vybere kolik bytů se má poslat zpět. Poté se naplní buffer s daty pro odeslání hodnotami v daných registrech. Nakonec se všechny tyto informace předají metodě *send_task*.

Dalšími dvěma funkcemi, které jsou v modulu používány jsou zápisy do diskrétních registrů a to ať zápis do konkrétního výstupu nebo pouze do jednoho. V případě zápisu do jednoho registru nejprve proběhnou všechny ověřovací procedury, jak již bylo zmíněno v minulém odstavci. Poté se ověří jestli se má výstup nastavit do stavu 1 nebo 0. Nakonec se na základě těchto informací provede zápis do příslušného registru a pomocí proměnné *is_change_out* se předá řídí funkci informace, že v registrech došlo ke změně. Stejně jako v předchozím případě se i zde po úspěšném provedení funkce odešle odpověď pomocí funkce *send_task*.

Kód níže ukazuje funkci pro naplnění bufferu hodnotami digitálních vstupů uložených v diskrétních registrech.

```
data_array[0] = 0x01;

if (no_of_register > 8)
{
        data_array[0] = 0x02;
}
if (no_of_register > 16)
{
        data_array[0] = 0x03;
}
if (no_of_register > 24)
{
        data_array[0] = 0x04;
}

uint no_of_bytes = (uint)data_array[0];
data_array[1] = digital_in[0];
data_array[2] = digital_in[1];
data_array[3] = digital_in[2];
data_array[4] = digital_in[3];

send_task(READ_DI, no_of_bytes + 1, data_array);
```

Metoda *send_task* je oproti *recieve_task* poměrně jednoduchá. V první řadě jsou zde deklarovány potřebné proměnné. Poté se provede vyčištění bufferu pro odeslání dat tak, aby nedošlo ke kolizi některých informací a data se poslala správně. Poté se buffer naplní podle zadaných parametrů a vytvoří se dva byty ve kterých je obsažen checksum. Nakonec se provede odeslání dat. Zde je potřeba myslet na několik důležitých věci, aby odesílání správně fungovalo. Na začátek je potřeba zapnout pin, který povolí odesílání převodníku, poté se vypne RX pin procesoru, vzhledem k tomu že přijímací pin převodníku je natrvalo uzemněn, pokud bychom tento krok neudělali, data by se vracela zpět a převodník by se tím zahltil. Samotné odesílání je jednoduchý cyklus, který ověří že je buffer s daty volný, pošle se příslušný byte z bufferu a počká se než dojde k odeslání bytu. Po odeslání celého bufferu se vypne pin, který povoluje odesílání a znova se zapne přijímání. Celou funkci lze vidět níže.

```
write(TEN, 1);
UCSR0B &= ~(1<<RXEN0);

for (int i = 0; i < lenght_sended_data; i++)
{
        while(!(UCSR0A &  (1<<UDRE0)));
        UDR0 = tx_data[i];
        while(!(UCSR0A &  (1<<TXC0)));
}

write(TEN, 0);
UCSR0B |= 1<<RXEN0;
```

Soubor obsahuje ještě několik podpůrných metod, bez kterých by nebylo možné správně s daty pracovat. První z nich je metoda *check_crc*, která slouží pro počítání checksum. Funkce využívá definice CRC-16-IBM, který je popsán v tabulce [x.y]. Návratovou hodnotou této metody je 16 bitové číslo se kterým se následné dále pracuje.

Další metody už jsou poměrně jednoduché a slouží spíše ke zpřehlednění kódu. Jsou to dvě inverzní funkce. Metoda $make_int$ vytváří ze dvou 8 bitových hodnot dvě 16 bitové a metoda *get_hb_lb* má za úkol obrácený proces. Nakonec jsou zde metody $flush\_rx\_data$ a $flush\_tx\_data$ které složí pro vyčištění bufferů ve kterých jsou obsažena data pro přijímání a odesílání.

9.3 Hlavní funkce programu

Hlavní metodou celého programu, která se volá při zapnutí je metoda *main*. Nejprve jsou zde volány metody pro inicializaci pinů, inicializace sériové komunikace pro UART a vyčištění registrů. Následně se provede aktivace přerušení. Poté už program vstupuje do nekonečné smyčky, ve které se ověřuje, jestli se přijali data, pokud ano vykoná se metoda *recieve_task*. Následně se ověří, jestli neproběhla změna vstupech nebo nebyla zapsána nová hodnota do registrů pro výstupy. Poté se celá smyčka opakuje.

K provedení inicializace uart komunikace je potřeba důkladně prostudovat dokumentaci k mikročipu. Jsou zde informace a názvy jednotlivých registrů ve kterých jsou uloženy parametry sériové komunikace. Nejprve se provádí nastavení rychlosti komunikace. Toto nastavení je obsaženo v registru $UBRR0H$ a $UBBR0L$. V tabulce pro rychlosti komunikace při 16 MHz nalezneme hodnotu, která pro zvolenou přenosovou rychlost. V našem tomto případě je komunikační rychlost 115200*bps* což odpovídá hodnotě 8. V našem případě je pro zmenšení chybovosti nastavena v registru $U2X0$ dvojnásobné rychlost. V tomto případě je tato hodnota 16. Další nastavení je pro 8 bitové slovo, tato hodnota odpovídá nastavení registrů $UCSZ01$ a $UCSZ02$ do hodnoty 1. V posledním kroku povolíme přijímaní a vysílání sériové komunikace a přerušení na přijímacím pinu. Toto natavení odpovídá hodnotám 1 v registrech $RXEN0$, $TXEN0$ a $RXCIE0$.

```
void uart_init()
{
        UBRR0H = 0;
        //115200
        UBRR0L = 16;
        // double speed
        UCSR0A = 1<<U2X0;
        // 8bit
        UCSR0C = 1<<UCSZ01 | 1<<UCSZ00;
        // enable receiver, rx interrupt
        UCSR0B = 1<<RXEN0 | 1<<TXEN0 | 1<<RXCIE0; // | 1<<TXCIE0;
}
```

Pro správné přijímaní dat je potřeba definovat funkci, která se vykoná při aktivaci přerušení na RX pinu. Tato funkce se definuje v takzvaném ISR. Pro RX pin je to funkce $ISR(USART\_RX\_vect)$. První co se vykoná je přečtení bytu v registru sériové komunikace. Následně se tento byte zapíše do bufferu. Poté je zde několik ověření, které zajistí že se vždy přijme správný počet bytů. Pokud je dosaženo správného počtu přijatých bytů, aktivuje se příznak, který zajistí vykonání zpracování dat v hlavní funkci.

```
ISR(USART_RX_vect)
{
        rx_byte = UDR0;
        rx_data[cnt_rx] = rx_byte;

        if((cnt_rx == 1) && ((rx_byte == 0x0F) || (rx_byte == 0x10)))
                cahnge_data_lenght = 1;

        if((cnt_rx == 6) && (cahnge_data_lenght == 1))
        {
                cahnge_data_lenght = 0;
                modbus_data_lenght = 9 + rx_byte;
        }

        cnt_rx++;

        if (cnt_rx == modbus_data_lenght)
        {
                check_rx_data = 1;
        }
}
```

Poslední dvě důležité metody slouží ke čtení hodnot na vstupech a zápisu hodnot na výstupy. V metodě *check_inputs* se provede kontrola jestli došlo ke změně na vstupech a poté

se hodnota ze vstupů přímo překlopí do registru vstupů. V metodě *check_output_state* se ověří jestli není aktivní příznak změny ve výstupních registrech a pokud ano dojde k propsání hodnot z registru na výstupy.

10

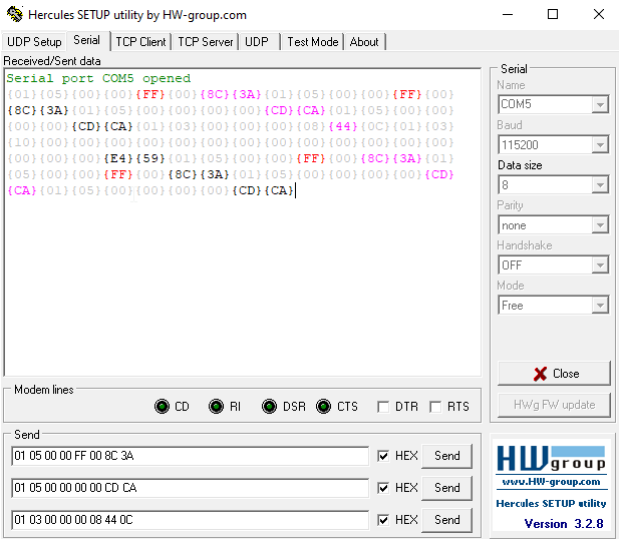
Aplikace softwaru a test funkce

Před aplikací software jsem k zařízení připojil Všechny předem připravené periferie. V první řadě tlačítkový modul se signalizační LED diodou a poté také programovací a komunikační kabel. Pro komunikaci jsem použil převodník z RS-485 na USB, který v počítači vytvoří virtuální sériová port se kterým lze komunikovat. Pro napájení jsem použil regulovatelný zdroj Manson, na kterém jsem nastavil vstupní hodnotu $24V$.

Obr. 10.1: Převodní RS-485 na USB od firmy PremiumCord [9]

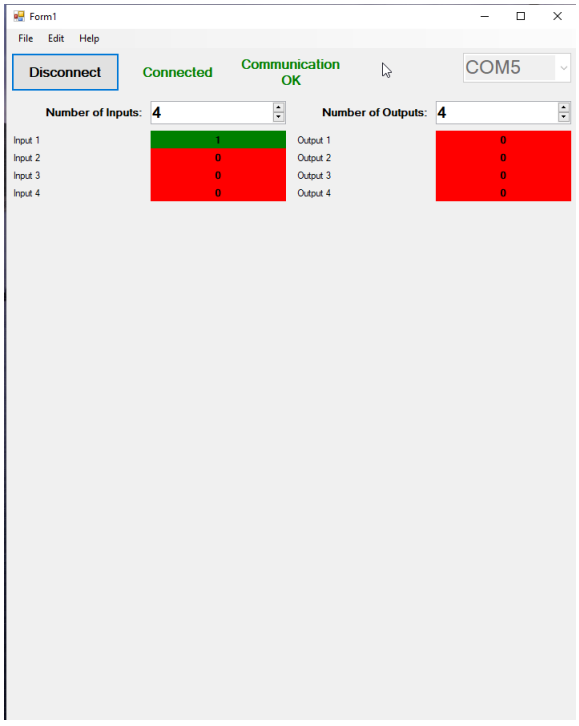
Pro prvotní komunikaci jsem použil software Hukrules, kterým lze na sériový port posílat data. Díky tomu jsem dokázal ověřit správnou funkčnost zápisu a čtení registrů. Pro zapnutí jsem použil příkaz $0x01$ $0x05$ $0x00$ $0x00$ $0xFF$ $0x00$ $0x8C$ $0x3A$, který slouží k zapsání hodnoty 1 do nulté pozice registru digitálních výstupů. Čímž se rozsvítila LED dioda připojená na výstup 1. Příkaz pro vypnutí je velmi podobný jen se hodnoty FF změní na 00. Příkaz tedy poté vypadá $0x01$ $0x05$ $0x00$ $0x00$ $0x00$ $0x00$ $0x30$ $0x14$.

Pro důkladnější testování bylo potřeba požít složitější aplikaci. Jelikož plno kvalitní komunikačních rozhraní je placených nebo nevyhovovali mým požadavkům, rozhodl jsem se, vytvořit si vlastní jednoduchou komunikační aplikaci. Tato aplikace dokáže sekvenčně posílat dotaz na stav registrů s digitálními vstupy a současně taký zapsat hodnotu do



Obr. 10.2: Ukázka zkoušky komunikace pomocí softwaru Herkules

digitálních výstupů. Je v ní možné vybrat do kolika registrů zapisuji a z kolika registrů chci číst informaci.



Obr. 10.3: Ukázka vlastního softwarového řešení pro komunikaci s modulem

11

Tvorba výrobní dokumentace

Pro možnost zařízení seriově vyrábět je potřeba zajistit kvalitní výrobní dokumentaci, které bude obsahovat všechny soubory potřebné pro výrobu a také postupy.

Výrobní dokumentace musí obsahovat schéma zapojení celého zařízení, gerber data pro výrobu DPS, seznam použitých součástek (tvz. BOM), rozložení součástek pro osazaní a nakonec .hex soubor ve kterém jsou obsažena data které se mají do zařízení naprogramovat.

12

Plán na další revize zařízení a možnost rozšiřujících modulů

Modul vyrobený v rámci této práce je pouze prototypem a vzhledem k současné situaci na trhu bylo potřeba s výběrem součástek místy velmi improvizovat. Proto příští revize tohoto modulu budou místy velmi pozměněné. První změnou bude výměna převodníku RS-485, pro potřeby tohoto modulu se vzhledem k ceně a dostatku místa zvolí více dostupný a hojně rozšířený převodník MAX485. Druhou velkou změnou bude výměna procesoru za moderní Atmega3208, v tomto případě půjde o rozsáhlejší zásah do modulu a programu. Pro pozdější vývoj však velmi usnadní práci.

Pro modul je plánováno množství rozšiřujících modulů. Pro tento případ je modulu připraven konektor na který je vyvedena sběrnice I2C. Díky tomu je možné přidat více vstupů a výstupů nebo moduly pro měření veličin s A/D převodníky.

Pro umístění modulu na zadní stranu tlačítkového modulu jsou plánovány dvě malé desky se třemi vývody a kolíkovou lištou, který se našroubují do modulu a IO modul se na ně následně nacvakne. Jelikož může být tlačítek více sebe je výhodnější umístit modul s procesorem na jedno z nich a na další použít rozšiřující modul na kterém bude pouze napájení a I2C převodník z digitálního vstupu nebo výstupu.

Další umístění může být na měřících nebo zakládacích modulech pro kabelové svazky, které se v testerech běžně vyskytují. Zde se vzhledem ke konstrukci pouze jednoduše nacvakne na zadní stranu modulu. V některých případech je v těchto modulech přímo hlídat některou veličin. Z tohoto důvodu je výhodné mít připraven modul s A/D převodníkem který bude tuto veličinu měřit.

V neposlední řadě je připraven programovací modul. Při sériové výrobě je potřeba moduly velmi snadno programovat. Z tohoto důvodu bude připraven modul na který se hlavní modul pouze nacvakne a spustí se programování.

13

Závěr

Cílem této práce bylo vytvořit kompaktní IO modul schopný pracovat na komunikační protokolu MODBUS. Z velké části se podařilo vytvořit funkční řešení, které sice má svá omezení, ale pro odladění těchto nedostatků by bylo potřeba delší testování, které nebylo vzhledem k časovému omezení možné.

Po hardwarové stránce se podařilo vytvořit prototyp, který splňuje téměř všechny požadavky. Z důvody většího množství nedostupných součástek bylo potřeba shánět spoustu náhrad a občas i velmi improvizovat. Toto řešení není z ekonomické stránky příliš výhodné a v této oblasti je potřeba provést jistou optimalizaci, tak aby bylo možné řešení nasadit do skutečného provozu.

Na ladění softwaru nebylo příliš mnoho času, proto má určité nedostatky, které je potřeba do budoucna odladit. Jako hlavní nedostatek vnímám nemožnost testování chování při více zapojených modulech. Celé testování aplikace probíhalo na simulované komunikaci, kde na jedné straně byl modul a na druhé převodník z RS-485 na USB, díky kterému je možné zařízení připojit k počítači a za pomoci aplikací testovat chování.

Literatura

[1] Modbus. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-05-26]. Dostupné z: https://en.wikipedia.org/wiki/Modbus

[2] RS-485. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-05-26]. Dostupné z: https://en.wikipedia.org/wiki/RS-485

[3] Protokol Modbus RTU v kostce (s podrobnými popisy a příklady) [online]. [cit. 2022-05-26]. Dostupné z: https://ipc2u.cz/blogs/news/protokolu-modbus-rtu-v-kostce-s-podrobnymi-popisy-a-priklady

[4] Tedia MU-3222A [online]. [cit. 2022-05-26]. Dostupné z: https://www.tedia.cz/produkty/mu322xa-mu325xa.html

[5] Advantech PCIE-1730 [online]. [cit. 2022-05-26]. Dostupné z: https://www.advantech.com/products/1-2mlkb0/pcie-1730/mod˙6d7a4820-cf93-4f6b-bc3a-09d4e1ca62bc

[6] Advantech PCIE-1730 Datasheet [online]. [cit. 2022-05-26]. Dostupné z: https://advdownload.advantech.com/productfile/PIS/PCIE-1730/Product% 20-% 20Datasheet/PCIE-173020180910101824.pdf

[7] ATmega328P Datasheet [online]. [cit. 2022-05-26]. Dostupné z: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P˙Datasheet.pdf

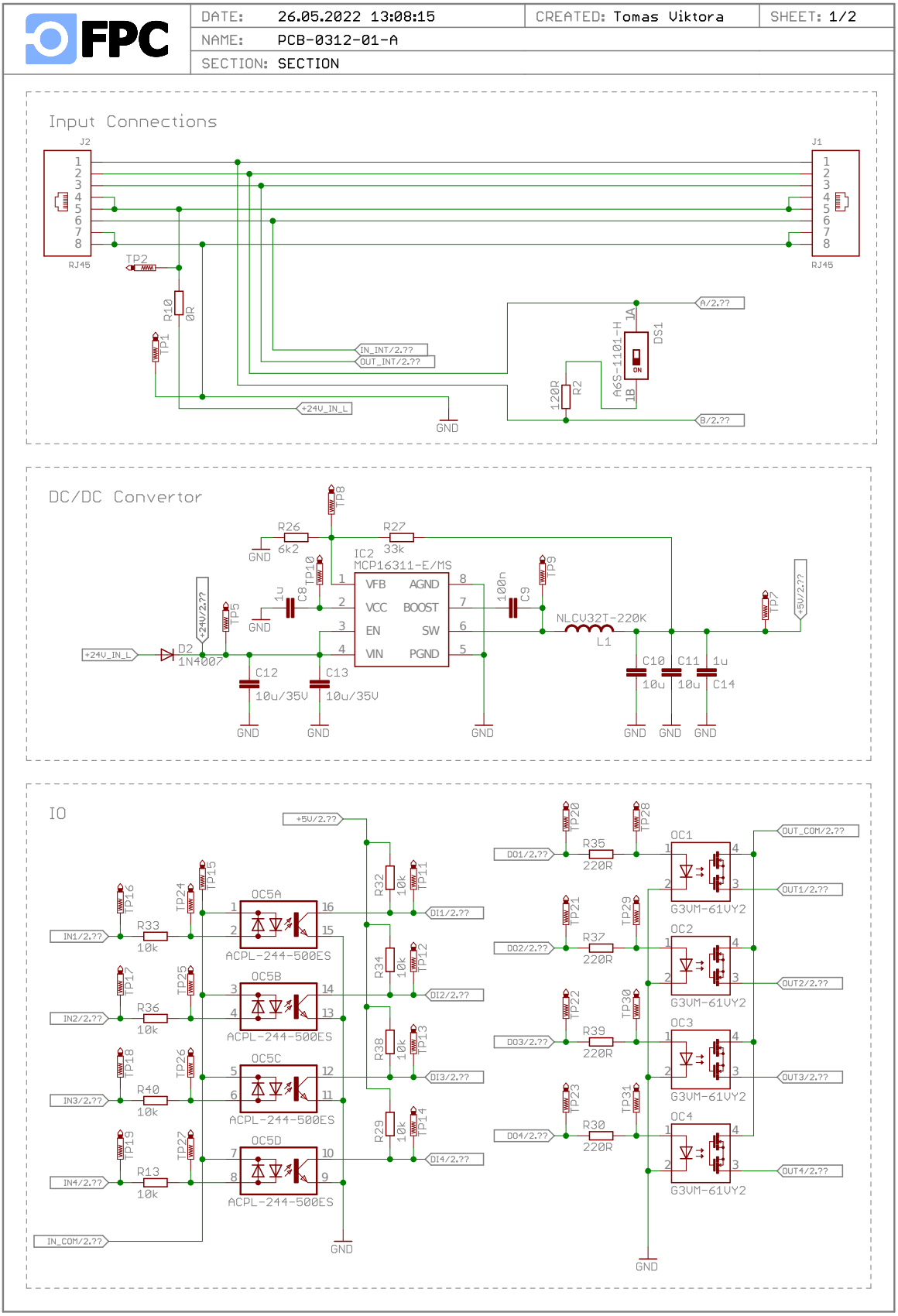
[8] Cyclic redundancy check. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-05-26]. Dostupné z: https://en.wikipedia.org/wiki/Cyclic˙redundancy˙check

[9] PremiumCord USB - USB2.0 na RS485 adapter [online]. In: . [cit. 2022-05-26]. Dostupné z: https://www.czc.cz/premiumcord-usb-usb2-0-na-rs485-adapter/80182/produkt

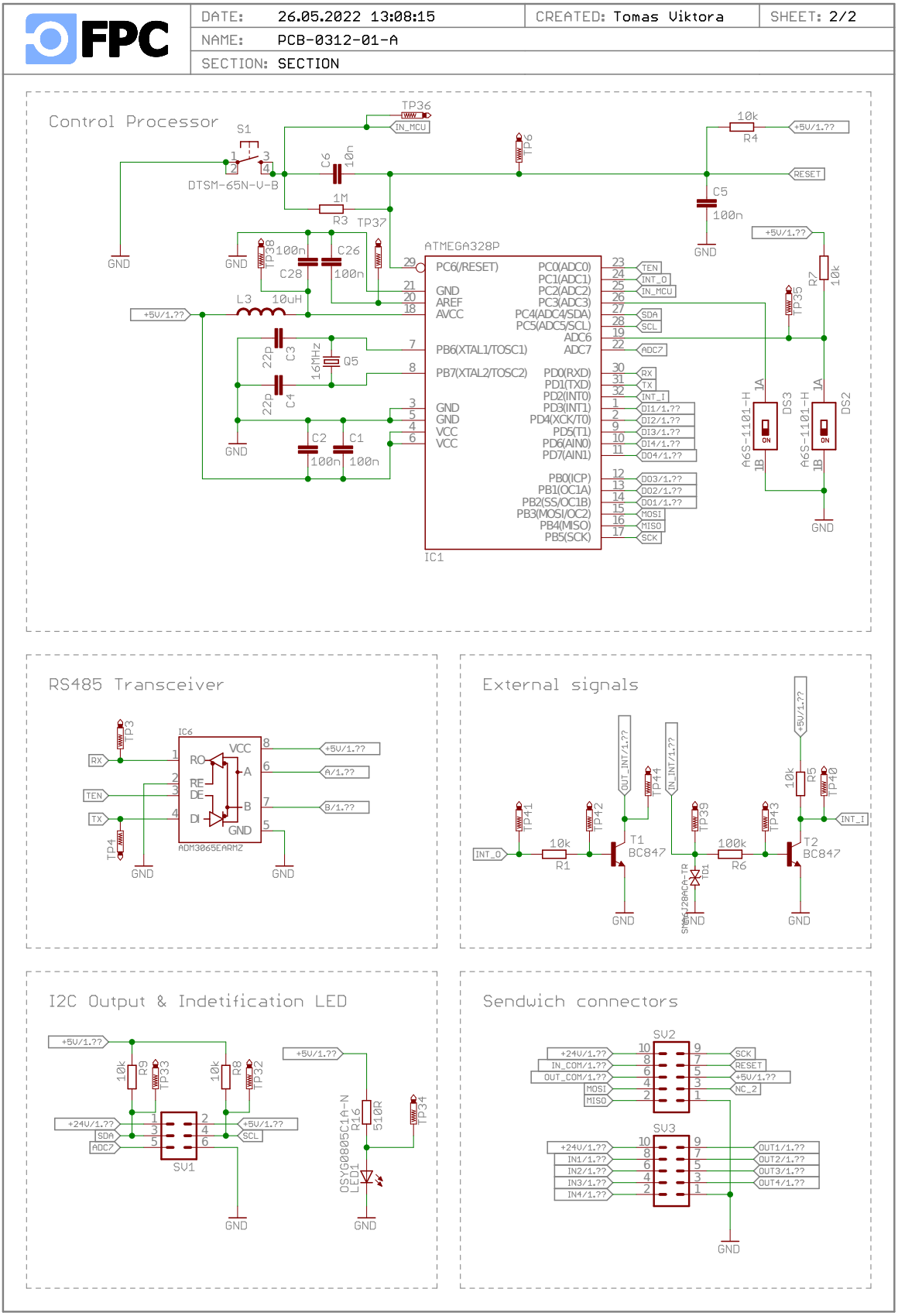
[10] MCP16311 Datasheet [online]. [cit. 2022-05-26]. Dostupné z: https://ww1.microchip.com/downloads/aemDocuments/documents/APID/ProductDocuments/I 30V˙Input-1A-Output-High-Efficiency-Integrated-Synchronous-Switch-Step-Down-Regulator-DS20005255D.pdf

Příloha A

Schémata zapojení



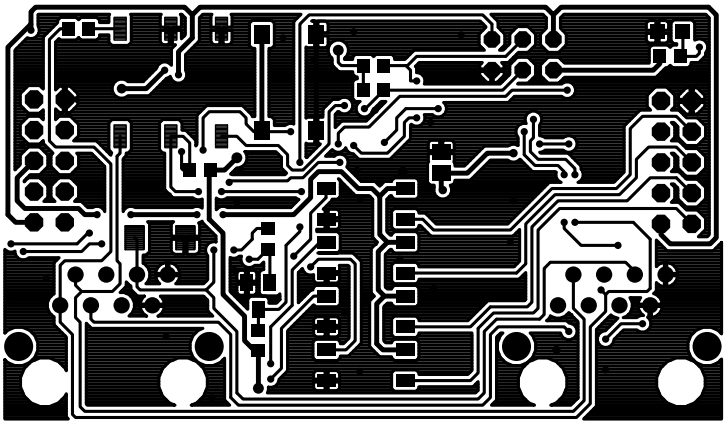
Obr. A.1: Schéma zapojení 1



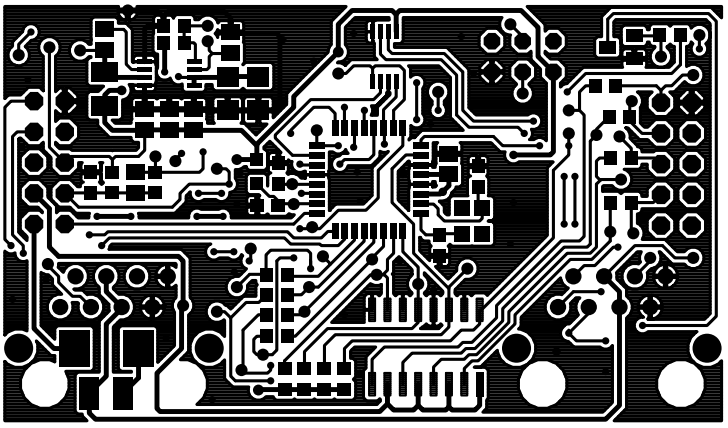
Obr. A.2: Schéma zapojení 2

Příloha B

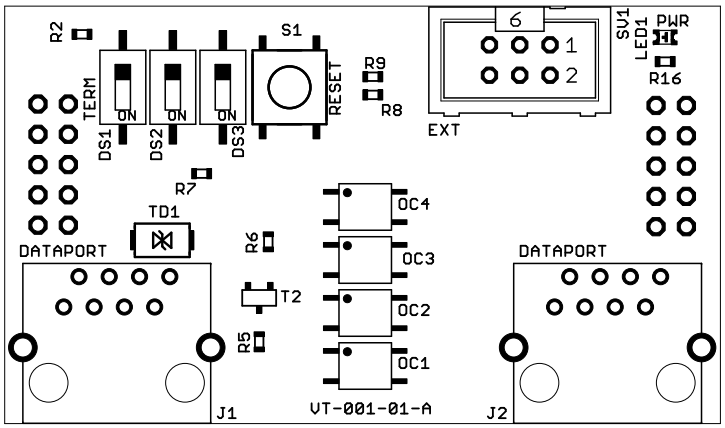
Desky plošných spojů



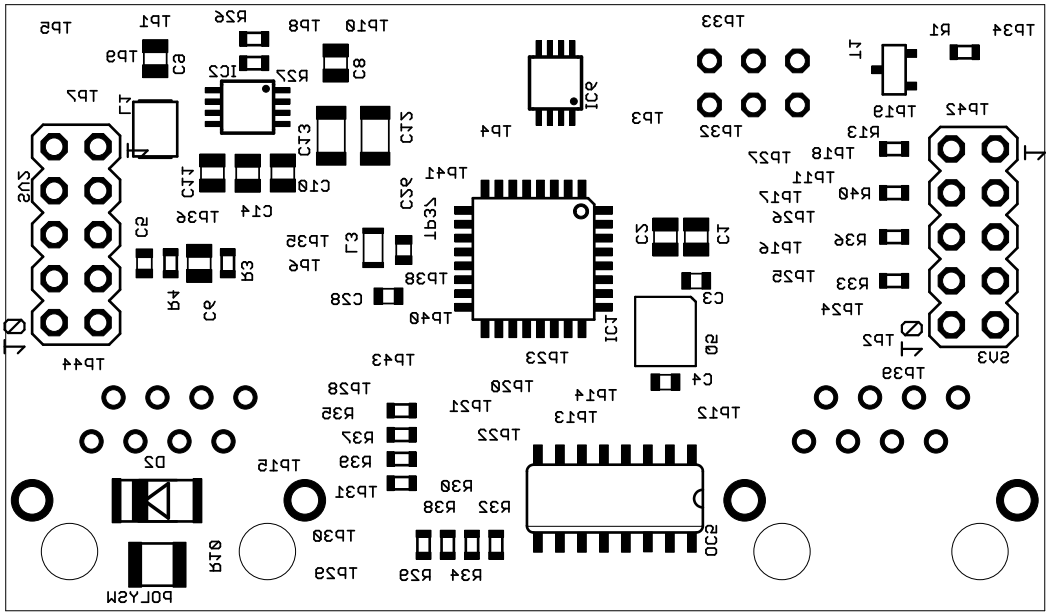
Obr. B.1: Nepájivá maska horní strany PCB



Obr. B.2: Nepájivá maska spodní strany PCB



Obr. B.3: Osazovací plán horní strany PCB



Obr. B.4: Osazovací plán spodní strany PCB

Příloha C

Zdrojový kód

```
1
2  /*
3   * iodef.h
4   *
5   * Created: 14.03.2022 8:50:19
6   *  Author: tomas.viktora
7   */
8   #include "std.h"
9
10 #ifndef IODEF_H_
11 #define IODEF_H_
12
13 //Inputs
14 #define IN1 D,3
15 #define IN2 D,4
16 #define IN3 D,5
17 #define IN4 D,6
18
19 //Outputs
20 #define OUT1 B,2
21 #define OUT2 B,1
22 #define OUT3 B,0
23 #define OUT4 D,7
24
25 //UART
26 #define RX D,0
27 #define TX D,1
28 #define TEN C,0
29
30 //External signals
31 #define INT_O C,1
32 #define INT_I D,2
33
34 //Reset watchdog
35 #define IN_MCU C,2
36
37 //Switch
38 #define SW1 C,3
39
40 uint no_of_inputs, no_of_outputs;
41 uchar last_in_state;
42 uint check_rx_data;
43 uint modbus_data_lenght;
44 uint cahnge_data_lenght;
45
46 volatile uchar rx_byte;
47
48 #endif /* IODEF_H_ */
49
```

```

/*
 * std.h
 *
 * Created: 27.04.2022 16:07:27
 *  Author: tomas.viktora
 */


#ifndef STD_H_
#define STD_H_

typedef unsigned char uchar;
typedef unsigned int uint;

#endif /* STD_H_ */
```

```

/* ------------------------------------------------------------------------
 *
 *        AVR BASIC IO PORT CONTROLS
 * ------------------------------------------------------------------------
 *
 *        Ing. Lukas PIVONKA, www.lucass.cz
 *
 *  12/5/2009
 *                - new defines strobe and strobe_us
 *                - norm_bool define renamed to nbool
 *
 */

#ifndef _IOCTRL_H_
#define _IOCTRL_H_

// _____DECLARATIONS_____________________________________________________________

// joins two parameters
#define concat(a, b) a ## b
// normalize a boolean, any value that corresponds to logical one is converted to number "1"
#define nbool(x) ((x)?1:0)
// set pin as output
#define setout_nodef(port, pin_no) (concat(DDR, port) |= _BV(pin_no))
#define setout(define) setout_nodef(define)
// set pin as input
#define setin_nodef(port, pin_no) (concat(DDR, port) &= ~_BV(pin_no))
#define setin(define) setin_nodef(define)
// read port pin no value
#define read_port_nodef(port, pin_no) (concat(PORT, port) & _BV(pin_no))
#define read_port(define) read_port_nodef(define)
// read pin no value
#define read_nodef(port, pin_no) (concat(PIN, port) & _BV(pin_no))
#define read(define) read_nodef(define)
// write port pin no value
#define write_nodef(port, pin_no, value) value?(concat(PORT, port) |= _BV(pin_no)):(concat(PORT, port) &= ~_BV(pin_no))
#define write(define, value) write_nodef(define, value)
// invert port pin no value
#define invert_nodef(port, pin_no) concat(PORT, port) ^= _BV(pin_no)
#define invert(define) invert_nodef(define)
// strobe pulse for three clocks (instruction time), level of the pulse depends on previous port value (zero = positive pul
#define strobe_nodef(port, pin_no) concat(PORT, port) ^= _BV(pin_no); concat(PORT, port) ^= _BV(pin_no)
#define strobe(define) strobe_nodef(define)
// strobe pulse for a number of microseconds (util/delay.h must be included and also F_CLK must be set for using this defin
#define strobe_us_nodef(port, pin_no, delay) concat(PORT, port) ^= _BV(pin_no); _delay_us(delay); concat(PORT, port) ^= _BV
#define strobe_us(define, delay) strobe_us_nodef(define, delay)


// ------------------------------------------------------------------------

#endif

```

```
1
2  /*
3   * MODBUS.h
4   *
5   * Created: 11.04.2022 13:22:30
6   *  Author: tomas.viktora
7   */
8
9   #include <avr/io.h>
10  #include <avr/interrupt.h>
11  #include <util/delay.h>
12  #include <stdio.h>
13  #include <string.h>
14  #include "ioctrl.h"
15  #include "iodef.h"
16  #include "std.h"
17
18
19  #ifndef MODBUS_H_
20  #define MODBUS_H_
21
22  #define SLAVE_ID 0x01 // prozatím, později zakomentovat
23
24  #define ID_BYTE 0
25  #define FUNCTION_BYTE 1
26  #define LENGHT_BYTE 2
27  #define DATA_START_BYTE 2
28
29  #define DO_SET_ON 0xFF00
30  #define DO_SET_OFF 0x0000
31
32  #define READ_DO 0x01
33  #define READ_DI 0x02
34  #define READ_AO 0x03
35  #define READ_AI 0x04
36  #define WRITE_SINGLE_DO 0x05
37  #define WRITE_SINGLE_AO 0x06
38  #define WRITE_MULTIPLE_DO 0x0F
39  #define WRITE_MULTIPLE_AO 0x10
40
41  #define NO_OF_STATIC_BYTES 2
42  #define NO_OF_CRC_BYTE 2
43
44  volatile uchar tx_data[32];
45  volatile uchar rx_data[32];
46  volatile uint cnt_rx;
47  volatile uint cnt_tx;
48  uchar digital_in[4];
49  uchar digital_out[4];
50  uint16_t analog_in[8];
51  uint16_t analog_out[8];
52
53  uint is_change_out;
54
55  #endif /* MODBUS_H_ */
56
57  uint16_t check_crc(uint data_length, uchar *data);
58  uint make_int(uchar crc_HB, uchar crc_LB);
59  void send_task(uchar function_code, uchar data_length, uchar *data);
60  void flush_rx_data(void);
61  void flush_tx_data(void);
62  void recieve_task(void);
63
```

```
1
2  /*
3  * MODBUS.c
4  *
5  * Created: 11.04.2022 13:19:16
6  *  Author: tomas.viktora
7  */
8  #include "MODBUS.h"
9  #include "std.h"
10
11
12 uint make_int(uchar HB, uchar LB)
13 {
14         uint16_t temp = 0;
15
16         temp = HB;
17         temp <<= 8;
18         temp |= LB;
19
20         return temp;
21 }
22
23 void flush_rx_data(void)
24 {
25         signed char i = (32 - 1);
26
27         while(i > -1)
28         {
29                 rx_data[i--] = 0x00;
30         };
31
32         cnt_rx = 0x00;
33 }
34
35
36 void flush_tx_data(void)
37 {
38         signed char i = (32 - 1);
39
40         while(i > -1)
41         {
42                 tx_data[i--] = 0x00;
43         };
44
45         cnt_tx = 0x00;
46 }
47
48 uint16_t check_crc(uint16_t data_length, uchar *data)
49 {
50         uint16_t crc_word = 0xFFFF;
51
52         unsigned int i = 0;
53         char bit = 0;
54
55         for( i = 0; i < data_length; i++ )
56         {
57                 crc_word ^= data[i];
58
59                 for( bit = 0; bit < 8; bit++ )
60                 {
61                         if( crc_word & 0x0001 )
62                         {
63                                 crc_word >>= 1;
64                                 crc_word ^= 0xA001;
65                         }
66                         else
67                         {
68                                 crc_word >>= 1;
69                         }
70                 }
71         }
72
73         return crc_word;
```

```
74 }
75
76 void get_hb_lb(uint value, uchar *HB, uchar *LB)
77 {
78         *LB = (value & 0x00FF);
79         *HB = ((value & 0xFF00) >> 0x08);
80 }
81
82 void recieve_task(void)
83 {
84         static uchar data_array[0x18] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
85         uint16_t start_register = 0x0000;
86         uint16_t no_of_register = 0x0000;
87         uint16_t state_digital_out = 0x0000;
88         uint16_t state_analog_out = 0x0000;
89         uint16_t crc = 0x0000;
90
91         uint error_label = 0;
92
93
94         if (rx_data[ID_BYTE] == SLAVE_ID)
95         {
96                 switch(rx_data[FUNCTION_BYTE])
97                 {
98                         case READ_DO:
99                         {
100                                start_register = make_int(rx_data[2],rx_data[3]);
101                                no_of_register = make_int(rx_data[4],rx_data[5]);
102                                crc = make_int(rx_data[7],rx_data[6]);
103
104                                if((start_register >= 0) && (start_register <= 31))
105                                {
106                                        if((no_of_register <= 32) && ((start_register + no_of_register -1) <= 31))
107                                        {
108                                                if(crc == check_crc(6, rx_data))
109                                                {
110                                                        data_array[0] = 0x01;
111
112                                                        if (no_of_register > 8)
113                                                        {
114                                                                data_array[0] = 0x02;
115                                                        }
116
117                                                        if (no_of_register > 16)
118                                                        {
119                                                                data_array[0] = 0x03;
120                                                        }
121
122                                                        if (no_of_register > 24)
123                                                        {
124                                                                data_array[0] = 0x04;
125                                                        }
126
127                                                        uint no_of_bytes = (uint)data_array[0];
128                                                        data_array[1] = digital_out[0];
129                                                        data_array[2] = digital_out[1];
130                                                        data_array[3] = digital_out[2];
131                                                        data_array[4] = digital_out[3];
132
133                                                        send_task(READ_DO, no_of_bytes + 1, data_array);
134                                                }
135                                                else
136                                                error_label = -1;
137                                        }
138                                        else
139                                        error_label = -1;
140                                }
141                                else
142                                error_label = -1;
143
144
145                                if(error_label == -1)
146                                {
```

```
147                                     data_array[0] = 0x02;
148                                     send_task(READ_DO | 0x80, 1, data_array);
149                             }
150
151                             break;
152                     }
153
154                     case READ_DI:
155                     {
156                             start_register = make_int(rx_data[2],rx_data[3]);
157                             no_of_register = make_int(rx_data[4],rx_data[5]);
158                             crc = make_int(rx_data[7],rx_data[6]);
159
160                             if((start_register >= 0) && (start_register <= 31))
161                             {
162                                     if((no_of_register <= 32) && ((start_register + no_of_register -1) <= 31))
163                                     {
164                                             if(crc == check_crc(6, rx_data))
165                                             {
166                                                     data_array[0] = 0x01;
167
168                                                     if (no_of_register > 8)
169                                                     {
170                                                             data_array[0] = 0x02;
171                                                     }
172
173                                                     if (no_of_register > 16)
174                                                     {
175                                                             data_array[0] = 0x03;
176                                                     }
177
178                                                     if (no_of_register > 24)
179                                                     {
180                                                             data_array[0] = 0x04;
181                                                     }
182
183                                                     uint no_of_bytes = (uint)data_array[0];
184                                                     data_array[1] = digital_in[0];
185                                                     data_array[2] = digital_in[1];
186                                                     data_array[3] = digital_in[2];
187                                                     data_array[4] = digital_in[3];
188
189                                                     send_task(READ_DI, no_of_bytes + 1, data_array);
190                                             }
191                                             else
192                                             error_label = -1;
193                                     }
194                                     else
195                                     error_label = -1;
196                             }
197                             else
198                             error_label = -1;
199
200                             if(error_label == -1)
201                             {
202                                     data_array[0] = 0x02;
203                                     send_task(READ_DI | 0x80, 1, data_array);
204                             }
205
206                             break;
207                     }
208
209                     case READ_AO:
210                     {
211                             start_register = make_int(rx_data[2],rx_data[3]);
212                             no_of_register = make_int(rx_data[4],rx_data[5]);
213                             crc = make_int(rx_data[7],rx_data[6]);
214
215                             uchar hb = 0x00;
216                             uchar lb = 0x00;
217
218                             if((start_register >= 0) && (start_register <= 7))
219                             {
```

```
220                                     if((no_of_register <= 8) && ((start_register + no_of_register -1) <= 7))
221                                     {
222                                             if(crc == check_crc(6, rx_data))
223                                             {
224                                                     data_array[0] = (2 * no_of_register);
225
226                                                     int j = 1;
227
228                                                     for (int i = 1; i < (2 * no_of_register) + 1; i++)
229                                                     {
230                                                             hb = 0x00;
231                                                             lb = 0x00;
232
233                                                             get_hb_lb(analog_out[start_register + i - 1], &hb, &lb);
234                                                             data_array[j] = hb;
235                                                             j++;
236                                                             data_array[j] = lb;
237                                                             j++;
238                                                     }
239
240                                                     send_task(READ_AO, ((2 * no_of_register) + 1), data_array);
241                                             }
242                                             else
243                                             error_label = -1;
244                                     }
245                                     else
246                                     error_label = -1;
247                             }
248                             else
249                             error_label = -1;
250
251                             if(error_label == -1)
252                             {
253                                     data_array[0] = 0x02;
254                                     send_task(READ_AO | 0x80, 1, data_array);
255                             }
256                             break;
257                     }
258
259                     case READ_AI:
260                     {
261                             start_register = make_int(rx_data[2],rx_data[3]);
262                             no_of_register = make_int(rx_data[4],rx_data[5]);
263                             crc = make_int(rx_data[7],rx_data[6]);
264
265                             uchar hb = 0x00;
266                             uchar lb = 0x00;
267
268                             if((start_register >= 0) && (start_register <= 7))
269                             {
270                                     if((no_of_register <= 8) && ((start_register + no_of_register -1) <= 7))
271                                     {
272                                             if(crc == check_crc(6, rx_data))
273                                             {
274                                                     data_array[0] = (2 * no_of_register);
275
276                                                     int j = 1;
277
278                                                     for (int i = 1; i < no_of_register + 1; i++)
279                                                     {
280                                                             hb = 0x00;
281                                                             lb = 0x00;
282
283                                                             get_hb_lb(analog_in[start_register + i - 1], &hb, &lb);
284                                                             data_array[j] = hb;
285                                                             j++;
286                                                             data_array[j] = lb;
287                                                             j++;
288                                                     }
289
290                                                     send_task(READ_AI, ((2 * no_of_register) + 1), data_array);
291                                             }
292                                             else
```

```
293                                             error_label = -1;
294                                     }
295                                     else
296                                     error_label = -1;
297                             }
298                             else
299                             error_label = -1;
300
301                             if(error_label == -1)
302                             {
303                                     data_array[0] = 0x02;
304                                     send_task(READ_AI | 0x80, 1, data_array);
305                             }
306                             break;
307                     }
308
309                     case WRITE_SINGLE_DO:
310                     {
311                             start_register = make_int(rx_data[2],rx_data[3]);
312                             state_digital_out = make_int(rx_data[4],rx_data[5]);
313                             crc = make_int(rx_data[7],rx_data[6]);
314                             uchar temp_state = 0x01;
315
316                             if((start_register >= 0) && (start_register <= 31))
317                             {
318
319                                     if((state_digital_out == DO_SET_ON) ||(state_digital_out == DO_SET_OFF))
320                                     {
321
322                                             if(crc == check_crc(6, rx_data))
323                                             {
324
325                                                     if(state_digital_out == DO_SET_ON)
326                                                     {
327
328                                                             if(start_register > 0x07)
329                                                             {
330                                                                     temp_state = start_register - 8;
331                                                                     digital_out[1] |= temp_state;
332                                                             }
333                                                             if(start_register > 0x0F)
334                                                             {
335                                                                     temp_state <<= start_register - 16;
336                                                                     digital_out[2] |= temp_state;
337                                                             }
338                                                             if(start_register > 0x17)
339                                                             {
340                                                                     temp_state <<= start_register - 24;
341                                                                     digital_out[3] |= temp_state;
342                                                             }
343                                                             else
344                                                             {
345                                                                     temp_state <<= start_register;
346                                                                     digital_out[0] |= temp_state;
347                                                             }
348
349                                                     }
350                                                     else
351                                                     {
352                                                             if(start_register > 0x07)
353                                                             {
354                                                                     temp_state <<= start_register - 8;
355                                                                     digital_out[1] &= (~temp_state);
356                                                             }
357                                                             if(start_register > 0x0F)
358                                                             {
359                                                                     temp_state <<= start_register - 16;
360                                                                     digital_out[2] &= (~temp_state);
361                                                             }
362                                                             if(start_register > 0x17)
363                                                             {
364                                                                     temp_state <<= start_register - 24;
365                                                                     digital_out[3] &= (~temp_state);
```

```
366                                                     }
367                                                     else
368                                                     {
369                                                             temp_state <<= start_register;
370                                                             digital_out[0]&= (~temp_state);
371                                                     }
372                                             }
373
374                                             data_array[0] = rx_data[2];
375                                             data_array[1] = rx_data[3];
376                                             data_array[2] = rx_data[4];
377                                             data_array[3] = rx_data[5];
378
379                                             is_change_out = 1;
380
381                                             send_task(WRITE_SINGLE_DO, 4, data_array);
382                                     }
383                                     else
384                                     error_label = -1;
385                             }
386                             else
387                             error_label = -1;
388                     }
389                     else
390                     error_label = -1;
391
392                     if(error_label == -1)
393                     {
394                             data_array[0] = 0x02;
395                             send_task(WRITE_SINGLE_DO | 0x80, 1, data_array);
396                     }
397
398                     break;
399             }
400
401             case WRITE_SINGLE_AO:
402             {
403                     start_register = make_int(rx_data[2],rx_data[3]);
404                     state_analog_out = make_int(rx_data[4],rx_data[5]);
405                     crc = make_int(rx_data[7],rx_data[6]);
406
407                     if((start_register >= 0) && (start_register <= 7))
408                     {
409                             if(crc == check_crc(6, rx_data))
410                             {
411                                     analog_out[start_register]= state_analog_out;
412
413                                     data_array[0] = rx_data[2];
414                                     data_array[1] = rx_data[3];
415                                     data_array[2] = rx_data[4];
416                                     data_array[3] = rx_data[5];
417
418                                     send_task(WRITE_SINGLE_AO, 4, data_array);
419                             }
420                             else
421                                     error_label = -1;
422                     }
423                     else
424                             error_label = -1;
425
426                     if(error_label == -1)
427                     {
428                             data_array[0] = 0x02;
429                             send_task(WRITE_SINGLE_AO | 0x80, 1, data_array);
430                     }
431
432                     break;
433             }
434
435             case WRITE_MULTIPLE_DO:
436             {
437                     start_register = make_int(rx_data[2],rx_data[3]);
438                     no_of_register = make_int(rx_data[4],rx_data[5]);
```

```
439                             uint no_of_bytes = rx_data[6];
440                             crc = make_int(rx_data[8 + no_of_bytes],rx_data[7 + no_of_bytes]);
441
442                             if((start_register >= 0) && (start_register <= 31))
443                             {
444                                     if((no_of_register <= 32) && ((start_register + no_of_register -1) <= 31))
445                                     {
446                                             if(crc == check_crc(7 + no_of_bytes, rx_data))
447                                             {
448                                                     for(int i = 0; i < no_of_register; i++)
449                                                     {
450                                                             digital_out[(start_register + i) / 8] &= ~(0b00000001 << (
451                                                             digital_out[(start_register + i) / 8] |= (((rx_data[7 + (i
452                                                     }
453
454                                                     data_array[0] = rx_data[2];
455                                                     data_array[1] = rx_data[3];
456                                                     data_array[2] = rx_data[4];
457                                                     data_array[3] = rx_data[5];
458
459                                                     send_task(WRITE_MULTIPLE_DO, 4, data_array);
460
461                                                     is_change_out = 1;
462                                             }
463                                             else
464                                             error_label = -1;
465                                     }
466                                     else
467                                     error_label = -1;
468                             }
469                             else
470                             error_label = -1;
471
472                             if(error_label == -1)
473                             {
474                                     data_array[0] = 0x02;
475                                     send_task(WRITE_MULTIPLE_DO | 0x80, 1, data_array);
476                             }
477                             break;
478                     }
479
480                     case WRITE_MULTIPLE_AO:
481                     {
482                             start_register = make_int(rx_data[2],rx_data[3]);
483                             no_of_register = make_int(rx_data[4],rx_data[5]);
484                             uint no_of_bytes = rx_data[6];
485                             crc = make_int(rx_data[8 + no_of_bytes],rx_data[7 + no_of_bytes]);
486
487                             if((start_register >= 0) && (start_register <= 7))
488                             {
489                                     if((no_of_register <= 8) && ((start_register + no_of_register -1) <= 7))
490                                     {
491                                             if(crc == check_crc(7 + no_of_bytes, rx_data))
492                                             {
493                                                     for(int i = 0; i < no_of_register;i++)
494                                                             analog_out[i + start_register] = make_int(rx_data[7 + i], r
495
496                                                     data_array[0] = rx_data[2];
497                                                     data_array[1] = rx_data[3];
498                                                     data_array[2] = rx_data[4];
499                                                     data_array[3] = rx_data[5];
500                                                     send_task(WRITE_MULTIPLE_AO, 4, data_array);
501                                             }
502                                             else
503                                             error_label = -1;
504                                     }
505                                     else
506                                     error_label = -1;
507                             }
508                             else
509                             error_label = -1;
510
511                             if(error_label == -1)
```

```
512                                 {
513                                         data_array[0] = 0x02;
514                                         send_task(WRITE_MULTIPLE_AO | 0x80, 1, data_array);
515                                 }
516                                 break;
517                         }
518
519                         default:
520                         {
521                                 data_array[0] = 0x01;
522                                 send_task(rx_data[FUNCTION_BYTE] | 0x80, 1, data_array);
523                                 break;
524                         }
525                 }
526         }
527
528         flush_rx_data();
529 }
530
531 void send_task(uchar function_code, uchar data_length, uchar *data)
532 {
533         uchar crc_hb = 0x00;
534         uchar crc_lb = 0x00;
535         uint lenght_sended_data = 0;
536
537         flush_tx_data();
538
539         tx_data[ID_BYTE] = SLAVE_ID;
540         tx_data[FUNCTION_BYTE] = function_code;
541
542         for(int i = 0; i < data_length; i++)
543         {
544                 tx_data[2+i] = data[i];
545         }
546
547         get_hb_lb(check_crc((data_length + NO_OF_STATIC_BYTES),tx_data), &crc_hb, &crc_lb);
548
549         tx_data[DATA_START_BYTE + data_length] = crc_lb;
550         tx_data[DATA_START_BYTE + data_length + 1] = crc_hb;
551
552         lenght_sended_data = NO_OF_STATIC_BYTES + data_length + NO_OF_CRC_BYTE + 2;
553
554         // Sending response
555
556         write(TEN, 1);
557         UCSR0B &= ~(1<<RXEN0);
558
559         for (int i = 0; i < lenght_sended_data; i++)
560         {
561                 while(!(UCSR0A &  (1<<UDRE0)));
562                 UDR0 = tx_data[i];
563                 while(!(UCSR0A &  (1<<TXC0)));
564         }
565
566         write(TEN, 0);
567         UCSR0B |= 1<<RXEN0;
568
569 }
570
```

```
 1
 2 /* ---------------------------------------------------------------------------
 3 *
 4 *        IO Module with Modbus Communication
 5 * ---------------------------------------------------------------------------
 6 *
 7 *        Target : ATmega328p
 8 *        Clock  : Xtal 16.0 MHz
 9 *
10 *        Author : Tomáš Viktora
11
12 */
13
```

```
14 #include <avr/io.h>
15 #include <avr/interrupt.h>
16 #include <util/delay.h>
17 #include <stdio.h>
18 #include <string.h>
19 #include "ioctrl.h"
20 #include "iodef.h"
21 #include "std.h"
22 #include "MODBUS.h"
23
24
25 ISR(USART_RX_vect)
26 {
27         rx_byte = UDR0;
28         rx_data[cnt_rx] = rx_byte;
29
30         if((cnt_rx == 1) && ((rx_byte == 0x0F) || (rx_byte == 0x10)))
31                 cahnge_data_lenght = 1;
32
33         if((cnt_rx == 6) && (cahnge_data_lenght == 1))
34         {
35                 cahnge_data_lenght = 0;
36                 modbus_data_lenght = 9 + rx_byte;
37         }
38
39         cnt_rx++;
40
41         if (cnt_rx == modbus_data_lenght)
42         {
43                 check_rx_data = 1;
44         }
45 }
46
47 // 115200bps/16MHz
48 void uart_init()
49 {
50         UBRR0H = 0;
51         //115200
52         UBRR0L = 16;
53         // double speed
54         UCSR0A = 1<<U2X0;
55         // 8bit
56         UCSR0C = 1<<UCSZ01 | 1<<UCSZ00;
57         // enable receiver, rx interrupt
58         UCSR0B = 1<<RXEN0 | 1<<TXEN0 | 1<<RXCIE0; // | 1<<TXCIE0;
59 }
60
61 void io_init()
62 {
63         setin(IN1);
64         setin(IN2);
65         setin(IN3);
66         setin(IN4);
67         setout(OUT1);
68         setout(OUT2);
69         setout(OUT3);
70         setout(OUT4);
71         setout(TEN);
72         setin(INT_I);
73         setout(INT_O);
74         setin(IN_MCU);
75         setin(SW1);
76
77         setin(RX);
78         setout(TX);
79
80         no_of_inputs = 4;
81         no_of_outputs = 4;
82         last_in_state = 0x00;
83         is_change_out = 0;
84         check_rx_data = 0;
85         modbus_data_lenght = 8;
86
```

```
87 }
88
89 void clear_register()
90 {
91         for(int i = 0; i < 4;i++)
92         {
93                 digital_in[i] = 0x00;
94                 digital_out[i] = 0x00;
95         }
96
97         for(int i = 0; i < 8;i++)
98         {
99                 analog_in[i] = 0x0000;
100                analog_out[i] = 0x0000;
101        }
102 }
103
104 void check_inputs()
105 {
106        uchar temp = (PIND >> 3) & 0b00001111;
107
108        if(last_in_state != temp)
109        {
110                last_in_state = temp;
111                digital_in[0] = (digital_in[0] & 11110000) | ((~temp) & 0b00001111);
112        }
113 }
114
115 void check_output_state()
116 {
117        uchar temp = 0x00;
118
119        if(is_change_out)
120        {
121                for(int i = 0; i < 4; i++)
122                {
123                        temp = (digital_out[0] >> i) & 0b00000001;
124
125                        switch (i)
126                        {
127                                case 0:
128                                {
129                                        write(OUT1,temp);
130                                        break;
131                                }
132                                case 1:
133                                {
134                                        write(OUT2,temp);
135                                        break;
136                                }
137                                case 2:
138                                {
139                                        write(OUT3,temp);
140                                        break;
141                                }
142                                case 3:
143                                {
144                                        write(OUT4,temp);
145                                        break;
146                                }
147                                default:
148                                        break;
149                        }
150                }
151
152                is_change_out = 0;
153        }
154 }
155
156
157 int main(void)
158 {
159        io_init();
```

```
160         uart_init();
161         clear_register();
162
163         sei();
164         //write(TEN, 1);
165
166         while (1)
167         {
168                 if (check_rx_data)
169                 {
170                         check_rx_data = 0;
171                         modbus_data_lenght = 8;
172                         recieve_task();
173                 }
174
175                 check_inputs();
176                 check_output_state();
177
178         }
179 }
180
```