VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

ROZŠIŘUJÍCÍ VÝUKOVÝ MODUL K MIKROKONTROLÉRU ATMEGA

EXTENSION EDUCATIONAL MODULE TO A MICROCONTROLLER ATMEGA

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — PETR NOVÁK
AUTHOR

VEDOUCÍ PRÁCE — ING. DANIEL ZUTH, PH.D.
SUPERVISOR

BRNO 2014

ZADÁNÍ ZÁVĚREČNÉ PRÁCE

(na toto místo vložte originál nebo kopii Vaší práce)

ABSTRAKT

Tato bakalářská práce se zabývá návrhem, realizací a navržením vzorových příkladů ke zvoleným perifériím. Práce bude sloužit jako dodatkový výukový modul k základové desce s mikrokontrolérem ATmega 128 pro předmět Mikroprocesorová technika. Podrobně je zde rozebrán popis návrhu, který byl proveden ve vývojovém prostředí EAGLE 5.4.0. Realizace a ověření funkčnosti byly zajištěny pomocí vytvořených programů ve vývojovém prostředí Atmel AVR studio 6.1.

ABSTRACT

This bachelor thesis is concerning with design, realization and proposing samples of exemplars for chosen peripherals. The thesis will serve as additional education module for motherboard with microcontroller ATmega 128, it will be used in subject called Microprocessor technology. It will be dealing there with description of design, which was made in integrated development environment called EAGLE 5.4.0. Realization and verification of functionality were done with created programs in integrated development environment Atmel AVR studio 6.1.

KLÍČOVÁ SLOVA

Výukový modul, ATmega 128, DPS, RFID

KEYWORDS

Education module,ATmega 128, DPS, RFID

PROHLÁŠENÍ O ORIGINALITĚ

Prohlašuji, že jsem tuto práci vypracoval sám, bez cizí pomoci pouze na základě použité literatury.

BIBLIOGRAFICKÁ CITACE

NOVÁK, P. Rozšiřující výukový modul k mikrokontroléru ATmega. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2014. 44 s. Vedoucí bakalářské práce Ing. Daniel Zuth, Ph.D.

PODĚKOVÁNÍ

Tímto bych chtěl velmi poděkovat vedoucímu mé bakalářské práce panu Ing. Danielovi Zuthovi Ph.D. za vedení a rady, které mi poskytl v průběhu vypracovávání.

1. ÚVOD

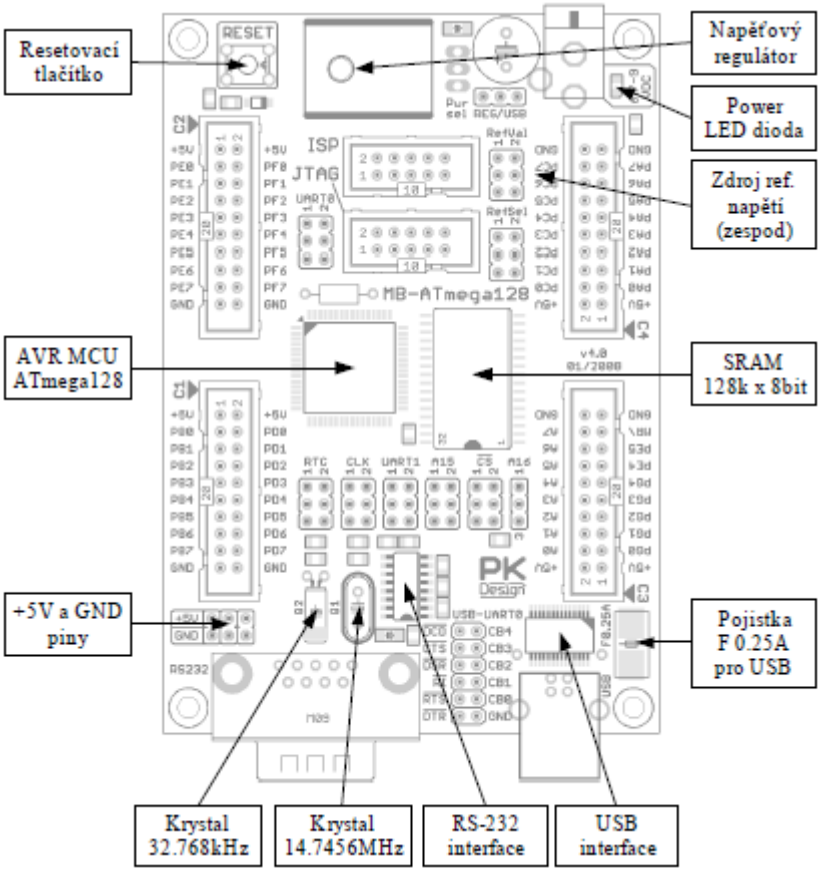
Cílem této práce je seznámení se s mikrokontroléry a jejich různými periferiemi, které budeme využívat. Tyto znalosti budeme následně aplikovat pro realizaci našeho rozšiřujícího modulu, který bude využíván v předmětu Hardware a mikroprocesorová technika.

Úkolem tohoto rozšiřujícího modulu bude seznámit studenty s principy komunikace přes UART rozhranní díky čtečce čipů, kde si budou moct vyzkoušet čtení ID karet. Také zde budou mít možnost vyzkoušet si převod z analogového spojitého signálu na digitální diskrétní hodnotu pomocí AD převodníku, tuto možnost budou poskytovat jak potenciometr, tak i analogový teploměr. Dále je zde digitální teploměr, pomocí kterého si budou moct vyzkoušet komunikaci přes 1-wire sběrnici a také si vyzkoušet princip funkce rotačního enkóderu, kde se řeší přerušení na pinech.

Teoretická část této práce se zabývá popisem jednotlivých komponentů a jejich způsoby komunikace. V další části bude uveden postup výroby plošného spoje. Nakonec budou v softwarové části podrobně rozebrány jednotlivé funkce.

2. VÝVOJOVÝ KIT MB-ATMEGA 128

Na začátek vás seznámím se základovou deskou a mikrokontrolerem, který jsme v naší práci využili. Tato deska je využívaná převážně pro výukové účely pro práci s 8-bitovým RISC mikrokontrolérem s maximálním výkonem 16MIPS. Pro maximální využití této desky již existuje rozmanitá sada rozšiřujících modulů, která se stále doplňuje o nové moduly. Deska neobsahuje žádné periferní obvody, které by byly napojeny přímo k obvodu MCU, což umožňuje, aby se celý systém dal zapojit dle našich představ. Na obrázku č. 1. lze vidět periferní zapojení desky.[1]



Obr. č. 1 Periférie vývojového kitu. [1]

2.1. Vlastnosti vývojového kitu

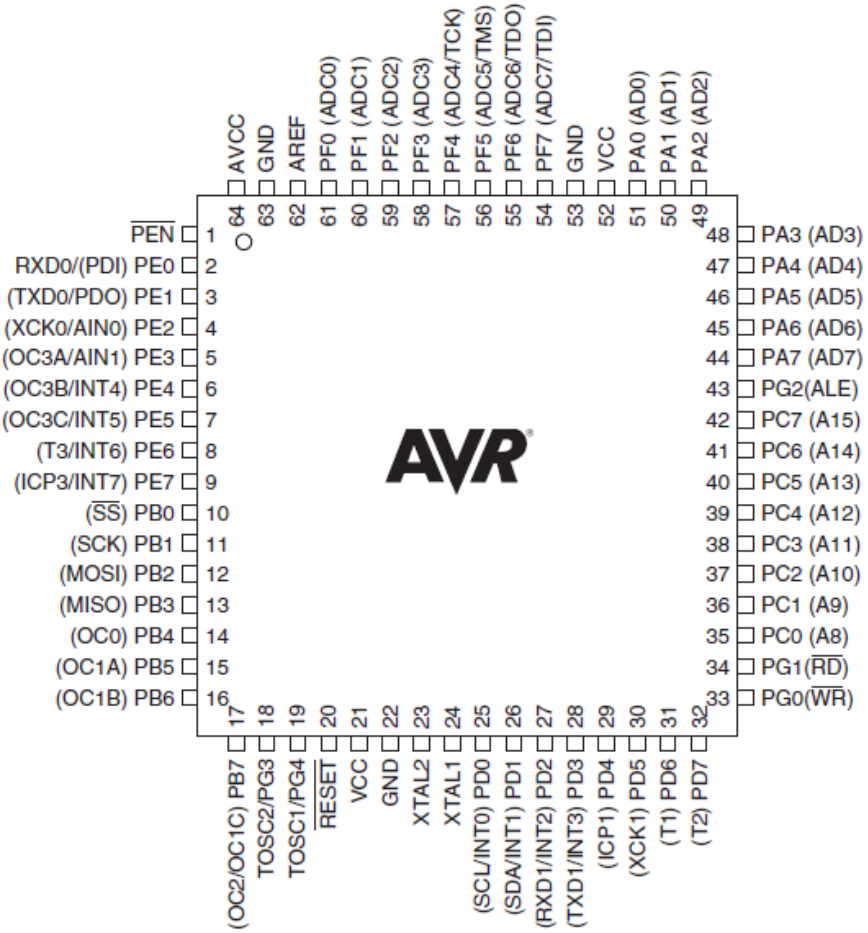
- deska obsahuje RISC-ový mikrokontrolér Atmel ATmega 128-16AU TQFP64
- napájecí napětí +5V je produkováno napěťovým regulátorem na desce
- asynchronní SRAM paměť 128k x 8bit, 55ns
- nastavitelná paměťová reference pro interní AD převodník, také možnost připojení externí paměťové reference
- deska je dávána s krystalem 14,7456MHz
- možnost resetování mikrokontroléru pomocí tlačítka RESET
- rozměry plošného spoje (v x š x d): 25mm x 74mm x 107m[1]

2.2. Vlastnosti mikrokontroléru

- 8-bitový RISC mikrokontrolér
- 32 osmibitových registrů
- 4kB interní SRAM paměti
- 8-kanálový 10-bitový A/D převodník
- JTAG rozhraní s možností programování a ladění
- dvě programovatelné USART komunikace
- master/slave SPI sériové rozhraní
- programovatelný Watch-dog časovač
- 53 programovatelných I/O vývodů
- napájecí napětí 4,5-5,5V[1]

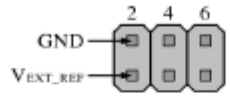
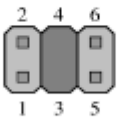
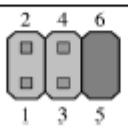
2.3. Nastavení vývojového kitu

Mikrokontrolér má šest vstupně/výstupní 8-bitových portů označené PAx až PFx a jeden 5-bitový port PGx, kde x nám symbolizuje číslo bitu portu které jsou v rozsahu 0-7, jak je vidět na obrázku č. 2. Pro naše účely jsme využili porty PA/PC pro připojení modelu s LCD displejem a porty PE/PF pro připojení našeho vyrobeného modulu.[2]



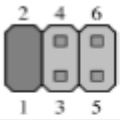
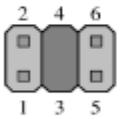
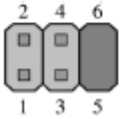
Obr. č. 2 Rozmístění pinů na mikrokontroléru. [2]

Naše moduly jsou na tyto piny připojeny pomocí konektoru MLW20, který je shodný jak pro naše moduly, tak i pro vývojový kit. V našem případě využíváme rozšiřující konektory CON4 pro připojení LCD modulu a CON2 pro připojení našeho modulu. Dále jsme museli nastavit JP1 pomocí jumperu, kterým jsme nastavili napěťovou referenci pro A/D převodník na piny 5 a 6 dle obr. č. 3. [1]

JP1	*Zobrazení*	*Funkce*
Bez propojky		Zdrojem napěťové reference je externí zdroj referenčního napětí připojený mezi vývody 1 (V_{EXT_REF}) a 2 (GND).
3 – 4		Zdrojem napěťové reference je výstup nastavitelného referenčního zdroje (TLV431 s JP9) umístěného na základové desce.
5 – 6		Zdrojem napěťové reference je napájecí napětí V_{CC} (oddělené LC filtrem).

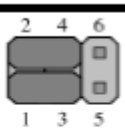
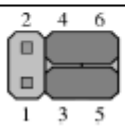
Obr. č. 3 Nastavení napěťové reference. [1]

Vývojový kit je připojen přes CON6, tento konektor slouží pro jeho programování přes ISP rozhraní. Referenční napětí jsme nastavili na V_{REF} = 2,5V, které se opět nastavuje pomocí jumperu dle obr. č. 4.[1]

JP9	*Zobrazení*	*Funkce*
1 – 2		V_{REF} = 1.25V
3 – 4		V_{REF} = 2.5V
5 – 6		V_{REF} = 4.096V

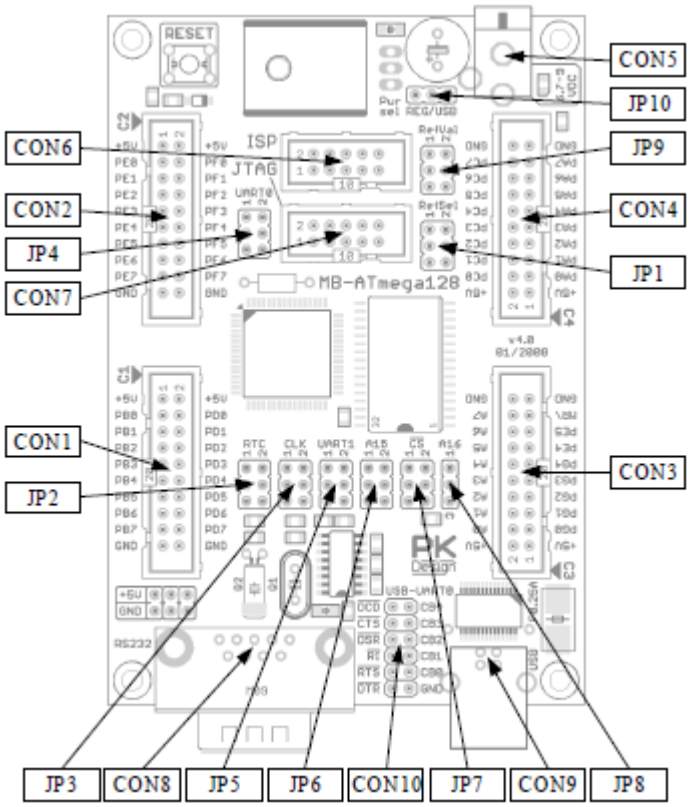
Obr. č. 4 Nastavení referenčního napětí. [1]

Bylo třeba také nastavit JP4, který nám slouží pro nastavení vývodů UART0 komunikace buď přes USB nebo přes CON2. Nastavení jumperů je na obr. č. 5. a odpovídá tomu zapojení přes piny 1-3 a 2-4.[1]

JP4	*Zobrazení*	*Funkce*
1 – 3 2 – 4		Vývody PE0M a PE1M jsou připojeny na příslušné vývody rozšiřujícího konektoru CON2.
3 – 5 4 – 6		Vývod PE0M mikrokontroleru je připojen na vývod RxD-0 obvodu FT232 (USB), vývod PE1M mikrokontroleru je připojen na vývod TxD-0 obvodu FT232 (USB).

Obr. č. 5 Nastavení UART0 vývodů. [1]

Rozmístění všech konektorů a propojek na vývojovém kitu je zobrazeno na obr. č. 6.



Obr. č. 6 Rozmístění konektorů a propojek. [1]

3. HARDWARE MODULU

3.1. Čtečka čipů RFID

RFID (Rádio frekvenční identifikace) je bezkontaktní identifikace, která nám slouží k přenosu nebo také ukládání dat pomocí elektromagnetických vln. Tyto systémy jsou schopny:

- zaznamenávat
- uchovávat
- nebo poskytovat informace v reálném čase[3]

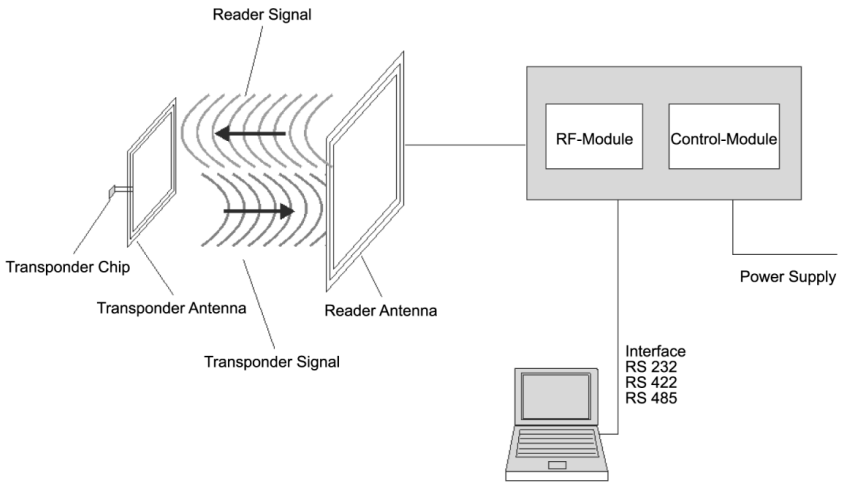
3.1.1. RFID systém

Tento systém se skládá ze tří základních komponent:

- tzv. RFID tag, který se skládá z čipu, cívky nebo antény, tyto tagy se dělí na aktivní nebo pasivní, rozdíl v nich je ten, že pasivní mají vlastní zdroj energie jako např. baterii
- tzv. RFID reader, který disponuje vysílacím/přijímacím obvodem s dekodérem a anténou. Někdy může být čtečka vybavena vlastním operačním systémem
- řídící software, který se dělí na podpůrné systémy a systémy na strategické úrovni řízení[3]

3.1.2. Princip RFID

RFID systémy pracují na principu rádiových vln, které pracují v různých vlnových délkách. Určujícím parametrem pro čtecí dosah a interakci s okolním prostředím je pracovní kmitočet. Čím vyšší frekvence, tím je rychlejší přenos dat, a taky zároveň delší vzdálenost na kterou je schopna RFID komunikovat s RFID tagem. Nicméně je zvětšena citlivost na přítomnost nežádoucích materiálů jako např. uhlík, kov nebo kapaliny, které výrazně ovlivňují šíření rádiových vln. Princip je znázorněn na obr č. 7.[3]



Obr. č. 7 Princip činnosti pasivního RFID. [3]

3.1.3. RFID pásma

Jsou zde čtyři hlavní frekvenční pásma:

- LowFrequency pásmo (125-134kHz)
  - velmi malá čtecí vzdálenost (cca 20cm) a nízká přenosová rychlost
  - převážně se využívá v identifikačních průkazech
  - kov a kapaliny nemají vliv na signál
- HighFrequency pásmo (13,56MHz)
  - čtecí vzdálenost až do 1 metru
  - dostatečná přenosová rychlost
  - čipy ve dvou provedeních „*Pouze čtení*“, nebo „*Čtení a zápis“* s kapacitou až po kilobyty
  - nejčastější využívané u docházkových systémů
- Ultra HighFrequency pásmo (860-960MHz)
  - čtecí dosah max. 7m
  - dnes nejrozšířenější pásmo pro identifikaci zboží
  - vysoká přenosová rychlost
- Microwave pásmo (2,45-5,8GHz)
  - čtecí dosah až 20m
  - nejvyšší přenosový rychlost
  - signál je extrémně pohlcován kapalinami
  - možnost kolize s Wi-Fi sítí[3]

3.1.4. RFID EM-18

Tato čtečka může komunikovat dvěma způsoby a to buď pomocí komunikace Wiegand 26 nebo RS 232, dále tato čtečka disponuje pinem SEL (select), který nám umožňuje výběr mezi výše zmíněnými komunikacemi. Dále je k této čtečce připojena dioda, která nám signalizuje zaznamenání RFID tagu.

3.1.5. Vlastnosti čtečky:

- napájecí napětí 5V
- elektrický proud < než 50mA
- frekvence 125kHz
- čtecí vzdálenost 10cm[4]

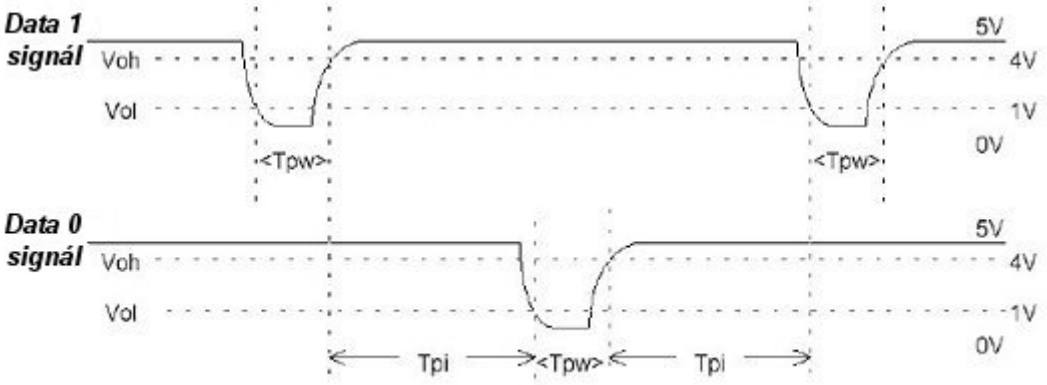
3.1.6. Wiegand protokol

Přenos je veden přes dva datové vodiče D0 a D1. Komunikace je sekvenční a bity se přenáší postupně. Přenos probíhá vysíláním krátkých impulsů o dané délce, když je tento

krátký impuls vyslán na D1, tak signál přenáší bit s hodnotou 1. Analogicky k tomuto platí pro D0, takže pokud je zde vyslán krátký impuls, tak je přenesen bit s hodnotou 0.

Tyto impulsy se opakují podle předem daného počtu impulsů (bitů) a z toho získáme naši proměnnou. Ukázka doby impulzů je na obr. č. 8.[5]

Symbol	Popis	Typické doby trvání
Tpw	**Doba pulsu**	50µs
Tpi	**Doba pausy**	2ms



Obr. č. 8 Vzor délky impulzů. [5]

Protokol se skládá z 8 bitů (Facility code), 16 bitů dat z karty a dvou paritních bitů, tyto bity nám zobrazuje tabulka č. 1.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
MSB	FC	FC	FC	FC	FC	FC	FC	FC	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	LSB

Tabulka č. 1. Pořadí bitů ve Wiegand protokolu[5]

MSB bit je paritní bit pro prvních 12 bitů, zbývajících 12 bitů má paritní bit LSB Pro MSB bit platí, že je nulový, pokud je počet jedniček v první části protokolu sudý, naopak pokud je počet jedniček lichý, tak je bit MSB nastaven. Bit LSB je nastavován/nulován opačně, je-li počet jedniček v druhé části telegramu lichý, tak je LSB nulový a pokud je počet jedniček sudý pak je LSB bit nastaven.[5]

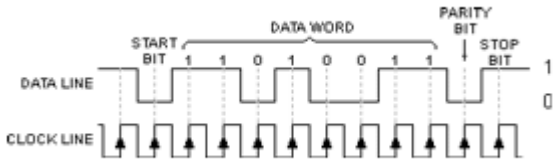
3.1.7. RS 232

Vlastnosti:

- rychlost přenosu dat 9600bd
- počet datových bitů: 8
- bez parity

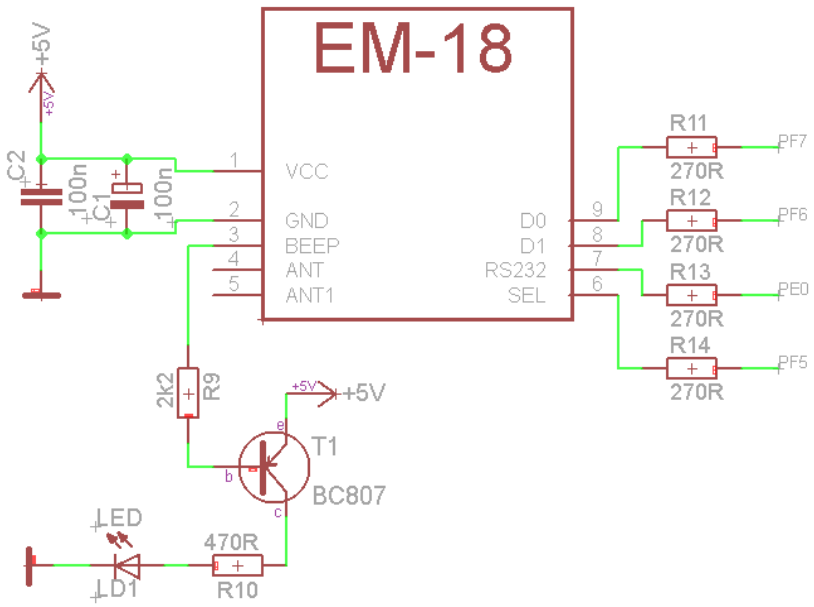
- 1 stop bit
- 5 bytů (id karty) + 1 byte (CRC – XOR)[4]

Využívá asynchronní komunikace pro přenos dat, což znamená, že jednotlivé znaky mohou být přeneseny s libovolným časovým odstupem mezi sebou. Díky tomuto různému časovému rozestupu se nedá určit, kde začíná další znak, a proto musíme být schopni určit příchod nového znaku. Toto ošetření je zajištěno tzv. Start-bit, kterým začíná každý nový znak. Po tomto znaku už následují datové bity, kterých je obvykle 8, následně zde může být paritní bit a znak je ukončen 1 nebo více tzv. Stop bit. Tento způsob přenosu znaků je znázorněn na obr. č. 9.[6]



Obr. č. 9 Asynchronní přenos znaků.[6]

Tato komunikace využívá dvě napěťové komunikace a to logickou 1 a logickou 0. Log. 1 je indikována zápornou hodnotou, a analogicky k tomu je Log. 0 přenášena kladnou úrovní výstupních vodičů. V našem případě jsou logické úrovně přenášeny ±5V. Základní komunikace probíhá přes dva datové signály a zem. Tyto signály jsou označeny RXD a TXD. Pro nás je důležitý pouze signál RXD, kterým komunikuje čtečka s počítačem, protože my pouze přijímáme data ze čtečky, žádné data do ní neposíláme. Schématické zapojení je na obr. č. 10.[6]



Obr. Č. 10 Schematické zapojení čtečky RFID EM-18.

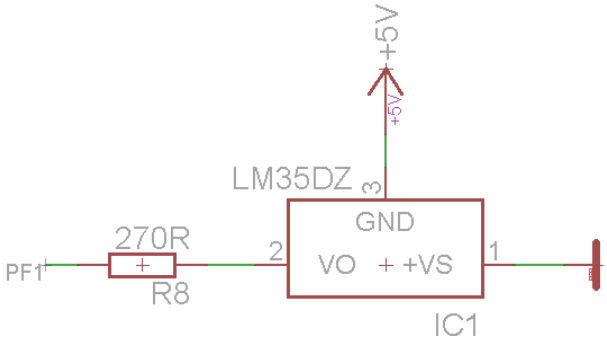
3.2. Teploměr LM35DZ

3.2.1. Vlastnosti teploměru

- napájecí napětí 4 až 30V
- kalibrovaný přímo v °C
- teplotní rozsah -55 až 150°C
- lineární závislost napětí na teplotě[7]

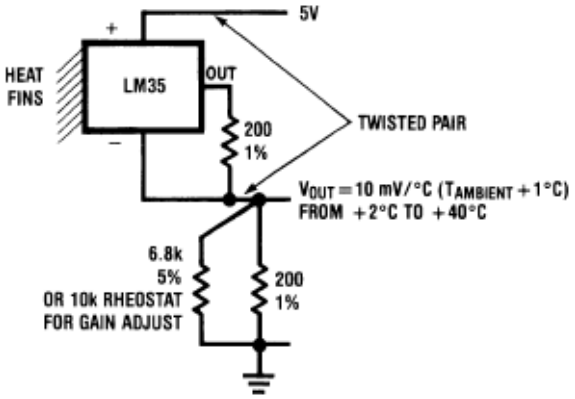
3.2.2. Popis teploměru

Použili jsme analogový teploměr LM35DZ s plastovým obalem TO-92Je to přesný snímač teploty s integrovaným obvodem, jehož výstup ve voltech je lineárně úměrný stupňům Celsia a to +10,0mV/°C. Nevyžaduje žádnou externí kalibraci, aby poskytl přesnost typickou pro tyto snímače a to je ±0,25°C pro měření pokojové teploty a ±0,75°C pro maximální měřené rozmezí. Pro napájení může být použit buď jeden zdroj napětí, nebo plusové a mínusové napájení. Vzhledem k tomu že teploměr odebírá pouze 60µA ze svého zdroje, tak se velmi málo zahřívá a to méně než 0,1°C v prostředí bez proudění. Na obr. č. 11 je vidět naše schématické zapojení tohoto teploměru.[7]

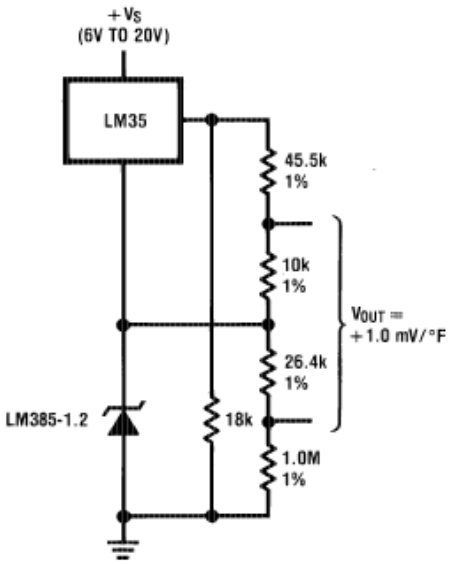


Obr. č. 11 Schematické zapojení teploměru LM35DZ.

Další možné způsoby zapojení tohoto teploměru jsou na obr. č. 12 a 13.

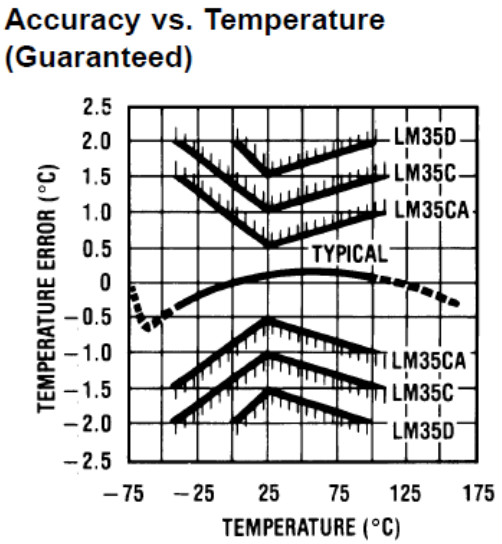


Obr. č. 12 Zapojení pomocí dvou drátů.[7]



Obr. č. 13 Zapojení pro měření teploty ve stupních Fahrenheit.[7]

LM35DZ může být aplikován k povrchu stejným způsobem jako jiné teploměry s integrovaným obvodem a to přilepením nebo tmelením a jeho teplota bude okolo 0,01°C povrchové teploty, pokud je teplota povrchu skoro stejná jako teplota vzduchu. Je-li ovšem teplota vzduchu mnohem větší nebo nižší, tak aktuální teplota LM35 bude prostřední teplota mezi teplotou povrchu a teplotou vzduchu. Toto platí zejména pro obal TO-92, kde jsou měděné vodiče hlavním přenašeč teploty do zařízení, takže jejich teplota může být blíže teplotě vzduchu než teplotě povrchu.Na obr. č. 14 je znázorněna přesnost teploměru pro různé obaly v závislosti na teplotě a teplotní odchylce.[7]



Obr. č. 14 Přesnost teploměru v závislosti teploty na teplotní odchylce.[7]

3.3. Teploměr DS18B20

3.3.1. Vlastnosti teploměru

- napájecí napětí 3 až 5,5V
- 1-wire připojení
- unikátní 64-bitový sérový kód
- rozsah měření od -55°C do +125°C (-67°F do + 257°F)
- přesnost v rozsahu od -10°C do + 85°C je ±0,5°C[8]

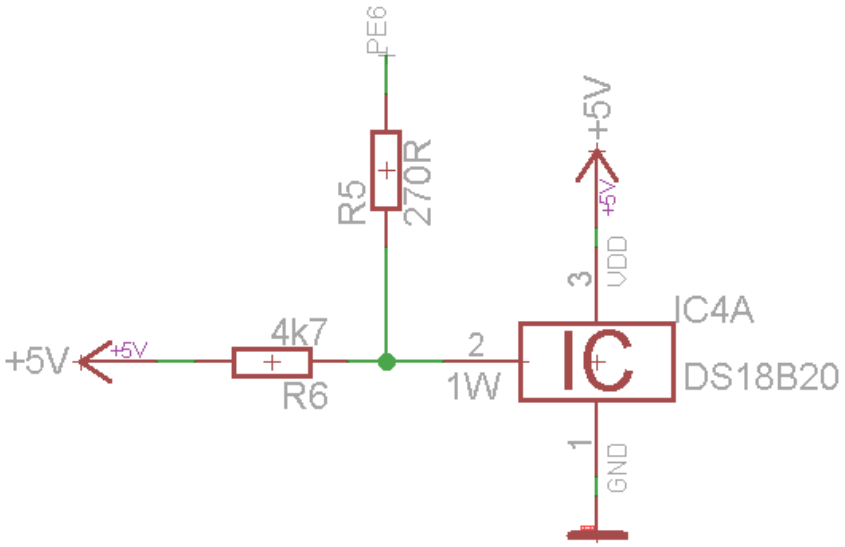
3.3.2. Popis teploměru

Tento digitální teploměr komunikuje přes 1-wire sběrnici se kterou komunikuje přes jednu datovou linku. DS18B20 může být také napájen přímo z datové linky pomocí tzv. „*parasite power*“, díky tomuto zapojení nepotřebujeme externí zdroj, ale pokud je tento teploměr takhle zapojen tak pin V_{DD} musí být uzemněn. Každý DS18B20 má vlastní unikátní 64-bitové sériové číslo, což umožňuje aby bylo více těchto teploměrů zapojených na jednu sběrnici na velké ploše. Toho se dá využívat např. u monitorování teploty v různých místnostech uvnitř budovy. Dále teploměr obsahuje „*Scratchpad memory“*, která obsahuje 2-bytový teplotní registr, který si ukládá naměřenou teplotu, 1-bitový registr pro spuštění alarmu při překročení spodní nebo horní hranice teploty a také registr pro konfiguraci rozlišení převodu teploty na digitální výstup a to na 9,10,11 nebo 12 bitů. Závislost teploty na digitálním výstupu je znázorněna v tabulce č. 2.[8]

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	0000 0111 1101 0000	07D0h
+85°C*	0000 0101 0101 0000	0550h
+25.0625°C	0000 0001 1001 0001	0191h
+10.125°C	0000 0000 1010 0010	00A2h
+0.5°C	0000 0000 0000 1000	0008h
0°C	0000 0000 0000 0000	0000h
-0.5°C	1111 1111 1111 1000	FFF8h
-10.125°C	1111 1111 0101 1110	FF5Eh
-25.0625°C	1111 1110 0110 1111	FE6Fh
-55°C	1111 1100 1001 0000	FC90h

Tabulka č. 2 Teplotní závislost na výstupu.[8]

V našem případě tento teploměr napájíme externím zdrojem, takže jsme využili standartní zapojení. Teploměr je nakonfigurován aby měřil ve °C, takže jsme ho nemuseli nijak nastavovat. Zapojení teploměru je na obr. č. 15. [8]



Obr. č. 15 Schematické zapojení teploměru DS18B20.[8]

3.4. Potenciometr PC1621

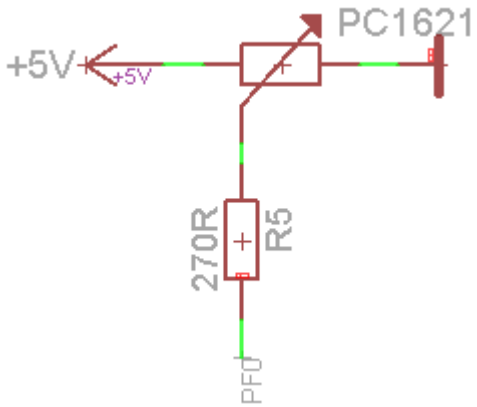
PC1621 je uhlíkový rotační potenciometr s kovovou drážkovanou osou otáčení. Potenciometr je nastavitelný rezistor, kde se velikost odporu mění vzhledem k pohybu potenciometru. Potenciometry mouho mít dva druhy pohybu a to buď lineární a nebo rotační což je náš případ. Potenciometr patří mezi odporové snímače polohy, tyto potenciometry mají několik vlastností a to :

- Rozlišovací schopnost
  - ta nám udává, jaký úhlový inkrement je náš potenciometr schopen spolehlivě rozlišit
- Linearita
  - to je největší odchylka vstupního napětí od vztažné přímky a udává se v procentech napájecího napětí
- Životnost
  - udává maximální možný počet otočení hřídelkou při zadaných provozních podmínkách
- Provozní krouticí moment
  - je největší možný krouticí moment v obou směrech otáčení
- Šum
  - vzniká při pohybu jezdce po vinutí a působí ho mechanické i elektrické efekty[9]

3.4.1. Vlastnosti potenciometru

- odpor 500k Ohm
- lineární průběh
- ztrátový výkon 0,125 W
- průměr osy 6 mm
- mechanický úhel otočení 300°
- elektrický úhel otočení 280°
- tolerance 20%[10]

Zapojení potenciometru je na obr. č. 16.



Obr. č. 16 Schematické zapojení potenciometru PC1621.

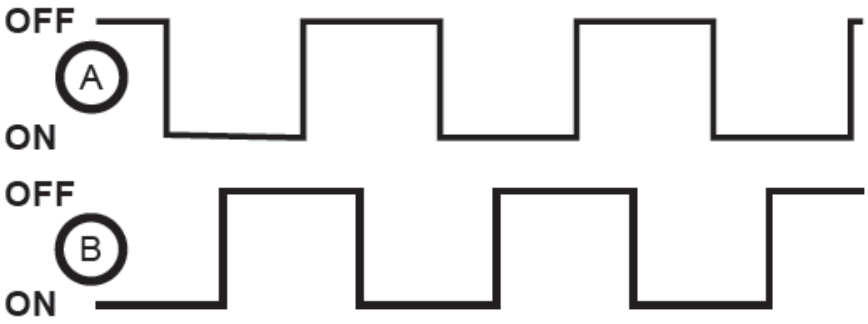
3.5. Rotační enkodér P-RE30S

Enkodér je elektromagnetický převodník pro převod rotačního pohybu na sekvence elektrických impulsů. Náš enkodér se řadí mezi inkrementální rotační snímače. Tyto snímače mohou mít několik provedení a to například jednokanálové u kterého jsou buď po obvodu koutouče nebo pravítka rovnoměrně rozmístěné značky a pomocí snímače přítomnosti se detekují tyto značky. Enkodér, který využíváme my je dvou kanálový, ten generuje dva obdelníkové průběhy, které jsou od sebe posunuty o 90 elektrických stupňů, tyto kanály jsou označovány A a B. Tento enkodér disponuje navíc ještě axiálním tlačítkem. Vzhledem k tomu, že všechny kontakty jsou mechanické, tak zde dochází k zákmitům. Z tohoto důvodu jsou zde na zem zapojeny kondenzátory.[11]

3.5.1. Vlastnosti

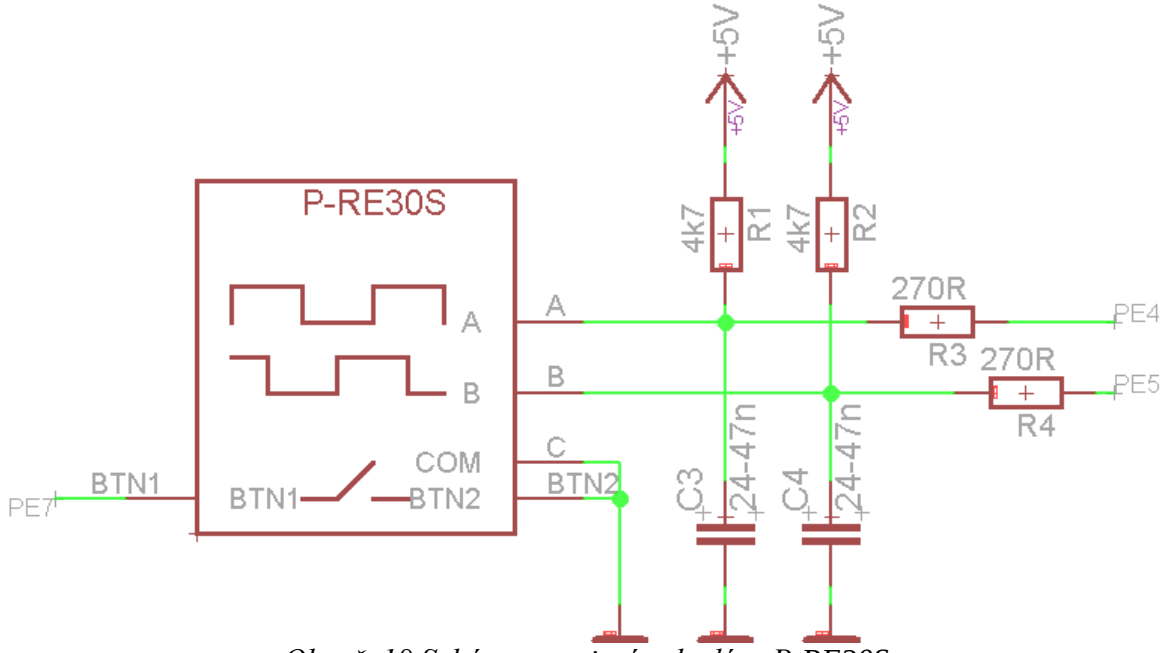
- elektrický proud max. 10mA
- počet kroků na 360° je 30
- maximální délka zákmitů kontaktů 5ms
- pracovní teplota -30 až +70°C
- životnost min. 15000 otočení[12]

Výstupní průběh můžeme vidět na obr. č. 17.



Obr. č. 17 Výstupní průběhy pro kanály A a B.[12]

Princip spočívá v tom, že při otočení se vyvolá přerušení na jednom pinu a my se ptáme, jaká je úroveň na druhém pinu a následně podle toho zda jsme dostali log.1 nebo log.0 přičteme/odečteme proměnnou. Schématické zapojení je na obr. č. 18.[11]



Obr. č. 18 Schéma zapojení enkodéru P-RE30S.

4. VÝROBA MODELU

Prvním krokem k výrobě bylo seznámení se s vývojovým prostředím, ve kterém jsme modul navrhovali. Následně jsme začali se samotným návrhem desky. Dále jsme přešli k samotné výrobě desky a osazení pinů. Před navržením jsme si vypočetli ochranný odpor, pro rezistory, které jsou připojeny ke každému vývodu k vývojového kitu. Použili jsme vzorec pro ohmův zákon, kde U = 5V, což je napájecí napětí vývojového kitu a I = 20mA je maximální odebíraný proud z I/O vývodu mikrokontroléru dle [1]. Rezistory s nejbližší vyšší hodnotou jsou s odporem R = 270Ω.

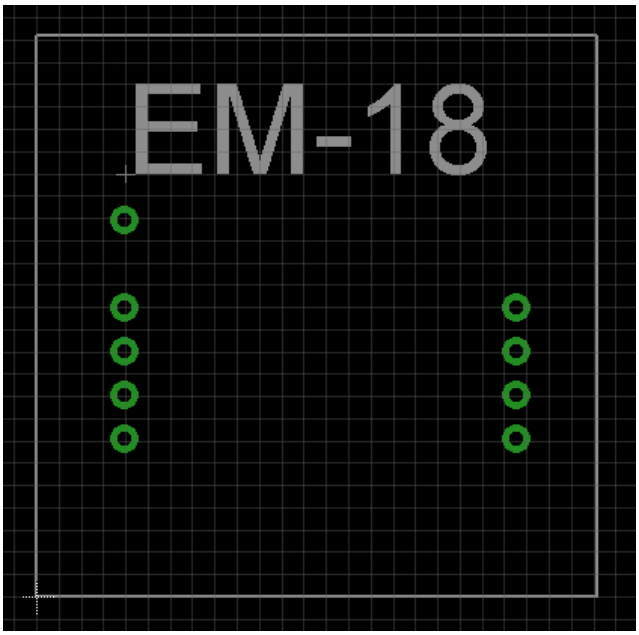
(1)

$$R = \frac{U}{I} = \frac{5}{0{,}02} = 250\Omega$$

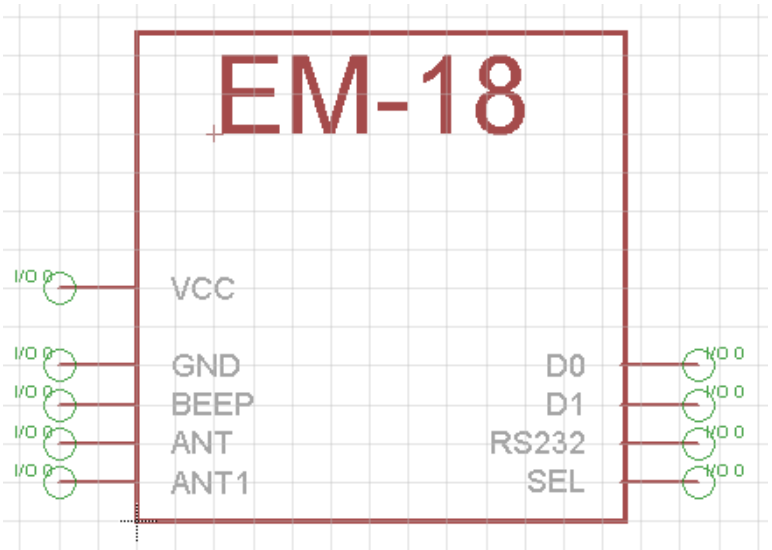
4.1. EAGLE 5.4

K navržení schématu a DPS jsme využili program Eagle.Postupovali jsme tak, že jsme si vyhledali datasheety k součástkám, které využíváme a podle nichž jsme navrhovali schématická zapojení v části „schematic“. Většina součástek se dala nalézt v interních knihovnách, nebo v knihovně, kterou jsme vyhledali [13] a některé jsme museli navrhnout a to RFID EM-18 a P-RE30S. Zde jsme také postupovali dle datasheetu, kde jsou zakótované rozměry a rozmístění pinů.

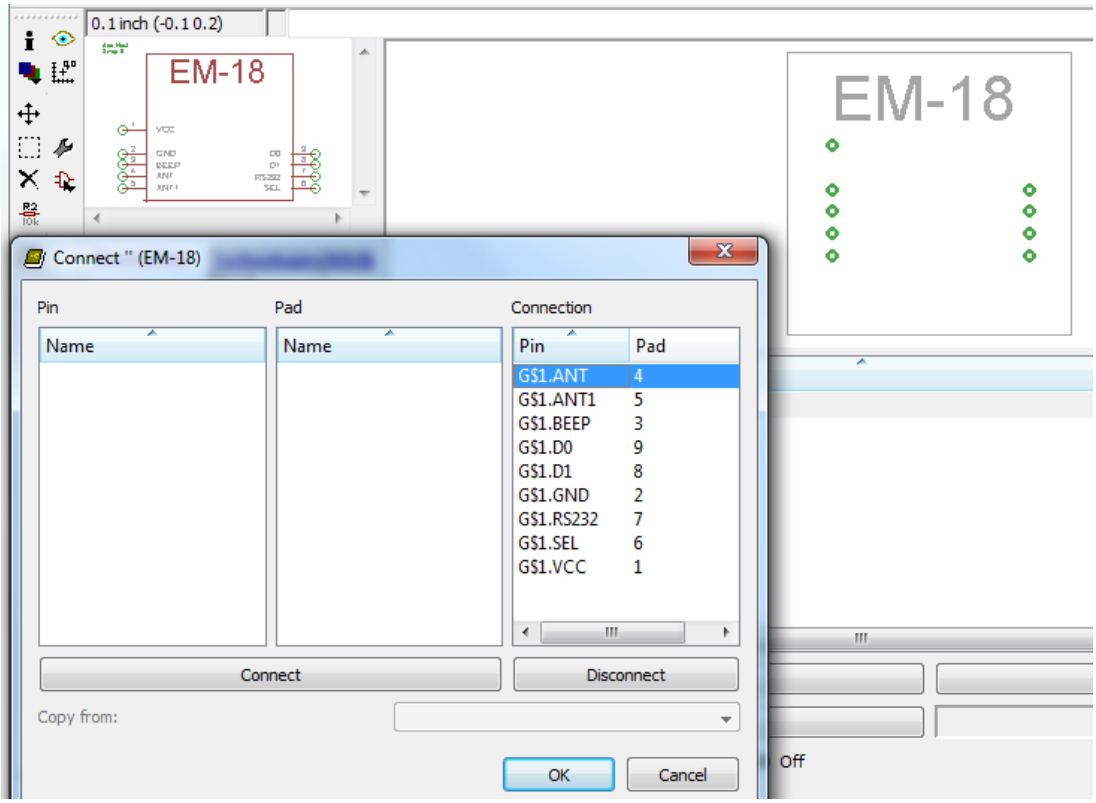
Tvorba těchto součástek se zkládala ze tří částí, první byla nakreslit součástku dle rozměrů a její rozmístění děr pro piny a také volba velikosti těchto děr, toto se dělalo v části „package“, následně se přešlo do části „symbol“, kde se nakreslila schématická značka dané součástky a přidal se tam potřebný počet pinů. Poslední částí je část „device“, kde propojíme nákres se schématem a připojíme zde piny na vytvořené díry. Jak vypadá tento proces je ukázáno na obr. č. 19-21.



Obr. č. 19 Návrh RFID v části „package“.



Obr. č. 20 Schematická značka a rozmístění pinů v části „symbol“.



Obr. č. 21 Připojení pinů na díry v části „device“.

Poté co jsme si vytvořili všechny součástky a navrhli schéma, které je v příloze na CD, tak jsme se přesunuli na návrh DPS, který se provádí v části „board“. Vzhledem k tomu, že program Eagle má tyto prostředí mezi sebou propojené, tak když jsme přidávali součástky v části „schematic“, tak se nám tyto součástky zároveň vkládaly i do části „board“. Zde jsme si součástky rozmístili a uspořádali, abychom ušetřili místo ale zároveň se neomezovali při tvoření cest. Navolili jsme si šířku cest 0,6mm z důvodu dobré výroby. Pouze v místech, kde by došlo ke zkratu jsme snížili šířku na 0,4mm.

Cesty jsme navrhovali ručně pomocí „Route“, je zde ještě možnost využít funkci „AutoRoute“, která nám tyto cesty sama propojí, ale tato funkce není vhodná pro jednostranné

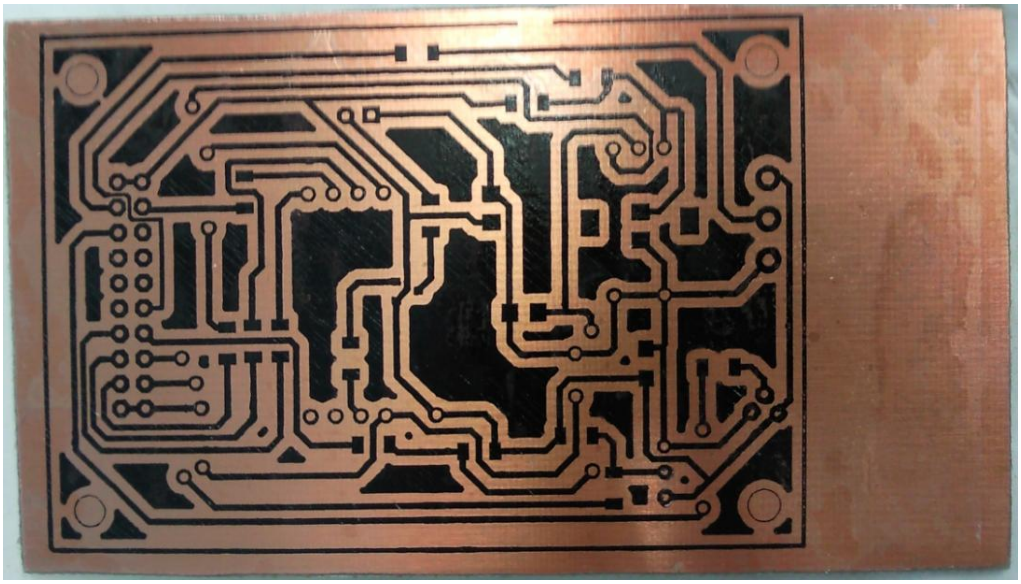
desky. Z tohoto důvodu jsme cesty navrhovali sami. Jak bylo zmíněno tak deska je jednostranná, a to z důvodu zvolené metody výroby. U návrhu jsme se nevyhnuli několika přemostění, která byly nezbytná. Navržené DPS je v příloze na CD.

4.2. Výroba DPS

Je zde několik způsobů jak vyrobit DPS, my jsme zvolili metodu kde se návrh vytiskne na samolepící papír. Tento papír je z jedné strany potáhnutý lepidlem. Na tuto stranu jsme návrh vytisknuli, aby se při nažehlení obtisknul na Cuprextitovou desku. Tutodesku jsme předtím museli nastříhat na rozměry vytištěné předlohy a následně ji důkladně očistit od nečistot pomocí tekutého písku a vody. Následně jsme lesku doleštili pomocí jemného smirkového papíru. Nakonec jsme ještě museli srazit hrany. Vzhled této desky je na obr. č. 22.

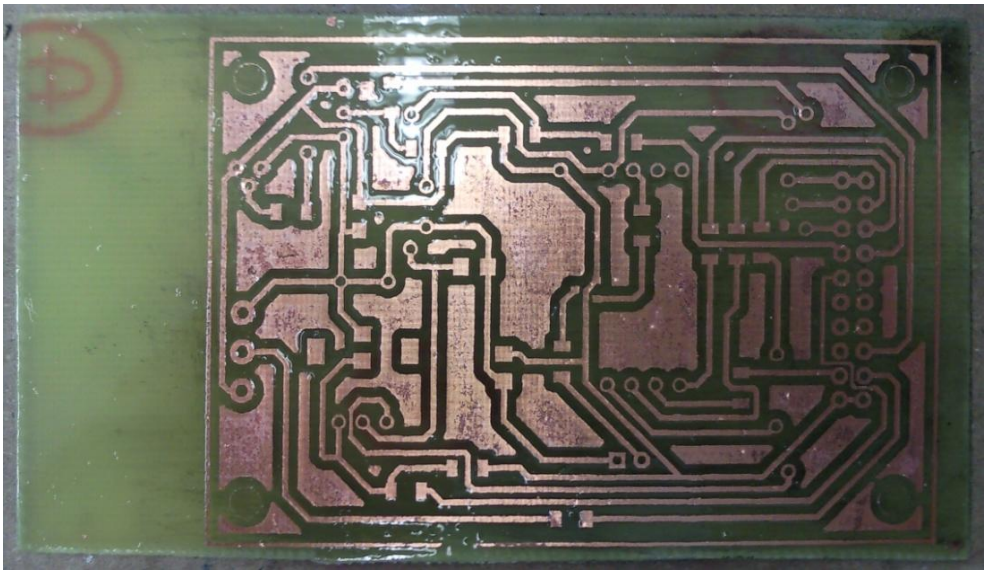
Obr. č. 22 Očištěná Cuprextitová deska.

Vytištěnou stranu papíru jsme přiložili na očištěnou část desky. Následně jsme to překryli papírem, aby žehlička nebylo přímo v kontaktu s papírem, na kterém je vytištěn návrh, protože by se rychle zahřál a mohlo by dojít k rozpití. Zažehlování trvalo cca 2-3 minuty. V tomto čase jsme střídavě vyvíjeli tlak na desku, aby se toner na ni přilepil. Po zažehlení se deska i s papírem položila do vodní lázně. Museli jsme počkat několik minut než se papír odlepil od desky. Tento proces se nám podařil až na třetí pokus. Hlavní výhodou této metody oproti metody fotocestou, která byla také jednou z možností, je to, že u fotocesty nevíme zda jsme někde udělali chybu až do odleptání. V našem případě jsme však vady mohli vidět už po odlepení papíru od desky, desku jsme mohli lehce očistit a postup podstoupit znovu. Na obr. č. 23 lze vidět jak se pomocí tepla toner přenesl na desku. Jak je z obrázku vidět tak jsme desku vystřihli o něco větší z důvodu lepší manipulace a přesnosti, aby se nám nestalo že se papír s návrhem posune mimo desku.



Obr. č. 23 Deska s nažehleným návrhem.

Následujícím krokem bylo leptání. To se provádělo tak, že se nahřál chlorid železitý, ten se nahříval v teplé vodní lázni. Předtím, než jsme započli s procesem leptání, jsme si druhou stranu desky udělali z lepící pásky úchyt, z důvodu lepší manipulace. Poté byla deska položena na hladinu do nádoby s choridem železitým. Deska se musela několikrát zvednout z hladiny a zkontrolovat zda se tam neutvořily vzduchové bubliny, které by zabránily leptání a znehodnotily desku. Když jsme se ujistili že zde nezůstaly žádné vzduchové bubliny, tak jsme desku nechali v klidu položenou na hladině. Leptání trvalo cca 20-30 minut. Po této době byla deska vyjmuta a zkontrolována, poté se osušila vzduchem a nastříkala proti korozi. Na obr. č. 24 je vidět jak deska vypadala po leptání.



Obr. č. 24 Vyleptaná a naimpregnovaná deska.

Nakonec se přistoupilo k osazení součástek a oživení desky. Při prvním osazování jsme si všimli že součástka RFID má piny zrcadlově obráceně. Tato chyba vznikla při návrhu této součástky. Z důvodu této chyby bylo nutné předělat jak tuto součástku, tak také návrh a následně podstoupit znova proces výroby. Deska má rozměry 7,8 x5,6 cm. Vzhled desky je vidět na obr. č. 25.

Obr. č. 25 Výsledný vzhled modulu.

5. SOFTWAROVÁ ČÁST

Programy pro jednotlivé součástky byl napsán v jazyce C ve vývojovém prostředí Atmel Studio 6.1. Napsané programy jsme následně kompilovali pomocí programu AVR ISP a programátoru UniProg-USB. Pro zobrazování na display jsme využili už hotovou knihovnu [13]. Při programování jsme se setkali s několika problémy, které zahrnovaly potíže s kompilátorem, který v novém Atmel Studiu neuměl zdvojovat registry. Daší problém se vyskytl když jsme chtěli použít funkci *sprintf*, když byl znak typu float, zde byl problém s Linkerem.

5.1. Potenciometr

Tento program byl využíván v předmětu Mikroprocesorová technika a s vedoucím jsme se domluvili, že použijeme stejný již ověřený program[14].Potenciometr je připojen k AD převodníku. AD převodník je zařízení, které nám převádí spojitý signál na digitální diskrétní hodnotu. Program funkce je, že měřená hodnota se porovnává s násobky referenčního napětí. Měření se provádí multiplexně, což má za následek prodlužování času potřebného k měření.

5.1.1. Vlastnosti AD převodníku

- doba převodu 13-260 µs
- rychlost až 15 kSPS
- vnitřní referenční napětí 2,56V
- maximální pracovní frekvence 200 kHz[14]

5.1.2. Použité registry

- ADMUX – slouží pro výběr kanálu a referenčního napětí
- ADCSRA – nastavení rychlosti AD převodu

- ADCW – obsahuje dva registry ADCH a ADCL[14]

Začali jsme funkcí *read_adc*, která nám precte hodnotu na vstupu AD převodníku, tento převodník má jeden vstupní parametr cislo *channel* a jeden výstupní a to hodnotu AD převodu.

```
unsigned int read_adc(unsigned char channel)
{
	ADMUX = channel;	// determination of converted channel
	ADCSRA |= 0x40;		/* Bit 6 - launching conversion. After finished
transmission this bit is cleared */

	while ((ADCSRA & 0x10)==0); /* loop this until 4. bit is 0 and after finished
                                                        conversion 4. bit will be
set to 1 */

	ADCSRA |= 0x10;		// reset of 4. bit
	return ADCW;  /* ADCW is fused registers ADCL and ADCH where is result of
transmission */
}
```

Dále jsme si načetli AD převodník pomocí několika registrů a také jsme si napsali několik funkcí pro náš LCD displej a to funkci *LCD_Init* kterou načítáme LCD, *LCD_Clear* pro vymazání LCD, také jsme za tyto funkce použili funkci *_delay_ms*, která nám slouží jako funkce zpoždění :

- ACSR – Bit 7 pokud je do tohoto bitu zapsaná log. 1, tak se odpojí napájení komparátoru, což nám poslouží k minimalizaci spotřeby
- SFIOR – tím jak jsme nastavili tento registr, tak nám slouží jako nulování
- ADMUX – Bit 7:6 = 1:1 tyto bity slouží pro výběr referenčního napětí dle kombinací bitů, jak je v tabulce č. 3

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Tabulka č. 3 Výběr zdroj referenčního napětí podle kombinací bitů 7:6.[2]

- ADCSRA – v tomto registru nám první tři bity nastavují dělící poměr pro odvození hodinového signálu ADC převodníku z hodinového kmitočtu CPU a 7 bit nám povoluje funkci ADC převodníku[14]

```
int main (void)
{
	/* Setting registers for AD convertor*/
	ACSR = 0x80;
	SFIOR = 0x00;
	ADMUX = 0x0;
	ADCSRA = 0x87;
```

```
	LCD_Init();
	LCD_Clear();
	_delay_ms(2);
```

Následně jsme si napsali funkci pro přepočet AD převodu na napětí. Zde je vstupním parametrem *read_adc*, který přiřazujeme naší proměnné *ADtransfer*, která je typu int a kterou budeme následně vypisovat na displej. Výstupem této funkce je proměnná *voltage*, kterou budeme vypisovat na spodní řádek.

Funkce *LCD_Position* nám slouží pro určení na jakém řádku se bude text který budeme vypisovat nacházet. Dále jsme použili funkci *sprintf*, která nám zapíše formátovaný výstup do pole *tmp* v paměti, následně toto pole vypíšeme na brazovku pomocí funkce *LCD_WriteString*. Tento postup provedeme stejně i pro výpis napětí. Na konec jsme si zde dali funkci *_delay_ms*, která nám zde slouží k ustálení hodnot na displeji.

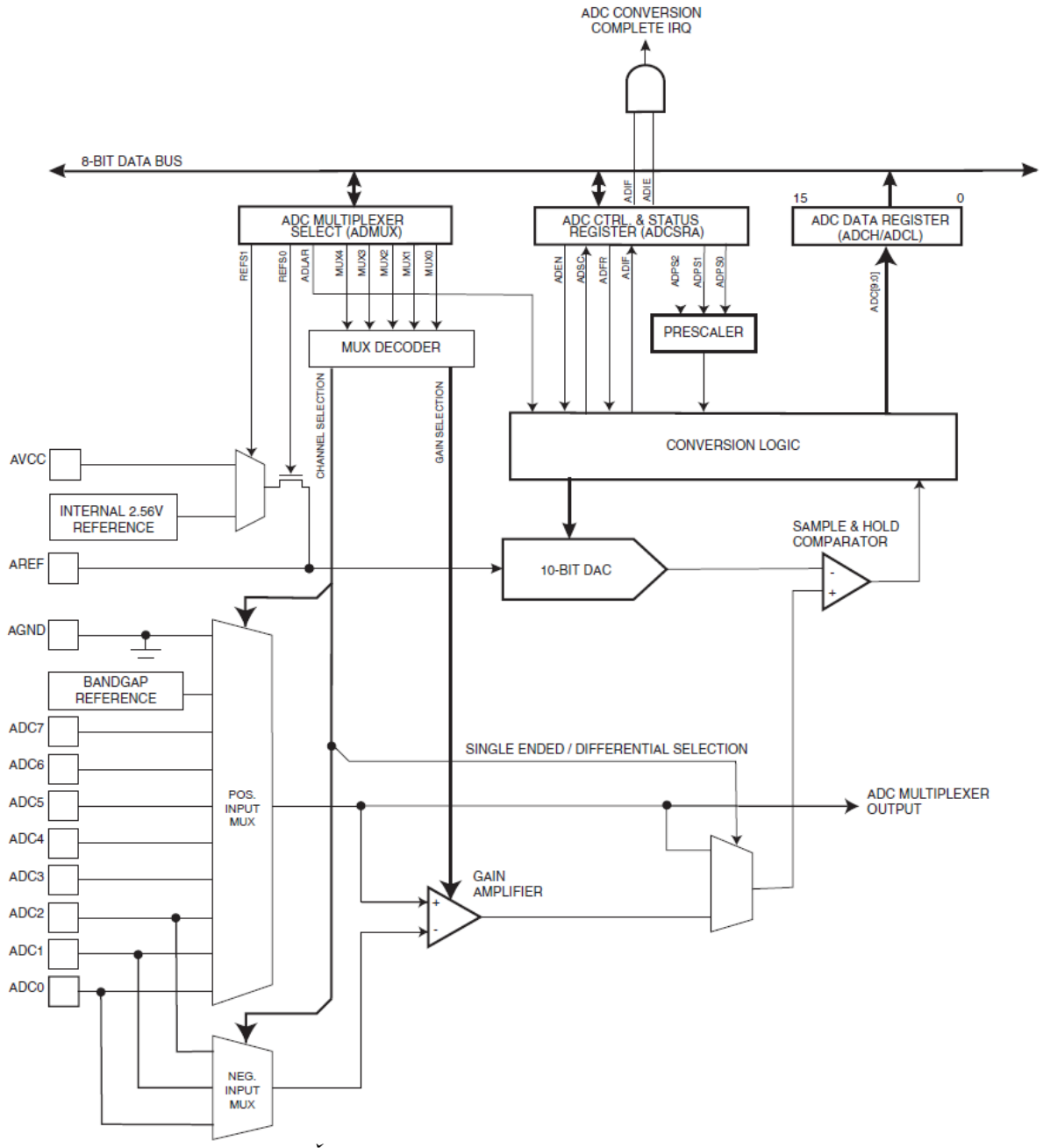
```
while (1)
	{
		ADtransfer=read_adc(0);
		voltage= ADtransfer*5;	// restatement ADtransfer to voltage
		voltage= voltage/1024;

		LCD_Position(0 , 0);
		sprintf (tmp, "ADprevod= %4d", ADtransfer);
		LCD_WriteString(tmp);
		LCD_Position(1 , 0);
		sprintf (tmp, "Napeti= %4.2f V", (double)voltage);
		LCD_WriteString(tmp);

		_delay_ms(500);

	};
}
```

Na obr. č. 26 můžeme vidět blokové schéma AD převodníku, znázorňující jeho princip.



Obr. Č. 26 Blokové schéma AD převodníku.[14]

5.2. Teploměry

Vzhledem k tomu, že používáme dva teploměry, tak jsme se rozhodli, že je dáme do jednoho programu a jejich teploty budeme sledovat ve dvou řádcích nad sebou. Pro analogový teploměr jsme využili stejnou funkci jako pro potenciometr viz kapitola **5.1**. Jedinou změnou byl převod AD převodu na teplotu, ten můžeme vidět v této části kódu:

```
float temperature;
unsigned int ADtransfer=read_adc(0);
temperature = (ADtransfer*0.5);

LCD_Position(0 , 0);
sprintf (tmp, "T1 = %4.2f", (double)temperature);
LCD_WriteString(tmp);
```

Pro náš digitální teploměr jsme využili externí knihovny a využili jsme z funkce, které jsme potřebovali. Tyto knihovny můžeme najít v příloze na CD. Jsou to knihovny *ds18x20.c, onewire.c,* a *crc8.c.*[15]

Prvně jsme si definovali adresu portu na kterém je náš teploměr, toto jsme provedli pomocí funkce *ow_set_bus.* Tuto funkci jsme použili z knihovny *onewire.c.*

```
/* Setting pin */
ow_set_bus(&PINE,&PORTE,&DDRE,PE6);
```

Dále jsme si deklarovali parametry *id* a *decelsius*, kde *id* je výrobní kód našeho teploměru, který se uložil do paměti ROM a *decelsius* jsou převedená data ze *scratchpad* paměti na náš parametr *decelsius*. Funkce pro přepočet těchto dat na parametr *decelsius* je v knihovně *ds18x20.* Tento teploměr je od výroby nastaven na 12-bitové rozlišení, které potřebujeme znát pro přepočet naší teploty. Vstup je parametr *measure* a výstupem je náš parametr *decelsius*, který budeme zobrazovat na displeji.

```
int16_t decicelsius;
```

Následně jsme využili funkci *DS18X20_start_meas.* Zde jsme měli na výběr mezi dvěma způsoby napájeními a to buď externí a nebo parasite. Vzhledem k našemu návrhu jsme zvolili externí napájení. Tato funkce si prvně ověří, zda je teploměr pod napětím, protože tento teploměr má klidový stav když je napájen. Pokud je teploměr napájen tak se spustí měření. Vstupem je zde *DS18X20_start_meas* a výstupem je *DS18X20_POWER_EXTERN*.

```
/* Setting on measuring */
DS18X20_start_meas( DS18X20_POWER_EXTERN, NULL );
```

Dále jsme využili funkci *_delay_ms*, která už byla předvolena v knihovně *ds18x20.h* a je zde nastavena hodnota 750ms pro zobrazení 12-bitů.

```
/* Waiting for output */
_delay_ms( DS18B20_TCONV_12BIT );
```

Další funkcí je funkce *DS18X20_read_decelsius_single* tuto funkci můžeme využít z toho důvodu že využíváme pouze jeden teploměr. Pomocí této funkce čteme převedenou teplotu ze *scratchpadu*. Vstupem této funkce je funkce *DS18X20_raw_to_decelsius,* která obsahuje převedá data na parametr *decelsius*, který je náš výstup z této funkce.

```
/* Loading output to decelsius */
DS18X20_read_decicelsius_single( id, &decicelsius );
```

Poslední funkcí je *DS18X20_format_from_decelsius*, tato funkce nám převádí parametr *decelsius* na řetězec, dále se zde nastavuje rozsah měření, v našem případě jsme nastavili parametr *n = 7*, který nám stanovuje rozsah od -55°C do +125°C, dále je zde zahrnuta podmínka pro, že když je *decelsius* menší než 0 tak se bude vypisovat záporná hodnota teploty. Také je zde nastaveno pole. Vstupem do této funkce je teplota a výstupem je tato teplota zapsaná v řetězci a tento řetězec následně vypíšeme na LCD displej pomocí funkce *LCD_WriteString* a vypíšeme ji na druhý řádek pomocí funkce *LCD_position.* Nakonec jsme nastavili funkci *_delay_ms* aby se nám data obnovovala každých 300ms.

```
/* Writing out temperature */
DS18X20_format_from_decicelsius(decicelsius, tmp, 7);

LCD_Position(1 , 0);
```

```
        LCD_WriteString(tmp);

        _delay_ms(300);

        };
}
```

5.3. Encodér

Enkodér pracuje na principu že při otočení se vyvolá přerušení na jednom pinu a my kontrolujeme stav toho druhého, dále náš enkodér disponuje axiálním tlačítkem. Při začátku programování jsme si napsali jednoduchou funkci pro přerušení a výpisem na obrazovku, tato funkce nám bohužel nefungovala. Z toho důvodu jsme se rozhodli že přeměříme piny jestli jsou správně zapojeny. Když jsme tyto piny proměřili, tak jsme zjistili že enkodér je špatně zapojen, že má některé piny přehozeny. Tuto závadu jsme opravili přerušením špatných cest a jejich přepájením. Následně jsme tyto úpravy aplikovali i do našeho návrhu DPS.

Při programování jsme začli tím že jsme si nastavili vecto přerušení pro pin PE5, takže jsme sestupno, nebo vzestupnou hranu na pinu PE4. V Této funkci jsme si nastavili, že při přerušení na pinu PE5 se bude do proměnné *encoder* zapisovat ±1 podle směru otáčení a nebo pokud zmářkneme axiální tlačítko, tak se zde bude zapisovat ±10.

```
ISR(INT5_vect){

        if ((PINE & _BV(PE4)))
        if(tens)
        encoder += 10;
        else
        encoder += 1;
        else
        if(tens)
        encoder -= 10;
        else
        encoder -= 1;
        }
```

Pak jsme opět využili funkce pro načtení LCD dispeleje jako v kapitole č. **5.1.**

Poté jsme si nastavili registry a přerušení dle datasheetu [2].

- EIMSK – je registr masek pro přerušení
- EICRB - registr přerušení pro piny 4 – 7, v našem případě se jedná o pin 5, a kdy bude přerušení vyvoláno, nám určí bity 3 a 4, jak je zobrazen v následující tabulce č. 4

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Any logical change on INTn generates an interrupt request
1	0	The falling edge between two samples of INTn generates an interrupt request.
1	1	The rising edge between two samples of INTn generates an interrupt request.

Tabulka č. 4 Tabulka zobrazující kdy se vyvolá přerušení.[2]

```
/* Setting pins and setting up interrupt for pin B */
	DDRE &= ~(_BV(PE4)|_BV(PE5) |_BV(PE7));
	EIMSK |= _BV(INT5);
	EICRB |= _BV(ISC51) | _BV(ISC50);
```

Zápis _BV() má stejnou funkci jako zápis << a to bitový posuv o 1 bit.Po nastavení přerušení jsme museli nastavit PORTE pro pin PE7, z toho důvodu že tento port máme napojen přímo na vývojový kit a není zde rezistor, tímto zápisem jsme nastavili pull-up pro tento port.

```
PORTE |= _BV(PE7);
```

Poté jsme napsali funkci, aby se na displeji přepisovaly hodnoty pouze při změně hodnoty. Vstup je zde parametr *last* a výstup je *encoder*.

```
	if(last != encoder){ // This is function to overwrite display
	   last = encoder;         // only after changing a value
```

Dalším krokem bylo opětovné využití již použité funkce v kapitole č. **5.1.** pro výpis na LCD displej. Jediný rozdíl byl ve výstupu z této funkce a to je v tomto případě parametr *encoder*.

```
	LCD_Clear();
	_delay_ms(10);
	LCD_Position(0 , 0);
	sprintf (tmp, "otoceni = %d", encoder);
	LCD_WriteString(tmp);
```

Posledním krokem bylo napsání funkce pro naše axiální tlačítko. V této funkci *button_state* máme ošetřeno axiální tlačítko, a to tak že při zmáčknutí se při jednom kroku na displeji nebude přičítat nebo odečítat 1 ale 10, také zde máme nastavený časovač *button_timer*, kvůli toho že se jedná o mechanické kontakty a proto zde dochází k zákmitům. Vstupem této funkce je *button_state* a výstupem je hodnota proměnné *tens*.

```
switch(button_state){
		case 0:
		if((PINE & _BV(PE7)) == 0){
			tens = !tens;
			button_state = 1;
			button_timer = 100;

		}
		break;
		case 1:
		if(button_timer){
			--button_timer;
			} else {
			button_state = 2;
		}
		break;
		case 2:
		if(PINE & _BV(PE7)){
			button_state = 0;
		}
		break;
	}
```

5.4. RFID čtečka

Čtečka má dva způsoby komunikace, nicméně my jsme se zabývali pouze komunikací přes protokol RS 232. Tato komunikace probíhá přes UART rozhraní. Začali jsme tím, že jsme si načetli UART rozhranní. To jsme provedli funkcí *init_uart*, zde jsme si prvně našli v tabulce hodnotu pro UBRR register, což je registr pro nastavení rychlosti přenosu. Zde jsme si našli že rychlost přenosu pro frekvenci našeho mikroprocesoru odpovídá 95 kbps. Také jsme si zde nastavili registr UCSR0B, ve kterém je na 7 bit RXCIE0 nastavení přerušení RX komunikace, což je komunikace čtečky s mikrokontrolérem. Na 4 bitu RXEN0 je aktivace USART0 přijímače. Dále jsme si zde nastavili registr UCSR0C ve kterém jsme si pomocí prvních dvou bitů UCSZ01 a UCSZ00 nastavili velikost znaků na 8-bitové, jak je vidět v tabulce č. 5.[2]

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Tabulka č. 5 Nastavení velikosti znaků.[2]

Celá funkce následně vypadá takto.

```
void init_uart(void) {

    UBRR0H = 0;
    UBRR0L = 95;

    UCSR0B = _BV(RXCIE) | _BV(RXEN0);
    UCSR0C = _BV(UCSZ01) | _BV(UCSZ00);

    rx_buffer_pos = 0;
    rx_timer = 0;
}
```

Dále jsme si nastavili vektor přerušení pro funkci *USART_RX_vect*, kde jsme přiřadili hodnotu proměné *c* registru UDR0, což je registr pro I/O data. Vstupem této funkce tedy je registr UDR0 a výstupem je funkce *rx_add_char*.

```
ISR(USART0_RX_vect) {

    char c = UDR0;
    rx_add_char(c);

}
```

Když jsme mněli tyto věci nastavené, tak jsme si definovali velikost paměti a po jaké době se má komunikace přerušit.

```
#define RX_BUFFER_SIZE 16
```

```
#define RX_TIMEOUT 5

char rx_buffer[RX_BUFFER_SIZE];       /* Buffer for RFID ID */
volatile uint8_t rx_buffer_pos = 0;   /* Count of bytes in rx_buffer */
volatile uint8_t rx_timer = 0;        /* Timeout variable. Should be decreased every
10 ms. */
```

Poté jsme si zavolali funkci *rx_add_char* z našeho přerušení, v této funkci jsme si jako první krok provedli kontrolu, jestli už není nějaká zpráva uložena v paměti, následně jsme zkontrolovali, jestli je v paměti dost místa pro celou zprávu a pokud ano, tak jsme do funkce *rx_buffer* uložili nové znaky. Vstupem této funkce je *rx_add_char* a výstupem je *rx_buffer*.

```
/* Call from ISR */
void rx_add_char(char c) {

    /* Check if there is an unread message in buffer */
    if(rx_buffer_pos && rx_timer == 0) {
        return;
    }

    /* Save new character if there is free space in buffer. */
    if(rx_buffer_pos < RX_BUFFER_SIZE) {
        rx_buffer[rx_buffer_pos++] = c;
    }

    /* Set message-end timeout */
    rx_timer = RX_TIMEOUT;
}
```

V další funkci kontrolujeme zda nám přišla celá zpráva a pokud ano, tak uzamkneme přerušení, aby se náhodou tato zpráva neznehodnotila v případě přiblížení tagu, když se znaky z prvního čtení ukládají. Poté co se znaky uloží do bufferu se přerušení opět zapne. Vstupem je funkce *rx_check* a výstupem je proměnná *size*.

```
/* Call from normal code */
/* buffer should be at least RX_BUFFER_SIZE bytes wide.*/
uint8_t rx_check(char *buffer) {
    isr_lock();

    /* Check if there is an unread message. */
    if(rx_buffer_pos==0 || rx_timer) {
        isr_unlock();
        return 0;
    }

    /* Copy message to user buffer. */
    uint8_t size = rx_buffer_pos;
    memcpy(buffer, rx_buffer, size);
    rx_buffer_pos = 0;

    isr_unlock();
    return size;
}
```

Následně jsme si navolili funkce na zobrazování zpráv na LCD displeji. První funkce *lcd_show* nám vypíše na první řádek naše ID tagu i s jeho CRC částí. Pomocí další funkce *lcd_second_line* budeme vypisovat na další řádek informační zprávu. Třetí funkce *lcd_clear*

nám vypisuje na první řádek zprávu, když je čtečka v klidovém stavu. Posldní funkcí pro LCD displej je funkce *init_lcd*, která slouží k načtení displeje.

```
/*                                                                      Display     */
/* ------------------------------------------------------------------------------- */


void lcd_show(char * bytes, uint8_t size) {
    LCD_Clear();
    LCD_Position(0,0);
    bytes[size] = 0;
    _delay_ms(30);
    LCD_WriteString(bytes);

}

void lcd_second_line(char * str){
       LCD_Position(1,0);
       LCD_WriteString(str);
}

void lcd_clear(void) {
    LCD_Clear();
    LCD_Position(0,0);
    _delay_ms(5);
    LCD_WriteCString(" Cekam ...");
}

void init_lcd(void) {
    LCD_Init();
    _delay_ms(5);
    lcd_clear();
}
```

V další části jsme nastavovali časovač pomocí funkce *init_timer*, zde máme definováno několik registrů. Prvním z nich je registr OCR0 výstupní porovnávací registr a jeho frekvence by mněla být cca 100Hz. Dalším registrem je registr masek TIMSK, kde na druhém bitu je OCIE0, který pokud je do něho zapsána log. 1, tak je povoleno porovnání přerušení. Posledním registrem je registr TCCR0, který řídí funkce časovače. Zde nás zajímají bity 0-3. První tři bity nastavují zdroj hodinového signálu jak můžeme vidět v tabulce č. 6.[2]

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk_{T0S}/(No prescaling)
0	1	0	clk_{T0S}/8 (From prescaler)
0	1	1	clk_{T0S}/32 (From prescaler)
1	0	0	clk_{T0S}/64 (From prescaler)
1	0	1	clk_{T0S}/128 (From prescaler)
1	1	0	clk_{T0S}/256 (From prescaler)
1	1	1	clk_{T0S}/1024 (From prescaler)

Tabulka č. 6 Nastavení zdroje hodinového signálu.[2]

Čtvrtý bit kontroluje frekvenci OCR0 a pomocí jakého režimu CTC (Clear timer on Compare). Toto nastavení tohoto bitu lze vidět v tabulce č. 7.[2]

Mode	WGM01[1] (CTC0)	WGM00[1] (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0 at	TOV0 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

Tabulka č. 7 Nastavení WGM01 pro režim CTC.[2]

Posledním nastavením v této funkci bylo nastavení vektoru přerušení časovače při shodě.

```
/*                                                                      Timer       */
/* ---------------------------------------------------------------------------- */

volatile uint16_t lcd_timer = 0;

void init_timer(void) {


    OCR0 = 143; // cca 100 Hz
    TIMSK = _BV(OCIE0);
    TCCR0 = _BV(WGM01) | _BV(CS00)| _BV(CS01)| _BV(CS02);

}

ISR(TIMER0_COMP_vect)
{
    if(lcd_timer) --lcd_timer;
    if(rx_timer) --rx_timer;
}
```

Dalším krokem bylo provedení funkce *checksum_hex_to_byte*. Pomocí této funkce převádíme hexové znaky na bajty. A aby se nám nevypisovaly různé znaky, ale jen čísla, tuto funkci jsme ošetřili tak, že pokud je to číslo větší než 9 v ASCII tabulce, odečteme od něho desítkovou hodnotu písmene „A“ a pokud bude číslo větší 5, odečteme od něho hodnotu malého písmene „a“ a přičteme k tomu číslu hodnotu 10. Jejím vstupem je proměná *byte* a výstupem proměná *ret*.

```
/* Converts char in ASCII to integer.
  checksum_hex_to_byte('A') == 10
  checksum_hex_to_byte('5') == 5
  checksum_hex_to_byte('f') == 15
  etc.
*/
uint8_t checksum_hex_to_byte(char byte) {
    uint8_t ret = byte - '0';

    if(ret > 9) {
        ret = byte - 'A';
        if(ret > 5)
            ret = byte -'a';
        ret += 10;
    }
```

```
    return ret;
}
```

V dalším kroku převádíme pomocí funkce *checksum_ascii_to_bytes*. Zde převádíme ASCII znaky na bajty. Vstupem funkce je pole předešlé funkce *checksum_hex_to_byte* a výstupem je pole vyděleno modulo 2.

```
/* Converts HEX ASCII to bytes ...
  s           input array of length size
  out         output array of length size/2

  for example: s = "A0B5"    ->    out = {160, 181}
*/
void checksum_ascii_to_bytes(char* s, uint8_t* out, uint8_t size) {
    uint8_t i;

    for(i=0; i < size; i++) {
        uint8_t out_pos = i/2;
        uint8_t byte = checksum_hex_to_byte(s[i]);

        if(i % 2 == 0) {
            out[out_pos] = 0;
            byte <<= 4;
        }

        out[out_pos] |= byte;
    }
}
```

Následovala funkce XOR pro ověření správnosti dat. Kontrolovali jsme bajt po bajtu.

```
/* Compute : bytes[0] XOR bytes[1] XOR ... XOR bytes[size-1] */
uint8_t checksum_compute_xor(uint8_t * bytes, uint8_t size) {
    uint8_t i, ret;

    ret = 0;
    for(i=0; i < size; i++) {
        ret ^= bytes[i];
    }

    return ret;
}
```

V hlavním programu jsme si prvně zkontrolovali, jestli jsou dvě pole délky pěti bajtů, což je délka ID tagu, stejné a pokud ano, vrátila se nám 1.

```
/* Compares two arrays of length 5. If there are the same values in both 1 is
returned. */
uint8_t check_card(uint8_t *c1, uint8_t *c2){
       uint8_t i;
       for(i=0; i < 5; i++){
              if(c1[i] != c2[i])
                     return 0;
       }

       return 1;
}
```

ID jednoho tagu jsme si navolili jako výchozí a porovnávali jsme k němu ostatní karty.

```
uint8_t special_card[] = {0x14,0x00,0xF3,0xC9,0xBE};
```

Dále jsme v hlavním programu využili funkce, které jsme si předtím nadefinovali, provedli jsme zde kontrolu CRC pro 5 bajtové ID s CRC. Následně jsme porovnávali vypočítáné CRC a získané CRC a podle toho jestli se nám shodovaly nebo ne, se nám zobrazují zprávy na spodním řádku displeje.

```
/* Init device. */
    init_uart();
    init_lcd();
    init_timer();

    /* Enable interrupts. */
    isr_unlock();

    state_t state = IDLE;

    while(1) {
        switch(state) {

            /* ------------------*/
        case IDLE: {

            char buffer[RX_BUFFER_SIZE+1];
            uint8_t bytes;

            /* Check if there is new message and copy it to buffer. */
            bytes = rx_check(buffer);

            /* If there is one then display it on LCD and change state. */
            if(bytes) {

                        /* We check crc for 5 byte long ID and 1 byte CRC. */
                if(bytes == 12) {
                    uint8_t ch1, ch2;
                    uint8_t tmp[5];

                                /* Convert ASCII HEX to byte array and compute crc
                                */
                    checksum_ascii_to_bytes(buffer, tmp, 10);
                    ch1 = checksum_compute_xor(tmp, 5);

                                /* Convert crc received from RFID module to byte */
                    checksum_ascii_to_bytes(buffer+10, &ch2, 2);

                                lcd_show(buffer, bytes);

                                /* Compare computed crc and received crc */
                    if(ch1 == ch2) {

                                        if (check_card(tmp,special_card)){
                                               char ok[] = "Pristup povolen";
                                               lcd_second_line(ok);
                                        } else {
                                               char ok[] = "Pristup odepren";
                                               lcd_second_line(ok);
                                        }
                    } else {
                        char error[] = "wrong crc";
                        lcd_second_line(error);
```

```
                    }
                } else {
                                char tmp[] = "no crc check";
                    lcd_show(buffer, bytes);
                                lcd_second_line(tmp);
                }


                lcd_timer = 250;
                state = DISPLAY;
            }
            break;
        }

        /* ------------------*/
        case DISPLAY:

            /* Just wait some time, then clear display and be ready for next message.
*/
            if(lcd_timer==0) {
                lcd_clear();
                init_uart();
                state = IDLE;
            }
            break;
        }

        _delay_ms(10);
    }

    /* Never get here ... */
    return 0;
}
```

6. ZÁVĚR

Pro naplnění cíle bakalářské práce bylo nutné se nejdříve zaměřit na problematiky všech komponentů a také příslušných periférií včetně využívaného mikrokontroléru.

Druhá část je zaměřena na popsání výukového kitu a mikroprocesoru, který využívá. Byly zde popsány jejich vlastnosti, periférie na které jsou připojené zařízení a také nutné nastavení pro potřebné účely.

Třetí část se zabývá důkladným popsáním jednotlivých komponentů, které byly využity pro náš modul. Především zde byly popsány jak jejich důležité vlatnosti, tak i jejich způsob komunikace.

Ve čtvrté části je zahrnut proces výroby DPS, od samotného návrhu schématu, kde jsme se setkali s problémem absence některých součástí. Přes návrh DPS a jeho výroby, až po objevenou chybu v návrhu, která vedla k předělání desky.

Poslední kapitola zahrnuje navržené programy pro výše zmíněné komponenty, abychom dokázali jejich funkčnost. Zde jsme se setkali s další chybou a tou byl špatně zapojený encoder. Tato chyba byla opravena přímo na desce, kde byly přerušeny špatné spoje a napájely se poté tak, jak měly. Tato úprava byla zpětně provedena i v návrhu, aby se tento návrh dal v budoucnu použít pro výrobu dalších modulů.

Cíle práce se podařilo splnit a vyrobený modul bude schopen plnit požadované funkce v předmětu mikroprocesorová technika.

7. SEZNAM POUŽITÉ LITERATURY

[1] PK-Design – Atmel AVR, Xilinx CPLD, FPGA Development (EVM) Boards. *Základová deska MB-ATmega128 v4.0* [on-line]. 2008 [cit. 2014-05-10]. Dostupné z: http://www.pk-design.net/HtmlCz/MB_ATmega128v4.html

[2] Atmel Corporation – Microcontrollers, 32-bit, and touch solution. Datasheet *ATmega 128* [on-line]. 2011 [cit. 2014-05-11]. Dostupné z: http://www.atmel.com/devices/atmega128.aspx

[3] RFID VŠB-Technical University of Ostrava. *RFID pro logistickou akademii* [on-line]. 2014 [cit. 2014-05-12]. Dostupné z: http://rfid.vsb.cz/cs/informace/

[4] EM-18 Rfid Reader | Rfid Card – Jaycon Systems Jaycon Systems LLC. *EM18 RFID Datasheet* [on-line]. 2014 [cit. 2014-05-12]. Dostupné z: http://www.jayconsystems.com/em-18-rfid-reader.html

[5] Popis protokolu Wiegand a řešení jeho čtení procesorem. *Protokol Wiegand* [on-line]. 2012 [cit. 2014-05-13]. Dostupné z: http://www.dhservis.cz/dalsi_1/wiegand.htm

[6] HW server představuje – Sériová linka RS-232 | HW.cz. *Obsah* [on-line]. 2005 [cit. 2014-05-14]. Dostupné z: http://www.hw.cz/rozhrani/hw-server-predstavuje-seriova-linka-rs-232.html#datovy_prenos

[7] Datasheet search site. ALLDATASHEET.COM – *Datasheet search site for Electronic Components and Semiconductors and other semiconductors* [on-line]. 1999 [cit. 2014-05-14]. Dostupné z: http://www.alldatasheet.com/datasheet-pdf/pdf/8875/NSC/LM35DZ.html

[8] Datasheet search site. ALLDATASHEET.COM – *Datasheet search site for Electronic Components and Semiconductors and other semiconductors* [on-line]. 2003 [cit. 2014-05-14]. Dostupné z: http://www.alldatasheet.com/datasheet-pdf/pdf/58557/DALLAS/DS18B20.html

[9] Ing.Luděk Kohout. *Snímače polohy* [on-line]. 2008 [cit. 2014-05-15]. Dostupné z: www.edumat.cz/texty/poloha.pdf

[10] PC1921BK500D | GM electronic. *PC1621BK500D* [online]. 2014 [cit. 2014-05-15]. Dostupné z: http://www.gme.cz/pc1621bk500d-p113-082

[11] Elektronika kvalitně. *Použití rotačních enkodér* [on-line]. 2007 [cit. 2014-05-16]. Dostupné z: http://elektronika.kvalitne.cz/ATMEL/necoteorie/tutorial/RotaryEncoder/RotaryEncoder.html

[12] GM electronics. *Datasheet 532-087.1* [on-line]. 2000 [cit. 2014-05-17]. Dostupné z:http://www.vo.gme.sk/_dokumentace/dokumenty/532/532-087/dsh.532-087.1.pdf

[13] PaJa – Eagle. *EAGLE* [on-line]. 2014 [cit. 2014-05-18]. Dostupné z: http://www.paja-trb.cz/eagle/index.html

[14] Zuth, Daniel. *Osnova VHT* [on-line]. 2010 [cit. 2014-05-18]. Dostupné z: http://hw.zuth.cz/08hodina.html

[15] Thomas, Martin. *DS18X20-Functions via one-wire-Bus* [on-line]. 2004 [cit. 2014-05-17]. Dostupné na: http://www.siwawi.arubi.uni-kl.de/avr-projects