ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

Katedra výkonové elektroniky a strojů

Bakalářská práce
Přepínací regulátor

Ivan Sadílek

2021

Zadání BP

1. Uvažujte možnost SW konfigurace a komunikace s nadřízeným systémem
2. Uvažujte možnost kombinování regulovaných a zpětnovazebních veličin
3. Navrhněte vhodnou optickou indikaci stavů regulátoru

Abstrakt

Následující práce pojednává o programování, funkčnosti a částečně hardwarové realizaci zařízení nazvaného jako přepínací regulátor (anglicky, Switching regulator). Jak již naznačuje název přepínací regulátor, finální produkt má za úkol, udržet veličinu ovládaného zařízení shodnou s veličinou řídící. K tomu využívá řízení uzavřené smyčky s veličinou zpětnovazební. V práci je popsána programová abstrakce vstupů, výstupů, hlavní algoritmus a příkazová komunikace regulátoru s počítačem.

Klíčová slova

řízení relé, vestavěné programování, ATmega328P, PlatformIO, Arduino

Abstract

The following work describes programming and hardware design of the Switching regulator device additionaly with its various features and functionalities. As the name hints, switching regulator, in its final form, has a task of controlling its output to match its input. To acomplish that switching regulator has a closed loop control system with a feedback input. This work deals with program abstraction of inputs, outputs, main algorithm and command based communication with a computer.

Key words

relay control, embedded programming, ATmega328P, PlatformIO, Arduino

Prohlášení

Tímto prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této bakalářské práce.

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

.....

V Plzni dne 22.05.2021 Ivan Sadílek

Poděkování

Tímto bych rád poděkoval firmě TCELE za nabídku spolupráce na produktu přepínacího regulátoru, a také panu Ing. Tomáši Chvátalovi za velice pomocné mentorské rady, připomínky a spolupráci. Dále bych chtěl také poděkovat vedoucímu bakalářské práce Ing. Petru Weissarovi Ph.D. za přínosné kontroly obsahu této práce.

Obsah

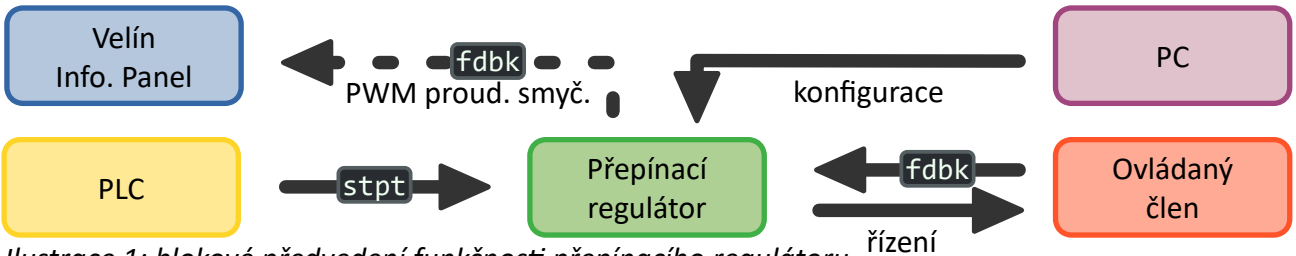
1 Úvod

Projekt byl založen na základech poptávky firmy ZPA-RP k firmě TCELE po zařízení na regulaci prvků palivového kombinátu Vřesová. Po dohodě a zvážení schopnosti splnění následujících nároků se firma rozhodla nabídku přijmout, a já mohl přistoupit k realizaci.

1. Realizace ovládání motoru ventilu spínáním fáze, síťového napětí.
2. Možnost zpracování vstupní řídící a zpětnovazební veličiny, proudovou smyčkou i napěťovou hladinou.
3. Výstup zpětnovazební hodnoty jako další proudová smyčka pro signalizaci na velín.
4. Adekvátní možnost jednoduchého nastavení funkčnosti zařízení a komunikace s počítačem.
5. Možnost manuální operace bez vstupní hodnoty PLC (využito k nastavení limit řídící a zpětnovazební veličiny, jak proudových, tak napěťových vstupů).

Postupným vývojem a plněním výše zmíněných bodů vznikl produkt zvaný jako přepínací regulátor. Název byl odvozen charakterem řízení spotřebiče, přepnutím fáze jednoho či druhého relé. Produkt představuje desku plošného spoje spolu s pasivními elektronickými komponenty a nedílnou logickou jednotkou mikroprocesoru ATmega328P, dále také jako MCU. Smyslem funkce regulátoru je spolehlivé a kontrolované zpracování řídící veličiny `stpt` z nadřazeného systému na výstup s korekcí pomocí zpětné vazby. V tomto případě se ovládá ventil a úhel jeho otevření představuje zpětnovazební veličinu `fdbk`.

Aby se dosáhlo definovaného chování výstupu regulátoru, byly zavedeny parametry. Těmi jsou maximální povolený rozdíl, hystereze, od řídící veličiny, limity intervalů vstupů, a výběr typu vstupu řídicí či zpětnovazební veličiny. Tyto parametry se liší pro dané řešení ovládání regulátorem a tudíž jsou charakterizovány jako jeho konfigurace. Přepínací regulátor pak dokážeme nejen dodat zákazníkovi (ZPA-RP) pro konkrétní problematiku, ale také ho nabízet jako univerzální realizaci fázového ovládání spousty spotřebičů.



Ilustrace 1: blokové předvedení funkčnosti přepínacího regulátoru.

2 Vývojové nástroje

2.1 Hardware – deska plošného spoje

Schéma regulátoru bylo tvořeno programem PADS Power spolu s výkresem desky plošného spoje. Deska regulátoru byla navržena a vyrobena v dvouvrstvém spoji. Rozměry byly zvoleny dle držáku DIN lišty k montáži v rozvaděči. Předpokladem je obsluha zařízení osobou minimálně poučenou, po otevření rozvaděče jsou na desce přístupné živé části.

2.2 Software – firmware

Zvolený osmibitový mikroprocesor ATmega328P rodiny AVR od firmy Atmel je velmi populární volbou. Podporu při vývoji k tomuto mikročipu tak nalezneme nejenom v jeho příslušném datasheetu, ale také v moderní platformě programování Arduino. Arduino platforma poskytuje základní univerzální funkce k manipulaci s piny mikroprocesoru a dalšími periferiemi, jako jsou interupty, časovače, sériový port, SPI či I2C komunikace. Vývojové prostředí Arduino IDE bohužel není zrovna ideální pro pokročilé používání C++. Vyžaduje projektové struktury stejných jmen, soubory typu (koncovky) .ino a nenabízí management vlastních knihoven a skriptů.

To vše se však mění s alternativou PlatformIO. Na své dokumentaci je PlatformIO charakterizováno jako multiplatformní, multiarchitekturový, multiframeworkový, profesionální nástroj pro inženýry vestavěných systémů a pro vývojáře softwaru, kteří píší aplikace pro vestavěné produkty [2]. PlatformIO je stavěno jako nástroj samotný a také jako plugin pro editory VSCode anebo CLion. U přepínacího regulátoru byl použit editor VSCode s rozšířením PlatformIO IDE. Programovalo se v jazyku C++ a konfiguračním souboru platformio.ini.

Nahrání kódu do mikroprocesoru se provádělo formou připojení ISP. ISP nám umožní plný přístup k paměti bez použití bootloaderu, části kódu spravující spuštění hlavního kódu či příjmu nového přes sériový port. ISP také umožní nastavení konfiguračních bitů, takzvaných „Fuse bits“. S těmito bity se například povolí vstup externího krystalu pro mikrokontrolér. PlatformIO spolu s Arduinem svojí otevřenou strukturou (Open Source) přitáhlo pozornost spousty vývojářů, a díky jejich úsilí existují velmi levné a také otevřené programátory ISP. U regulátoru bylo užito jednoho takového programátoru, a to klonu USBtinyISP s ATtiny44.

3 Vstupy

Jak již vychází z nároků zákazníka a snahy udržet zařízení univerzální pro více situací, regulátor obsahuje vstupy typu proudové smyčky a napěťové hladiny pro řídící i zpětnovazební veličinu. Jejich zpracování se odehrává jak v hardware, tak software části regulátoru.

3.1 Vstupní obvody

Obvody byly vytvořeny na základě úvahy užití vnitřní napěťové reference ATmega328P pro analog-digitální převodníky, a to 1,1V. Tato reference je stabilnější než klasická reference vůči napětí mikroprocesoru A_{Vcc} . Reference se dá více stabilizovat jejím připojením na A_{REF} zapsáním REFSn bitů ADMUX registru, jak nasvědčuje dokumentace. Pin A_{REF} se připojí přes oddělovací kondenzátor k nulovému potenciálu desky, což zajistí menší vliv šumu.

ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 24-3. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal V_{ref} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V Voltage Reference with external capacitor at AREF pin

Dokumentace 1: Popis užití kombinace bitů REFS1 a REFS0 v ADMUX [1]

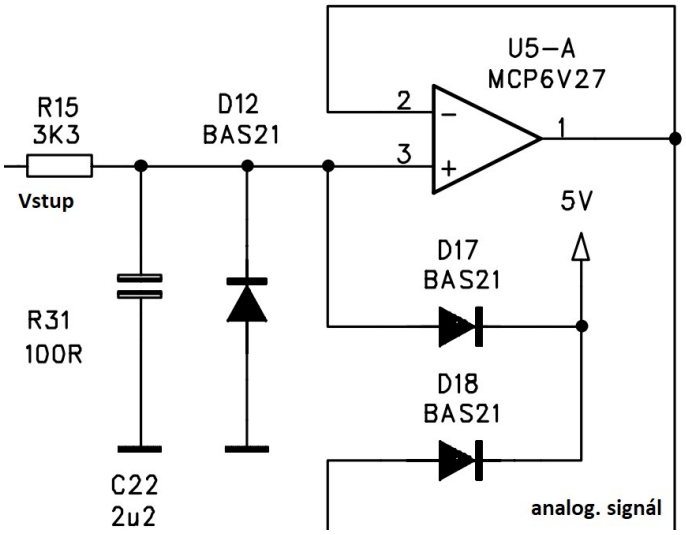


Schéma 1: RC filtr spolu s ochrannými diodami na straně signálu

Vstupy také obsahují RC filtr s funkčností dolní propustě k omezení rychlých jevů. Signál jde poté kolem ochranných diod. Dioda zapojená vůči nulovému potenciálu desky zajišťuje svedení proudu při výskytu záporného potenciálu. Další dioda zapojená vůči potenciálu 5V chrání proti připojení vysokého potenciálu. Nakonec je signál obvodově oddělen přes operační zesilovač v režimu napěťového sledovače (zesílení $A=1$), další dolní propusť RC filtru a ochrannou Schottkyho diodou proti zápornému napětí na straně mikrokontroléru.

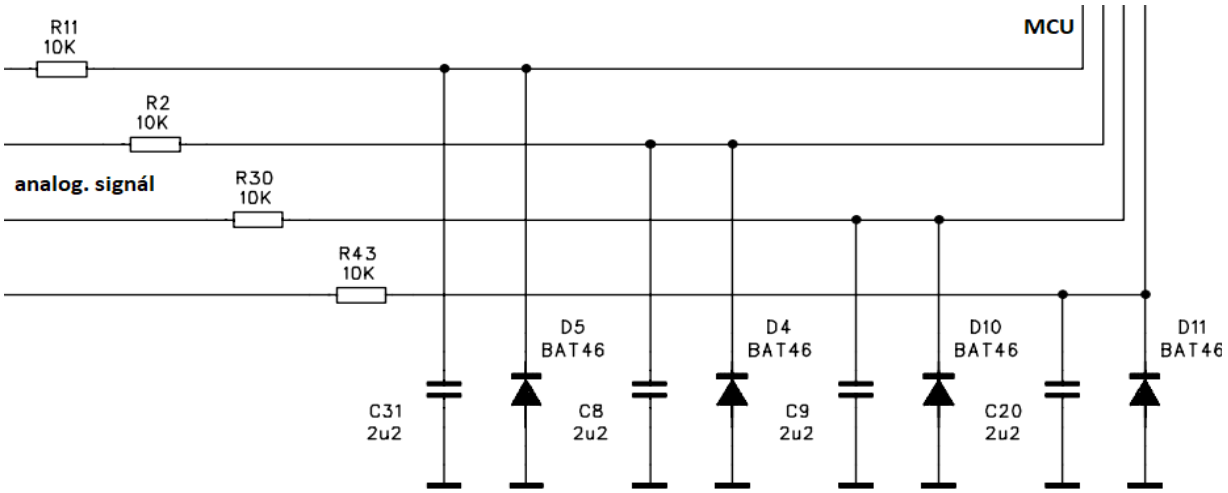


Schéma 2: RC filtry a ochranné Schottkyho diody na straně mikroprocesoru

3.1.1 Proudová smyčka (4-20mA)

Interpretuje analogovou hodnotu jako okamžitou velikost proudu ve smyčce obvodu. Historicky je proudová smyčka častý a spolehlivý způsob přenosu analogové informace v průmyslu. Je především oblíbená pro ne-degradaci informace/proudu vzdáleností a odolnost vůči rušení. Zároveň stanovením intervalu 4-20mA, dokážeme jednoduše určit poruchový stav, často například hodnotu menší než 4mA (rozpojení obvodu).

Pro univerzálnost je na desce regulátoru realizována možnost připojení jak pasivních proudových senzorů (připojení do vstupu, senzor se napájí z desky pomocí vývodu 24V), tak připojení pro aktivní proudová čidla či zdroj (připojení do vstupu a do vývodu nulového potenciálu). Oba proudové vstupy obsahují polyswitch k nedestruktivní ochraně proti přeproudu. Informace proudové smyčky je snímána na dvojici paralelních rezistorů.

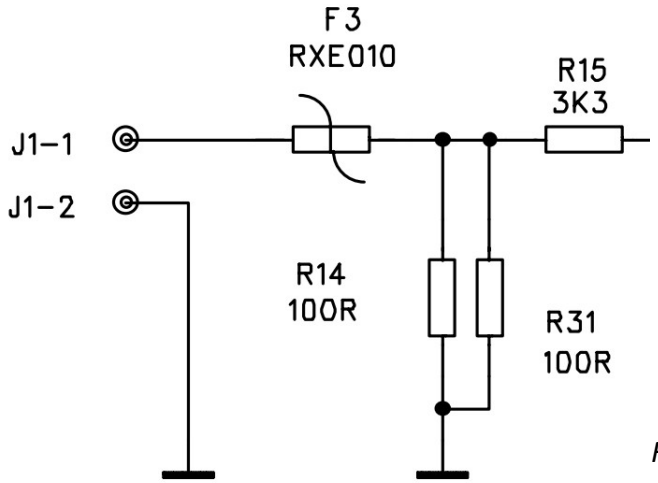


Schéma 3: Rezistory obvodu vstupu proudové smyčky

$$I = 20\,mA\,;\,U_{AREF} = 1{,}1\,V$$

$$R = \frac{U_{AREF}}{I} = \frac{1{,}1}{0{,}02} = 55\,\Omega$$

$$zvoleno\,50\,\Omega$$

$$R = \frac{R_{14} \cdot R_{31}}{R_{14} + R_{31}} = \frac{100 \cdot 100}{100 + 100} = 50\,\Omega$$

Rovnice 1: Postup volby velikosti odporů

Použití 50 ohmů jako odporu k snímání napětí neplní celkový rozsah AD převodníku. Tento nedostatek se snadno ošetří přes komunikaci s PC nastavením limit intervalu veličiny v mikroprocesoru. Připojením kalibrovaného zdroje proudu jsme pak schopni kompenzovat jednotlivé desky regulátoru na odchylky velikosti snímacího odporu.

3.1.2 Napěťová hladina (0-10V)

Jedná se o nejjednodušší způsob zpracování informace elektrickým signálem a to přímo jeho velikostí okamžitého napětí. Rozsah 0-10V byl zvolen z důvodu jeho užití a standardizace v průmyslu. Zároveň velikostí rozsahu dodává lepší odolnost vůči rušení. Za cenu jednoduchosti se tento signál nedá vést daleko, přílišný odpor vedení by nám omezil rozsah veličiny/napětí.

V regulátoru je vstup pro napěťovou hladinu zpracován děličem napětí. Pro omezení vstupního proudu byl zvolen velký odpor 39kΩ a k němu dopočítán druhý odpor následovně:

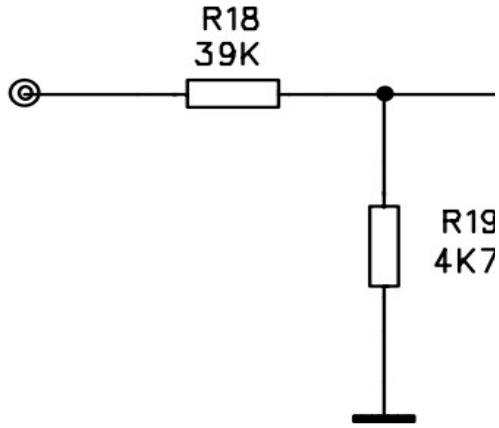


Schéma 4: Rezistory obvodu vstupu napěťové hladiny, R20 patří druhému napěťovému vstupu

$$U = 10\,V\,;\,U_{AREF} = 1{,}1\,V\,;\,R_{18} = 39\,k\Omega$$

$$U_{AREF} = U \cdot \frac{R_{19}}{R_{18} + R_{19}}$$

$$\vdots$$

$$R_{19} = \frac{R_{18} \cdot \frac{U_{AREF}}{U}}{1 - \frac{U_{AREF}}{U}} = \frac{39000 \cdot \frac{1{,}1}{10}}{1 - \frac{1{,}1}{10}} = 4820{,}225\,\Omega$$

$$zvoleno\,4{,}7\,k\Omega$$

Rovnice 2: Postup volby rezistorů napěťového děliče

Jako u proudového vstupu bylo použito menšího odporu k záznamu lehce menšího napětí než je plné $U_{AREF} = 1{,}1\,V$, později obdobně softwarově nakalibrováno koncovými polohami.

3.1.3 Manuální ovládání – tlačítka

K zajištění manuální operace regulátoru dle zadání deska disponuje dvěma mikrospínači. K definování jasného logického stavu potřebuje vstupní pin mikroprocesoru referenci v obou logických stavech. Toho se dosáhne pomocným rezistorem, který dodává referenci při rozpojeném stavu tlačítka. Dle typu reference pak existuje názvosloví zapojení s pull-up či pull-down rezistorem.

Jak je vidět, schéma regulátoru neobsahuje žádné pull-up či pull-down rezistory, a jejich stav by pak byl nejistý (takzvaný floating state). Zvolené piny tlačítek jsou totiž interně napojeny k pull-up rezistoru ATmega328P a tak stavu jedničky.

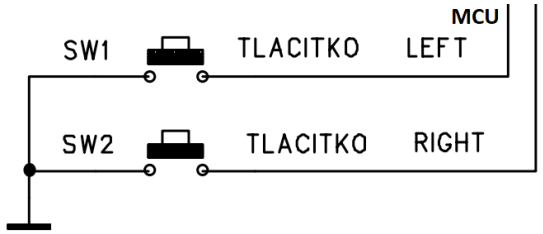


Schéma 5: Obvod tlačítek na regulátoru, vstupy vedeny přímo na mikrokontrolér

3.2 Programová abstrakce

Oproti hardware obvodové části vstupů lze v software řešit vstup univerzálně. C++, neboli takzvané C s třídami („C with classes“), svým objektovým přístupem nabízí možnosti logické a čitelné programové abstrakce daných komponentů. Objekty lze dále vkládat a použít v jiných třídách, stejně jako datové typy.

3.2.1 Analogové vstupy, struktura input

Stavěny jednotnou strukturou, jelikož proudová smyčka se od napěťové hladiny z pohledu mikrokontroleru neliší (jak již řešeno, mikroprocesoru se dostane napětí v rozsahu 0 až cca 1,1V s analogovou referencí 1,1V). AD převodník mikroprocesoru ATmega328P dosahuje 10 bitového rozlišení vůči referenci. Tím pádem se programové hodnoty hýbají v celočíselném rozsahu 0 až 1023.

I přes obvodové ošetření dodávaný signál stále může obsahovat určitý šum, a to zvlášť v klidovém stavu, kdy signál mohou ovlivnit vnější jevy, či šum vznikne případně samotným analog-digitál převodem. Programové řešení logiky regulátoru mikrokontrolérem nabízí možnost algoritmických filtrací, či jiného zpracování časově proměnného signálu.

Zde knihovna ResponsiveAnalogRead[1][8] platformy Arduino zvolená k eliminaci šumu u regulátoru disponuje skvělými vlastnostmi, jak již zmíněno u bodů její motivační části dokumentace:

- Softwarové odfiltrování šumu a zamezení jeho vlivu na změnu výstupní hodnoty
- Snaha o zachování poměrně rychlé reakce na skutečné změny hodnoty.
- Prevence skokového chování výstupní hodnoty. Preferujeme plynulou, spojitou změnu.

1 V případě elektronické verze - interaktivní prezentace algoritmu https://codepen.io/dxinteractive/full/zBEbpP

```
struct input {
  bool active;
  ResponsiveAnalogRead adc;
  float precentage;
  limit lim;
  error errState;

  float getVal() {return precentage;}
  void update() {
    adc.update();
    precentage = constrain(
      map(adc.getValue(), lim.low, lim.high, 0, 10000),
      0,
      10000
    ) / 100.0;

    if(adc.getValue() < lim.low - limReserve) {
      errState = belowLim;
    } else if(adc.getValue() > lim.high + limReserve) {
      errState = aboveLim;
    } else {
      errState = noErr;
    }
  }
};
```

Kód 1: Struktura `input`, která má na starosti zpracování vstupu.

`input` je základní strukturou k čtení a porovnávání veličin. Obsahuje následující prvky:

- binární informaci `active`, která určuje pro zbytek algoritmu, zda je tento vstup aktivní.
- Objekt `adc` typu `ResponsiveAnalogRead`, ke čtení a operacím vstupního signálu.
- Pracovní hodnotu `precentage` desetinného čísla v rozsahu 0 až 100. Slouží pro porovnávání řídící a zpětnovazební veličiny, a také k jasnému zobrazení stavů zmíněných veličin přes sériovou komunikaci, či případně budoucí displej.
- Objekt `lim` typu `limit` uchovávající data koncových poloh.

```
struct limit {
  int16_t low, high;
};
```

Kód 2: Datová struktura horní a spodní limity

- Enumerace `errState` typu `error`. Určuje chybový stav vstupu, který je dále zobrazen optickou vizualizací. Současné chyby charakterizují přítomnost veličiny mimo konfigurovaný interval a neměnnost zpětnovazební veličiny vůči řídící.

```
enum error {noErr, belowLim, aboveLim, notMoving};
```

Kód 3: Enumerace chybových stavů

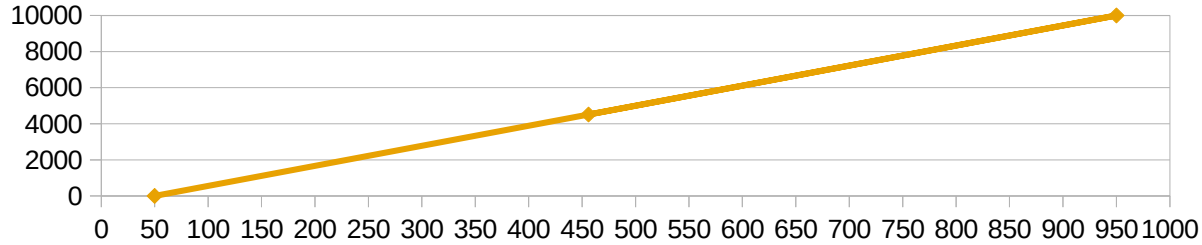
- Getter `getVal()` vracející `precentage`.
- Funkce `update()` zpracovávající čtenou veličinu.

Mimo výše zmíněné proměnné se logika vyhodnocení samotného vstupu odehrává ve funkci `update()`. Zde se provede výpočet/obnova `adc.update()` objektu `adc` s logikou knihovny ResponsiveAnalogRead [8]. Z objektu je poté možné získat zpracovanou hodnotu pomocí `adc.getValue()`. Hodnota je zároveň kombinací funkcí `constrain(map(...))` lineárně interpolována a omezena z intervalu s konfigurovanými koncovými polohami na interval 0.0 až 100.0. Kód obsahuje celočíselné mapování s hodnotami `10000`, jelikož její interní definice Arduina je stavěna s `long` typem vstupních parametrů. Interpolovaná a omezená hodnota je následně uložena do proměnné `precentage`. S tou se nakonec vyhodnocuje chybový stav enumerace `errState`.

```
long map(long x, long in_min, long in_max, long out_min, long out_max) {
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

Kód 4: Definice funkce `map()` v Arduinu

_	min	max	y = map(x)
in (x)	50	950	456
out (y)	0	10000	4511.11



Ilustrace 2: Grafická ukázka lineární interpolace funkce `map()`

Se strukturou `input` regulátor může zpracovat všechny své analogové vstupy a vyhodnotit je na absolutní veličiny spolu s definicemi chyb.

```
friend bool operator==(input &obj1, input &obj2) {
  uint8_t *ptr1 = (uint8_t *) &obj1, *ptr2 = (uint8_t *) &obj2;
  return (*ptr1 == *ptr2);
}
```

Kód 5: Přetížení operátoru přímé rovnosti pro `input`

V dalších kapitolách chceme navíc volit typ vstupu řídicí a zpětnovazební veličiny. Jelikož typy vstupu se mohou lišit rozsahy, jsou k jejich kombinacím přiřazeny jednotlivé hodnoty povolené rozdíly/hystereze. K zjištění, jestli je aktivní vstup typu proudové smyčky či napěťové hladiny, vypomůže tento přetížený operátor rovnosti. Testuje, zdali se adresy objektů `input` shodují.

3.2.2 Struktura module

K jednoduchému použití řídicí `stpt` a zpětnovazební veličiny `fdbk` existuje struktura `module`.

```
struct module {
  input c, v;

  input &getInput() {return c.active ? c : v;}

  void setInput(input &_input) {
    c.active = 0;
    v.active = 0;
    _input.active = 1;
  }
} stpt, fdbk;
```

Kód 6: Struktura `module` *, zároveň s vytvořením její instance* `stpt` *a* `fdbk`

Jejím úkolem je uchovat `input` strukturu vstupů obou veličin a pomocné funkce. Obsahuje tedy:

- `input c` pro vstup proudové smyčky $c \Rightarrow current$ a obdobně `input v` pro vstup napěťové hladiny $v \Rightarrow voltage$
- `&getInput()` vracející objekt `input` s pravdivou hodnotou `active`. Předpokladem je, že se musí zabránit stavu, kdyby došlo k pravdivým hodnotám `active` obou vstupů.
- Funkce `setInput()` k nastavení chtěného aktivního vstupu. Předem resetuje hodnoty `active` pro oba vstupy, poté nastaví pravdivý stav pro zvolený objekt vstupu `input`.

3.2.3 Struktura hysteresis

K volitelné hodnotě porovnání veličin `stpt` a `fdbk` regulátor obsahuje strukturu `hysteresis`.

```
struct hysteresis {
  float c, v, h;
} hyst;
```

Kód 7: Struktura `hysteresis` s hodnotami pro veškeré kombinace

Výstupní hodnota `input` objektů je typu `float` a tudíž tímto typem deklarujeme i povolené hystereze. V regulátoru jsou zavedené celkem tři proměnné hodnoty hystereze k porovnání kombinací:

- `c` - `stpt` a `fdbk` jako vstupy proudové smyčky
- `v` - `stpt` a `fdbk` jako vstupy napěťové hladiny
- `h` - obě kombinace vstupu jedné napěťové hladiny a jedné proudové smyčky

O správný výběr typu hystereze se stará funkce `getHyst()`. Zavolá si aktivní instanci typu vstupu a výše zmíněným přetíženým operátorem objektu `input` ji porovná vůči proudové a napěťové instanci.

3.2.4 Logické vstupy

Veškeré akce místního manuálního ovládání spadají pod stavové operace dvou přítomných mikrospínačů. Snaha splnit zadání zákazníka vytvořila následující myšlenku akcí tlačítek:

- Vstup do módu manuálního ovládání stiskem jakéhokoliv tlačítka.
- V manuálním módu - sepnutí relé korespondující k sepnutému tlačítku po dobu jeho sepnutí.
- Návrat do automatického režimu čtení řídící veličiny stisknutím obou tlačítek po určitou dobu.

K jasnějšímu chování jsme se vyvarovali akcí tlačítek zahrnující více stisknutí a podobně. Ke korektnímu a jednoduchému vyhodnocení těchto akcí vypomáhá otevřená knihovna EasyButton [7].

```
#include <EasyButton.h>
EasyButton btn1(BUTTON1_PIN);
EasyButton btn2(BUTTON2_PIN);
```

Kód 8: Tvorba objektu mikrospínačů s argumentem náležících pinů

Po deklaraci objektů typu `EasyButton`, s nutným argumentem pinu mikrospínače, získáme přístup k jejich velmi užitečným funkcím `wasPressed()` a `pressedFor(x)`. Dle popisu dokumentace fungují následovným způsobem:

- `wasPressed()` vrátí pravdu, pokud bylo tlačítko v posledním čtení stisknuté.
- `pressedFor(x)` vrátí pravdu, pokud bylo tlačítko stisknuté a dále stisknuté po dobu argumentu `x`.

Dále knihovna zajišťuje takzvanou vlastnost „debounce“, zabrání vyhodnocení rychlých zákmitů, jako stisknutí tlačítka. V konstruktoru objektu `EasyButton` je možná volba minimálního rozpoznaného času na stisk `debounce_time`, avšak standardní argument `35` milisekund je již vhodným časem. S přítomností pull-up logiky se zároveň hodí invertování logiky parametrem `active_low = true`, stisknutí tlačítka připojí nulový potenciál, tudíž aktivně nízkou úroveň.

```
EasyButton(uint8_t pin, uint32_t debounce_time = 35, bool pullup_enable = tr
ue, bool active_low = true)
```

Kód 9: Konstruktor objektu `EasyButton`, obsahuje standardní argumenty pro správnou logiku

4 Výstupy

Hlavním funkčním výstupem regulátoru je dvojice relé. Dalším výstupem je PWM (pulsně šířková modulace) ovládání galvanicky oddělené proudové smyčky, pro signalizaci na velín. Posledně jsou na desce regulátoru přítomny LED pro optickou signalizaci výstupů, stavu zpětnovazební veličiny, manuálního ovládání a chybových stavů.

4.1 Výstupní obvody

4.1.1 Relé

Relé v sepnutém stavu dodávají na své korespondující výstupy fázi napájení regulátoru L (230V AC). Připojené spotřebiče jsou tím pádem chráněny pojistkou a varistorem proti přepětí, stejně jako zdroj regulátoru. Obě relé jsou také vybaveny na straně připojení zhášecí sériovou RC kombinací vůči nulovému přívodu síťového rozvodu N. Zhášecí obvod omezuje špičky při rozpojování indukčností.

Špičky tvořené rozpojením indukce samotného relé jsou omezeny jejich ovládacím integrovaným obvodem ULN2804. Jedná se o pole darlingtonových NPN tranzistorů s antiparalelní diodou na výstupu. Obvod je napájen z 24V DC, dle stavů obvodu je toto napojení připojeno na výstupy O_n .

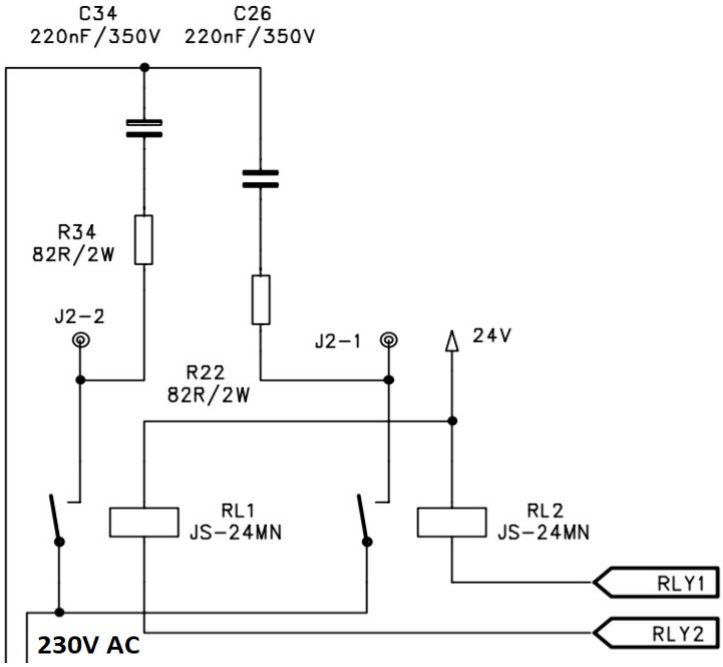


Schéma 6: Relé, jejich výstupní piny a zhášecí obvody

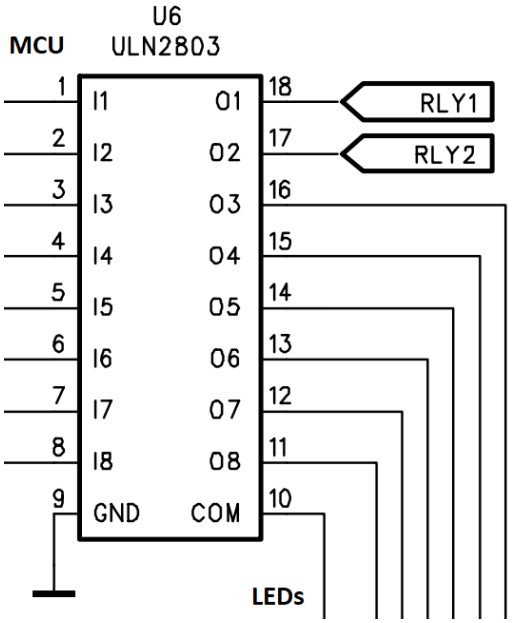


Schéma 7: Obvod ULN2804 spínající relé

Dvojice relé výstupu fáze dodává velikou variabilitu ovládání, jako například:

- Směrové ovládání trojfázového motoru pomocí nadřazených obvodů, přehození fáze externími stykači napájenými relé regulátoru.
- Směrové ovládání trojfázového motoru v zapojení s kondenzátorem, umožnující jednofázové napájení.
- Zapojení s výstupy do H můstku k ovládání jednofázového motoru.
- Alternativní funkce jako řízení tepelného tělesa jedním výstupem a zpětnou vazbou senzorem, nebo čerpadla se senzorem hladiny vody.

4.1.2 PWM – proudová smyčka

Obvod výstupu proudové smyčky zpětnovazební veličiny do nadřazeného systému, v případě zákazníka, jako signalizace na velín. Fundamentální částí obvodu proudové smyčky je čip XTR115, který při konstantním napětí ovlivňuje velikost proudu ve smyčce změnou odporu interního či externího výkonového prvku/tranzistoru. XTR115 se pasivně napájí z obvodu proudové smyčky na základě jejího rozsahu 4-20mA.

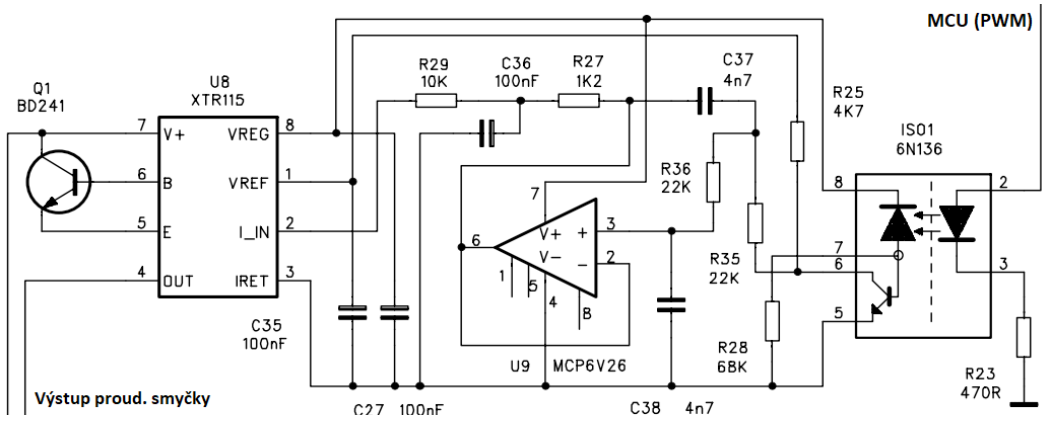


Schéma 8: Obvod proudové smyčky s XTR115

Přívod informace regulátoru je galvanicky oddělen rychle reagujícím optočlenem. Integrovaný obvod se řídí proudovým vstupem I_IN z výstupu optočlenu přes aktivní dolní propust druhého řádu [6].

4.1.3 LED optická signalizace

Šest LED v sérii se svými omezujícími odpory jsou zapojeny přes zbývající piny obvodu ULN2804. Při pravdivém stavu na vstupu obvodu jsou tak napájeny 24V DC. Důvodem nepřímého napájení z mikroprocesoru, bylo omezení výkonu a tak tepla vedeného přes lineární převodník 5V DC zdroje regulátoru.

4.2 Programová abstrakce

4.2.1 Relé

Důležitou podmínkou používání dvojice fázového ovládání, byla myšlenka zamezení stavu sepnutí obou relé. Prvním krokem k neomylné jistotě, bylo tedy zavedení jednoduchého makra, kdy jeden pin resetujeme dříve, než nastavíme druhý.

K ovládání spotřebiče regulátorem se vždy v jeden moment využívá jedno či žádné relé. Z této reality byla dvojice relé programově přirovnána enumeraci jejich možných stavů. Hodnotou této enumerace se dále makrem `digitalSwitch()` a funkcí `relayOut()` ovládají výstupy regulátoru.

```
#define digitalSwitch(pinOff, pinOn) \
  digitalWrite(pinOff, 0); \
  digitalWrite(pinOn, 1);
```

Kód 10: makro nejdříve resetu a pak setu dvou volených pinů

```
enum relay {noRel, rel1, rel2} relState = noRel;
```

Kód 11: Enumerace stavů relé

```
void SwitchingRegulator::relayOut() {
  switch(relState) {
  case rel1:
    digitalSwitch(rel2Pin, rel1Pin);
    break;
  case rel2:
    digitalSwitch(rel1Pin, rel2Pin);
    break;
  case noRel:
    digitalWrite(rel1Pin, 0);
    digitalWrite(rel2Pin, 0);
    break;
  }
}
```

Kód 12: Funkce nastavující a resetující výstupy relé dle programového relé stavu

Funkce čistě sestává ze switch-case bloku se vstupem požadovaného stavu relé. Dle něj se:

- `rel1` - prvně se zajistí rozepnutí relé 2, až poté sepne relé 1
- `rel2` - prvně se zajistí rozepnutí relé 1, obdobně se poté sepne relé 2
- `noRel` - obě relé se rozepnou, cílem je klidový stav

4.2.2 PWM – proudová smyčka

Řízení proudové smyčky PWM signálem využívá pomocných funkcí platformy Arduino. Na pinech podporujících PWM výstup lze poté jednoduše zavolat funkci.

```
void analogWrite(uint8_t pin, int value);
```

Kód 13: Funkce analogového výstupu platformy Arduino [4]

Dle arduino reference tato funkce s použitým pinem 10 (`#define FEEDBACKPWM_PIN 10`) generuje PWM signál s frekvencí 490 Hz a 8 bitovým rozlišením střídy `value` (0-255). Zpětnovazební veličina se poté jednoduše replikuje jako výstup funkcí `map()` s limitami aktivního výstupu a surovou hodnotou aktivního výstupu z `fdbk.getInput().adc.getValue()`.

4.2.3 LED optická signalizace

Logika optické vizualizace je situována v hlavičkovém souboru `Ledstatus.h`. Neblokujícího běhu je zde dosaženo objekty `DelayTimer`. Tyto pomocné objekty, využívají funkci `millis()` [4] k vyhodnocení svých intervalů, stavů. Blikání LED používá konkrétně funkci `isToggled()`, která funguje obdobně jako klopný obvod.

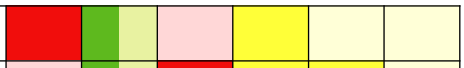
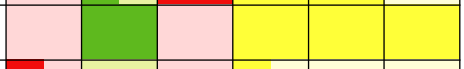
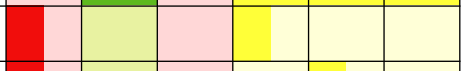
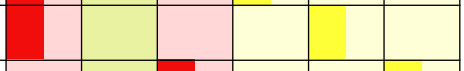
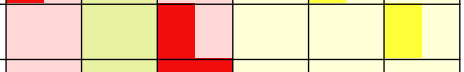
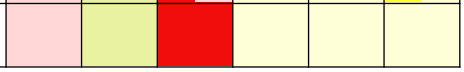
```
DelayTimer blinkTimer(200);
DelayTimer stpt_fdbk_Timer(2000);

bool ledbar(uint8_t multiplier) {
  return reg.getDist() < reg.getHyst() * multiplier;
}
```

Kód 14: Objekty `DelayTimer` blikání a vizualizace obou veličin, funkce `ledbar()` vizualizace vzdálenosti veličin

Šestice LED vyjadřuje stav regulátoru následujícím způsobem:

- Zelená LED v nechybovém stavu svítí při rovnosti vstupních veličin. Při aktivním stavu jednoho z relé bliká.
- Žluté LED v nechybovém stavu zobrazují blízkost veličin, při korektní funkci se postupně rozsvěcují. V chybovém stavu blikají pro jednotlivé stavy chyb.
- Červené LED v nechybovém stavu svítí při sepnutí jejich přiřazeného relé. V chybovém stavu blikají pro přiřazenou řídící či zpětnovazební veličinu kde se chyba nachází.

Vstupy jsou v rozsahu či regulátor v manuální režimu	Relé 1 sepnuté, diference veličin velká	
	Relé 2 sepnuté, diference veličin malá	
	Diference veličin žádná	
Vstupy jsou mimo rozsah (chybový stav) a regulátor je v auto režimu. Červené LED určují řídící či zpětnovazební veličinu.	Řídící veličina pod limitou	
	Řídící veličina nad limitou	
	Zpětnovazební veličina se nehýbe	
	Zpětnovazební veličina není chybou	

Ilustrace 3: Ukázka různých stavů s jejich LED vizualizací

Výsledkem snahy indikovat každý stav aktivitou alespoň jedné z LED se v chybovém stavu při přepínání mezi veličinami značí neproblematická svícením červené LED.

5 Zdroje

Deska regulátoru se napájí z rozvodné sítě 230V AC. Vstup je na začátku přepěťově chráněn kombinací pojistky a varistoru, dále je větví k výstupům přes relé kontakty a k síťovému filtru. Síťový filtr zabraňuje průstupu rušení na následující spínací zdroj 230V AC - 24V DC, a komponenty jím napájené.

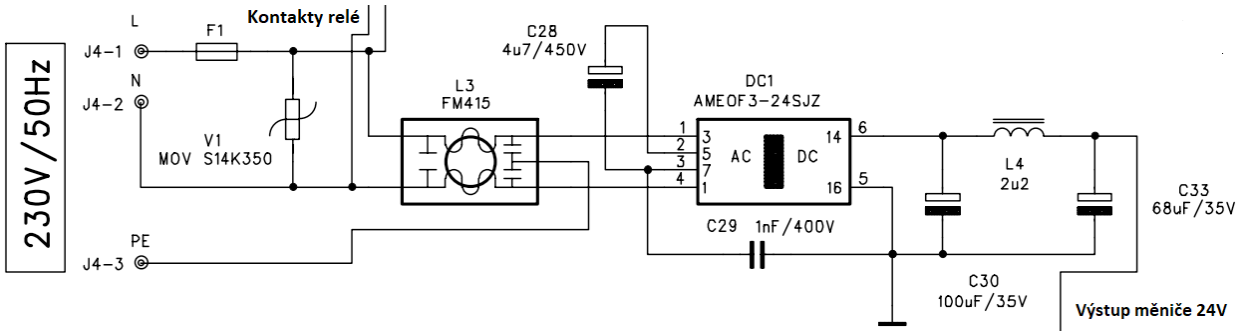


Schéma 9: Obvod větvení sítě, filtru a spínacího zdroje

Rozvod stejnosměrného napětí sestává z 24 a 5 voltů. Vzhledem ke zpracování analogových signálů na straně mikroprocesoru bylo rozhodnuto použití lineárního stabilizátoru k tvorbě 5V. Výstup spínaného zdroje má v sérii dodatečný polyswitch k ochraně přeproudu, pak je 24V dále větveno k napájení relé, diod a také k lineárnímu stabilizátoru. Lineární stabilizátor je obklopen filtračními kondenzátory a LC filtrem dolní propusti u vstupu k odstranění případných dynamických jevů.

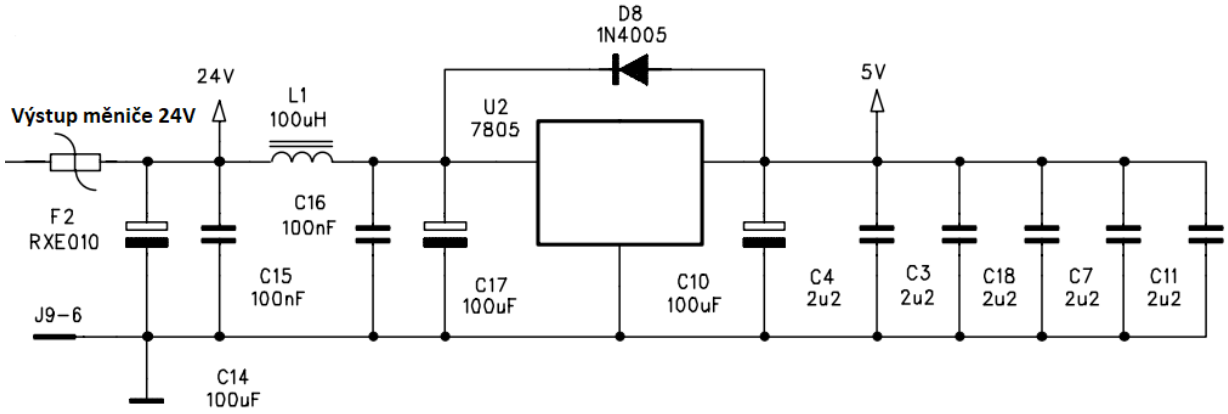


Schéma 10: Obvod lineárního stabilizátoru

Napětí 5V a 24V je zároveň přes další polyswitche vyvedeno k napájení připojených senzorů. Například pasivní senzory proudové smyčky zapojeny na 24V a vstup proudové smyčky.

K zachování linearity OZ a plnému rozsahu k nulovému potenciálu obsahuje regulátor integrovaný obvod s kondenzátorovou pumpou k tvorbě záporného napětí -5V. To je přes křemíkovou diodu omezeno na -0.7V a dodáno jako záporný potenciál napájení OZ.

6 Program a jeho hlavní smyčka

Ať už realizované knihovnami či vlastním časováním, principielní funkčnosti regulátoru vyžadují rychlou odezvu, reakci. Jelikož je logickou jednotkou desky mikrokontrolér ATmega328P, který nemá více jader či jiné možnosti paralelního vykonání instrukcí, je nutné ho naprogramovat s myšlenkou neblokujících funkcí. Příkladem je použití objektu `DelayTimer` u optické signalizace, místo blokující funkce `delay()` [4].

Projektová struktura PlatformIO sestává ze zdrojové složky **src** a **lib** k organizaci projektových knihoven, například `SwitchingRegulator`. Dodatečně projekt ještě obsahuje složku **include** pro organizaci hlavičkových souborů, vhodnou pro definice pinů. Program začíná hlavním souborem main.cpp.

```
/*
  Project: Switching Regulator
  Version: 1.5.0
  Author:  Ivan Sadílek, sadilekivan@gmail.com
*/
#include <Arduino.h>
#include <avr/wdt.h>
#include <Tools.h>
#include <DefinePins.h>

#include <EasyButton.h>
EasyButton btn1(BUTTON1_PIN);
EasyButton btn2(BUTTON2_PIN);

#include <SwitchingRegulator.h>
SwitchingRegulator reg(SETPOINTC_PIN, FEEDBACKC_PIN, SETPOINTV_PIN, FEEDBACK
V_PIN, RELAY1_PIN, RELAY2_PIN, FEEDBACKPWM_PIN, 0);

#include <LedStatus.h>
```

Kód 15: Počátek souboru main.cpp

Kód postupně vkládá platformu Arduina s její funkčností, pomocné avr watchdog timer funkce a definice pinů regulátoru. Následně zmíněnou `EasyButton` knihovnu pro mikrospínače a objekt `SwitchingRegulator` celkové abstrakce desky regulátoru s vstupy a výstupy. Poté se vloží oddělená logika LED optické signalizace. main.cpp dále definuje příkazovou komunikaci funkcí `cmdTree()` ukázanou níže v další kapitole.

Programová smyčka v Arduino platformě nabízí alternativu hlavní funkce `main()` funkcemi `setup()` a `loop()`. Jak vyznívá z anglických názvů `setup()` proběhne jednou při spuštění programu a poté se opakuje `loop()` do nekonečna, nebo odpojení napájení.

6.1 Setup - příprava

```
void setup() {
  analogReference(INTERNAL);
  pinMode(RLED1_PIN, OUTPUT);
  pinMode(RLED2_PIN, OUTPUT);
  pinMode(YLED_PIN, OUTPUT);
  pinMode(GLED1_PIN, OUTPUT);
  pinMode(GLED2_PIN, OUTPUT);
  pinMode(GLED3_PIN, OUTPUT);
  btn1.begin();
  btn2.begin();

  cmd.begin("SwitchingRegulator", Serial);
  wdt_enable(WDTO_60MS);
}
```

Kód 16: Jednorázová funkce spuštění `setup()`

Funkce `setup()` připravuje hardware mikroprocesoru regulátoru. Zmíněná vnitřní analogová reference se nastaví funkcí `analogRefrence(INTERNAL)` z platformy Arduino. S `pinMode()` se nastaví piny LED na výstupy a dle dokumentace `EasyButton` se připraví objekty mikrospínačů svými funkcemi `begin()`. Obdobně je připraven globální objekt `cmd` z vlastní knihovny `CmdManager`. Jeho parametry jsou: textový řetězec názvu zařízení, globální objekt `Serial` platformy Arduino. Poslední řádek nastaví nastaví watchdog timer (časovač hlídacího psa).

6.2 Loop - smyčka

```
void loop() {
  wdt_reset();
  btn1.read();
  btn2.read();
  switch (reg.mode) {
    case reg.automat:
      reg.update();
      if(btn1.wasPressed() || btn2.wasPressed())
        reg.mode = reg.manual;
      break;
    case reg.manual:
      if(btn1.pressedFor(250) && btn2.releasedFor(250)) {
        reg.override(reg.rel1);
      } else if(btn2.pressedFor(250) && btn1.releasedFor(250)) {
        reg.override(reg.rel2);
      } else {
        reg.override(reg.noRel);
      }
      if(btn1.pressedFor(1000) && btn2.pressedFor(1000))
        reg.mode = reg.automat;
      break;
  }
  ledStatus();
  Tools::update();
}
```

Kód 17: Hlavní smyčka programu `loop()`

Opakující se funkce `loop()` při správném chodu resetuje watchdog timer a provádí čtení, obnovu potřebných objektů. S pomocí objektů mikrospínačů se zde vyhodnocuje režim regulátoru, jak popsáno myšlenkou u kapitoly programové abstrakce logických vstupů (3.2.4). Daný režim a stavy se vizualizují pomocí LED, funkcí `ledStatus()`.

Metoda `Tools::update()` vztahující se na objekty vlastních knihoven jako `DelayTimer` a `CmdManager` postupně zavolá jednotlivé `update()` funkce. Důvodem je čitelnost a prevence zapomenutí zavolání `update()` funkce nějakého z objektů.

6.3 SwitchingRegulator – objektová abstrakce přepínacího regulátoru

K čitelnému a organizovanému kódu je funkčnost regulátoru oddělena od hlavního souboru main.cpp do vlastní projektové knihovny a třídy. Jak naznačeno v souvisejících blocích kódu s třídou `SwitchingRegulator` je zde kompletní logika čtení, výběru a zpracování vstupů `fdbk`, `stpt` spolu s porovnáním a nastavením výstupů `rel1`, `rel2`.

```
float SwitchingRegulator::getDist() {
  return abs(stpt.getInput().getVal() - fdbk.getInput().getVal());
}
```

Kód 18: Výpočet vzdálenosti zpetnovazební a řídící veličiny v normalizovaném intervalu

```
void SwitchingRegulator::calcStatus() {
  if(getError() == 0 && getDist() > getHyst()) {
    if(stpt.getInput().getVal() > fdbk.getInput().getVal()) {
      relState = rel1;
    } else {
      relState = rel2;
    }
  } else {
    relState = noRel;
  }
}
```

Kód 19: Vyhodnocení programového stavu relé na základě přítomnosti veličin ve vzájemném intervalu hystereze

Funkce `calcStatus()` v případě chyby s vyhodnocením z `getError()` odpojuje obě relé.

S pomocí všech funkcí regulátoru poté smyčková funkce `update()` zařídí jednotlivé čtení vstupů a zápis stavů výstupu dle parametrů.

Obdobou funkcí je `override(rel)`, ta však nezapisuje výstup dle definované logiky, ale nýbrž ho bere jako parametr. Je využita v manuálním režimu regulátoru s řízením mikrospínači.

```
bool SwitchingRegulator::getError() {
  return
    stpt.getInput().errState != noErr ||
    fdbk.getInput().errState != noErr;
}
```

Kód 20: Vyjádření stavu v chybě z chybové enumerace

```
enum mode {automat, manual} mode = automat;
```

Kód 21: Enumerace režimů regulátoru, auto a manuál

7 Komunikace

Hlavní předností regulátoru a budoucích produktů TCELE stavěných na mikrokontrolérech je uživatelsky jednoduchá, avšak velmi rozmanitá manipulace zařízení sériovým portem. Ideou je možnost nastavení přes různé programy sériového terminálu, či vlastní terminál firmy TCELE. Ten umožnuje automatický sken sériových portů k rozpoznání podporovaného zařízení a sériové rychlosti.

7.1 Obvod USB, USART

Komunikace mezi počítačem a mikroprocesorem není možná napřímo. K její realizaci je třeba svázat USB periferii počítače a disponující periferii ATmega328P, programovatelné USART. Toho se dosáhne vhodným převodníkem jako je například zvolený čip FT230X od FTDI. Mimo datovou komunikaci umožňuje nastavení užitečných identifikačních parametrů strany počítače.

Použitý obvod FT230X vychází z doporučeného schématu jeho datasheetu. Ze schématu se přebrala velikost odporů v sérii s USB datovými linkami a kapacita kondenzátoru pro tvorbu vnitřního napětí. Napájení FT230X je úmyslně odděleno a vedeno ze strany konektoru od počítače. Lehce se tak šetří jeho spotřeba a zajistí se jeho reset při připojení (reset v našem zájmu na základě zkušeností kolegy Ing. Tomáše Chvátala).

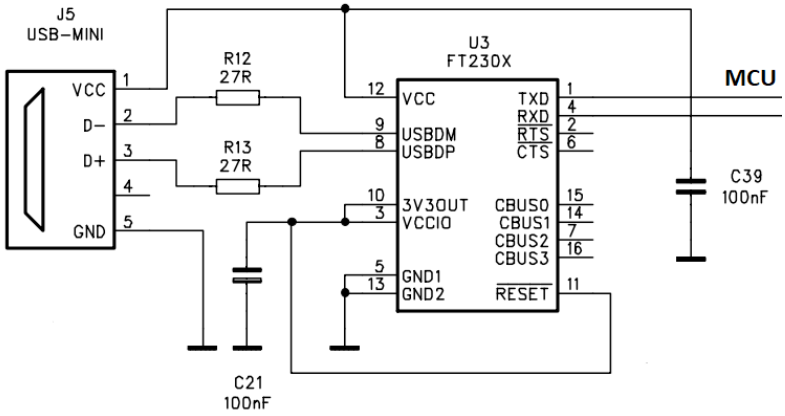


Schéma 11: Převodník USB na základní USART, FT230x

Výhodou integrovaných obvodů firmy FTDI je jejich přímá podpora na velmi často užívaném operačním systému Windows 10. Není tím pádem pro zákazníka nutné manuálně instalovat ovladače těchto zařízení, buď jsou již nainstalovány či případně je windows nainstaluje sám, pomocí služby Windows Update. Čip FT230X umožňuje také zápis informačních EEPROM parametrů pro ovladač PC. Pro produkt vhodné jako jasné rozpoznání zařízení na PC u zákazníka. Zápis se provádí na PC s programem FT_PROG.

7.2 Příkazová komunikace (Serial Command Interface – SCI)

Příkazová komunikace se zakládá na operacích čtení či zápisu parametru/ů. Případně chceme provést libovolnou část kódu - akci. Jelikož se struktura a funkce příkazů kompletně či částečně liší v jednotlivých projektech, je v našem zájmu si zavést určitý způsob, funkce, se kterými se programově vytvoří přístup k daným parametrům a akcím. K pokrytí velkého množství parametrů jsem se zároveň rozhodl, zavést možnost libovolně stromově větvit a řetězit jednotlivá koncová místa příkazu. Jinými slovy, příkazy připomínají objektový přístup s odlišností syntaxu děliče, jako mezery. Příkazy pak mohou nabírat tvar cesty jako:

- <čtení> <objekt1>
- <čtení> <objekt1> <objekt2> <objekt3>
- <zápis> <objekt1> <objekt2> <objekt3> <parametr či pole parametrů>
- <zápis> <objekt1> <objekt2> <parametr či pole parametrů>

Řešením problematiky komunikace vznikla knihovna `CmdManager.h` se svým příkazovým rozhraním SCI (Serial Command Interface) na zvoleném sériovém portu.

```
cmd.begin("SwitchingRegulator", Serial);
```

Kód 22: Připravující funkce `begin()` objektu příkazové komunikace

Knihovna poskytuje globální objekt `cmd`, se kterým se inicializuje příkazová komunikace funkcí `begin()`. Zde je nutné zvolit používaný sériový objekt (v mnoha případech je jen jeden a to `Serial` [4]. Volba je zde pro podporu desek s více objekty sériového portu, jako je například ESP32) a název zařízení, pod kterým se může identifikovat na sériovém terminálu. Údaj o verzi firmware a názvu zařízení je vždy přístupný na rezervovaných cestách:

- verze firmware `SELF SOFT`
- jméno zařízení `SELF NAME`
- jméno spolu se zařízením `SELF`

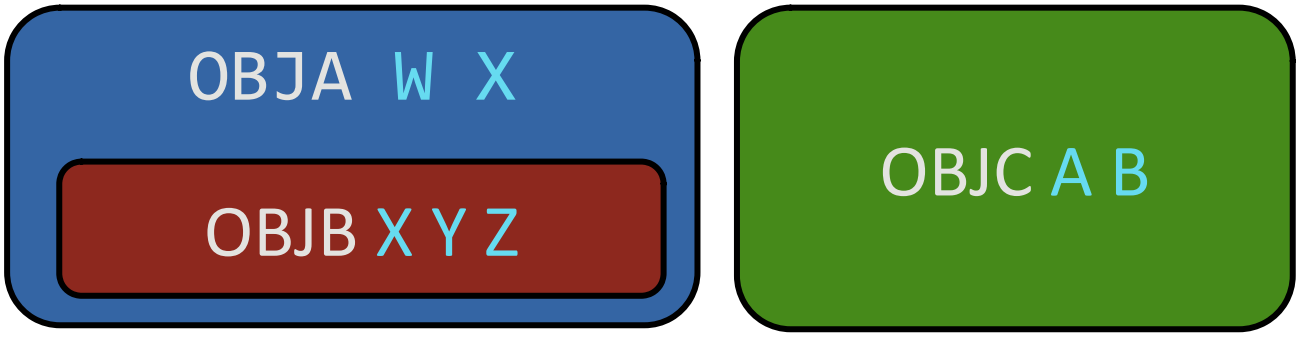
Mezi rezervované příkazy patří, ozvěna `ECHO`, reset mikroprocesoru `RESET` skokem na adresu 0, a informační příkaz pomoci `HELP` s obsahem stručného vysvětlení použití příkazů SCI. Častými příkazy jsou `SAVE` a `LOAD` pro ukládání a načítání dat. Nejsou rezervovány, ale otevřeny k naprogramování vlastní metody. Jejich akce jsou totiž pevně spjaté s konkrétním typem paměti. V našem případě využity u interní EEPROM paměti mikroprocesoru.

7.2.1 Názorná ukázka v sériovém terminálu

```
HH:MM:SS.sss - ECHO test 1, 2, 3
HH:MM:SS.sss - test 1, 2, 3
```

Komunikace 1: Testování SCI

Příkazem ozvěny `ECHO` je možné otestovat funkčnost příkazové komunikace. Jednoduše vrací svůj vstup zpátky přes sériový port do PC.



Ilustrace 3: Množinová vizualizace cest prvků OBJA, OBJB, OBJC

K výše vizualizovaným prvkům je možné příkazovými operacemi přistupovat následovně.

```
HH:MM:SS.sss - R OBJA
HH:MM:SS.sss – OBJA 8 0.0000000
HH:MM:SS.sss - R OBJC
HH:MM:SS.sss – OBJC 1 0
HH:MM:SS.sss - R OBJA OBJB
HH:MM:SS.sss - OBJA OBJB 0.0000000 130 458
```

Komunikace 2: Čtení parametrů příkazem `R`

Příkaz čtení specifikovaný charakterem `R` jako „read“ postupně hledá koncové místo. Začne hledáním `OBJA` v rámci celého stromu, vrací parametry `W X`. Stejným způsobem se dostaneme na parametry `A B` v `OBJC`. Dalším příkazem čtení se opět hledá `OBJA`, avšak potom se větví v rámci `OBJA` a hledáme `OBJB`. Po jeho nalezení vrací parametry spojené s cestou `OBJA OBJB` `X Y Z`, v tomto případě pole parametrů, jedna reálná a dvě celočíselné hodnoty oddělené mezerami.

```
HH:MM:SS.sss - W OBJA 2 3
HH:MM:SS.sss – OBJA 2 3.0000000
HH:MM:SS.sss - W OBJC 0 0
HH:MM:SS.sss – OBJC 0 0
HH:MM:SS.sss - W OBJA OBJB 1.5 33 0.2
HH:MM:SS.sss - OBJA OBJB 1.5000000 33 0
```

Komunikace 3: Zápis parametrů na jednotlivé cesty příkazem `W`

Obdobně je možné provést zápis s `W` „write“. Za příkaz dodáme parametry oddělené mezerou. Funkčnost SCI se v mnoha případech snaží vyrovnat s nekorektně zadanými parametry, jak je vidět třetí parametr je celočíselná hodnota, a dodaný parametr `0.2` se zkrátil/zdegradoval na `0` (neplatí zde zaokrouhlení, pouze se vezme použitelná část dodané hodnoty).

```
HH:MM:SS.sss - W OBJC
HH:MM:SS.sss - OBJC <> NaN <> NaN
HH:MM:SS.sss - W OBJA OBJB 1.5 ab
HH:MM:SS.sss - OBJA OBJB 1.5000000 <> NaN <> NaN
HH:MM:SS.sss - R OBJD
HH:MM:SS.sss - <OBJD> ?
HH:MM:SS.sss - W OBJD 356700
HH:MM:SS.sss - <OBJD 356700> ?
HH:MM:SS.sss - R OBJC OBJD
HH:MM:SS.sss – OBJC 1 0 <OBJD> ?
```

Komunikace 4: Výpis odpovědí SCI na chybně zadané příkazy

V případě nerozpoznání anebo nedodání očekávaného parametru anebo jeho části, vrátí příkaz zpětnou vazbu jako při správném zapsání spolu s informací o chybných parametrech, NaN (Not a Number). Při neúspěchu hledání cesty se výraz vrací na terminál jako nerozpoznaný.

Je nutné podotknout postup hledání cesty pomocí mezerových děličů zleva doprava. Jejím efektem jsou poslední dva řádky kde SCI neexistující podobjekt cesty vyvodí jako nerozpoznaný, ale vrátí rozpoznaný počátek cesty.

```
HH:MM:SS.sss - W REG L C 509 890 509 890
HH:MM:SS.sss - REG L C 509 890 509 890
HH:MM:SS.sss - W REG L V 980 1005 980 1005
HH:MM:SS.sss - REG L V 980 1005 980 1005
HH:MM:SS.sss - W REG H 2 5 15
HH:MM:SS.sss - REG H 2.0000000 5.0000000 15.0000000
HH:MM:SS.sss - W REG IN C C
HH:MM:SS.sss - REG IN C C
HH:MM:SS.sss - SAVE
HH:MM:SS.sss – R REG
HH:MM:SS.sss – REG C 43.2999990 43.0400010 V 60.0000000 64.0000000
```

Komunikace 5: Konkrétní použití SCI v sériovém terminálu

Zde je vidět užití příkazové komunikace při konfigurování regulátoru na testovací sestavu. Postupně se nastaví limity vstupů, jejich povolené hystereze a pak se volí typ vstupu. Nastavení se pro jistotu uloží do EEPROM a čtením `REG` cesty vidíme, že veličina řídící je shodná se zpětnovazební, v rámci tolerance povolené hystereze.

7.2.2 Implementace SCI

Po inicializaci objekt `cmd` s pomocí `Serial` objektu platformy Arduino ve funkci `update()` zkouší opakovaně číst buffer sériového portu. Očekává se přítomnost terminátoru, charakteru nového řádku `'\n'`, jako indikaci k zpracování příkazu. Když se tak stane, funkce uloží buffer jako objekt typu `String` do proměnné `str` a pokračuje dále analýzou tohoto příkazu. Postupnou iterací se testuje, o jakou operaci se jedná, zdali je cesta validní, a které akce má provést. Vitální funkcí v této iteraci je `passCmd()`, testuje shodu výrazu argumentu `cmd` s výrazem na počátku proměnné `str`.

```
bool CmdManager::passCmd(const String cmd) {
  if(str.startsWith(cmd) && (str == cmd || str[cmd.length()] == ' ')) {
    str.remove(0, cmd.length());
    str.trim();
    if(verbose)
      reply(cmd);
    return 1;
  } else {
    return 0;
  }
}
```

Kód 23: `passCmd()`, základní funkce ke zpracování příchozí komunikace

Výsledkem `passCmd()` se řídí řada pomocných funkcí k jednoduššímu programování příkazových cest. Implementace kódu v jazyku C++ zde umožnila šablonové psaní (`template`) a vektorové expanze argumentů (`...`). Díky těmto přednostem C++ je zdrojový kód `CmdManager.h` univerzálnější, mnohem čitelnější a při programování uživatelských cest ulehčuje schopnými funkcemi práci.

Pokud po průchodu analýzou v proměnné `str` zůstanou nějaké znaky, společně s rozpoznanými výrazy, se v `reply()` a `serialReply()` vypíší na sériový terminál.

7.2.3 Funkce k tvorbě příkazových cest

```
void setTree(cb_t callback) {tree = callback;}
```

Kód 24: `setTree()` k nastavení funkce s cestami příkazů k proměnným

V základním nastavení si objekt `cmd` při analýze příkazu v globálním prostoru zavolá funkci `cmdTree()`. V případě potřeby je možné změnit tuto funkci s `setTree()`.

```
#define newCmd(cmd) const String cmd = #cmd
```

Kód 25: Pomocné makro k založení příkazů

K tvorbě očekávaných příkazů knihovna obsahuje pomocné makro k založení proměnné, typu `String`, stejného názvu a obsahu. Odlehčuje tak zápis uvozovek v parametrech funkcí a v IDE umožňuje rychlé přejmenování. Rozbalení makra v programu regulátoru by vypadalo takto `newCmd(REG) => const String REG = "REG"`.

Šablonové funkce na specifikaci cest, odpovědí a parametrů jsou:

- `reply(args...)`, spojí své argumenty mezerou a přidá je k odpovědi pro `serialReply()`
- `dir(cmd, []{})`, zavolá svůj druhý argument, lambda funkci, pokud se první argument vyhodnotí s `passCmd()` jako pravdivý. Pokud pokračujeme programem v rámci lambda funkce, zapamatujeme si toto odvětvení programu. Při ukončení analýzy se můžeme rovnou vracet.
- `dirNoReturn(cmd, []{})` je obdobou funkce `dir()` bez ukončení za rámcem lambda funkce.
- `rw(cmd, args...)`, opět porovná první argument s cestou příkazu. Při úspěchu zapíše anebo čte dodaný parametr či pole parametrů.
- `ro(cmd, args...)`, obdoba `rw()` s ochranou proti zápisu.
- `readCB([]{})`, zavolá svou lambda funkci pokud příkaz provádí operaci čtení.
- `writeCB([]{})`, provede svou lambda funkci pokud příkaz zapisuje.

Rámec `cmdTree()` v programu regulátoru hojně využívá výše popsaných funkcí. Provádí se jimi veškerá konfigurace i identifikace zařízení. Vkládáním `dir()` funkcí do sebe argumentem lambda se efektivně tvoří větvení cest příkazů. K širšímu přizpůsobení parametrů, jako například zadávání charakteru k výběru typu vstupu, jsem použil kombinace `readCB()` s `reply()` ke čtení `writeCB()` spolu s `dirNoReturn()` k zápisu.

```
void cmdTree() {
  cmd.dir(REG, []{
    cmd.dir(L, []{
      cmd.rw(C, reg.stpt.c.lim.low, reg.stpt.c.lim.high, reg.fdbk.c.lim.low,
 reg.fdbk.c.lim.high);
      cmd.rw(V, reg.stpt.v.lim.low, reg.stpt.v.lim.high, reg.fdbk.v.lim.low,
 reg.fdbk.v.lim.high);
    });
    cmd.rw(H, reg.hyst.c, reg.hyst.v, reg.hyst.h);
    cmd.rw(PWM, reg.pwm.val, reg.pwm.lim.low, reg.pwm.lim.high);
    cmd.dir(IN, []{
      cmd.readCB([]{
        String inputStpt, inputFdbk;
        if(reg.stpt.getInput() == reg.stpt.c) inputStpt = "C"; else inputStp
t = "V";
        if(reg.fdbk.getInput() == reg.fdbk.c) inputFdbk = "C"; else inputFdb
k = "V";
        cmd.reply(inputStpt);
        cmd.reply(inputFdbk);
      });
      cmd.writeCB([]{
        cmd.dirNoReturn(C, []{reg.stpt.setInput(reg.stpt.c);});
        cmd.dirNoReturn(V, []{reg.stpt.setInput(reg.stpt.v);});

        cmd.dirNoReturn(C, []{reg.fdbk.setInput(reg.fdbk.c);});
        cmd.dirNoReturn(V, []{reg.fdbk.setInput(reg.fdbk.v);});
      });
    });
    cmd.dir(RAW, []{
      cmd.readCB([]{
        cmd.reply(C);
        cmd.reply(reg.stpt.c.adc.getValue());
        cmd.reply(reg.fdbk.c.adc.getValue());
        cmd.reply(V);
        cmd.reply(reg.stpt.v.adc.getValue());
        cmd.reply(reg.fdbk.v.adc.getValue());
      });
    });
    cmd.readCB([]{
      cmd.reply(C);
      cmd.reply(reg.stpt.c.getVal());
      cmd.reply(reg.fdbk.c.getVal());
      cmd.reply(V);
      cmd.reply(reg.stpt.v.getVal());
      cmd.reply(reg.fdbk.v.getVal());
    });
  });
  cmd.dir("SAVE", []{reg.saveData();});
  cmd.dir("LOAD", []{reg.loadData();});
}
```

Kód 26: Specifikace příkazových cest a jejich parametrů s `cmdTree()` přepínacího regulátoru

8 EEPROM paměť

S EEPROM (Elektricky mazatelná programovatelná paměť) je možné zachovat data konfigurace regulátoru přes cyklus vypnutí a zapnutí zařízení. Interní přítomnost paměti EEPROM byl jeden z faktorů při výběru mikrokontroléru, nechceme konfigurovat regulátor po každém zapnutí.

Ukládání parametrů zařizuje abstrakce regulátoru `SwitchingRegulator` svým objektem `EEPROMManager` z knihovny `EEPROMManager.h`. `EEPROMManager` obsluhuje logiku změny adresy při postupném čtení a zápisu proměnných do EEPROM. Dodané proměnné se zpracují funkcemi `get()` a `put()` objektu `EEPROM` platformy Arduino. Motivací této knihovny je ulehčení serializace při ukládání, načítání proměnných pomocí šablonových metod s variabilní délkou argumentů.

```
struct address {
  uint16_t start = 0, current = 0,  end = 0;
} address;
```

Kód 27: Struktura `address` s daty k orientaci pamětí EEPROM

Struktura `address` přebírá z konstruktoru objektu `start`, načítáním a ukládáním dat inkrementuje proměnnou `current` a na konci řetězce ji zapíše do proměnné `end`. Další objekty `EEPROMManager` mohou tak použít konečnou adresu pro další zápis, čtení.

```
void SwitchingRegulator::saveData() {
  mem.save(
    stpt.c.lim,
    fdbk.c.lim,
    stpt.v.lim,
    fdbk.v.lim,
    stpt.c.active,
    stpt.v.active,
    fdbk.c.active,
    fdbk.v.active,
    hyst
  );
}
```

Kód 28: Implementace `EEPROMManager` v `SwitchingRegulator`. Funkce ukládá limity, aktivní typ vstupu a hystereze.

Funkce `loadData()` obsahuje stejné argumenty ve stejném pořadí ke čtení dat.

9 Ostatní Software ochrany

9.1 Watchdog timer (WDT)

Už několikrát zmíněný watchdog timer má na starosti reset mikroprocesoru při zástavě chodu programu [1]. K resetu dojde, pokud přeteče čítač watchdogu. Pro chod programu musíme tedy periodicky nulovat čítač s funkcí `wdt_reset()`.

Určená doba přetečení je nastavená v `setup()` s funkcí `wdt_enable(WDTO_60MS)`. 60 milisekund je poměrně dost času na jednu smyčku programu. Důvodem rezervy je poměrně časově náročný zápis do paměti EEPROM. Přepis jednoho parametru trvá přibližně 3,3 ms.

9.2 Brown-out detekce (BOD)

Termín Brownout popisuje částečný a dočasný pokles napětí. Při poklesu napájecího napětí MCU se snižuje stabilita exekuce instrukcí, data mohou degradovat a mikroprocesor tak mít nepředvídatelné chování [9]. K prevenci tohoto stavu je ATmega328P vybaven obvody brownout detekce. Aktivita této detekce se nastavuje konfiguračními bity. Pro regulátor byla zvolena detekce s typickou hladinou napětí V_{BOT} 4.3V.

Table 29-17. BODLEVEL Fuse Coding(1)(2)

BODLEVEL 2:0 Fuses	Min. V_{BOT}	Typ V_{BOT}	Max V_{BOT}	Units
111	BOD Disabled			
110	1.7	1.8	2.0	V
101	2.5	2.7	2.9	
100	4.1	4.3	4.5	

Dokumentace 2: Tabulka bitové volby BOD levelu ATmega328P [1]

Při detekci nízkého napětí se mikroprocesor resetuje do doby, než je napětí opět optimální, to jest vyhovuje podmínce.

```
upload_flags =
  -e ;erase before upload
  -b 1000000 ;upload speed
  -Ulfuse:w:0xFF:m  ;External oscilator
  -Uhfuse:w:0xD7:m  ;Preserve EEPROM when erasing, boot 256 bytes
  -Uefuse:w:0xFC:m  ;Brown out 4.3v
```

Kód 29: Nastavení pojistek s platformio.ini [3] souborem při nahrání programu ISP

10 Nápady pro zlepšení

S novými zkušenostmi v programování C++ bych určitě upravil vlastní knihovny regulátoru.

Knihovna `CmdManager.h` by mohla nabízet lepší funkce. Cílem by bylo jednodušší a jasnější programování cest pro příkazy.

Řetězy parametrů metod knihovny `EEPROMManager.h` by se mohly tvořit na jednom místě, na rozdíl od specifikace dvou obdobných řetězců na psaní a čtení. Dle toho řetězce pak ukládat a načítat data bez argumentů ve funkci.

11 Závěr

Postupným vývojem se splnily všechny zákazníkovy nároky. Úkolem firmy TCELE bylo kreslení desky plošného spoje, při kterém jsem částečně figuroval, a dodání firmware k oživení regulátoru. Ten jsem realizoval v jazyku C++ s platformou Arduino.

V souhrnu, výstupem regulátoru je dvojice relé, dodávající síťové napětí v sepnutém stavu. Korespondující relé se spíná na základě stavů vybraných vstupů regulátoru. Na výběr jsou dva typy vstupu, proudová smyčka anebo napěťová hladina. Je-li zpětnovazební veličina příliš pod veličinou řídící - sepni jedno relé, je-li příliš nad - sepni druhé. V přítomnosti zpětnovazební veličiny v pásmu hystereze s řídící, rozepni obě relé. Výběr typu vstupů, velikosti pásma hystereze a limity se konfigurují sériovým portem. K budoucímu užití jsou parametry uloženy do paměti EEPROM. Deska obsahuje LED k vizualizaci stavu výstupu či chyb vstupů. Obsahuje také dva mikrospínače k manuálnímu ovládání výstupu, dvojice relé.

Práce na vývoji přepínacího regulátoru byla velmi přínosná. Cestou jsem se potýkal se spoustou problémů jak v programování, tak při nákresu desky plošného spoje. U regulátoru jsem poprvé užil objektového programování v C++, které velice napomohlo čitelnosti kódu pro nynější úpravy a zdokonalení. Příkladem je přidání limit na kalibraci PWM výstupní proudové smyčky.

Ve finálním úhlu pohledu se přepínací regulátor chová spolehlivě, nabízí jednoduchou, často jednorázovou, konfiguraci s PC a na desce vizualizuje stav chodu či chyby. S výsledným produktem jsem spokojen, ale co je důležitější, že spokojen je hlavně zákazník.

12 Seznam zdrojů

Znalosti a metodika programování C++ s:

MATOUŠEK, David. *C++ bez předchozích znalostí*. Brno: Computer press, 2016. ISBN 9788025146408

[1] Atmel Corporation. [online katalogový list]. *ATmega328P*. ©2015 [cit. 19.5.2021]. Dostupné z: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

[2] PlatformIO, 2014. What is PlatformIO? [online]. 23.7.2020. [cit. 19.5.2021] Dostupné z: https://docs.platformio.org/en/latest/what-is-platformio.html

[3] PlatformIO, 2014. ATmega328P/PA [online]. 11.12.2020. [cit. 19.5.2021] Dostupné z: https://docs.platformio.org/en/latest/what-is-platformio.html

[4] Arduino, 2021. Language Reference. In: *arduino.cc* [online]. [cit. 19.5.2021] Dostupné z: https://www.arduino.cc/reference/en/

[5] PAONESSA, Simon a MCDUFFEE, Bruce, Precision Digital Corporation. Back to Basics: The Fundamentals of 4-20 mA Current Loops. In: *predig.com* [online]. [cit. 19.5.2021] Dostupné z: https://www.predig.com/indicatorpage/back-basics-fundamentals-4-20-ma-current-loops

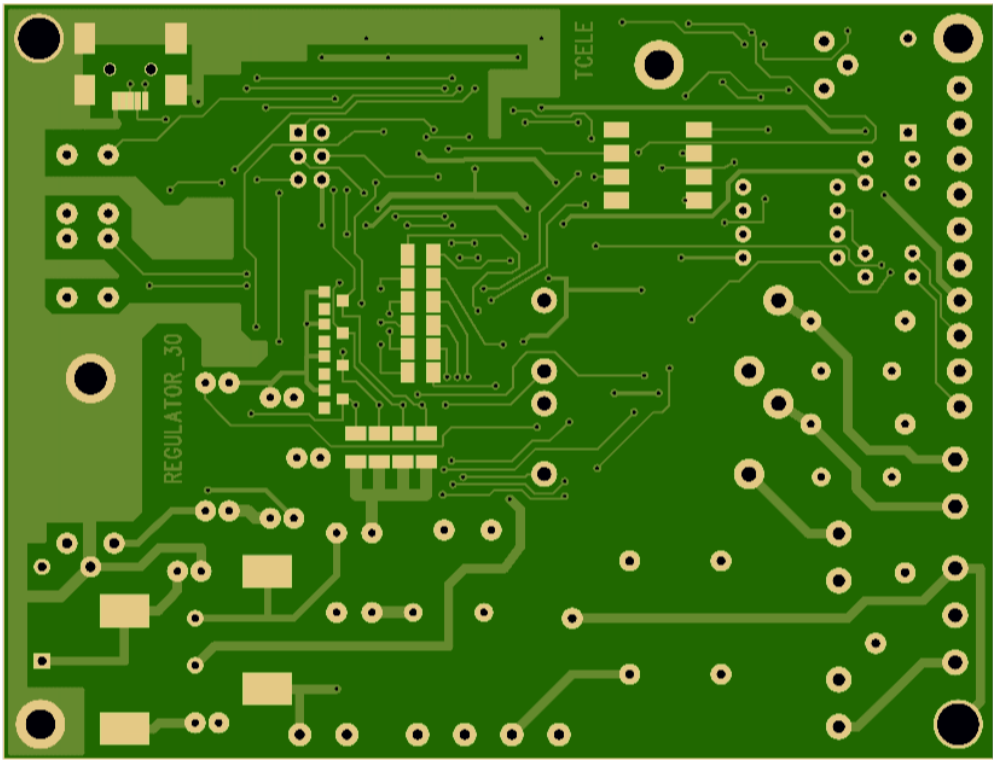
[6] jrt, 2018. Micro:bit a PWM. In: *robodoupe.cz* [online]. [cit. 19.5.2021] Dostupné z: http://robodoupe.cz/2018/microbit-a-pwm/

[7] ARIAS, Evert, 2020. EasyButton. In: *github.com* [online]. [cit. 19.5.2021] Dostupné z: https://github.com/evert-arias/EasyButton

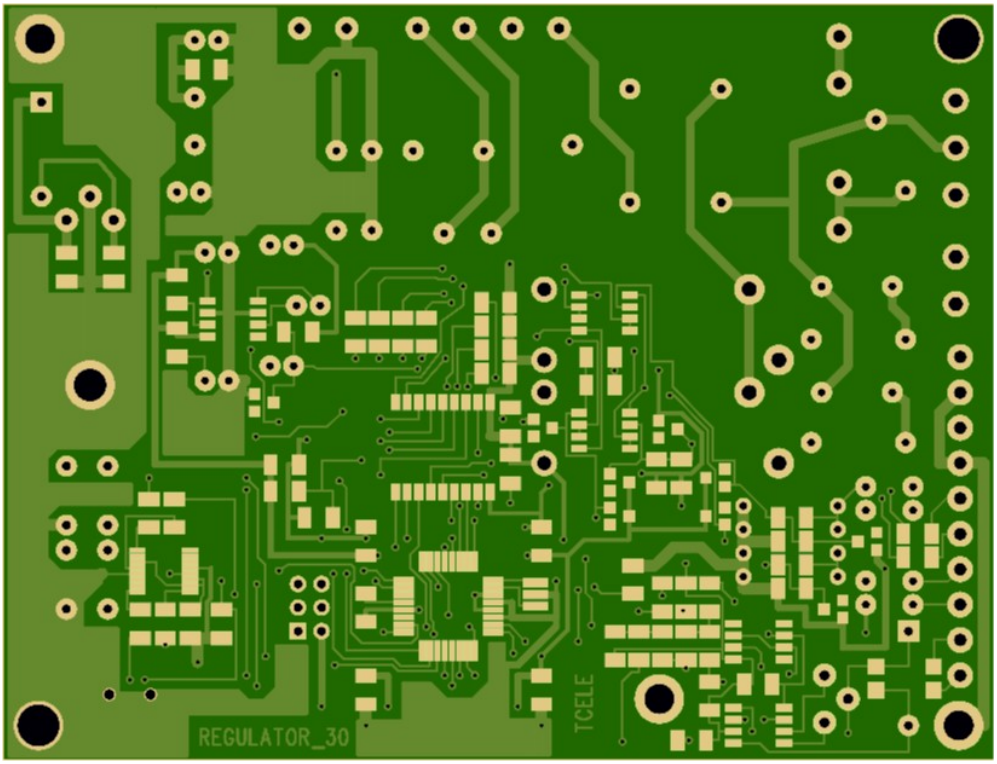
[8] CLARKE, Damien, 2019. ResponsiveAnalogRead. In: *github.com* [online]. [cit. 19.5.2021] Dostupné z: https://github.com/dxinteractive/ResponsiveAnalogRead

[9] COLLEY, Stephen, 2019. What Is Brown Out Reset in Microcontrollers? How to Prevent False Power-Downs. In: *allaboutcircuits.com* [online]. [cit. 19.5.2021] Dostupné z: https://www.allaboutcircuits.com/technical-articles/what-is-brown-out-reset-microcontroller-prevent-false-power-down/#:~:text=Brown%20Out%20Reset%20is%20an,Reset%20can%20prevent%20another%20problem

13 Přílohy



Ilustrace 4: Vrchní strana DPS



Ilustrace 5: Spodní strana DPS

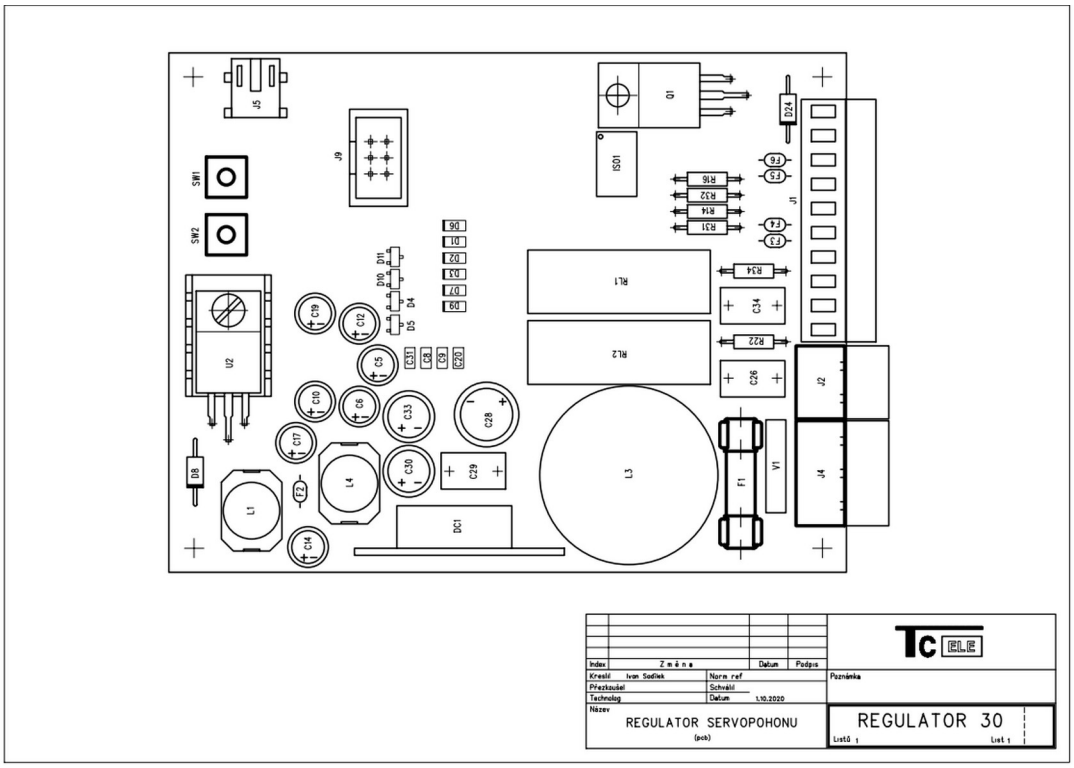


Schéma 12: Vrchní strana nákresu DPS

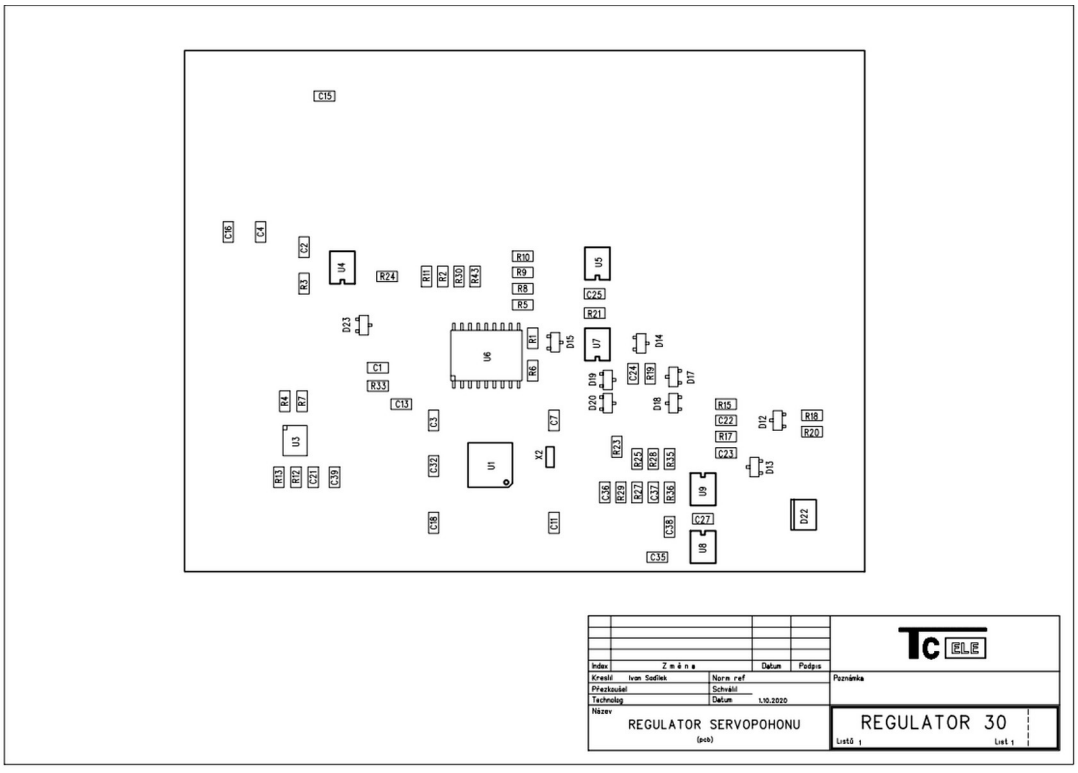


Schéma 13: Spodní strana nákresu DPS

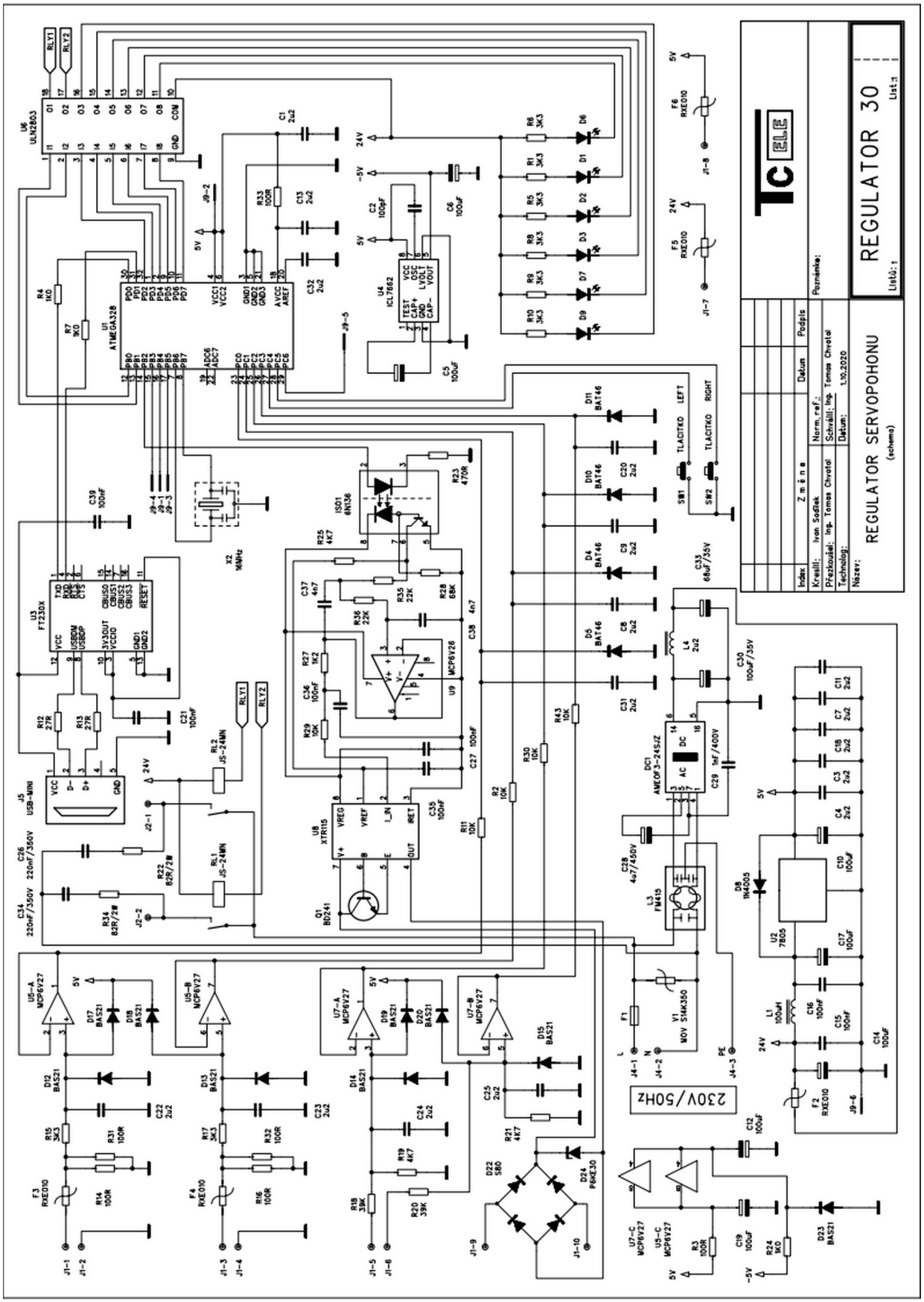
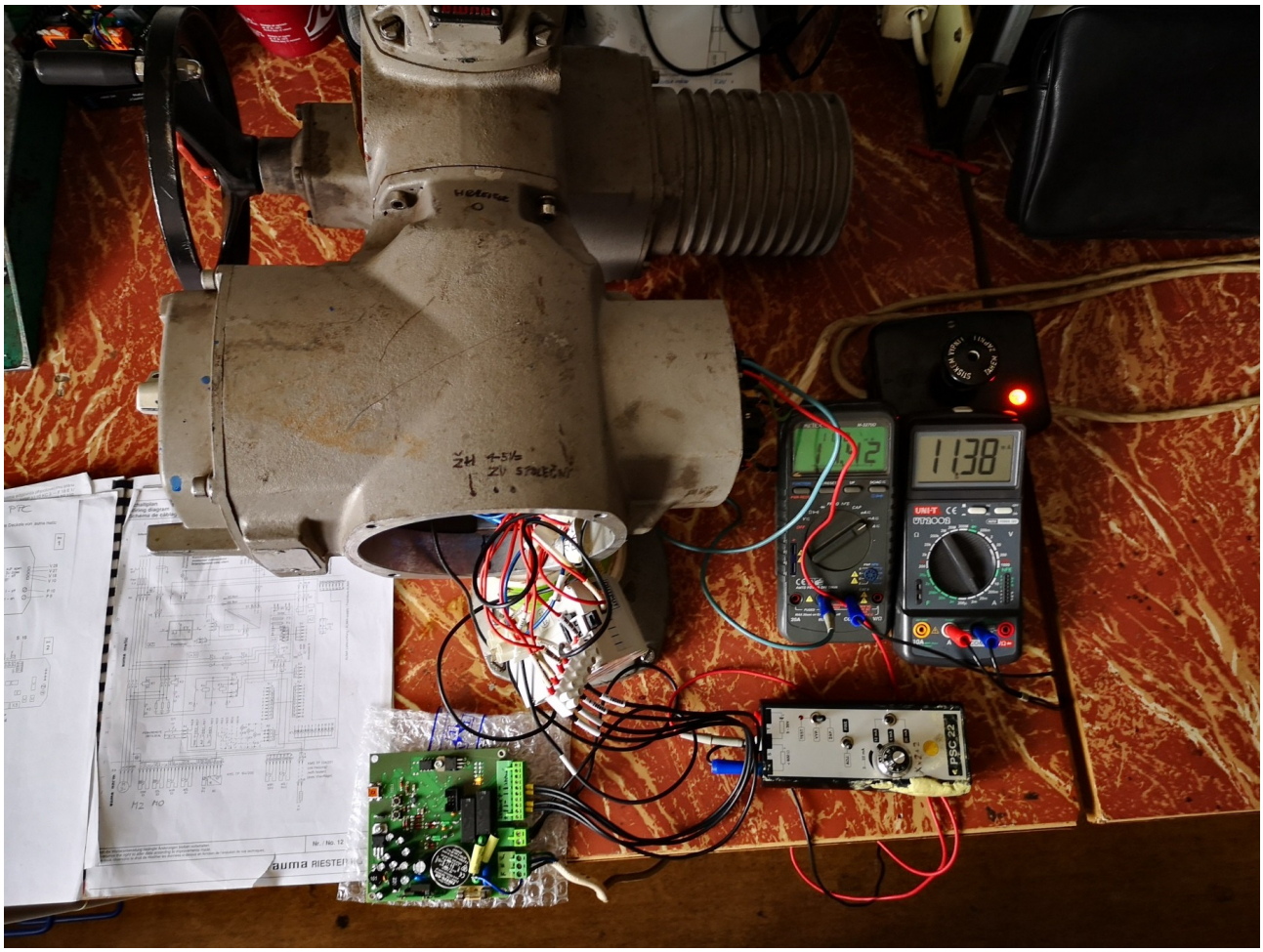


Schéma 14: Elektrické schéma přepínacího regulátoru



Ilustrace 6: Zákazníkova aplikace prototypové desky přepínacího regulátoru jako ovládání motoru ventilu.