ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ
KATEDRA APLIKOVANÉ ELEKTRONIKY A TELEKOMUNIKACÍ

Bakalářská práce

Odpalovací zařízení pro rakety na tuhé palivo

2014 **Martin Wolmut**

Anotace

Výsledkem práce je pult s LCD displejem řízený mikroprocesorem AT89C51CC03. Tímto pultem se ovládá rampa, ze které se odpalují modely raket na tuhá paliva. Rampa umožňuje nastavení směru dráhy letu modelu. Odpálení rakety je realizováno pyrotechnickým palníkem s elektronickým spínačem. Dále je k pultu připojen obvod pro měření rychlosti větru, kvůli bezpečnosti odpalování raket. Na LCD displeji můžeme volit různé režimy odpalu raket a také je na něm zobrazena aktuální hodnota rychlosti větru a informace o napájení v obvodu. Na pultu je pomocí LED diody signalizován stav palníku, zda je nebo není připojen.

Klíčová slova

Odpalovací zařízení, raketové motorky, řízení náklonu odpalovací rampy, mikrokontrolér, programovací jazyk C, CPLD

Annotation

The result of my project is a control panel with LCD, controlled by AT89C51CC03 microprocessor. The main purpose of this panel is to control a launch pad for rocket models powered by solid fuels. The launch pad allows the user to set the direction of the flight. Ignition of the rocket is secured by pyrotechnical detonator with an electronic switch. The panel utilizes a wind measuring circuit to ensure safety during the launch. The LCD allows choosing different modes of launching the rocket. The display also shows information about actual wind speed and current circuit voltage. LED on the panel indicates whether the detonator is connected to the panel.

Key words

Launcher placement, model rocket motor, inclination launcher placement control, microcontroller, C programming language, CPLD

Seznam zkratek

CPLD	Complex programmable logic device
EEPROM	Electrically erasable programmable read-only memory
JTAG	Joint test action group
LCD	Liquid-crystal display
LED	Light emitting diode
MKO	Monostabilní klopný obvod
PAL	Programmable array logic
PC	Personal computer
RMK	Raketový modelářský klub
SMT	Stanice mladých techniků
SO	Středně odolný
TPH	Tuhé pohonné hmoty

Prohlášení

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

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 ……. dne…….. podpis:……

Poděkování

Děkuji Ing. Petru Wolmutovi za poskytnutí odborných rad v oblasti softwarové a elektronické části práce, dále bych chtěl poděkovat Lumíru Honzíkovi za rady v mechanické oblasti práce. Další člověk, který si zaslouží poděkování je pan Doc. Dr. Ing. Vjačeslav Georgiev, který mi dával cenné rady při psaní dokumentace k mé práci.

Obsah:

1. Úvod

Práce se zabývá problematikou odpalování modelů raket na tuhá paliva. S tím souvisí výroba modelů raket. Způsoby odpalování raket a s tím spojená automatizace odpalování. Cílem této práce je vytvořit a ozkoušet odpalovací rampu na rakety, řízenou odpalovacím pultem.

1.1. Raketové modelářství

Začátky raketového modelářství v České republice se rozvíjí kolem roku 1962. V tomto roce byl založen RMK (raketový modelářský klub) v Praze. V Plzni byl RMK založen roku 1971. Raketové modelářství je úzká oblast modelářství, při kterém se vyrábějí raketové modely. Vyrábějí se vymyšlené rakety nebo tzv. makety. Makety se mají s co největší přesností podobat opravdovým raketám, ale v menší velikosti.

1.2. Výroba raket

Raketové modely mohou být buď vymyšlené, nebo to mohou být makety raket. Rakety, které se používají na závody, musejí splňovat určitá kritéria pro závodění. Například hmotnost modelu nesmí přesáhnout 1500g. Všechna kritéria a pravidla se dají najít na stránkách Svazu modelářů České republiky.

Rakety by měli být lehké, proto se vyrábějí z papírové lepenky, která je namotána v několika vrstvách na železnou trubku. Po sundání se tento trup rakety dále upraví. K modelu se vyrobí z balzy stabilizátory a hlavice. Pro hladké přistání rakety a také kvůli bezpečnosti, musí být každá raketa vybavena brzdným zařízením (padák). Zhotovený model by měl být schopen odstartovat více než jednou.

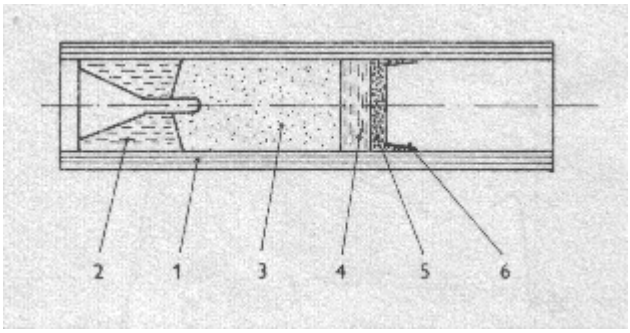
1.3. Závody

Svaz modelářů České republiky pořádá pro modeláře z celé ČR, kteří se zabývají výrobou raket, závody. Na tyto závody je dobré se připravit, a proto je užitečné vlastnit odpalovací pult s rampou. V Plzni je ve SMT – Stanice mladých techniků provozován kroužek pro raketové modeláře, kterého jsem byl členem. Tento kroužek má na starosti Lumír Honzík, který je zároveň ředitelem Hvězdárny a planetária v Plzni. Projekt, který jsem začal vytvářet tj. odpalovací pult s rampou na odpalování modelů raket, je určen pro Hvězdárnu a

planetárium v Plzni. Tato organizace mi také celý projekt financovala, po dokončení práce jí bude hotový výrobek věnován.

1.4. Raketové motory

Základem každé rakety je motor. Pro modely se používaly motory označované RM, dnes se používají motory s označením Rapier.

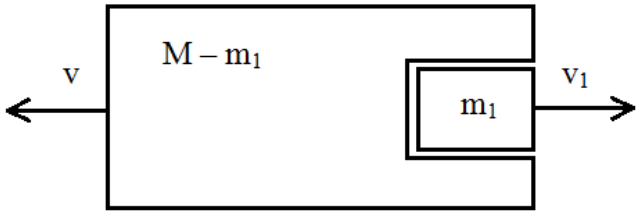


Obrázek 1.1: Řez motorkem s označením RM, 1 - papírová trubka, 2 - lisovaná keramická tryska, 3 - TPH s kuželovou zážehovou dutinou, 4 - zpožďovací složˇ, 5 - výmetná náplň, 6 - krytka

Obrázek 1.2 : Ukázka raketových motorů RM

Pro pochopení problematiky vzletu rakety si uvedeme základní fyzikální vztahy a ukážeme na obrázku motoru. Potřebuje znát Newtonovy zákony a z nich vyplývající zákon zachování hybnosti. Je-li z tělesa o celkové hmotnosti M odebrána rychlostí v_1 jeho část o hmotnosti m_1, těleso o nynější hmotnosti $M - m_1$ se dá do pohybu rychlostí v.

$$(M - m_1) * v = m_1 * v_1 \tag{1.1}$$



Obrázek 1.3 : Princip pohybu rakety s raketovým motorkem

Vlivem hoření raketového motoru se z něj uvolňují pevné i plynné částečky. Tím vzniká reaktivní síla, která způsobí to, že se raketa začne pohybovat rovnoměrně se zvyšující se rychlostí.

Tah raketového motoru si označíme jako F. Ten je závislý na hmotnostním průtokovém množství spalin m a na efektivní rychlosti výtrysku spalin z motorkové trysky W_{ef}.

$$F = m * W_{ef} \quad [\text{ N }] \tag{1.2}$$

Pomocí celkového impulzu I_c vyjádříme výkon raketového motoru. Ze vzorce 1.3 je vidět, že I_c je plocha pod křivkou F v závislosti na čase hoření motoru.

$$I_c = \int_0^t F * dt \tag{1.3}$$

1.5. Měření energie potřebné k zapálení palníku

Při měření energie palníku, byl použit pyrotechnický palník. Během měření bylo zjištěno, že při inicializaci palníku se nerozpojí obvod. Jelikož není obvod rozpojen, nelze na osciloskopu odečíst hodnotu energie (tato energie by byla plocha pod křivkou proudu).

Dále bylo při měření pozorováno, že pyrotechnický palník byl zažehnut při napětí na vstupu 12 V a procházejícím proudu kolem 1 A. Bezpečný proud byl 0,9 A. Výrobce uvádí u tohoto typu palníku inicializační proud 2,12 A a bezpečný proud 0,45 A.

1.6. Odpalování raketových motorů

Pro ruční odpalování modelů se používá zápalná šňůra, nebo stopina. Dnes se spíše používá odpalování pomocí elektrických palníků. Tyto palníky se odpalují pomocí mžikového iniciačního proudu. Těchto palníků existuje více druhů.

Palník – je přípravek, který se používá k zažehnutí různých pyrotechnických pomůcek. Kromě jiného se také nechá použít k odpalování modelů raket.

1.6.1 Pyrotechnický palník

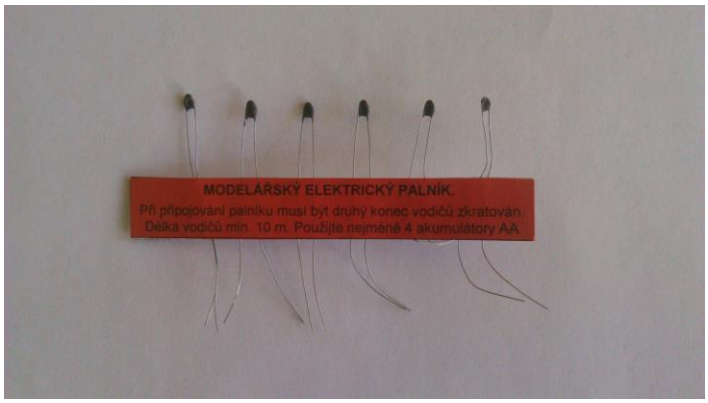
Není primárně určen k odpalování modelů raket, ale k použití při odpalování ohňostrojů a zábavné pyrotechniky. Vzhledem k jeho dobrým vlastnostem se dá použít při odpalování modelů raket.

Parametry elektrického palníku SO (středně odolný):
iniciační proud: 2,12 A – 4ms impuls
bezpečný proud: 0,45 A
odpor: 0,48 ~ 0,60 Ω
délka přívodních vodičů: 20 cm

Obrázek 1.4 : Elektrický palník typ SO

1.6.2 Modelářský elektrický palník

Tento typ palníku je dodáván výrobcem k raketovým motorkům. Palník je vyroben z odporového drátu a uprostřed je zápalná směs. Bohužel výrobce neuvádí bližší specifikaci palníku. Pouze uvádí, že k inicializaci je potřeba nejméně 4 akumulátorů typu AA. To by mělo být napětí kolem 6 V.



Obrázek 1.5: Modelářský elektrický palník

2. Požadavky na odpalovací pult

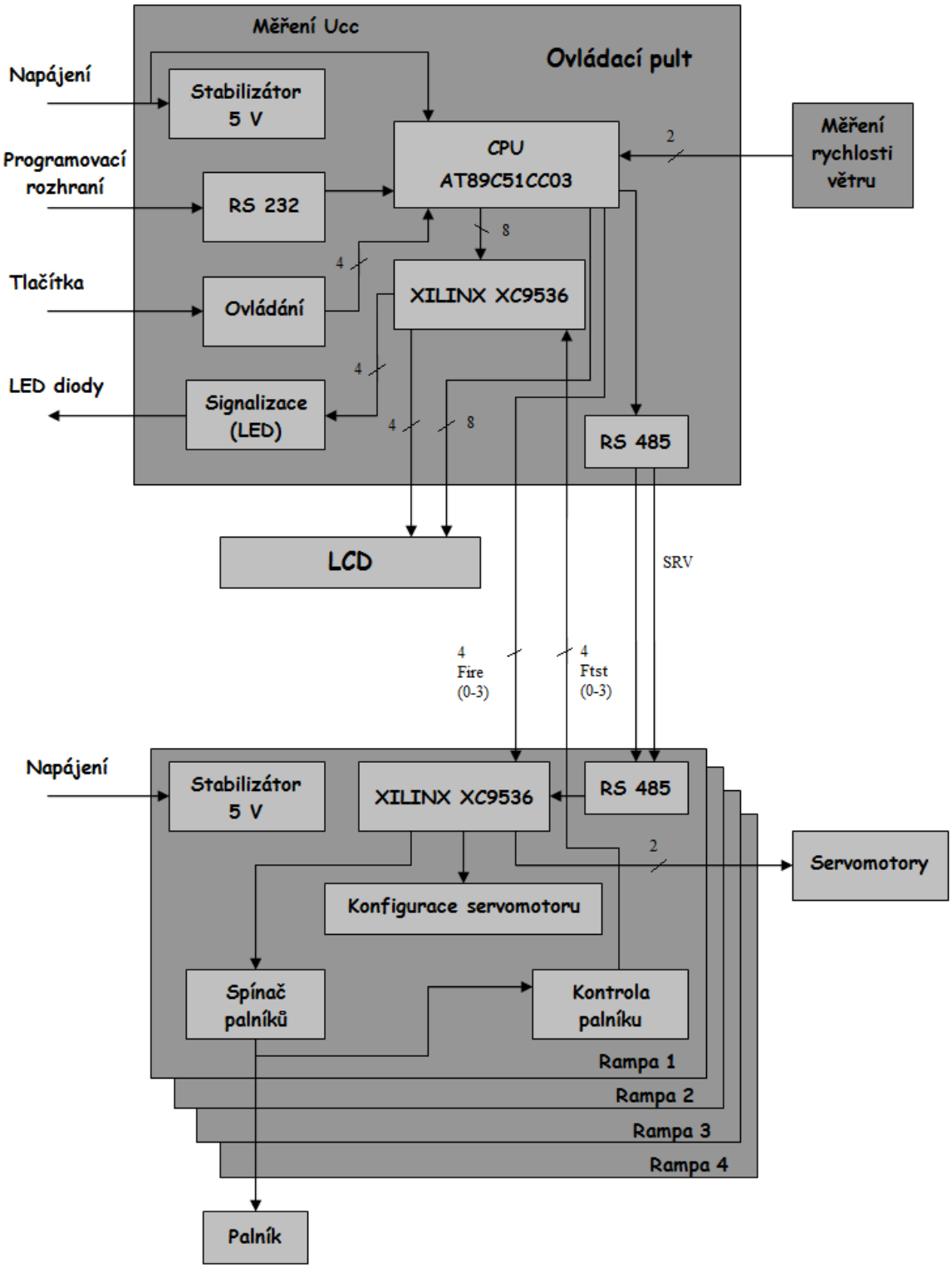
Ovládací pult by měl mít být ovládán mikrokontrolérem, který se bude starat o funkčnost celého ovládacího pultu a odpalovací rampy. Na pultu by mělo být připraveno několik konektorů na odpalovací rampy. Dále by zde měl být konektor na připojení měřiče rychlosti větru. Měření rychlosti větru musí být u pultu z důvodu bezpečnosti. Při příliš silném větru se rakety nesmění odpalovat. Na pultu musí být samozřejmě tlačítko, kterým odpálíme raketu, stop – tlačítko, kterým můžeme v případě nutnosti zastavit start rakety a uzamykání pultu. Také by měl být na pultu displej, který bude uživatele informovat o nastavení jak pultu, tak odpalovací rampy.

2.1. Návrh odpalovacího pultu na rakety

Celý projekt byl rozdělen na tři základní části:

a) Ovládací pult
b) Odpalovací rampa
c) Měření rychlosti větru

3. Blokové schéma



Obrázek 3.1: Blokové schéma ovládacího pultu

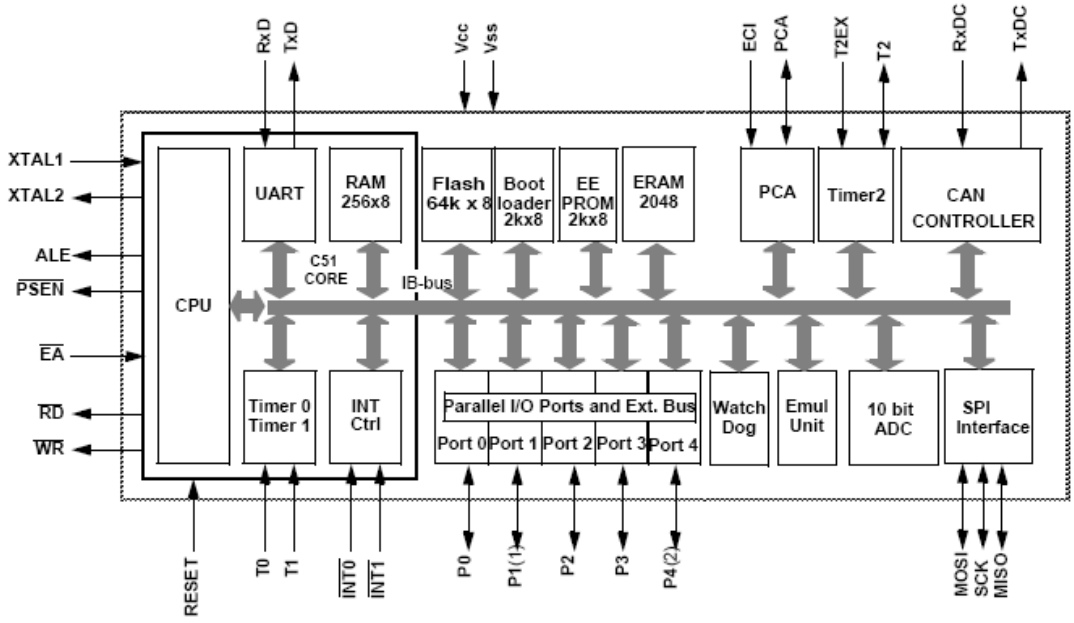
4. Ovládací pult

Celý projekt včetně ovládacího pultu je ovládán mikrokontrolérem. Pomocí mikrokontroléru je na grafickém LCD displeji zobrazována aktuální rychlost větru. Ta je měřena pomocí optického snímače (viz měření rychlosti větru). Dále se na displeji zobrazuje nastavení zpoždění odpalu raket, směr natočení odpalovací rampy a náklon odpalovací rampy.

4.1.1 Funkce mikrokontroléru v panelu odpalovací rampy

- Indikace připojení palníku
- Zobrazení aktuálního natočení vodící tyče na odpalovací rampě na LCD displeji
- Zobrazení rychlosti větru a aktuálního napájecího napětí na LCD displeji
- Ovládání a komunikace s obvody XILINX
- Ovládání zažehnutí palníku
- Nastavení času zpoždění (0 – 10 sec) k zažehnutí palníku.

Na základové desce patřící k ovládacímu pultu najdeme jako hlavní součást desky mikrokontrolér od firmy Atmel s označením AT89C51CC03. Tento mokrokontrolér slouží jako řídící jednotka celého projektu, použitý typ spadá do skupiny 8 – bitových mikroprocesorů, z řady 80C51. V režimu X2 je díky maximální rychlosti hodin mikroprocesoru 30 MHz dosaženo 300 ns cyklů. AT89C51CC03 má 64 Kb Flash paměť včetně In-System Programming (programování mikroprocesoru bez nutnosti vyjmutí z desky), dále je použita 2 KB bootovací flash paměť EEPROM. Tato paměť je permanentní a dá se přeprogramovat pouze část informací v paměti.



Obrázek 4.1 : Blokové schéma mikroprocesoru AT89C51CC03

K programování pevných pamětí typu flash je použito sériové rozhraní RS 232, přes které se tyto paměti programují z PC.

Programové vybavení odpalovací rampy uložené v pevné paměti mikrokontroléru AT89C51CC03 bylo sestaveno v jazyce C. Tento jazyk je nejpoužívanějším jazykem pro tzv. embedded aplikace.

4.2. Komunikační rozhraní

RS 232 – sériové rozhraní

U tohoto rozhraní mohou mezi sebou komunikovat pouze dvě zařízení, na vzdálenost max. 15 m při přenosové rychlosti 20 Kb/s. V praxi se dá dosáhnout lepších výsledků. Můžeme zvyšovat vzdálenost, ale bude nám klesat přenosová rychlost.

Pro propojení stačí 3 vodiče:

- Rx – příjem
- Tx – vysílání
- GND – zem

RS 232 používá dvě napěťové úrovně log "0", "1"

log "1" je indikována záporným napětím

log "0" je indikována kladným napětím

RS 485 – sériové rozhraní

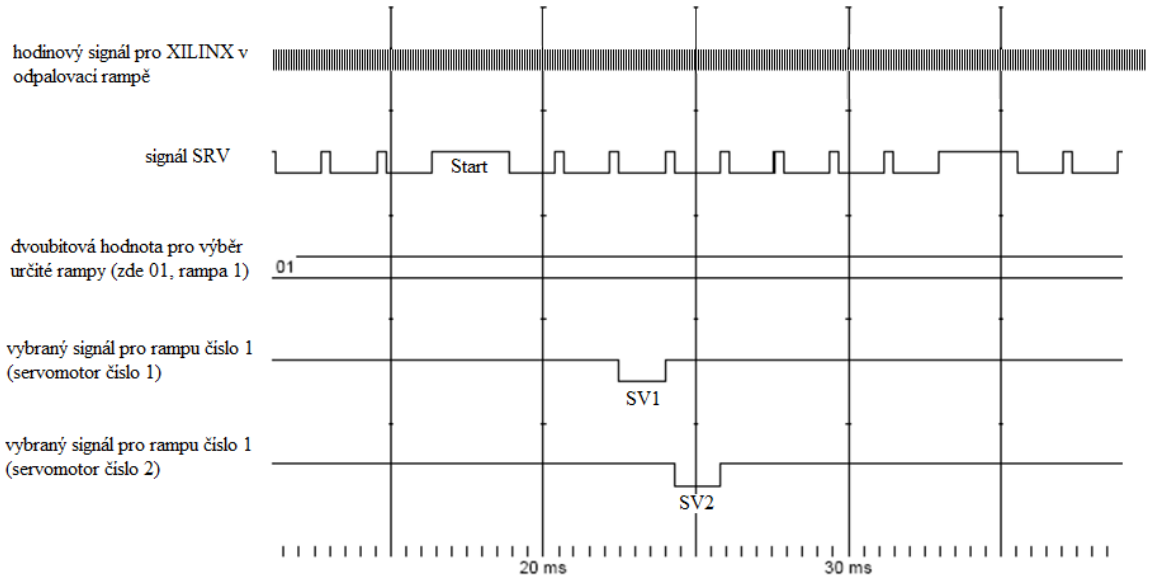
Používá se jeden pár vodičů pro oba směry toku dat, je tedy třeba směr toku dat přepínat. Vzdálenost, na kterou RS 485 přenáší data je 1200 m při maximální přenosové rychlosti 10 Mb/s (samozřejmě záleží na délce vedení). Na rozdíl od RS 232 se zde nevyhodnocuje napětí mezi vodičem a zemí, ale rozdílové napětí mezi vodiči, díky tomuto vyhodnocování napětí lze eliminovat indukované rušivé napětí vztažené k nulovému potenciálu země.

4.3. Popis komunikace mezi ovládacím pultem a odpalovací rampou

Ovládací pult komunikuje s odpalovacím pultem pouze jednosměrně, pomocí rozhraní RS485. Komunikuje se tím způsobem, že se vysílá série pulzů. Komunikace je vidět na obrázku 4.2.

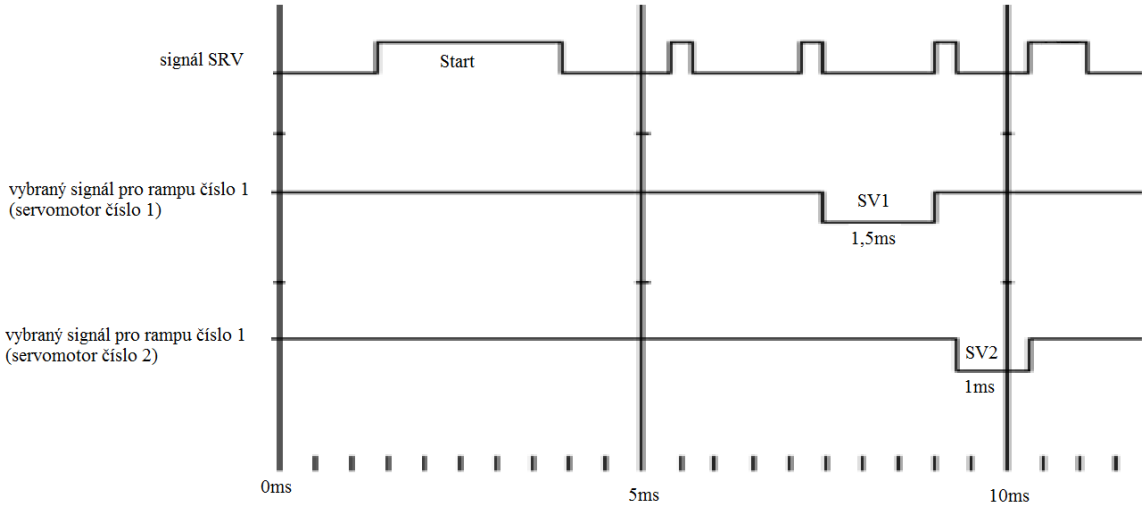
Signál SRV je společný pro všechny čtyři rampy. Jako první je vyslán pulz s označením Start, podle tohoto signálu pozná XILINX v odpalovací rampě, že začíná sekvenci pulzů. V každé odpalovací rampě je přepínač, jehož nastavení (logická hodnota 0-3) nám určuje číslo rampy. Tato dvoubitová hodnota je přijímána obvodem XILINX v odpalovací rampě. Podle hodnoty z přepínače jsou poté vybírány pulzy pro danou rampu. Pro nultou rampu bude vybrán pulz číslo 0 a 1. Pro rampu číslo 1 budou vynechány první dva pulzy a budou vybrány pulzy 2 a 3. Vždy první signál z těchto dvou je pro první servomotor a druhý pro druhý servomotor. Na obrázku 4.2 vidíme vybranou rampu číslo 1. Pulzy označené jako SV1 a SV2 jsou pulzy pro rampu číslo jedna a servomotor číslo jedna a dva.

Obvod typu XILINX, který je umístěn v odpalovací rampě, pouze vybírá signály pro danou rampu a dané servomotory a posílá je do příslušných servomotorů. Otočení servomotorů je určeno délkou pulzu SV1 a SV2. Tato doba by mohla být od 0,9ms do 2,1ms, ale zde není potřeba se servomotorem hýbat až do jeho druhé krajní polohy (2,1ms), proto je maximální délka pulzu SV1 a SV2 menší než 2,1ms. Vzhledem k přepočtu a natočení servomotorů v odpalovací rampě je nulová poloha servomotoru pro 1,5ms. Tyto pulzy jsou dále vyvedeny z XILINXu v odpalovací rampě přes tranzistory (otočí pulz z 0 na 1) do samotných servomotorů.



Obrázek 4.2 : Příklad signálů pro zvolenou rampu číslo 1

Komunikace je připravena k tomu, aby se mohlo s každou odpalovací rampou hýbat zvlášť. Momentálně je uživatelské rozhraní na ovládacím pultu uděláno tak, že nastavuje pouze jednu hodnotu natočení a to pro rampu 0. Pokud bychom měli připojeno více odpalovacích ramp, mohli bychom natáčet pouze rampu 0 a ostatní by byly pořád ve stejné poloze. Nebo bychom mohli na všech rampách nastavit přepínač na hodnotu 00, poté by se všechny připojené rampy pohybovali stejně.



Obrázek 4.3: Příklad signálů pro různé natočení servomotoru 1 a 2

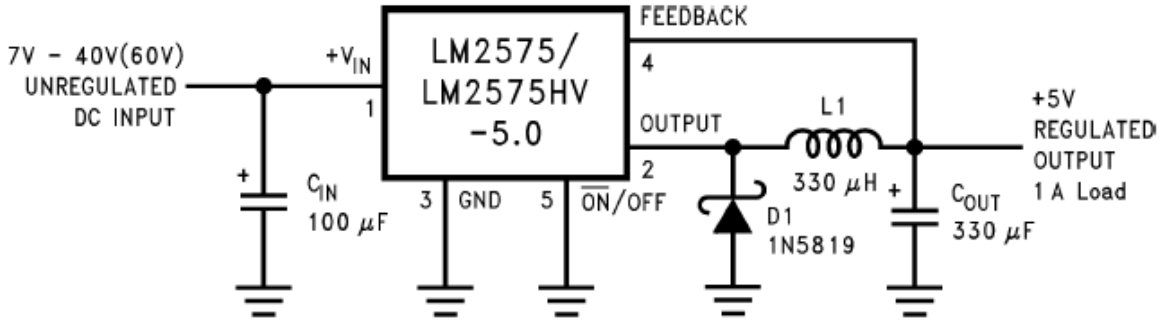
Na oblázku 4.3 vidíme společný signál SRV a dva vybrané signály pro rampu číslo 1. Délka pulzu s označení SV1 je 1,5ms, to znamená, že servomotor je nastaven v nulové

poloze. Délka pulzu s označením SV2 je 1ms, tento servomotor bude natočen o 50°. Hodnota 50° je pouze teoretická, protože v rampě by takovéto natočení nebylo možné. Pro natočení servomotoru o 1°je změna délky ovládacího pulzu 10us.

4.4. Regulátor napětí

Integrovaný spínaný regulátor napětí LM2575T – 05

Tento integrovaný spínaný regulátor nastavuje vstupní napětí (zde použité 12 V stejnosměrných) na požadované napětí 5 V stejnosměrných s výstupem 1 A. Je zde použit tento regulátor kvůli odběru proudu LCD displeje (podsvícení), který je značný oproti ostatním součástkám, proto by se obyčejný integrovaný lineární stabilizátor hodně zahříval a musel by u něj být umístěn veliký chladič.



Obrázek 4.4 : Katalogové zapojení spínaného regulátoru LM2575T – 05, které je použito na plošném spoji ovládacího pultu

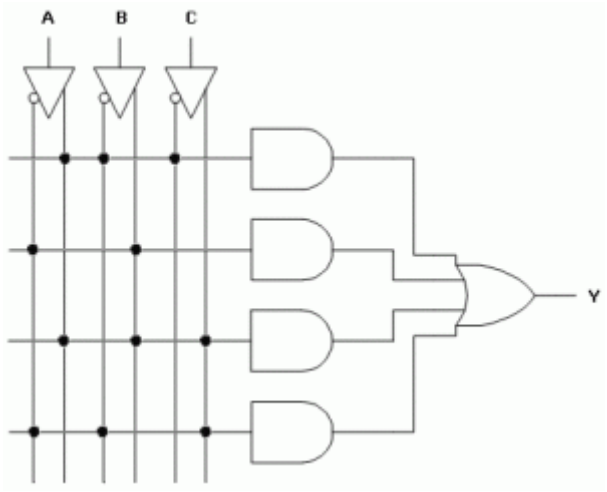
4.5. Obvody typu CPLD a programovatelný logický obvod XILINX

CPLD (complex programmable logic device) – komplexní programovatelné hradlové pole.

Současné obvody CPLD již mohou nahradit několik tisíc nebo i několik set tisíc logických hradel. CPLD se většinou programují pomocí vyhrazeného rozhraní (např. JTAG – použito i u XILINX XC9536) až poté co jsou zapájeny do desky plošných spojů.

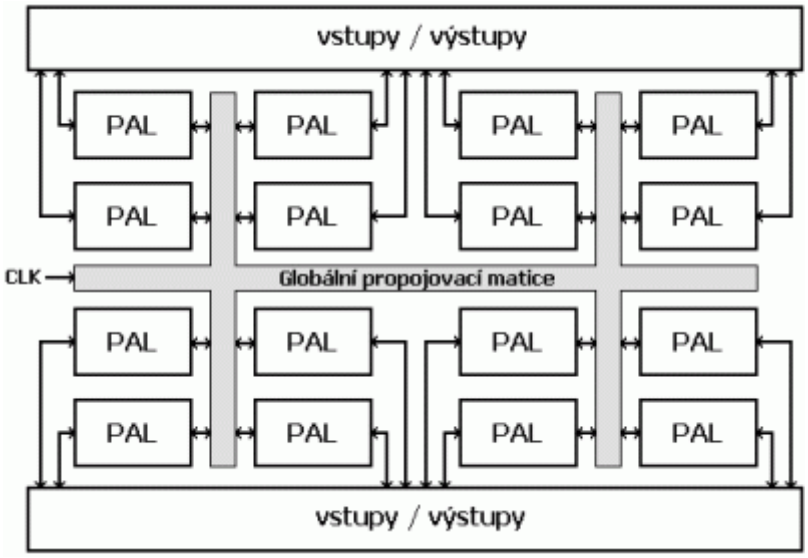
Popis obvodu CPLD

Začněme u makrobuňky. Ta tvoří vždy jeden paměťový člen doplněný o řadu pomocných hradel. Podoba makrobuňky se v každém obvodu liší.



Obrázek 4.5 : Jeden z nejjednodušších obvodů – PAL

Obvody PAL, mají programovatelné pole AND a pevně realizované pole OR, tvořící vždy součet několika sousedních termů přivedený do výstupního obvodu.
Tyto obvody se staly základem obvodům CPLD. Jednotlivé obvody typu PAL jsou propojeny pomocí globální propojovací matice. Z těchto obvodů typu PAL vychází výstupy, nebo vstupy, podle naprogramování.



Obrázek 4.6 : Typická struktura obvodu CPLD

Sériové programovací rozhraní JTAG (Joint Test Action Group)

Je standard definovaný normou IEEE 1149.1. Jedná se o architekturu Boundary-Scan pro testování plošných spojů, programování FLASH pamětí apod. Přestože JTAG je standardizovaný, konektory na připojení JTAG adaptéru již nikoliv. Většina výrobců používá vlastní pinout, přičemž je většinou použit konektor "header" s roztečí 2,54 mm.

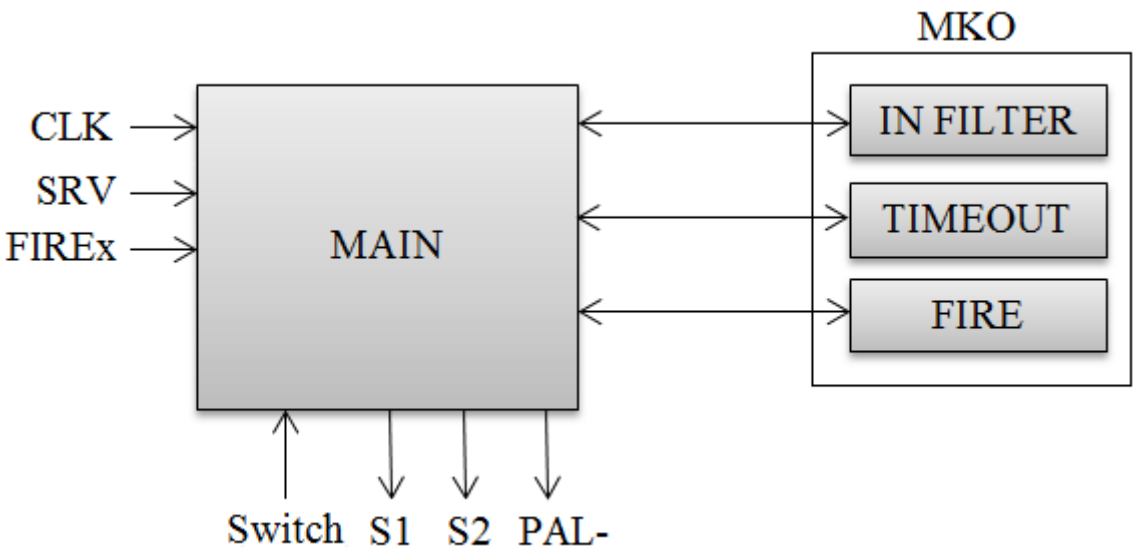
XILINX XC9536

Je programovatelný logický obvod poskytující CPLD. XILINX je napájen stejnosměrným napětím 5 V, použitý typ má 44 pinů. Uživatel si může nastavit až 34 pinů na I/O. Dá se programovat přes rozhraní JTAG. Program byl napsán v jazyku VHDL a nahrán do programovatelného logického obvodu přes rozhraní JTAG.

Jazyk VHDL

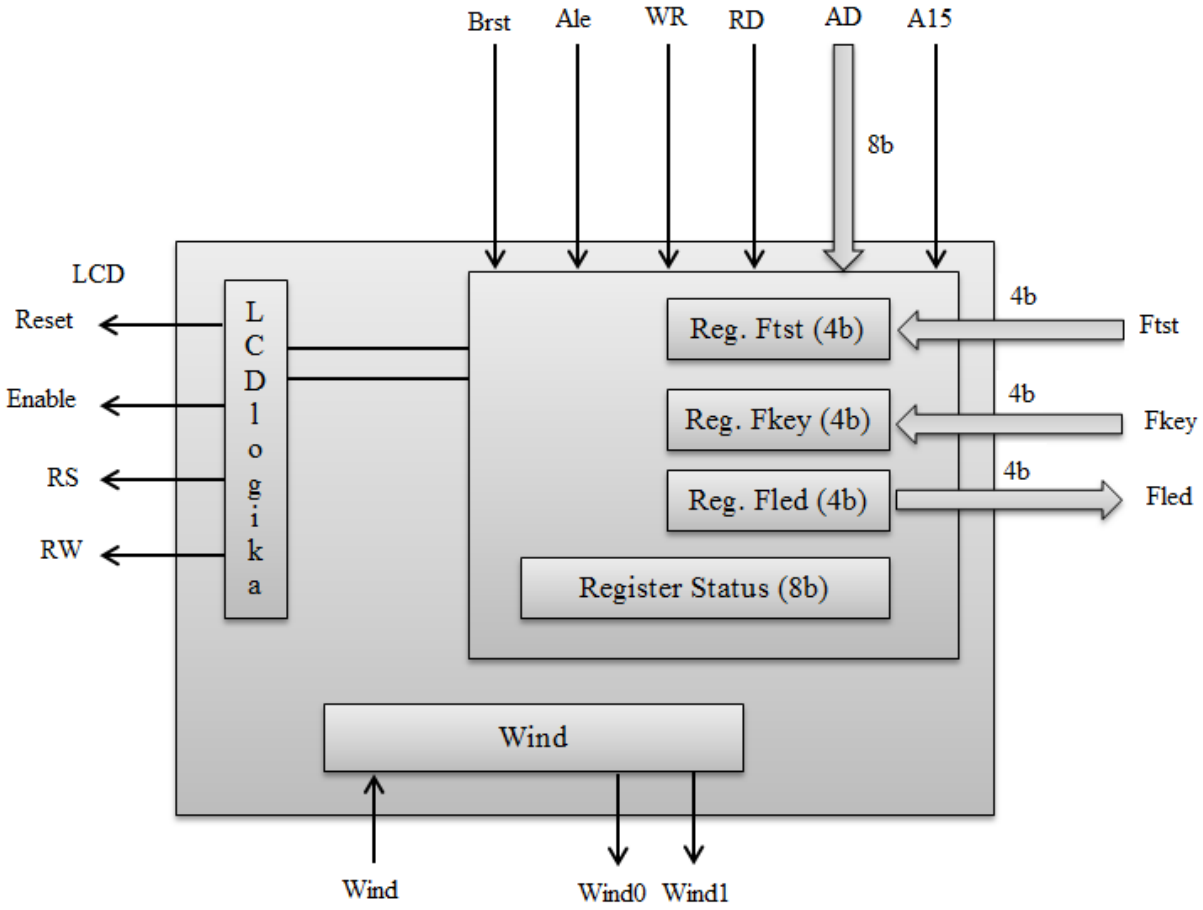
Je programovací jazyk určený k popisu (návrhu) velmi rozsáhlých číslicových systémů. Jedná se o jazyk určený k popisu hardware. Tímto jazykem můžeme navrhovat a simulovat digitální integrované obvody (programovatelné hradlové pole). Jazyk VHDL byl použit pro návrh obou programovatelných logických obvodů (XILINX).

4.6. Popis funkce programovatelných logických obvodů XILINX



Obrázek 4.7: Blokové schéma obvodu XILINX umístěném v odpalovací rampě

V obvodu XILINX v odpalovací rampě je naprogramováno řízení servomotorů a obvodu spínání palníku (zdrojový soubor RampaPld.vhd). K časování výstupních signálů a filtraci vstupního signálu Data se využívá znovu spustitelný MKO (zdrojový soubor MonoStable.vhd). MKO INFILTER s časovou konstantou 17us zajišťuje odolnost vstupního datového signálu proti rušení, MKO TIMEOUT s časovou konstantou 1,5ms slouží k vyhledávání nulujícího impulzu ve vstupních datech. Monostabilní klopný obvod FIRE je navržen k pulznímu sepnutí výkonového obvodu spínání palníku, časová konstanta je nastavena na 10ms. Vstupem Switch se nastaví číslo, podle použité rampy (0 – 3). Výstupy S1 a S2 ovládají natočení servomotorů. Výstupem PAL- je řízeno spínání palníku. Vstup SRV je vstupní signál přijímaný po sériové lince RS 485.



Obrázek 4.8: Blokové schéma obvodu XILINX umístěném v ovládacím pultu

V obvodu XILINX v ovládacím pultu je zrealizována pomocná logika ovládání LCD displeje, předzpracování signálu z měřiče rychlosti větru, zpracování signálu z tlačítek umístěných na ovládacím pultu a testovacích signálů z odpalovací rampy. Dále je zde umístěn registr pro ovládání LED diod na ovládacím pultu. Řídící a stavové registry jsou mapovány v

externím paměťovém prostoru mikrokontroléru AT89C51CC03. Spojení s mikrokontrolérem zajišťují signály externí adresové/datové sběrnice.

5. Odpalovací rampa

Rampa musí mít nastavitelnou tyč, ze které odpalujeme rakety, kvůli směru a rychlosti větru. Nastavením azimutu a sklonu tyče můžeme částečně nastavit směr letu rakety. Na straně rampy, by měla být dioda, která nám indikuje připojení palníku.

Rampa je vybavena druhým použitým programovatelným logickým obvodem XILINX, který zajišťuje potřebné vstupy a výstupy. XILINX je ovládán mikrokontrolérem AT89C51CC03. Přenos informací potřebných pro pohyb obou modelářských servomotorů je realizován pomocí sériové linky RS 485. Do rampy je potřeba přivést jak stejnosměrné napětí 12 V, tak stejnosměrné napětí 5 V. SS napětí 5 V, docílíme integrovaným stabilizátorem na 5 V. SS napětí 12 V je potřeba pro nabití kondenzátoru. Přes vybíjení tohoto kondenzátoru se po získání informace pro odpálení zažehne palník a tím pádem se odpálí raketa.

Spínání zažehnutí palníku je realizováno polem řízeným tranzistorem (FET). Výhody oproti bipolárním tranzistorům je - velmi malý úbytek napětí v sepnutém stavu a tedy i malé výkonové ztráty. Z toho pak vyplývá menší (nebo žádný) chladič, menší rozměry i hmotnost. Velmi důležitá je i jednoduchost buzení těchto tranzistorů napětím přímo z výstupů logických obvodů, operačních zesilovačů nebo komparátorů.

Pro získání informace o přítomnosti nebo nepřítomnosti palníku jsou použity komparátory obvodu LM 393. Na schématu je vidět, že komparátory jsou zapojeny proti sobě, tím získáme hysterezi.

5.1. Návrh pohonu k natočení vodící tyče

Pro realizaci natočení vodící tyče na rampě pomocí servomotorů, je potřeba přepočítat kartézské souřadnice na válcové.

Převodní vztahy pro zaměřovací program odpalovací rampy

Vstupy:

- náklon [°] – experimentálně byl zjištěn největší možný rozsah a ten musel být zanesen do programu jako omezení, aby nebyly zničeny servomotory
- azimut [°] – je vztažen k člověku, který stojí za opalovacím pultem, směrem kterým člověk kouká je azimut rovný nule, poté doprava otáčení tyče je azimut kladný

Výstupy:

- α – natočení jižního servomotoru
- β – natočení východního servomotoru

Konstanty:

Konstanty byly změřeny a vloženy do programu. Pro lepší práci s nimi byly všechny vloženy ve stejných jednotkách.

h – vzdálenost mezi kulovým uchycením vodicí tyče a rovinou šoupátek

X_0 – vzdálenost mezi osou servomotoru X a místem, kudy prochází tyčka šoupátkem při nulovém náklonu

Y_0 - vzdálenost mezi osou servomotoru Y a místem, kudy prochází tyčka šoupátkem při nulovém náklonu

Výpočet:

Nejprve si zavedeme pomocnou proměnnou S, která popisuje absolutní vzdálenost bodu, ve kterém se mají křížit táhla od nulové polohy. Hodnotu S lze vypočítat ze vzorce:

$$S = \mathrm{tg}\,\sigma \cdot h$$

Kde σ je náklon vodicí tyče.

S azimutem není třeba provádět nic. Pouze v případě, že by bylo třeba definovat světové strany jinak, než je servomotor X na jihu a servomotor Y na východě, by bylo možné v tomto bodě udělat matematické pootočení přičtením nějaké konstanty.

Další krok je zjištění hodnot sinu a kosinu azimutu:

$$\sin A = A_s$$

$$\cos A = A_c$$

Je nutné věnovat pozornost tomu, zda řídící program počítá se stupni, či s radiány, v případě že s radiány, je nutné je převést na stupně (nebo azimut na radiány).

Následuje zjištění relativních souřadnic *x* a *y*:

$$x = A_c \cdot S$$

$$y = A_s \cdot S$$

Tyto souřadnice udávají relativní polohu místa, kde se má nacházet křížení táhel oproti nulové poloze, tedy poloze křížení při nulovém náklonu. Jednotky souřadnic jsou stejné, jako jednotky, v nichž jsou změřeny konstanty.

Následuje převod na absolutní souřadnice, tedy souřadnice, na jejichž osách leží osy otáčení obou servomotorů.

$$X = X_0 + x$$

$$Y = Y_0 + y$$

Jednotky souřadnic zůstávají stejné.

Nyní již lze zjistit úhly natočení obou servomotorů:

$$\alpha = tg^{-1}\left(\frac{y}{X}\right)$$

$$\beta = tg^{-1}\left(\frac{x}{Y}\right)$$

Tyto úhly je opět nutné převést z radiánů na stupně (nebo s nimi počítat jako s radiány). Orientace je zvolena tak, že pro oba servomotory platí: Dívám-li se na servomotor z jeho horní strany (na „vrtulku“) a táhlo míří směrem ode mě, při záporných úhlech se servomotor natáčí vlevo (proti směru hodinových ručiček) a při kladných úhlech se natáčí doprava (po směru hodinových ručiček).

5.2. Stabilizátor napětí

Integrovaný stabilizátor napětí LM 7805CT

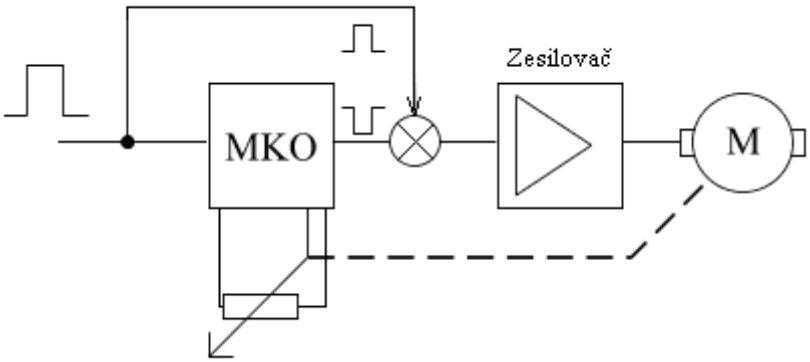
Tento stabilizátor je lineární a používá na stabilizování kladné větve napětí. Zde stabilizuje 12 V SS na 5 V SS. Maximální výstupní proud je 1 A.

Obrázek 5.1 : Integrovaný stabilizátor napětí LM 7805CT

5.3. Ovládání servomotorů

RC (radio control) servomotor:

Obrázek 5.2 : Řez modelářským servomotorem



Obrázek 5.3 : Vnitřní zapojení modelářského servomotoru

Přivedeme vstupní pulz do monostabilního klopného obvodu (0,9 – 2,1 ms); 0,9 ms značí jednu krajní polohu (natočení 0°) a 2,1 ms značí druhou krajní polohu (natočení 360°). Za MKO porovnáme pulz přivedený na vstup s pulzem z MKO. Pokud je jejich rozdíl nulový, motor se neotočí. V případě že není nulový, motor se otáčí do té doby, než se pulzy vyrovnají. Motor se poté zastaví. Pohyb motoru je přenesen přes zpětnou vazbu a potenciometr do MKO.

Modelářské RC servomotory se připojují kablíkem o třech žílách, dvě žíly jsou pro napájení (DC 4,8 – 6 V) a třetí kablík pro přenos impulzů.

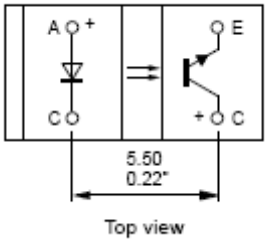
Program na ovládání servomotorů je napsán tak, že se oba servomotory pohybují společně, a při tomto pohybu opisují kružnici o určitém poloměru.

6. Měření rychlosti větru

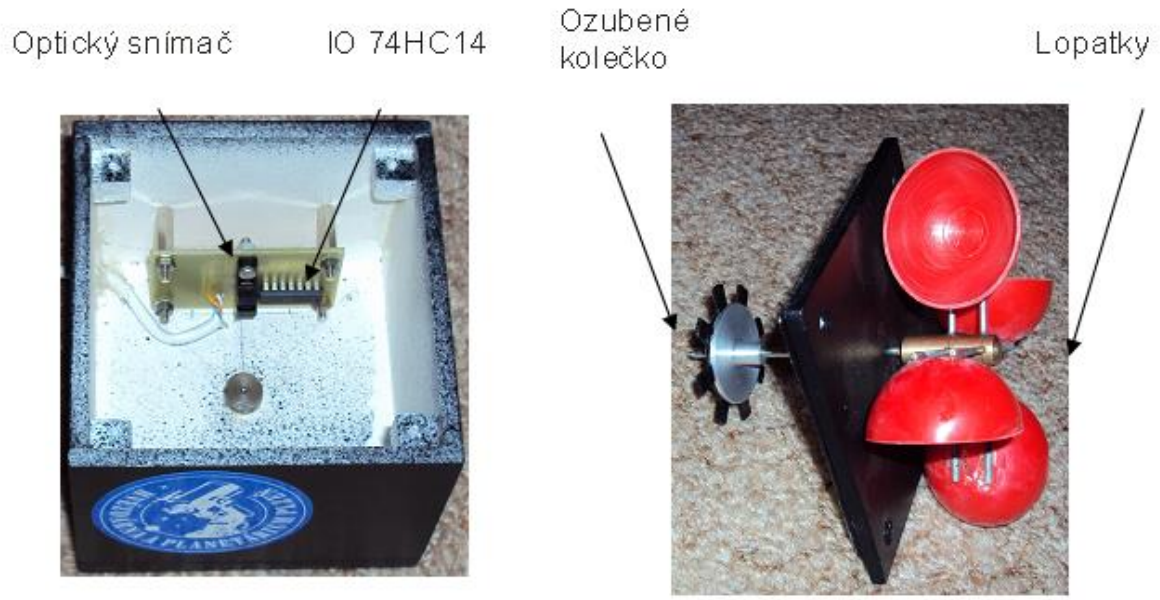
Měření rychlosti větru je provedeno optickým snímačem TCST2000. Optickým snímačem prochází kolečko o osmi zubech, při každém proběhnutí zubu snímačem je vygenerován pulz. Tento pulz se poté dále zpracuje. V mikroprocesoru se převede frekvence na rychlost větru. Pro úpravu signálu ze snímače slouží integrovaný obvod 74HC14. Informace o intenzitě větru je důležitá proto, že při vysoké rychlosti větru již není možné podle bezpečnostních předpisů rakety odpalovat.

Optický snímač TCST2000:

Obrázek 6.1 : Optický snímač TCST2000



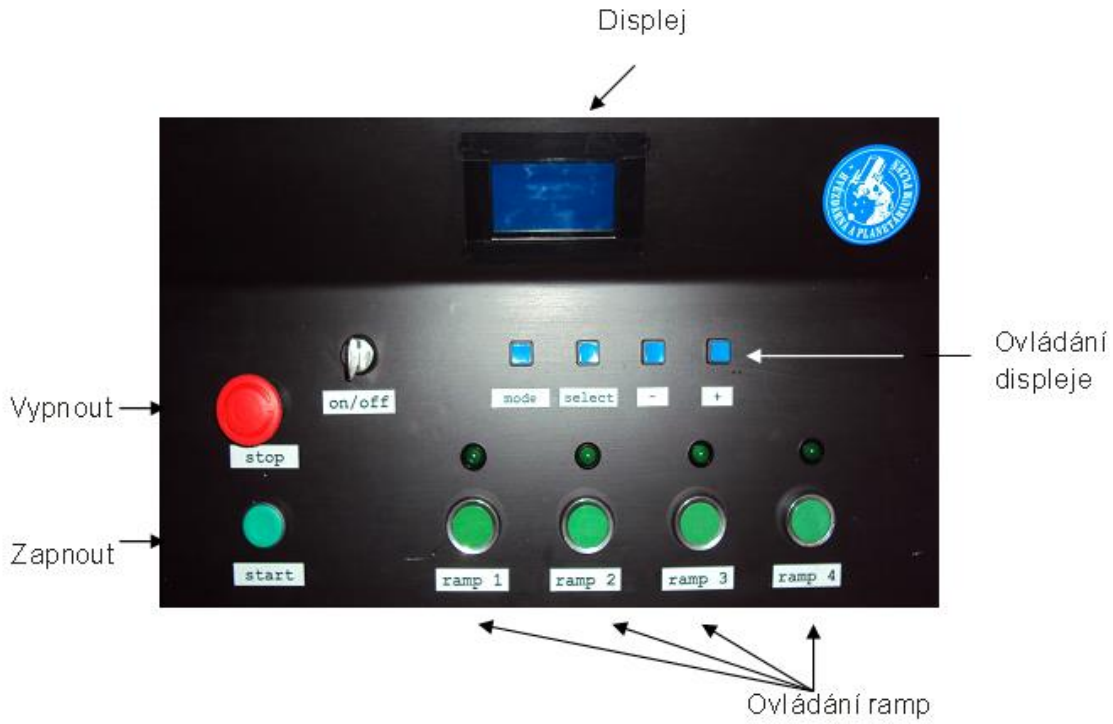
Obrázek 6.2 : Vnitřní zapojení optického snímače



Obrázek 6.3 : Popis rozebraného měřiče rychlosti větru

7. Popis a návod na použití ovládacího pultu

7.1. Ovládací pult



Obrázek 7.1 : Popis odpalovacího pultu

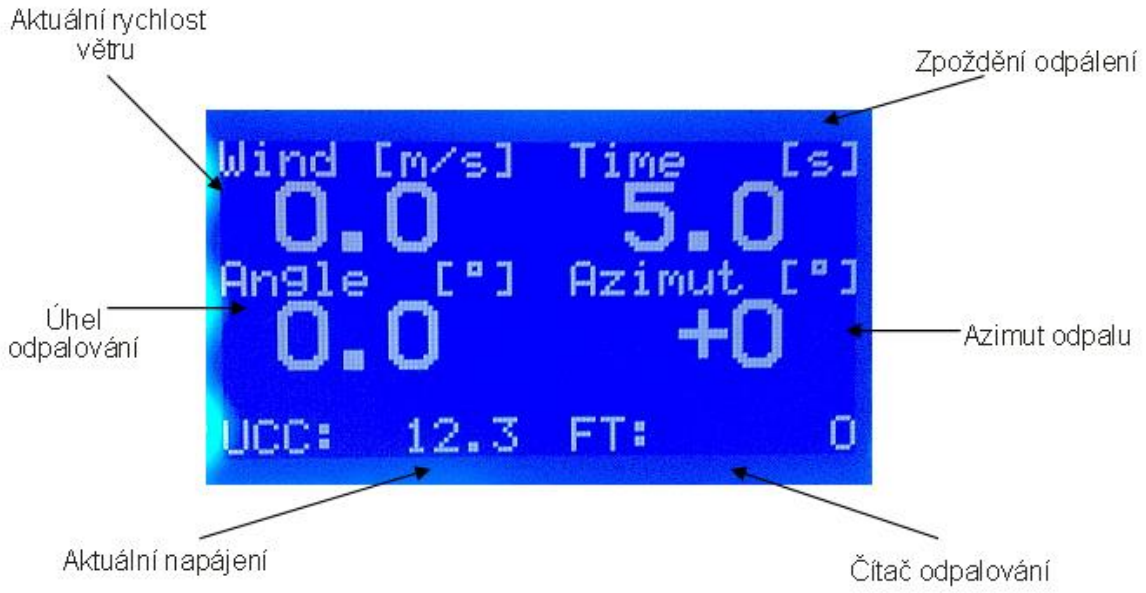
7.2. Návod na použití pultu

Do celého pultu je potřeba přivést stejnosměrné napětí 12 V. Dále je nutné zapojit do příslušných konektorů kabel od krabičky na měření větru a kabel od odpalovací rampy. Otočíme klíčem, tím se odblokuje napájecí obvod, nyní můžeme zapnout celý pult tlačítkem start. Zapne se nám displej, na kterém vidíme příslušné informace (viz obrázek). Rampa by měla být instalována do největší možné vzdálenosti od pultu a měřič rychlosti větru musí být umístěn 2 m nad zemí, jelikož v této výšce zpravidla raketa opouští vodící tyč rampy. Pomocí modrých tlačítek se můžeme pohybovat na displeji a nastavovat různé údaje. Použitím čtyř zelených tlačítek (pro každou rampu jedno) můžeme odpalovat rakety, které manuálně umístíme na rampu a připojíme k nim palník. Po připojení palníku se nám rozsvítí zelená LED dioda na pultu a zelená LED dioda na rampě (signalizace připojení okruhu palníku).

7.3. LCD displej a jeho součásti

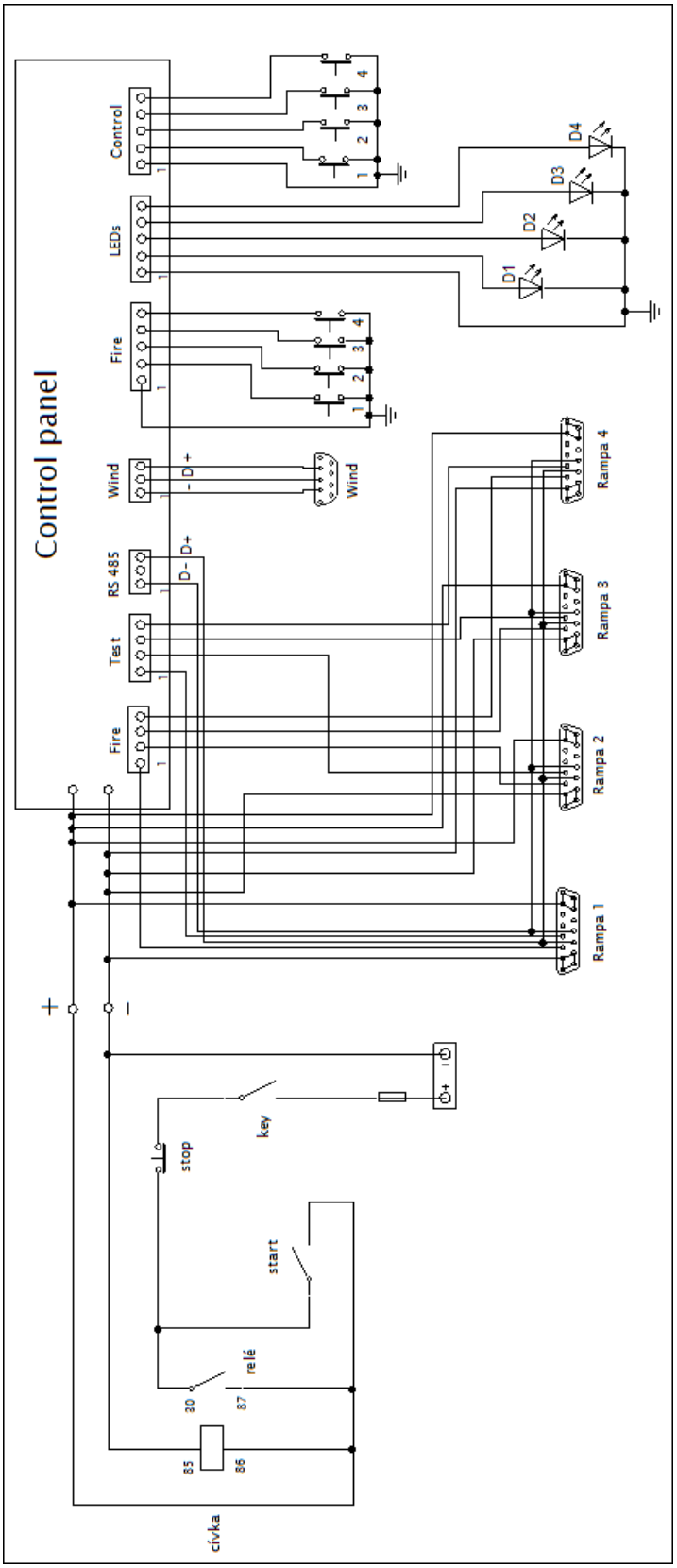
Další blok na základní desce je LCD displej, který je ovládán přes mikroprocesor a také je spojen s programovatelným logickým obvodem. Na LCD displeji si lze zvolit určitá kritéria pro odpal raket. Tlačítka, přes která se ovládá displej, jsou vedena přes mikroprocesor. Signalizace o uzavřeném okruhu palníků vychází z programovatelného logického obvodu XILINX.

7.4. Popis grafického LCD displeje

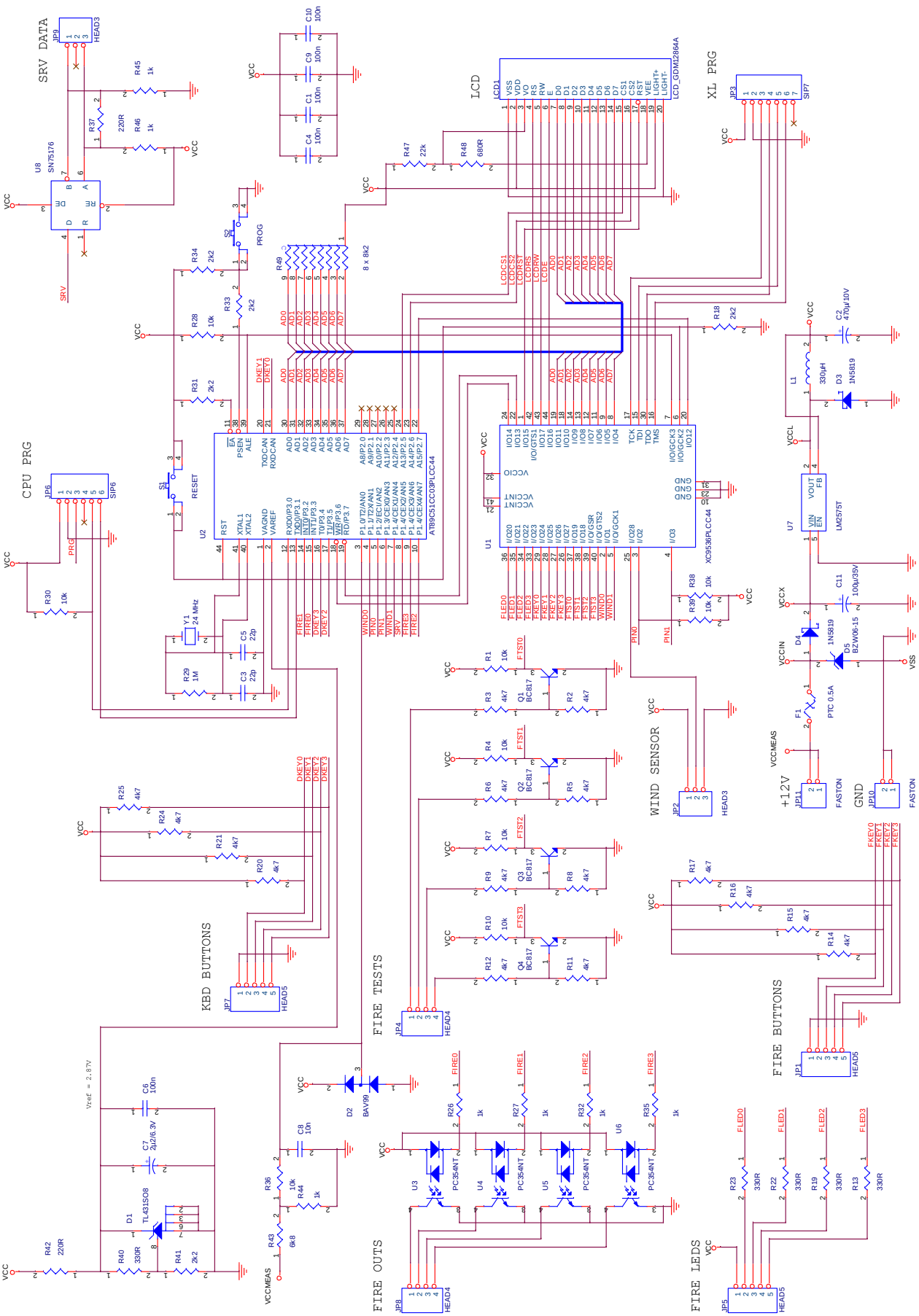


Obrázek 7.2 : Popis grafického displeje použitém na ovládacím pultu

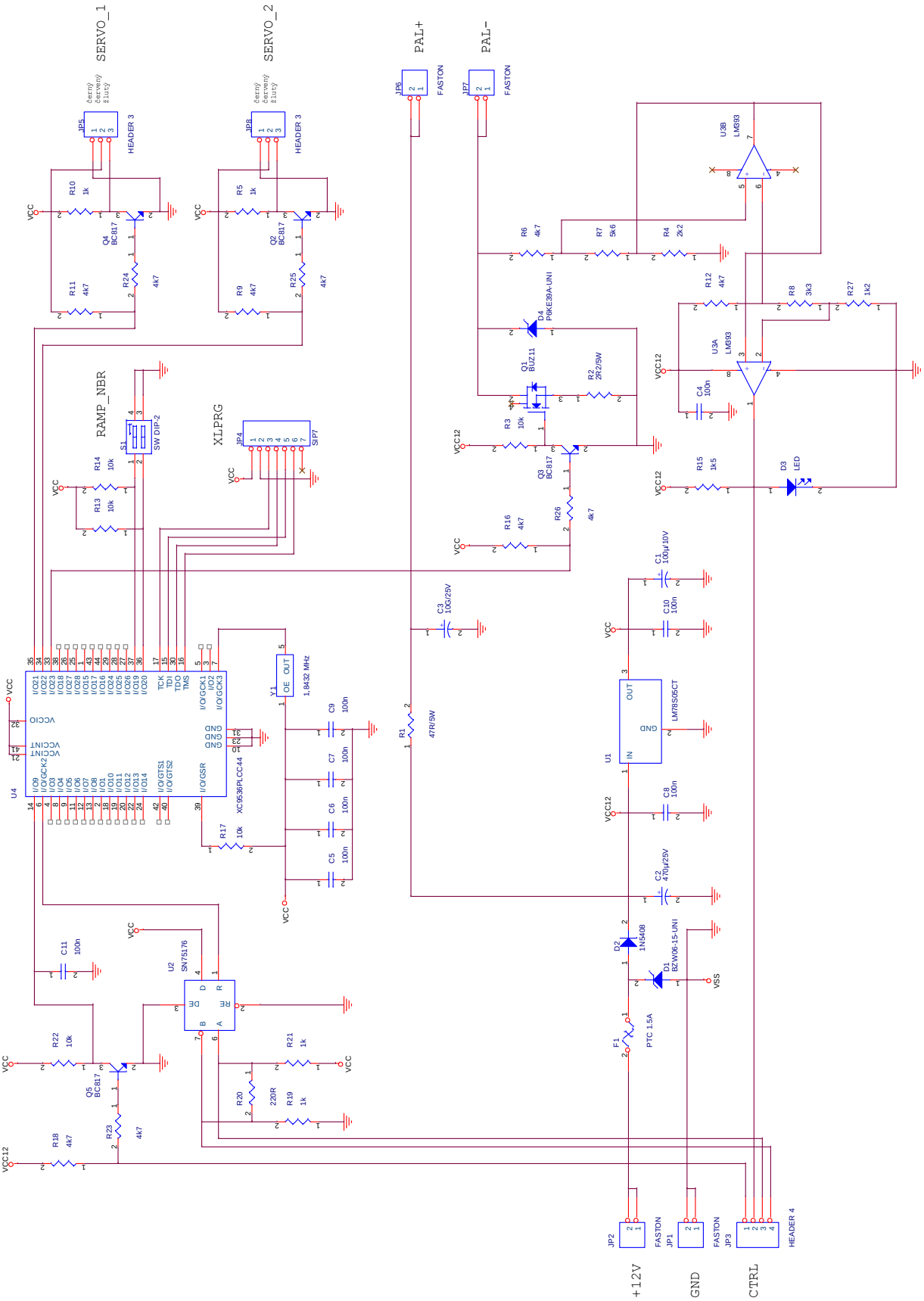
8. Schémata zapojení



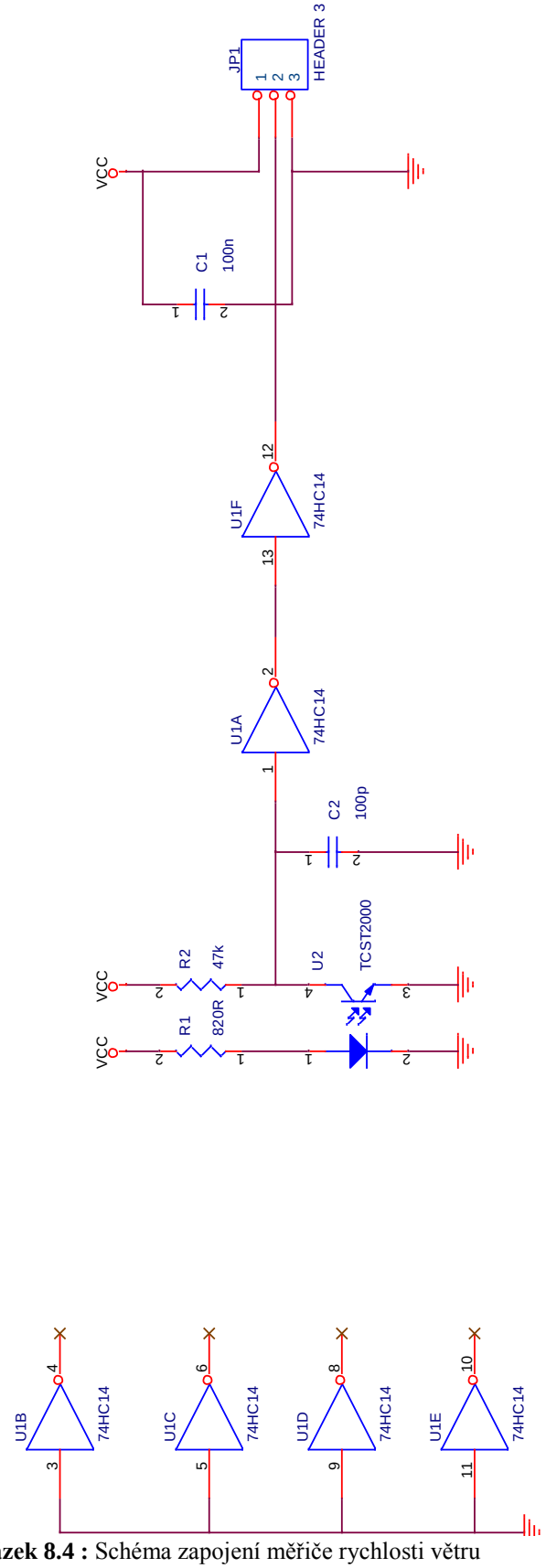
Obrázek 8.1 : Blokové schéma zapojení ovládacího pultu



Obrázek 8.2 : Schéma zapojení ovládacího pultu



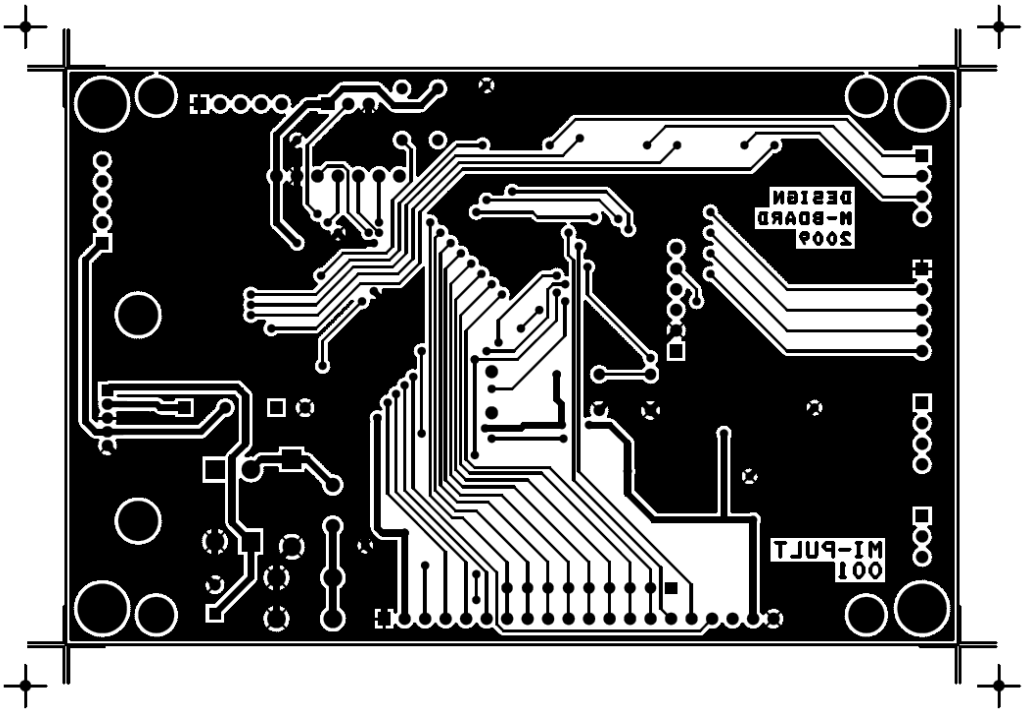
Obrázek 8.3 : Schéma zapojení odpalovací rampy



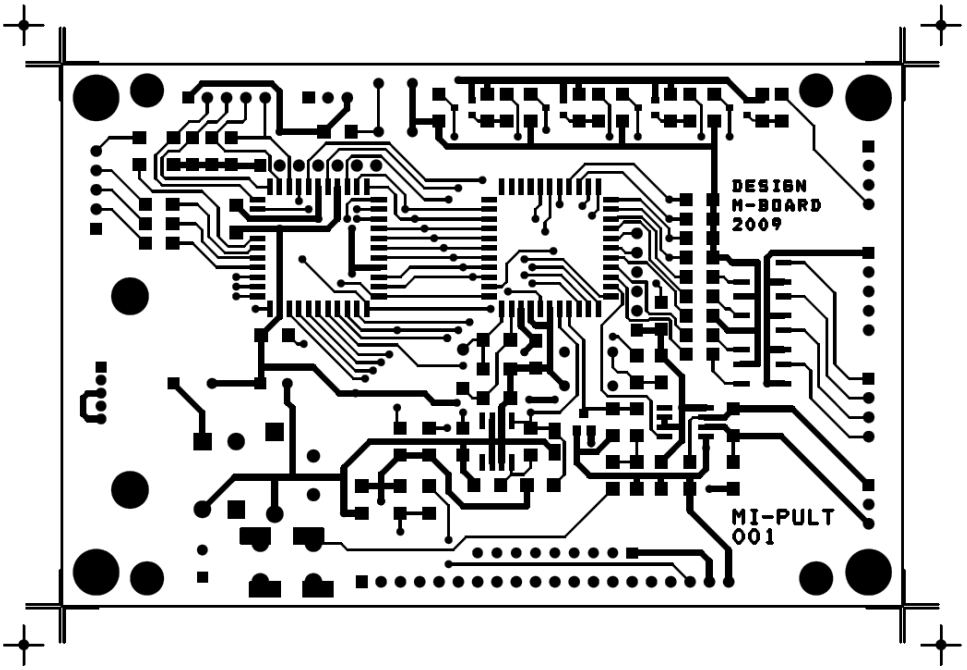
Obrázek 8.4 : Schéma zapojení měřiče rychlosti větru

9. Nákresy plošných spojů

9.1. Ovládací pult

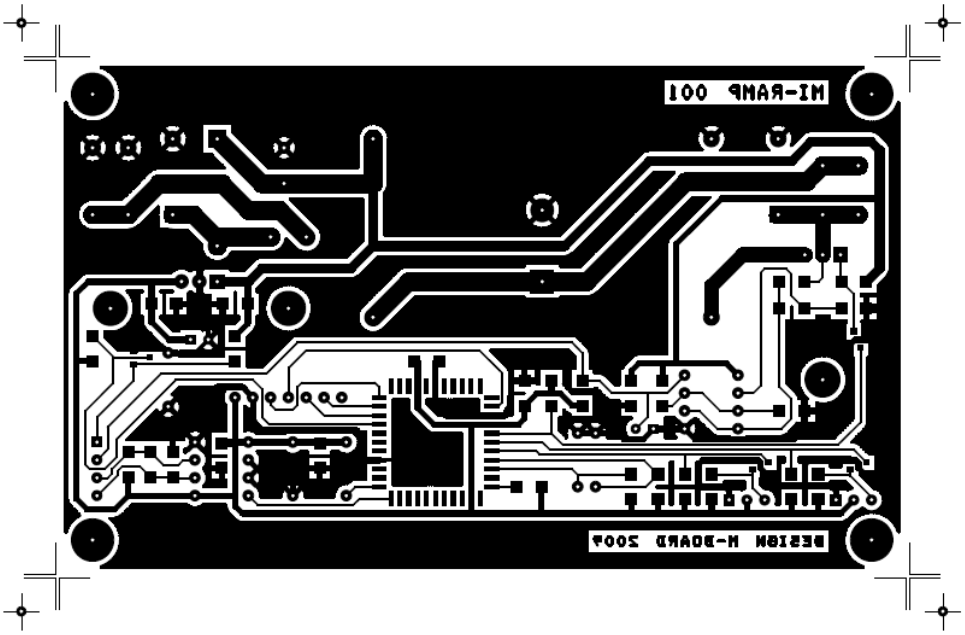


Obrázek 9.1 : Spodní strana plošného spoje



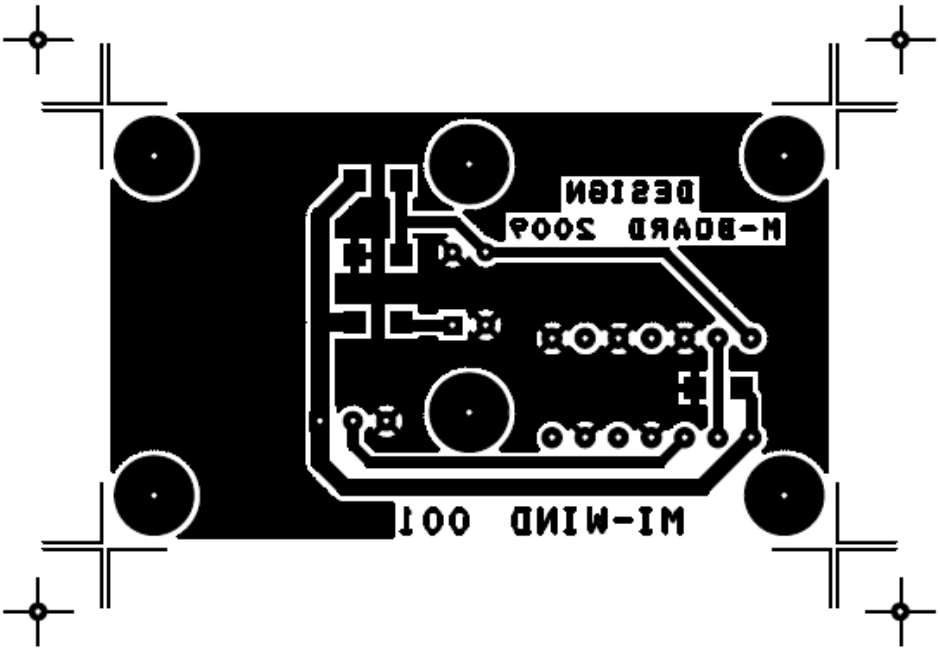
Obrázek 9.2 : Horní strana plošného spoje

9.2. Odpalovací rampa



Obrázek 9.3 : Spodní strana plošného spoje

9.3. Měření rychlosti větru

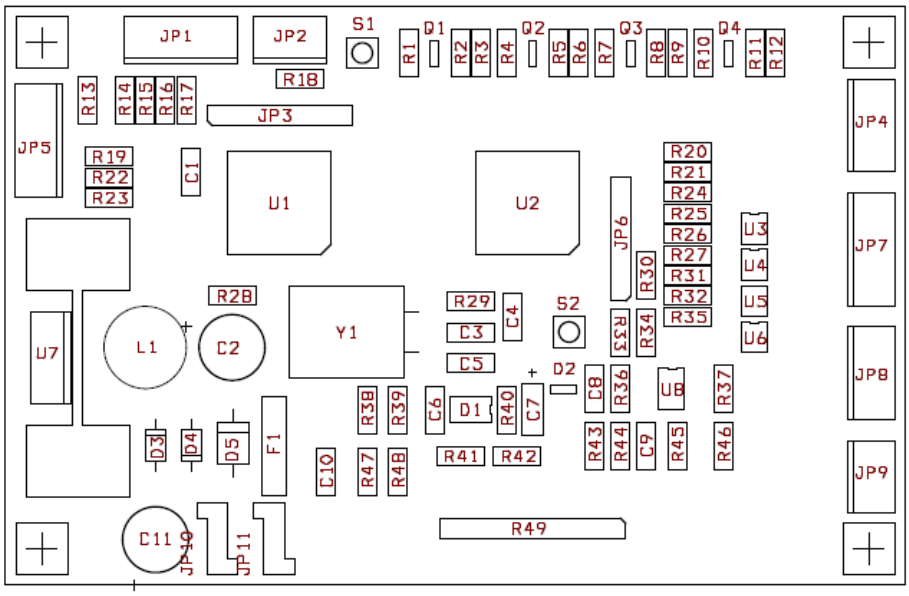


Obrázek 9.4 : Spodní strana plošného spoje

10. Předlohy použité k osazení plošných spojů

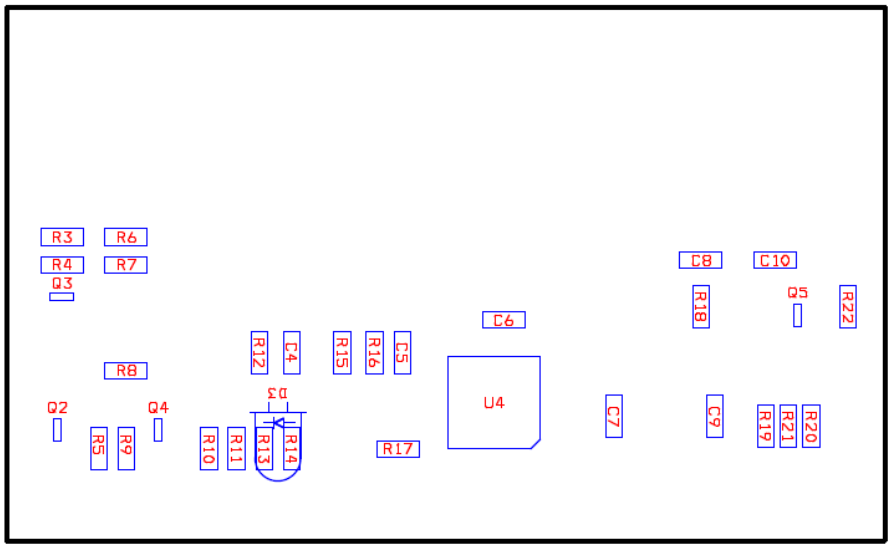
10.1. Odpalovací pult

Spodní strana osazení plošného spoje – na spodní straně plošného spoje je osazen LCD displej

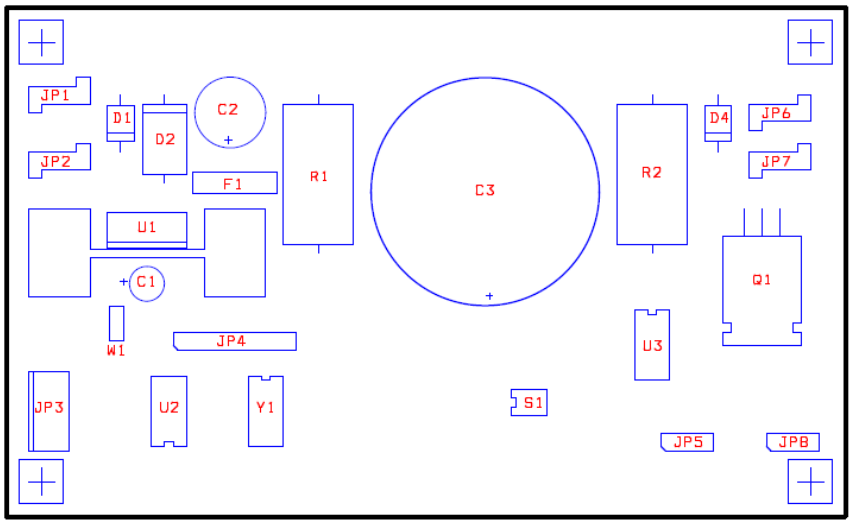


Obrázek 10.1 : Horní strana osazení plošného spoje

10.2. Odpalovací rampa

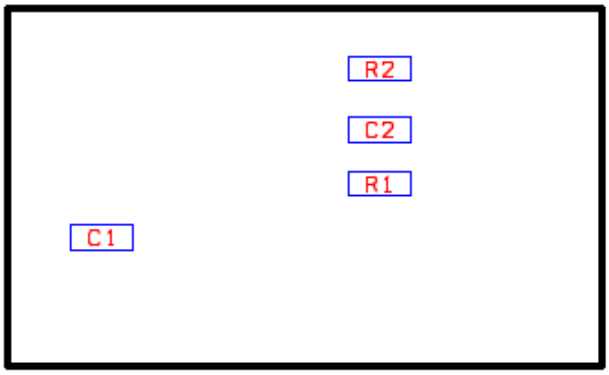


Obrázek 10.2 : Spodní strana osazení plošného spoje

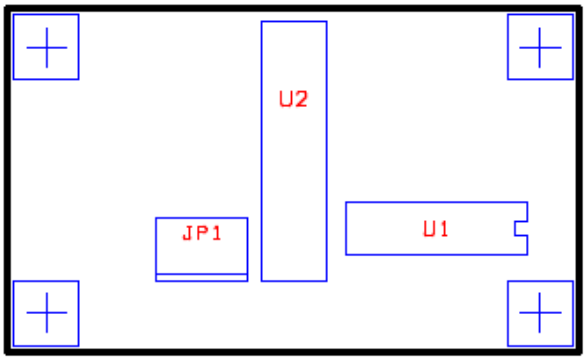


Obrázek 10.3 : Horní strana osazení plošného spoje

10.3. Měření rychlosti větru



Obrázek 10.4 : Spodní strana osazení plošného spoje



Obrázek 10.5 : Horní strana osazení plošného spoje

11. Použité součástky

11.1. Odpalovací pult

Součástka	Hodnota (označení)
C1,C4,C6,C9,C10	100n
C2	470µ/10V
C3,C5	22p
C7	2µ2/6,3V
C8	10n
C11	100µ/35V
D1	TL431
D2	BAV99
D3,D4	1N5819
D5	BZW06-15
F1	PTC 0,5A
JP1,JP5,JP7	vidlice 5 pin
JP2,JP9	vidlice 3 pin
JP3	lišta 7 pin
JP8,JP4	vidlice 4 pin
JP6	lišta 6 pin
JP11,JP10	FASTON
LCD1	LCD MG12864A-SBC/H
L1	330µH
Q1,Q2,Q3,Q4	BC817
R1,R4,R7,R10,R28,R30,R36,R38,R39	10k
R2,R3,R5,R6,R8,R9,R11,R12,R14	4k7
R15,R16,R17,R20,R21,R24,R25	4,k7
R13,R19,R22,R23,R40	330R
R18,R31,R33,R34,R41	2k2
R26,R27,R32,R35,R44,R45,R46	1k
R29	1M
R42,R37	220R
R43	6k8
R47	22k
R48	680R
R49	8 × 8k2
S1	P-B1720B
S2	P-B1720B
U1	XC9536 PLCC44
U2	AT89C51CC03 PLCC44
U3,U4,U5,U6	PC354NT
U7	LM2575T-05
U8	SN75176BP
Y1	Q 24 MHz FUND

11.2. Odpalovací rampa

Součástka	Hodnota (označení)
C1	100µ/10V
C2	470µ/25V
C3	10G/25V
C4,C5,C6,C7,C8,C9,C10,C11	100n
D1	BZW06-15
D2	1N5408
D3	LED 10 mm
D4	P6KE39A
F1	PTC 1,5A
JP1,JP2,JP6,JP7	FASTON
JP3	vidlice 4 pin
JP4	lišta 7 pin
JP5,JP8	vidlice 5 pin
Q1	BUZ11
Q2,Q3,Q4,Q5	BC817
R1	47R/5W
R2	2R2/5W
R3,R13,R14,R17,R22	10k
R4	2k2
R5,R10,R19,R21	1k
R6,R9,R11,R12,R16,R18,R23,R24,R25,R26	4k7
R7	5k6
R8	3k3
R15	1k5
R20	220R
R27	1k2
S1	DIP 02
U1	LM7805CT
U2	SN75176BP
U3	LM393
U4	XC9536 PLCC44
Y1	Q 1,8432 MHz

11.3. Měření rychlosti větru

Součástka	Hodnota (označení)
C1	100n
C2	100p
JP1	vidlice 3 pin
R1	820 R
R2	47k
U1	74HC148
U2	TCST2000

12. Přínos práce

Zhotovením této práce jsem se dozvěděl řadu informací z oblasti elektroniky a programování. Musel jsem najít mnoho informací a prostudovat některé materiály související s touto prací. Nejsložitější pro mne bylo programování mikroprocesoru. Na další problémy jsem narazil při výrobě mechanických částí práce. Během zkoušení výrobku se objevily drobné problémy na plošných spojích, které musely být pro správnou funkci odstraněny. Všechny části práce jsem vyrobil sám, mimo napájecího zdroje, který byl zakoupen.

Výrobek bude používán Hvězdárnou a planetáriem Plzeň. Bude sloužit, jako demonstrační prostředek pro odpalování raket. Zatím je k ovládacímu pultu připojena pouze jedna odpalovací rampa, ale pult je připraven i na připojení zbylých tří ramp, které se chystám zhotovit v budoucnu.

13. Závěr

Požadavek na přesnost nastavení polohy vodicí tyče odpalovací rampy nebyl implicitně stanoven. Přesnost nastavení je limitována přesnosti modelářských servomotorů a mechanických dílů, ve kterých je ukotvena vodicí tyč. Nastavení polohy konce vodicí tyče na pomyslné kružnici s nulovým a největším poloměrem (300 mm) bude proto záviset na přesnosti mechanických dílů (cca 0,5°) a přesnosti servomotoru (1°) s tím, že chyba nastavení servomotoru je přenášena na konec vodicí tyče přes páku s horním ramenem (odpalovací) 880 mm dolním ramenem 90 mm. U běžných modelářských servomotorů, je přesnost natočení v rozmezí 1° – 2°. Tedy chyba na spodním rameni 1° se projeví na konci horního ramene jako chyba nastavení 16 mm.

Měřič rychlosti větru byl vyroben amatérsky, takže přesnost měření je ovlivněna nedokonalostí mechanických částí měřiče rychlosti. Přesnost měření rychlosti větru nebyla stanovena, byla pouze odhadnuta na 15%. Údaj o rychlosti větru je pouze pro zamezení odpálení rakety při vysokých rychlostech větru. Odpalovací pult s celým příslušenstvím, je používán pro osobní účely, nebo při akcích pořádaných Hvězdárnou a planetáriem Plzeň, pro kterou byl tento projekt vytvořen. Obvody rampy a ovládacího panelu byly realizovány a jsou plně funkční.

14. Použité zdroje informací

[1] http://raketove.modely.sweb.cz

[2] http://www.pyroshop.cz

[3]http://hw.cz

[4]http://cs.wikipedia.org

[5]http://pandatron.cz

[6] Pinker, J., Poupa, M.: Číslicové systémy a jazyk VHDL,
BEN Praha 2006, ISBN 80-7300-198-5

[7]Herout, P.: Učebnice jazyka C, KOPP České Budějovice 1994, ISBN 80-85828-21-9

[8] Karel Jeřábek a kol.: Raketové modely (časopis), Naše vojsko Praha 1983

Přílohy

14.1. Zdrojové kódy pro odpalovací rampu

```
library IEEE;                                                   -- standard library
use IEEE.STD_LOGIC_1164.ALL;
                                                        -- component entity definition
entity MonoStable is
        generic(  ToutCnt : integer := 0;                       -- timeout constant
                  EdgeTrig : boolean := false );                -- trigger type
        port(     Clock : in std_logic;                         -- basic clock
                  Enable : in std_logic;-- enable input for timing
                  Start : in std_logic;                         -- synchronous start signal
                  Output : out std_logic := '0' );              -- timeout output signal
end MonoStable;

                                                        -- component architecture
architecture MS of MonoStable is

        signal    Started   : std_logic := '0';                 -- internal "started state" register

                                                        -- component body
        begin
                  Timer : process (Clock, Enable, Start)                -- component procces
                  variable  Ti                  : integer range 0 to ToutCnt;       -- internal timeout variable

                  begin
                            if Clock'event and Clock = '1' then         -- clock rising edge
                                      if Ti > 0 then                    -- timeout is running
                                                Output <= '1';          -- output active
                                      else
                                                Output <= '0';          -- output inactive
                                      end if;

                                      if Enable = '1' then              -- enable pulse active
                                                Started <= Start;       -- synchronized start register
                                                if EdgeTrig = true and Start = '1' and Started = '0' then-- edge trigger
                                                          Ti := ToutCnt;            -- start-up timeout value
                                                elsif  EdgeTrig = false and Start = '1' then    -- level trigger
                                                          Ti := ToutCnt;            -- start-up timeout value
                                                else                                -- counting
                                                          if Ti > 0 then            -- timeout running
                                                                    Ti := Ti - 1;   -- count down
                                                          else
                                                                    Ti := 0;        -- stay at zero
                                                          end if;
                                                end if;
                                      else
                                                Ti := Ti;                   -- no change
                                      end if;
                            end if;
                  end process Timer;

end MS;

-----------------------------------
-- Components definition package
-----------------------------------

library IEEE;                                                   -- standard library
use IEEE.STD_LOGIC_1164.all;
```

```
package McComps is                              -- package definition
                                                -- retriggerable MonoStable
        component MonoStable
                generic(  ToutCnt : integer := 0;               -- timeout constant
                          EdgeTrig : boolean := false );        -- trigger type
                port(     Clock : in std_logic;                 -- basic clock
                          Enable : in std_logic;                -- enable input for timing
                          Start : in std_logic;                 -- synchronous start signal
                          Output : out std_logic );             -- timeout output signal
        end component MonoStable;

end McComps;

package body McComps is                                         -- package body

-- empty by that time

end McComps;

library IEEE;                                                   -- standard library
use IEEE.STD_LOGIC_1164.ALL;

use work.McComps.all;                                           -- components package, definition

entity RampaPld is                                              -- main entity, inputs & outputs
        Port(     Clock             : in std_logic;             -- basic clock
                  xServoData        : in std_logic;             -- negative serial data input (servo control)
                  ServoSelect       : in std_logic_vector(1 downto 0);-- servo selection signals from switches
                  xServo1           : out std_logic;            -- output (negative) for servo 1
                  xServo2           : out std_logic;            -- output (negative) for servo 2
                  FireIn            : in std_logic;             -- negative FIRE input
                  xFireOut  : out std_logic                     -- negative FIRE output
                );
end RampaPld;

architecture Struct of RampaPld is                              -- main architecture

                                                                -- servo select coding typedef
        type SrvCode is (Srv1, Srv2, Srv3, Srv4, Srv5, Srv6, Srv7, Srv8);

                                                                -- constants
        constant  XtalFreq_Hz               : integer  := 1843000;      -- basic clock frequency

        constant  SrInTimeConst_us  : integer  := 1500;         -- serial data start pulse time [us]
        constant  FilterTimeConst_us: integer  := 17;           -- serial data filter time [us]
        constant  FireTimeConst_us  : integer  := 10000;        -- fire pulse time [us]

                                                                        -- global signals
        signal BaseEna100us             : std_logic;                    -- 100us enable signal
        signal TimerOut                 : std_logic;                    -- serial data monostable output
        signal ActSrv                   : SrvCode := Srv1;              -- actual servo code
        signal Srv1Sel, Srv2Sel         : SrvCode;                      -- servo 1 & 2 code settings
        signal FireOut                  : std_logic;                    -- fire monostable input & output
        signal FilterOut                : std_logic;                    -- servo data glitch filter output
        signal OldFilterOut             : std_logic;

                                                                        -- body of the structure
        begin
                SrInTout : MonoStable       generic map(
                ToutCnt => SrInTimeConst_us / 100,                      -- serial data start timing
                EdgeTrig => false)                                      -- no edge triggered
                                                                port map(
                                                                        Clock => Clock,
                                                                        Enable => BaseEna100us,
```

```
                                                    Start => not(xServoData),
                                                    Output => TimerOut );

InFilter : MonoStable       generic map(
ToutCnt => FilterTimeConst_us * XtalFreq_Hz / 1000000,      -- filter timing
EdgeTrig => true)                                           -- edge triggered
                                                port map(
                                                    Clock => Clock,
                                                    Enable => '1',
                                                    Start => xServoData,
                                                    Output => FilterOut );

Fire    : MonoStable    generic map(
                ToutCnt => FireTimeConst_us / 100,          -- fire timing
                EdgeTrig => true)                           -- edge triggered
                                                port map(
                                                    Clock => Clock,
                                                    Enable => BaseEna100us,
                                                    Start => FireIn,
                                                    Output => FireOut );
xFireOut <= not FireOut;

OnClk : process (Clock)                                     -- servo data proccess, clocked
constant  Div100Const       : integer := XtalFreq_Hz / 10000;-- recompute base freq. to 100us counter
        variable  Div100            : integer range 0 to Div100Const;       -- counter variable

        begin
                if Clock'event and Clock='1' then                       -- clock rising edge
                        if Div100 < Div100Const then                    -- not overflow
                                Div100 := Div100 + 1;                   -- count up
                                BaseEna100us <= '0';                    -- enable signal inactive
                        else                                            -- overflow
                                Div100 := 0;                            -- clear counter
                                BaseEna100us <= '1';        -- enable signal active pulse
                        end if;

                        OldFilterOut <= FilterOut;
                        if TimerOut = '0' then                  -- monostable timeout
                                ActSrv <= Srv1;                 -- clear sequence state machine

        -- valid rising edge on input signal
                        elsif FilterOut = '1' and OldFilterOut = '0' then
                                case ActSrv is                  -- servo state machine
                                        when Srv1 => ActSrv <= Srv2;
                                        when Srv2 => ActSrv <= Srv3;
                                        when Srv3 => ActSrv <= Srv4;
                                        when Srv4 => ActSrv <= Srv5;
                                        when Srv5 => ActSrv <= Srv6;
                                        when Srv6 => ActSrv <= Srv7;
                                        when Srv7 => ActSrv <= Srv8;
                                        when Srv8 => ActSrv <= Srv1;-- back to first state
                                end case;
                        end if;
                end if;
end process OnClk;

with ServoSelect select                                     -- servo1 selection by switches
        Srv1Sel <=      Srv1 when "00",                     -- first pair
                        Srv3 when "01",                     -- second pair
                        Srv5 when "10",                     -- third pair
                        Srv7 when others;                   -- fourth pair
with ServoSelect select                                     -- servo2 selection by switches
        Srv2Sel <=      Srv2 when "00",                     -- first pair
                                Srv4 when "01",             -- second pair
```

```
                                                Srv6 when "10",         -- third pair
                                                Srv8 when others;       -- fourth pair

        xServo1 <= xServoData when Srv1Sel = ActSrv else '1';           -- servo1 output
        xServo2 <= xServoData when Srv2Sel = ActSrv else '1';           -- servo2 output

end Struct;
```

14.2. Zdrojové kódy ovládacího pultu

```
library IEEE;                                                           -- standard library
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity Pult001 is                                                       -- main entity, inputs & outputs
    Port(
            Brst            : in    std_logic;
            Ale             : in    std_logic;
            nWr             : in    std_logic;
            nRd             : in    std_logic;
            Ad              : inout std_logic_vector(7 downto 0);
            A15             : in    std_logic;

            WindIn          : in    std_logic;
            Ftst            : in    std_logic_vector(3 downto 0);
            Fkey            : in    std_logic_vector(3 downto 0);
            Fled            : out   std_logic_vector(3 downto 0);
            LcdRstIn  : in  std_logic;

            nLcdReset       : out   std_logic;
            LcdEnable       : out   std_logic;
            LcdRs           : out   std_logic;
            LcdRw           : out   std_logic;
            Wind0           : out   std_logic;
            Wind1           : out   std_logic
    );
end Pult001;

architecture Struct of Pult001 is                                       -- main architecture

    constant StatusVal : std_logic_vector(7 downto 0) := "10101010";
    constant LcdRead        : std_logic := '1';
    constant LcdData        : std_logic := '1';

    constant LcdWrite : std_logic := not(LcdRead);
    constant LcdCtrl        : std_logic := not(LcdData);

    signal      LowAddress      : std_logic_vector(2 downto 0);
    signal      StatusReg       : std_logic_vector(7 downto 0);
    signal      Address         : std_logic_vector(3 downto 0);
    signal      LedsReg         : std_logic_vector(3 downto 0);

    signal      FkeyData : std_logic_vector(7 downto 0);
    signal      FtstData  : std_logic_vector(7 downto 0);
    signal      FireLeds  : std_logic_vector(7 downto 0);
    signal      RdData          : std_logic_vector(7 downto 0);

    begin

        Address(3) <= A15;
        Address(2 downto 0) <= LowAddress;
        FtstData(7 downto 4) <= "0000";
```

```
		FtstData(3 downto 0) <= not(Ftst);
		FkeyData(7 downto 4) <= "0000";
		FkeyData(3 downto 0) <= not(Fkey);
		FireLeds(7 downto 4) <= "0000";
		FireLeds(3 downto 0) <= LedsReg;

		nLcdReset <= not(LcdRstIn or Brst);
		LcdEnable <= not(nWr) or not(nRd) when Address(3 downto 2) = "10" and Brst = '0' else '0';
		LcdRw <= LcdWrite when Address(3 downto 1) = "101" else LcdRead;
		LcdRs <= LcdCtrl when Address = "1011" else LcdCtrl when Address = "1001" else LcdData;
		Fled <= not(LedsReg);

		Ad <= RdData when nRd = '0' and Address(3 downto 2) = "11" else "ZZZZZZZZ";

		with Address select                                              -- RD address decoder
			RdData <=	StatusReg	when "1111",
					FkeyData	when "1110",
					FtstData	when "1101",
					FireLeds	when "1100",
					"00000000"	when others;

		AddressLatch : process(Ale,Brst)                                 -- low address latch
		begin
			if Brst = '1' then                                       -- async. reset by Brst
				LowAddress <= "000";
			else
				if Ale'event and Ale='0' then                    -- Ale falling edge
					LowAddress <= Ad(2 downto 0);            -- latch address
				end if;
			end if;
		end process AddressLatch;

		DataWrite : process(Ale,Brst,nWr)                                -- write data to registers
		begin
			if Brst = '1' then                                       -- async. resets by Brst
				StatusReg <= "00000000";
				LedsReg <= "0000";
			else
				if Ale'event and Ale='1' then                    -- Ale rising edge
					StatusReg <= StatusVal;                  -- status value to register
				end if;
				if nWr'event and nWr='1' then                    -- nRW risng edge, end of write
					case Address is
						when "1100" => LedsReg <= Ad(3 downto 0);
						when others => null;
					end case;
				end if;
			end if;
		end process DataWrite;

		Wind0 <= WindIn;
		Wind1 <= not(WindIn);

end Struct;


//---------------------------------
// Pult001 servo communication (PCA)
//---------------------------------

#include <Atmel.h>
#include "ServoComm.h"

static	idata		unsigned short	PCApulse[8];
```

```
static	code const	unsigned short	PCAstart[8] =	{	0, 7200, 14400, 21600,
		28800, 36000, 43200, 50400
	};
static				unsigned char	PCAindex = 0;			// index of pulses


				// PCA update value
void PCA_UpdatePulse(unsigned char Index, unsigned short Length)
{
	if (Length > 7000)
		return;

	Length += PCAstart[Index];				// add start-time
	EC = 0;
				// disable PCA interrupts
	PCApulse[Index] = Length;				// update pulse value
	EC = 1;
				// enable PCA interrupts
}

void PCA_Setup(bit DefValues)
	// PCA setup
{
	unsigned char	i;

	if (DefValues)
		// default values
		for (i=0;i<8;i++)
			// first default pulses
			PCA_UpdatePulse(i, 6000);

	CMOD = CMOD_ECF;
				// ECF, CPS => Fosc / 6(12), from PCAX2 too
	CCAP2H = 0;
				// first PCA zero cross
	CCAP2L = 0;
	CCAPM2 = CCAPMX_ECOM|CCAPMX_MAT|CCAPMX_TOG|CCAPMX_ECCF;	// high speed out mode for PCA2,
interrupt enabled
	EC = 1;
				// enable PCA interrupts
	CR = 1;
				// PCA timer run
}
void PCA_Handler(void) interrupt 6				// PCA common interrupt
{
	if (CF)
				//  PCA counter over, zero cross
	{
		CF = 0;
				//  clear flag
		P1_5 = 0;
				// first active pulse, hard synchronization
		PCAindex = 0;
	}

	if (CCF2)
				// PCA2 flag
	{
		CCF2 = 0;
				// clear flag
		if (!P1_5)
				// after space, after toggle
```

```
                {
                        CCAP2H = (unsigned char)(PCApulse[PCAindex] >> 8);       // pulse length value
                        CCAP2L = (unsigned char)(PCApulse[PCAindex] & 0x00FF);

                        if (PCAindex < 7)
                        // next pulse index
                                PCAindex++;
                        else
                                PCAindex = 0;
                }
                else
                        // after pulse
                {
                        CCAP2H = (unsigned char)(PCAstart[PCAindex] >> 8);       // next pulse start value
                        CCAP2L = (unsigned char)(PCAstart[PCAindex] & 0x00FF);
                }
                CCAPM2 |= CCAPMX_ECOM;
                        // re-enable PCA2
        }
}

//-------------------------------
// Pult001 display visualization
//-------------------------------

#include <Lcd12864.h>
#include <TextFonts.h>
#include <EeDriver.h>
#include "IoSpacePult001.h"
#include "Pult001Cfg.h"
#include "Screens.h"

#include <stddef.h>
#include <stdio.h>

                unsigned long   Wind = 0;                                       // wind speed value
                unsigned short  FireClock = 0,                                  // fire activity timer
                                                CpuTime = 65535;        // minimum cree CPU cycles
                float                   Ucc = 0.;                               // UCC value

                unsigned char   Cursor = 0;                                     // cursor position index

static  bit                             CursorBlink = 0;                        // cursor blinking


                        // fonts
static  code    tFontTypeDef    Font5x7         = {5, 1, 32, 127, 1, 0, Ericsson_GA628_5x7, LcdByteWrite};
static  code    tFontTypeDef    Font10x14       = {10, 2, 32, 63, 1, 0, Ericsson_GA628_10x14, LcdByteWrite};
static  code    tFontTypeDef    FontSym9x8      = {9, 1, 32, 40, 0, 0, Symbol_9x8, LcdByteWrite};
//static  code  tFontTypeDef    Font13x23       = {13, 3, 32, 63, 1, 0, Arial_Narrow_13x23, LcdByteWrite};

static void ScreenMain(tScrAction Action)                                       // MAIN screen
{
        switch (Action)
        {
                case ScrFastUpd:
                        PreparePrint(&Font10x14, 4, 0);
                        printf("%4.1f", Pult001Cfg.Alfa / 10.);
                        PreparePrint(&Font10x14, 1, 70);
                        printf("%4.1f", (float)Pult001Cfg.FireTime / 20.);
                        PreparePrint(&Font10x14, 4, 70);
                        printf("%+4d", Pult001Cfg.Rotation);

                        PreparePrint(&Font5x7, 7, 99);
```

```
                        printf("%5u", FireClock / 20);
                break;

                case ScrUpd:
                        PreparePrint(&Font10x14, 1, 0);
                        if (Wind > 999)
                                puts("----");
                        else
                        printf("%4.1f", (float)Wind / 10.);
                        PreparePrint(&Font5x7, 7, 36);
                        printf("%4.1f", Ucc);
                        PreparePrint(&Font10x14, 1, 117);
                        puts(CursorBlink && Cursor == 1 ? "*" : " ");
                        PreparePrint(&Font10x14, 4, 47);
                        puts(CursorBlink && Cursor == 2 ? "*" : " ");
                        PreparePrint(&Font10x14, 4, 117);
                        puts(CursorBlink && Cursor == 3 ? "*" : " ");
                break;

                default:
                        PreparePrint(&Font5x7, 0, 0);
                        puts("Wind [m/s]");
                        PreparePrint(&Font5x7, 3, 0);
                        puts("Angle  [`]");
                        PreparePrint(&Font5x7, 0, 70);
                        puts("Time   [s]");
                        PreparePrint(&Font5x7, 3, 70);
                        puts("Azimut [`]");

                        PreparePrint(&Font5x7, 7, 0);
                        puts("UCC:");
                        PreparePrint(&Font5x7, 7, 70);
                        puts("FT:");

/*
                        LcdLineH(6, 0x28, 0, 127);
                        LcdLineV(7, 7, 42);
                        LcdLineV(0, 7, 56);
                        LcdLineV(0, 7, 70);
                        LcdLineV(7, 7, 84);
                        LcdLineH(6, 0xE8, 42, 42);
                        LcdLineH(6, 0xEF, 56, 56);
                        LcdLineH(6, 0xEF, 70, 70);
                        LcdLineH(6, 0xE8, 84, 84);

                        LcdLineH(6, 0x2B, 59, 67);
*/
        }
}

static void ScreenTrim(tScrAction Action)                       // TRIMMER (second) screen
{
        switch (Action)
        {
                case ScrFastUpd:
                        PreparePrint(&Font5x7, 0, 84);
                        printf("%+05d", Pult001Cfg.Trim[0][0]);
                        PreparePrint(&Font5x7, 1, 84);
                        printf("%+05d", Pult001Cfg.Trim[0][1]);
                        PreparePrint(&Font5x7, 2, 84);
                        printf("%+05d", Pult001Cfg.Trim[1][0]);
                        PreparePrint(&Font5x7, 3, 84);
                        printf("%+05d", Pult001Cfg.Trim[1][1]);
                        PreparePrint(&Font5x7, 4, 84);
```

```
                printf("%+05d", Pult001Cfg.Trim[2][0]);
                PreparePrint(&Font5x7, 5, 84);
                printf("%+05d", Pult001Cfg.Trim[2][1]);
                PreparePrint(&Font5x7, 6, 84);
                printf("%+05d", Pult001Cfg.Trim[3][0]);
                PreparePrint(&Font5x7, 7, 84);
                printf("%+05d", Pult001Cfg.Trim[3][1]);
        break;

        case ScrUpd:
                PreparePrint(&Font5x7, 0, 120);
                puts(CursorBlink && Cursor == 1 ? "*" : " ");
                PreparePrint(&Font5x7, 1, 120);
                puts(CursorBlink && Cursor == 2 ? "*" : " ");
                PreparePrint(&Font5x7, 2, 120);
                puts(CursorBlink && Cursor == 3 ? "*" : " ");
                PreparePrint(&Font5x7, 3, 120);
                puts(CursorBlink && Cursor == 4 ? "*" : " ");
                PreparePrint(&Font5x7, 4, 120);
                puts(CursorBlink && Cursor == 5 ? "*" : " ");
                PreparePrint(&Font5x7, 5, 120);
                puts(CursorBlink && Cursor == 6 ? "*" : " ");
                PreparePrint(&Font5x7, 6, 120);
                puts(CursorBlink && Cursor == 7 ? "*" : " ");
                PreparePrint(&Font5x7, 7, 120);
                puts(CursorBlink && Cursor == 8 ? "*" : " ");
        break;

        default:
                PreparePrint(&Font5x7, 0, 0);
                puts("Trim R1_S1:");
                PreparePrint(&Font5x7, 1, 0);
                puts("Trim R1_S2:");
                PreparePrint(&Font5x7, 2, 0);
                puts("Trim R2_S1:");
                PreparePrint(&Font5x7, 3, 0);
                puts("Trim R2_S2:");
                PreparePrint(&Font5x7, 4, 0);
                puts("Trim R3_S1:");
                PreparePrint(&Font5x7, 5, 0);
                puts("Trim R3_S2:");
                PreparePrint(&Font5x7, 6, 0);
                puts("Trim R4_S1:");
                PreparePrint(&Font5x7, 7, 0);
                puts("Trim R4_S2:");
    }
}

static void ScreenServis(tScrAction Action)                    // SERVIS (third) screen
{
    switch (Action)
    {
        case ScrFastUpd:
                PreparePrint(&Font5x7, 5, 84);
                printf("%5.3f", (float)Pult001Cfg.WindTrim / 1000.);
        break;

        case ScrUpd:
                PreparePrint(&Font5x7, 2, 84);
                printf("%u", CpuTime);
                PreparePrint(&Font5x7, 3, 84);
                printf("%lu", Pult001Cfg.WriteCount);
                PreparePrint(&Font5x7, 5, 120);
                puts(CursorBlink && Cursor == 1 ? "*" : " ");
```

```
                break;

                default:
                        PreparePrint(&Font5x7, 0, 19);
                        puts("*** SERVICE ***");
                        PreparePrint(&Font5x7, 2, 0);
                        puts("CPU timing   :");
                        PreparePrint(&Font5x7, 3, 0);
                        puts("EE write cnt :");
                        PreparePrint(&Font5x7, 5, 0);
                        puts("Wind trim    :");
        }
}

void RunScreen(tScrAction Action, unsigned char Index)                  // screen real-time update
{
        switch (Action)
        {
                case ScrFixed:          LcdClear();         break;      // clear LCD
                case ScrFastUpd:    break;
                case ScrUpd:            CursorBlink = !CursorBlink;  break;   // cursor inversion

                default:   return;
        }

        switch (Index)
        {
                case 0:    ScreenMain(Action);              break;
                case 1:    ScreenTrim(Action);              break;
                case 2:    ScreenServis(Action);        break;
        }
}

//-----------------------------------------
// Pult001 I/O space registers & functions
//-----------------------------------------

#include <Atmel.h>
#include <Lcd12864.h>
#include "IoSpacePult001.h"

// ---------------- interface address constants

#define   LCD_UCC_STARTUP       1000                    // start-up wait cycles for Ucc stabilization
#define   CPLD_STATUS_VAL 0xAA                          // status check fixed value from CPLD

#define   IO_SPACE_ADDR   0x8004                        // global control & status struct xdata address

// ---------------- virtual variables

tIoSpace  IoSpace   _at_      IO_SPACE_ADDR;            // global control/status registers within CPLD

tLcdIface Lcd0 _at_ LCD0_ADDR;                          // LCD left half interface
tLcdIface Lcd1 _at_ LCD1_ADDR;                          // LCD right half interface

// ---------------- functions

void Wait4IoUccOk(void)                                 // wait for CPLD start-up & Ucc stabilization
{
        unsigned long       i;

        while (IoSpace.CheckCpld != CPLD_STATUS_VAL);   // check CPLD status
        for(i=0;i<LCD_UCC_STARTUP;i++);                 // wait for Ucc - display start-up
}
```

```
void RstLineFc(bit Activate)                                    // LCD RESET line function
{
        if (Activate)
                LCD_RST_BIT = 1;
        else
                LCD_RST_BIT = 0;
}

//-----------------------
// Pult001 configuration
//-----------------------

#include <EeDriver.h>
#include "Pult001Cfg.h"

xdata   tPult001Cfg                     Pult001Cfg;
        // configuration


code    const   tPult001Cfg     DefPult001Cfg =  {      // default configuration
                                                        0,
                                                        0, 0, 0,
                                                        0, 0, 0, 0, 0, 0, 0, 0,
                                                        1000,
                                                 };

void ReadCheckCfg(void)
{
        unsigned char       i, j;

        EeReadData(&Pult001Cfg, 0, sizeof(Pult001Cfg));                 // read data from EEPROM
        if (Pult001Cfg.WriteCount == 0xFFFFFFFF)
                Pult001Cfg.WriteCount = DefPult001Cfg.WriteCount;
        if (Pult001Cfg.FireTime > MAX_FIRE_TIME)
                Pult001Cfg.FireTime = DefPult001Cfg.FireTime;
        if (Pult001Cfg.Alfa > MAX_ALFA_ANGLE)
                Pult001Cfg.Alfa = DefPult001Cfg.Alfa;
        if ((Pult001Cfg.Rotation > MAX_AZIMUT) || (Pult001Cfg.Rotation < -MAX_AZIMUT))
                Pult001Cfg.Rotation = DefPult001Cfg.Rotation;

        for (i=0;i<2;i++)
                for (j=0;j<4;j++)
                        if ((Pult001Cfg.Trim[i][j] > MAX_SRV_TRIM) || (Pult001Cfg.Trim[i][j] < -MAX_SRV_TRIM))
                                Pult001Cfg.Trim[i][j] = DefPult001Cfg.Trim[i][j];

        if (Pult001Cfg.WindTrim > MAX_WIND_TRIM)
                Pult001Cfg.WindTrim = DefPult001Cfg.WindTrim;
}

//-----------------------
// Pult001 main programm
//-----------------------

#include <Atmel.h>
#include <Lcd12864.h>
#include <EeDriver.h>
#include <AdcDriver.h>
#include "Pult001Cfg.h"
#include "Screens.h"
#include "ServoComm.h"
#include "Wind.h"

#include <stddef.h>
```

```
#include <math.h>

typedef  enum    {_Fwait, _Ftime, _Fprep, _Fout, _Fend}tFireState;
typedef  struct  Fire {
                                    tFireState          State;
                                    unsigned char       Wait;
                                    unsigned char       Time;
                                    unsigned char       Led;
                           }tFire;

typedef  enum    {_Bwait, _Bon, _Brpt}tButtonState;
typedef  struct  Button {
                                    tButtonState        State;
                                    unsigned char       Wait;
                                    unsigned char       Rpt;
                                    unsigned char       Flag;
                           }tButton;

static   unsigned short   Cnt50ms = TIME_50ms;                  // 50ms timing counter
static   unsigned char    CntDisp = 0,                          // display update counter [50ms]
                                    ScreenIndex = 0,
                  // actual screen index
                                    ScreenReturn = 0,
                  // counter for return to main screen (SCR_MAININDEX)
                                    ButtState = 0,
                  // actual buttons state
                                    BcCnt = 0;
                  // buttons timing counter [50ms]
static   bit                      Flag50ms = 0,
                  // 50ms timer flag
                                    ButtChange = 0;
                  // buttons change flag

static   tFire            Fire[4];
static   tButton          Button[4];

static   signed short     Asrv1, Asrv2;

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

static void StartUp(void)                                       // basic startup function
{
         SET_X2_MODE();
                  // X2-mode
         DISABLE_ALE();
                  // ALE with external memory cycles only
         USE_INTRAM();
                  // internal RAM
         WdStart(WD_TOUT_DIV);
                  // setup & start WD

#ifndef SIMULATE
         LcdReset();
                  // reset LCD
         Wait4IoUccOk();
                  // wiat for  correct UCC
#endif
         LcdClear();
                  // clear entire LCD

                  // init timer 0 - main timer
         CKCON |= 0x02;
                  // T0 - X2 independent
```

```
	TMOD |= 0x02;
		// T0 - 8bit auto-reload
	TH0 = TL0 = 256 - TIMING_100us;
		// 100us tminig value
	ET0 = 1;
		// enable INT
	TR0 = 1;
		// enable timer
}

void MainTimer(void) interrupt 1				// 100us timer interrupt handler
{
	if (!--Cnt50ms)
		// each 50ms
	{
		Cnt50ms = TIME_50ms;

		Flag50ms = 1;
	}
}

void DispButtons(void)					// button's handler
{
	unsigned char	i;

	for (i=0;i<4;i++)
		switch (Button[i].State)
		{
			case _Bwait:
				if (DBT_DATA & (1<<i))
				{
					Button[i].Flag = 1;
					Button[i].Wait = 0;
					Button[i].Rpt = 0;
					Button[i].State = _Bon;
				}
			break;

			case _Bon:
				if (!(DBT_DATA & (1<<i)) && !Button[i].Flag)
					if (Button[i].Wait < DBT_OFFTIME)
						Button[i].Wait++;
					else
						Button[i].State = _Bwait;
				else
				{
					Button[i].Wait = 0;
					if (i > 1)					// plus & minus keys
						if (DBT_DATA & (1<<i))
							if (Button[i].Rpt < DBT_FIRST_RPT)
								Button[i].Rpt++;
							else
							{
								Button[i].Flag = 1;
								Button[i].State = _Brpt;
							}
						else
							Button[i].Rpt = 0;
				}
			break;

			case _Brpt:
				if (!(DBT_DATA & (1<<i)))
				{
```

```
                                        Button[i].Wait = 0;
                                        Button[i].Rpt = 0;
                                        Button[i].State = _Bon;
                                }
                                else
                                {
                                        if (Button[i].Wait < DBT_REPEAT)
                                                Button[i].Wait++;
                                        else
                                        {
                                                Button[i].Flag = 1;
                                                Button[i].Wait = 0;
                                        }
                                }
                        break;

                        default:
                                Button[i].State = _Bwait;
                }
}

void MakeCursor(void)
{
        unsigned char       MaxCursor;

        switch (ScreenIndex)
        {
                case 0:   MaxCursor = SCR_CRNBR_MAIN;       break;
                case 1:   MaxCursor = SCR_CRNBR_ADV;        break;
                case 2:   MaxCursor = SCR_CRNBR_CFG;        break;

                default:  return;
        }

        if (Cursor < MaxCursor)
                Cursor++;
        else
                Cursor = 1;
}
void ChangeData(void)
{
        signed char                 Change;
        unsigned char       i, j;

        if (Button[DBTM_PLUS].Flag)                         // plus switch action
        {
                ScreenReturn = SCREEN_TIMMING;              // setup timer for return to main screen & clear cursor
                Button[DBTM_PLUS].Flag = 0;
                Change = 1;
        }
        else
                if (Button[DBTM_MINUS].Flag)                // minus switch action
                {
                        ScreenReturn = SCREEN_TIMMING;      // setup timer for return to main screen & clear cursor
                        Button[DBTM_MINUS].Flag = 0;
                        Change = -1;
                }
                else
                        return;

        switch (ScreenIndex)
        {
                case 0:
                        switch (Cursor)
```

```
		{
			case 1:
				if (Change < 0)
				{
					if (Pult001Cfg.FireTime > 1)
						Pult001Cfg.FireTime -= 2;
				}
				else
				{
					if (Pult001Cfg.FireTime < (MAX_FIRE_TIME-1))
						Pult001Cfg.FireTime += 2;
				}
			break;

			case 2:
				if (Change < 0)
				{
					if (Pult001Cfg.Alfa > 0)
						Pult001Cfg.Alfa -= 1;
				}
				else
				{
					if (Pult001Cfg.Alfa < MAX_ALFA_ANGLE)
						Pult001Cfg.Alfa += 1;
				}
			break;

			case 3:
				if (Change < 0)
				{
					if (Pult001Cfg.Rotation > -MAX_AZIMUT)
						Pult001Cfg.Rotation -= 1;
					else
						Pult001Cfg.Rotation = MAX_AZIMUT;
				}
				else
				{
					if (Pult001Cfg.Rotation < MAX_AZIMUT)
						Pult001Cfg.Rotation += 1;
					else
						Pult001Cfg.Rotation = -MAX_AZIMUT;
				}
			break;
		}
	break;

	case 1:
		i = (Cursor - 1) >> 1;
		j = (Cursor - 1) & 1;
		if (Change < 0)
		{
			if (Pult001Cfg.Trim[i][j] > -MAX_SRV_TRIM)
				Pult001Cfg.Trim[i][j]--;
		}
		else
		{
			if (Pult001Cfg.Trim[i][j] < MAX_SRV_TRIM)
				Pult001Cfg.Trim[i][j]++;
		}
	break;

	case 2:
		if (Change < 0)
		{
```

```
                                        if (Pult001Cfg.WindTrim > 0)
                                                Pult001Cfg.WindTrim--;
                                }
                                else
                                {
                                        if (Pult001Cfg.WindTrim < MAX_WIND_TRIM)
                                                Pult001Cfg.WindTrim++;
                                }
                        break;
                }
}

void FireMachine(void)                                                  // fire state machine
{
        unsigned char   i,
                                        fd = 0;
        bit                             Active = 0;

        for (i=0;i<4;i++)
                switch (Fire[i].State)
                {
                        case _Fwait:
                                if (IoSpace.FireTests & (1<<i))
                                        IoSpace.FireLeds |= (1<<i);
                                else
                                        IoSpace.FireLeds &= ~(1<<i);
                                if ((IoSpace.FireKeys & (1<<i)) && (IoSpace.FireLeds & (1<<i)))
                                {
                                        Fire[i].Wait = 0;
                                        Fire[i].State = _Ftime;
                                        Fire[i].Time = Pult001Cfg.FireTime;
                                        Active = 1;
                                }
                        break;

                        case _Ftime:
                                Active = 1;
                                if (!Fire[i].Time)
                                {
                                        Fire[i].Wait = F_PREP_WAIT;
                                        Fire[i].State = _Fprep;
                                }
                                else
                                {
                                        Fire[i].Time--;
                                        if (!Fire[i].Wait)
                                        {
                                                IoSpace.FireLeds ^= (1<<i);
                                                if (Fire[i].Time > (MAX_FIRE_TIME / 2))
                                                        Fire[i].Wait = F_BLINK_LONG;
                                                else
                                                        if (Fire[i].Time > (MAX_FIRE_TIME * 3 / 10))
                                                                Fire[i].Wait = F_BLINK_MID;
                                                        else
                                                                Fire[i].Wait = F_BLINK_SHORT;
                                        }
                                        else
                                                Fire[i].Wait--;
                                }
                        break;

                        case _Fprep:
                                Active = 1;
                                IoSpace.FireLeds ^= (1<<i);
```

```
                                        if (!Fire[i].Wait)
                                                Fire[i].State = _Fout;
                                        else
                                                Fire[i].Wait--;
                                break;

                                case _Fout:
                                        Active = 1;
                                        fd |= (1<<i);
                                        IoSpace.FireLeds &= ~(1<<i);
                                        Fire[i].Wait = F_END_WAIT;
                                        Fire[i].State = _Fend;
                                break;

                                default:                                                        // _Fend
                                        Active = 1;
                                        IoSpace.FireLeds &= ~(1<<i);
                                        if (!Fire[i].Wait)
                                                Fire[i].State = _Fwait;
                                        else
                                                Fire[i].Wait--;
                        }

                        FIRE_WRITE(fd);
                        if (Active)
                        {
                                if (FireClock < 65535)
                                        FireClock++;
                        }
                        else
                                FireClock = 0;
}

void MakeAngles(void)                                           // make servo angles from alfa & azimut
{
        float       x, y, arad, r;

        arad = (float)(Pult001Cfg.Alfa) / 10. / DEG2RAD;        // alfa angle to radians
        r = SRV_VDIST * tan(arad);                              // alfa to radius

        arad = (float)(Pult001Cfg.Rotation + 180) / DEG2RAD;    // azimut angle to radians
        x = r * sin(arad);
                // point axis relative to center
        y = r * cos(arad);

                // servo angles
        Asrv2 = (signed short)(SRV_ACONST * (atan((y + SRV_HPOS) / (x + SRV_HPOS)) - PI4));
        Asrv1 = (signed short)(SRV_ACONST * (atan((y - SRV_HPOS) / (x + SRV_HPOS)) + PI4));
}

main()
                // MAIN
{
        unsigned char       i;

        StartUp();
                // basic init
        ReadCheckCfg();
        ADC_Setup(ADCF_CH0);

        MakeAngles();
        for (i=0;i<4;i++)
                // first values for PCA
        {
```

```
                PCA_UpdatePulse((i<<1), 6000 + Pult001Cfg.Trim[i][0] - Asrv1);
                PCA_UpdatePulse((i<<1)+1, 6000 + Pult001Cfg.Trim[i][1] - Asrv2);
        }
        PCA_Setup(0);
                // setup PCA for servo comm
        T2_Setup();

        EA = 1;
                // enable interrupts

        RunScreen(ScrFixed, ScreenIndex);                       // first screen init

        while(1)
                // main loop
        {
                WdRst();
                // reset WD
                if (Flag50ms)
                // timer event
                {
                        Flag50ms = 0;
                // clear flag

                        ADC_StartConv(0);
                // start conversion for UCC measurement

                        FireMachine();
                        DispButtons();

                        if (Button[DBTM_MODE].Flag)
                // screen switch action
                        {
                                Button[DBTM_MODE].Flag = 0;

                                Cursor = 0;
                // clear cursor

                                if (Pult001Cfg.WriteCount < 0xFFFFFFFE)         // save data
                                        Pult001Cfg.WriteCount++;
                                EeWriteData(&Pult001Cfg, 0, sizeof(Pult001Cfg));

                                if (ScreenIndex < SCREENS_NUM - 1)              // new screen index
                                        ScreenIndex++;
                // next USER screen
                                else
                                        ScreenIndex = 0;
                // first USER screen
                                RunScreen(ScrFixed, ScreenIndex);               // init new screen
                                ScreenReturn = SCREEN_TIMMING;                  // setup timer for return to
main screen & clear cursor
                                CntDisp = 0;
                // immediately update
                        }

                        if (ScreenReturn)
                // screen return timing active
                        {
                                ScreenReturn--;
                // count down
                                if (!ScreenReturn)
                // return to main screen now
                                {
                                        Cursor = 0;
                // clear cursor
```

```
                                        if (Pult001Cfg.WriteCount < 0xFFFFFFFE)                 // save data
                                                Pult001Cfg.WriteCount++;
                                        EeWriteData(&Pult001Cfg, 0, sizeof(Pult001Cfg));

                                        if (ScreenIndex != SCR_MAIN_INDEX)              // no main screen
                                        {
                                                ScreenIndex = SCR_MAIN_INDEX;
                        // return to main screen
                                                RunScreen(ScrFixed, ScreenIndex);       // init
                                                CntDisp = 0;
                        // immediately update
                                        }
                                }
                        }

                        if (Button[DBTM_SEL].Flag)                                      // select switch action
                        {
                                ScreenReturn = SCREEN_TIMMING;
                                Button[DBTM_SEL].Flag = 0;
                                MakeCursor();
                        }

                        ChangeData();

                        Ucc = MAKE_UCC(AdcValue);

                        RunScreen(ScrFastUpd, ScreenIndex);             // fast update actual screen
                        if (!CntDisp)
                // time to display update
                        {
                                CntDisp = DISPLAY_TIMING;
                                RunScreen(ScrUpd, ScreenIndex);         // standard update actual screen
                        }
                        else
                        {
                                CntDisp--;
                // timer decrement
                                Wind = T2_Read();
                // wind compute
                                if (Wind)
                // divide by zero check
                                        Wind = ((unsigned long)WIND_CONST) / Pult001Cfg.WindTrim / Wind;
                // compute
                        }

                        MakeAngles();
                        for (i=0;i<4;i++)
                        {
                                PCA_UpdatePulse((i<<1), 6000 + Pult001Cfg.Trim[i][0] - Asrv1);
                                PCA_UpdatePulse((i<<1)+1, 6000 + Pult001Cfg.Trim[i][1] - Asrv2);
                        }

                        ET0 = 0;
                        if (Cnt50ms < CpuTime)
                                CpuTime = Cnt50ms;
                        ET0 = 1;
                }

        }
}

//-----------------------
// Pult001 configuration
```

```
//-----------------------

#ifndef _PULT001CFG_
#define _PULT001CFG_

#include "IoSpacePult001.h"
                                        // main definitions
#define   TIMING_100us      200         // 10kHz, 100us main timing, 2Mhz clock
#define   TIME_50ms         500         // 50ms, [100us] counter
                                        // display button settings
#define   DBT_OFFTIME       2           // off stabilize time [50ms]
#define   DBT_FIRST_RPT     15          // first repeat wait [50ms]
#define   DBT_REPEAT        0           // repeat timing [50ms]
                                        // screen & display timing definitions
#define   SCR_MAIN_INDEX    0           // MAIN screen index
#define   SCR_TRIM_INDEX    1           // TRIM screen index
#define   SCR_SRV_INDEX     2           // SERVIS screen index
#define   SCREENS_NUM       3           // number of screens
#define   SCR_CRNBR_MAIN    3           // number of cursor positions on MAIN screen
#define   SCR_CRNBR_ADV     8           // number of cursor positions on ADV screen
#define   SCR_CRNBR_CFG     1           // number of cursor positions on CFG screen

#define   SCREEN_TIMMING    160         // return to MAIN screen timing 50ms]
#define   DISPLAY_TIMING    5           // display update timing [50ms]
                                        // fire timing settings
#define   F_PREP_WAIT       20          // 1 sec
#define   F_END_WAIT        40          // 2 sec
#define   F_BLINK_LONG      20          // 1 sec
#define   F_BLINK_MID       10          // 0.5 sec
#define   F_BLINK_SHORT     5           // 0.25 sec
                                        // angle compute constants
#define   PI                3.1415926535      // PI number
#define   PI4               (PI/4.)           // 45° ...
#define   DEG2RAD           (180./PI)         // degrees to radian conversion
#define   SRV_HPOS          (80./1.4142135623)// servo horizontal position from center
#define   SRV_VDIST         106.              // servo vertical position from case
#define   SRV_ACONST        (1769./PI4)       // servo angle const [-/°]
                                        // wind compute constants
#define   WIND_R            0.035             // wind-meter radius [m]
#define   WIND_CONST        (2000000.*2.*PI*WIND_R/8.*1000.*10.)// 2MHz clock, 8 pulses, R, trim1000 => [0.1m/s]

                                        // limit values
#define   MAX_EEWR_CNT      0xFFFFFFFE        // max EEPROM counter
#define   MAX_FIRE_TIME     200               // max fire time [0.05s]
#define   MAX_ALFA_ANGLE    150               // max alfa angle [0.1°]
#define   MAX_AZIMUT        180               // max azimut [°]
#define   MAX_SRV_TRIM      1000              // max servo trim [0.25us]
#define   MAX_WIND_TRIM     1000              // max wind-meter trim [-]

typedef   struct    Pult001Cfg {              // configuration structure
                                  unsigned long     WriteCount;     // EEPROM write counter
                                  unsigned char     FireTime;       // fire timer
                                  unsigned char     Alfa;           // alfa angle
                                  signed short      Rotation;       // azimut
                                  signed short      Trim[4][2];     // servo trim values
                                  unsigned short    WindTrim;       // wind-meter constant
                                  }tPult001Cfg;

extern    xdata     tPult001Cfg     Pult001Cfg;                     // configuration variable (XRAM)

extern    void      ReadCheckCfg(void);                             // read & check configuration grom EEPROM

#endif
```