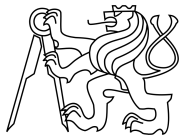
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Vestavný systém pro letecký emulátor
Student:	Michal Buchovecký
Vedoucí:	Ing. Pavel Kubalík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačové inženýrství
Katedra:	Katedra číslicového návrhu
Platnost zadání:	Do konce zimního semestru 2017/18

Pokyny pro vypracování

Prozkoumejte existující řešení hardwarové podpory leteckých simulátorů.
Navrhněte hardwarové řešení podpory leteckého simulátoru - emulátoru.
Pro řešení vyberte vhodný procesor umožňující ovládání MPC (multi control panel) pro Boeing 737 a EFIS (Electronic flight instrument system) panel.
Navržené řešení zrealizujte.
Pro komunikaci s leteckým emulátorem vytvořte skript využívající FSUIPC knihovnu.
Pro ověření správné funkce vytvořte jednoduchou maketu MPC a EFIS panelu.
Výsledné řešení propojte a otestujte s leteckým simulátorem Microsoft Flight Simulator X.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

doc. Ing. Hana Kubátová, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 30. září 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra číslicového návrhu

Bakalárska práca

Vstavaný systém pre letecký emulátor

Michal Buchovecký

Vedúci práce: Ing. Pavel Kubalík, Ph.D.

7. decembra 2016

Poďakovanie

Chcel by som poďakovať všetkým, ktorí mi akokoľvek pomohli pri tvorbe tejto bakalárskej práce, no najmä vedúcemu práce, Ing. Pavlovi Kubalíkovi, Ph.D., za vedenie a cenné pripomienky. Takisto by som chcel poďakovať svojej rodine, ktorí ma urputne podporovali počas celého bakalarského štúdia.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 7. decembra 2016 .....

Odkaz na túto prácu

Buchovecký, Michal. *Vstavaný systém pre letecký emulátor*. Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Táto práca sa zaoberá stavbou ovládacích prvkov určených pre Microsoft Flight Simulator X. Hlavným cieľom je vytvoriť prototyp MCP a EFIS panelu pre lietadlo Boeing 737NG, ktoré budú cenovo nenáročne, a zároveň jednoduché na výrobu. Na stavbu týchto panelov sú použité súčiastky, ktoré sú bežne dostupné na našom trhu.

Výsledkom práce sú produkty založené na vývojovej doske Arduino, ktoré umožňujú jednoduchšie ovládanie simulátora a zároveň poskytujú užívateľovi realistickejší zážitok z lietania.

Klíčová slova Mode Control Panel, Elektronický letový informačný systém, Arduino, Microsoft Flight Simulator X

Abstract

This thesis is focused on building of control components designed for Microsoft Flight Simulator X. The main goal is to build prototype of MCP and EFIS panel for aircraft Boeing 737NG, which will be low cost and also simply for manufacture. The parts which are used for building these panels, are generally available on our market.

The result of this thesis are products, which are based on development board Arduino. These products give more easily controlling of simulator and providing more realistic flying experience to user.

Keywords Mode Control Panel, Electronic Flight Instrument System, Arduino, Microsoft Flight Simulator X

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

V súčasnej dobe moderných technológií si už každý nadšenec lietania môže splniť svoj sen a stať sa aspoň čiastočne pilotom, minimálne v tom virtuálnom svete, ktorý dnes už nemá až tak ďaleko od skutočnej reality. K tomu všetkému nie je potrebný žiadny teoretický ani praktický výcvik. Pre realistickú simuláciu postačuje len výkonnejší počítač, letecký simulátor a výhodou je pripojenie k internetu, odkiaľ je možné čerpať dáta o aktuálnom počasí. V tom momente môže užívateľ začať okamžite svoju kariéru virtuálneho pilota s podmienkami, aké majú v danom čase aj piloti v skutočnosti.

Jedným z najpoužívanejších leteckých simulátorov je Microsoft Flight Simulator X. Hlavnými dôvodmi populárnosti tohto simulátora aj po 10 rokoch, je výborný fyzikálny model, grafika a rozsiahla miera úprav, vďaka ktorej si môžete do simulátora pridávať napríklad nové modely lietadiel, nové scenérie letísk a mnoho ďalšieho, čím je možné udržiavať simulátor stále aktuálny.

Vďaka otvorenému API, ktoré majú takmer všetky letecké simulátory, je možné vyvíjanie hardverových ovládacích prvkov, ktorými sa táto práca bude zaoberať, a tým umožňuje dosiahnuť realistickejšiu simuláciu. V práci priblížim stav súčasných hotových riešení, následne sa budem venovať návrhu vlastného zariadenia až po implementáciu a testovanie zariadenia. Výsledkom by mal byť funkčný prototyp MCP a EFIS panelu.

Kapitola 1

Existujúce riešenia

Na trhu momentálne existuje viacero výrobcov, ktorí ponúkajú na predaj od rôznych komponentov určených na stavbu jednotlivých panelov, hotové Plug and Play zariadenia, až po kompletné kokpity vrátane konštrukcie. Jednotlivé riešenia sa pohybujú v cenových kategóriach od niekoľkých stoviek eur za panel, až po 20 000 eur za kompletný kokpit.

V tejto kapitole si zhrnieme niektoré zaujímavé existujúce riešenia, ktoré sú aktuálne dostupné. Rozdeliť ich môžme do dvoch kategórií. Prvou sú kompletné zariadenia s vlastným riadiacim softvérom. Druhou kategóriou je softvér, ktorý dokáže vygenerovať samotný riadiaci softvér, no hardvér je nutné postaviť vlastnoručne. Táto kategória je zaujímavá pre ľudí, ktorí nevedia programovať, no chcú zariadenie „šité na mieru."

1.1 Kompletné zariadenia

1.1.1 Saitek Pro Flight Multi Panel

Saitek Pro Flight Multi Panel je LED displej s plne ovládateľným autopilotom, ktorý pracuje v reálnom čase s väčšinou leteckých simulátorov. Doplnkové funkcie taktiež podporujú jednoduché ovládanie automatických otáčok motorov, klapiek alebo výškového kormidla.[1]

Tento panel je vhodný ako univerzálne riešenie, keďže pomocou neho, je možné ovládať takmer každé lietadlo v simulátore. Panel je Plug and Play, čo je pri týchto paneloch veľkou výhodou. Jeho nevýhodou je však privysoká cena. Cena tohto panelu je 150 eur.

1.1.2 VRInsight MCP Combo II - Boeing MCP

Zariadenie od VRInsight ponúka všetky funkcie panelu, ktoré sú aj v skutočnom lietadle Boeingu. Podporované sú štandardné lietadlá Boeing v simulátoroch Microsoft Flight Simulator 2004, Microsoft Flight Simulator X

Obr. 1.1: Saitek Pro Flight Multi Panel [1]

Obr. 1.2: VRInsight MCP Combo II - Boeing MCP [2]

a Lockheed Martin PREPAR 3D. Okrem týchto sú podporované aj doplnkové lietadlá od vývojárskych spoločností Level-D a PMDG. K správnemu fungovaniu tohto panela je potrebné nainštalovať doplnkový software VRiSim.[2]

Toto zariadenie je akousi kombináciou EFIS, MCP a COM panelu. Ovládací panel sa približuje tomu, aký môžeme nájsť aj v skutočnom lietadle, čo pridáva užívateľovi pocit ovládania skutočného lietadla, a nie len obyčajnej hry. K dispozícií je aj 8 programovatelných tlačidiel, na ktoré si môže užívateľ nastaviť ovládanie ľubovolných funkcií. Panel okrem iného disponuje aj kompletným podsvietením. Cena za toto zariadenie sa pohybuje okolo 310 eur.

1.1.3 Opencockpits MCP 737NG V3H

Replika MCP Boeingu 737 vo verzii od Honeywellu prináša funkcie rovnaké ako aj v reálnom lietadle. Pre správne fungovanie zariadenia je potrebný SIOC software, ktorý je prepojený so simulátorom cez FSUIPC alebo IOCP. Zaria-

Obr. 1.3: Opencockpits MCP 737NG V3H [3]

Obr. 1.4: CPFlight BOEING 737 PRO MCP [4]

denie podporuje simulátory X-Plane, FS2004 a FSX, v ktorých funguje takisto so štandardnými lietadlami Boeing, a modelami od tretích strán ProSim, Project Magenta, iFly, PMDG.[3]

Zariadenie je kvalitnou replikou originál panelu Boeingu 737. Pri tomto panely je škoda absencie selektoru na nastavovanie uhla náklonu, solenoidného spínača na zapnutie automatických otáčok motorov, a takisto chýba podsvietenie. Tento panel sa cenovo pohybuje okolo 475 eur.

1.1.4 CPFlight BOEING 737 PRO MCP

MCP 737 PRO je dokonalá replika skutočného MCP Boeingu 737. Zariadenie je kompatibilné s každým Boeing lietadlom pre Microsoft Flight Simulator vďaka utilite Project Magenta. K správnemu fungovaniu je potrebný dodatkový software, ktorý je dostupný na stránke výrobcu.[4]

Tomuto panelu nie je čo vytknúť. Panel je vo verzii Collins, čiže ide o novšiu verziu. Každý jeden prepínač funguje rovnako ako v skutočnom lietadle. Zaujímavosťou je aj prítomnosť solenoidného páčkového spínača na ovládanie automatických otáčok motorov, ktorý má schopnosť sa za určitých okolností sám prepnúť. Cena tejto skvelej repliky sa vyšplhala na 1218 eur.

1.2 Softvérové riešenie

1.2.1 MobiFlight

S MobiFlightom je možné jednoducho nakonfigurovať moduly bez akejkoľvek znalosti programovania. Vďaka grafickému užívateľskemu rozhraniu je konfigurácia veľmi jednoduchá. MobiFlight podporuje Microsoft Flight Simulator X, Prepar3D a X-Plane.[5]

Výhodou tohto programu je, že aj užívateľ, ktorý nevie programovať, si dokáže jednoducho a lacno postaviť rôzne ovládacie prvky vďaka podrobným návodom na stránkach výrobcu. Užívateľ je však limitovaný mikrokontrolérmi. V dobe písania tejto práce boli podporované len Arduino Mega2560 a Sparkfun Micro Pro. Takisto nie je možné používať akékoľvek súčiastky, čo môže náročnejšiemu užívateľovi prekážať. Na stavbu ovládacieho prvku je možné použiť iba mikrospínače, enkódery, 7 segmentové displeje a LED diódy. Vývojári však v budúcnosti sľubujú podporu ďalších súčiastok ako servomotory a krokové motory.

1.2.2 Link2FS

O čosi jednoduchší program je Link2FS. Ten je možné použiť nielen na letecký simulátor, ale aj na akúkoľvek inú hru. Ovládanie je založené na posielaní klávesových skratiek do vybraného programu, v našom prípade to je akýkoľvek letecký simulátor. K stavbe ovládacieho prvku pomocou tohto programu postačuje Arduino a následne súčiastky, ktoré chce užívateľ používať.[6]

Kapitola 2

Analýza a návrh

Táto kapitola sa zaoberá analýzou a návrhom ako softvérovej, tak aj hardvérovej časti zariadení.

2.1 Požiadavky

Jedným z hlavných cieľov tejto práce je zhotoviť zariadenia, ktoré budú v rámci možnosti, čo najviac podobné, ako vizuálne, tak aj funkčne, skutočným. Práca sa zaoberá MCP a EFIS panelom, ktoré sú vyznačené na obrázku 2.1.

Bude teda potrebné navrhnúť zariadenia, ktoré budú schopné komunikovať

Obr. 2.1: Vyznačený MCP a EFIS panel Boeingu 737 [7]

so simulátorom a meniť nastavenia jednotlivých ovládacích prvkov lietadla. Zároveň bude nutné zaistiť aj spätnú väzbu resp. reakcie simulátora na zmenu nastavení, prípadné rôznych varovaní a pod. Keďže chceme zaistiť aj vizuálnu podobu panelu, bude potrebné pre toto zariadenie vyrobiť aj vhodný obal, do ktorého sa upevní hardware a jednotlivé súčiastky.

Ako už bolo spomenuté v tejto kapitole, na trhu sa v súčasnosti nachádza mnoho ovládacích prvkov určených pre simulátor. Všetky však spája jedna nevýhoda, ktorou je privysoká cena. S tým súvisí aj ďalšia požiadavka, ktorou je pokus o odlíšenie sa od ostatných výrobcov tým, že zariadenie by malo byť lacné na výrobu.

2.2 Výber leteckého simulátora

Neoddeliteľnou súčasťou pre funkčnosť zariadení je letecký simulátor, ktorý budú zariadenia ovládať. Medzi najpoužívanejšie simulátory patrí X-Plane, Prepar3D a Microsoft Flight Simulator X. Všetky z týchto simulátorov sú natívne spustiteľné len v operačnom systéme Windows okrem X-Plane, ktorý je dostupný aj pre macOS a Linux. Napriek tomu však doplnkové modely lietadiel tretích strán sú väčšinou dostupné len pre Windows. Nikoho teda neprekvapí, prečo padol výber platformy práve na Windows.

X-Plane 10 Global je posledná verzia tohto simulátora, ktorá je nepretržite vyvíjana už takmer 20 rokov. Táto nová verzia prináša nový presnejší systém letových vlastností využívajúc virtuálny veterný tunel, nové lietadlá od vetroňov až po kozmickú raketu, presnejšie riadenie letovej prevádzky, detailnejšie prepracované scenérie letísk, presný terén s cestami, riekami a pod. Verzia takisto prináša podporu pre 64 bitové a multijadrové procesory.[8]

Prvá verzia simulátora Prepar3D bola vydaná koncom roka 2010. Simulátor bol a je stále založený na Microsoft ESP, inak povedané ide o Microsoft Flight Simulator X pre komerčné účely. Prvá verzia tohto simulátora bola až na grafické rozhranie takmer totožná s pôvodným simulátorom od Microsoftu. Postupom času však boli opravené jednotlivé chyby a pridané nové funkcie. Hoci simulátor je primárne určený pre komerčné účely, licencia hovorí, že je dostupný pre tých, ktorí trénujú, simulujú alebo sa učia. Z toho vyplýva, že ak chcete simulátor používať aj doma, stačí ho využívať na vzdelávacie účely a nie ako zábavu. [9]

Z vyššie spomenutých je stále najpoužívanejší Microsoft Flight Simulator X. Napriek tomu, že simulátor bol vydaný v roku 2006 popularite vďačí hlavne kvôli neustálemu vývoju rôznych doplnkov, lietadiel a scenérií. Pre tento simulátor takisto existuje aj jeden z najvydarenejších modelov Boeingu 737NG od spoločnosti PMDG.

Tento model obsahuje presnú simuláciu takmer všetkých systémov, ktoré dosiaľ neobsahuje žiaden iný. Model bol vyvíjany viac ako 3 roky, s technickými

informáciami priamo od spoločnosti Boeing a s tímom ľudí, ktorí pracujú s týmto lietadlom aj v skutočnosti.[10]

Pre vývoj zariadení mi prišlo najvhodnejšie zvoliť simulátor od Microsoftu. Pre tento simulátor je dostupné SDK, vďaka ktorému sa ľahšie vyvíjajú softvérove a hardvérove doplnky, má stále dobrú podporu a výborne spracovaný model Boeingu 737. Flight Simulator X je okrem iného aj cenovo najdostupnejší. Kúpiť sa dá už od 25 eur na rozdiel od X-Plane, ktorý stojí dvojnásobok. Prepar3D má cenovku oveľa vyššie. Zariadenia budú primárne vyvíjané pre Flight Simulator X a model lietadla od PMDG.

2.3 Komunikácia so simulátorom

Komunikácia medzi simulátorom a vývojovou doskou bude prebiehať cez sériovú linku. K čítaniu a zapisovaniu polôh jednotlivých prepínačov, rotačných enkóderov a pod. nám poslúži doplnok FSUIPC.

FSUIPC4 je najnovšou verziou doplnku pre Microsoft Flight Simulator X, ktorý poskytuje rozhranie pre ostatné programy na čítanie a zapisovanie dát, všetkých druhov, týkajúcich sa simulácie. Tento doplnok je nevyhnutnou súčasťou mnohých ďalších doplnkov pre simulátor. V skutočnosti ho však nie je potrebné zvlášť kupovať, keďže väčšinou je zakúpený už vývojármi a spolu s doplnkom, ktorý ho využíva, sa nainštaluje aj FSUIPC.[11]

Ovládanie bude fungovať na základe krátkych správ odoslaných cez sériovú linku či už z vývojovej dosky do simulátora, alebo naopak. Tieto správy bude spracovávať skript napísaný v jazyku LUA.

Programovací jazyk LUA patrí do vysokoúrovňových skriptovacích jazykov, do ktorých môžeme zaradiť aj Python, Ruby, či JavaScript. Tieto jazyky ponúkajú vývojárom jednoduchú prácu so štruktúrovanými datami, dynamicky typovanými premennými, automatickou správou pamäti, a mnoho ďalších vysokoúrovňových techník zjednodušujúcich a zrýchľujúcich vývoj.[12]

2.4 Výber súčiastok

2.4.1 Mode Control Panel

Na výrobu MCP bude potrebné použiť 14 mikrospínačov (viď obr. 2.2), ktoré budú slúžiť na aktiváciu jednotlivých funkcií autopilota. Okrem mikrospínačov budú potrebné aj 3 tlačidlové spínače (viď obr. 2.3), jeden na prepínanie jednotiek rýchlosti medzi uzlami a machovým číslom, a ďalšie dva sa používajú na rýchlu zmenu rýchlosti a výšky v FMS.

Na nastavenie kurzu (course), rýchlosti, smeru (heading), výšky a rýchlosti stúpania bude potrebné použiť 6 rotačných enkóderov (viď obr. 2.4). V skutočnosti sa na nastavenie smeru (heading) používa kombinácia rotačného enkódera s rotačným prepínačom. Samotný enkóder sa používa na nastavovanie

smeru (heading) a prepínač na nastavenie uhla nábehu. Táto súčiastka je však na bežnom trhu nedostupná.

Ďalej budú potrebné 4 páčkove spínače (viď obr. 2.5), ktoré budú použité pre aktiváciu resp. deaktiváciu „flight director“, automatických otáčok motorov a jeden na odpojenie autopilota.

Na zobrazenie aktuálnej hodnoty kurzu (course), rýchlosti, smeru (heading), výšky a rýchlosti stúpania budú potrebné 7 segementové displeje. Z dôvodu šetrenia digitálnych vstupov, som sa rozhodol použiť 8 ciferné sedem segmentové LED displeje s radičom MAX7219, ktoré využívaju sériové periferné rozhranie (viď obr. 2.6).

2.4.2 Electronig flight instrument system panel

K funkčnému vyhotoveniu EFIS panelu bude potrebných 7 mikrospínačov (viď obr. 2.2). Tie sa používajú na zapnutie meteorologického radaru a terenného radaru. Pomocou ďalších je možné na navigačnom displeji v lietadle zobrazovať jednotlivé navigačné body, letiská a pod.

Takisto tu sú použité aj tlačidlové spínače (viď obr. 2.3), ktorými je možné na primárnom displeji prepnúť jednotky výšky z stôp na metre a zobraziť vektor letovej trate.

Na zobrazenie naladeného všesmerového majáka resp. nesmerového rádiomajáka slúžia dva páčkove spínače. Spínač má 3 voľby VOR, OFF a ADF, preto bude potrebné použiť spínač typu ON - OFF - ON (viď obr. 2.7).

Podobne ako na MCP aj tu sa nachádzajú dva kusy rotačných enkóderov v kombinácií s rotačným prepínačom, ktoré majú navyše ešte aj mikrospínač. Používajú sa na nastavenie aktuálneho tlaku a výšky rozhodnutia. Ako bolo už vyššie uvedené, túto súčiastku nie je možné zohnať, preto bude nahradená obyčajným rotačným enkóderom s mikrospínačom.

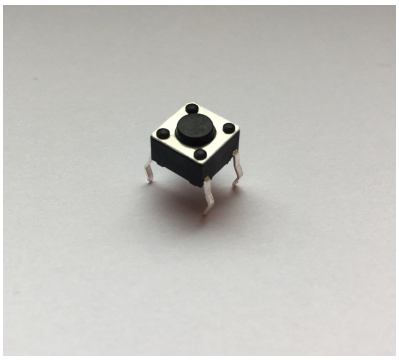
Použité sú tu ešte dva rotačné enkódery s mikrospínačom. Jeden z nich je použitý na zmenu zobrazenia mapy na navigačnom displeji, a druhý ako zoom práve zobrazenej mapy (viď obr. 2.4).

2.4.3 Výber mikrokontroléra

Obidva zariadenia, ktorými sa zaoberá táto práca, sa skladajú z veľkého počtu súčiastok, hlavne MCP. Ten, pre schopnosť ovládať všetky systémy na panely, potrebuje až 46 digitálnych vstupov, z toho 17 nám zaberú len LED diódy signalizujúce práve aktívne sytémy autopilota.

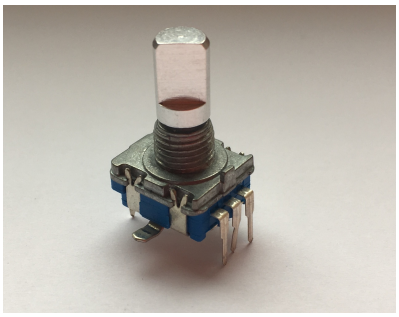
EFIS panel je k vstupom šetrnejší. Tomu bude pre samotné fungovanie postačovať 20 digitálnych vstupov.

V úvahu prichádza použiť mikrokontrolér Arduino Mega2560 (viď obr. 2.8), ktorý je osadený procesorom ATmega2560. Najnovšia verzia tohto mikrontroléra disponuje 256 KB operačnej pamäte, a je teda ideálny pre väčšie

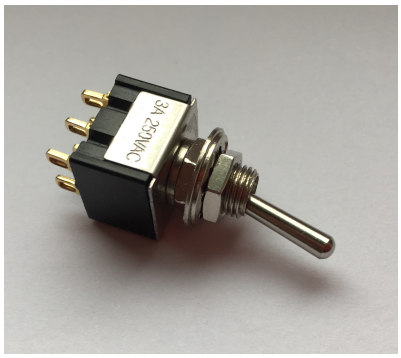


Obr. 2.2: 1 - pólový mikrospínač ON - OFF

Obr. 2.3: 1 - pólový tlačítkový spínač ON - OFF

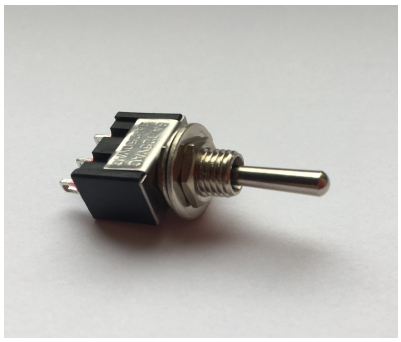


Obr. 2.4: Rotačný enkóder

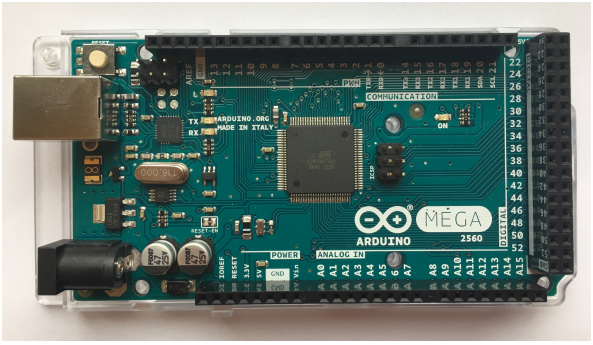


Obr. 2.5: 2 - pólový páčkový spínač ON - ON

Obr. 2.6: 8 ciferný sedem segmentový displej



Obr. 2.7: 1 - pólový páčkový spínač ON - OFF - ON



Obr. 2.8: Mikrokontrolér Arduino Mega2560

Tabuľka 2.1: Výrobné náklady MCP

Typ	Počet ks	Cena s DPH
Arduino Mega	1	5,38 EUR
Mikrospínač	14	1,35 EUR
Tlačidlový spínač	3	1,79 EUR
Rotačný enkóder	6	2,53 EUR
Páčkový spínač	4	1,99 EUR
Displej	6	7,47 EUR
Prepojovacie káble		1,92 EUR
Spolu		22,43 EUR

projekty, v ktorých je potrebný veľký počet vstupov resp. výstupov. Mikrokontrolér dokáže obslúžiť 16 analógových a až 54 digitálnych vstupov. K dispozícií sú 4 sériove porty, a podpora SPI a I2C.[13]

Obidva panely budú mať teda svoj vlastný mikrokontrolér.

2.4.4 Kryt zariadení

Prototypy zariadení budú osadené v kryte, ktoré boli vytlačené na 3D tlačiarni. Rovnako tak boli vytlačené aj nadstavce na rotačné enkódery. Modely týchto veci boli prevzaté zo stránky `http://www.thingiverse.com`. V budúcnosti by však bolo vhodnejšie použiť priehľadné plexisklo, aby bolo možné panel podsvietiť.

2.4.5 Výrobné náklady

V tabuľkách 2.1 a 2.2 sú uvedené ceny z doby nákupu (august 2016). V nákladoch ešte nie sú započítane náklady na výrobu obalu, pretože prototyp bol vytlačený len na 3D tlačiarni, každopádne už teraz je možné usúdiť, že celkové náklady na výrobu budú značne nižšie, než ceny replík, ktoré sú na trhu.

Tabuľka 2.2: Výrobné náklady EFIS panelu

Typ	Počet ks	Cena s DPH
Arduino Mega	1	5,38 EUR
Mikrospínač	7	0,67 EUR
Tlačidlový spínač	2	1,18 EUR
Rotačný enkóder	4	1,68 EUR
Páčkový spínač	2	1,08 EUR
Prepojovacie káble		1,34 EUR
Spolu		11,33 EUR

2.5 Zapojenie súčiastok MCP

2.5.1 Mikrospínače a tlačidlové spínače

Pre toto riešenie bude použitých 17 spínačov, preto ich je rozumné zapojiť do matice 5x4, čím môžeme ušetriť až 8 vstupov. Toto zapojenie potrebuje 9 digitálnych vstupov a bude pripojené k týmto vývodom mikroprocesora:

PE4 Digitálny pin 2

PE5 Digitálny pin 3

PG5 Digitálny pin 4

PE3 Digitálny pin 5

PH3 Digitálny pin 6

PH4 Digitálny pin 7

PH5 Digitálny pin 8

PH6 Digitálny pin 9

PB4 Digitálny pin 10

2.5.2 Rotačné enkódery

Každý enkóder má tri piny. Dva z nich je potrebné zapojiť ako vstup, a jeden je uzemnenie. K zapojeniu bude potrebných 12 vstupov a budú pripojené takto:

PB5 Digitálny pin 11

PB6 Digitálny pin 12

PA0 Digitálny pin 22

PA1 Digitálny pin 23

PA2 Digitálny pin 24

PA3 Digitálny pin 25

PA4 Digitálny pin 26

PA5 Digitálny pin 27

PA6 Digitálny pin 28

PA7 Digitálny pin 29

PC7 Digitálny pin 30

PC6 Digitálny pin 31

GND Uzemnenie

2.5.3 Páčkove spínače

Na tomto panely sú osadené 4 páčkove spínače. Pre tieto spínače budú teda potrebné 4 digitálne vstupy.

PJ1 Digitálny pin 14

PJ0 Digitálny pin 15

PH1 Digitálny pin 16

PH0 Digitálny pin 17

GND Uzemnenie

2.5.4 Displeje

Samotný displej má 5 pinov. Tri vstupy DIN, LOAD, CLK a piny pre napájanie a uzemnenie. Ďalšie displeje sú pripojené sériovo k prvému.

PC2 Digitálny pin 35

PC1 Digitálny pin 36

PC0 Digitálny pin 37

GND Uzemnenie

VCC Napájanie

2.5.5 LED diódy

K signalizácii aktívnych systémov bude potrebných 17 LED diód resp. 17 vstupov, a takisto uzemnenie.

PD6 Digitálny pin 38

PG2 Digitálny pin 39

PG1 Digitálny pin 40

PG0 Digitálny pin 41

PL7 Digitálny pin 42

PL6 Digitálny pin 43

PL5 Digitálny pin 44

PH4 Digitálny pin 45

PL3 Digitálny pin 46

PL2 Digitálny pin 47

PL1 Digitálny pin 48

PL0 Digitálny pin 49

PB3 Digitálny pin 50

PB2 Digitálny pin 51

PB1 Digitálny pin 52

PB0 Digitálny pin 53

PB7 Digitálny pin 13

GND Uzemnenie

2.6 Zapojenie súčiastok EFIS

2.6.1 Mikrospínače a tlačidlové spínače

Na tomto panely budú okrem klasických mikrospínačov a tlačidlových spínačov aj spínače na rotačných enkóderoch. Aj v tomto prípade bude vhodné zapojiť ich do matice. Tentokrát však bude postačovať matica 4x4, kedže spínačov je len 13.

PE4 Digitálny pin 2

PE5 Digitálny pin 3

PG5 Digitálny pin 4

PE3 Digitálny pin 5

PH3 Digitálny pin 6

PH4 Digitálny pin 7

PH5 Digitálny pin 8

PH6 Digitálny pin 9

GND Uzemnenie

2.6.2 Páčkové spínače

Použité páčkove spínače sú typu ON - OFF - ON, preto bude pre jeden spínač potrebné použiť dva vstupy a uzemnenie.

PB4 Digitálny pin 10

PB5 Digitálny pin 11

PB6 Digitálny pin 12

PA0 Digitálny pin 22

GND Uzemnenie

2.6.3 Rotačné enkódery

Rovnako ako na MCP aj tu budú potrebné pre rotačný enkóder dva vstupy a jedno uzmnenie. Rotačných enkóderov je 4, čo zaberie 8 vstupov.

PA1 Digitálny pin 23

PA2 Digitálny pin 24

PA3 Digitálny pin 25

PA4 Digitálny pin 26

PA5 Digitálny pin 27

PA6 Digitálny pin 28

PA7 Digitálny pin 29

PC7 Digitálny pin 30

GND Uzemnenie

Tabuľka 2.3: Prenosové rýchlosti pri sériovom prenose dát [14]

Prenosová rýchlosť	Doba prenosu jedného bitu	Doba prenosu jedného bajtu
1200	833 μs	8,33 ms
2400	416 μs	4,16 ms
4800	208 μs	2,08 ms
9600	104 μs	1,04 ms
19200	52 μs	520 μs
38400	26 μs	260 μs
115200	8,6 μs	86 μs

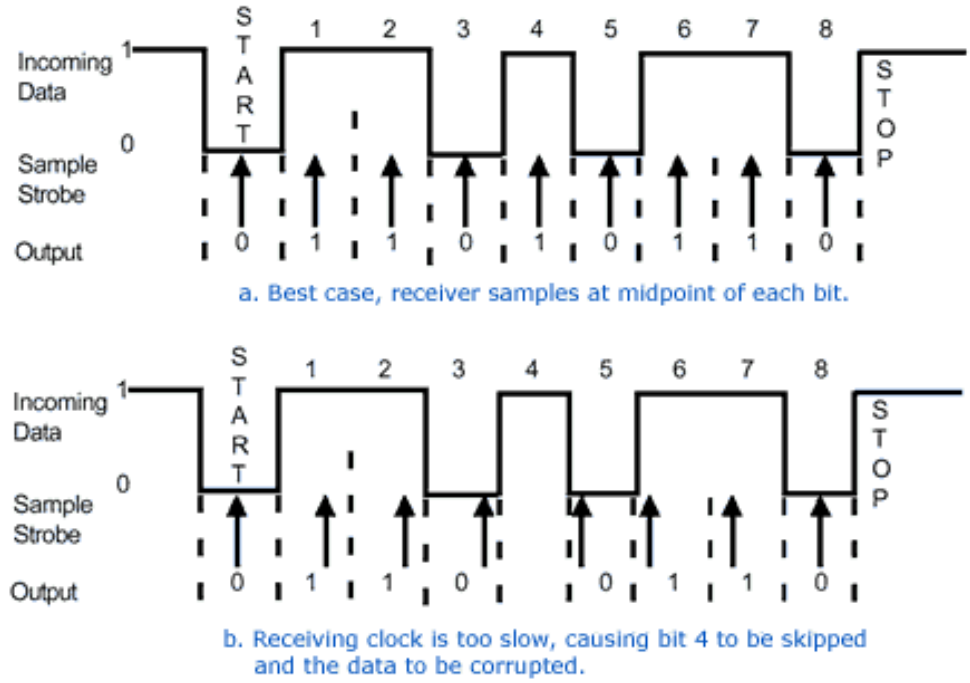
2.7 Sériová komunikácia

Sériový port používa prenosové zariadenie typu UART, asynchronný prenos dát, pri ktorom sú synchronizačné značky vložené na začiatok a koniec väčšieho celku bitov. Najčastejšie sa jedna o 8 bitov, čiže celý bajt. Oba zariadenia, tak ako prijímacia, tak aj vysielacia strana sa najprv musí vhodným spôsobom nakonfigurovať, aby prijímač vedel, v akom formáte má dáta očakávať, a ako rýchlo je nutné vzorkovať dátovú linku, alebo aká ja prenosová rýchlosť. Na obidvoch zariadeniach sa musí najprv nastaviť počet prenášaných bitov v jednom celku, prenosová rýchlosť uvádzaná v bitoch za sekundu, dĺžka stop-bitu a v niektorých prípadoch aj to, akým spôsobom sa prenáša paritný bit. [14]

Datový vodič sa pred prenosom nachádza v kľudovom stave, čo znamená vysokú úroveň napätia. Pred začiatkom odosielania 8 bitov je najprv poslaný takzvaný start bit, ktorý má vždy nulovú hodnotu, čo znamená, že je zaručené, že sa kľudový stav linky vždy zmení. Na strane prijímacieho zariadenia sa musí čo najpresnejšie rozoznať zmena stavu resp. zostupná hrana, ktorá reprezentuje začiatok start bitu. Od tejto chvíle prijímacia strana vie, že sa príjme predom daný počet bitov, s predom nastavenou bitovou rýchlosťou. Na konci prenášanej sekvencie môže byť prenesený aj paritný bit a za ním hneď nasta-leduje stop bit. Ten má vždy hodnotu logickej jednotky, čo znamená vysokú úroveň napätia. [14]

2.7.1 Prenosové rýchlosti

Typické prenosové rýchlosti sériovej linky sú odvodené od násobku 300 bitov za sekundu. Niektoré staršie zariadenia však mali aj nižšie rýchlosti napr. 50, 75, či 150 bitov. V nasledujúcej tabuľke 2.3 sú uvedené prenosové rýchlosti platné pri nastavení prijímača aj vysielača, na prenos ôsmich datových bitov, jedného start bitu a stop bitu s dĺžkou odpovedajúcou dĺžke bežného bitu. Toto nastavenia sa skrátene zapisuje ako 8N1.[14]

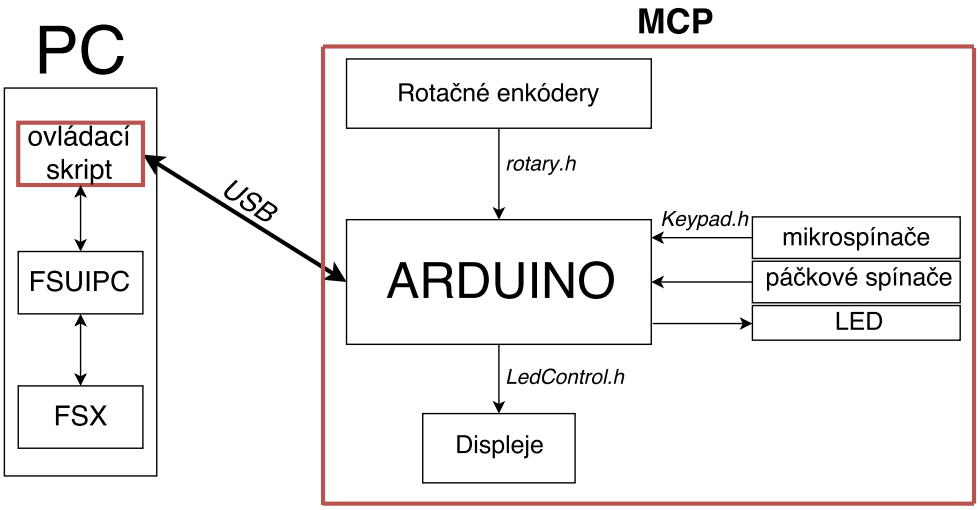


Obr. 2.9: Asynchronný prenos jedného bajtu po sériovej linke. Na hornej časti obrázku je zobrazený ideálny prípad, kedy sa hodnoty jednotlivých bitov vzorkujú presne v ich polovičke, nižšie je potom situácia, ktorá nastane, ak vysielač vysiela dáta rýchlejšie alebo naopak príjmač používa nižšiu vzorkovaciu frekvenciu - na konci prenosu ôsmich bitov už dochádza ku chybe. [14]

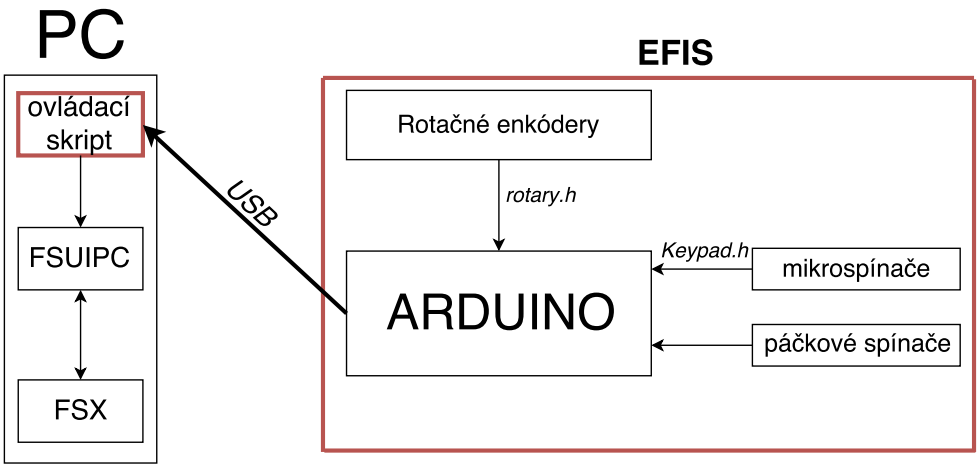
2.8 Riadiace signály

Ako už bolo uvedené, komunikácia medzi mikrokontrolérom a simulátorom bude prebiehať na základe krátkych správ odosielaných cez sériovú linku. Pre jednotlivé spínače sa bude posielať názov práve stlačeného tlačítka, resp. prepnutého spínača. V prípade rotačného enkódera sa bude okrem názvu posielať aj aktuálna hodnota.

Ak bude do riadenia zasiahnuté priamo v simulátore pomocou klávesnice alebo myši, budú správy odoslané zo simulátora vyzerať rovnako. V tomto prípade to však bude mať zmysel len pri zmenách hodnôt na displejoch, ktoré budeme potrebovať zmeniť aj na Arduine. Ak ku konkrétnemu tlačítku prislúcha aj LED dióda, signalizujúca, či je systém aktívny, po stlačení tlačítka odošle Arduino informáciu do simulátora, a spätne nám príde správa, o indikácii, či sa má LED dióda rozsvietiť alebo nie. Toto riešenie je potrebné kvôli tomu, že nie vždy je konkrétny systém autopilota možné aktivovať. Preto so stlačením tlačítka nie je možné považovať systém za aktivovaný resp. de-



Obr. 2.10: Bloková schéma zapojenia MCP



Obr. 2.11: Bloková schéma zapojenia EFIS

aktivovaný.

2.9 Zapojenie

Na obrázku 2.10 a 2.11 sú uvedené blokové schémy zapojenia MCP resp. EFIS panelu. Cieľom tejto práce je implementácia ovládacieho skriptu, a kompletné zhotovenie, vrátane implementácie, MCP resp. EFIS panelu. Uvedené knižnice *rotary.h, Keypad.h* a *LedControl.h*, sú použité k jednoduchšiemu ovládaniu displejov, rotačných enkóderov a mikrospínačov.

Kapitola 3

Realizácia

Tretia kapitola tejto práce sa bude venovať realizácií resp. implementácií samotných panelov. Kapitola bude popisovať ako softvérovu, tak aj hardvérovu časť. Ku každej súčiastke je uvedená schéma zapojenia, popísana implementácia pre Arduino, a takisto príslušná časť skriptu pre FSUIPC.

Softvérová časť pre mikrokontrolér Arduino bola vyvíjaná vo vývojovom prostredí Arduino IDE. Naopak na skript, ktorý je vyvíjaný v skriptovacom jazyku LUA, postačuje aj akýkoľvek textový editor. Na jednotlivé schémy zapojení bol použitý program Fritzing.

3.1 Základný program

3.1.1 Sériová komunikácia

V prvej fáze realizácie bolo potrebné sfunkčniť jednoduchú komunikáciu medzi Arduinom a simulátorom. K riešeniu tohto problému boli použité funkcie pre sériovú komunikáciu zo štandardnej knižnice Arduina. Rýchlosť komunikácie bola nastavená na 115200. Z uvedenej knižnice sú v tomto projekte použité tieto funkcie:

Serial.begin(long speed, config) funkcia nastavuje prenosovú rýchlosť komunikácie, druhým argumentom môžeme nastaviť počet datových bitov, paritu alebo stop bity (štandardne je nastavených 8 dátovych bitov, bez parity a jeden stop bit resp. SERIAL_8N1)

Serial.flush() funkcia čaká, kým sa nedokončí odosielanie dát

Serial.available() funkcia vracia počet bajtov, dostupných na čítanie zo sériového portu

Serial.read() funkcia číta prichádzajúce dáta zo sérioveho portu

Serial.write(char str) funkcia zapisuje binárne dáta na sériovy port

Na strane simulátora bola na sériovú komunikáciu využitá FSUIPC knižnica pre skriptovací jazyk LUA. Pre otvorenie sériovej komunikácie slúži funkcia:

com.open("port", speed, handshake) port napr. "COM1", rýchlosť komunikácie a handshake slúži na riadenie toku dát

Po zahájení komunikácie bola na otestovanie funkčnosti zvolená parkovacia brzdu, zatiaľ len štandardného lietadla. K Arduinu bola pripojená jedna LED dióda, ktorá sa po aktivácií parkovacej brzdy lietadla v simulátore mala rozsvietiť.

3.1.2 Sledovanie stavu ovládacích prvkov

Aby bolo možné pomocou Arduina správne reagovať na zmenu jednotlivých tlačidiel v simulátore, je potrebné vytvoriť funkciu, ktorá bude tieto dáta odosielať na sériovú linku. Na tento problém je vhodné použiť knižnicu Event z FSUIPC.

Táto knižnica umožňuje zostaviť pluginy, ktoré namiesto toho, aby neustále v smyčke kontrolovali stav nejakých vecí, môžu byť naprogramované tak, aby boli nečinné a spustili sa až v momente, kedy dôjde k zmene stavu sledovanej veci. V tomto prípade sú sledované stavy tlačidiel, kontroliek a pod.[15]

V praxi to znamená, že nie je potrebné neustále spúšťať funkciu na kontrolu jednotlivých spínačov, ale funkcia bude automaticky zavolaná, až dôjde k zmene jej stavu. Toto je možné docieliť využitím funkcie **event.offset(offset, "type", "function-name")**. Zavolanie tejto funkcie zavolá uvedenú funkciu, ktorá musí byť definovaná pred týmto riadkom, v momente, keď dôjde k zmene hodnoty špecifikovaného offsetu. Offset je pamäťová adresa, v ktorej je uložená aktuálna hodnota určitého ovládacieho prvku. Typ, je dátový typ hodnoty, ktorá je uložená v pamäti. Offset pre parkovaciu adresu je 0x0BC8 a typ je UB, teda unsigned 8 bit.

Zoznam jednotlivých offsetov a ich dátových typov pre štandardné lietadlá nájdeme v dokumentácií FSUIPC. V prípade lietadiel od tretích strán nájdeme tieto informácie zvyčajne v ich SDK súboroch.

3.1.3 Spustenie prvého programu

Zdrojové kódy 3.1 a 3.2 sú jednoduchou ukážkou komunikácie medzi Arduinom a simulátorom. Pre správnu funkčnosť je potrebné nahrať zdrojový kód do mikrokontroléra a k pinu 10 pripojiť LED diódu. Skript pre FSUIPC je potrebné skopírovať do zložky Modules, ktorý sa nachádza v zložke, kde je nainštalovaný simulátor.

Po spustení simulátora je nutné v nastaveniach FSUIPC povoliť skript a priradiť mu klávesovú skratku, ktorou sa bude spúšťať.

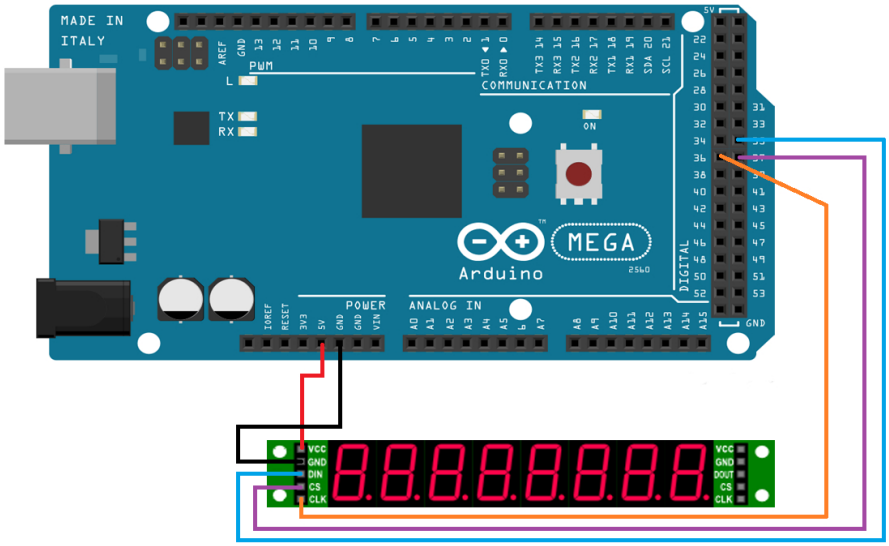
Zmenou stavu parkovacej brzdy, simulátor zavolá funkciu PARKING_BRAKE, a tá podľa stavu pošle príslušný znak na sériovú linku. Arduino zatiaľ čaká, kým sa na sériovej linke neobjavia dáta, ktoré by mohol prijať. Po prijatí znaku sa následne rozhodne, či sa dióda rozsvieti (prijatý znak „a“) alebo nie (prijatý znak „b“).

Zdrojový kód 3.1: Základný program - Arduino

```
1  const int brakeLed = 10;
2  char state;
3
4  void setup() {
5    Serial.begin(115200);
6    pinMode(brakeLed, OUTPUT);
7  }
8
9  void loop() {
10   if (Serial.available() > 0) {
11     state = Serial.read();
12   }
13   if (state == 'a') {
14     digitalWrite(brakeLed, HIGH);
15   }
16   else if (state == 'b') {
17     digitalWrite(brakeLed, LOW);
18   }
19 }
```

Zdrojový kód 3.2: Základný skript - LUA

```
1  dev = com.open("COM8", 115200, 0)
2
3  if dev == 0 then
4      ipc.display("Could not open device port")
5      ipc.exit()
6  end
7
8  function PARKING_BRAKE(offset, value)
9      if value == 0 then
10         com.write(dev, "b")
11     else
12         com.write(dev, "a")
13     end
14 end
15
16 event.offset(0x0BC8, "UW", "PARKING_BRAKE")
```



Obr. 3.1: Zapojenie 7 segmentového displeja

3.2 Zapojenie a implementácia súčiastok - MCP

V tejto časti je popísana softvérová a hardvérová implementácia MCP. Z hľadiska zložitosti je MCP na implementáciu náročnejší, preto budú jednotlivé postupy popísané detailnejšie. V nasledujúcej časti, popisujúcej implementáciu EFIS panelu, budú vyzdvihnuté hlavne rozdiely oproti MCP, keďže samotný postup je veľmi podobný.

3.2.1 Displeje

V tomto projekte sú použité 8 ciferné 7 segmentové displeje, postavené na čipe MAX7219, ktorý displeje ovláda. Ich veľkou výhodou je, že vďaka využívaniu sériového periférneho rozhrania, sú na pripojenie potrebné len 3 digitálne vstupy. Za normálnych okolností by jeden takýto displej potreboval až 16 digitálnych vstupov. Pri použití viacerých displejov je možné ďalšie displeje zapojiť kaskádovo, maximálne však 8, bez nutnosti použitia ďalšieho vstupu.[16]

Pred prvým zapojením displeja bolo potrebné pripajkovať piny k displeju, aby bolo možné displej pripojiť. Následne pripojenie displeja k Arduinu demonštruje obrázok 3.1.

3.2.1.1 Knižnica LedControl

S použitím knižnice LedControl je možné tieto displeje jednoducho ovládať. Najprv je pomocou konštruktoru potrebné vytvoriť objekt typu LedControl.

```
LedControl(int dataPin, int clockPin, int csPin, int numDevices)
```

dataPin pin, ku ktorému je pripojený dátovy pin

clockPin pin, ku ktorému je pripojený clock pin

csPin pin, ku ktorému je pripojený cs pin

numDevices počet kaskádovo zapojených displejov

Po vytvorení objektu, je potrebné inicializovať displej. Inicializácia pozostáva zo zobudenia čipu MAX7219 z úsporného režimu, nastavenia jasu a „vyčistenia“ displeja. Následne je už možné pomocou vhodnej funkcie zobrazovať samotné hodnoty na displeji.

```
void shutdown(int addr, bool b)
```

addr adresa displeja, ktorý chceme ovládať

b true - displej prejde do úsporného režimu, false - displej prejde do normálneho režimu

```
void setIntensity(int addr, int intensity)
```

addr adresa displeja, ktorý chceme ovládať

intensity jas displeja od 0 (najslabší) do 15 (najsilnejší)

```
void clearDisplay(int addr)
```

addr adresa displeja, ktorý chceme ovládať

```
void setDigit(int addr, int digit, byte value, boolean dp);
```

addr adresa displeja, ktorý chceme ovládať

digit pozícia, na ktorú chceme zobraziť hodnotu (0...7)

value hodnota, ktorú chceme zobraziť

dp nastaví desatinnú čiarku

Zdrojový kód 3.3 je ukážkou, ako je na displeji zobrazovaný kurz lietadla. Každá cifra kurzu, ktorý chceme zobraziť, je vypisovaná zvlášť vo for cykle. Pre ostatné hodnoty, ktoré sú zobrazované na displejoch je zdrojový kód obdobný.

Zdrojový kód 3.3: Zobrazenie kurzu na 7 segmentovom displeji

```
#include "LedControl.h"

LedControl lc(35,36,37,1);
void setup(){
  ...
  lc.shutdown(0,false);
  lc.setIntensity(0,10);
  lc.clearDisplay(0);
  ...
}

void loop(){
  ...
}

void setHeadingDisplay(int val){
  for(int i = 0; i<3; i++){
    int dispVal = val % 10;
    val = val/10;
    lc.setDigit(0, i, dispVal, false);
  }
}
```

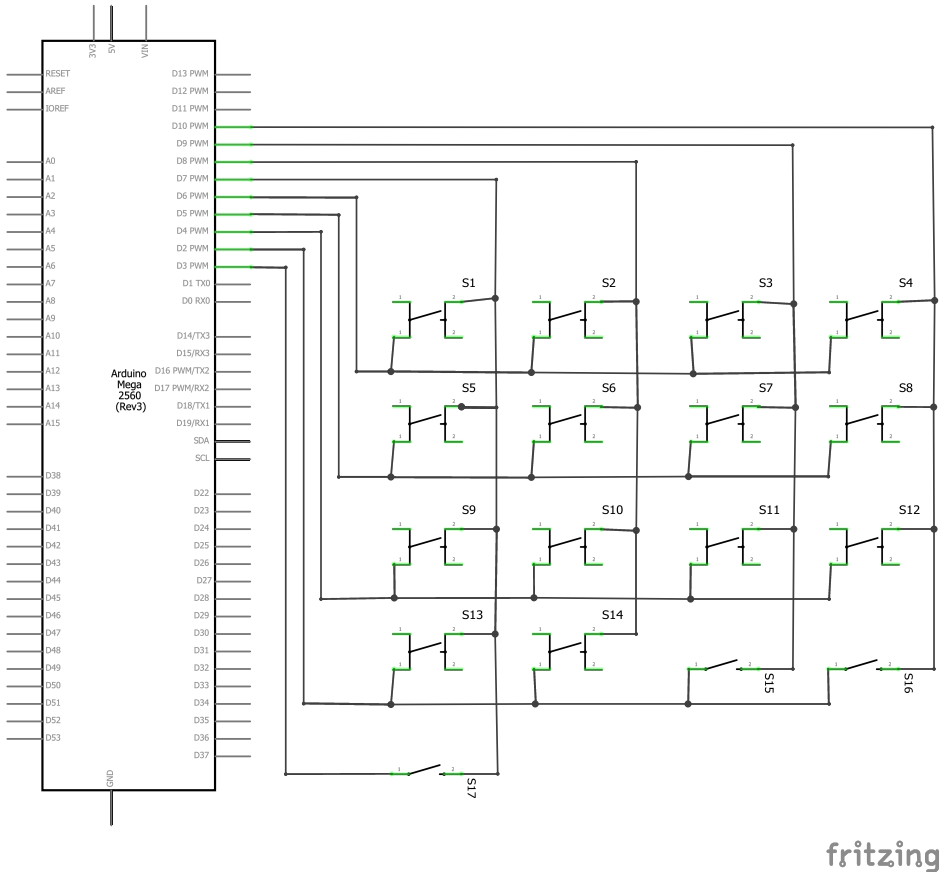
3.2.1.2 Prerušenie

Pre displej zobrazujúci rýchlosť na MCP, bolo potrebné naimplementovať aj funkciu, na ktorú je potrebné využiť prerušenie. V prípade nastavenia rýchlosti, ktorá by bola nad maximálnu konštrukčnú rýchlosť lietadla alebo by zas bola príliš nízka, a atakovala by pádovú rýchlosť, sa vedľa hodnoty rýchlosti rozbliká číslo 8. Prerušenie je vyriešené pomocou pretečenia časovača.

Hodiny na Arduine Mega tikajú rýchlosťou 16MHz. Časovač, ktorý je použitý má 16 bitov. Tento časovač má čítač, ktorý je inkrementovaný každým tiknutím hodín, maximálna hodnota, ktorú môže tento čítač dosiahnuť pri 16 bitoch je teda 65 536. K dosiahnutiu prerušenia je možné využiť aj komparačný register, do ktorého sa vloží konkrétna hodnota, a ak čítač dosiahne danú hodnotu, spustí sa obsluha prerušenia. Za štandarných okolností by k prerušeniu dochádzalo zhruba každé 4ms (65 536/16 000 000), čo je príliš často.

Pre blikanie čísla 8 by bolo ideálne dosiahnuť prerušenie každú sekundu, preto je nutné využiť deličku. Na dosiahnutie požadovaného prerušenia je potrebné použiť deličku 1024. Pri tejto deličke sa bude čítač inkrementovať frekvenciou 15 625Hz. Perióda impulzu je teda 1/15 625 sekúnd, ak požadujeme jednu sekundu potrebujeme 15 625 impulzov. Do porovnávacieho registra je teda potrebné vložiť hodnotu 15 624 (15 625 - 1), keďže sa počíta od nuly.

Toto nastavenie zabezpečí prerušenie každú sekundu, a podľa nastavenia aktuálnej rýchlosti sa rozhodne, či sa osmička na displeji rozbliká alebo nie.



Obr. 3.2: Schéma zapojenia mikrospínačov a tlačidlových spínačov do matice

3.2.2 Mikrospínače a tlačidlové spínače

Na obidvoch paneloch je použitých mnoho tlačidiel. Každé jedno tlačidlo za normálnych okolností potrebuje jeden digitálny vstup. Ako však už bolo uvedené v návrhu riešenia, pre úsporu vstupov, sú tlačidlá zapojené do matice. V prípade MCP ide o maticu 5x4. Na obrázku 3.2 je schéma zapojenia tejto matice pre MCP panel.

3.2.2.1 Knižnica Keypad

Knižnica Keypad umožňuje používanie maticovej klávesnice. Táto knižnica je veľmi šikovná, a preto už nie je potrebné riešiť typický problém tlačidiel, ktorým sú zákmity.

Pomocou konštruktoru sa vytvorí objekt typu Keypad. Jeho prototyp je nasledovný:

```
Keypad(char *userKeymap, byte *row, byte *col, byte numRows, byte
    numCols);
```

userKeymap mapa kláves

row piny, ku ktorým sú pripojené riadky

col piny, ku ktorým sú pripojené stĺpce

numRows počet riadkov

numCols počet stĺpcov

Následne na identifikáciu stlačeného tlačidla je nutné použiť funkciu **char getKey();** , ktorá vráti znak tlačidla definovaného v userKeymap. Zdrojový kód 3.4 popisuje riešenie maticovo zapojených tlačidiel MCP.

Zdrojový kód 3.4: Ukážka riešenia maticovo zapojených tlačidiel

```
1  #include <Keypad.h>
2
3  const byte ROWS = 5;
4  const byte COLS = 4;
5
6  char keys[ROWS][COLS] = {
7    {'1', '2', '3', '4'},
8    {'5', '6', '7', '8'},
9    {'9', 'A', 'B', 'C'},
10   {'D', 'E', 'F', 'G'},
11   {'H', 'I', 'J', 'K'} //I, J, K are not used
12   };
13
14 byte rowPins[ROWS] = {23,22,25,24,27};
15 byte colPins[COLS] = {53,51,49,47};
16 Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS,
       COLS);
17
18 void setup() {
19   ...
20 }
21
22 void loop() {
23   ...
24   readPushButtons();
25   ...
26 }
27
28 void readPushButtons(){
29   char key = keypad.getKey();
30   if(key != NO_KEY){
```

```
31      switch(key){
32        case '1':
33        {
34          //do something
35          break;
36        }
37        case '2':
38        {
39          //do something
40          break;
41        }
42       ...
43    }
44  }
```

3.2.3 Páčkove spínače

Na MCP sa nachádzajú 4 páčkové spínače, ktoré sú k Arduinu pripojené pomocou interného pull-up rezistora. Na čítanie ich stavu postačuje klasická funkcia **digitalRead**, ktorá číta vstup zadaného pinu a vracia 0 resp. 1, v závislosti od aktuálneho stavu spínača.

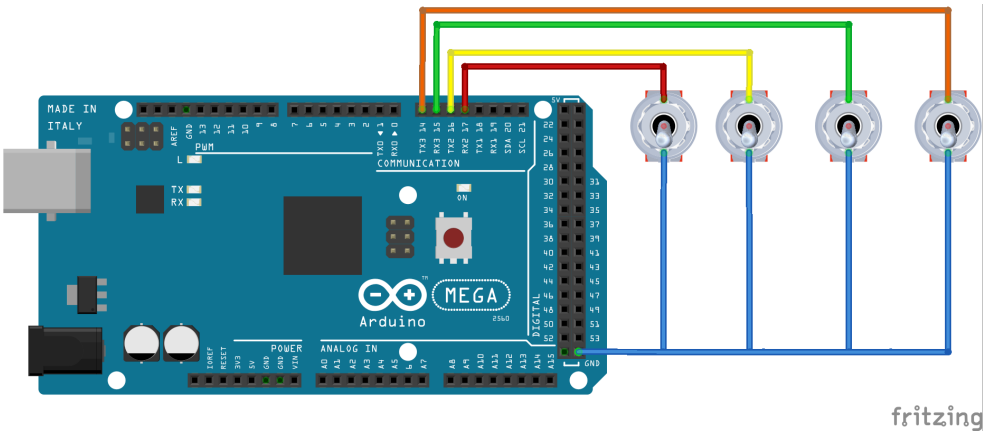
V tomto prípade je potrebné aj dodatočne riešiť zákmity spínačov. Tento problém je možné vyriesiť pomocou funkcie **millis**, ktorý vracia čas v milisekundách ako dlho je program spustený.

Najprv je potrebné prečítať stav spínača, ak je stav iný od predchádzajúceho stavu, uložíme si čas, v ktorom k zmene došlo. Následne odčítame aktuálny čas od času, kedy došlo k zmene stavu spínača. Ak je rozdiel väčší ako 50 milisekúnd (túto hodnotu je možné predĺžiť zmenou premennej debounceDelay), znamená to, že nešlo o falošný signál, ale skutočne došlo k zmene stavu.

Ukážka 3.5 demonštruje riešenie čítania stavu a ošetrenie zákmitov páčkových spínačov. Obrázok 3.3 zobrazuje schému zapojenia týchto spínačov pre MCP.

Zdrojový kód 3.5: Ukážka riešenia páčkových spínačov

```
1  #define AUTO_THROTTLE_SWITCH 52
2  int AT_val = 0;
3  int lastATstate = LOW;
4  unsigned long lastDebounceTime = 0;
5  unsigned long debounceDelay = 50;
6
7  void setup(){
8    pinMode(AUTO_THROTTLE_SWITCH, INPUT_PULLUP);
9    ...
10 }
11
12 void loop(){
13   ...
14   readSwitches();
```



Obr. 3.3: Schéma zapojenia dvojpolohových páčkových spínačov

```
15    ...
16  }
17
18  void readSwitches(){
19    int reading = digitalRead(AUTO_THROTTLE_SWITCH);
20    if(reading != lastATstate){
21      lastDebounceTime = millis();
22    }
23    if((millis() - lastDebounceTime) > debounceDelay){
24      if(reading != AT_val){
25        //do something
26        AT_val = reading;
27      }
28    }
29    lastATstate = reading;
30  }
```

3.2.4 Rotačné enkódery

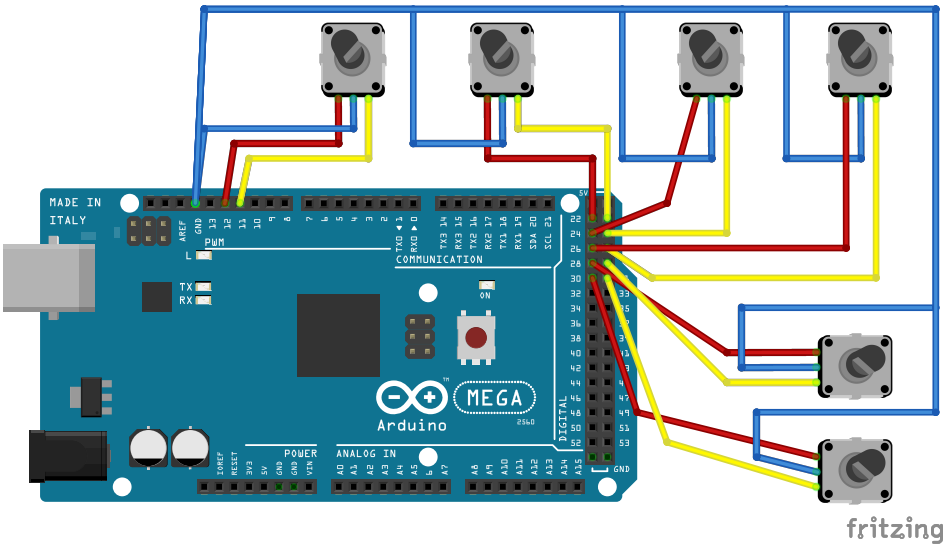
Rotačný enkóder je súčiastka snímajúca polohu, s ktorým je možné otáčať obomi smermi do nekonečna. Vďaka ním môžeme meniť smer lietadla, jeho výšku, rýchlosť a pod. Takisto ako pri obyčajných tlačidlách aj tu dochádza k nežiadúcim zákmitom. V prípade rotačných enkóderov je ošetrenie týchto zákmitov o niečo zložitejšie ako v prípade tlačidiel. Tento problém rieši knižnica Rotary Encoder.

3.2.4.1 Knižnica Rotary Encoder

Táto knižnica rieši samotné čítanie polohy rotačného enkódera a zároveň ošetruje zákmity, ku ktorým môže dochádzať.

Tabuľka 3.1: Zmena výstupného kódu pri pohybe rotačného enkódera [17]

Pozícia	Pin A	Pin B
poloha 0	0	0
1/4	1	0
2/4	1	1
3/4	0	1
poloha 1	0	0



Obr. 3.4: Schéma zapojenia rotačných enkóderov

Každý rotačný enkóder má 3 piny. Jeden spoločný, pin A a pin B. Pri zmene polohy o jednu pozíciu či už doľava, alebo doprava sa generuje špecifická sekvencia výstupného kódu. Pre lepšie pochopenie, tabuľka 3.1 zobrazuje danú sekvenciu kódu, ktorá sa generuje pri zmene polohy o jednu pozíciu. [17]

Z tabuľky 3.1 teda vidíme, že pri pohybe z jednej pozície do druhej, dochádza k zmene výstupného kódu až 4-krát. Na dekódovanie tohto kódu sa využíva jednoduchý stavový automat. Vždy keď dôjde k zmene kódu sa posúvame po jednotlivých stavoch, až kým nedôjde k vykonaniu kompletného pohybu o jednu pozíciu, či už v smere hodinových ručičiek alebo v protismere. Následne daná funkcia vráti smer v ktorom sa rotačný enkóder pohol.

Schéma 3.4 zobrazuje zapojenie rotačných enkóderov použitých pre MCP. Zdrojový kód 3.6 demonštruje prácu s rotačným enkóderom s využitím Rotary Encoder knižnice. Po vytvorení objektu typu Rotary, sa na prečítanie zmeny polohy používa funkcia **process()**. Tá, ako už bolo spomenuté, vracia smer, v ktorom došlo k zmene polohy.

Zdrojový kód 3.6: Ukážka využitia Rotary Encoder knižnice

```
#include "rotary.h"

Rotary heading= Rotary(7,8);

void setup(){
  ...
}

void loop(){
  ...
  readRotaryEncoders();
  ...
}

void readRotaryEncoders(){
  result = heading.process();
    if(result){
      if(result == DIR_CCW){
        //do something (counter clock wise)
      }
      else if(result == DIR_CW){
        //do something (clock wise)
      }
    }
}
```

3.2.4.2 Používanie rotačných enkóderov

Na MCP sa nachádza 6 rotačných enkóderov. Pomocou nich sa nastavuje kurz (course), smer (heading), výška, rýchlosť a rýchlosť vertikálneho stúpania resp. klesania.

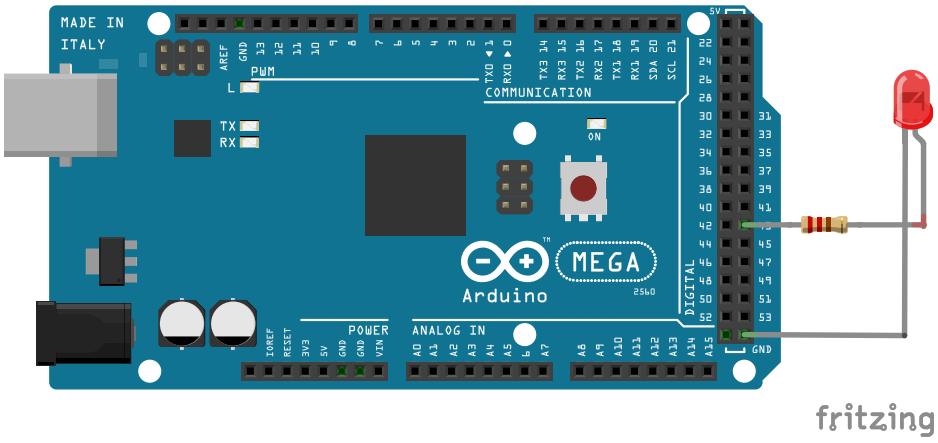
Rotačné enkódery kurz (course) a smer (heading) bolo nutné naimplementovať tak, aby výsledne hodnoty boli v rozmedzí 0 až 359 , keďže sa jedna v podstate o uhly.

Rýchlosť lietadla na MCP je možné nastavovať v dvoch jednotkách. Jednotky sa prepínajú pomocou tlačidlového spínača umiestneného hneď vedľa displeja rýchlosti. Na výber je machovo číslo alebo indikovaná vzdušná rýchlosť. Rozpätie hodnôt pre machovo číslo je 0.60 - 0.82. Indikovanú rýchlosť je možné nastaviť od 100 do 340 uzlov.

Výška sa nastavuje v stopách. Tu je možné nastaviť maximálnu vyšku 50 000 stôp. Rýchlosť stúpania/klesania je uvádzaná vo vystúpaní resp. zostúpaní nastavených stôp za minútu. Nastaviť je možné od -7900 po +6000 stôp za minútu.

3.2.5 LED diódy

LED diódy su ovládané funkciou **ledControl**, ktorej parametre sú konkrétny pin diódy, ktorú chceme ovládať, a stav, či sa má dióda zapnúť, alebo vypnúť.



Obr. 3.5: Schéma zapojenia LED diódy

Funkcia sa zavolá vždy, ak simulátor odošle správu o zmene stavu akejkoľvek z notifikačných diód. V ukážke 3.7 je zobrazené ovládanie jednej LED diódy pomocou tejto funkcie. Na obrázku 3.5 je znázornené zapojenie diódy.

Zdrojový kód 3.7: Ovládanie LED diód

```
#define CMD_A_BUTTON 43
...

void setup() {
    pinMode(CMD_A_BUTTON, OUTPUT);
    ...
}

void loop() {
    ...
    ledControl(CMD_A_BUTTON, 1); //turn on LED
    ...
}

void ledControl(int pin, int valL){
    if(valL == 1){
        digitalWrite(pin, HIGH);
    }
    else if(valL == 0){
        digitalWrite(pin, LOW);
    }
}
```

3.2.6 Odosielanie zmeny stavu

Po zapojení a implementácií všetkých súčiastok, je potrebné ich stav odosielať. Informácia sa na sériovú linku odošle hneď po zaregistrovaní zmeny stavu tým, že sa zavolá funkcia **writeOffset** (viď zdrojový kód 3.8). Jej parametre sú názov ovládacieho prvku, a zároveň hodnota. Následne sa na sériovú linku odošle správa v tvare „názov:hodnota“. Správa je zakončená znakom nového riadku.

Zdrojový kód 3.8: Funkcia odosielajúca stav danej súčiastky

```
1 void writeOffset(const char* controlName, int value){
2   char sendData[7];
3   sprintf(sendData, "%s:%d\n", controlName, value);
4   Serial.write(sendData);
5   Serial.flush();
6 }
```

V simulátore je potrebné čítať sériovú linku. Aj tu je nápomocná funkcia z knižnice Event. Nie je preto potrebné v smyčke čakať na dáta, ale akonáhle budú k dispozícií, funkcia **event.com(handle, term, "function-name")**, nám automaticky zavolá uvedenú funkciu na prijatie správy. Parameter term, je oddeľovač. To znamená, že uvedenej funkcii sa pošle ako parameter prijatý reťazec, po zadaný oddelovač.

Zdrojový kód 3.9: Ukážka časti skriptu, ktorý spracúvava prijatý reťazec

```
1  function readSerial(handle, string)
2   serial_string = serial_string .. string
3   if string.find(serial_string, '\n') ~= nil then
4    state = parseValue(serial_string)
5    controlName = parseControlName(serial_string)
6    setControl(controlName, state)
7    serial_string = ""
8    end
9  end
10
11 function parseValue(forParse)
12  index = string.find(forParse, ":")
13  value = forParse.sub(forParse, index+1)
14  return value
15 end
16
17 function parseControlName(forParse)
18  index = string.find(forParse, ":")
19  controlName = forParse.sub(forParse, 1, index-1)
20  return controlName
21 end
22
23 function setControl(controlName, state)
24   if controlName == "alt" then
25    ipc.control(84137, state*100)
26    flag_alt = 1
```

```
27    return
28    ...
29  end
30
31  function ALTITUDE(offset, value)
32    if flag_alt ~= 1 then
33     ipc.log("WRITE - ALT:" .. value/100)
34    end
35    flag_alt = 0
36  end
37
38  event.offset(0x652E, "UW", "ALTITUDE")
39  event.com(dev, 6, '\n', "readSerial")
```

V okamžiku, kedy sú dostupné dáta na sériovej linke sa zavolá funkcia **readSerial**, ktorá overí, či je správa zakončená novým riadkom. V prípade, že je podmienka splnená, zavolajú sa funkcie na spracovanie reťazca. V opačnom prípade sa čaká, kým nie je doručená aj zvyšná časť. K tomuto prípadu by ale nemalo dochádzať.

Funkcie **parseValue** a **parseControlName** parsujú z prijatého reťazca názov ovládacieho prvku a hodnotu. Následne je zavolaná funkcia **setControl**, ktorá podľa názvu nastaví konkrétny ovládací prvok. V ukážke kódu 3.9 je uvedené nastavenie výšky na MCP už pre lietadlo Boeing 737 od PMDG. Funkcia **ipc.control** požaduje parametre kód ovládacieho prvku a hodnotu. Kód pre výšku je 84134. Samotná výška sa nastavuje v stovkách, preto je premenná state ešte vynásobena 100. Ostatné ovládacie prvky sa nastavujú obdobne. Zoznam jednotlivých kódov nájdeme v SDK súboroch PMDG.

V zdrojovom kóde 3.9 je možné si všimnúť aj pomocnú premennú flag_alt. Pri zmene výšky, je opäť možné pomocou funkcie **event.offset** zavolať nejakú inú funkciu (v prípade zmeny výšky sa volá funkcia ALTITUDE). Výška, rovnako aj iné hodnoty na 7 segmentovom displeji, sa kvôli rýchlosti, nastavujú priamo rotačným enkóderom najprv na displeji, a definitívna hodnota sa odošle do simulátora. V prípade zmeny hodnoty priamo v simulátore, by sa na displeji zobrazovala nesprávna hodnota. Práve pre tento prípad je vhodné využiť túto funkciu. Ak sa zmení hodnota priamo v simulátore, zavolá sa funkcia **ALTITUDE**, premenná flag_alt je rovná 1, a tým pádom je podmienka splnená. Následne sa odošle správa pre Arduino s aktuálnou hodnotou. Bez tejto premennej by nebolo možné rozlíšiť, či došlo k zmene hodnoty priamo v simulátore alebo nie (keďže k volaniu tejto funkcie dochádza vždy pri zmene hodnoty, či už otočením rotačného enkódera alebo priamo v simulátor), a dochádzalo by k odosielaniu aktuálnej hodnoty aj v prípade, ak by sa hodnota zmenila pomocou rotačného enkóderu. Toto by však bolo zbytočné, pretože by sa tá istá hodnota zapisovala na displej dvakrát.

3.3 Zapojenie a implementácia súčiastok - EFIS

Za hlavný rozdiel v implementácii tohto panelu oproti MCP považujem to, že tu nie je potrebné zabezpečiť žiadnu spätnú interakciu zo strany simulátora. To znamená, že komunikácia prebieha len jedným smerom, a to z Arduina do simulátora. Dôvodom je, že na tomto panely sa nenachádzajú žiadne LED diódy, displeje a pod., ktoré môžu meniť svoj stav nezávislé od samotného ovládania.

3.3.1 Mikrospínače

Na EFIS panel je potrebné použiť 14 mikrospínačov, a preto nám v tomto prípade postačuje matica 4x4. Na rozdiel od MCP sa tu používajú aj mikrospínače na rotačných enkóderoch. Implementácia je rovnaká ako na ukážke 3.4. Schéma pre zapojenie týchto spínačov je takisto založená na podobnom princípe ako schéma 3.2.

3.3.2 Páčkové spínače

Na prepínanie zobrazovania naladených VOR alebo ADF majákov sú použité dva trojpolohové spínače. Implementácia v Arduine je rovnaká ako v zdrojovom kóde 3.5. V skripte pre FSUIPC je však potrebné ukladať do pomocnej premennej aktuálnu polohu spínača. Na obrázku 3.6 je zobrazená schéma zapojenia týchto trojpolohových spínačov.

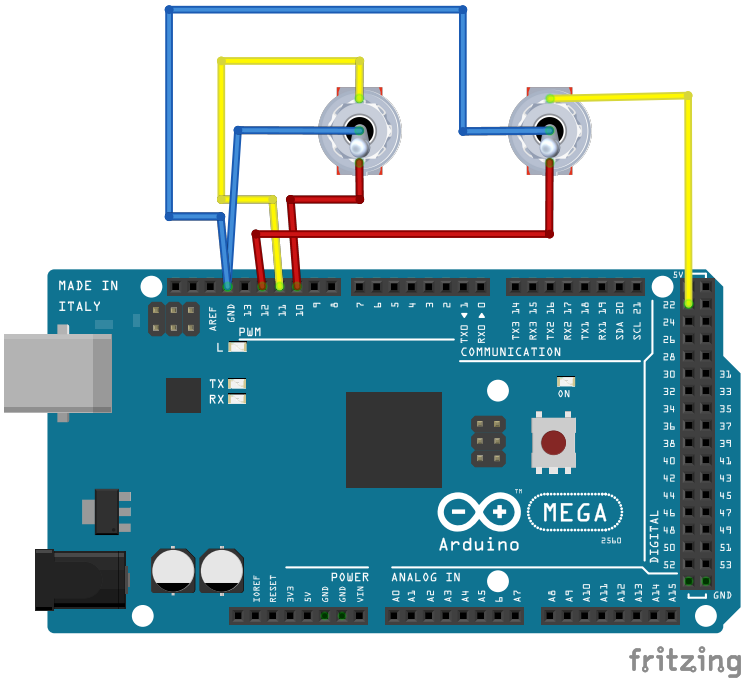
3.3.3 Rotačné enkódery

Ako už bolo spomenuté, v skutočnosti sa na tomto panely nachádzajú dva rotačné enkódery v kombináci s rotačným prepínačom. Nahradené sú obyčajnými rotačnými enkódermi. Schéma zapojenia je až na použité piny rovnaká ako na obrázku 3.4.

V prípade implementácie je tu oproti MCP menší rozdiel, keďže nepotrebujeme žiadny čítač, iba rozoznávať smer otočenia. Pre systémy ovládajúce tieto rotačné enkódery, nie je možné nastaviť priamo konkrétnu hodnotu ako v prípade napr. výšky na MCP. Tu je možné hodnotu iba inkrementovať resp. dekrementovať. Ukážka 3.10 popisuje spôsob implementácie, ktorý je pre všetky rotačné enkódery na EFIS panely rovnaký.

Zdrojový kód 3.10: Ukážka implementácie rotačných enkóderov pre EFIS panel

```
1  ...
2  //reading barometric rotary encoder
3   char result = baro.process();
4   if(result){
5     if(result == DIR_CCW){
6         writeOffset("BAR", 0);
```



Obr. 3.6: Schéma zapojenia trojpolohových páčkových spínačov

```
7          }
8        else if(result == DIR_CW){
9          writeOffset("BAR", 1);
10       }
11     }
12     ...
```

3.4 Rozdiely medzi štandardnými a doplnkovými lietadlami

Vývoj ovládacích zariadní pre štandardné lietadla je o čosi jednoduchší. Pre každý jeden ovládací prvok existuje offset, ktorý je možné prečítať, a zároveň aj prepísať. Ich zoznam je možné nájsť v dokumentácií FSUIPC, keďže sú pre všetky štandardné lietadla rovnaké. Offset teda môžeme prečítať funkciou **ipc.readXX(offset)**, kde XX je dátový typ hodnoty a následne prepísať konkrétnou hodnotu pomocou funkcie **ipc.writeXX(offset, value)**.

Prvou nevýhodou lietadiel od tretích strán sú rôzne offsety. Vytvorený skript pre FSUIPC teda nie je univerzálny, a nebude fungovať s každým lietadlom dostupným pre simulátor. Pre správne fungovanie bude potrebná úprava tohto skriptu, ktorá však nie je zložitá.

Tabuľka 3.2: Zoznam kódov reprezentujúcich úkony myšou

Typ	Hex	Dec
kliknutie pravým tlačidlom	0x80000000	2147483648
kliknutie ľavým tlačidlom	0x20000000	536870912
koliesko nahor	0x00004000	16384
koliesko nadol	0x00002000	8192

Takisto treba podotknúť, že pri doplnkových lietadlách nemajú všetky ovládacie prvky svoje offsety, a preto aktuálny stav niektorých prvkov nevieme prečítať. Hodnoty offsetov pri niektorých lietadlách tiež nie je možné prepísať a sú iba určené na čítanie. Preto je pri zmene ovládacieho prvku nutné použiť funkciu **ipc.control(n, value)**, ktorá simuluje ovládanie myšou. Parameter n je kód ovládacieho prvku (tie nájdeme v SDK súboroch), a value je kód reprezentujúci dvojité alebo obyčajné kliknutie ľavým resp. pravým tlačidlom, pohyb kolieskom na myši a pod. Jednotlivé kódy možných úkonov sú takisto uvedené v dokumentácií. Použité kódy v tomto projekte, sú uvedené v tabuľke 3.2.

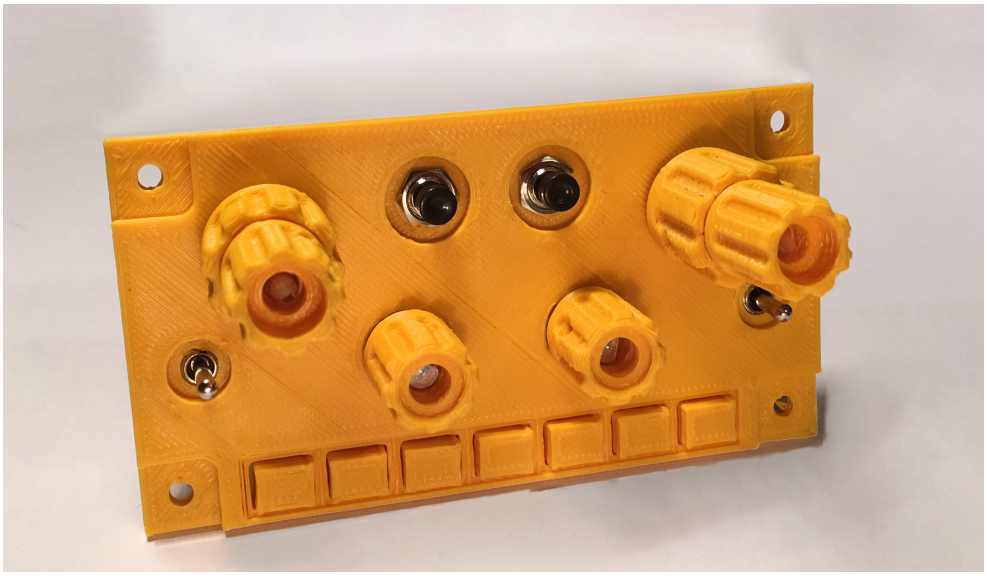
V prípade zmeny hodnoty rýchlosti, rýchlosti stúpania, výšky, kurzu (course) a smeru (heading), je možné ako parameter zvoliť priamo hodnotu, ktorú chceme nastaviť.

3.5 Problémy počas implementácie

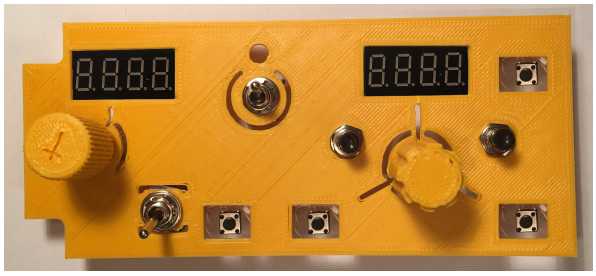
Pri implementácii panelov som narazil na problém, ktorý spočíval v nesprávnom prijímaní reťazcov na strane simulátora. Chyba je konkrétne v utilite FSUIPC, a problém je v štádiu riešenia vývojármi.

Pri prijímaní riadiacich signálov odoslaných z Arduina, nedošlo vždy k prijatiu celého reťazca, ale len jeho časti (napr. namiesto alt:3000, bol prijatý reťazec alt:3). Zvyšná časť bola prijatá spolu s ďalším reťazcom resp. pri ďalšom stlačení tlačidla, prepnutia spínača a pod., čo mohlo teoreticky nastať aj o minútu neskôr (chýbajúce nuly z predchádzajúceho príkladu s novým reťazcom 000alt:3100). Týmto problémom dochádzalo teda k prípadom, kedy simulátor na jednotlivé príkazy z Arduina vôbec nereagoval, keďže ich nevedel rozpoznať.

Tento problém som konzultoval s vývojárom, ktorý mi predbežne poradil nastaviť a dodržiavať fixnú dĺžku prijímaných reťazcov vo funkcii **event.com**, a zároveň nastaviť aj oddeľovač. Dĺžka je nastavená na 6 znakov a ako oddeľovač je používaný nový riadok resp. znak \n (viď riadok 39 v zdrojovom kóde 3.9).



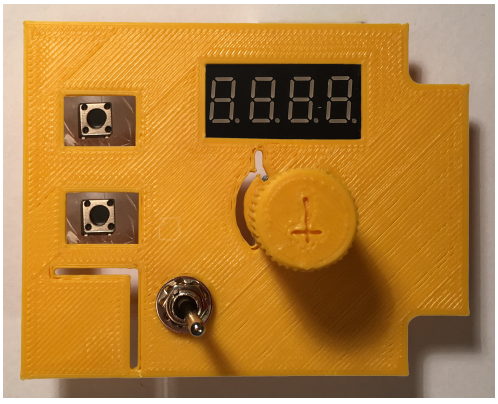
Obr. 3.7: Vyrobený EFIS panel



Obr. 3.8: Vyrobené MCP 1/4

Obr. 3.9: Vyrobené MCP 2/4

Obr. 3.10: Vyrobené MCP 3/4



Obr. 3.11: Vyrobené MCP 4/4

Kapitola 4

Testovanie

4.1 Softvérová sériová linka

Počas samotného testovania, či na Arduine všetko funguje správne, a zároveň, či sa odosielajú správne riadiace signály, boli využité výpisy odosielané na softvérovú sériovú linku.

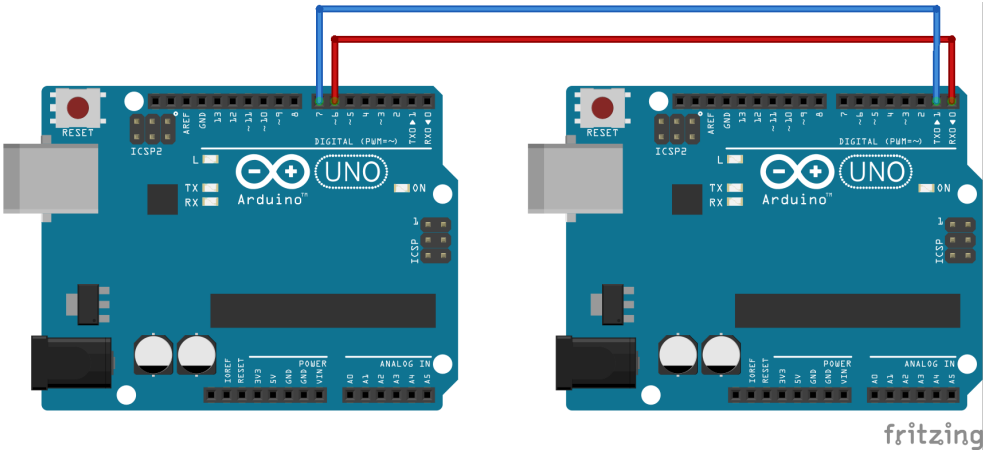
Spočiatku prebiehal vývoj na Arduine Uno, ktoré má len jeden sériový port, a ten bol už použitý na odosielanie dát pre simulátor. V takomto prípade je výhodné použiť knižnicu SoftwareSerial, ktorá umožňuje nasimulovať sériovú komunikáciu na dva zvolené digitálne piny. Na toto riešenie však budeme potrebovať ďalšie Arduino, ktoré bude dáta prijímať. Do Arduina, z ktorého chceme dáta odosielať je potrebné nahrať kód, v ktorom bude najprv vytvorený objekt typu SoftwareSerial s pinmi, ktoré chceme na túto komunikáciu používať, a potom klasicky komunikáciu povoliť pomocou funkcie **begin()**. Do druhého Arduina postačuje nahrať základný zdrojový kód bez akýchkoľvek konfigurácií a spustiť nejaký program na čítanie sériovej linky.

Zdrojový kód 4.1 zobrazuje základný princíp konfigurácie a odoslanie správy cez softvérovú sériovú linku. Potrebné prepojenie Arduin je možné vidieť na obrázku 4.1.

Zdrojový kód 4.1: Ukážka konfigurácie softvérovej sériovej linky

```
1  SoftwareSerial mySerial(6,7); // RX, TX
2
3  void setup() {
4    mySerial.begin(4800);
5    mySerial.println("Hello world");
6  }
7
8  void loop() {
9    ...
10 }
```

Využitie softvérovej sériovej linky je uľahčujúce aj v prípade používania Arduino Mega, ktoré disponuje viacerými sériovymi portami. Pri častom pre-



Obr. 4.1: Prepojenie Arduin na využitie softvérovej sériovej linky

hrávaní kódu do Arduina je trochu otravné neustále vypínať a znova zapínať program, ktorý otvára sériovú linku, aby sme mohli jednotlivé testovacie výpisy sledovať.

S využitím tejto knižnice, bola otestovaná funkčnosť všetkých pripojených periférií, a zároveň odosielanie správnych dát o zmene stavov.

V simulátore je možné vypisovať dáta cez skript pomocou funkcie **ipc.log("string")**, ktoré sa vypisujú v konzole FSUIPC. Pomocou tejto funkcie sme takisto otestovali skript, či v prípade zmeny nastavení priamo v simulátore sa odošlú správne hodnoty pre Arduino.

4.2 Test zariadení

Na test samotných zariadení som použil dve modelové situácie. V prvom prípade to bol krátky let zameraný na častejšie nastavovanie jednotlivých ovládacích prvkov. V tom druhom prípade som testoval zariadenie na spoľahlivú funkčnosť aj počas dlhších letov.

4.2.1 Short-haul let

Let prebiehal medzi letiskom M. R. Štefánika v Bratislave a letiskom v Košiciach, s využitím VOR to VOR navigácie. Na tejto letovej trase sú 3 všesmerové majáky, ktoré boli využité k navigácií.

Pri využívaní tohto typu navigácie je potrebné prelaďovať medzi jednotlivými majákmi a podľa mapy správne nastaviť radiálu. Požadovanú radiálu nastavíme pomocou rotačného enkódera kurzu (course) na MCP. Zároveň je potrebné aktivovať systém VOR LOC.

Pred samotným vzletom sa na MCP nastavuje kurz odletovej dráhy. V tomto prípade bol pre dráhu 04 v Bratislave kurz 039, ktorý bol nastavený pomo-

cou rotačného enkódera smeru (heading). Rýchlosť bola nastavená podľa vypočítanej rýchlosti rozhodnutia, ku ktorej pripočítame 15 uzlov. Zároveň je potrebné aktivovať automatické otáčky motorov pomocou páčkoveho spínača A/T ARM. Podľa prideleného počiatočného stúpania od riadiaceho letovej prevádzky, bola výška nastavená na 5000 stôp. Nastavená bola takisto aj radiála 079, podľa ktorej som sa po vzlete orientoval a napokon zapol „flight director“ na strane kapitána. V závislosti od aktuálneho počasia bol na EFIS panely nastavený tlak ovzdušia.

Po vzlete bol aktivovaný autopilot pomocou tlačidla A/P, zapnutie udržovanie nastavenej rýchlosti, výšky, a samozrejme systém VOR LOC na navigáciu podľa všesmerových majákov.

Počas priblíženia lietadla k dráhe 01 Košiciach, bol využitý systém ILS, ktorý po naladení správnej frekvencie, udržuje lietadlo správnym smerom a správnym uhlom klesania až po samotnú dráhu. Tento systém aktivujeme na MCP stlačením tlačidla APP.

4.2.2 Long-haul let

Pri tomto lete som takisto odlietal z Bratislavy a letel som do New Yorku. Počas tohto letu som kvôli jeho dĺžke využil modernejšiu a pohodlnejšiu navigáciu RNAV. V FMS sa podľa naplánovanej trasy nastavia jednotlivé body ktorými má lietadlo preletieť.

Počiatočné nastavenia pred letom boli podobné ako v prípade short-haul letu. Po vzlete som aktivoval autopilota, horizontálnu navigáciu (tlačidlo LNAV), a takisto vertikálnu navigáciu (tlačidlo VNAV). O zvyšok sa postaral samotný FMS systém a autopilot, ktorý podľa vložených parametrov volil rýchlosť lietadla, rýchlosť stúpania a klesania a pod., tak, aby bol let, čo najviac ekonomický.

4.2.3 Výsledok testovania

Oba testy dopadli dobre. V priebehu oboch letov, som počas testovania nezaznamenal s jednotlivými zariadeniami žiadne problémy.

Kapitola 5

Budúce práce

Vývoj jednotlivých ovládacích prvkov pre simulátor ma veľmi zaujal. V budúcnosti by som primárne chcel vylepšiť súčasne zariadenia. Po vyriešení týchto vecí by som sa určite chcel venovať aj výrobe ďalších panelov a postupne aj stavbe celého kokpitu.

5.1 Úprava súčasných zariadení

Hlavnou prioritou je úprava predného panelu resp. samotného krytu. Súčasný prototyp je vytlačený na 3D tlačiarni bez popisov. Túto vec by som chcel vylepšiť. Predný panel by mohol byť vyrobený z priehľadného plexiskla, na ktorý by sa dali vygravírovať jednotlivé popisky, a zároveň aj podsvietiť. Súčasné mikrospínače by som takisto chcel nahradiť tlačidlami s integrovanou LED diódou, vďaka čomu by bolo viditelné, ktorý systém je momentálne aktívny.

5.2 Modularita

V prípade vývoja ďalších zariadení by bol potrebný pre každé Arduino jeden USB port, jeden sériový port, a takisto samostatný skript pre každý panel, čo v prípade viacerých panelov nie je najideálnejšie riešenie.

Tento problém by mohla riešiť modularita jednotlivých panelov. Hlavné Arduino by komunikovalo cez sériovú linku so simulátorom, a k nemu by sa pripájali ďalšie. Komunikácia medzi jednotlivými Arduinami by mohla prebiehať pomocou sériovej zbernice I^2C.

Týmto riešením by sme značne ušetrili niekoľko USB portov, ktoré by v prípade väčšieho projektu nemuseli postačovať.

5.3 Súčiastky

Najťažším problémom však znova bude dostupnosť jednotlivých súčiastok. Napríklad na hornom panely lietadla sa nachádzajú veľmi atypické súčiastky v podobe trojitých rotačných enkóderov a pod., ktoré na trhu nie sú vôbec dostupné.

Záver

V tejto práci som sa zaoberal analýzou, návrhom a nakoniec samotnou implementáciou ovládacích prvkov pre Microsoft Flight Simulator X. Výsledkom tejto práce su funkčné prototypy MCP a EFIS panelu, určených pre ovládanie Boeingu 737, postavené na mikrokontroléri Arduino.

Komunikácia medzi Arduinom a leteckým simulátorom prebieha vďaka utilite FSUIPC cez sériovú linku. Pri každej zmene stavu jednotlivých ovládacích prvkov sa odošle krátky reťazec, na základe ktorého dochádza k zmene nastavenia v simulátore.

Obe zariadenia je možné postaviť za cenu nepresahujúcu 40 eur, čo je oproti terajším dostupným riešeniam 10-násobne menej.

Cieľ tejto práce bol splnený, čo poukazujú aj výsledky testovania, počas ktorého nedošlo k odhaleniu žiadných problémov. Postavené zariadenia sú teda plne funkčné a pripravené na používanie.

Literatúra

[1] Saitek. *Pro Flight^{TM} Multi Panel for PC [online]*, [cit. 2016-10-07]. Dostupné z: http://www.saitek.com/uk/prod/multi.html

[2] VRinsight Co.,Ltd. *Welcome to VRinsight Shop [online]*, 2014, [cit. 2016-10-07]. Dostupné z: http://www.vrinsightshop.com/shop/step1.php?number=3

[3] Opencockpits S.L. *MCP 737NG V3H [online]*, 2016, [cit. 2016-10-08]. Dostupné z: http://www.opencockpits.com/catalog/mcp-737ng-v3h-p-486.html

[4] CPflight S.r.l. *MCP cpflight [online]*, október 2016, [cit. 2016-10-08]. Dostupné z: http://www.cpflight.com/sito/dettagli/mcp737pro.asp

[5] Möbius IT-Dienstleistungen. *MobiFlight + FSX + Arduino + FSUIPC = Your Homecockpit! [online]*, 2016, [cit. 2016-10-13]. Dostupné z: http://www.mobiflight.com

[6] JimsPage. *FSX arduino mega arduino duemilanove flight simulator ARDUINO KEYS [online]*, 2012, [cit. 2016-10-13]. Dostupné z: http://www.jimspage.co.nz/arduino_keys_beta.htm

[7] Planes.cz. *OY-MRG - B737-7L9 - Praha - Ruzyně (PRG/LKPR) - planes.cz*, máj 2011, [cit. 2016-11-23], úprava obrázka vyznačením EFIS a MCP. Dostupné z: http://www.planes.cz/cs/photo/1107445/b737-7l9-oy-mrg-cimber-sterling-cim-qi-praha-ruzyne-prg-lkpr

[8] Valve Corporation. *X-Plane 10 Global - 64 Bit on Steam [online]*, 2016, [cit. 2016-10-09]. Dostupné z: http://store.steampowered.com/app/292180/

[9] simFlight Network. *Taking a look at Lockheed Martin Prepar3D V3 [online]*, 2016, [cit. 2016-10-09]. Dostupné z: http:

//www.simflight.com/2016/01/02/taking-a-look-at-lockheed-martin-prepar3d-version-3-part-1/

[10] Precision Manuals Development Group. *PMDG 737NGX [online]*, [cit. 2016-10-09]. Dostupné z: https://www.precisionmanuals.com/pages/product/FSX/ngx8900.html

[11] simFlight GmbH. *PETE DOWSON - FSUIPC4 [online]*, 2016, [cit. 2016-10-09]. Dostupné z: http://secure.simmarket.com/pete-dowson-fsuipc4.phtml

[12] Root.cz. *Programovací jazyk LUA*, Marec 2009, [cit. 2016-11-13]. Dostupné z: https://www.root.cz/clanky/programovaci-jazyk-lua/

[13] EVANS Martin, J. N. a. J. H.: *Arduino in action*. New York, USA: Manning Publications, Máj 2013, ISBN 978-1-617290-24-4, str. 368.

[14] Root.cz. *Sériovy port - RS232C [online]*, November 2008, [cit. 2016-10-14]. Dostupné z: https://www.root.cz/clanky/seriovy-port-rs-232c/

[15] Dowson, P.: FSUIPC: Lua Library Reference. Marec 2010, [cit. 2016-11-07]. Dostupné z: http://www.cityviewed.co.uk/images/uploads/directory/fsuip/FSUIPC_Lua_Library.pdf

[16] Embeded-Lab. *INTRODUCING A NEW SERIAL (SPI) 8-DIGIT SEVEN SEGMENT LED DISPLAY MODULE USING MAX7219*, 2016, [cit. 2016-11-02]. Dostupné z: http://embedded-lab.com/blog/introducing-a-new-serial-spi-8-digit-seven-segment-led-display-module-using-max7219/

[17] GitHub, Inc. *Rotary Encoder Arduino Library*, 2016, [cit. 2016-11-15]. Dostupné z: https://github.com/brianlow/Rotary

Dodatok **A**

Zoznam použitých skratiek

MCP Mode Control Panel

EFIS Electronic flight instrument system

COM Communication

SDK Software Development Kit

FMS Flight management system

LED Light-Emitting Diode

VOR Omnidirectional Radio Range

ADF Automatic direction finder

USB Universal Serial Bus

ILS Instrument Landing System

RNAV Area Navigation

Dodatok B

Inštalačný manuál

Tento návod popisuje postup, ako správne nahrať program do mikrokontroléru Arduino, a následne nastaviť všetky potrebné veci k správnemu fungovaniu panelov.

B.1 Nahratie programu do mikrontroléra

Pre nahratie programu do Arduina je potrebné Arduino IDE, ktoré je dostupné na oficiálnych stránkach Arduina. Vývojové prostredie je dostupné pre Microsoft Windows XP a novší, macOS a Linux. Po stiahnutí a nainštalovaní, spustíme vývojové prostredie a pripojíme Arduino. Ďalej postupujeme podľa nasledujúcich bodov.

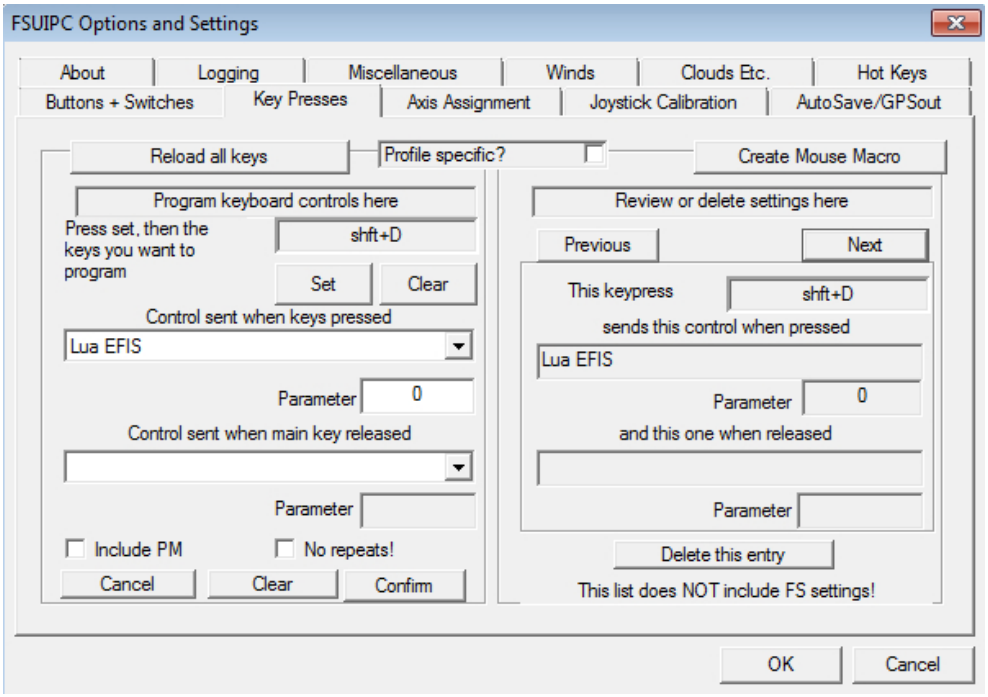
1. Vyberieme správny mikrokontrolér a jeho port: Tools –> Board –> Arduino/Genuino Mega, Tools –> Serial Port
2. Otvoríme súbor so zdrojovým kódom: File –> Open ... (MCP/EFIS.ino)
3. Po otvorení súboru nahráme program do mikrokontroléru: Sketch –> Upload
4. Po zobrazení „Upload done“, je program úspešne nahratý a pripravený

B.2 Úprava konfiguračného súboru

Na povolenie dátovej komunikácie s lietadlom Boeing 737 od PMDG je potrebná úprava súboru **737NGX_Options.ini**. Súbor sa nachádza v zložke, kde je nainštalovaný simulátor a následne pod zložkou **PMDG\PMDG 737 NGX**. Na koniec tohto súboru je potrebné pridať nasledujúce dva riadky:

[SDK]

EnableDataBroadcast=1



Obr. B.1: Konfigurácia FSUIPC

B.3 Konfigurácia FSUIPC

Požadovaný LUA skript skopírujeme do zložky **Modules**, ktorý sa nachádza v zložke nainštalovaného simulátora. Následne spustíme simulátor, z ponuky vyberieme lietadlo od PMDG , potvrdíme a spustíme let.

1. V menu vyberieme Add-ons –> FSUIPC
2. Po zobrazení nastavení prepneme na kartu **Key Presses**.
3. Pri položke „Press set, then the keys you want to program“ - klikneme na Set a stlačíme klávesovú kombináciu, ktorou následne budeme spúšťať skript.
4. „Control sent when keys pressed“ - v zozname vyhľadáme názov súboru so skriptom (pred názvom samotného skriptu je vždy Lua)
5. Klikneme na „Confirm“
6. Potvrdíme OK
7. Po stlačení klávesovej skratky sa skript spustí, a zariadenie je pripravené k používaniu

Pri každom spustení simulátora je najprv potrebné spustiť skript pomocou klávesovej skratky, ktorú sme nastavili. Automatické spúštanie skriptu, žiaľ, FSUIPC nepodporuje.

Dodatok C

Obsah priloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementácie
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF