ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra radioelektroniky

Justovací modul kmitočtového filtru

Frequency Filter Adjustment Module

Diplomová práce

Studijní program: Elektronika a komunikace
Studijní obor: Audiovizuální technika a zpracování signálů

Vedoucí práce: Ing. Stanislav Vítek, Ph.D.

Bc. Václav Král

Praha 2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Král** Jméno: **Václav** Osobní číslo: **420277**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra radioelektroniky**

Studijní program: **Elektronika a komunikace**

Studijní obor: **Audiovizuální technika a zpracování signálů**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Justovací modul kmitočtového filtru

Název diplomové práce anglicky:

Frequency Filter Adjustment Module

Pokyny pro vypracování:

Navrhněte a implementujte justovací modul kmitočtového filtru. Modul generuje jednorázově nebo v sekvencích řídicí signály paralelního rozhraní kmitočtového filtru podle vložených justovacích parametrů filtru. Justovací parametry filtru lze vkládat manuálně z ovládacího panelu justovacího modulu nebo z nadřízeného testovacího počítače prostřednictvím USB rozhraní. Justovací parametry filtru jsou zobrazeny na LCD displeji. Ovládací panel obsahuje klávesnici s rotačním voličem.
Program justovacího modulu musí vykonávat tyto základní funkce:
1) Volbu kmitočtového pásma filtru (UHF/VHF)
2) Nastavit výchozí technologické kmitočty (Jsou mimo základní kmitočtové pásmo.)
3) Nastavit ručně nebo automaticky kmitočet filtru
4) Vložit sekvenční tabulku kmitočtových kroků pro typ kmitočtového filtru.
5) Reset tabulky kmitočtů.
Řídicím jádrem justovacího modulu je jednodeskový počítač typu Ardunio / Raspberry Pi.

Seznam doporučené literatury:

[1] MOLLOY, Derek. Exploring Raspberry Pi: interfacing to the real world with embedded Linux. John Wiley & Sons, 2016.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Stanislav Vítek, Ph.D., 13137

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **12.02.2018** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **30.09.2019**

Ing. Stanislav Vítek, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 8. 1. 2019

Václav Král

Poděkování

Chtěl bych poděkovat panu Ing. Stanislavu Vítkovi, Ph.D. a panu Ing. Vladimíru Šmakalovi z firmy Rohde&Schwarz za odborné vedení práce, za pomoc a rady při zpracování této práce.

Abstrakt

Cílem práce bylo navrhnout a realizovat justovací modul pro kmitočtový filtr. Justovací modul byl rozdělen na dvě části. Část ovládání justovacího modulu umožňuje vyhodnocení vstupu z periferií a PC a zajišťuje požadované funkce zařízení včetně převodu frekvence na specifický binární kód s využitím operačního systému reálného času FreeRTOS. Tento binární kód je pomocí SPI sběrnice odeslán do paralelního rozhraní, které zajišťuje sériově-paralelní převod a přenos ostatních řídících signálů mezi filtrem a modulem.

Abstract

The goal of this project was to design and implement frequency filter adjustment module. Adjustment module was divided into two parts. Adjustment module control part allows to evaluate input from peripherals and PC and provides required functions including conversion of the frequency to the specific binary code using Real Time Operating System FreeRTOS. This code is then sent via SPI bus to the parallel interface, which ensures serial-to-parallel transfer and other communication between filter and module.

Klíčová slova

řízení; justovací modul; vývojová sada; Arduino; operační systém reálného času; FreeRTOS

Key words

controlling; adjustment module; development kit; Arduino; real time operating system; FreeRTOS

Obsah

1. Úvod

Cílem této práce je navrhnout a realizovat justovací modul elektromechanického filtru. Tento modul bude generovat řídící signály paralelního rozhraní filtru podle vložených parametrů. Justovací parametry bude možné vkládat manuálně z ovládacího panelu justovacího modulu nebo z nadřízeného počítače prostřednictvím USB rozhraní. Justovací parametry filtru budou zobrazeny na LCD displeji. Ovládací panel bude obsahovat klávesnici s rotačním voličem.

U elektromechanického kmitočtového filtru dochází k ladění rezonanční frekvence pohybem pístu uvnitř dutinové rezonanční komory. Filtr přijímá požadovanou frekvenci jako specifický binární kód, který dekóduje a nastaví píst do odpovídající polohy. K zajištění správné rezonanční frekvence je nutné pozici manuálně doladit. Zamýšlený modul slouží měřícímu technikovi k předání požadované frekvence do filtru, aby mohl ověřit a případně upravit skutečně nastavenou frekvenci.

Současný justovací modul umožňuje pouze přímé zadání frekvence ze svého ovládacího panelu. Navrhovaný modul zastane tuto funkci a umožní využití dalších funkcí, které usnadní práci měřícího technika. Mezi tyto funkce patří kontrola příslušnosti požadované frekvence do daného kmitočtového pásma, změna aktuální frekvence o krok vybraný z tabulky nebo krok manuálně zadaný a průchod tabulkou kmitočtových kroků.

Navrhovaný modul nabídne i snadnější ovládání. Na současném modulu se frekvence nastavovala zvětšením nebo zmenšením jednotlivých číslovek již zadané frekvence. Na navrhovaném modulu si bude měřící technik moci zvolit zadávání frekvence na klávesnici ovládacího panelu a volit různé funkce. K posunu frekvence o krok nebo k průchodu tabulkou frekvencí bude velmi užitečný rotační volič. K dispozici bude také ovládání z počítače.

V teoretické části bude představen elektromechanický filtr a požadavky pro nastavení frekvence. Dále bude popsána možnost využití vývojových sad a knihoven k urychlení návrhu modulů. Poté bude představen operační systém reálného času FreeRTOS a nakonec budou popsány různé protokoly sériové komunikace.

Praktická část bude obsahovat jednotlivé kroky návrhu justovacího modulu. Nejdříve bude uveden návrh ovládací části modulu, kam patří například implementace požadovaných funkcí a ovládacích periferií. Následovat bude návrh paralelního rozhraní pro přenos požadované frekvence do kmitočtového filtru. Nakonec bude zhodnocen aktuální stav návrhu justovacího modulu.

2. Teoretická část

V následující části budou představeny elektromechanický filtr a užitečná teorie.

2.1. Elektromechanický filtr

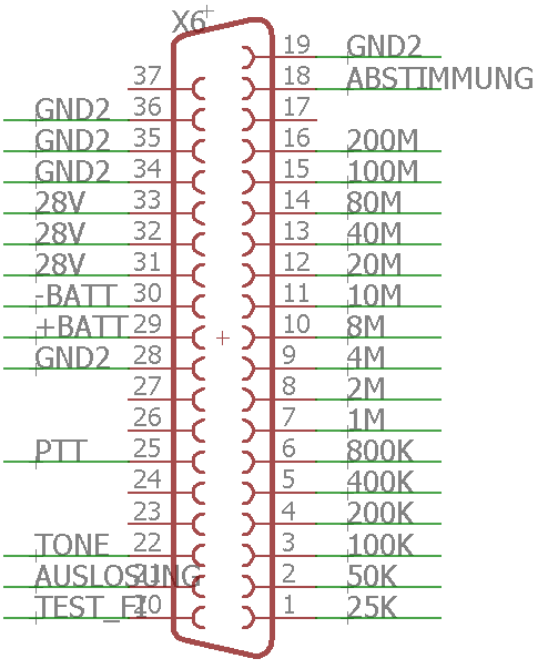
Obrázek 1 - Elektromechanický filtr R&S [1]

Firma Rohde&Schwarz vyrábí elektromechanické filtry R&S řady FT*, FD* a FU*. Tyto filtry jsou používané jak v pozemních stanicích, tak i v lodní dopravě. Filtr chrání přijímače proti rušivým signálům – zabraňuje intermodulacím, desenzitizaci rádia, potlačuje cross-modulační produkty, atd. Při vysílání dochází pomocí filtru k útlumu širokopásmového šumu způsobeného syntetizátorem a zesilovačem, k potlačení rušivých emisí zesilovače a harmonických frekvencí generovaných v koncových stupních nebo k potlačení intermodulačních produktů způsobených těsnou blízkostí více vysílačů. [1].

Podle typu fungují filtry v pásmu VHF (100 MHz až 162 MHz) a/nebo UHF (225 MHz až 400 MHz). Některé filtry obsahují obvod BYPASS pro ladění tísňových frekvencí. Tyto obvody jsou filtry naladěné na pevnou frekvenci a jsou zařazeny paralelně k filtru, který se běžně používá.

Filtr je naladěn na požadovanou frekvenci pohybem pístu v dutinové rezonanční komoře. Kvůli mechanické povaze ladění může docházet k určitým nepřesnostem naladěné frekvence. Filtr je proto nutné kalibrovat. Při kalibraci je zadána požadovaná frekvence a poté je postupným dolaďováním zjištěna poloha pístu, kdy rezonanční frekvence v komoře odpovídá požadované frekvenci. Kalibrační data jsou poté uložena v řídící elektronice filtru.

Kmitočtový filtr přijímá požadovanou frekvenci pomocí D-SUB konektoru s 37 piny. Schéma osazení konektoru je na dalším obrázku (viz Obrázek 2). Na piny 31 až 33 tohoto konektoru je nutné přivést napájecí napětí frekvenčního filtru, jehož typická hodnota je 28 V. Piny 19, 28 a 34 až 36 jsou spojeny se zemí napájecího napětí. Druhá možnost napájení filtru je napájení z baterie na pinech 29 a 30. Požadovaná frekvence je převedena na specifický kód a přenesena do filtru pomocí pinů 1 až 16 uvedených na obrázku (viz Obrázek 2).



Obrázek 2 – Schéma osazení pinů konektoru

Nastavení kmitočtu se řídí paralelní kombinací binárních hodnot kódu na rozhraní. Kód se dá získat porovnáváním převáděné frekvence s tabulkou binárního řízení. Pokud je převáděná frekvence vyšší než první hodnota v tabulce, je na dané pozici zapsána log.1. Poté je od převáděné frekvence odečtena první frekvence z tabulky a pokračuje se porovnáváním převáděné frekvence s druhou frekvencí z tabulky. Pokud je převáděná frekvence nižší než hodnota frekvence v tabulce, je na dané pozici zapsána log.0 a přejde se k porovnávání s dalším záznamem v tabulce. Například frekvenci 321.5 MHz bude odpovídat kód 1100100001010100. Filtr získá frekvenci z kódu sečtením frekvencí z tabulky, kterým odpovídá log.1 – z uvedeného příkladu 200M + 100M + 20M + 1M + 400k + 100k = 321.5 MHz. Frekvenční vstupy používají pozitivní TTL logiku.

Filtr signalizuje pomocí poklesu napětí na pinu 18 probíhající ladění Abstimmung. Na pinu 20 je připojen signál Auslösung, který značí, že je filtr nastaven a není aktivní signál TEST. Pomocí pinů 20, 22 a 25 se do filtru dá poklesem napětí odeslat signál TEST (blokuje nastavování filtru), TONE (blokuje BYPASS při rádiovém příjmu) a PTT (blokuje BYPASS při rádiovém vysílání). Všechny tyto signály používají negativní TTL logiku. Ostatní piny konektoru nejsou využity.

2.2. Využití vývojových sad

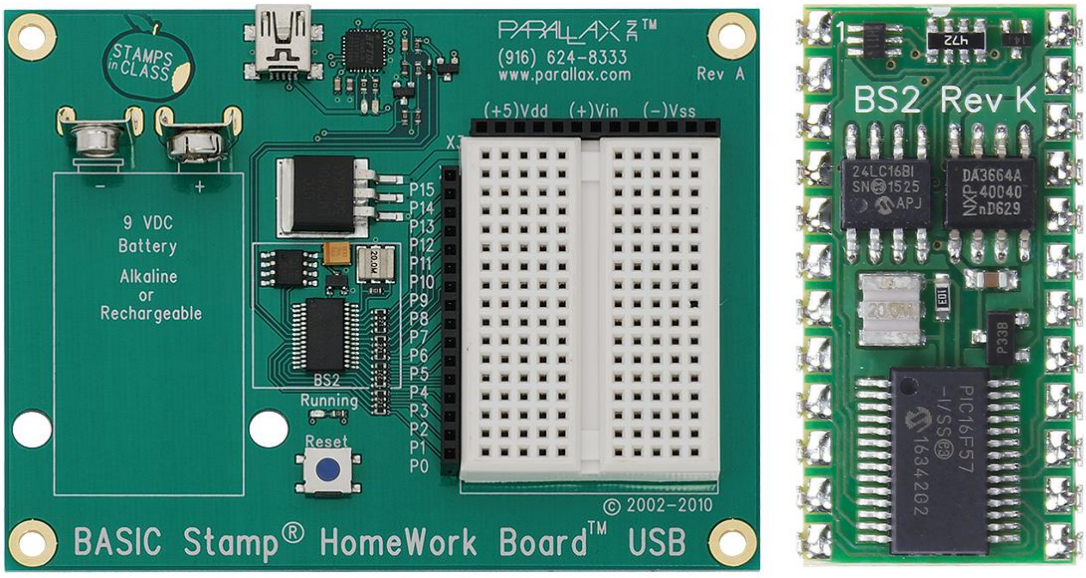
Návrh nových zařízení bývá časově náročná záležitost. Často je jako základ zařízení potřeba mikroprocesor, který přistupuje k vstupům a výstupům a reaguje na ně. Navrhnout takové zařízení od začátku může být časově náročné. V dnešní době mnoho výrobců poskytuje vývojové sady obsahující mikroprocesorové desky a programovací prostředí. Tyto desky obsahují různé mikroprocesory a mnoho různých druhů analogových i digitálních vstupů a výstupů, které můžou být použity pro čtení dat z různých senzorů, jako například snímače teploty a tlaku nebo akcelerometry, nebo k ovládání motorků. Programy můžou běžet samostatně na mikroprocesorové desce, ale existuje i možnost komunikace s připojeným počítačem.

2.2.1. BASIC Stamp 2

Firma Parallax Inc. nabízí mikroprocesorovou desku založenou na mikroprocesoru BASIC Stamp 2 (viz Obrázek 3). Kromě mikroprocesoru s CPU, ROM a interpreterem jazyku BASIC obsahuje deska i 2 kB I^2C EEPROM paměti, keramický rezonátor zajišťující hodinový kmitočet, napěťový regulátor a 16 externích vstupů/výstupů. Deska může být napájena například z 9V baterie. [2]

Pro programování je použita odnož jazyku BASIC nazvaná PBASIC. Tato varianta obsahuje běžné funkce mikroprocesoru, jako PWM, sériovou komunikaci, komunikaci pomocí protokolů I^2C a 1-Wire, ovládání běžných ovladačů LCD displejů, serv a motorků. Také je možné použít RC obvod k vyčtení analogové hodnoty.

Kromě desek založených na mikroprocesoru BASIC Stamp 2 nabízí firma Parallax Inc. i desky s výkonnějším mikroprocesorem Propeller. Výrobky této firmy jsou orientovány hlavně pro použití v robotice a vzdělávání. Cena samostatného BASIC Stamp 2 modulu se pohybuje okolo $49.

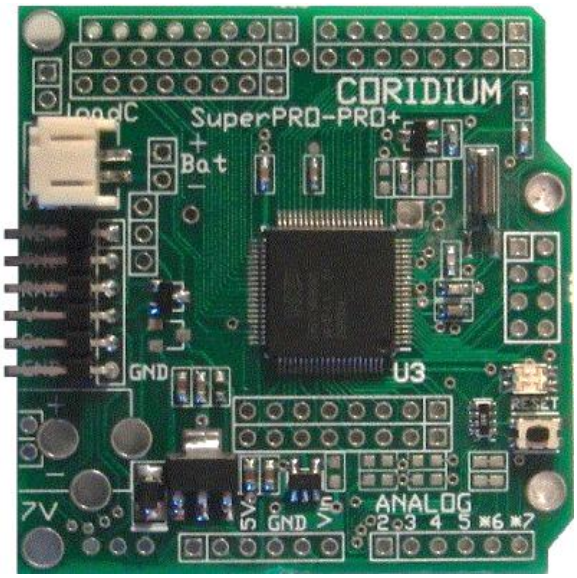


Obrázek 3 - Studijní deska HomeWork Board (vlevo) a samostatný BASIC Stamp 2 modul (vpravo) [2]

2.2.2. SuperPRO – LPC1756

Deska SuperPRO – LPC1756 (viz Obrázek 4) od firmy Coridium Corporation je jedna z několika desek založených na procesorech typu ARM, které firma nabízí.

Deska obsahuje mikroprocesor LPC1756 od firmy NXP typu ARM Cortex-M3, který běží na kmitočtu 100 MHz. K dispozici je 256 kB programové FLASH paměti a 32 kB RAM paměti. Ke komunikaci je možné využít mnoho standardů včetně USB, SPI nebo I^2C. [3] Spolu s 52 digitálními vstupy/výstupy jsou dostupné i čtyři 12-bitové A/D převodníky a jeden 10-bitový D/A převodník. K programování se používá jazyk BASIC a vývojový software je volně dostupný. Deska se dá pořídit za $49. [4]

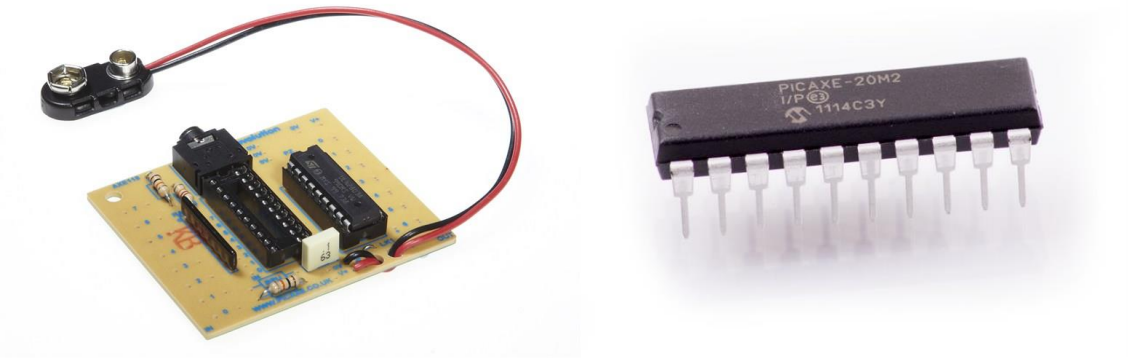


Obrázek 4 - Deska SuperPro - LPC1756 [4]

2.2.3. Picaxe

Desky Picaxe jsou orientovány převážně na vzdělávání, ale jsou oblíbené i pro tvorbu různých hobby projektů. Jako základ desek jsou použity mikroprocesory rodiny PIC16 nebo PIC18 od firmy Microchip Technology Inc. K dispozici jsou tak mikroprocesory s hodinovým kmitočtem 32 nebo 64 MHz, 2 až 16 kB programové paměti, 128 B až 1 kB RAM paměti a 8 až 40 vstupů/výstupů. Komunikovat lze pomocí standardů I^2C, SPI, RS-232 a 1-Wire. Mikroprocesory lze programovat jazykem BASIC nebo pomocí grafického rozhraní s použitím volně dostupného softwaru.

Vybraný mikroprocesor se dá vložit do samostatně prodávané desky (viz Obrázek 5). Desky jsou orientovány na použití v robotice a umožňují připojení zařízení, která požadují vysoký výstupní výkon (až 500 mA), jako motorky, solenoidní cívky nebo relé, přímo k desce. Sada desky a mikroprocesoru stojí podle typu okolo $10. [5]

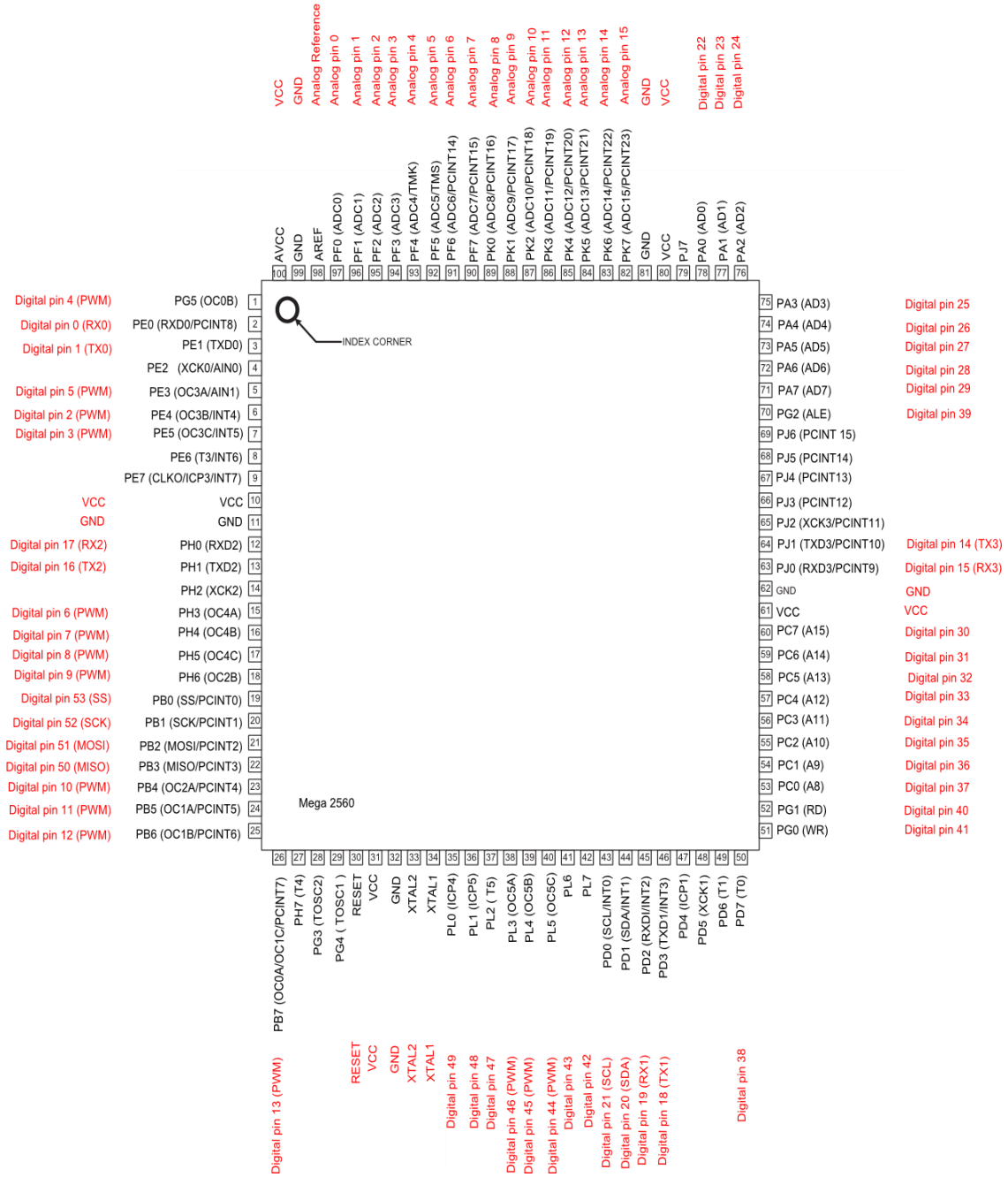


Obrázek 5 - PICAXE-20 Project Board (vlevo) a PICAXE-20M2 (vpravo)

2.2.4. Arduino Mega 2560

Arduino je open-source platforma poskytující hardwarové návrhy a software pro jednodeskové mikrokontroléry. Arduino bylo založeno jako studijní pomůcka a díky otevřenosti se dočkalo mnoha vylepšení od široké veřejnosti. Dnes se desky Arduino používají od výuky přes Internet věcí (IoT) nebo nositelnou elektroniku (wearables) až po vědecké nástroje. [6]

Základem desky Arduino Mega 2560 (viz Obrázek 7) je 8-bitový AVR mikroprocesor ATmega2560 s 256 kB flash paměti (z toho 8 kB zabere bootloader), 8kB SRAM a 4 kB EEPROM paměti. Namapování pinů desky Arduino na piny mikroprocesoru je vidět na následujícím obrázku (viz Obrázek 6).

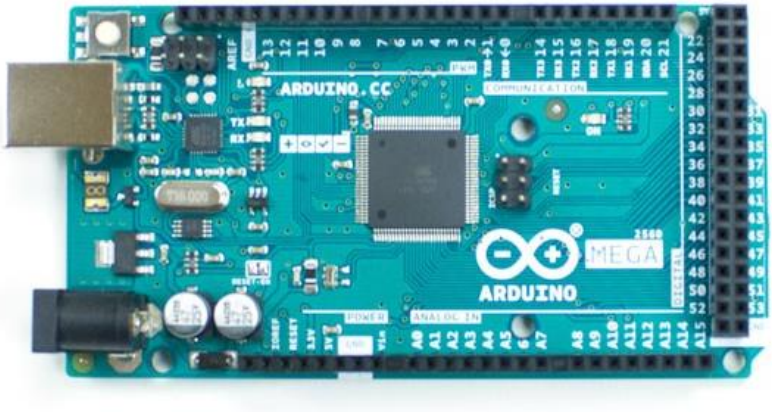


Obrázek 6 - Mapa pinů mezi deskou Arduino a mikroprocesorem ATmega2560

Deska Arduino Mega 2560 má 54 digitálních pinů, které lze použít jako vstup nebo výstup. Funkce pinu se nastaví pomocí funkce pinMode(), na pin je pak možné zapsat hodnotu funkcí digitalWrite(), případně vyčíst logickou úroveň pomocí funkce digitalRead(). Každý pin pracuje na 5 V a má (ve výchozím stavu odpojený) pull-up rezistor o hodnotě 20 až 50 kΩ. Každým pinem může protékat doporučený proud 20 mA. Překročením proudu 40 mA dochází k poškození mikroprocesoru.

Určité digitální piny jsou vyhrazené speciálním funkcím. Pro odesílání (TX) a příjem (RX) sériových TTL dat jsou určeny následující funkce a piny - Serial: 0

(RX) a 1 (TX); Serial 1: 19 (RX) a 18 (TX); Serial 2: 17 (RX) a 16 (TX); Serial 3: 15 (RX) a 14 (TX). Piny 0 a 1 jsou připojeny k odpovídajícím pinům čipu ATmega16U2, který zajišťuje převod sériového signálu na USB rozhraní.



Obrázek 7 - Arduino Mega 2560

Další digitální piny můžou být využity pro sledování externích přerušení. Piny 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3) a 21 (interrupt 2) lze naprogramovat ke spuštění přerušení při nízké nebo vysoké úrovni, při náběžné nebo sestupné hraně nebo při změně úrovně. Přerušení se nastavuje funkcí attachInterrupt().

Každý z digitálních pinů 2 až 13 a 44 až 46 lze použít jako výstup 8-bitové PWM pomocí funkce analogWrite().

Deska také podporuje SPI komunikaci přes digitální piny 50 (MISO), 51 (MOSI), 52 (SCK) a 53 (SS), případně lze využít ICSP hlavičku, kam jsou signály také vyvedeny.

K digitálnímu pinu 13 je připojena LED dioda, která svítí při vysoké úrovni na pinu 13.

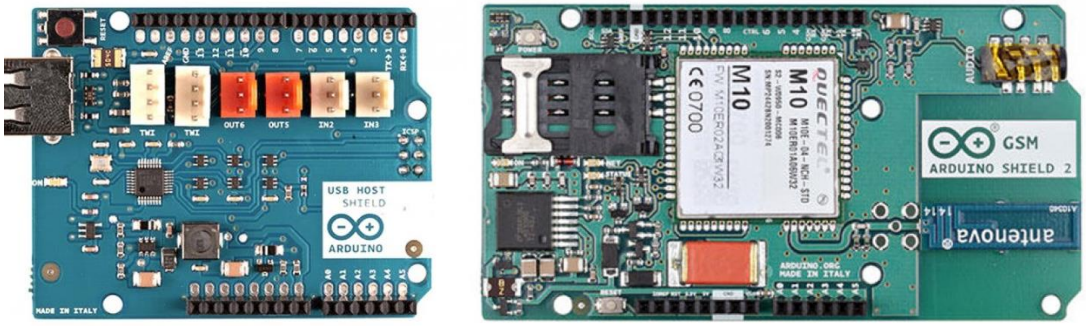
Digitální piny 20 (SDA) a 21 (SCL) podporují I^2C komunikaci.

Deska Mega 2560 má navíc 16 analogových vstupů A0 až A15, každý s 10 bitovým A/D převodníkem pracujícím od 0 do 5 V. Horní mez napětí lze změnit přivedením externího referenčního napětí na pin AREF. Pro snadné přidání resetovacího tlačítka je k dispozici i pin RESET, který po přivedení nízké logické úrovně resetuje desku.

Při nahrávání nového programu je užitečný automatický resetovací obvod. Pin DTR komunikačního čipu ATmega16U2 je přes 100 nF kapacitor připojen k resetovacímu pinu desky. Vývojový software před nahráváním nového programu do desky odešle pokyn k nastavení nízké úrovně na pin DTR, deska je resetována a může dojít k nahrání programu. [7]

K napájení desky lze použít USB konektor nebo externí zdroj stejnosměrného napětí 7 až 12 V. Napájecí napětí z externích zdrojů je převedeno na 5 V pomocí lineárního regulátoru.

Pro další rozšíření hardwarových funkcí slouží rozšiřující desky (shields), které se dají nasadit na desku Arduino a využijí některé piny k rozšíření desky o nové funkce a ostatní ponechají volné k dalšímu použití. Mezi oficiální rozšiřující desky patří například Arduino USB Host Shield, který umožní připojení USB klávesnice nebo myši, nebo Arduino GSM Shield, který umožňuje volání nebo připojení k internetu (viz Obrázek 8).



Obrázek 8 - Rozšiřující deska USB Host Shield (vlevo) a GSM Shield (vpravo)

Pro psaní programů (sketch) pro desky Arduino je dostupný vývojový software Arduino IDE. V tomto vývojovém prostředí se používá upravený jazyk C/C++ a programy se dají rozšířit o C++ knihovny. Programovat se ale dá v jakémkoliv vývojovém prostředí, které podporuje mikroprocesor použitý na dané desce.

Arduino IDE je dostupné pro Windows, macOS a Linux (viz Obrázek 9). Kódový editor poskytuje různé funkce jako dokončování závorek nebo zvýrazňování syntaxe a snadnou kompilaci a nahrání programu do desky pomocí jednoho tlačítka. Součástí vývojového prostředí je i sériový monitor, který umožňuje textovou komunikaci mezi vývojovým prostředím a deskou.

Obrázek 9 - Vývojové prostředí Arduino IDE pro Windows

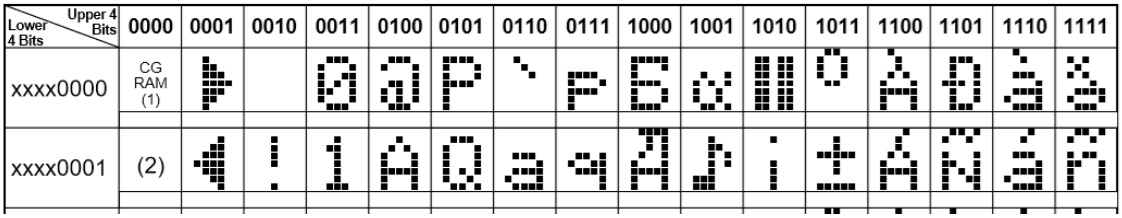
Každý program se skládá ze dvou základních funkcí. Funkce setup() proběhne pouze jednou při naběhnutí programu. V této funkci se definuje sériová komunikace a ukládají konstanty. Po naběhnutí a průchodu funkcí setup() se přechází na funkci loop(). Tato funkce je nekonečná smyčka a mikroprocesor opakovaně vykonává zde uvedené příkazy. Programy jsou ukládány v textové podobě s příponou ‘.ino‘.

2.3. Knihovny

Knihovny jsou balíčky funkcí pro specifické použití. Namísto opětovného programování často používaných funkcí pro každý program zvlášť lze využít předem připravené funkce. Představme si nyní několik takových knihoven.

2.3.1. LiquidCrystal

Knihovna pro Arduino LiquidCrystal umožňuje ovládání LCD displejů založených na čipsetu Hitachi HD44780, které jsou použity ve většině LCD pro zobrazování textu. K přenosu znaků lze použít 4 nebo 8 vodičů plus signály RS, ENABLE a případně ovládací signály pro R/W. [8] Signálem RS se rozlišuje mezi instrukčními a datovými registry ovladače displeje, signálem R/W se volí čtení nebo zapisování dat a signálem E se značí začátek zápisu nebo čtení dat. Znaky jsou v ovladači displeje uloženy v registrech. První 4 bity signálu vybírají sloupec a druhé 4 bity řádek z tabulky znaků (viz Obrázek 10). Knihovna zajišťuje převedení znaku na kód a jeho odeslání do displeje. [9]



Obrázek 10 - Ukázka tabulky znaků uložené v ovladači displeje [9]

Pomocí této knihovny je možné nastavit pozici kurzoru, zvolit směr přidávání znaků nebo například posouvání textu delšího, než je počet znaků na displeji. Po namapování pinů displeje stačí uživateli pro zobrazení textu smazat současný text, nastavit pozici kurzoru na začátek řádku a odeslat požadovaný text. Na následujícím obrázku je uveden jednoduchý program, který zobrazí na LCD displeji dobu od začátku běhu programu (viz Obrázek 11).

```
#include <LiquidCrystal.h> // deklarace pouziti knihovny

LiquidCrystal lcd(23, 25, 27, 29, 31, 33); // prirazeni pinu displeje
// LiquidCrystal lcd(rs, enable, d4, d5, d6, d7)

unsigned int cas = 0; // pocitani casu

void setup() {
  lcd.begin(16,2); // definice dipsleje o 2 radcich po 16 znacich
}

void loop() {
  lcd.clear(); // smazani textu na displeji
  lcd.setCursor(0,0); // posun kurzoru na zacatek prvniho radku
  lcd.print("Od startu ubehlo:"); // zobrazeni textu
  lcd.setCursor(0,1); // posun kurzoru na zacatek druheho radku
  lcd.print(cas); // zobrazeni casu
  lcd.print(" sekund");
  cas++; // zvyseni casu
  delay(1000); // prodleva 1 sekunda
}
```

Obrázek 11 - Ukázka definice LCD displeje a zobrazení času od začátku běhu programu

2.3.2. Keypad

Knihovna Keypad pro Arduino se používá k připojení klávesnic s maticovým vyčítáním. Vývody řádků a sloupců klávesnice lze připojit přímo na piny desky Arduino díky využití interních pull-up rezistorů. Uživatel přiřadí každé klávese ASCII znak a namapuje řádky a sloupce klávesnice na piny desky Arduino. Knihovna potom umožňuje získat znak odpovídající stisknuté klávese, zjistit stav kláves (nečinnost, stisknutí, uvolnění a držení) nebo například čekat na stisknutí. [10] Na následujícím obrázku (viz Obrázek 12) je příklad programu, který při stisknutí klávesy odešle odpovídající znak do PC.

```
#include <Keypad.h>

const byte rows = 4; // ctyri radky
const byte cols = 3; // tri sloupce
char keys[rows][cols] = { // prirazeni znaku tlacitkum
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'#','0','*'}
};
byte rowPins[rows] = {5, 4, 3, 2}; // piny, ke kterym jsou pripojene radky
byte colPins[cols] = {8, 7, 6}; //piny, ke kterym jsou pripojene sloupce
// definice klavesnice
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, rows, cols );

void setup() {
  Serial.begin(9600); // zahajeni seriove komunikace s PC
}

void loop() {
  char customKey = keypad.getKey(); // zjisteni stisknute klavesy
  if (customKey) { // pokud doslo k stisknuti jakekoliv klavesy
    Serial.println(customKey); // odesle se odpovidajici znak do PC
  }
}
```

Obrázek 12 - Ukázka programu, který při stisknutí klávesy odešle odpovídající znak do PC

2.3.3. Java AWT

Knihovna AWT pro jazyk Java umožňuje vytvoření jednoduchého grafického uživatelského prostředí. [11] Knihovna se skládá ze dvou částí, a to balíčku java.awt, který zajišťuje grafickou stránku, a balíčku jav.awt.event, který vyřizuje události (stisknutí tlačítka, poloha myši, …).

Obrázek 13 - Různé dostupné komponenty v knihovně AWT [12]

Grafická část obsahuje dva hlavní prvky. Komponenty (Component) jsou základní jednotky jako tlačítko (Button) nebo popisek (Label) (viz Obrázek 13). Kontejnery (Container), jako rámec (Frame) a panel (Panel), slouží k umístění komponent do určitého rozložení. Kontejner může v sobě obsahovat další

podkontejner. Ukázka vytvoření tlačítka je na dalším obrázku (viz Obrázek 14). [12]

```
Panel pnl = new Panel();   // vytvoreni panelu
Button btn = new Button("OK");  /// nove tlacitko s nazvem OK
Pnl.add(btn);   // pridani tlacitka do panelu
```

Obrázek 14 - Ukázka vytvoření tlačítka s knihovnou AWT [12]

Pro vyhodnocení události je nutné definovat rozhraní, které obsahuje objekt Listener. Když je vyhodnocena interakce, provede se uvedená funkce (viz Obrázek 15).

```
public interface ActionListener {
   public void actionPerformed(ActionEvent evt);
}
```

Obrázek 15 - Definice rozhraní v AWT [12]

Při vyhodnocení události je detekován zdroj a volány všechny aktivní objekty Listener. Listener musí převzít kontrolu nad volanou funkcí a zajistit její vykonání (viz Obrázek 16).

```
public class AWTCounter extends Frame implements
ActionListener {
@Override
   public void actionPerformed(ActionEvent evt) {
        // pozadovana akce po stisknuti
   }
}
```

Obrázek 16 - Definice požadované akce po stisknutí tlačítka [12]

Nakonec je pro každé tlačítko přidán objekt Listener (viz Obrázek 17).

```
btn.addActionListener(this);
```

Obrázek 17 - Přidání objektu Listener k tlačítku [12]

2.3.4. RXTX

Knihovna RXTX pro jazyk Java poskytuje možnost využití sériové nebo paralelní komunikace pro Java Development Toolkit. [13]

Nejdříve je nutné detekovat dostupné porty (viz Obrázek 18).

```
Vector<String> ports = getPortList();

String[] nazvyPortu = new String[ports.size()];

for (int i = 0; i < ports.size(); ++i) {

     nazvyPortu[i] = ports.elementAt(i);

}

return nazvyPortu;
```

Obrázek 18 - Ukázka detekce dostupných portů

Uživatel nyní může vybrat požadovaný port podle názvu. Poté dojde k vyhledání identifikačního čísla portu s daným názvem a pokusu o vytvoření spojení (viz Obrázek 19).

```
serialPort = (SerialPort)
PortId.open(this.getClass().getName(), TIME_OUT);

serialPort.setSerialPortParams(DATA_RATE,

                SerialPort.DATABITS_8,

                SerialPort.STOPBITS_1,

                SerialPort.PARITY_NONE);

// open the streams

input = new BufferedReader(new
InputStreamReader(serialPort.getInputStream()));

output = serialPort.getOutputStream();


serialPort.addEventListener(this);

serialPort.notifyOnDataAvailable(true);
```

Obrázek 19 - Ukázka navázání spojení s vybraným portem [13]

Nyní lze odesílat a přijímat data (viz Obrázek 20).

```
if (oEvent.getEventType() == SerialPortEvent.DATA_AVAILABLE){
      String inputLine=null;
      if (input.ready()) {
          inputLine = input.readLine();
          return inputLine;
      }
}
```

Obrázek 20 - Ukázka kódu zajišťujícího příjem dat

2.4. Operační systémy reálného času (RTOS)

V běžném počítačovém programu jsou instrukce vykonávány v pořadí, v jakém jsou zapsané v kódu. Toto zaručuje naprostou předvídatelnost běhu programu, nelze ale zaručit, že instrukce proběhnou v předem daný okamžik. Zároveň nelze zpracovávat více instrukcí zdánlivě najednou (multitasking).

Operační systém je počítačový program, který zajišťuje základní funkce počítače a poskytuje služby ostatním programům, které na daném počítači běží. Programy běžící nad operačním systémem bývají jednodušší, protože odpadá nutnost ošetřovat interakci s hardwarem, kterou zajišťuje právě operační systém.

Operační systémy umožňují běh více programů zdánlivě zároveň. Toto bývá zajištěno částí OS zvanou scheduler. Scheduler rozhoduje, který program bude v danou chvíli mít přístup k výpočetnímu výkonu procesoru. Rychlým přepínáním přístupu k procesoru mezi různými programy vzniká iluze současného běhu programů.

Scheduler se může řídit různými pravidly podle požadavků uživatele. Základní verze přiděluje každému běžícímu programu spravedlivý podíl na čase, kdy může běžet (např. Unix). V jiných aplikacích je nutné zajistit včasnou reakci programů na vstup uživatele (např. Windows).

Operační systém reálného času (RTOS) je definován použitím scheduleru, který zajišťuje deterministické vykonávání instrukcí. Pro provedení instrukcí, které mají pevné časové požadavky, je nutné mít předvídatelný scheduler. Toho je dosaženo přiřazením priority každému vláknu instrukcí. Uživatel rozhodne, která vlákna jsou důležitější než jiná, a scheduler potom umožní vykonávání vlákna, které má v daný moment nejvyšší prioritu a které je připravené k běhu. [14]

2.4.1. FreeRTOS

FreeRTOS je projekt přinášející funkce operačních systémů reálného času na mikrokontroléry. Mikrokontroléry mají omezené hardwarové prostředky a vykonávají specifické funkce. Není tedy potřeba úplného operačního systému. FreeRTOS poskytuje jádro RTOS (kernel), které zajišťuje základní funkce jako scheduler, komunikaci mezi vlákny (nazývanými úlohy = tasks), časování a synchronizaci.

Kernel reálného času nemusí sloužit jen k včasné reakci programu. Kernel zajišťuje také časování (které tedy není závislé na hardwaru), modularitu (každá úloha plní jasný úkol), opětovné využití kódu (kód je nezávislý na hardwaru), vyšší účinnost (úlohy běží, pouze když je to potřeba) a další.

FreeRTOS je volně dostupný k použití i v komerčních produktech. Vývoj systému je podporován službou placené podpory. Dobrá dostupnost vedla k širokému rozšíření systému – FreeRTOS kernel je dnes dostupný na více než 35 architekturách mikroprocesorů.

2.4.2. Distribuce FreeRTOS

FreeRTOS je možné kompilovat na více než 20 kompilérech pro více než 35 architektur mikroprocesorů. Každá kombinace kompiléru a mikroprocesoru má vlastní port FreeRTOS. Tento port se skládá ze sady zdrojových kódů v jazyce C. Umožnění funkce RTOS je velmi snadné, stačí zkompilovat zdrojové soubory pro daný port spolu s kódem uživatele. Pro urychlení práce je pro každý port dostupná demo aplikace, z které může uživatel začít budovat svůj program.

FreeRTOS je definovaný v hlavičkovém souboru (FreeRTOSConfig.h). V tomto souboru jsou zvoleny základní funkce kernelu pro danou aplikaci, jako například typ scheduleru (kooperativní nebo preemptivní).

Všechny programy musí obsahovat zdrojové soubory tasks.c a list.c. Téměř vždy je využit soubor queue.c, který umožňuje použití řad a semaforů. Volitelné jsou potom soubory timers.c (softwarový časovač), event_groups.c a croutine.c (téměř nepoužívané).

Každý port také obsahuje soubor definující datové typy. Datový typ TickType_t konfiguruje periodické přerušení tick interrupt. Tento datový typ používá 16- nebo 32-bitovou unsigned hodnotu k uložení počtu tiků procesoru. Datový typ BaseType_t je definován jako nejúčinnější datový typ pro danou architekturu.

Pro snadnou orientaci v programu je zavedena konvence jmenování proměnných. Každý název proměnné by měl obsahovat předponu definující její typ („c“ pro char, „s“ pro int16_t = short, „l“ pro int32_t = long, „x“ pro BaseType_t a jiné nestandardní typy, dále „u“ pro unsigned hodnoty a „p“ pro proměnné typu pointer). Názvy funkcí obsahují předponu odrážející typ návratové hodnoty a typ funkce (Task, Queue, Timer, …). Stejně jako je běžný datový typ int nahrazen typem BaseType_t, jsou i hodnoty TRUE a FALSE nahrazeny makry pdTRUE a pdFALSE (případně pdPASS a pdFAIL).

2.4.3. Správa paměti

Paměť ve FreeRTOS kernelu je dynamicky alokována při běhu programu pokaždé, když je vytvořen nebo smazán objekt kernelu. FreeRTOS převádí správu paměti ze základní vrstvy do přenosné vrstvy, aby bylo zajištěno správné časování. Toto navíc umožňuje uživateli použít vlastní implementaci, pokud je to potřeba.

Použití základních funkcí malloc() a free() je podporované, ale není doporučené. Namísto těchto funkcí jsou používané funkce pvPortMalloc() a vPortFree(). Každý port obsahuje 5 různých ukázkových implementací těchto funkcí (soubory heap_1.c až heap_5.c).

Heap_1 je základní možnost správy paměti. Paměť je alokována pro všechny objekty před startem scheduleru v daném paměťovém poli a po dobu běhu programu se nemění. Heap_2 umožňuje uvolnění paměti a alokaci nových míst

v paměti i za běhu programu. Heap_3 obsahuje bezpečnou implementaci standardních funkcí malloc() a free() formou pozastavení scheduleru. Heap_4 je obdoba heap_2, ale s lepší správou paměťových bloků. Hrozí tak menší šance fragmentace paměti. Heap_5 je obdobou heap_4, ale je možné alokovat paměť z více paměťových prostorů.

2.4.4. Správa úloh

Úlohy (tasks) jsou běžné funkce v jazyce C. Musí ovšem být typu void a přijímat parametr typu void pointer. Každá úloha je samostatný program, který obsahuje vstupní bod, nekonečnou smyčku, a nesmí skončit (nesmí obsahovat příkaz return nebo dojít na konec kódu). Ukončení úlohy je zajištěno externě. Jedna funkce může být použita k vytvoření více úloh, které mají vlastní stack a vnitřní proměnné definované ve funkci.

Úlohy mají dva základní stavy. Úloha ve stavu Running je právě vykonávána na procesoru. Ve stavu Running může být v daný okamžik pouze jediná úloha. Všechny ostatní úlohy ve stavu Not Running neběží na procesoru a jejich proměnné a kontexty jsou uloženy. Po přechodu do stavu Running se navrátí vykonávání úlohy k poslední instrukci a uloženému stavu paměti, která byla vykonávána před vstupem do stavu Not Running.

Úlohy jsou vytvořeny pomocí funkce xTaskCreate() (viz Obrázek 21). První argument ukazuje na funkci, která definuje úlohu. Druhý argument obsahuje pojmenování úlohy. Ve třetím argumentu je definována velikost paměti. Čtvrtý argument umožňuje předání parametru do funkce. Pomocí pátého argumentu se volí priorita úlohy. V šestém argumentu je možno přiřadit úloze handle, kterým poté může být úloha referencována.

```
BaseType_t xTaskCreate(    TaskFunction_tpvTaskCode,
                                const char* const pcName,
                                uint16_t usStackDepth,
                                void *pvParameters,
                                UbaseType_t uxPriority,
                                TaskHandle_t *pxCreatedTask);
```

Obrázek 21 – Příklad vytvoření úlohy [14]

Úlohy mohou být vytvořeny nejen při startu programu, ale i za běhu pomocí funkce vTaskCreate(). Zároveň je možné smazat úlohu pomocí funkce vTaskDelete(), pokud již není dále potřeba.

Priorita úlohy je definována při jejím vytvoření a může být změněna i při běhu programu pomocí funkce vTaskPrioritySet(). Maximální počet priorit je nastaven v hlavičkovém souboru FreeRTOSConfig.h. Scheduler může pro výběr úlohy, která má vstoupit do stavu Running použít dvě metody. Univerzální metoda je implementována v jazyce C a je možné ji využít ve všech portech FreeRTOS. V

této metodě není omezen maximální počet priorit, je ale doporučené používat jejich co nejnižší počet. V metodě optimalizované pro danou architekturu je použit assemblerový kód. Vykonání kódu je tak rychlejší, ale maximální počet úloh je omezen na 32.

Scheduler zajišťuje, že vždy poběží úloha s nejvyšší prioritou. V případě, že existuje více úloh se stejnou prioritou, zajistí scheduler spravedlivé rozdělení času na procesoru mezi tyto úlohy. Každé úloze je přidělen jeden Time Slice, na jehož konci vyhodnotí scheduler další úlohu, která má přejít do stavu Running.

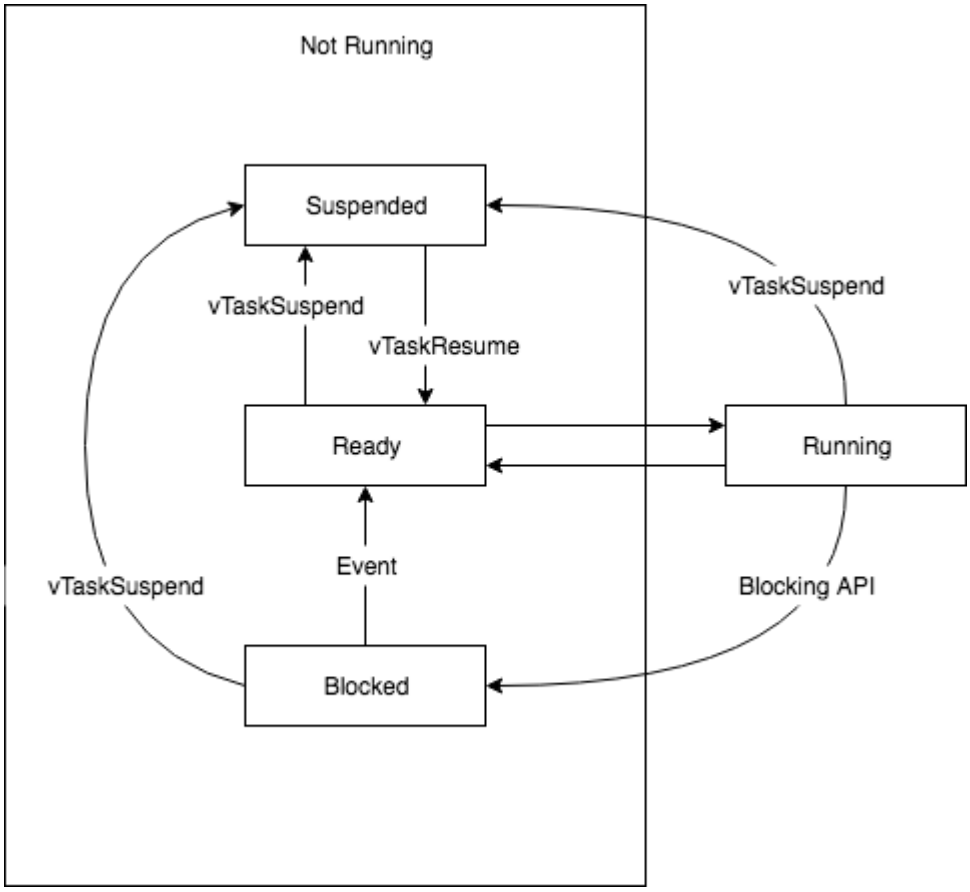
Základní stavy Running a Not Running umožňují pouze běh úlohy s nejvyšší prioritou (případně střídání úloh s nejvyšší prioritou). Pro úplné využití RTOS je nutné navrhnout úlohy tak, aby byly řízené událostmi. Takováto úloha poběží pouze po nastání určité události. Když událost nenastane, scheduler zvolí jinou úlohu, která běžet může. Tím pádem může dojít k běhu úlohy s nižší prioritou, pokud nebyla úloha s vyšší prioritou aktivována.

Úloha čekající na událost se nachází ve stavu Blocked, který spadá pod stav Not Running. Úloha může čekat na časovanou událost (například definované zpoždění) nebo synchronizační událost (událost, která pochází z jiné úlohy nebo z přerušení – například čekání na data). Zároveň s čekáním na synchronizační událost je možné zvolit i časový limit, po který může úloha čekat.

Dalším možným stavem úlohy je stav Suspended, také podstav stavu Not Running. Úloha ve stavu Suspended není dostupná scheduleru. Úlohy se přepínají do stavu Suspended a z něj pomocí funkcí vTaskSuspend() a vTaskResume().

Úlohy ve stavu Not Running, které nejsou ve stavu Blocked ani Suspended, jsou ve stavu Ready. Tyto úlohy jsou připraveny k běhu, ale nebyly schedulerem vybrány.

V každý okamžik musí existovat alespoň jedna úloha, která je schopná vstoupit do stavu Running. Za tímto účelem je automaticky při spuštění scheduleru vytvořena úloha Idle s prioritou 0. Tato úloha typicky obsahuje pouze prázdnou nekonečnou smyčku a obstarává vyčištění paměti. V některých případech může být tato úloha využita ke sledovaní využití procesoru nebo k umístění procesoru do Low Power módu.



Obrázek 22 - Přehled stavů FreeRTOS

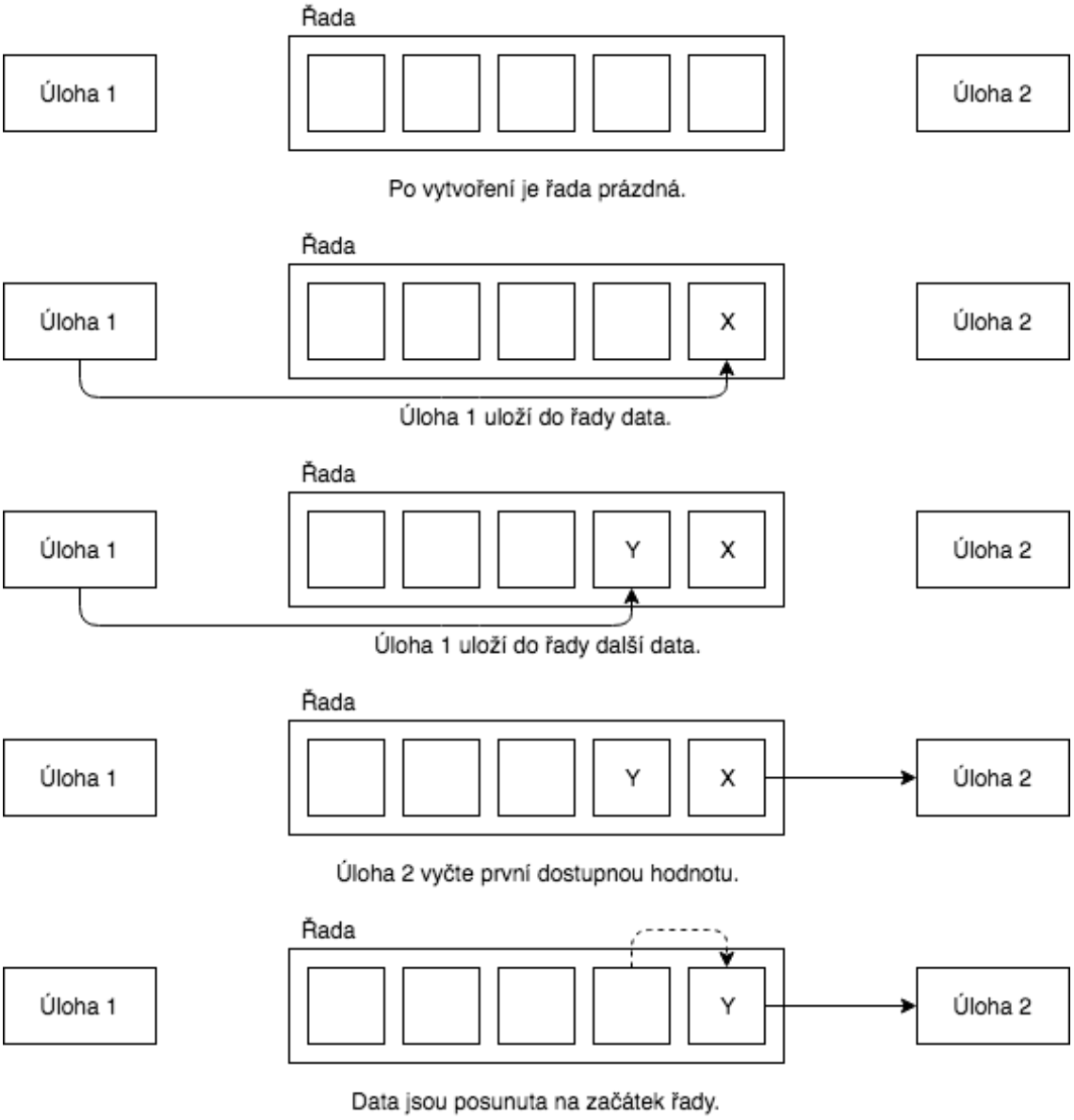
Algoritmus, podle kterého scheduler vybírá, která úloha bude přepnuta do stavu Running, je možné vybrat v hlavičkovém souboru.

Nejčastěji používaný algoritmus využívá upřednostněnou preempci s time slicingem. Úlohy mají pevně danou prioritu a budou zablokovány ze stavu Running, pokud úloha s vyšší prioritou přejde do stavu Ready. Stejně tak budou úlohy zablokovány na konci periody Time Slice, pokud je ve stavu Ready úloha se stejnou prioritou.

K dispozici je také kooperativní algoritmus scheduleru, kde dochází ke změně úloh pouze, když se úloha sama zablokuje, nebo když požádá o běh scheduleru.

2.4.5. Správa řad

Řady (queues) umožňují komunikaci mezi úlohami a mezi úlohou a přerušením. Řada obsahuje data určitého typu o určité délce. Tyto hodnoty jsou dány při vytvoření řady. Řady obvykle slouží jako FIFO paměti. Řady jsou dostupné všem úlohám, aby do nich zapisovaly i aby z nich vyčítaly, ale typicky vyčítá z řady pouze jediná úloha. Úloha při pokusu o vyčítání z řady může vstoupit do stavu Blocked po definovaný čas, dokud nejsou dostupná nějaká data. Pokud nejsou přijata žádná data po uplynutí blokovacího času, pokračuje úloha ve vykonávání. Stejně tak může být definován čas, po který je úloha zablokovaná a čeká na uvolnění místa v řadě, pokud se pokouší zapsat data do plné řady.



Obrázek 23 - Použití řady k přenosu dat

Řady se vytváří pomocí funkce xQueueCreate(), ve které jsou specifikovány délka řady a datový typ hodnot, které bude řada předávat. Poté je možné do řady zapsat data pomocí funkce xQueuSendToBack() nebo xQueueSendToFront(). Data se z řady vyčítají funkcí xQueueReceive().

Pokud je potřeba použít jednu řadu pro přenos dat s různými datovými typy nebo pro přenos dat z více zdrojů s nutností identifikace těchto zdrojů, je možné použít řadu, do které se ukládají struktury. Struktura potom může obsahovat libovolný formát dat spolu s identifikací zdroje.

Pokud je potřeba předávat mezi úlohami větší data, je vhodné do řady umístit místo samotných dat pouze pointer na tyto data. Potom je ale nutné zajistit, že tato data nebudou změněna za běhu (tedy příjemce dostane přesně ta data, která byla aktuální v okamžik jejich odeslání).

Několik řad je možné spojit do sady řad. Pokud úloha čeká na data z více řad, nemusí tak kontrolovat každou řadu jednotlivě.

Typicky jsou data z řady po přečtení smazána. Řadu je ale také možno použít jako úschovnu, kdy v ní data zůstanou zachována i po přečtení. K tomuto slouží funkce xQueueOverwrite() a xQueuePeek().

2.4.6. Správa softwarového časovače

Softwarové časovače (timer) slouží k vykonání funkcí v určitý čas nebo s určitou periodou. Časovače jsou plně pod kontrolou jádra FreeRTOS a nepotřebují žádnou hardwarovou podporu. Po uplynutí definované periody dojde k zavolání obslužné funkce čítače. Tato funkce proběhne od začátku do konce (na rozdíl od úloh). U jednorázového časovače dojde k jedinému provedení obslužné funkce, po kterém je nutné časovač manuálně resetovat. Naproti tomu u samo-obnovujících časovačů dochází k vykonání obslužné funkce periodicky.

Všechny obslužné funkce jsou vykonávány uvnitř RTOS Daemon úlohy, která je vytvořena automaticky při startu programu spolu s řadou časovače, která slouží k posílání příkazů jako reset časovače. Pro Daemon úlohu platí stejná pravidla běhu jako pro ostatní úlohy.

Časovač lze vytvořit pomocí funkce xTimerCreate(), zahájen nebo resetován může být funkcí xTimerStart() a zastaven pomocí funkce xTimerStop(). Časovač se rozběhne až po spuštění scheduleru. Při změně periody časovače funkcí xTimerChangePeriod() se nová perioda počítá od okamžiku změny.

2.4.7. Správa přerušení

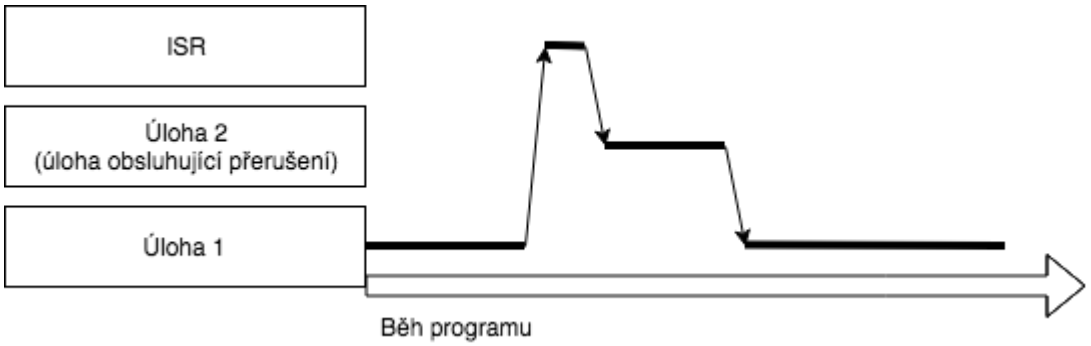
Jelikož běh úloh je řízen událostmi, je nutné zajistit zjišťování těchto událostí. První možností je neustále v softwaru vyhodnocovat, jestli nastala událost. Druhou možností je využití přerušení. Jelikož přerušení je hardwarová funkce daného zařízení, dojde při jeho vyhodnocení v obslužné funkci přerušení (ISR) k pozastavení běhu uživatelského programu. Proto je důležité minimalizovat množství příkazů v samotném ISR a většinu vyhodnocení přenechat na speciální úloze. Pokud je potřeba z ISR volat funkce obsluhující FreeRTOS, je nutné použít tyto funkce s příponou „FromISR“. Tímto jsou ošetřeny případy, kdy volaná funkce souvisí s úlohou, kterou obsluhuje, a tedy nemá kontext ke svému chodu.

Po spuštění přerušení tedy dojde k pozastavení programu a proběhnutí ISR, které umožní běh úlohy obsluhující dané přerušení. ISR je potom ukončen a spolu s tím i samotné přerušení programu. Vyhodnocení přerušení potom proběhne v obslužné úloze, až scheduler vyhodnotí, že tato úloha může proběhnout.

Ke spuštění obslužné úlohy je možné využít binární semafor, kterým synchronizujeme obslužnou úlohu s ISR. Binární semafor si můžeme představit jako řadu o délce 1, která drží binární hodnotu. Pokud je semafor prázdný, je

možné jej udělit pomocí funkce xSemaphoreGive() (popřípadě xSemaphoreGiveFromISR()). Úloha může čekat na převzetí semaforu, které zjistí funkcí xSemaphoreTake(). Když si úloha semafor převezme, je semafor uvolněn k dalšímu použití.

Obsluha přerušení může vypadat následovně: Obslužná úloha 2 bude mít nejvyšší možnou prioritu, ale bude zablokovaná hned od začátku programu. Při příchodu přerušení dojde v ISR pomocí semaforu k odblokování obslužné úlohy, a jelikož tato má vyšší prioritu než všechny ostatní úlohy, scheduler okamžitě ukončí právě běžící úlohu 1 a přepne úlohu obsluhující přerušení do stavu Running (viz Obrázek 24).



Obrázek 24 - Obsluha přerušení pomocí semaforu

Při použití binárního semaforu je možné, že dojde ke ztrátě informací. Pokud obsluhující úloha stále běží a přijdou další dvě přerušení, první přerušení bude uloženo do binárního semaforu, ale druhé bude ztraceno. V tomto případě je možné použít čítací semafor, jehož hodnota se pro každé udělení zvýší o 1 a při každém převzetí sníží o 1.

Pokud chceme z ISR přenést nejenom informaci o události, ale zároveň i nějaká data, je nutné použít řady s funkcí xQueueSendToBackFromISR() nebo xQueueSendToFrontFromISR(). Takovýto přenos dat je ovšem neefektivní, doporučuje se použít semafor a data vyčíst z bufferu v obslužné úloze.

2.4.8. Správa zdrojů

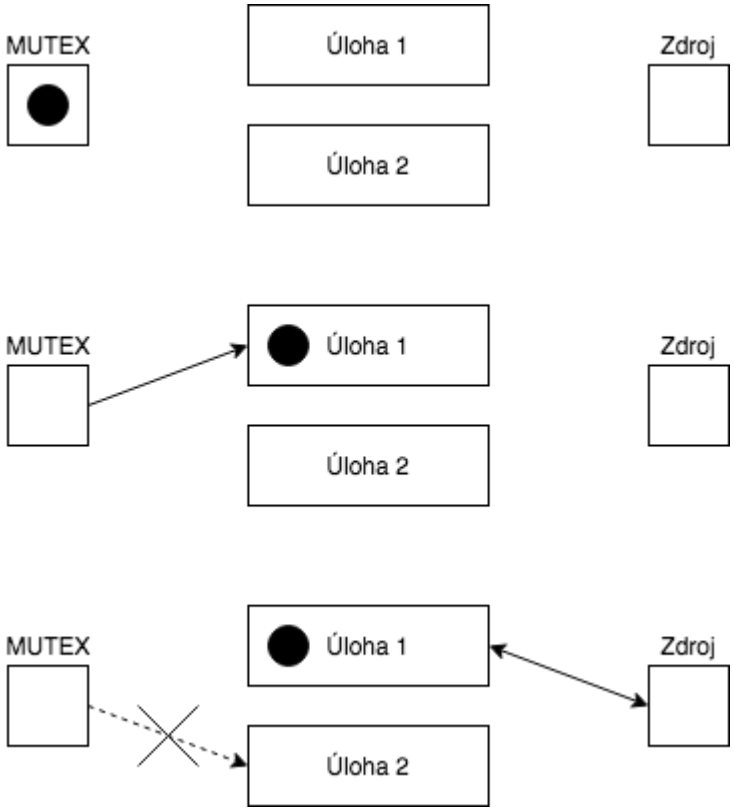
V systému, který využívá multitasking, je nutno zajistit, že určitý zdroj není využíván více úlohami najednou. Může například docházet k narušení zprávy vypisované na displej nebo přepsání globálních proměnných.

K zajištění stability zdrojů je využito principu vzájemného vyloučení. Ke zdroji je umožněn přístup pouze jediné úloze a není dostupný pro ostatní, dokud jej tato úloha opět neuvolní.

Základní možností je použití kritických sekcí. Při použití příkazů taskENTER_CRITICAL() a taskEXIT_CRITICAL() jsou dočasně vypnuty přerušení a scheduler a jakékoli zdroje použité mezi těmito příkazy budou exkluzivní dané úloze. Pokud jsou příkazy v kritické sekci příliš dlouhé, může

ovšem nastat situace, kdy nebude zachyceno přerušení. Pokud nechceme deaktivovat přerušení, ale jenom scheduler, je možné použít funkce vTaskSuspendAll() a vTaskResumeAll(). Přerušení bude zachyceno a jeho obslužná úloha notifikována, nedojde ovšem k přepnutí na tuto obslužnou úlohu, dokud nebude scheduler odblokován.

Další možností k řízení přístupu je využití speciálních binárních semaforů s názvem mutex (MUTual EXclusion). Pokud je mutex k dispozici, může si jej úloha vzít a přistupovat ke zdroji. Po ukončení práce se zdrojem tento mutex úloha vrátí. Pokud je daný mutex zabrán, musí úloha čekat na jeho uvolnění (viz Obrázek 25). Na rozdíl od předešlých možností musí při použití mutexů tvůrce programu zajistit, že každý přístup ke zdroji je podmíněn daným mutexem.



Obrázek 25 - Použití mutexů k řízení přístupu ke zdrojům

Při použití mutexů může docházet k situacím, kdy úloha s vyšší prioritou čeká na mutex, který zabrala úloha s nižší prioritou (inverze priorit). Tyto situace je nutné řešit buď opatrným návrhem práce se zdroji, nebo dočasným navýšením priority úlohy, která zabrala mutex.

Stejně tak je nutné předejít situacím, kdy více úloh používá více mutexů a tyto úlohy začnou čekat jedna na druhou, až budou mít k dispozici druhý mutex (deadlock). V tomto případě může dojít k úplnému zacyklení programu.

Nejbezpečnější formou vzájemného vyloučení je použití speciální hlídací úlohy, která má výlučný přístup k danému zdroji (gatekeeper task). Ostatní úlohy musí k přístupu k tomuto zdroji využít služby hlídací úlohy. Úlohy můžou například

využít řadu k poslání zprávy a hlídací úloha tuto zprávu předá na sériové rozhraní. Nikdy tak nedojde ke kolizi používání zdroje.

2.4.9. Skupiny událostí

Na rozdíl od řad a semaforů, které slouží k předání informace o jediné události, a to vyčkávající úloze s nejvyšší prioritou, můžou být skupiny událostí (event groups) použity k odblokování všech úloh čekající na jednu událost nebo specifickou kombinaci více událostí. Jejich použitím je možné snížit využití operační paměti, protože jedna skupina událostí dokáže zaujmout funkci několika semaforů.

Každá událost je reprezentována určitým bitem v proměnné typu EventBits_t. Skupiny událostí jsou samostatným objektem FreeRTOS a můžou k nim přistupovat jak úlohy, tak ISR. Jednotlivé bity lze nastavit funkcí xEventGroupSetBits() a vyčíst bity a čekat na správnou událost lze pomocí funkce xEventGroupWaitBits(). Pro synchronizaci úloh je možné využít funkce xEvenGroupSync(), která nevynuluje hodnoty po přečtení.

2.4.10. Meziúlohová oznámení

Zatím představené možnosti komunikace mezi úlohami vyžadovaly vytvoření určitého objektu (řada, semafor, …). Pro odeslání zprávy přímo do druhé úlohy lze použít meziúlohového oznámení (task notification). Oznámení je rychlejší než použití objektů a zároveň používá méně operační paměti, ale není vždy nejvhodnější. Oznámení se například nedá použít k poslání informací z ISR do úlohy. Zároveň oznámení slouží pouze ke komunikaci mezi 2 úlohami.

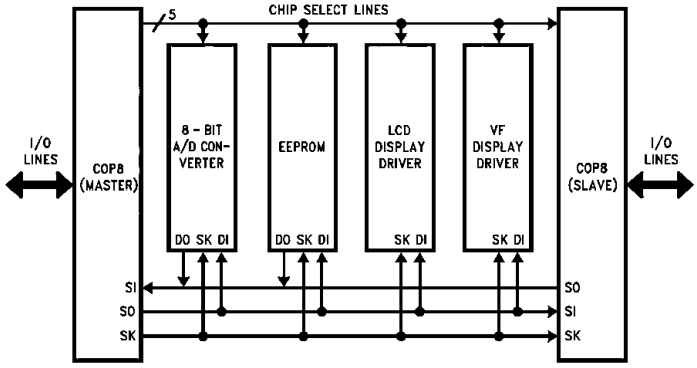
2.5. Komunikace v počítačích

K přenosu signálů v počítačích lze použít 2 různé typy komunikace. V paralelní komunikaci je přenášeno více bitů najednou (typicky 1 byte) za jeden hodinový cyklus. Naproti tomu v sériové komunikaci se dají přenášet libovolně dlouhá slova typicky pomocí 3 vodičů, ale přenáší se pouze jediný bit za hodinový cyklus. Při shodném hodinovém cyklu je tedy paralelní komunikace rychlejší, ale hodinový cyklus sériové komunikace může dosahovat několikanásobku cyklu paralelní komunikace a vyrovnat se tak rychlosti přenosu paralelní komunikaci. Díky menšímu počtu potřebných vodičů a srovnatelné rychlosti se sériová komunikace upřednostňuje před paralelní.

Pro sériovou komunikaci existuje několik sběrnic a příslušících protokolů, jako například MicroWire, I^2C, 1-Wire nebo SPI.

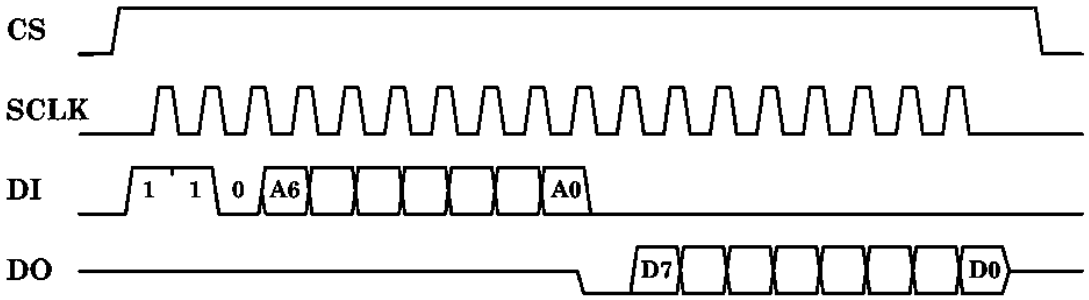
2.5.1. MicroWire

Standard MicroWire byl zaveden firmou National Semiconductors. Jedná se o předchůdce protokolu SPI. K přenosu dat se používají tři vodiče. Při náběžné hraně hodinového signálu SCLK dochází k detekci dat na datových vodičích SO/DI (Serial Out, Data In – odesílání dat do periferie) a DO/SI (Data Out, Serial In – příjem dat z periferie). K těmto třem vodičům je možné připojit mnoho periferních obvodů. Mikroprocesor vybere požadovaný periferní obvod pomocí příslušného vodiče CSn (Chip Select) (viz Obrázek 26). Délka slova není specifikována, ale obvykle se používají 4-bitové slabiky. [15]



Obrázek 26 - Připojení více periferií na sběrnici MicroWire [16]

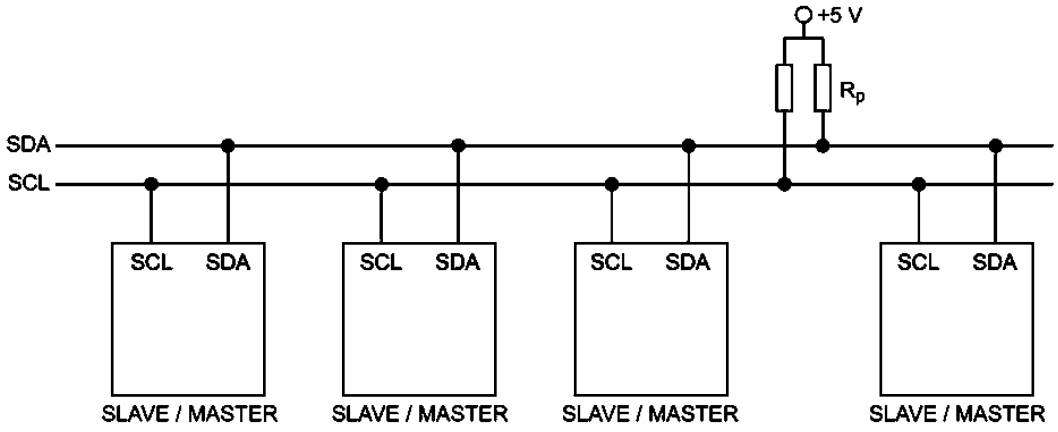
Přenos po sběrnici je zahájen odesláním hodnoty log.1 do periferního obvodu a následuje určitý počet bitů s příkazem. Pokud se jedná o obousměrnou komunikaci, dojde po odeslání příkazu k vyčítání dat z periferního obvodu. Typický časový průběh signálu při komunikaci standardem MicroWire je na následujícím obrázku (viz Obrázek 27).



Obrázek 27 - Časový průběh signálů na sběrnici MicroWire [15]

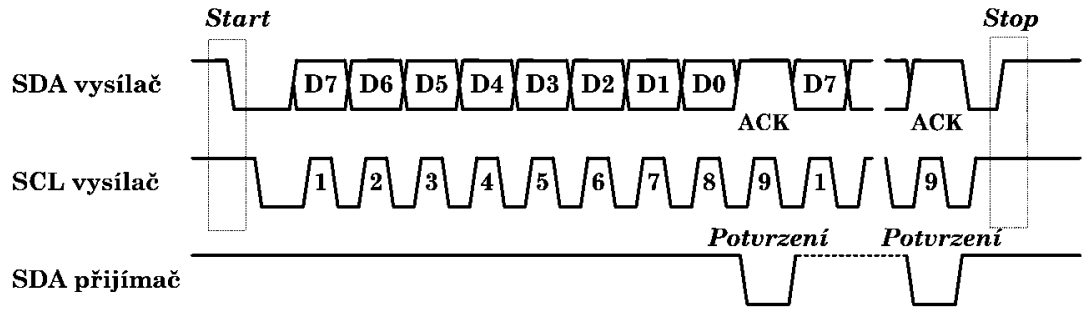
2.5.2. I²C

Firma Philips uvedla standard I²C (Inter-Integrated-Circuit-Bus). Tento protokol bývá často využíván ve spotřební elektronice, jako například u dekodéru barev v TV nebo u rozhraní telefonní linky. Ke komunikaci postačují dva vodiče, a to hodinový signál SCL (Serial Clock) a datový signál SDA (Serial Data). V klidovém režimu je na obou vodičích udržována hodnota log.1 pomocí pull-up rezistorů (viz Obrázek 28). Ke sběrnici je možné připojit mnoho zařízení, kdy typ zařízení se může měnit mezi Master a Slave podle potřeby. Každé zařízení má 7-mi nebo 10-bitovou adresu, která se skládá z typu zařízení a čísla zařízení na sběrnici. [16]



Obrázek 28 - Připojení zařízení ke sběrnici I²C [16]

Hodnota signálu SDA se smí měnit pouze, pokud je signál SCL v log.0 (kromě začátku a ukončení komunikace). Komunikace začíná, když zařízení typu Master stáhne linku SDA do log.0. Pokud probíhá na sběrnici komunikace (SDA je v log.0), musí se Master přepnout do režimu Slave a čekat na uvolnění linky. Po každém přeneseném bytu je přijímačem potvrzeno jeho přijetí (Ack = 0 znamená úspěšné přijetí, Ack = 1 vyvolá ukončení komunikace). Ukončení komunikace generuje vždy Master. Časový průběh komunikace je na následujícím obrázku (viz Obrázek 29). [15]

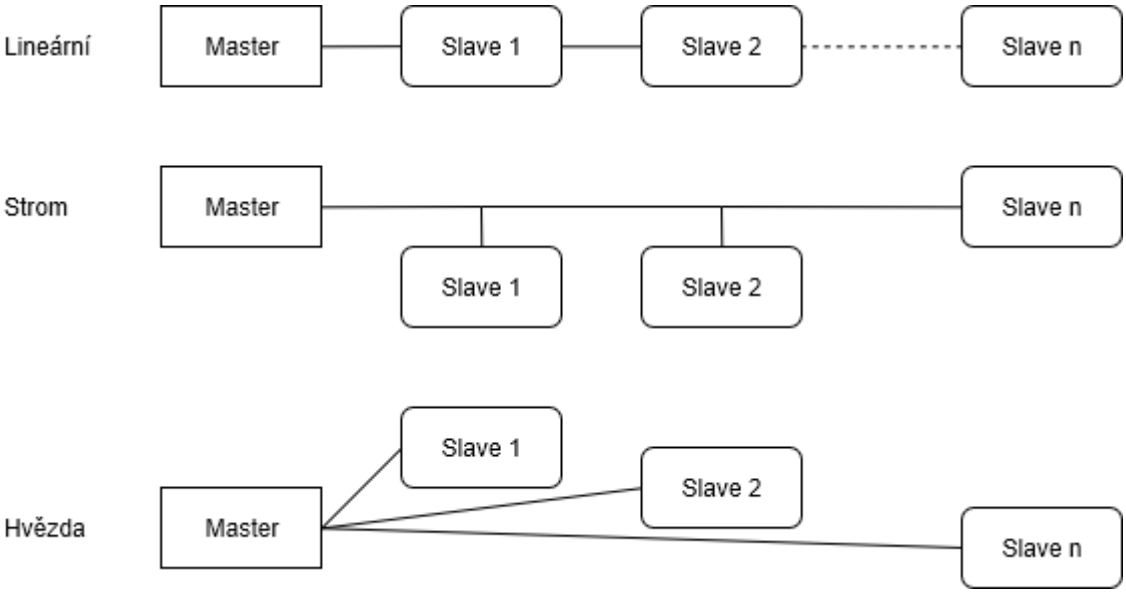


Obrázek 29 - Příklad komunikace na sběrnici I²C [15]

2.5.3. 1-Wire

Protokol 1-Wire (také označovaný MicroLAN) navrhla firma Dallas Semiconductor Corp. (dnes Maxim). Tato relativně pomalá sběrnice slouží k připojení specializovaných zařízení jako lékařské senzory nebo docházkové systémy.

Mikroprocesor slouží jako jediný Master pro až stovky Slave zařízení (viz Obrázek 30), které jsou připojeny přes jeden datový vodič a jeden zemnící vodič. U této sběrnice není využíván hodinový signál, je proto kritické dodržovat správné časování signálů. Připojené zařízení může být napájeno přímo z datového vodiče, kdy napájecí napětí v log.0 je udržováno vyrovnávacím kapacitorem, který je integrován v každém zařízení Slave. [16]

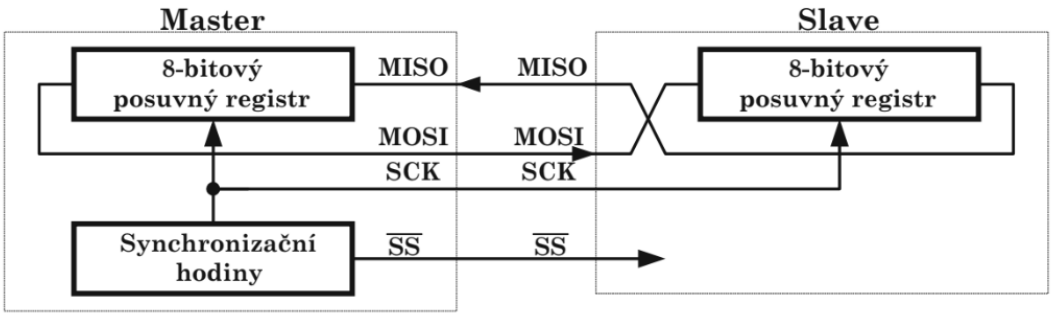


Obrázek 30 - Možnosti připojení Slave zařízení ke sběrnici 1-Wire

2.5.4. SPI

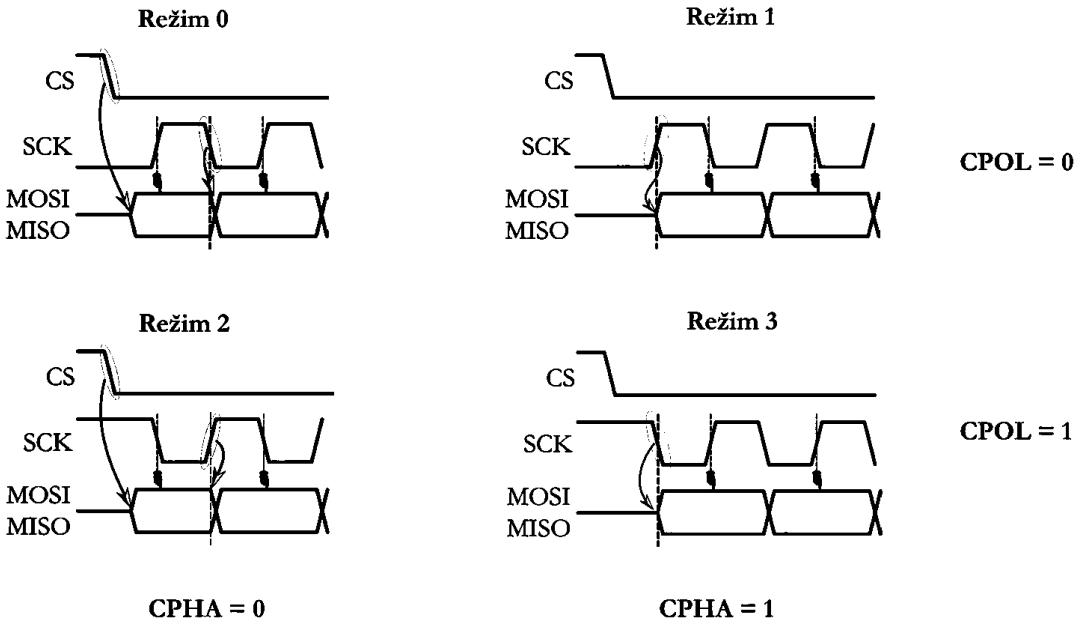
Sběrnice SPI (Serial Peripheral Interface) byla vyvinuta firmou Motorola. Tato sběrnice je velmi oblíbená a mnoho výrobců nabízí obvody s tímto rozhraním. Používá se hlavně v rámci desek tištěných spojů a pro komunikaci s blízkými periferiemi.

Ke komunikaci slouží tři signály. Hodinový signál SCK řídí přenos dat po datových vodičích MOSI (Master Out, Slave In – zápis dat do periferie) a MISO (Master In, Slave Out – čtení dat z periferie). Přenos dat je obousměrný, protože na mikroprocesoru dochází zároveň s odesíláním dat vodičem MOSI i ke čtení dat na vodiči MISO. Požadovaný obvod Slave je vybrán uvedením příslušného vodiče SS (Slave Select) do úrovně log.0 (viz Obrázek 31). [16]



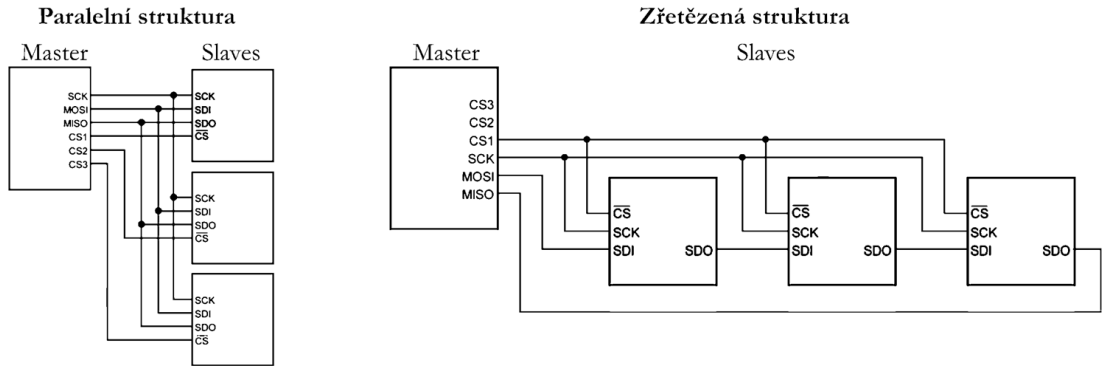
Obrázek 31 - Propojení procesoru s jedním podřízeným obvodem sběrnicí SPI [15]

Pro připojení obvodů, které nemají přímou podporu protokolu SPI je možné měnit formát přenosu pomocí změny bitu polarity (CPOL – klidová úroveň) a fáze (CPHA – hrana, při které dochází ke vzorkování) hodinového signálu. Komunikace na sběrnici SPI tak může probíhat ve čtyřech režimech (viz Obrázek 32). Pro klidový režim hodin v nízké úrovni (CPHA = 0) dochází k vzorkování dat při náběžné (CPOL = 0) nebo sestupné (CPOL = 1) hraně hodinového signálu. Pro klidový režim hodin ve vysoké úrovni (CPHA = 1) dochází k vzorkování dat při sestupné (CPOL = 0) nebo náběžné (CPOL = 1) hraně hodinového signálu. [15]



Obrázek 32 - Časové průběhy pro různé režimy komunikace po sběrnici SPI [16]

Zařízení Slave lze na sběrnici zapojit v paralelní nebo zřetězené struktuře (viz Obrázek 33). Ve zřetězené struktuře je výstup jednoho Slave zařízení připojen na vstup dalšího Slave zařízení. Všechna takto spojená zařízení mají stejný signál SS. V paralelní struktuře je každé Slave zařízení připojeno na datové vodiče a aktivní zařízení se vybírá příslušným signálem SS. Obě struktury lze kombinovat. [16]



Obrázek 33 - Struktury připojení Slave zařízení na sběrnici SPI [16]

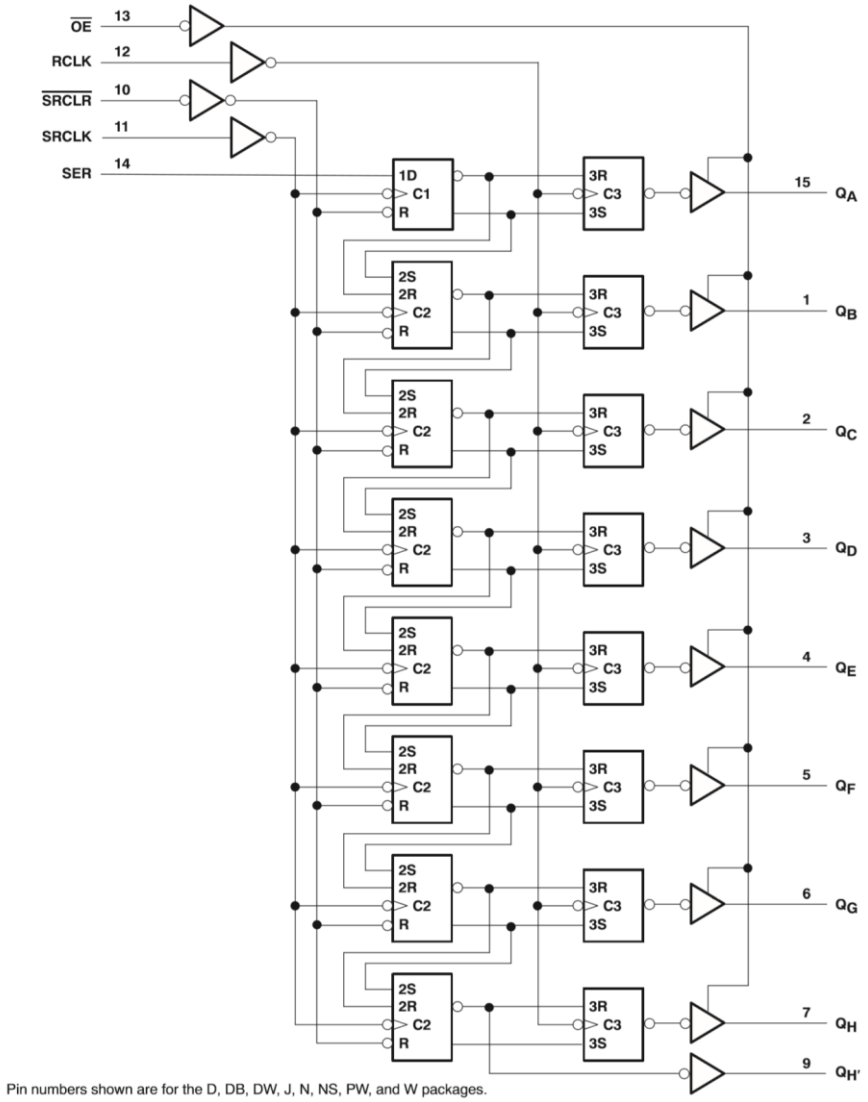
2.5.5. Obvod typu 595

Jako jednoduchý podřízený obvod pro SPI komunikaci lze použít obvod typu 595. Jedná se o 8-bitový serial-in, parallel-out posuvný registr. Sériová data jsou přivedena na pin SER a s každou náběžnou hranou hodinového impulsu na pinu hodin posuvného registru (SRCLK) dojde k posunu dat. Při detekci náběžné hrany na pinu hodin ukládacího registru (RCLK) dojde k zapsání dat z posuvného registru do ukládacího registru. Signály SRCLK a RCLK mohou být spojeny, potom jsou data v ukládacím registru zpožděna o jeden hodinový cyklus oproti datům v posuvném registru. Posuvný registr se dá smazat pomocí signálu /SRCLR a výstupy ukládacího registru Q_A-Q_H jsou ve stavu vysoké impedance, dokud nejsou aktivovány poklesem úrovně na pinu /OE do log.0 (viz Tabulka 1). Výstup $Q_{H'}$ slouží jako sériový výstup. Na tento pin lze připojit pin SER dalšího obvodu. [17] Schéma obvodu typu 595 je uvedeno na následujícím obrázku (viz Obrázek 34).

Obvod typu SN54HC595 může pracovat s napětím V_{CC} od 2 do 6 V a ke každému výstupu je možné připojit až 15 LSTTL zátěží. Maximální proudová spotřeba obvodu I_{CC} je 80 µA. Maximální proud do vstupů je 1 µA a typický výstupní proud z výstupů je ±6 mA při napětí 5 V. Maximální proud dodávaný do zátěže by neměl překročit 35 mA pro jeden výstup nebo 70 mA pro celou součástku. Na výstupy by nemělo být přivedeno vyšší napětí než V_{CC}. [17]

Tabulka 1 - Pravdivostní tabulka pro obvod typu 595 [17]

Vstupy					Funkce
SER	SRCLK	/SRCLR	RCLK	/OE	
X	X	X	X	H	Výstupy Q_A-Q_H vypnuty.
X	X	X	X	L	Výstupy Q_A-Q_H zapnuty.
X	X	L	X	X	Smazání posuvného registru.
L	↑	H	X	X	První prvek posuvného registru uveden do log.0 Další prvek obsahuje data předešlého prvku.
H	↑	H	X	X	První prvek posuvného registru uveden do log.1 Další prvek obsahuje data předešlého prvku.
X	X	X	↑	X	Zapsání dat z posuvného do ukládacího registru.



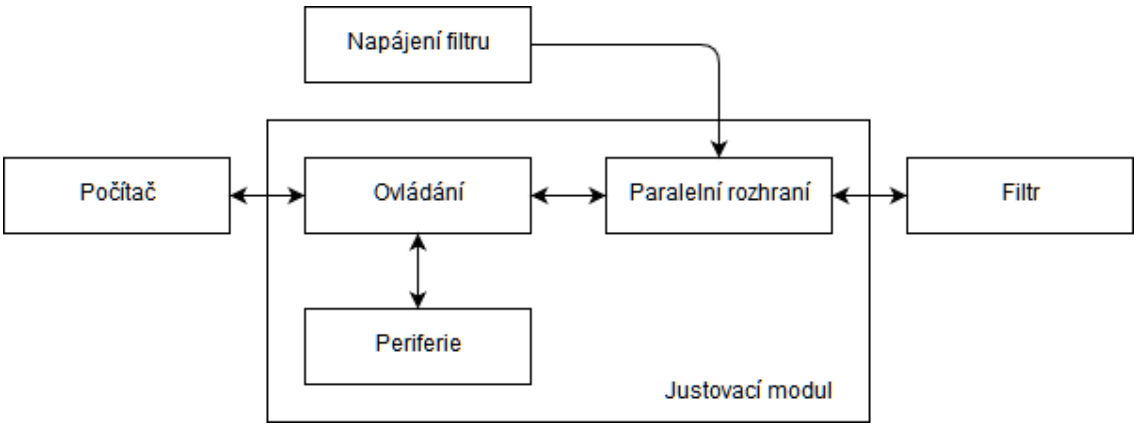
Obrázek 34 - Schéma obvodu typu 595 pro pozitivní logiku [17]

3. Praktická část

V následujícím textu se budeme zabývat návrhem justovacího modulu.

3.1. Základní rozdělení zařízení

S ohledem na technické požadavky bude nejvhodnější rozdělit justovací modul na 2 části (viz Obrázek 35). První část zajistí ovládání modulu a připojení periferií jako klávesnice a displej. Druhá část bude připojena pomocí kabelu a zajistí přenos příkazů mezi justovacím modulem a frekvenčním filtrem. Pro omezení počtu vodičů kabelu bude použita sériová komunikace k přenosu kódu frekvence.

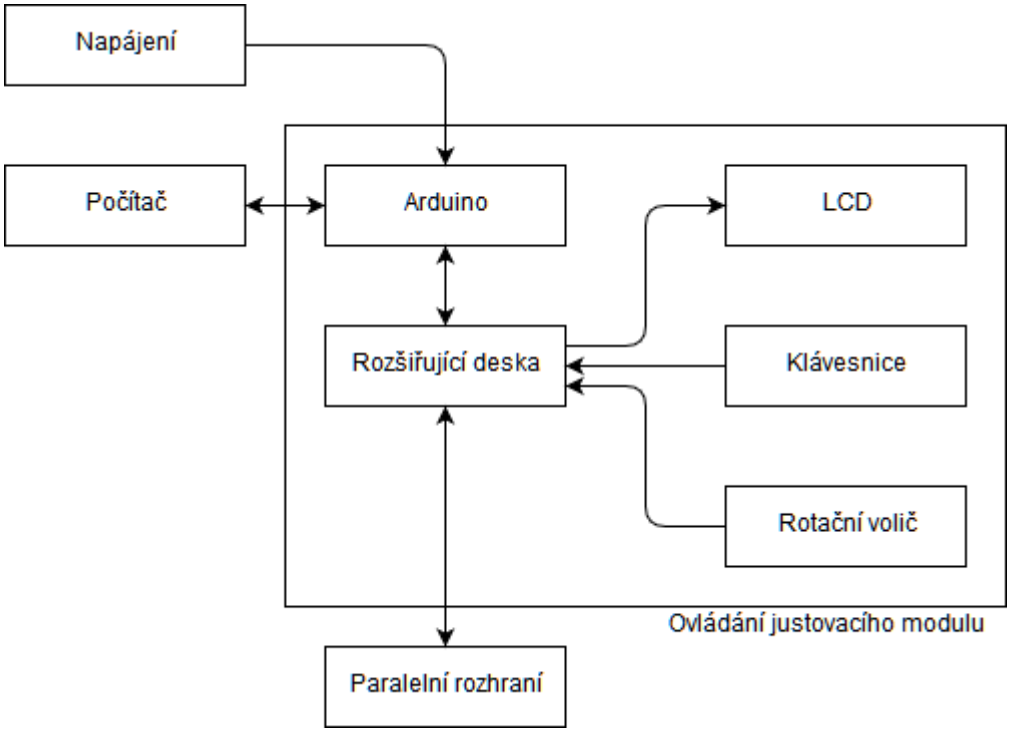


Obrázek 35 - Základní schéma zařízení

3.2. Ovládání justovacího modulu

Jako základ ovládací části justovacího modulu byla vybrána vývojová deska Arduino Mega 2560. Tato deska poskytuje dostatečný výkon pro běh ovládacího programu, komunikaci s počítačem pomocí USB rozhraní, rozhraní pro sériovou komunikaci i dostatečný počet vstupů a výstupu pro připojení všech periferií. Navíc jsou k dispozici i připravené knihovny pro ovládání těchto periferií.

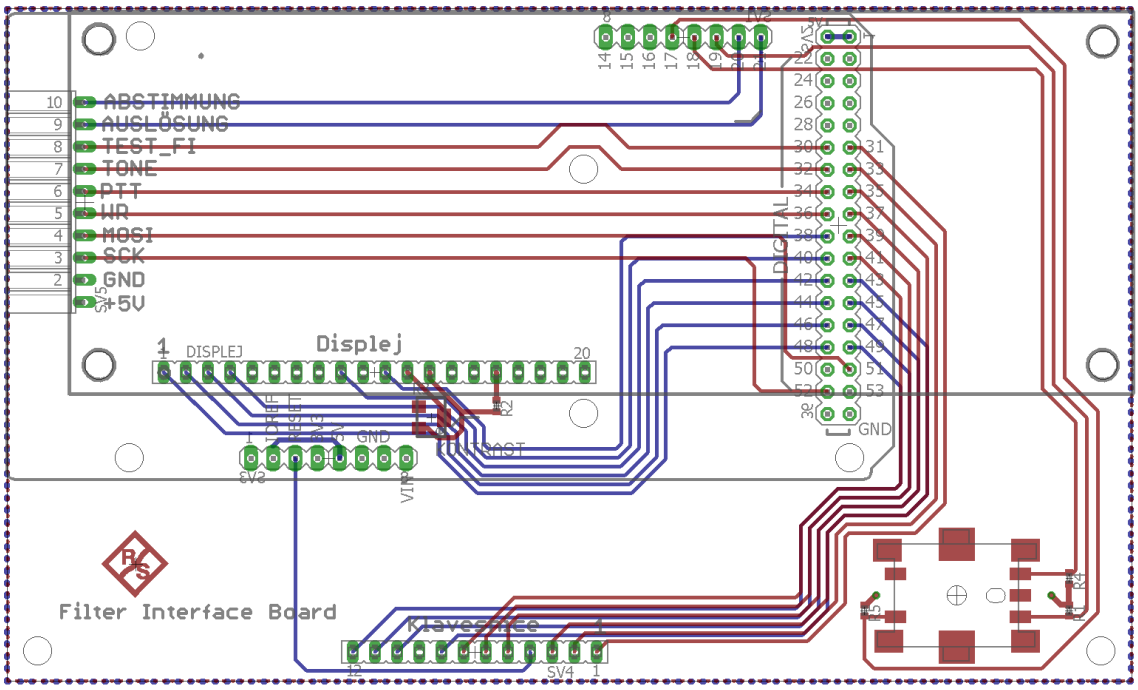
Bude navržena rozšiřující deska, která bude ze spodní strany kopírovat rozložení pinů desky Arduino a bude ji tedy možné přímo nasadit. Z horní strany budou osazeny konektory pro připojení periferií. (viz Obrázek 36)



Obrázek 36 - Schéma ovládání justovacího modulu

3.2.1. Interface Board

Pro připojení periferií byla navržena rozšiřující deska Interface Board (viz Příloha C), která ze spodní strany kopíruje rozložení pinů desky Arduino, a bude jí možné přímo na desku Arduino nasadit. Rozšiřující deska obsahuje konektor pro displej spolu s obsluhujícími obvody, konektor pro klávesnici a rotační volič spolu s obsluhujícími obvody.



Obrázek 37 - Návrh rozšiřující desky

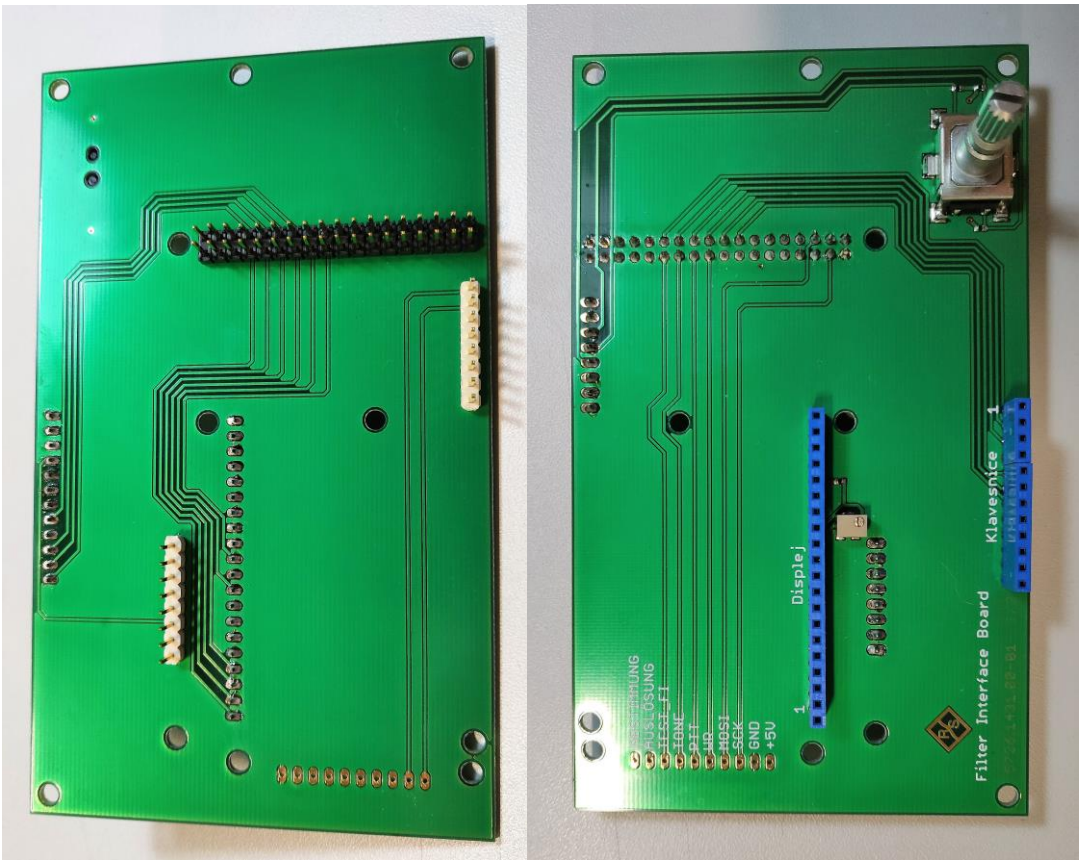
Tato deska musí také zajistit přenos dat do paralelního rozhraní. Jak už bylo uvedeno dříve, pro přenos kódu frekvence, který má 16 bitů, bude vhodnější použít sériový přenos oproti paralelnímu. Deska Arduino podporuje mimo jiné i sériový komunikační standard SPI. Jeho použitím se zredukuje počet vodičů pro přenos kódu frekvence z 16 na 3 (signály MOSI, SCK a WR, signál MISO nebude použit). V paralelním rozhraním poté bude potřeba použít sériově-paralelní převodník.

Mezi výstupní signály filtru patří signály Abstimmung a Auslösung. Signál Abstimmung značí poklesem úrovně na log 0, že ve filtru právě probíhá ladění frekvence. Signál Auslösung je v log 1, když je filtr nastaven a není aktivní signál TEST.

Filtr dále přijímá signály TEST, který v log 0 blokuje nastavování filtru, TONE, který v log 0 blokuje BYPASS při příjmu radiového signálu, a PTT, který v log 0 blokuje BYPASS při odesílání rádiového signálu.

Ve výsledku bude potřeba 10 vodičů k připojení ovládání justovacího modulu k paralelnímu rozhraní.

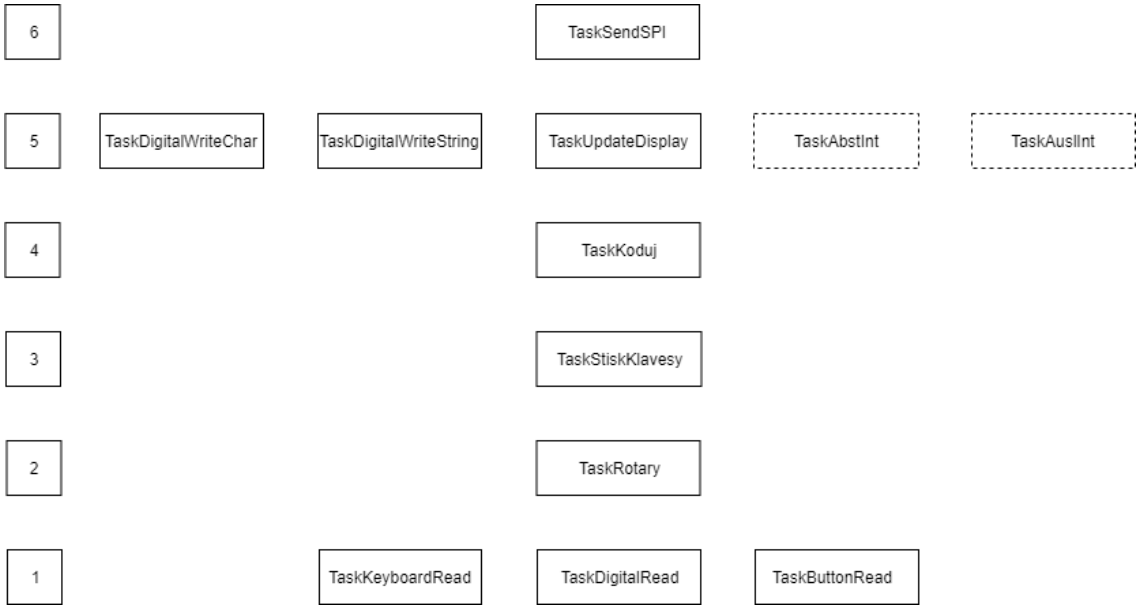
Rozšiřující deska tedy obsahuje rotační volič a konektory pro displej a klávesnici. Velikost desky a umístění konektorů bylo voleno pro snadné umístění celku do krabičky (viz Obrázek 37 a Obrázek 38 nebo Příloha D).



Obrázek 38 - Interface Board

3.2.2. Přehled programu

Obsluhující program bude využívat operačního systému reálného času FreeRTOS. Využitím takového systému lze snadno řídit běh programu a zajistit včasnou reakci na vstupy uživatele. Každá funkce (úloha) se chová jako samostatný program, má tedy část počátečních definic a část s nekonečnou smyčkou. Běh úlohy je umožněn schedulerem, který vybírá úlohu ve stavu Ready s nejvyšší prioritou (priorita 0 je nejnižší).



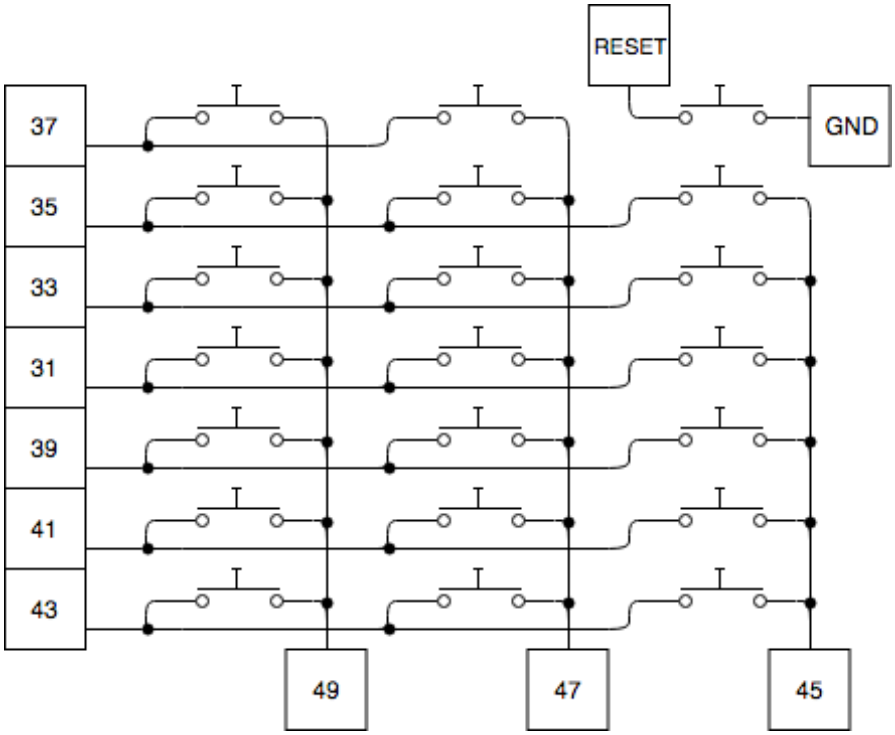
Obrázek 39 - Přehled úloh a jejich priorit

3.2.3. Zadávání pokynů

Čtení vstupů z periferií obsluhují úlohy s prioritou 1 (klávesnice, tlačítko rotačního kodéru a sériová komunikace) a 2 (otočení rotačního kodéru). U úloh s prioritou 1 dochází k opakovanému vyčítání, mají proto tyto úlohy nejnižší prioritu. K vyčítání dochází, když žádná jiná úloha nepotřebuje běžet. Vyčítání otočení rotačního kodéru má o stupeň vyšší prioritu, aby se zajistilo správné vyhodnocení vstupu.

3.2.3.1. Klávesnice

Hlavní možností zadávání pokynů je klávesnice. K dispozici byla membránová klávesnice o rozměru 3x7 tlačítek (viz Obrázek 41). Kontrola stisknutí tlačítka probíhá pomocí vyčítání řádků a sloupců. Jedno tlačítko není součástí maticového vyčítání, ale má nezávislé vývody. Bylo proto vybráno jako resetovací tlačítko a připojeno přímo na resetovací pin desky Arduino. Schéma zapojení klávesnice k pinům desky Arduino je na následujícím obrázku. (viz Obrázek 40)



Obrázek 40 - Schéma zapojení klávesnice k pinům desky Arduino

Vstup z klávesnice je vyhodnocen v úloze TaskKeyboardRead s použitím knihovny Keypad.h [10]. Po namapování řádků a sloupců tlačítek na klávesnici je každému tlačítku přiřazen ASCII znak (viz Tabulka 2). Při stisknutí tlačítka je příslušný znak předán pomocí řady xQueueKeyboard k dalšímu zpracování v úloze TaskStiskKlavesy. Kontrola stisknutí klávesy probíhá jednou za 50 ms, aby se předešlo zákmitům (úloha je po tento čas zablokovaná).

Tabulka 2 - Rozložení klávesnice a odpovídající znaky

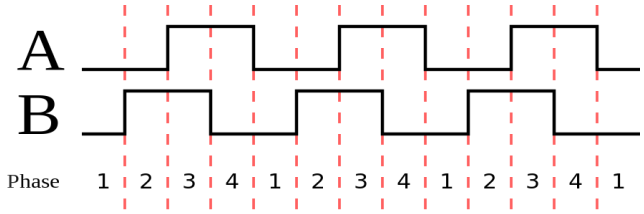
Obrázek 41 - Použitá membránová klávesnice

Tabulka frekvencí '+'	Krokování '='	RESET 'r'
TONE 'o'	TEST 'e'	PTT 'p'
7 '7'	8 '8'	9 '9'
4 '4'	5 '5'	6 '6'
1 '1'	2 '2'	3 '3'
Des. čárka ','	0 '0'	Pásmo 'k'
Zrušit 'z'	Smazat 's'	MHz 'M'

3.2.3.2. Rotační kodér

Obrázek 42 - Rotační volič Bourns PEC11S [18]

Rotační volič umožní snadnější změnu o krok nebo procházení tabulky kroků. U tohoto typu voliče dochází k vyhodnocení otočení pomocí detekování změny fáze signálů na pinech A a B (viz Obrázek 43). Například při otočení voliče ve směru hodinových ručiček dojde ke změně fáze ze 2 na 3. To odpovídá změně logických hodnot na pinech AB z 01 na 11. Ze znalosti tabulky přechodů a předešlé a současné fáze lze vyhodnotit směr otáčení.

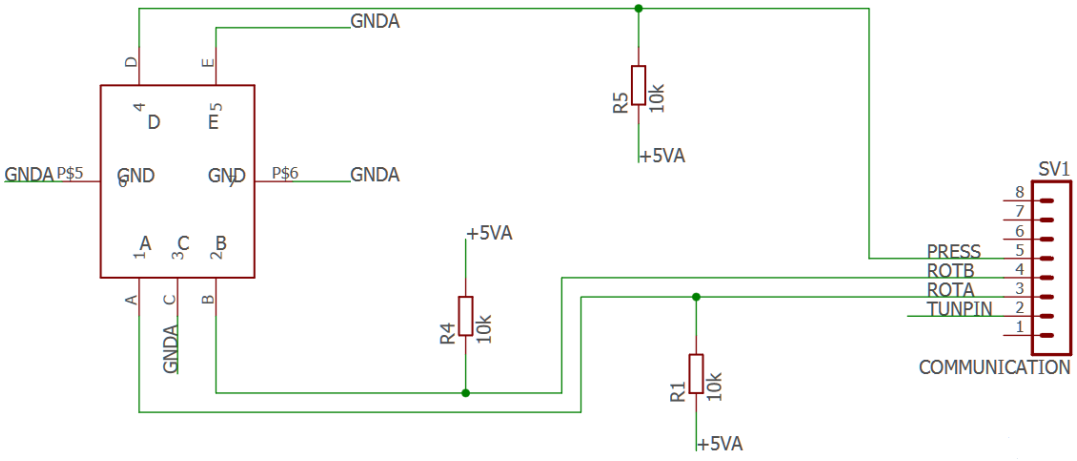


Obrázek 43 - Fáze pinů rotačního voliče

Byl navržen obvod pro rotační volič (viz Obrázek 44). Piny voliče byly připojeny k pinům desky Arduino podle následující tabulky (viz Tabulka 3).

Tabulka 3 - Připojení pinů voliče na desku Arduino

Pin voliče	A	B	C	D	E
Pin Arduino	19	18	GND	17	GND



Obrázek 44 - Schéma zapojení rotačního voliče

Pro efektivnější běh programu bylo zvoleno vyhodnocení vstupu rotačního kodéru pomocí přerušení. Pokud je vyhodnocena změna napětí na pinu A nebo B kodéru, dojde v obslužných funkcích přerušení (ISR) doEncoderA() nebo do EncoderB() k udělení semaforu xRotarySemaphore, na který je zablokovaná úloha TaskRotary(). Tím dojde k rychlému ukončení ISR a samotné vyhodnocení vstupu je provedeno, když je úloze TaskRotary() umožněn běh.

Jelikož během otočení kodéru o jeden krok dochází ke dvěma změnám fáze signálu, lze použit přechod mezi těmito fázemi k vyhodnocení směru otočení. Úloha tedy čeká na první změnu signálu a uloží si hodnoty vstupu A a B. Při následující změně vstupů si opět uloží hodnoty a z tabulky přechodů vyhodnotí směr otočení. Pro otočení ve směru hodinových ručiček odešle do řady xQueueKeyboard znak '>' odpovídající posunu nahoru. Pro opačný směr otočení dojde k odeslání znaku '<' pro posun dolů (viz Obrázek 45). Při vyhodnocení zakázaných přechodů nedochází k odeslání žádného znaku. Úloha je potom zablokována po dobu 10 ms pro omezení zákmitů. Po uplynutí této doby dojde k vyčištění semaforu a čekání na další vstup.

```
xSemaphoreTake(xRotarySemaphore, 65535 );  // cekani na umozneni behu
rotaryAold = digitalRead(rotaryAPin);   // ulozeni hodnot
rotaryBold = digitalRead(rotaryBPin);
xSemaphoreTake(xRotarySemaphore, 10 );  // vyhodnocuje az druhe preruseni
rotaryA = digitalRead(rotaryAPin);
rotaryB = digitalRead(rotaryBPin);
if (((!rotaryA) && (rotaryB)) && ((rotaryAold) && (rotaryBold))) {
  cKey = '>';
} else if {…
```

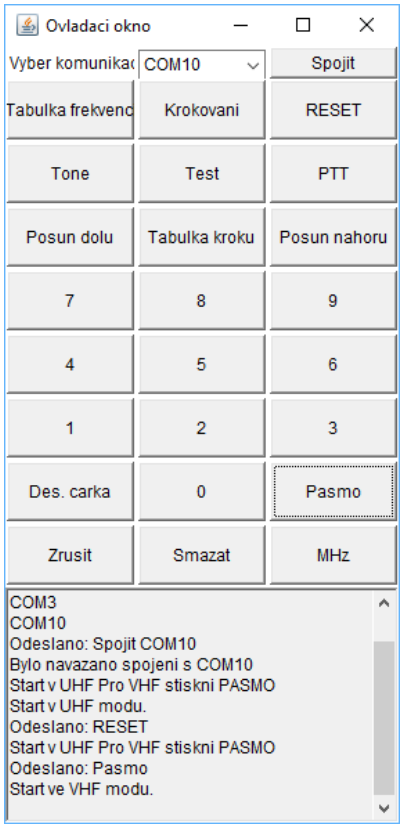
Obrázek 45 - Příklad vyhodnocení vstupu rotačního kodéru

Použitý volič také umožňuje stisknutí, které je vyhodnocováno v úloze TaskButtonRead. Po detekci stisknutí tlačítka rotačního voliče je k dalšímu zpracování předán pomocí řady xQueueKeyoboard znak '-' odpovídající průchodu tabulkou přednastavených kroků. Zákmity tlačítka jsou omezeny zablokováním funkce po dobu 450 ms po předešlém stisknutí a vstup je jinak vyhodnocován každých 50 ms. [19]

3.2.3.3. Ovládání z počítače

Deska Arduino podporuje odesílání a příjem textových zpráv přes sériové rozhraní USB. Ze samotné vývojové aplikace pro Arduino pro PC je možné posílat a přijímat znaky přes sériové rozhraní. Z hlediska používání bude ale lepší vytvořit grafické uživatelské prostředí, které umožní volbu dané funkce a odeslání příslušného znaku, namísto odesílání znaku přímo uživatelem (viz Příloha A).

Pro tuto aplikaci byl zvolen jazyk Java s použitím knihovny AWT [11] pro vytvoření grafického prostředí a knihovny RXTX [13] pro správu sériové komunikace (viz Příloha B). Po spuštění aplikace dojde k detekci všech dostupných sériových portů počítače. Uživatel vybere z nabídky port, ke kterému je připojena deska Arduino a dojde k navázání spojení. Aktivují se tlačítka se stejným rozložením, jako má již popsaná klávesnice. Po stisknutí tlačítka je přes sériové rozhraní odeslán příslušný znak. Pod tlačítky je umístěna konzole, kde je uveden záznam odeslaných a přijatých zpráv (viz Obrázek 46).



Obrázek 46 - Ovládací okno uživatelského rozhraní v PC

Vstup z počítače je v Arduinu vyhodnocen v úloze TaskDigitalRead(). Pokud jsou dostupná sériová data, dojde k převzetí semaforu pro sériovou komunikaci, vyčtení přijatých dat a odeslání znaku do řady xQueueKeyboard.

3.2.4. Vyhodnocení vstupu

K vyhodnocení vstupů ze všech periferií dochází v úloze TaskStiskKlavesy s prioritou 3. Po spuštění programu lze stisknutím tlačítka Pásmo (znak ‘k’) přepnout modul do režimu VHF, jinak program naběhne v režimu UHF. Než začne samotné vyhodnocování vstupů, načtou se základní proměnné pro odpovídající pásmo.

Poté čeká úloha na vstup z řady xQueueKeyboard. Přijatý znak je vyhodnocen s ohledem na aktuální mód provozu. Mód provozu je uložen v globální proměnné xModes a pro zajištění stability dat vyčítán přes semafor xModesSemaphore do

dočasné proměnné xModesTemp (viz Obrázek 47). Každá změna dočasných dat je uložena do globální proměnné. Jednotlivé bity proměnné xModes značí: signály TONE, TEST, PTT, pásmo (0=UHF, 1=VHF), režim tabulky přednastavených frekvencí, indikace zadání dvou desetinných čárek, režim vlastního kroku, indikace začátku zadávání.

```
if (xSemaphoreTake(xModesSemaphore, (TickType_t) 5 ) == pdTRUE )
{ // prevzeti semaforu pro upravu modu, pokud je dostupny

      xModesTemp = xModes;  // nacteni aktualniho modu

      xSemaphoreGive(xModesSemaphore); // vraceni semaforu

}
```

Obrázek 47 - Přečtení aktuálního módu

Pouze určité kombinace módu a vstupního znaku jsou přípustné, pro ostatní kombinace dojde k vypsání chybové hlášky.

3.2.4.4. Signály TONE, TEST, PTT

Pro znaky ‘o’ (signál TONE), ‘e’ (signál TEST) a ‘p’ (signál PTT) dojde k přepnutí odpovídajícího bitu proměnné xModes a přepnutí logické hodnoty odpovídajícího výstupního pinu desky Arduino (viz Obrázek 48).

```
xModesTemp = xModesTemp ^ 0b10000000; // zmena modu

if ((xModesTemp & 0b10000000) == 0b10000000) {//kontrola modu

      digitalWrite(tonePin, LOW);} // zapsani nizke urovne

else {

      digitalWrite(tonePin, HIGH); // zapsani vysoke urovne

}
```

Obrázek 48 - Přepnutí signálu TONE

3.2.4.5. Volba kroku

Pokud je příznak zadávání prázdný a není zapnutý režim průchodu tabulkou frekvencí, je možné zapnout režim vlastního kroku (znak ‘=’). Uživatel potom může zadat vlastní krok v rozmezí 25 kHz až 10 MHz.

Je také možné volit přednastavené kroky z tabulky. Stisknutím tlačítka rotačního kodéru dojde k odeslání znaku ‘=’ a cyklickému průchodu kroky z tabulky (1, 2, 5 a 10 MHz). Pokud byl dříve zadán vlastní krok, je krok po stisknutí tlačítka kodéru resetován na 1 MHz.

3.2.4.6. Tabulka frekvencí

Pro každé pásmo je připravená tabulka přednastavených frekvencí. Režim průchodu tabulkou je přepnut znakem '+', pokud neprobíhá zadávání. Ihned dojde k nastavení naposledy zvolené pozice v tabulce. Další pozice se volí rotačním kodérem.

3.2.4.7. Posun o krok a průchod tabulkou frekvencí

Pro znaky '<' (posun dolů) nebo '>' (posun nahoru) je přípustné vyhodnocení, když neprobíhá zadávání frekvence (příznak zadávání je prázdný).

Když není aktivní režim frekvencí, dochází ke změně poslední zadané frekvence o zvolený krok. Kód poslední frekvence je převeden na frekvenci, ke které je přičten nebo odečten krok. Pokud nová frekvence spadá mimo pracovní pásmo, je vypsána hláška a posun je zrušen. V opačném případě dojde k odeslání nové frekvence pomocí řady xQueueFrekvence do úlohy TaskKoduj, které je potom udělen semafor xKodujSemaphore. Úloha TaskStiskKlavesy pak čeká na příchod kódu přes řadu xQueueKod. Kód je uložen pro další využití a zároveň odeslán přes řadu xQueueSPIData do úlohy TaskSendSPI k zápisu na paralelní rozhraní.

```
char prenos = 0;
for (char i = 5; i >= 0; i--) {
      dekodovano[i]=dekodovano[i]+dekodovano_krok[i]+prenos;
      if (dekodovano[i] > 9) {
            dekodovano[i] = dekodovano[i] - 10;
            prenos = 1;}
      else {
            prenos = 0;
      }
}
```

Obrázek 49 - Přičtení kroku k současné frekvenci

V režimu průchodu tabulkou frekvencí dojde k vyhodnocení pozice v tabulce, a pokud je posun možný, dojde k převodu nové frekvence na kód, jinak je vypsána chybová hláška.

3.2.4.8. Oprava zadávání

Pokud probíhá zadávání frekvence nebo kroku, je možné smazat poslední zadaný znak tlačítkem smazat (znak 's'). Tlačítkem zrušit (znak 'z') pak v jakémkoli režimu dojde k návratu k základnímu režimu zadávání frekvence.

3.2.4.9. Zadávání frekvence a kroku

Stisknutím tlačítek s číslicí lze podle zvoleného režimu zadávat frekvenci nebo krok. Hodnota je zadávána v řádu MHz. V programu je opatřeno vícenásobné stisknutí desetinné čárky a překročení povolené délky. Zadané znaky jsou uloženy v poli pro další zpracování.

Po stisknutí tlačítka MHz (‘M’) dojde k vyhodnocení zadané hodnoty. Nejdříve dojde k hledání desetinné čárky a převodu zadané hodnoty na kHz, aby mohla být hodnota uložena do pole o délce 6.

Pokud spadá zadaná hodnota dovnitř povoleného pásma, je uložena jako vlastní krok, případně v režimu zadávání frekvence odeslána k převodu na kód a zapsána na paralelní rozhraní.

3.2.5. Převod frekvence na kód

Filtr přijímá frekvenci jako kombinaci 16 bitů binárních hodnot (viz Tabulka 4).

Tabulka 4 - Binární kód

Bit	15	14	13	12	11	10	9	8
Frekvence [MHz]	200	100	80	40	20	10	8	4
Bit	7	6	5	4	3	2	1	0
Frekvence [MHz]	2	1	0.8	0.4	0.2	0.1	0.050	0.025

Úloha TaskKoduj s prioritou 4 čeká na umožnění k běhu semaforem xKodujSemaphore. Následně je vyčtena frekvence z řady xQueueFrekvence. Přijatá frekvence je převedena na kód postupným porovnáváním s hodnotou v tabulce a odečtením dané hodnoty od frekvence (viz Obrázek 50). Kód je pak zapsán do řady xQueueKod.

```
xKod = 0;
char polohad = 0;
char polohah = 2;
for (char k = polohad; k < polohah; k++) {
      xKod = xKod << 1;
      if (cFrekvence[0] >= tab[k]) {
            cFrekvence[0] = cFrekvence[0] - tab[k];
            xKod++;
}}
```

Obrázek 50 - Převod řádu stovek MHz na kód

3.2.6. Odeslání kódu na paralelní rozhraní

V úloze TaskSendSPI dochází k odeslání kódu na paralelní rozhraní. Úloha má nejvyšší prioritu 6, ale je v zablokovaném stavu, dokud nejsou v řadě xQueueSPIData dostupná data, která jsou poté pomocí knihovny SPI odeslána. Následně je odeslán impuls, který zajistí přepis dat z posuvného registru sériově-paralelním převodníku do jeho výstupního registru.

```
SPI.beginTransaction(SPISettings(100, LSBFIRST, SPI_MODE0));
delay(10);
SPI.transfer16(sReceivedValue);
delay(5);
digitalWrite(wrPin, HIGH);
delay(10);
digitalWrite(wrPin, LOW);
SPI.endTransaction();
```

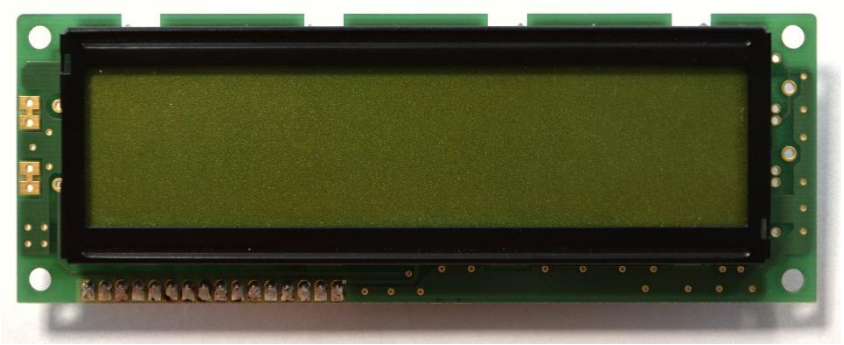
Obrázek 51 - Odeslání dat přes SPI

3.2.7. Zobrazení dat

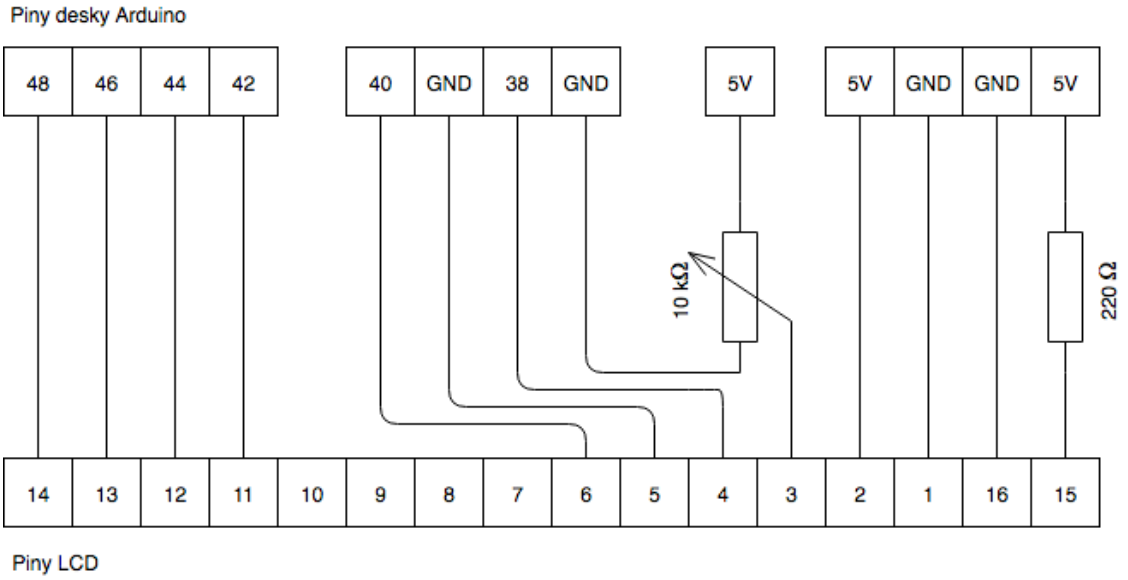
Všechny způsoby zobrazení dat mají prioritu 5, tedy druhou nejvyšší. Dojde tak k včasnému výpisu dat, ale jelikož jsou všechny úlohy pro zobrazení řízené událostmi, nedochází k neustálému blokování úloh s nižší prioritou.

3.2.7.10. LCD displej

Pro zobrazení aktuálního stavu zařízení byl vybrán LCD displej MC1602J-SYL (viz Obrázek 52) [20]. Tento displej disponuje 2 řádky po 16 znacích. Displej byl doplněn obvodovými prvky podle doporučení výrobce a připojen k desce Arduino podle následujícího schéma (viz Obrázek 53).



Obrázek 52 - LCD displej MC1602J-SYL



Obrázek 53 - Schéma zapojení LCD displeje

Pro zápis informace na displej byla použita knihovna pro Arduino LiquidCrystal.h [8]. Po namapování pinů displeje zajistí tato knihovna zápis znaků na displej.

Obsluhu displeje zajišťuje úloha TaskUpdateDisplay(). Ve své smyčce čeká úloha na řadu obsahující zprávu, která má být zobrazena na displeji (xQueueDisplay). Přijatá zpráva je zalomena po 16 znacích pro zobrazení na obou řádkách. Po uplynutí doby zobrazení zprávy je vyhodnocen aktuální mód ovládání a dojde k zobrazení odpovídajících informací, jako například právě zadaná frekvence. Aktuální stav může být na displej vypsán i bez zaslání stavové zprávy pomocí semaforu xDisplaySemaphore.

```
lcd.clear();
if ((xModesTemp & 0b00001010  ) == 0b00000010) {
      lcd.print("Zadej krok:");
      lcd.setCursor(0, 1);
      for (char i = 0; i < 8; i++) {
            lcd.print(zadano[i]);
      }
}
```

Obrázek 54 - Výpis na displej při zadávání kroku

3.2.7.11. Sériová komunikace

Pokud je zapnut výstup sériových dat (xSerialOutput == pdTRUE), je stavová zpráva zároveň odeslána pomocí řady xQueueSerialString do úlohy TaskDigitalWriteString, kde je zpráva po uvolnění sériové komunikace odeslána do počítače.

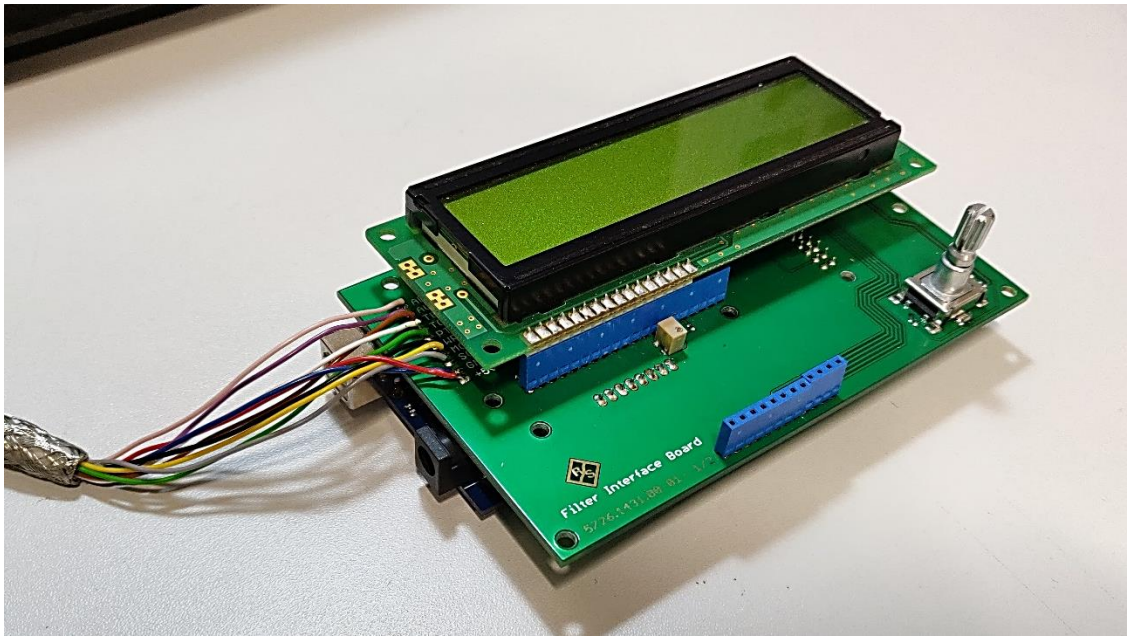
3.2.8. Signály Abstimmung a Auslösung

Při poklesu logické úrovně na pinu 20 dojde k přerušení, které je vyhodnoceno v ISR AbstInt. Pro omezení délky ISR dojde pouze k přepnutí priorit obsluhující úlohu TaskAbstInt. Tato úloha má za běžného chodu prioritu 0, tedy nejnižší možnou, a je stále v zablokovaném stavu. Po vyhodnocení přerušení dojde ke změně priority na prioritu 5. Sdílením stejné priority s úlohami pro zobrazování dat může stále docházet k výpisu informace o přerušení na displej, obsluhující úloha ale zablokuje všechny úlohy s nižší prioritou. Během trvání nízké úrovně signálů Abstimmung (= probíhá nastavování filtru) tak nedochází k vyhodnocování vstupů. Při každé změně logické úrovně na pinu 20 dochází k přerušení a k vyhodnocení, jakou má obsluhující úloha mít prioritu.

Pro signál Auslösung platí stejné vyhodnocení, kdy se vyhodnocuje pokles napětí na pinu 21. Tento signál nebude používán.

3.2.9. Shrnutí funkcí ovládání justovacího modulu

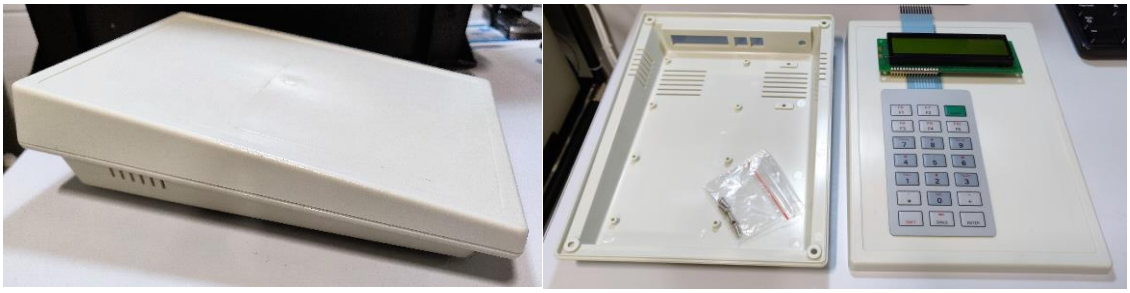
Shrňme si nyní funkce ovládání justovacího modulu. K desce Arduino jsou připojeny klávesnice, displej, rotační volič, počítač přes USB rozhraní a paralelní rozhraní. Program běžící na desce Arduino obstará vyhodnocení vstupu z periferií. Po naběhnutí programu je možné tlačítkem PÁSMO změnit pracovní pásmo z UHF na VHF. Poté lze zadat frekvenci v MHz. Pouze frekvence spadající do pracovního pásma je přijata. Dále lze procházet tabulkou kroků nebo zadat vlastní krok a právě zadanou frekvenci o vybraný krok měnit. K dispozici je i průchod tabulkou specifických frekvencí. Pokud právě probíhá ladění filtru na novou frekvenci, je vyhodnocování vstupu deaktivováno. Nová frekvence je převedena na specifický binární kód a odeslána sériově do paralelního rozhraní.



Obrázek 55 – Ovládání justovacího modulu

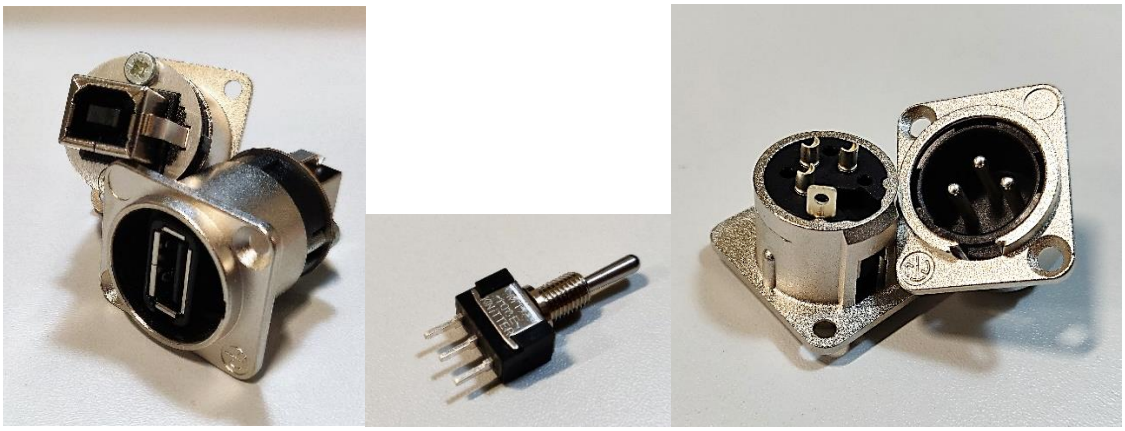
3.2.10. Fyzická realizace ovládání justovacího modulu

Pro umístění ovládání justovacího modulu byla vybrána panelová krabička COMBIPLAST CP-15-37 (viz Obrázek 56). Jelikož se jedná o univerzální krabičku, bude nutné nechat vyfrézovat otvory pro LCD displej, rotační kodér a kabel klávesnice spolu s otvory pro uchycení vnitřních částí modulu a konektory.



Obrázek 56 - Krabička s možným rozložením periferií

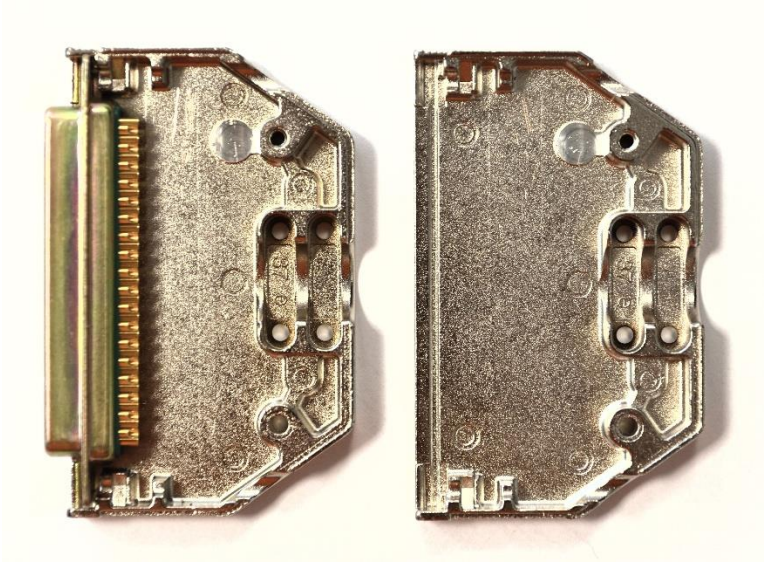
Mimo napájení +5V pro samotný justovací modul, které bude přivedeno přes USB-A/USB-B průchodku, je nutné přivést i napájení +28V pro ovládaný filtr. Napájení filtru je potom potřeba přivést na paralelní rozhraní buď na piny +28V a GND2, nebo na piny +BATT A -BATT. Pro přepínání mezi těmito větvemi bude do krabičky umístěn třístavový přepínač a do paralelního rozhraní povede vedle deseti-žilového kabelu se signálovými vodiči i čtyř-žilový kabel s napájecími vodiči. Obě napájecí větve používají stejná napětí, stačí tak jediný zdroj +28V, kterým se bude v panelu přepínat mezi větvemi +28V a BATT.



Obrázek 57 - Průchodka USB, přepínač napájecího napětí a konektor +28V napájení filtru

3.3. Paralelní rozhraní

Paralelní rozhraní musí zajistit přivedení kódu frekvence a ostatních signálů na piny konektoru D-SUB (viz 2.1). Dále je potřeba zajistit přívod napájecího napětí filtru +28V nebo BATT+/BATT- na tento konektor. Celé paralelní rozhraní by se mělo vejít do krytu konektoru.

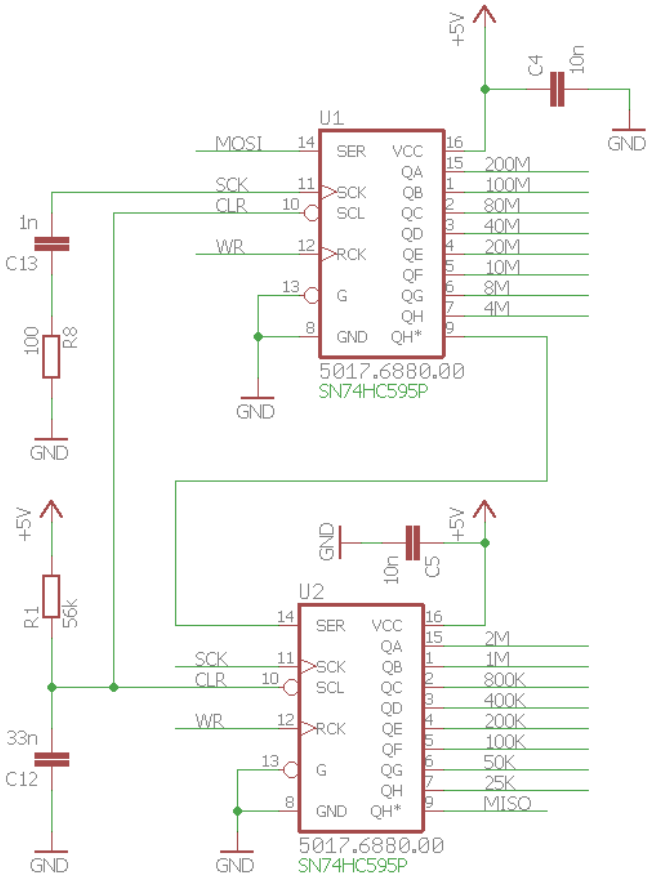


Obrázek 58 - Konektor a kryt

3.3.1. Převod kódu frekvence

Kód frekvence je odeslán pomocí sériového rozhraní standardem SPI. Se změnou hodinového signálu SCK se odesílá další znak na vodič signálu MOSI.

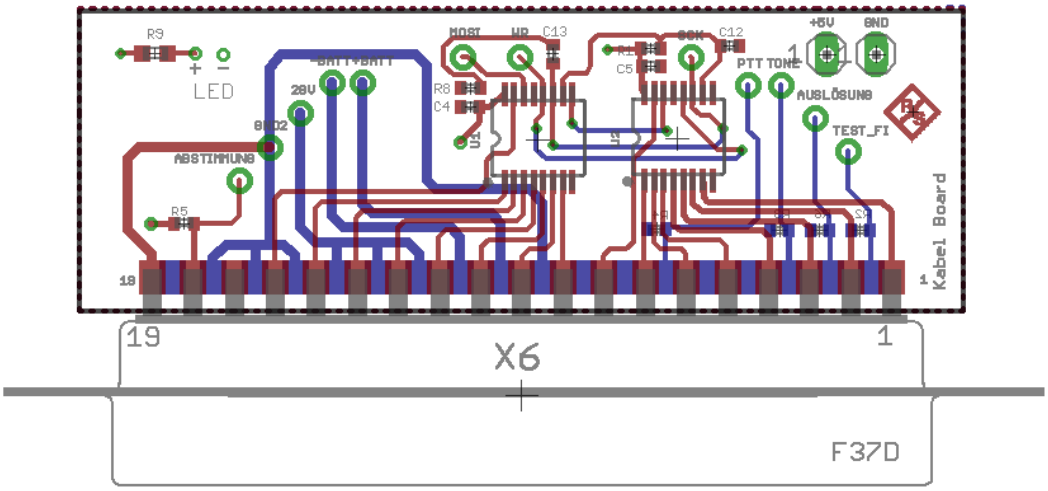
Pro převod sériového signálu na paralelní byl vybrán obvod SN74HC595P od firmy TI. [17] Jedná se o 8-bitový posuvný registr s 3-stavovým výstupem. Obvod také disponuje sériovým výstupem. Je tak možné zařadit 2 tyto součástky do série a zajistit 16-bitový paralelní výstup (viz Obrázek 59 nebo Příloha F). Přivedením signálu WR se hodnoty uložené v posuvných registrech přivedou na výstupní piny.



Obrázek 59 - Schéma zapojení sériově-paralelních převodníků

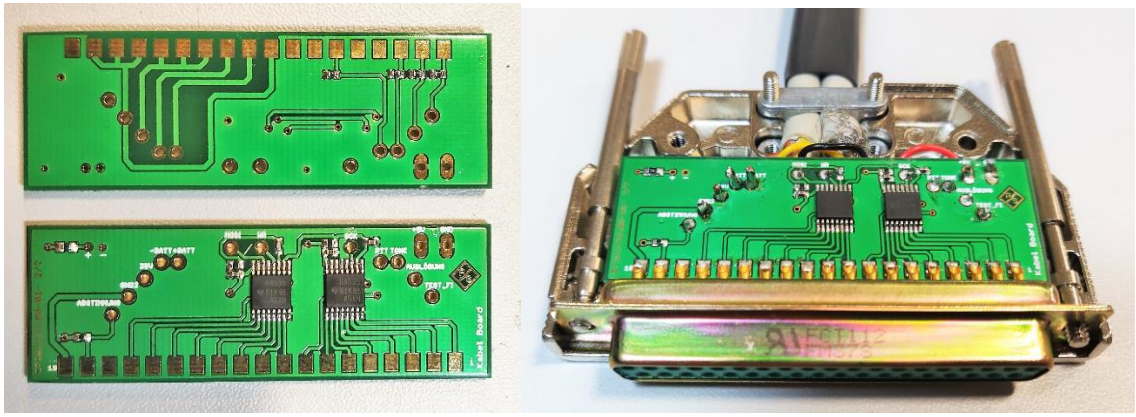
3.3.2. Kabel Board

Posuvné registry a obvody vstupního filtru a napájení je nutné umístit dovnitř krytu D-SUB konektoru (viz Obrázek 58). Rozměry desky tak můžou být maximálně 59x20 mm.



Obrázek 60 - Návrh desky pro paralelní rozhraní

Proto byla navržena deska plošných spojů s pojmenováním Kabel Board pro umístění těchto součástek. Konektor D-SUB bude umístěn tak, že jedna řada pinů bude přípájena na vrchní stranu desky a druhá řada na spodní stranu (viz Obrázek 60 a Obrázek 61 nebo Příloha G).



Obrázek 61 - Kabel Board

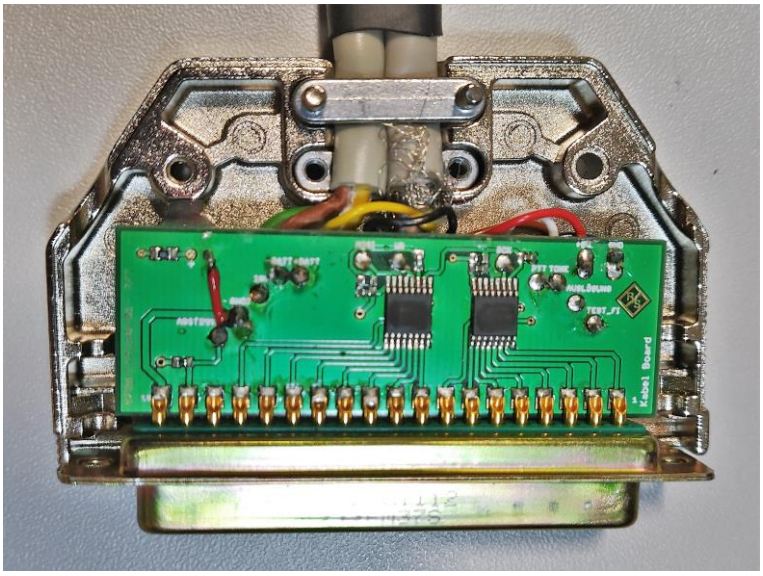
3.4. Ověření funkčnosti justovacího modulu

Před připojením k filtru bylo nutné ověřit funkčnost justovacího modulu. Všechny vstupní periferie byly vyzkoušeny jak jednotlivě, tak i při používání více vstupů najednou. Při otáčení rotačního kodéru dochází při rychlém otočení k přeskakování fáze signálu. Tato chyba je minimalizována zpracováním vstupu v ovládacím programu, kdy v takovém případě dojde k odmítnutí vstupu.

Reakce programu jsou včasné a přesné, nepovolené vstupy pro daný režim jsou správně ignorovány a dochází k vypsání chybové hlášky. Zobrazení dat je také plně funkční, ať už na displeji nebo v GUI v počítači. Na paralelním rozhraní byla ověřena správná funkce výstupních signálů TONE, TEST a PTT. Frekvence ve formě binárního kódu je také přivedena na odpovídající piny konektoru. Pokles úrovně signálu na pinu Abstimmung způsobí požadované přerušení programu spolu se zobrazením informace o přerušení.

Po ověření funkčnosti všech signálů byl justovací modul vyzkoušen přímo k nastavování elektromechanického filtru. Napájecí napětí +5V a +28V bylo zajištěno programovatelným zdrojem RS HMP4030. Justovací modul bude zajišťovat ovládání 2 typů filtrů, které používají rozdílné typy konektorů. K dispozici byl pouze filtr s kulatým konektorem, bylo proto potřeba použít kabelovou redukci. Tato redukce nemá zapojené signály TONE a PTT, nemohla proto být ověřena jejich funkčnost.

Při prvním zapojení justovacího modulu k filtru došlo k nastavení aktuální frekvence na filtr, ale při další změně frekvence se už filtr nenastavoval. Po krátkém hledání chyby bylo zjištěno, že filtr nemá propojenou zem příslušnou logickým signálům se zemí napájení. Do Kabel Boardu byla proto přidána propojka země napájení Arduina se zemí napájení filtru +28V (viz Obrázek 62).



Obrázek 62 - Úprava Kabel Boardu pro spojení zemí

Po této úpravě již vždy docházelo k nastavení filtru ihned po změně frekvence v justovacím modulu. Správně byl také přijímán signál Abstimmung, kdy během ladění filtru došlo k vypsání informace na displej a zablokování vstupních periferií. Byla ověřena i funkčnost signálu TEST, při jehož aktivaci bylo zablokováno ladění filtru. U justovacího modulu tak byly úspěšně otestovány všechny dostupné funkce.

Obrázek 63 - Testování funkčnosti justovacího modulu

3.5. Možná rozšíření modulu

Ačkoliv po umístění do krabičky bude modul plně funkční, je potřeba zlepšit snadnost používání.

Napájecí kabely +5V i +28V budou připojeny do krabičky přes konektory, kabel vedoucí k paralelnímu rozhraní je ovšem napevno spojený s Interface Boardem. Pro snadnější manipulaci by bylo vhodné paralelní rozhraní také zapojit do krabičky přes nějaký konektor.

Dále použití signálů TONE, TEST a PTT je sice indikováno na displeji, pro větší přehlednost by ale bylo dobré umístit na přední panel krabičky informační diody, které by rozsvícením značily použití těchto signálů.

Také bude potřeba upravit tabulku přednastavených frekvencí. V současné verzi programu slouží frekvence v tabulce pouze k otestování funkčnosti. Po konzultaci se zkušebním technikem bude do tabulky uložena často používaná sekvence frekvencí, aby byla tato funkce více užitečná.

Do budoucna se počítá s integrací tohoto justovacího modulu s automatickým měřícím systémem řízeným z počítače. Justovací modul nyní podporuje základní ovládání z počítače formou posílání textových řetězců přes sériové rozhraní USB, spojení tohoto modulu a automatického měřícího systému ovšem bude vyžadovat určité změny.

4. Závěr

Cílem práce bylo navrhnout a realizovat justovací modul pro kmitočtový filtr. Tento modul by měl umožňovat přímé zadání frekvence, změnu zadané frekvence o krok, volbu kroku z tabulky nebo zadaní vlastního kroku a průchod tabulkou přednastavených frekvencí.

Justovací modul byl rozdělen na dvě části. Část ovládání justovacího modulu umožňuje vyhodnocení vstupu z periferií a PC a zajišťuje požadované funkce zařízení včetně převodu frekvence na specifický binární kód. Modul je založen na vývojové sadě Arduino Mega 2560 s využitím operačního systému reálného času FreeRTOS. Pro připojení periferií byla navržena rozšiřující deska Interface Board, ke které jsou připojeny LCD displej, klávesnice a rotační kodér. Pro ovládání z PC bylo vytvořeno GUI v jazyce Java.

Binární kód je pomocí SPI sběrnice odeslán přes kabel do paralelního rozhraní, které zajišťuje sériově-paralelní převod pomocí obvodu typu 595 a přenos ostatních řídících signálů mezi filtrem a modulem. Pro toto rozhraní byla navržena deska Kabel Board, která se rozměry vejde do krytu konektoru, kterým se filtr připojuje.

Funkčnost modulu byla nejdříve otestována bez připojení filtru. Ovládací program vyhodnocuje vstupy uživatele přesně a včasně. Nepovolené vstupy jsou ošetřeny chybovou hláškou a dále ignorovány. Zadaná frekvence je převedena na kód a tento kód je přítomen na příslušných pinech konektoru. Stejně tak byla ověřena přítomnost výstupních signálů modulu i detekce vstupních signálů.

Následně byl modul otestován k nastavení kmitočtového filtru. Po drobné úpravě Kabel Boardu byla ověřena plná funkčnost justovacího modulu ve spojení s filtrem. Po dokončení krabičky tento justovací modul nahradí stávající řešení při kalibracích elektromechanických filtrů.

5. Seznam literatury

[1] Rohde&Schwarz, „R&S®M3SR FD221 UHF Automatic Filters and Multicouplers," 2018. [Online]. Available: https://www.rohde-schwarz.com/us/product/m3sr-fd221-productstartpage_63493-11488.html.

[2] Parallax Inc., „BASIC Stamp 2 Microcontroller Module," 2018. [Online]. Available: https://www.parallax.com/product/bs2-ic.

[3] NXP, „LPC1756," 2017. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/LPC1759_58_56_54_52_51.pdf.

[4] Coridium Corporation, „SuperPRO - LPC1756," 2018. [Online]. Available: https://www.coridium.us/coridium/shop/boards/bd25-superpro.

[5] Picaxe, „PICAXE-20 Project Board," 2018. [Online]. Available: http://www.picaxe.com/Hardware/Project-Boards/PICAXE-20-Project-Board/.

[6] D. K. Fisher a P. J. Gould, „Open-Source Hardware Is a Low-Cost Alternative for Scientific Instrumentation and Research," *Modern Instrumentation,* sv. 1, č. 2, pp. 8-20, 2012.

[7] Arduino, „Arduino Mega 2560 rev3," 2018. [Online]. Available: https://store.arduino.cc/arduino-mega-2560-rev3.

[8] Arduino, „LiquidCrystal Library," 2018. [Online]. Available: https://www.arduino.cc/en/Reference/LiquidCrystal.

[9] HITACHI, „Sparkfun - Hitachi HD447780U," 1999. [Online]. Available: https://www.sparkfun.com/datasheets/LCD/HD44780.pdf.

[10] M. Stanley a A. Brevig, „Arduino Playground Keypad," 2018. [Online]. Available: http://playground.arduino.cc/Code/Keypad.

[11] Oracle, „Package java.awt," 2018. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html.

[12] C. Hock-Chuan, „Java Programming Tutorial," 2018. [Online]. Available: https://www3.ntu.edu.sg/home/ehchua/programming/java/J4a_GUI.html#zz-2.1.

[13] T. Jarvi, „RXTX FAQ," 2007. [Online]. Available: http://rxtx.qbang.org/wiki/index.php/FAQ.

[14] R. Barry, „Mastering the FreeRTOS Real Time Kernel - A Hands-On Guide," 2016. [Online]. Available: https://www.freertos.org.

[15] P. Skalický, Přístrojové aplikace mikropočítačů, Vydavatelství ČVUT, 2004.

[16] P. Máša, „Materiály k přednáškám předmětu MAM," 2018.

[17] Texas Instruments, „SN54HC595,“ 2017. [Online]. Available: http://www.ti.com/lit/ds/symlink/sn54hc595.pdf.

[18] Bourns, „PEC11S Series - 12 mm SMD Incremental Encoder,“ 2018. [Online]. Available: http://www.bourns.com/data/global/pdfs/PEC11S.pdf.

[19] Arduino, „Reading Rotary Encoders,“ 2018. [Online]. Available: https://playground.arduino.cc/Main/RotaryEncoders.

[20] Palm Technology Co., LTD., „PMC1602J-SYL,“ [Online]. Available: http://www.gst-lcd.com/spec/PMC1602J-SYL.pdf.

6. Seznam příloh

Příloha A – Grafické uživatelské prostředí pro PC

Umístění: //GUI/GUI.jar

Příloha B – Zdrojový kód GUI – projekt v programu Eclipse

Umístění: //pcGUI/

Příloha C – Schéma rozšiřující desky pro Arduino

Umístění: //interface_board_sch.png

Příloha D – Návrh rozšiřující desky pro Arduino

Umístění: //interface_board_brd.png

Příloha E – Zdrojový kód programu pro Arduino

Umístění: // rizeni_FreeRTOS_v01 / rizeni_FreeRTOS_v01.ino

Příloha F – Schéma desky paralelního rozhraní

Umístění: // kabel_board_sch.png

Příloha G – Návrh desky paralelního rozhraní

Umístění: // kabel_board_brd.png

Příloha H – Obrázky

Umístění: //Obrázky/