Česká zemědělská univerzita v Praze

Technická fakulta

Katedra využití strojů

Přístrojová deska pro studentskou formuli CULS

Semestrální práce

Autor: **Tomáš Křeček**

Vedoucí práce: doc. Ing. Stanislava Papežová, CSc.

2024

OBSAH

1 Zadání

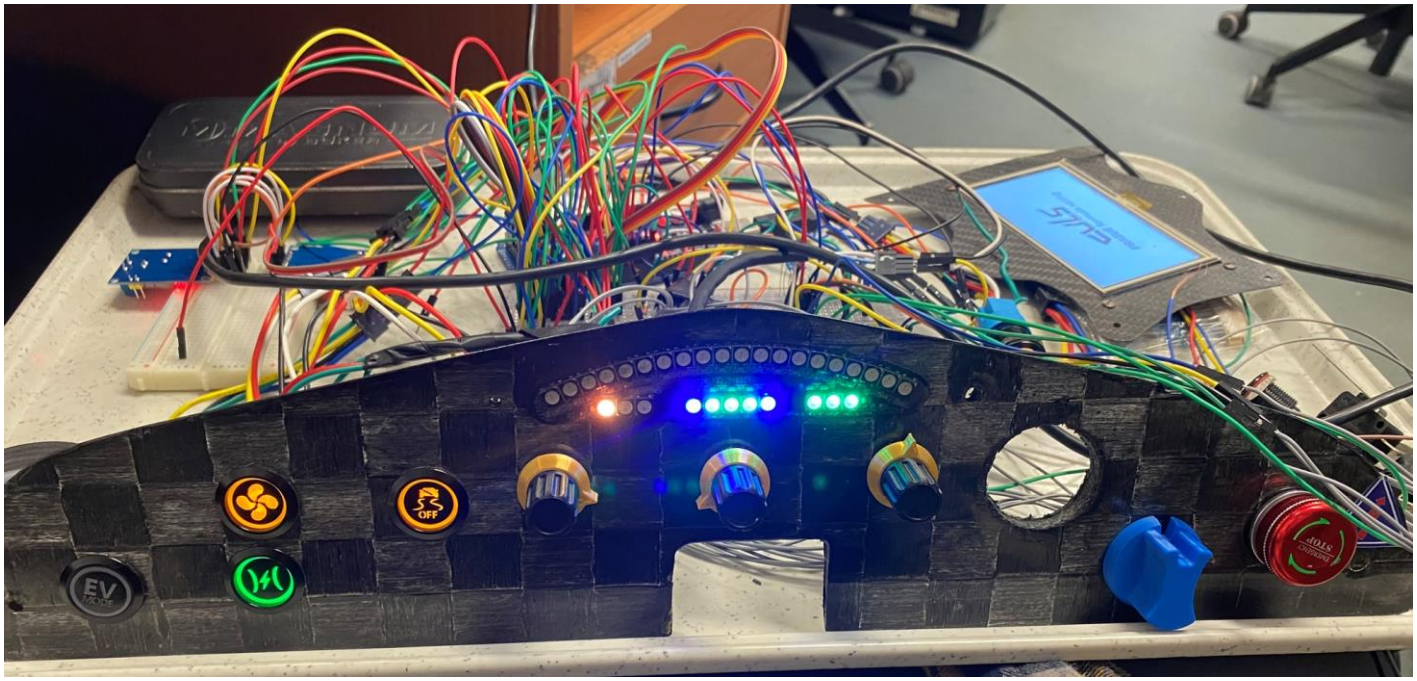
Přístrojová deska

Deska obsluhuje periferní prvky na přístrojové desce a volantu. Sbírá data z prvků a odesílá je na CANBUS. Z CANu také přichází data, která jsou interpretována na ledky a displej Nextion (bez použití knihovny). Tlačítka jsou oddělena od MCU pomocí optočlenů.

Platforma: ATmega32U4, MCP2515, MCP2551, WS2812B, PS2801-1-F3-A

- 5x Tlačítko na CANBUS tak přímo na konektor
- 3x Potenciometr na CAN
- 1x čtyř polohový přepínač (mody auta) na CAN
- 1x Start tlačítko CAN a přímo na konektor
- 2x pádlo z volantu CAN a přímo konektor (vyřešit na jajé to bude zemi)
- 2x tlačítko z volantu CAN a přímo konektor (vyřešit na jajé to bude zemi)
- 1x UART pro displej
- 1x PWM pin pro led pásek
- 5x výstup pro řízení podsvícení tlačítek (možno přepnout buď z programově nebo prostě svícení po zmáčknutí)

2 Foto přístrojové desky



3 Úvod

Cílem tohoto projektu je vytvořit přístrojovou desku, která obsluhuje různé periferní prvky na přístrojové desce a volantu. Systém bude sbírat data z tlačítek, přepínačů, potenciometrů a dalších prvků, a tato data odesílat přes CANBUS. Z CANBUS budou přijímána data, která budou interpretována pomocí LED pásku a UART displeje Nextion.

4 Specifikace

- Platforma: Arduino Mega 2560 programer – ATmega32U4
- Komunikace: CANBUS (použití MCP2515), UART
- Vstupy a výstupy:
    - **4x podsvícené tlačítko přístrojová deska** (CANBUS a přímo na konektor)
    - **1x podsvícené tlačítko Kill switch přístrojová deska** (CANBUS a přímo na konektor)
    - **3x potenciometr přístrojová deska** (CANBUS)
    - **1x deseti polohový přepínač přístrojová deska** (módy auta, CANBUS)
    - **1x start tlačítko přístrojová deska** (CANBUS a přímo na konektor)
    - **2x pádlo pro řazení na volantu** (volant, CANBUS a přímo na konektor)
    - **2x tlačítko z volantu** (CANBUS a přímo na konektor)
    - **1x Nextion display NX 4827T043_011UART Na volantu** (UART pro komunikaci s displejem)
    - **1x PWM výstup pro zobrazování otáček auta** (pro řízení LED pásku)

5 Použitý hardware

5.1 Použité komponenty

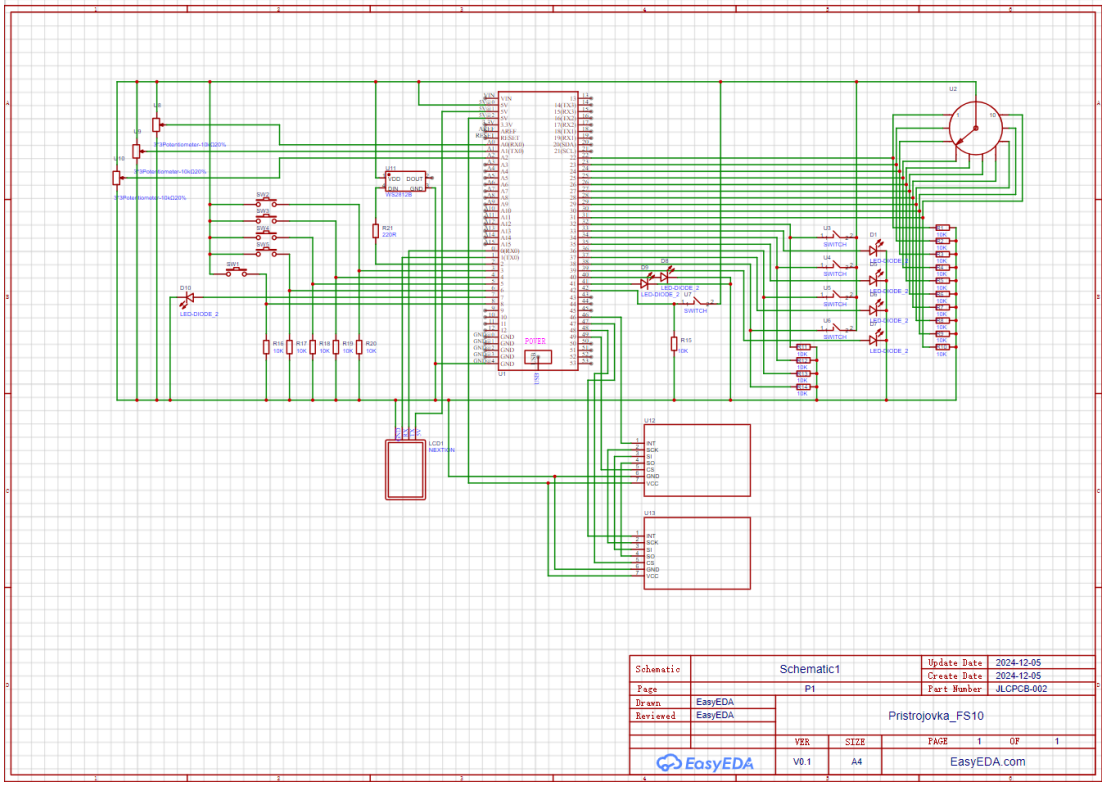
- **Arduino Mega 2560**: Hlavní mikrokontroler s dostatečným počtem digitální a analogových pinů a podporou CANBUS přes externí moduly. Bez konkurenční cena a spolehlivost.
- **MCP2515 CANBUS moduly**: Implementace sběrnice CANBUS. Pomocí Pinů, které podporuje Arduino MISO, MOSI, CSK, CS, IM. Tyto moduly byli použity hned dva z důvodu, že studentská formule má již dva CANy jede Prioritní, na kterém komunikuje ECU jednotka a stěžejní řídící prvky formule. Druhý sekundární CAN složí k posílání telemetrických dat a dalších prvků k ovládání auta.
- **LED pásek WS2812B**: Jedná se o programovatelný RGB pásek, který je upraven do tvaru vhodného použití pro zobrazování světelné signalizace při řízení formule. Je nepájen 5 V a používá 31 jednotlivých ledek. Ovládán je pomocí knihovny fastLED.h a požívá PWM signál.
- **Display Nextion NX 4827T043_011UART:** Jedná se o display využívající komunikaci přes UART piny RX0/TX0. Zde na tomto display jsou zobrazovaná telemetrická data jako je zařazený rychlostní stupeň, otáčky motoru, rychlost formule, tlaky a teploty vody, paliva, oleje a sání. Dále zde můžeme najít Napětí baterie a další telemetrická data. Tento display je dotykový, a proto je vhodné ho používat na volantu, aby pilot mohl kdykoliv vyčíst potřebné informace.

5.2 Použité součástky

- **Rezistory – 220R:** Pro regulaci napětí před diodami a světelným podsvícením
- **Rezistory – 10k:** Zařizující pull down funkci tlačítek.
- **Potenciometry – 10k:** Lineární rezistor pro nastavení analogových hodnot
- **Deseti polohový přepínač:** Kruhový přepínač mezi 10 polohami
- **Kill switch tlačítko:** Speciální tlačítko pro bezpečné vypnutí elektrického obvodu formule.

- **Podsvícené přepínače:** Spínání digitální vstupů – přepínání různých funkcí na formuli:
    - Odsávání z podlahy
    - Chlazení
    - Elektrické spínání Boost
    - EV mód
- **Tlačítka do řadící páčky na volantu:** Prodloužené tlačítko o plechový jazýček
- **Podsvícené tlačítko pro start motoru:** Automobilové tlačítko připojené přímo do jednotky motoru
- **Tlačítka na volantu:** Běžná tlačítka na volantu pro pohodlné přepínání různých funkcí během jízdy bez nutnosti pustit volant.

6 Schéma zapojení



7 Popis softwaru

Software zajišťuje komplexní řízení a komunikaci mezi jednotlivými moduly automobilového systému. Obsahuje správu LED pásku pro vizualizaci dat, zpracování vstupů z potenciometrů, tlačítek a přepínačů, a aktualizaci displeje Nextion. Dále integruje komunikaci přes CAN bus, umožňuje odesílání a přijímání dat v reálném čase a poskytuje nástroje pro diagnostiku a ladění. Celý systém je modulární, přehledný a připravený na rozšíření.

7.1 Header files

V hlavičkových souborech jsou obsaženy popisy jednotlivých modulů programu, což zajišťuje jejich přehlednost a snadné pochopení. Hlavičkové soubory zároveň obsahují všechny klíčové definice (#define), jako jsou nastavení pinů, ID zpráv a další konfigurační parametry, které umožňují snadnou editaci a přizpůsobení celého systému bez nutnosti zasahovat do implementačních souborů.

```
#pragma once
//// ==================INCLUDES==================
#include <FastLED.h>

//// ==================DEFINES==================
// FastLED nastavení LED Pásku
#define LED_PIN     2
#define NUM_LEDS    31
#define BRIGHTNESS  100
#define LED_TYPE    WS2812B
#define COLOR_ORDER GRB

// Rozsahy a barvy pro RPM
#define RPM_GREEN_START 500
#define RPM_GREEN_END   6000
#define RPM_YELLOW_START 6001
#define RPM_YELLOW_END   11000
#define RPM_RED_START    11001
#define RPM_LIMITER      12500

// Rozsahy a barvy pro teplotu vody
#define WATER_COLD 40
#define WATER_NORMAL 85
#define WATER_HOT 95

// Rozsahy a barvy pro teplotu oleje
#define OIL_COLD 50
#define OIL_NORMAL 120
#define OIL_HOT 130
#define OIL_LIMIT 140

//// ==================INICIALIZATION==================
// Inicializace pomocí knihovny FastLED.h
void ledStripeInit();

//// ==================FUNKCTIONS==================
// Funkce pro míchání barev
CRGB blendColors(CRGB color1, CRGB color2, uint8_t blendAmount);

//Rosvicí led kompletní funkce pro RPM, GEAR, WATERTEMP, OILTEMP
void setLEDs(int rpm, int gear, int waterTemp, int oilTemp);
```

7.2 Main.cpp

Soubor main.cpp slouží jako hlavní řídicí část programu, která propojuje všechny moduly systému. Obsahuje:

- **Inicializaci systému**: LED pásek, CAN bus, potenciometry, přepínače a tlačítka.
- **Testovací režim**: Simulace hodnot pro LED pásek.
- **Čtení vstupů**: Potenciometry, tlačítka, 10polohový přepínač, startovací tlačítko a kill switch.
- **Zpracování dat**: Načítání dat z CAN bus a jejich odesílání.
- **Ovládání výstupů**: Aktualizace LED pásku a displeje Nextion. Soubor tak zajišťuje komunikaci a synchronizaci mezi jednotlivými moduly systému.

```
//// ===============INCLUDES================
#include "led_stripe.h"
#include "can_bus.h"
#include "sw_pot_wheel_data.h"
#include "wheel_disp.h"

//// ==================DEFINES==================
bool testing_mode = true;
#define SERIAL_MONITOR_BAUD_RATE 9600
#define POWER_UP_SAFETY_DELAY 3000

//// ==================VARIABLES==================
int rpm;
int gear;
int waterTemp;
int oilTemp;

//// ===============SETUP================
void setup() {
  delay(POWER_UP_SAFETY_DELAY);  // Power-up safety delay

  Serial.begin(SERIAL_MONITOR_BAUD_RATE);  // Inicializace Serial Monitoru

  ledStripeInit();  // Inicializace led pásku

  canBusInit(); // Inicializace CAN bus

  swPotModeInit(); // Inicializace tlačítek, potenciometrů, 10 polohového přepínače, killswitche, startovacího tlačítka, pádla pod volantem a volantových
}
```

```
//// ===============LOOP================
void loop() {
  if (testing_mode) simulateValues();  // Testovací funkce pro Led pásek
  else setLEDs(rpm, gear, waterTemp, oilTemp); // Funkce pro zobrazování hodnot z auta na LED Pásek

  readPotentiometerData();  // Čtení potenciometrů

  readTenStateSwitch(); // Získání aktuálního stavu 10polohového přepínače

  readStartSwitch(); // Funkce pro zpracování startovacího tlačítka

  processFourBoardSwitches(); // Načtení dat ze 4 tlačítek z přístrojové desky

  processKillSwitch(); // Funkce pro  čtení kill switche

  wheelSwitchesRead(); // Čtení stavů dvou tlačítek z volantu a řadících pádel

  serialPrintSwPotWheel(); // Tisk pomocných proměnných do seriového monitoru

  sendAllData(); // Funkce pošle na can všechna data získaná z přístrojové desky

  readCANMessages(); // Načte všechny data z canu a uloží do promněných

  updateNextionDisplay();
}
```

7.3 Led_stripe.cpp

Soubor led_stripe.cpp zajišťuje ovládání LED pásku pomocí knihovny FastLED. Obsahuje funkce pro vizualizaci otáček motoru, převodových stupňů, teploty vody a oleje pomocí barevného kódování, gradientů a blikání. Hlavní funkce setLEDs aktualizuje LED pásek na základě aktuálních dat, zatímco simulateValues umožňuje simulaci realistického chování systému v testovacím režimu. Tento modul je klíčový pro vizuální zpětnou vazbu systému a podporuje snadné přizpůsobení efektů.

```cpp
// ====================INCLUDES====================
#include <led_stripe.h>


CRGB leds[NUM_LEDS];

// Časovač pro efekt odrážení
unsigned long previousMillis = 0;
const long interval = 250;  // Interval pro blikání LED (250 ms)
const long intervalRPM = 60;  // Interval pro blikání LED (250 ms)
bool blinkState = false;

// ====================INICIALIZATION====================
// Inicializace pomocí knihovny FastLED
void ledStripeInit(){
  FastLED.addLeds<LED_TYPE, LED_PIN, COLOR_ORDER>(leds, NUM_LEDS).setCorrection(TypicalLEDStrip);
  FastLED.setBrightness(BRIGHTNESS);
  FastLED.clear();
  FastLED.show();
}

// ====================FUNKCTIONS====================
// Funkce pro míchání barev
CRGB blendColors(CRGB color1, CRGB color2, uint8_t blendAmount) {···

void setLEDs(int rpm, int gear, int waterTemp, int oilTemp) {
  setRPM(rpm);
  setGear(gear);
  setWaterTemp(waterTemp);
  setOilTemp(oilTemp);
  //leds[21] = CRGB::Red;
  //leds[22] = CRGB::Black;
  // leds[23] = CRGB::Red;
  FastLED.show();
}
```

```
void setRPM(int rpm) {
  if (rpm < 12000) {
    int rpmIndex = map(rpm, 6000, 12000, 19, 0);
    for (int i = 19; i >= 0; i--) {
      if (i >= rpmIndex) {
        uint8_t blendAmount = map(i, 19, 0, 0, 255);
        leds[i] = blendColors(CRGB::Green, CRGB::Red, blendAmount);
      } else {
        leds[i] = CRGB::Black;
      }
    }
  } else if (rpm < 12300) {
    for (int i = 19; i >= 0; i--) {
      leds[i] = CRGB::Red;
    }
  } else if (rpm < 12500) {
    unsigned long currentMillis = millis();
    if (currentMillis - previousMillis >= intervalRPM) {
      previousMillis = currentMillis;
      blinkState = !blinkState;
    }
    CRGB color = blinkState ? CRGB::Red : CRGB::Black;

    for (int i = 19; i >= 0; i--) {
      leds[i] = color;
    }
  }
}
```

```
void setWaterTemp(int waterTemp) {
  // Reset all related LEDs
  leds[23] = CRGB::Black;
  leds[24] = CRGB::Black;
  leds[25] = CRGB::Black;
  leds[26] = CRGB::Black;
  leds[27] = CRGB::Black;

  if (waterTemp < 50) {
    leds[23] = CRGB::Blue;
    leds[27] = CRGB::Blue;
  } else if (waterTemp < 85) {
    leds[23] = CRGB::Blue;
    leds[24] = CRGB::Green;
    leds[25] = CRGB::Green;
    leds[26] = CRGB::Green;
    leds[27] = CRGB::Blue;
  } else if (waterTemp < 95) {
    leds[23] = CRGB::Blue;
    leds[24] = CRGB::Red;
    leds[25] = CRGB::Red;
    leds[26] = CRGB::Red;
    leds[27] = CRGB::Blue;
  } else {
    // Blink effect
    unsigned long currentMillis = millis();
    if (currentMillis - previousMillis >= interval) {
      previousMillis = currentMillis;
      blinkState = !blinkState;
    }
    CRGB color = blinkState ? CRGB::Red : CRGB::Black;
    leds[23] = color;
    leds[24] = color;
    leds[25] = color;
    leds[26] = color;
    leds[27] = color;
  }
}
```

```
void setOilTemp(int oilTemp) {
  if (oilTemp < 60) {
    leds[28] = CRGB::Blue;
    leds[29] = CRGB::Blue;
    leds[30] = CRGB::Blue;
  } else if (oilTemp < 120) {
    leds[28] = CRGB::Green;
    leds[29] = CRGB::Green;
    leds[30] = CRGB::Green;
  } else if (oilTemp < 130) {
    leds[28] = CRGB::Orange;
    leds[29] = CRGB::Orange;
    leds[30] = CRGB::Orange;
  }

  else if (oilTemp < 140) {
    leds[28] = CRGB::Red;
    leds[29] = CRGB::Red;
    leds[30] = CRGB::Red;
  } else if (oilTemp < 150) {

    unsigned long currentMillis = millis();
    if (currentMillis - previousMillis >= interval) {
      previousMillis = currentMillis;
      blinkState = !blinkState;
    }
    CRGB color = blinkState ? CRGB::Red : CRGB::Black;
    leds[28] = color;
    leds[29] = color;
    leds[30] = color;
  }
}
```

7.4 Sw_pot_wheel_data.cpp

Soubor sw_pot_wheel_data.cpp zajišťuje čtení a zpracování vstupů z potenciometrů, tlačítek, přepínačů, řadících pádel, startovacího tlačítka a kill switche. Obsahuje funkce pro inicializaci pinů, čtení hodnot a ovládání LED indikací. Data z těchto vstupů jsou zpracovávána pro další použití, přičemž jejich aktuální stavy lze ladit pomocí výpisu na sériový monitor. Tento modul je nezbytný pro správné propojení uživatelského ovládání se zbytkem systému.

```cpp
//// ==================INCLUDES==================
#include "sw_pot_wheel_data.h"


//// ==================VARIABLES==================
//Startovací tlačítko
bool softLedState = LOW;          // Stav softwarové LED
bool lastButtonState = LOW;       // Předchozí stav tlačítka
bool currentButtonState = LOW;    // Aktuální stav tlačítka
unsigned long lastDebounceTime = 0; // Čas poslední změny stavu tlačítka
const unsigned long debounceDelay = 50; // Zpoždění pro debounce

//Proměné potenciometrů
int potValue1 = 0;
int potValue2 = 0;
int potValue3 = 0;

//Stavy tlačítek na volantu a řadících pádel
int shiftUpState = 0;
int shiftDownState = 0;
int leveTlacState = 0;
int praveTlacState = 0;

int selectedPosition = -1;
String stateText = "-1";
String switchStates = "";
int arrSwitchs[numSingleSwitches];

//// ==================INICIALIZATION==================
void swPotModeInit(){ ···

//// ==================FUNKCTIONS==================
// Funkce pro čtení analogových potenciometrů
void readPotentiometerData(){
  potValue1 = analogRead(POTENTIOMETER_1_PIN);
  potValue2 = analogRead(POTENTIOMETER_2_PIN);
  potValue3 = analogRead(POTENTIOMETER_3_PIN);
}
```

```
//// ==================INICIALIZATION==================
void swPotModeInit(){
  // Nastavení pinů přepínače jako vstup
  for (int i = 0; i < numSwitchPins; i++) pinMode(switchPins[i], INPUT);

  // Nastavení pinů pro 4 přepínače na přístrojové desce a jejich LED
  for (int i = 0; i < numSingleSwitches; i++) {
    pinMode(switchInputs[i], INPUT);
    pinMode(ledOutputs[i], OUTPUT);
    digitalWrite(ledOutputs[i], LOW);
  }

  // Nastavení pro startovací tlačítko
  pinMode(START_BUTTON_PIN, INPUT);
  pinMode(START_BUTTON_LED_PIN, OUTPUT);
  digitalWrite(START_BUTTON_LED_PIN, softLedState);

  // Nastavení pro kill switch
  pinMode(KILL_SWITCH_BUTTON_PIN, INPUT);
  pinMode(KILL_SWITCH_LED_1_PIN, OUTPUT);
  pinMode(KILL_SWITCH_LED_2_PIN, OUTPUT);

  // Nastavení pro tlačítka na volantu a řadící pádla
  pinMode(SHIFT_UP_PIN, INPUT);
  pinMode(SHIFT_DOWN_PIN, INPUT);
  pinMode(LEFT_WHEEL_BUTTON_PIN, INPUT);
  pinMode(RIGHT_WHEEL_BUTTON_PIN, INPUT);
}
```

```
//// ==================FUNKCTIONS==================
// Funkce pro čtení analogových potenciometrů
void readPotentiometerData(){
  potValue1 = analogRead(POTENTIOMETER_1_PIN);
  potValue2 = analogRead(POTENTIOMETER_2_PIN);
  potValue3 = analogRead(POTENTIOMETER_3_PIN);
}

// Načtení dat ze 4 tlačítek z přístrojové desky
void processFourBoardSwitches() {
  String result = "";
  for (int i = 0; i < numSingleSwitches; i++) {
    if (digitalRead(switchInputs[i]) == HIGH) {
      digitalWrite(ledOutputs[i], HIGH);
      result += "Vyp " + String(i + 1) + ": Zap | ";
      arrSwitchs[i] = 1;
    } else {
      digitalWrite(ledOutputs[i], LOW);
      result += "Vyp " + String(i + 1) + ": Vyp | ";
      arrSwitchs[i] = 0;
    }
  }
  switchStates = result;
}

// Funkce pro  čtení
void processKillSwit
  int digitalRead(uint8_t pin)
  int buttonState = digitalRead(KILL_SWITCH_BUTTON_PIN);
  if (buttonState == HIGH) {
    digitalWrite(KILL_SWITCH_LED_1_PIN, HIGH);
    digitalWrite(KILL_SWITCH_LED_2_PIN, LOW);
  } else {
    digitalWrite(KILL_SWITCH_LED_1_PIN, LOW);
    digitalWrite(KILL_SWITCH_LED_2_PIN, HIGH);
  }
}
```

```
// Čtení stavů dalších tlačítek z volantu a pádel
void wheelSwitchesRead(){
  shiftUpState = digitalRead(SHIFT_UP_PIN);
  shiftDownState = digitalRead(SHIFT_DOWN_PIN);
  leveTlacState = digitalRead(LEFT_WHEEL_BUTTON_PIN);
  praveTlacState = digitalRead(RIGHT_WHEEL_BUTTON_PIN);
}

// Funkce pro zpracování startovacího tlačítka
void readStartSwitch() {
  bool reading = digitalRead(START_BUTTON_PIN);
  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }
  if ((mil                                    > debounceDelay) {
    if (re                                    ) {
      curr
      if (currentButtonState == HIGH) {
        softLedState = !softLedState;
        digitalWrite(START_BUTTON_LED_PIN, softLedState);
      }
    }
  }
  lastButtonState = reading;
}

// Pošle int data o startovacím tlačítku
int sendDataStarButton(){
  if (softLedState) return 1;
  else return 0;
}

// Načtení stavu z desetipolohového switche
void readTenStateSwitch(){
  selectedPosition = readSwitchPosition();
  stateText = (selectedPosition != -1) ? String(selectedPosition + 1) : "Neurčený";
}
```

bool currentButtonState
Aktuální stav tlačítka

```
// Funkce pro čtení polohy 10polohového přepínače
int readSwitchPosition() {
  for (int i = 0; i < numSwitchPins; i++) {
    if (digitalRead(switchPins[i]) == HIGH) {
      return i;
    }
  }
  return -1;
}

// Výpis do Serial Monitoru
void serialPrintSwPotWheel(){
  Serial.print("Pot 1: ");
  Serial.print(potValue1);
  Serial.print(" | Pot 2: ");
  Serial.print(potValue2);
  Serial.print(" | Pot 3: ");
  Serial.print(potValue3);
  Serial.print(" | Stav: ");
  Serial.print(stateText);
  Serial.print(" | ");
  Serial.print(switchStates);
  Serial.print("StartTlac: ");
  Serial.print(softLedState ? "Zap" : "Vyp");
  Serial.print(" | KillSw: ");
  Serial.print(digitalRead(KILL_SWITCH_BUTTON_PIN) == HIGH ? "Zap" : "Vyp");
  Serial.print(" |                                    ");
  Serial.print(shif                                    (const char [15])" | ShiftDown: "
  Serial.print(" | ShiftDown: ");
  Serial.print(shiftDownState);
  Serial.print(" | LeveTlac: ");
  Serial.print(leveTlacState);
  Serial.print(" | PraveTlac: ");
  Serial.println(praveTlacState);
}
```

7.5 Wheel_disp.cpp

Soubor wheel_disp.cpp zajišťuje komunikaci s displejem Nextion prostřednictvím UART. Obsahuje funkci updateNextionDisplay, která přiřazuje hodnoty proměnných systému (například tlak oleje, rychlost, AFR) k odpovídajícím komponentám na displeji a odesílá je jako příkazy. Funkce sendCommand odesílá jednotlivé příkazy na displej ve správném formátu, včetně ukončovacích bajtů. Tento modul umožňuje přehledné zobrazování systémových dat na displeji Nextion v reálném čase.

```cpp
//// ==================INCLUDES==================
#include "wheel_disp.h"

//// ==================VARIABLES==================
SoftwareSerial nextion(RX_PIN, TX_PIN);

//// ==================FUNKCTIONS==================
//Funkce která přiřazuje dannou promněnou k danné adrese na display
void updateNextionDisplay() {
  char buffer[32];

  sprintf(buffer, "%s=%.2f", NEXTION_WATER_PRESSURE, (double)canData.waterPressure);
  sendCommand(buffer);

  sprintf(buffer, "%s=%.2f", NEXTION_OIL_PRESSURE, (double)canData.oilPressure);
  sendCommand(buffer);

  sprintf(buffer, "%s=%.2f", NEXTION_THROTTLE_POS, (double)canData.throttlePosition);
  sendCommand(buffer);

  sprintf(buffer, "%s=%.2f", NEXTION_FUEL_PRESSURE, (double)canData.fuelPressure);
  sendCommand(buffer);

  sprintf(buffer, "%s=%.2f", NEXTION_INTAKE_PRESS, (double)canData.intakePressure);
  sendCommand(buffer);

  sprintf(buffer, "%s=%.2f", NEXTION_AFR, (double)canData.afr);
  sendCommand(buffer);

  sprintf(buffer, "%s=%.2f", NEXTION_SPEED, (double)canData.speed);
  sendCommand(buffer);

  sprintf(buffer, "%s=%.2f", NEXTION_BATTERY_LOW, (double)canData.batteryLow);
  sendCommand(buffer);

  sprintf(buffer, "%s=%.2f", NEXTION_BATTERY_HYBRID, (double)canData.batteryHybrid);
  sendCommand(buffer);
}
```

```cpp
// Funkce pro odeslání příkazu na displej
void sendCommand(const char* cmd) {
  nextion.print(cmd);        // Odeslat příkaz
  nextion.write(0xFF);       // Ukončovací bajty
  nextion.write(0xFF);
  nextion.write(0xFF);
}
```

7.6 Can_bus.cpp

Soubor can_bus.cpp zajišťuje komunikaci přes CAN bus pomocí knihovny MCP_CAN. Obsahuje funkce pro inicializaci dvou CAN kanálů (PriorityCAN a SecondaryCAN), odesílání dat získaných z potenciometrů, přepínačů a tlačítek, a zpracování přijatých zpráv. Funkce sendAllData odesílá data na CAN bus, zatímco processCANMessage zpracovává příchozí zprávy podle jejich ID a ukládá je do globální struktury CANData. Modul také obsahuje pomocné funkce, jako je sendCANMessage pro formátování a odesílání jednotlivých zpráv, a podporuje čtení CAN zpráv přes readCANMessages. Tento soubor hraje klíčovou roli v propojení řídicího systému se zbytkem automobilové infrastruktury.

```
//// ====================INCLUDES====================
#include "can_bus.h"

// ====================VARIABLES====================
MCP_CAN priorityCAN(PRIORITY_CAN_CS);   // Objekt pro PriorityCAN
MCP_CAN secondaryCAN(SECONDARY_CAN_CS); // Objekt pro SecondaryCAN

CANData canData; // Definice globální proměnné

// Stavové proměnné pro přerušení
//volatile bool priorityCANInterrupt = false; //NENÍ IMPLEMENTOVÁNO
//volatile bool secondaryCANInterrupt = false; //NENÍ IMPLEMENTOVÁNO

//// ====================INICIALIZATION====================
void canBusInit(){
    if (priorityCAN.begin(MCP_ANY, CAN_SPEED, CAN_CLOCK) == CAN_OK) {
    Serial.println("PriorityCAN initialized successfully");
    priorityCAN.setMode(MCP_NORMAL);
  } else {
    Serial.println("Error initializing PriorityCAN");
    while (1);
  }

  if (secondaryCAN.begin(MCP_ANY, CAN_SPEED, CAN_CLOCK) == CAN_OK) {
    Serial.println("SecondaryCAN initialized successfully");
    secondaryCAN.setMode(MCP_NORMAL);
  } else {
    Serial.println("Error initializing SecondaryCAN");
    while (1);
  }

  // Nastavení přerušení
  pinMode(PRIORITY_CAN_INT, INPUT);
  pinMode(SECONDARY_CAN_INT, INPUT);
}
```

```
//// ==================FUNKCTIONS==================
// Pošle všechna získaná data z přístrojové desky a volantu na CAN
void sendAllData(){
  sendCANMessage(secondaryCAN, CAN_ID_POT1, analogRead(POTENTIOMETER_1_PIN));
  sendCANMessage(secondaryCAN, CAN_ID_POT2, analogRead(POTENTIOMETER_2_PIN));
  sendCANMessage(secondaryCAN, CAN_ID_POT3, analogRead(POTENTIOMETER_3_PIN));
  sendCANMessage(secondaryCAN, CAN_ID_SWITCH_POSITION, readSwitchPosition());
  sendCANMessage(secondaryCAN, CAN_ID_SOFTWARE_SWITCH, sendDataStarButton());
  sendCANMessage(secondaryCAN, CAN_ID_SHIFT_UP, digitalRead(SHIFT_UP_PIN));
  sendCANMessage(secondaryCAN, CAN_ID_SHIFT_DOWN, digitalRead(SHIFT_DOWN_PIN));
  sendCANMessage(secondaryCAN, CAN_ID_LEVE_TLAC, digitalRead(LEFT_WHEEL_BUTTON_PIN));
  sendCANMessage(secondaryCAN, CAN_ID_PRAVE_TLAC, digitalRead(RIGHT_WHEEL_BUTTON_PIN));
  sendCANMessage(secondaryCAN, CAN_ID_KILL_SWITCH, digitalRead(KILL_SWITCH_BUTTON_PIN));
  for (int i = 0; i < 4; i++) {
    int switchState = digitalRead(switchInputs[i]);
    sendCANMessage(secondaryCAN, CAN_ID_SWITCH_INPUT1 + i, switchState);
  }
}

//Pomocná funkce pro posílání zpráv na CAN
void sendCANMessage(MCP_CAN &canBus, unsigned int id, int value) {
  byte buf[1] = { (byte)value };
  canBus.sendMsgBuf(id, 0, 1, buf);
}
```

```
//Funkce pro čtení dat z prioritního canu
void processCANMessage(long unsigned int id, unsigned char* data, unsigned char len) {
  switch (id) {
    case CAN_ID_GEAR:
      canData.gear = data[0]; // Předpoklad: 1 bajt pro převodový stupeň
      break;
    case CAN_ID_RPM:
      canData.rpm = (data[0] << 8) | data[1]; // Předpoklad: 2 bajty pro otáčky
      break;
    case CAN_ID_WATER_TEMP:
      canData.waterTemp = data[0]; // Předpoklad: 1 bajt pro teplotu
      break;
    case CAN_ID_OIL_TEMP:
      canData.oilTemp = data[0]; // Předpoklad: 1 bajt pro teplotu
      break;
    case CAN_ID_WATER_PRESSURE:
      canData.waterPressure = ((data[0] << 8) | data[1]) / 100.0; // Předpoklad: 2 bajty, děleno 100
      break;
    case CAN_ID_OIL_PRESSURE:
      canData.oilPressure = ((data[0] << 8) | data[1]) / 100.0;
      break;
    case CAN_ID_THROTTLE_POSITION:
      canData.throttlePosition = ((data[0] << 8) | data[1]) / 100.0;
      break;
    case CAN_ID_FUEL_PRESSURE:
      canData.fuelPressure = ((data[0] << 8) | data[1]) / 100.0;
      break;
    case CAN_ID_INTAKE_TEMP:
      canData.intakeTemp = data[0];
      break;
    case CAN_ID_INTAKE_PRESSURE:
      canData.intakePressure = ((data[0] << 8) | data[1]) / 100.0;
      break;
    case CAN_ID_AFR:
      canData.afr = ((data[0] << 8) | data[1]) / 100.0;
      break;
    case CAN_ID_SPEED:
      canData.speed = ((data[0] << 8) | data[1]) / 100.0;
      break;
```

```
    case CAN_ID_SPEED:
      canData.speed = ((data[0] << 8) | data[1]) / 100.0;
      break;
    case CAN_ID_BATTERY_LOW:
      canData.batteryLow = ((data[0] << 8) | data[1]) / 100.0;
      break;
    case CAN_ID_BATTERY_HYBRID:
      canData.batteryHybrid = ((data[0] << 8) | data[1]) / 100.0;
      break;
    case CAN_ID_TIME:
      canData.time = (data[0] << 24) | (data[1] << 16) | (data[2] << 8) | data[3]; // 4 bajty pro čas
      break;
    case CAN_ID_DATE:
      canData.date = (data[0] << 24) | (data[1] << 16) | (data[2] << 8) | data[3]; // 4 bajty pro datum
      break;
    default:
      // Nepodporované ID
      break;
  }
}

// Funkce pro čtení CAN zpráv
void readCANMessages() {
  long unsigned int rxId;          // ID přijaté zprávy
  unsigned char len = 0;           // Délka datové části
  unsigned char rxBuf[8];          // Buffer pro data

  // Kontrola dostupnosti zprávy na PriorityCAN
  if (priorityCAN.checkReceive() == CAN_MSGAVAIL) {
    if (priorityCAN.readMsgBuf(&rxId, &len, rxBuf) == CAN_OK) {
      processCANMessage(rxId, rxBuf, len); // Zpracování zprávy
    }
  }
}
```

```
/* NENÍ IMPLEMENTOVÁNO
// Přerušovací rutiny PRIO
void handlePriorityCANInterrupt() {
  priorityCANInterrupt = true;
}

// Přerušovací rutiny SEC
void handleSecondaryCANInterrupt() {
  secondaryCANInterrupt = true;
}
*/
```

8 Závěr

Projekt přístrojové desky pro studentskou formuli představuje komplexní řešení pro integraci periferních zařízení, vizualizaci dat a komunikaci přes sběrnici CAN. Software efektivně propojuje jednotlivé moduly systému, jako jsou LED pásek, displej Nextion, potenciometry, přepínače a tlačítka, což umožňuje zpracování dat v reálném čase a jejich přehledné zobrazení. Díky modulární struktuře softwaru je systém snadno rozšiřitelný a přizpůsobitelný. Celkově projekt splňuje všechny stanovené požadavky a poskytuje funkční a přehlednou platformu, která významně přispívá k řídicímu systému formule, zlepšuje uživatelskou interakci a usnadňuje ladění i diagnostiku systému.