Středoškolská technika 2023

Setkání a prezentace prací středoškolských studentů na ČVUT

Inteligentní kurník

David Pavlík

SPŠ a VOŠ Písek, Karla Čapka 402, 397 11 Písek

Čestné prohlášení

„Prohlašuji, že jsem práci na téma Aplikace mikroprocesorové techniky (inteligentní kurník) vypracoval samostatně a výhradně s použitím citovaných pramenů v souladu s Metodickým pokynem pro zadání a realizaci maturitní práce a normou ČSN ISO 690:2011.“

V Písku dne …….. ……

Podpis žáka

Anotace

Tato práce se zabývá inteligentní správou automatického či manuálního ovládání a sledování chlívku pro drůbež. Pro vytvoření věrohodné kopie reálného kurníku bude navrhnut a sestaven prototyp s možností simulace. Hlavním prvkem celé sestavy bude model Arduino MEGA v kombinaci s ESP8266 pro možnost vypsat aktuální hodnoty z periferií na webové rozhraní. Pro snadné ovládaní bez nutnosti připojení k internetu bude sloužit OLED displej s rotačním enkodérem. Práce rovněž obsahuje schéma zapojení jednotlivých částí a vysvětluje jejich použití.

Klíčová slova

Chlívek pro drůbež, inteligentní kurník, mikrokontrolér, Arduino MEGA, ESP8266, webové rozhraní, OLED displej, hlavní nabídka, rotační enkodér, senzor teploty a vlhkosti, senzor hladiny vody, krokový motor, řadič pro krokový motor

Annotation

The graduation thesis deals with the intelligent management of automatic or manual control and monitoring of a poultry house. A prototype with simulation capability will be designed and built to create a plausible replica of a real chicken house. The main element of the whole assembly will be an Arduino MEGA model combined with ESP8266 for the possibility to output the actual values from the peripherals to a web interface. An OLED display with a rotary encoder will be used for easy control without the need for an internet connection. The thesis also includes a wiring diagram of the different parts and explains their use.

Keywords

Poultry hen house, intelligent hen house, microcontroller, Arduino MEGA, ESP8266, web interface, OLED display, main menu, rotary encoder, temperature and humidity sensor, water level sensor, stepper motor, stepper motor controller

Obsah

1. TEORETICKÝ ÚVOD

Cílem této práce je vytvořit inteligentní systém pro ovládání a monitorování prostředí v chlívku pro drůbež, který bude poskytovat možnosti komerčně prodávaných systémů. Hlavním problémem těchto komerčně prodávaných výrobků je poměrně vysoká pořizovací cena vzhledem k možnostem, které nabízejí. Pro běžného uživatele se tak tyto výrobky stávají nedostupnými. Tento problém lze vyřešit užitím open-source prostředků a finančně dostupnějších programovatelných modulů i pro možnost budoucí modifikace. V této práci je pro řídící jednotku použit modul Arduino MEGA v kombinaci s ESP8266, který nabízí mimo jiné i připojení k internetu. Uživatel má tak možnost ovládat celý systém přes webové rozhraní a sledovat aktuální podmínky. K Arduinu jsou pak připojeny další rozšiřující moduly, díky kterým lze snadno sledovat hodnoty a informovat tak uživatele o stavu. Sestava se skládá z OLED displeje, LED modulu, rotačního enkodéru, snímače teploty a vlhkosti vzduchu, snímače hladiny vody, modulu pro ovládání krokového.

1.1. Mechanické provedení

Inteligentní kurník je moderní technologické řešení, které umožňuje automatizované ovládání různých funkcí v kurníku, včetně napájení, osvětlení, otevírání a dalších. Jednou z důležitých součástí inteligentního kurníku je také mechanické řešení, které umožňuje například automatické otevírání a zavírání dveří kurníku. Mechanické řešení ovládání kurníku může být realizováno pomocí různých technologií a principů. Jedním z nejčastěji používaných způsobů je použití elektromotoru a převodovky, které umožňují pohyb dveří kurníku v závislosti na signálu z řídícího systému. Další možností je využití hydraulického systému, který pracuje na principu převodu tlaku kapaliny pro pohyb dveří. Při návrhu a realizaci mechanického ovládání kurníku je třeba zohlednit řadu faktorů, jako jsou rozměry kurníku, hmotnost dveří, rychlost pohybu, spolehlivost, bezpečnost a další. Je také důležité zajistit, aby ovládací mechanismus byl odolný vůči povětrnostním vlivům a nebyl náchylný k poruchám.

1.2. Uživatelské rozhraní

Součástí této problematiky je také uživatelské ovládání, které umožňuje uživateli monitorovat a ovládat jednotlivé funkce kurníku. Uživatelské rozhraní (dále jen UI) je důležitou součástí vývoje jakéhokoli softwarového systému, včetně řešení pro inteligentní kurník. UI se zabývá způsobem, jakým uživatel interaguje s aplikací nebo zařízením, a může mít zásadní vliv na uživatelskou zkušenost a efektivitu používání. Inteligentní kurník představuje moderní řešení pro chov drůbeže, které využívá technologií jako jsou senzory, připojené zařízení, algoritmy strojového učení a další. Pro úspěšnou realizaci inteligentního kurníku je nezbytné navrhnout uživatelské rozhraní, které bude intuitivní, snadno ovladatelné a bude umožňovat přehledné sledování a řízení kurníku. Uživatelské rozhraní by mělo poskytovat uživateli přehled o aktuálním stavu kurníku, umožňovat nastavovat parametry prostředí (například teplotu, vlhkost, osvětlení) a sledovat výkonnost chovu (například počet snesených vajec, růst drůbeže). Dalším důležitým aspektem je integrace UI s ostatními komponentami inteligentního kurníku, jako jsou senzory a další technologie. UI by mělo umožňovat jednoduché a rychlé získání dat z různých zdrojů a jejich následné zpracování a interpretaci. Úspěšná realizace inteligentního kurníku závisí na kvalitním návrhu uživatelského rozhraní, které bude snadno ovladatelné, přehledné a umožní efektivní řízení chovu drůbeže.

1.3. Technické vybavení

Jednou z důležitých složek této technologie je technické vybavení, které zahrnuje různé senzory, akční prvky a mikrokontroléry, které umožňují automatizované řízení různých funkcí kurníku. Jedním z hlavních cílů technického vybavení v inteligentním kurníku je zlepšení kvality života zvířat a snížení jejich stresu. K tomu může sloužit například automatizovaný systém krmných dávek, který umožňuje přesně dávkovat krmivo a zabezpečit tak rovnoměrné příjmy potravy pro zvířata. Dále může být využito senzorů pro monitorování teploty, vlhkosti a kvality ovzduší, aby bylo zajištěno optimální klima v kurníku. Tyto prvky mohou také pomoci v prevenci a detekci nemocí. Dalším důležitým faktorem technického vybavení v inteligentním kurníku je zvýšení efektivity chovu a snížení nákladů. To může být dosaženo například automatizovaným systémem sběru vajec, který umožňuje rychlejší a efektivnější sběr vajec a také minimalizuje riziko poškození vajec nebo ztráty vajec z různých důvodů. Kromě toho mohou být v inteligentním kurníku využívány různé senzory a měřicí zařízení, které umožňují sledovat a optimalizovat podmínky pro

drůbež, například teplotu, vlhkost a osvětlení. Tímto způsobem lze vytvořit prostředí, které je přizpůsobeno potřebám drůbeže a zajišťuje optimální podmínky pro chov a vývoj. To vše přispívá k vysoké efektivitě chovu a snižuje náklady na provoz kurníku, což jsou klíčové faktory pro úspěšné podnikání v oblasti drůbežářství. Správná regulace těchto parametrů je nezbytná pro udržení vhodného prostředí pro chov drůbeže. Mikrokontroléry jsou centrální částí technického vybavení inteligentního kurníku. Tyto mikrokontroléry umožňují řízení a monitorování různých funkcí kurníku, včetně monitorování teploty, vlhkosti a osvětlení a řízení akčních prvků pro regulaci těchto parametrů. Bezdrátové moduly jsou další důležitou součástí technického vybavení inteligentního kurníku. Tyto moduly umožňují bezdrátovou komunikaci mezi mikrokontroléry a jinými zařízeními, jako jsou například mobilní telefony nebo počítače.

1.4. Dálková správa

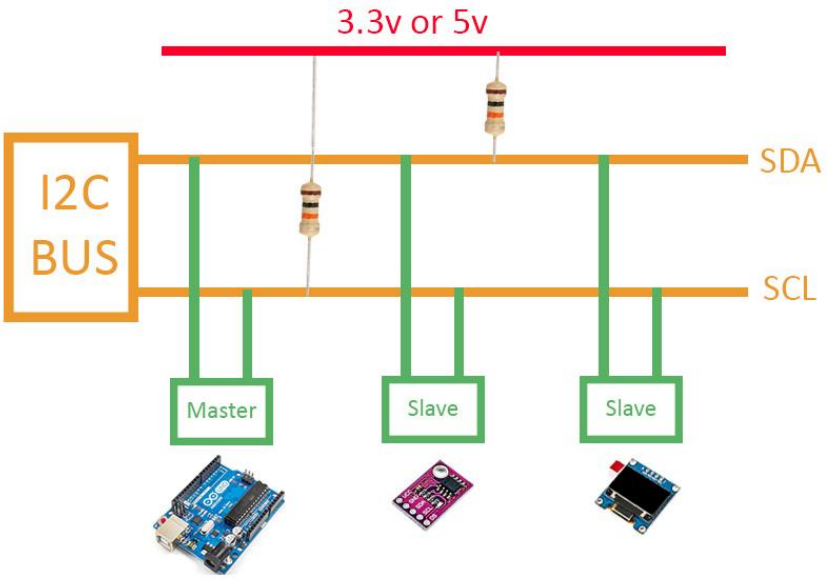
Dalším z klíčových prvků inteligentního kurníku je jeho ovládání, které umožňuje uživatelům monitorovat a řídit různé funkce kurníku z jakéhokoli místa a kdykoli. Kromě uživatelského ovládání na samotném kurníku je však důležité zohlednit také možnost dálkové správy kurníku. Dálková správa inteligentního kurníku se využívá především pro zabezpečení a monitorování chovu drůbeže a umožňuje uživatelům sledovat a řídit kurník z pohodlí svého domova, nebo dokonce z jiného města či země. Například pomocí moderních senzorů a kamer umístěných v kurníku lze sledovat stav zdraví zvířat, jejich stravovací návyky a chování. Díky tomu je možné včas odhalit případné nemoci nebo problémy v chovu, což umožní rychlé a účinné zasahování. Toto řešení je velmi užitečné zejména pro majitele větších farem, kteří se musí na delší dobu vzdálit od svých kurníků. Díky dálkové správě mají takoví majitelé kurníků možnost monitorovat a řídit své kurníky z dálky. Další výhodou dálkové správy je možnost ovládání a automatizace některých činností v kurníku. Například je možné nastavit automatický krmící systém, který bude podle předem stanovených parametrů přesně dávkovat krmivo zvířatům. Stejně tak lze využít automatického klimatizačního systému, který bude udržovat optimální teplotu a vlhkost v kurníku. Dálková správa inteligentního kurníku přináší mnoho výhod pro chovatele drůbeže. Kromě možnosti monitorování a automatizace některých procesů umožňuje také úsporu času a nákladů spojených s chovem zvířat. Díky dálkové správě lze totiž efektivněji řídit a optimalizovat celý chovný proces, což může mít pozitivní vliv na výsledky chovu. Pro realizaci dálkové správy inteligentního kurníku je třeba zvolit vhodnou technologii, která umožní bezdrátové

připojení k internetu a komunikaci s kurníkem. V současné době existuje mnoho různých řešení, které umožňují dálkovou správu kurníku, například pomocí WiFi, Bluetooth nebo mobilních sítí.

1.5. Sériová komunikace

Sériová komunikace umožňuje přenášet data mezi dvěma zařízeními pomocí sériového portu. V případě desky Arduino MEGA + WiFi ATmega2560 + ESP8266 4Mb CH340G se používá UART (Universal Asynchronous Receiver/Transmitter) rozhraní pro sériovou komunikaci mezi mikrokontrolérem a dalšími zařízeními. ESP8266 je integrovaný Wi-Fi modul, který umožňuje připojení desky k internetu. Tím se zvyšuje možnost propojení desky s dalšími zařízeními, které jsou připojeny k internetu, jako jsou například senzory, obrazovky nebo další mikrokontroléry. Při použití sériové komunikace mezi desku Arduino MEGA + WiFi ATmega2560 + ESP8266 4Mb CH340G a jiným zařízením je důležité dodržovat několik základních pravidel. Patří mezi ně například nastavení stejné rychlosti komunikace na obou zařízeních, správné použití příkazů pro odesílání a přijímání dat a také správné nastavení parity, stop bitů a délky datových bitů. Sériová komunikace je jednoduchý, ale velmi užitečný způsob, jak propojit desku Arduino MEGA + WiFi ATmega2560 + ESP8266 4Mb CH340G s dalšími zařízeními a senzory. V kombinaci s Wi-Fi modulem se otevírají nové možnosti pro řízení a sledování různých zařízení z dálky, což je velmi užitečné například pro automatizaci a IoT aplikace.

1.6. Komunikace I2C



obr. 1 Příklad zapojení I2C sběrnice

I2C, zkratka pro Inter-Integrated Circuit, je komunikační protokol, který lze také označit zkratkou IIC. Tento komunikační protokol je široce používán v mikrokontrolérech. Při této komunikaci spolu komunikují nadřízené a podřízené jednotky (nadřízené jednotky jsou obvykle hlavní komponenty, jako jsou mikrokontroléry, a podřízené jednotky jsou senzory, moduly a další komponenty použité v obvodu) prostřednictvím dvou linek:

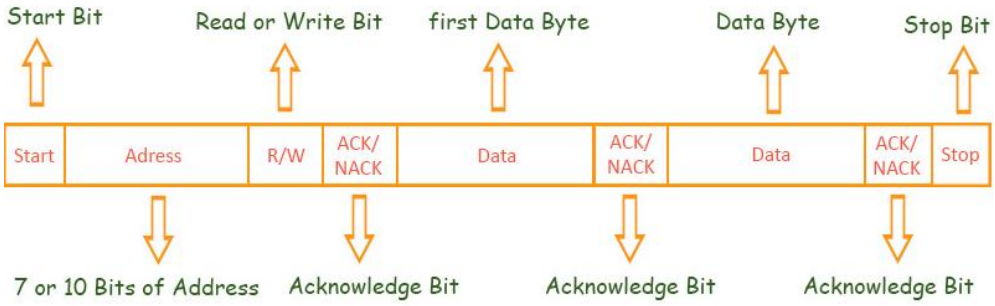
- **SDA (Serial Data):** Linka pro přenos a příjem dat mezi Master a Slave
- **SCL (Serial Clock):** Linka pro odesílání hodin

Nejdůležitější vlastnosti komunikačního protokolu I2C jsou následující:

- Není tak rychlý (ale pro většinu aplikací je dostatečně rychlý).
- Vhodný pro krátké vzdálenosti (např. maximální vzdálenost při 100KHz je 1 metr).
- Synchronní komunikace
- Data jsou přenášena sériově
- Logická úroveň může být 3,3 V i 5 V

V komunikačním protokolu I2C existují dva typy zařízení, která se připojují ke sběrnici I2C: Master a slave. V tomto protokolu může být k jednomu masteru, jako je mikrokontrolér, připojeno více slave zařízení. A také jeden slave může být řízen a monitorován více mastery, ale pouze jeden z masterů může být v daném okamžiku aktivní. Každý master si může

vybrat, se kterým slave bude komunikovat. Tento výběr se provádí prostřednictvím 7bitových adres slave (v některých případech 10bitových). Ke sběrnici I2C tak může být připojeno přibližně 128 podřízených zařízení. Jinými slovy má tedy každý slave svou vlastní specifickou adresu I2C a pro mastery neexistuje žádná adresa. Data přenášená na sběrnici I2C, synchronizovaná hodinovou linkou, SCL. Za synchronizaci je zodpovědný master.



obr. 2 Ukázka zpracování dat pomocí I2C

Počet bajtů dat, která chceme přenést, není omezen. Na sběrnici I2C se neustále přenášejí data a informace do a ze sběrnice master a slave. Jak již bylo zmíněno, master zahajuje komunikaci na sběrnici a zároveň je zdrojem hodinového impulsu.

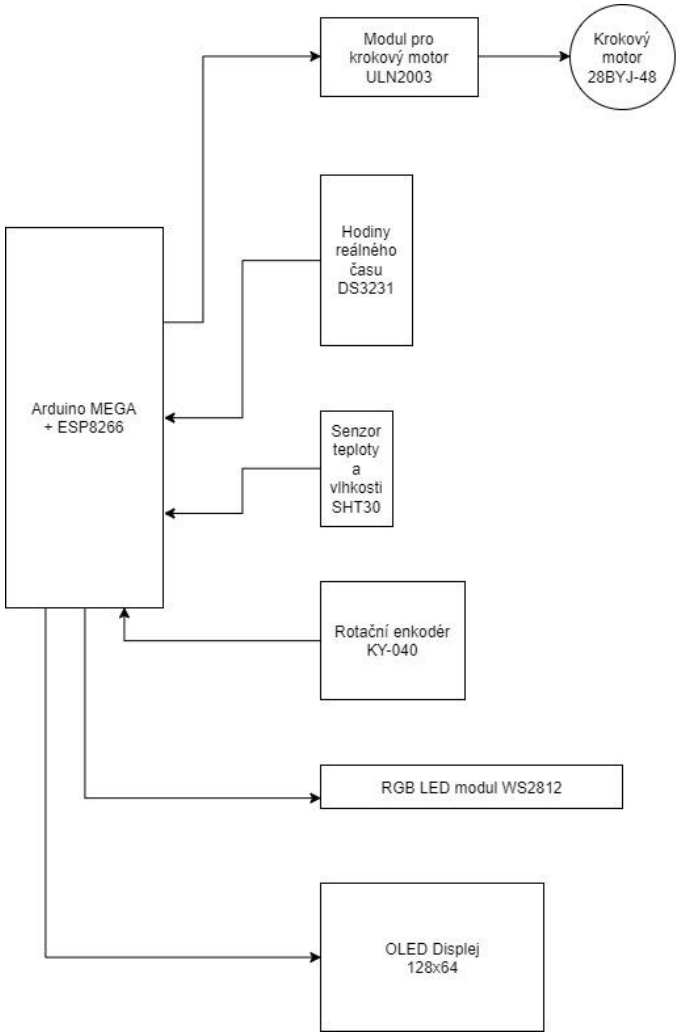
2. AUTORSKÉ ŘEŠENÍ

Návrh systému při řešení realizace inteligentního kurníku zahrnuje návrh hardwaru a softwaru. Tento proces začíná definicí požadavků a cílů, které chceme dosáhnout. Důležitým faktorem je také zvolení vhodného mikrokontroléru, který bude mít dostatečné výpočetní možnosti pro řízení kurníku v našem případě právě deska Arduino MEGA + WiFi ATmega2560 + ESP8266 4Mb CH340G. Poté je nutné navrhnout celkovou architekturu systému, zahrnující například senzory teploty a vlhkosti, hladiny vody, rotační enkodér s tlačítkem, OLED displej a krokový motor. Tyto součástky by měly být propojeny pomocí vhodných komunikačních rozhraní, jako například I2C nebo SPI. Dalším krokem je návrh celého zapojení a vytvoření blokového schéma pro zapojení jednotlivých součástek a propojení s mikrokontrolérem. Při návrhu zapojení je třeba brát v úvahu potřebný výkon, spotřebu energie, umístění součástek a další faktory ovlivňující funkčnost a spolehlivost celého systému. Následně je třeba navrhnout software pro řízení kurníku, který bude pracovat s daty ze senzorů, ovládat akční prvky. Software by měl umožnit uživateli sledovat a komunikovat se systémem, jako je například sledování teploty a vlhkosti prostředí, možnost nastavit časový harmonogram automatického otevírání dveří a další funkce. Po dokončení návrhu a výrobě hardwaru a softwaru je nutné provést testování a ověření funkčnosti celého systému. Případné chyby a nedostatky je třeba odstranit a zajistit spolehlivý chod kurníku.

2.1. Konstrukce

Pro konstrukci bylo nutné navrhnout prototyp, který měl za úkol sloužit jako inspirace pro vytvoření prototypu z dřeva. Použil jsem kartonovou krabici, která skvěle splňovala účel, a mohl jsem si ji tak přizpůsobit, jak jsem potřeboval. Nejprve jsem naměřil stěny a pomocí kancelářské sešívačky jsem jednotlivé díly uchytil k sobě. Následně jsem vytvořil střechu, která měla být sundavací a nabízet tak možnost nahlédnout dovnitř kurníku. Rovněž jsem uchytil podstavu kurníku pomocí malých hřebíků. Připravil jsem si i šoupající se dveře nahoru a dolu pro vytvoření představy, jak realizovat výsledné otevírající se a zavírající se dveře. Jakmile jsem vytvořil dokonalou kopii z kartonu, stačilo pouze návrh realizovat ze dřeva. Jako materiál jsem použil OSB desku pro snadnou konstrukci. Bonusem byla možnost otevírání malých dvířek díky pantům.

2.2. Blokové schéma zapojení



obr. 3 Blokové schéma zapojení

3. SOFTWARE

Návrh systému při řešení realizace inteligentního kurníku zahrnuje návrh hardwaru a softwaru. Tento proces začíná definicí požadavků a cílů, které chceme dosáhnout. Důležitým faktorem je také zvolení vhodného mikrokontroléru, který bude mít dostatečné výpočetní možnosti pro řízení kurníku v našem případě právě deska Arduino MEGA + WiFi ATmega2560 + ESP8266 4Mb CH340G. Poté je nutné navrhnout celkovou architekturu systému, zahrnující například senzory teploty a vlhkosti, hladiny vody, rotační enkodér s tlačítkem, OLED displej a krokový motor. Tyto součástky by měly být propojeny pomocí vhodných komunikačních rozhraní, jako například I2C nebo SPI. Dalším krokem je návrh celého zapojení a vytvoření blokového schéma a nebo navrhnutí desky plošných spojů (PCB) pro zapojení jednotlivých součástek a propojení s mikrokontrolérem. Při návrhu zapojení je

třeba brát v úvahu potřebný výkon, spotřebu energie, umístění součástek a další faktory ovlivňující funkčnost a spolehlivost celého systému. Následně je třeba navrhnout software pro řízení kurníku, který bude pracovat s daty ze senzorů, ovládat akční prvky. Software by měl umožnit uživateli sledovat a komunikovat se systémem, jako je například sledování teploty a vlhkosti prostředí, možnost nastavit časový harmonogram automatického otevírání dveří a další funkce. Po dokončení návrhu a výrobě hardwaru a softwaru je nutné provést testování a ověření funkčnosti celého systému. Případné chyby a nedostatky je třeba odstranit a zajistit spolehlivý chod kurníku.

3.1. Hlavní třída

Hlavní třída je jádrem inteligentního systému, který je navržen pro řízení kurníku. Účelem třídy je poskytnout základní funkcionalitu pro čtení a zpracování dat ze senzorů, řízení krokového motoru pro ovládání dveří a ovládání osvětlení prostoru.
Třída obsahuje různé proměnné, které uchovávají aktuální hodnoty z periferií, jako je senzor hladiny vody, teploty a vlhkosti. Rovněž uchovává informace o stavu ovládaných zařízení, jako jsou dveře a světlo. Obsahuje pak i funkce pro inicializaci sériové komunikace a dalších periferií.

3.1.1. Deklarace proměnných

Na začátku programu je deklarována konstanta, která obsahuje číslo pinu pro senzor hladiny vody. Následují deklarace několika proměnných pro uložení hodnoty senzoru hladiny vody, času posledního čtení senzorů a procentuální hodnoty senzoru hladiny vody. Dále jsou deklarovány proměnné pro uložení aktuální teploty a vlhkosti vzduchu, které jsou měřeny pomocí dalších senzorů. Další deklarované proměnné slouží k uchování informací o stavu dveří a osvětlení, stejně jako k uchování nastavených hodin pro otevírání a zavírání dveří. Kromě toho jsou v programu deklarovány proměnné pro ukládání aktuálního času a data, stejně jako proměnná pro uchování informací o stavu připojení k Wi-Fi. V programu jsou také deklarovány proměnné pro počet kroků krokového motoru, požadavky na

otevírání a zavírání dveří a světla z webového rozhraní a stav nastavování času v hlavní nabídce.

```
const byte WATER_SENSOR = A3;
unsigned long waterValue = 0;
unsigned long waterValuePercent = 0;

unsigned long previousReadMillis = 0;
const unsigned long READ_INTERVAL_MS = 1000;

float temperatureValue = 0;
float humidityValue = 0;

String serialPeripheralData = "";
String serialDataPackage = "";

boolean doorOpened = false;
boolean lightTurned = false;

String openingModes[2] = {"Auto", "Manual"};
int selectedOpeningMode = 0;

String lightingModes[2] = {"Auto", "Manual"};
int selectedLightingMode = 0;

int motorStepCount = 1500;
```

obr. 4 Deklarace proměnných

3.1.2. Inicializace periferií

Funkce *„setup()“* se volá pouze jednou na začátku programu a inicializuje všechny periferie pomocí dalších funkcí. Funkce *„setupSerial()“* je zodpovědná za inicializaci sériové komunikace mezi řídícím systémem a ESP8266. V této funkci dochází ke spuštění sériové komunikace pomocí *„Serial.begin()“*, která inicializuje sériovou komunikaci na portu, kde je připojena seriová linka. V tomto případě se komunikace inicializuje na rychlosti 9600 baud. Dále se spouští funkce *„Serial3.begin()“*, která inicializuje sériovou komunikaci na portu 3, kde je připojeno ESP8266. Rychlost komunikace na tomto portu je nastavena na 115200 baud. Funkce *„initializeLED()“* slouží pro inicializování LED modulu. Dále pak *„initializeMainMenu()“*, která se stará o správné spuštění OLED displeje s rotačním enkodérem. Funkce *„initializeWaterSensor()“* je určena pro senzor hladiny vody, kde dochází k definování pinu ke kterému je senzor připojen.

```
void setup() {
  setupSerial();
  initializeLED();
  initializeMainMenu();
  initializeWaterSensor();
  initializeTemperatureSensor();
  initializeRTC();
  initializeStepperMotor();
}
```

obr. 5 Inicializace periferií

3.1.3. Aktualizace hodnot

Funkce *„loop()“* se volá neustále v cyklu a zahrnuje dvě hlavní funkce. Funkce *„getMainMenu()“* zobrazuje hlavní nabídku na displeji pro uživatele a funkce *„systemControl()“* zajišťuje kontrolu systému a aktualizaci hodnot z periferií.

```
void loop() {
  getMainMenu();
  systemControl();
}
```

obr. 6 Aktualizace systému a hlavní nabídky

3.2. Kontrola systému

Kontrola systému slouží k monitorování a aktualizaci periferií. V kódu jsou definovány funkce pro čtení senzorů, ovládání dveří a světel, a také pro komunikaci s jinými zařízeními pomocí sériového rozhraní.

3.2.1. Aktualizace

Funkce *systemControl()* provádí kontrolu systému čtením hodnot z periferií a následně jeho aktualizaci. Nejdříve se definuje proměnná *currentMillis* jako čas, který uběhl od spuštění programu. Tuto proměnnou používáme k ověření, zda uplynulo dostatečné množství času pro další čtení senzorů a řízení zařízení. Podmínka *if* ověřuje, zda uběhlo dostatečné množství času od posledního čtení senzorů a řízení zařízení. Pokud je podmínka splněna, pokračuje se s dalšími příkazy. Pokud ne, funkce se ukončí. Proměnná *previousReadMillis* se aktualizuje na hodnotu *currentMillis*, aby se uložil čas posledního provedení příkazů uvnitř *if* podmínky. Poté dojde k aktualizaci jednotlivých periferií. Funkce *readWaterSensor()* je určena pro čtení senzoru hladiny vody, *readTemperatureSensor()* slouží pro čtení senzoru teploty a vlhkosti, *getCurrentTime()* pro získání aktuálního času, *serialCommunication()* pro odeslání dat pomocí sériové komunikace, *doorControl()* pro řízení logiky dveří a *lightControl()* pro řízení logiky osvětlení. Jakmile jsou provedeny všechny příkazy uvnitř *if* podmínky, funkce *systemControl()* se ukončí. Tento kód se spouští periodicky, aby bylo možné průběžně kontrolovat stav zařízení, číst senzory a řídit jeho funkce. Četnost spouštění je určena hodnotou konstanty *READ_INTERVAL_MS*, která definuje, jak často má docházet k aktualizaci systému.

```
void systemControl() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousReadMillis >= READ_INTERVAL_MS) {
    previousReadMillis = currentMillis;

    readWaterSensor();
    readTemperatureSensor();
    getCurrentTime();
    serialCommunication();
    doorControl();
    lightControl();
  }
}
```

obr. 7 Čtení hodnot z periferií

3.2.2. Logika pro řízení dveří

Tato funkce je zodpovědná za řízení dveří v systému. Rozhoduje, zda mají být dveře otevřeny nebo zavřeny, v závislosti na zvoleném režimu a současném stavu dveří. Funkce začíná kontrolou, zda je zvolen manuální nebo automatický režim otevírání dveří. Pokud je zvolen režim 0, znamená to, že dveře budou otevírány a zavírány v určitých časech. Pokud je zvolen režim 1, dveře budou otevírány nebo zavírány pouze na základě žádosti uživatele. V případě režimu 0 se funkce nejprve podívá na stav dveří. Pokud jsou dveře uzavřené a zároveň je aktuální čas rovný času otevření dveří, funkce spustí funkci pro otevření dveří. Pokud jsou dveře otevřené a je čas na jejich uzavření, funkce spustí funkci pro uzavření dveří. V případě režimu 1 funkce kontroluje, zda byla zaslána žádost o otevření nebo zavření dveří a zda jsou dveře v požadovaném stavu. Pokud jsou dveře uzavřené a byla zaslána žádost o otevření, funkce spustí funkci pro otevření dveří. Pokud jsou dveře otevřené a byla zaslána žádost o uzavření, funkce spustí funkci pro uzavření dveří.

```
void doorControl() {
  if (selectedOpeningMode == 0) {
    if (!doorOpened && doorState == 2 && currentHour == openDoorHour && currentMinute == openDoorMinute) {
      openDoor();
    }
    else if (doorOpened && doorState == 1 && currentHour == closeDoorHour && currentMinute == closeDoorMinute) {
      closeDoor();
    }
  }
  else if (selectedOpeningMode == 1) {
    if (doorRequest == 1 && doorState == 2 && !doorOpened) {
      openDoor();
    }
    else if (doorRequest == 2 && doorState == 1 && doorOpened) {
      closeDoor();
    }
  }
}
```

obr. 8 Manuální a automatický režim pro ovládání dveří

3.2.3. Logika pro řízení osvětlení

Funkce *lightControl()* zkontroluje, zda je vybrán automatický nebo manuální režim ovládání světel a provede odpovídající akce. Pokud je vybrán režim ovládání světel 0, pak funkce *lightControl()* zkontroluje, zda jsou světla zapnuta a zda je aktuální čas stejný jako

čas, kdy se dveře otevřely. Pokud ano, světla se vypnou pomocí funkce lightOff(). Stejně tak, pokud jsou světla vypnuta a je čas zavřít dveře, funkce lightControl() zkontroluje, zda je aktuální čas stejný jako čas, kdy se dveře zavřely, a pokud ano, zapne světla pomocí funkce lightOn(). Pokud je vybrán režim ovládání světel 1 (tj. manuální ovládání), pak funkce *lightControl()* zkontroluje, zda byl vyslán požadavek na zapnutí nebo vypnutí světel pomocí proměnné *lightRequest*, zda jsou světla ve správném stavu a zda nejsou již zapnuta (pokud se jedná o požadavek na zapnutí světel) nebo vypnuta (pokud se jedná o požadavek na vypnutí světel). Pokud jsou tyto podmínky splněny, funkce *lightControl()* provede odpovídající akci pomocí funkcí *lightOn()* nebo *lightOff()*.

```
void lightControl() {
  if (selectedLightingMode == 0) {
    if (lightTurned && lightState == 1 && currentHour == openDoorHour && currentMinute == openDoorMinute) {
      lightOff();
    }
    else if (!lightTurned && lightState == 2 && currentHour == closeDoorHour && currentMinute == closeDoorMinute) {
      lightOn();
    }
  }
  else if (selectedLightingMode == 1) {
    if (lightRequest == 1 && lightState == 2 && !lightTurned) {
      lightOn();
    }
    else if (lightRequest == 2 && lightState == 1 && lightTurned) {
      lightOff();
    }
  }
}
```

obr. 9 Manuální a automatický režim pro ovládání osvětlení

3.2.4. Ovládání dveří

První příkaz *doorOpened = true* nastaví proměnnou *doorOpened* na hodnotu *true*, což znamená, že dveře jsou nyní otevřené. Druhý příkaz *doorState = 1* nastavuje stav dveří na hodnotu 1, což slouží jako indikátor pro vzdálenou správu, že dveře jsou otevřené. Třetí příkaz *rotateStepperMotor(true, motorStepCount)* spustí funkci *rotateStepperMotor*, která ovládá krokový motor pro otočení dveří do požadované polohy. Tento příkaz specifikuje směr otáčení motorem (v tomto případě *true* znamená, že motor se bude otáčet ve směru hodinových ručiček) a počet kroků (nastavený v proměnné *motorStepCount*), které motor musí vykonat. Dále se vypíše zpráva do sériového monitoru, která informuje o tom, že dveře byly úspěšně otevřeny. Funkce *closeDoor()* pracuje podobně jako *openDoor()*, ale naopak nastaví proměnnou *doorOpened* na hodnotu *false*, což indikuje, že jsou dveře zavřené. Nastaví také proměnnou *doorState* na hodnotu 2, aby bylo možné detekovat stav dveří pro vzdálenou správu, že jsou dveře zavřené. Poté volá funkci *rotateStepperMotor()* s parametrem *false*, což znamená, že se krokový motor bude otáčet opačným směrem, aby uzavřel dveře. Nakonec se opět vypíše zpráva do sériového monitoru, která oznamuje, že dveře byly úspěšně uzavřeny.

```
void openDoor() {
  doorOpened = true;
  doorState = 1;
  rotateStepperMotor(true, motorStepCount);
  Serial.println("# [OZNAMENI] Dvere otevreny");
}
```

obr. 10 Logika pro otevírání dveří

```
void closeDoor() {
  doorOpened = false;
  doorState = 2;
  rotateStepperMotor(false, motorStepCount);
  Serial.println("# [OZNAMENI] Dvere zavreny");
}
```

obr. 11 Logika pro zavírání dveří

3.2.5. Ovládání světla

Tento kód obsahuje funkce *lightOn()* a *lightOff()*, které slouží k zapínání a vypínání světla. Funkce *lightOn()* nejprve nastaví proměnnou *lightTurned* na hodnotu *true*, což indikuje, že je světlo zapnuté. Poté nastaví proměnnou *lightState* na hodnotu 1, aby vzdálené správě bylo jasné, že je světlo zapnuté. Funkce dále volá funkci *setWhiteLED()*, která zapíná bílou LED diodu, která osvětluje místnost. Nakonec se vypíše zpráva do sériového monitoru, která oznamuje, že světlo bylo úspěšně zapnuto. Funkce *lightOff()* pracuje podobně jako *lightOn()*, ale naopak nastaví proměnnou *lightTurned* na hodnotu *false*, což indikuje, že je světlo vypnuté. Nastavuje také proměnnou *lightState* na hodnotu 2, aby vzdálené správě bylo jasné, že je světlo vypnuté. Poté volá funkci *setWhiteLED()* s parametrem *false*, což znamená, že se bílá LED dioda vypne a místnost se zatemní. Nakonec se opět vypíše zpráva do sériového monitoru, která oznamuje, že světlo bylo úspěšně vypnuto.

```
void lightOn() {
  lightTurned = true;
  lightState = 1;
  setWhiteLED(true);
  Serial.println("# [OZNAMENI] Svetlo zapnuto");
}
```

obr. 12 Logika pro zapnutí světla

```
void lightOff() {
  lightTurned = false;
  lightState = 2;
  setWhiteLED(false);
  Serial.println("# [OZNAMENI] Svetlo vypnuto");
}
```

obr. 13 Logika pro vypnutí světla

3.3. Sériová komunikace s ESP8266

Tato část slouží k nastavení a řízení sériové komunikace mezi Arduinem a ESP modulem. Zahrnuje dvě funkce - *setupSerial()* a *serialCommunication()*.

3.3.1. Inicializace sériové komunikace

Funkce *setupSerial()* nastavuje sériové připojení pomocí funkcí *Serial.begin(9600)* a *Serial3.begin(115200)*. Dále pak metoda *Wire.begin()*, která inicializuje I2C komunikaci mezi zařízeními. Rovněž je zde cyklus *while*, který čeká na připojení sériových portů pomocí funkce *delay()*. Nakonec se vypíše oznámení o spuštění systému na sériový port *Serial*.

```
void setupSerial() {
  Wire.begin();
  Serial.begin(9600);
  Serial3.begin(115200);

  while(!Serial && !Serial3) {
    delay(10);
  }
  Serial.println("# [OZNAMENI] Spousteni skriptu PowerHatch.ino");
}
```

obr. 14 Inicializace sériových portů

3.3.2. Komunikace mezi zařízeními

Funkce se stará o sériovou komunikaci mezi Arduinem a ESP modulem. Nejprve vytváří řetězcovou proměnnou *serialPeripheralData*, do které ukládá data, která chce Arduino poslat ESP modulu. Data jsou oddělena znakem „&“ pro snadné zapsání a následné rozklíčování. Tato data zahrnují hodnoty teploty, vlhkosti a procentuálního stavu hladiny vody. Poté funkce odesílá vytvořený řetězec pomocí funkce *Serial3.println(serialPeripheralData)*, což odesílá data na ESP modul. Funkce zároveň kontroluje, zda je nějaká data k dispozici pro příjem z ESP modulu. Pokud ano, funkce postupně čte přijatá data pomocí cyklu *while* a ukládá je do řetězcové proměnné *serialDataPackage*. Nakonec funkce rozděluje *serialDataPackage* pomocí funkce *splitString* na jednotlivé části a ukládá je do proměnných *doorRequest*, *lightRequest* a *wifiState*. Tyto proměnné slouží k ovládání periferií a k získání stavu o připojení WiFi. Data jsou oddělena

znakem „€“ a jsou zpracována pomocí funkce *splitString()*, která rozdělí řetězec na části podle oddělovače a vrátí požadovanou hodnotu jako celočíselné číslo.

```
void serialCommunication() {
  serialPeripheralData = "";
  serialPeripheralData += "&";
  serialPeripheralData += temperatureValue;
  serialPeripheralData += "&";
  serialPeripheralData += humidityValue;
  serialPeripheralData += "&";
  serialPeripheralData += waterValuePercent;
  Serial3.println(serialPeripheralData);

  if (Serial3.available()) {
    serialDataPackage = "";

    while (Serial3.available()) {
      serialDataPackage += char(Serial3.read());
    }
    Serial.print(serialDataPackage);

    doorRequest = splitString(serialDataPackage, '€', 1).toInt();
    lightRequest = splitString(serialDataPackage, '€', 2).toInt();
    wifiState = splitString(serialDataPackage, '€', 3).toInt();
  }
}
```

obr. 15 Přijímání a odesílání dat pomocí sériové komunikace

3.4. Hlavní nabídka

Hlavním účelem tohoto systému je ovládání několika funkcí, jako je například otevírání dveří, zapínání světel, nastavení časovače pro otevírání dveří a jiné. Je tedy nutné implementovat hlavní nabídku pro OLED displej s rotačním enkodérem, která umožňuje uživateli vybrat si požadovanou funkci i bez nutnosti připojení k internetu.

3.4.1. Implementace knihoven a deklarace proměnných

Je zapotřebí zahrnout několik knihoven pro správnou funkci OLED displeje, grafického rozhraní a rotačního enkodéru. Konkrétně se jedná o knihovny pro OLED displej *Adafruit_SH110X.h* a *Adafruit_GFX.h*, knihovnu pro rotační enkodér *ClickEncoder.h* a knihovnu pro interní časovač *TimerOne.h*. Dále se v kódu nachází několik konstant, jako například adresa I2C sběrnice pro komunikaci s OLED displejem, šířka a výška displeje v pixelech počet počet položek v hlavní nabídce. V kódu se také nachází proměnné, které uchovávají aktuálně vybranou položku v hlavní nabídce, poslední vybranou položku, aktuální okno a aktuální stránku. Rovněž jsou zde proměnné pro detekci točení enkodéru nahoru, dolu a stisknutí enkodéru. Poslední část kódu definuje proměnné pro uchování informací o poslední a aktuální hodnotě enkodéru. Nakonec se definují jednotlivé položky,

které se následně zobrazí v hlavní nabídce a definuje se displej s rotačním enkodérem.

```
#include "Adafruit_SH110X.h"
#include "Adafruit_GFX.h"
#include "ClickEncoder.h"
#include "TimerOne.h"

const byte I2C_ADDRESS = 0x3c;
const byte SCREEN_WIDTH = 128;
const byte SCREEN_HEIGHT = 64;
const byte OLED_RESET = -1;

const byte CLK = A0;
const byte DT = A1;
const byte SW = A2;

const int ITEMS_COUNT = 8;
const int FRAME_COUNT = 6;
int selectedMenuItem = 1;
int lastMenuItem = 1;
int currentFrame = 1;
int currentPage = 1;

boolean EncoderUp = false;
boolean EncoderDown = false;
boolean EncoderMiddle = false;

int16_t last, value;
```

obr. 16 Deklarace proměnných pro hlavní nabídku

3.4.2. Inicializace

Kód začíná tím, že vytváří nový objekt třídy *ClickEncoder*, který reprezentuje rotační enkodér a bude použit v inteligentním systému. Vypne se akcelerace enkodéru a vloží se prodleva pro stabilizaci enkodéru. Následně se inicializuje interní časovač s periodou 1000 ms a přiřadí se funkce *timerIsr* jako přerušení tohoto časovače. Poté se inicializuje displej s

konkrétní adresou I2C sběrnice, na které je připojen, a vymaže se zobrazovací paměť. Po 2 sekundové prodlevě se displej zapne. Nakonec se uloží počáteční hodnota enkodéru.

```
void initializeMainMenu() {
  encoder = new ClickEncoder(DT, CLK, SW);
  encoder->setAccelerationEnabled(false);
  delay(250);

  Timer1.initialize(1000);
  Timer1.attachInterrupt(timerIsr);

  display.begin(I2C_ADDRESS, true);
  display.clearDisplay();
  display.display();
  delay(2000);

  last = encoder->getValue();
}
```

obr. 17 Inicializace hlavní nabídky

3.4.3. Logika ovládání

Funkce *getMainMenu()* zahrnuje několik operací, jako například omezení hodnot proměnných otevření a zavření dveří v rozmezí 0-23 pro hodiny a 0-59 pro minuty, vykreslení hlavní nabídky pro uživatele, čtení hodnoty enkodéru a získání stavu tlačítka na enkodéru. Následuje kontrola stavu enkodéru a případné aktualizace proměnných v závislosti na aktuálním výběru uživatele. Pokud se enkodér otočí ve směru hodinových ručiček, vybere se další položka nabídky a v případě, že se uživatel nachází na stránce nastavení počtu kroků motoru, hodin automatického otevírání a zavírání, zvýší se počet. To samé platí i v opačném směru otáčení, jen v opačné logice. Pokud uživatel zmáčkne tlačítko enkodéru a bude se nacházet v nastavení času pro automatické otevírání a zavírání dveří, tak se zvýší stav nastavení času. Rovněž stisknutím tlačítka uživatel potvrzuje a ukládá nastavení pro jednotlivé položky se kterými systém následně pracuje.

```
void getMainMenu() {
  openDoorHour = constrain(openDoorHour, 0, 23);
  openDoorMinute = constrain(openDoorMinute, 0, 59);
  closeDoorHour = constrain(closeDoorHour, 0, 23);
  closeDoorMinute = constrain(closeDoorMinute, 0, 59);
  drawMenu();
  readRotaryEncoder();

  ClickEncoder::Button button = encoder->getButton();
  if (button == ClickEncoder::Clicked) {
    EncoderMiddle = true;

    if (currentPage == 2 && selectedMenuItem == 5) {
      settingTimeState += 1;
    }
  }
```

obr. 18 Získání hlavní nabídky pro uživatele

3.4.4. Vykreslení

Tento kód představuje funkci *drawMenu()*, která slouží k vykreslení uživatelského rozhraní na displej. Funkce obsahuje podmínky *if*, které zajišťují, že se na displeji zobrazí správné položky a další informace. První podmínka kontroluje, zda je aktuální stránka rovna 1. Pokud ano, funkce vykreslí aktuální čas a teplotu, poté vykreslí položky menu, které jsou výběrem zobrazovány podle toho, na kterém snímku se uživatel nachází. Každý snímek obsahuje tři položky a uživatel může pohybovat nahoru a dolů, aby mohl vybrat požadovanou položku. Další podmínky *if* kontrolují, zda se uživatel nachází na druhé stránce hlavní nabídky a zda byla vybrána specifická položka. V tomto případě se na displeji zobrazí další informace, jako např. nastavení módu osvětlení nebo nastavení času pro otevírání a zavírání dveří. Kromě toho se v této funkci používají další funkce pro vykreslení různých prvků na displeji, jako jsou řetězce, časové hodnoty nebo výběr položek z hlavní nabídky.

```
void drawMenu() {
  if (currentPage == 1) {
    display.setTextSize(1);
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK);
    display.setCursor(0, 0);
    display.print(currentHour);
    display.print(":");
    display.print(currentMinute);
    display.print(" | ");
    display.print(temperatureValue, 1);
    display.print((char)247);
    display.print("C ");
    display.print(humidityValue, 1);
    display.print("%");
    display.drawLine(0, 10, 128, 10, SH110X_WHITE);
```

obr. 19 Vykreslení hlavní nabídky

3.4.5. Zobrazení stránek

Kód popisuje funkce, které slouží k zobrazení informací na displeji. Funkce zahrnují zobrazení stránky s časem otevření a zavření, informací o systému, nabídky pro nastavení hodnot typu *String* a *Integer* a také označení aktuálně zvolené položky uživatelem. Funkce *displayTimePage* slouží k zobrazení stránky s časem otevření a zavření. Funkce očekává čtyři parametry: *menuItem* (název položky v hlavní nabídce), *openHour* (hodina otevření), *closeHour* (hodina zavření), *openMinute* (minuta otevření) a *closeMinute* (minuta zavření). Funkce *displaySystemInfoPage* slouží k zobrazení informací o systému, jako je datum, stav Wi-Fi připojení, verzi a autora. Funkce očekává jeden parametr *menuItem*, což jak bylo vysvětleno je název položky v hlavní nabídce. Funkce *displayIntPage* slouží k zobrazení nabídky pro nastavení hodnoty typu *Integer*. Funkce očekává dva parametry: *menuItem* a

value (hodnota proměnné typu *Integer*). Funkce *displayStringPage* slouží k zobrazení nabídky pro nastavení hodnoty typu *String*. Funkce *displaySelectedItem* slouží k označení aktuálně zvolené položky uživatelem. Funkce očekává tři parametry: *item* (název položky), *position* (pozice vybrané položky) a *itemSelected* (pokud je *true*, položka je označena jako vybraná).

```
void displayIntPage(String menuItem, int value) {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SH110X_WHITE, SH110X_BLACK);
  display.setCursor(0, 0);
  display.print(menuItem);
  display.drawLine(0, 15, 128, 15, SH110X_WHITE);
  display.setCursor(5, 25);
  display.print("Pocet kroku motoru:");
  display.setTextSize(2);
  display.setCursor(5, 35);
  display.print(value);
  display.display();
}
```

obr. 20 Příklad pro zobrazení stránky s nastavením číselné hodnoty

3.4.6. Čtení rotačního enkodéru

Tento kód obsahuje funkci *readRotaryEncoder*, která čte hodnotu z enkodéru (rotačního snímače) a ukládá ji do proměnné *value*. Dále porovnává aktuální hodnotu enkodéru s poslední hodnotou a nastavuje proměnnou EncoderUp nebo EncoderDown, a to s časovým zpožděním 150 ms. V této funkci je využita celočíselná aritmetika, kdy je hodnota z enkodéru dělena dvěma a následně porovnávána s poslední hodnotou.

```
void readRotaryEncoder() {
  value += encoder -> getValue();

  if (value / 2 > last) {
    last = value / 2;
    EncoderDown = true;
    delay(150);
  }
  else if (value / 2 < last) {
    last = value / 2;
    EncoderUp = true;
    delay(150);
  }
}
```

obr. 21 Čtení rotačního enkodéru

3.4.7. Interní časovač rotačního enkodéru

Tento kód definuje funkci *timerIsr()*, která je zavolána v přerušení časovače (timeru) a volá funkci *service()* na objektu, který je uložen v ukazateli *encoder*. Tato funkce slouží k

obsluze enkodéru a je využita k přečtení aktuálního stavu enkodéru a k vyvolání událostí, jako jsou změny směru rotace nebo kliknutí na tlačítko.

```
void timerIsr() {
  encoder -> service();
}
```

obr. 22 Přiřazení interního časovače pro enkodér

3.5. Ovládání LED modulu

Třída představuje řízení LED modulu pomocí knihovny *Adafruit_NeoPixel.h*. Třída obsahuje metody pro inicializaci LED modulu a pro nastavení barvy všech LED diod.

3.5.1. Implementace knihoven a deklarace konstant

Konstanta *LED_PIN* určuje, na kterém pinu je připojen LED modul, konstanta *NUM_LEDS* určuje počet LED diod na modulu. Pomocí konstruktoru třídy *Adafruit_NeoPixel* se vytvoří instance *rgb_leds*, kde se předávají parametry *NUM_LEDS*, *LED_PIN* a parametr pro nastavení typu LED modulu *NEO_GRB* + *NEO_KHZ800*.

```
#include <Adafruit_NeoPixel.h>

#define LED_PIN 7
#define NUM_LEDS 8

Adafruit_NeoPixel rgb_leds = Adafruit_NeoPixel(NUM_LEDS, LED_PIN, NEO_GRB + NEO_KHZ800);
```

obr. 23 Implementace knihoven pro LED modul

3.5.2. Inicializace

Funkce *initializeLED()* slouží k inicializaci LED modulu. V této funkci se pomocí metody *begin()* inicializuje instance třídy *Adafruit_NeoPixel* a pomocí funkce *delay()* se čeká 10 milisekund, aby se modul stihl inicializovat.

```
void initializeLED() {
  rgb_leds.begin();
  delay(10);
}
```

obr. 24 Inicializace LED modulu

3.5.3. Nastavení barvy

Funkce *setColorLED()* slouží k nastavení barvy pro všechny LED diody připojené k RGB LED modulu. Funkce bere tři argumenty: *red*, *green* a *blue*, což jsou hodnoty intenzity červené, zelené a modré složky barvy v rozsahu 0-255. Na základě těchto hodnot se vytvoří color pomocí funkce *rgb_leds.Color()*, která vytváří 32bitový *integer* s hodnotami barev v

pořadí *green*, *red*, *blue* (tedy G-R-B). Poté funkce nastaví pomocí *rgb_leds.setPixelColor()* vytvořenou barvu pro každou diodu. Nakonec se pomocí *rgb_leds.show()* zobrazí všechny změny najednou, což zabraňuje problikávání LED při postupném nastavování barvy jednotlivých diod.

```
void setColorLED(byte red, byte green, byte blue) {
  uint32_t color;
  color = rgb_leds.Color(red, green, blue);

  for (int led_number = 0; led_number < NUM_LEDS; led_number++) {
    rgb_leds.setPixelColor(led_number, color);
  }
  rgb_leds.show();
}
```

obr. 25 Nastavení barvy pro LED modul

3.6. Hodiny reálného času

Kód implementuje třídu pro správu hodin reálného času (RTC). Kód obsahuje inicializační funkci pro inicializaci RTC a získání aktuálního času pomocí RTC. Třída využívá knihovnu *RTClib.h* pro komunikaci s RTC_DS3231.

3.6.1. Inicializace

Funkce *initializeRTC()* inicializuje RTC a pokud se nepodaří najít RTC, vypíše se varování. Pokud byly hodiny RTC vypnuty a ztratily svůj napájecí zdroj, funkce nastaví čas na aktuální datum a čas.

```
void initializeRTC() {
  if (!rtc.begin()) {
    Serial.println("! [VAROVANI] Nepodarilo se najit hodiny realneho casu");
  }

  if (rtc.lostPower()) {
    Serial.println("! [VAROVANI] Automaticky se nastavuji hodiny realneho casu");
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }
}
```

obr. 26 Inicializace hodin reálného času

3.6.2. Získání aktuálního času a datumu

Tento kód definuje funkci *getCurrentTime()*, která získá aktuální čas z RTC a uloží několik globálních proměnných s aktuální hodnotou času. Konkrétně uloží aktuální hodinu, minutu, rok, měsíc a den. Následně tato funkce vypíše aktuální čas na sériový port pomocí funkce *Serial.print()*.

```
void getCurrentTime() {
  DateTime now = rtc.now();
  currentHour = now.hour();
  currentMinute = now.minute();

  currentYear = now.year();
  currentMonth = now.month();
  currentDay = now.day();

  Serial.print("# [OZNAMENI] Cas: ");
  Serial.print(currentHour);
  Serial.print(':');
  Serial.print(currentMinute);
  Serial.println();
}
```

obr. 27 Získání aktuálního času a datumu

3.7. Senzor teploty a vlhkosti

Třída *TemperatureSensor* představuje senzor teploty a vlhkosti. Třída využívá knihovny pro komunikaci s I2C a knihovny pro senzor teploty a vlhkosti SHT31 od firmy Adafruit.

3.7.1. Inicializace

Funkce inicializuje senzor teploty a vlhkosti při spuštění programu. Pokud inicializace selže, funkce vypíše varování na sériový port. Používá metodu *begin()* z třídy Adafruit_SHT31 k inicializaci senzoru.

```
void initializeTemperatureSensor() {
  if (!TemperatureSensor.begin()) {
    Serial.println("! [VAROVANI] Nepodarilo se najit senzor teploty a vlhkosti");
  }
}
```

obr. 28 Inicializace senzoru teploty a vlhkosti

3.7.2. Čtení aktuální teploty a vlhkosti

Kód popisuje funkci *readTemperatureSensor()*, která slouží k čtení hodnot z teplotního a vlhkostního senzoru a jejich vypsání na sériovou linku. Konkrétně se zavolají metody *readTemperature()* a *readHumidity()* na instanci třídy *TemperatureSensor* a načtené hodnoty se uloží do globálních proměnných *temperatureValue* a *humidityValue*. Následně se provede kontrola, zda byly hodnoty úspěšně načteny, a pokud ne, vypíše se chybové hlášení. Pokud jsou hodnoty správně načteny, tak se vypíše textový řetězec, který obsahuje aktuální teplotu a vlhkost. Tento řetězec se vypisuje na sériový port a obsahuje jednotky pro teplotu a vlhkost.

```
void readTemperatureSensor() {
  temperatureValue = TemperatureSensor.readTemperature();
  humidityValue = TemperatureSensor.readHumidity();

  if (isnan(temperatureValue) || isnan(humidityValue)) {
    Serial.println("! [VAROVANI] Nelze ziskat hodnoty ze senzoru teploty a vlhkosti");
    return;
  }

  Serial.print("# [OZNAMENI] Teplota: ");
  Serial.print(temperatureValue);
  Serial.write("\xC2\xB0");
  Serial.print("C");
  Serial.print("\t");
  Serial.print("Vlhkost: ");
  Serial.print(humidityValue);
  Serial.println("%");
}
```

obr. 29 Čtení hodnot ze senzoru teploty a vlhkosti

3.8. Senzor hladiny vody

Třída *WaterSensor* slouží pro senzor hladiny vody v rámci inteligentního systému.

3.8.1. Inicializace

V inicializaci *initializeWaterSensor()* se nachází příkaz, kde se nastavuje pin ke kterému je připojen senzor hladiny vody pomocí funkce *pinMode()*.

```
void initializeWaterSensor() {
  pinMode(WATER_SENSOR, INPUT);
}
```

obr. 30 Inicializace senzoru hladiny vody

3.8.2. Čtení hodnoty hladiny vody

Nejprve se pomocí funkce *analogRead()* načte aktuální hodnota senzoru a uloží se do proměnné *waterValue*. Poté se pomocí funkce *map()* přepočítá procentuální hodnota hladiny vody a uloží se do proměnné *waterValuePercent*. Nakonec se hodnota hladiny vody a její procentuální hodnota vypíše na sériovou linku pomocí funkcí *Serial.print()* a *Serial.println()*.

```
void readWaterSensor() {
  waterValue = analogRead(WATER_SENSOR);
  waterValuePercent = map(waterValue, 0, 1023, 0, 100);

  Serial.print("# [OZNAMENI] Hodnota hladiny vody: ");
  Serial.print(waterValue);
  Serial.print("\t");
  Serial.print("Procentuální hodnota: ");
  Serial.println(waterValuePercent);
}
```

obr. 31 Čtení hodnot ze senzoru hladiny vody

3.9. Krokový motor

Tato třída popisuje ovládání krokového motoru. V této třídě jsou definovány čtyři piny, které jsou připojeny k jednotlivým vodičům krokového motoru.

3.9.1. Inicializace

Piny, které jsou připojeny k jednotlivým fázím krokového motoru jsou definovány jako výstupy. To znamená, že tyto piny budou použity k ovládání krokového motoru, aby se mohl otáčet v požadovaném směru o požadovaný počet kroků.

```
void initializeStepperMotor() {
  pinMode(STEPPER_PIN1, OUTPUT);
  pinMode(STEPPER_PIN2, OUTPUT);
  pinMode(STEPPER_PIN3, OUTPUT);
  pinMode(STEPPER_PIN4, OUTPUT);
}
```

obr. 32 Inicializace krokového motoru

3.9.2. Vykonání kroku

V této funkci se pomocí přepínače *switch* určuje, jaký pin z krokového motoru bude aktivní a v jaké kombinaci se mají nastavit ostatní piny, aby se provedl jeden krok. Proměnná *stepCount* uchovává počet kroků krokového motoru. Pokud se funkce volá s parametrem *direction* nastaveným na *true*, znamená to, že se krok provádí vpřed, takže se k *stepCount* přičte 1. V opačném případě se krok provádí dozadu a k *stepCount* se odečte 1. Na konci funkce se proměnná *stepCount* upravuje pomocí operace modulo 4, aby se zajistilo, že se vždy použije správná kombinace pinů a motor se nezasekne.

```
void oneStep(bool direction) {
  switch (stepCount % 4) {
    case 0:
      digitalWrite(STEPPER_PIN1, HIGH);
      digitalWrite(STEPPER_PIN2, LOW);
      digitalWrite(STEPPER_PIN3, LOW);
      digitalWrite(STEPPER_PIN4, LOW);
      break;
    case 1:
      digitalWrite(STEPPER_PIN1, LOW);
      digitalWrite(STEPPER_PIN2, HIGH);
      digitalWrite(STEPPER_PIN3, LOW);
      digitalWrite(STEPPER_PIN4, LOW);
      break;
    case 2:
      digitalWrite(STEPPER_PIN1, LOW);
      digitalWrite(STEPPER_PIN2, LOW);
      digitalWrite(STEPPER_PIN3, HIGH);
      digitalWrite(STEPPER_PIN4, LOW);
      break;
```

obr. 33 Vykonání kroku u krokového motoru

3.9.3. Otáčení krokového motoru

Metoda *rotateStepperMotor()* slouží k ovládání krokového motoru a přijímá dva parametry: směr otáčení *direction* a počet kroků *steps*, které má krokový motor vykonat. V této metodě se pomocí cyklu *for* volá metoda *oneStep()*, která otočí motor o jeden krok.

```
void rotateStepperMotor(bool direction, int steps) {
  for (int i = 0; i < steps; i++) {
    oneStep(direction);
    delay(2);
  }
}
```

obr. 34 Otáčení krokového motoru

3.10. Webové rozhraní

Kód slouží k vytvoření webového rozhraní pro inteligentní systém, který zahrnuje WiFi modul ESP8266 pro komunikaci s internetem. Tento kód využívá knihovny *ESP8266WiFi.h* pro komunikaci pomocí ESP8266 modulu a knihovny *ESPAsyncWebServer* pro asynchronní webové rozhraní. Dále obsahuje definici indexu webového prostředí, který je následně použit v inicializaci serveru. Samotná inicializace serveru se pak provádí na portu 80, který je standardně používán pro HTTP komunikaci. Webové rozhraní obsahuje informace o teplotě, vlhkosti a hladině vody, a také tlačítka pro ovládání dveří a světla.

3.10.1. Zahrnutí knihoven a deklarování proměnných

Je nezbytné deklarovat a inicializovat proměnné, konstanty a další aspekty pro správné fungování webového rozhraní. Hlavním účelem kódu je číst data ze senzorů (teploty, vlhkosti a hladiny vody) a poskytnout je prostřednictvím webového rozhraní. To je realizováno pomocí knihovny *ESPAsyncWebServer.h*, která umožňuje asynchronní komunikaci a zpracování HTTP požadavků. Proměnné *temperatureValue*, *humidityValue* a *waterValuePercent* slouží pro uchování hodnot ze senzorů. Konstanty *ssid* a *password* obsahují informace o názvu a hesle pro Wi-Fi síť. Proměnné *READ_INTERVAL_MS* a *previousReadMillis* slouží k nastavení intervalu, v kterém jsou čteny hodnoty ze senzorů. Proměnné *serialPeripheralData* a *serialDataPackage* jsou využity pro komunikaci s Arduinem přes sériovou linku. Proměnné *doorState*, *lightState* a *wifiState* uchovávají aktuální stav dveří, světla a Wi-Fi připojení.

```
#include "ESP8266WiFi.h"
#include "ESPAsyncWebServer.h"

AsyncWebServer server(80);

float temperatureValue = 0;
float humidityValue = 0;
float waterValuePercent = 0;

const String ssid = "nazev_site";
const String password = "heslo";

const unsigned long READ_INTERVAL_MS = 1000;
unsigned long previousReadMillis = 0;

String serialPeripheralData = "";
String serialDataPackage = "";

int doorState = 0;
int lightState = 0;
int wifiState = 0;
```

obr. 35 Deklarace proměnných pro ESP8266

3.10.2. Definice webového rozhraní

Tento kód obsahuje HTML, který vytváří webové rozhraní pro ovládání inteligentního systému. Soubor obsahuje HTML kód, CSS styly a JavaScript skript pro zobrazení teploty, vlhkosti vzduchu a hladiny vody. Na webovém rozhraní se zobrazí příslušné ikony s

```
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML>
<html>
<head>
  <meta charset="UTF-8" meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" integrity="sha384-fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr"
  crossorigin="anonymous">
  <style>
    html {
      color: #ffffff;
      font-family: Arial;
      background-color: #151515;
      display: inline-block;
      margin: 0px auto;
      text-align: center;
    }
    h2 {
      font-size: 3.0rem;
    }
    p {
      font-size: 3.0rem;
      text-align: left;
    }
    .units {
      font-size: 1.2rem;
    }
    .values {
      font-size: 1.5rem;
      vertical-align:middle;
      padding-bottom: 15px;
    }
```

obr. 36 Vytvoření indexu pro webové rozhraní

příslušnými popisky a hodnotami. Dále jsou na stránce dvě tlačítka pro ovládání zařízení: „Otevřít dveře“ a „Zapnout světlo“, které se po kliknutí změní. Skript obsahuje tři funkce *setInterval*, které se spouštějí každých 10 sekund a zajišťují aktualizaci hodnot teploty, vlhkosti a hladiny vody, které jsou získávány pomocí protokolu *HTTP GET*. Dále obsahuje dvě funkce pro obsluhu tlačítek „Otevřít dveře“ a „Zapnout světlo“. Po kliknutí na tlačítko se pomocí protokolu *HTTP GET* odesílá zpráva na server, která spustí příslušnou akci na zařízení. Kód využívá konstantu *PROGMEM* pro uložení řetězců v paměti programu, což pomáhá šetřit paměť mikrokontroléru.

3.10.3. Inicializace webového rozhraní

Tento kód představuje funkci *setup()*, kde se nastavují inicializační parametry. Kód v prvních dvou řádcích inicializuje sériovou komunikaci s rychlostí 115200 baudů. Dále se provádí připojení k Wi-Fi pomocí názvu sítě *ssid* a hesla *password* uložených v proměnných. Pokud se nepodaří připojení k Wi-Fi, cyklus *while* čeká jednu sekundu a poté vypisuje hlášku o selhání připojení a nastavuje proměnnou *wifiState* na hodnotu 0. Pokud se podaří připojení, vypisuje se hláška o úspěšném připojení a adresa IP. Proměnná *wifiState* je nastavena na hodnotu 1.

Dále se v kódu inicializuje server pomocí *AsyncWebServer*. Server obsluhuje HTTP požadavky a určuje, jaké akce se mají provést při jednotlivých požadavcích. Například při požadavku na kořenovou cestu (tj. "/") se odešle soubor *index_html* a použije se funkce *processor()*. Další řádky definují jednotlivé URL adresy pro získání hodnot teploty, vlhkosti a hladiny vody. Požadavky na tyto adresy jsou zpracovány funkcemi *readTemperature()*, *readHumidity()* a *readWater()* a jsou odeslány zpět klientovi jako textový řetězec.

Poslední část kódu definuje URL adresy pro ovládání dveří a světla. Pokud jsou dveře otevřeny nebo zavřeny, je nastavena proměnná *doorState* na hodnotu 1 nebo 2. Stejně tak, pokud je světlo zapnuto nebo vypnuto, je nastavena proměnná *lightState* na hodnotu 1 nebo 2.

3.10.4. Získání hodnot a zpracování dat

Tato část se zabývá sběrem dat z různých senzorů, konkrétně teplotního senzoru, senzoru vlhkosti vzduchu a senzoru hladiny vody. Každá funkce čte hodnotu ze svého příslušného senzoru a vrací ji jako řetězec *String*. V každé funkci je použit podmíněný výraz *if*, který kontroluje, zda byla hodnota ze senzoru úspěšně načtena. Pokud ne, program vypíše varování na sériový port a vrátí řetězec „--“. Pokud se podaří hodnotu načíst, program ji vypíše na sériový port a vrátí ji jako řetězec.

```
String readTemperature() {
  if (isnan(temperatureValue)) {
    Serial.println("! [VAROVANI] Nepovedlo se ziskat hodnotu teploty");
    return "--";
  }
  else {
    Serial.println(temperatureValue);
    return String(temperatureValue);
  }
}
```

obr. 37 Čtení hodnot teploty ze sériové komunikace

```
String readHumidity() {
  if (isnan(humidityValue)) {
    Serial.println("! [VAROVANI] Nepovedlo se ziskat hodnotu vlhkosti vzduchu");
    return "--";
  }
  else {
    Serial.println(humidityValue);
    return String(humidityValue);
  }
}
```

obr. 38 Čtení hodnot vlhkosti ze sériové komunikace

```
String readWater() {
  if (isnan(waterValuePercent)) {
    Serial.println("! [VAROVANI] Nepovedlo se ziskat hodnotu teploty");
    return "--";
  }
  else {
    Serial.println(waterValuePercent);
    return String(waterValuePercent);
  }
}
```

obr. 39 Čtení hodnot hladiny vody ze sériové komunikace

Funkce *processor* pak zpracovává řetězce typu *String* a vrací odpovídající hodnoty ze senzorů. Funkce bere vstupní parametr *var*, který specifikuje, jakou hodnotu získat, a volá odpovídající funkce *readTemperature*, *readHumidity* nebo *readWater*, které vracejí teplotu, vlhkost a stav hladiny vody. Pokud je parametr *var* roven některému z těchto tří řetězců, funkce *processor* volá příslušnou funkci pro získání hodnoty a vrací ji. Pokud parametr *var* neodpovídá žádné z těchto tří hodnot, funkce *processor* vrátí prázdný řetězec.

```
String processor(const String& var) {
  if(var == "TEMPERATURE"){
    return readTemperature();
  }
  else if(var == "HUMIDITY"){
    return readHumidity();
  }
  else if(var == "WATER"){
    return readWater();
  }
  return String();
}
```

obr. 40 Zpracování dat ze sériové komunikace

```
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("# [OZNAMENI] Nepodarilo se pripojit k Wi-Fi");
    wifiState = 0;
  }
  Serial.print("# [OZNAMENI] Pripojeno k Wi-Fi ");
  Serial.println(WiFi.localIP());
  wifiState = 1;

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/html", index_html, processor);
  });
  server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", readTemperature().c_str());
  });
  server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", readHumidity().c_str());
  });
  server.on("/water", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", readWater().c_str());
  });
  server.on("/buttonOpenDoor", HTTP_GET, [](AsyncWebServerRequest *request) {
    Serial.println("# [OZNAMENI] Dvere otevreny");
    doorState = 1;
  });
  server.on("/buttonCloseDoor", HTTP_GET, [](AsyncWebServerRequest *request) {
    Serial.println("# [OZNAMENI] Dvere zavreny");
    doorState = 2;
  });
  server.on("/buttonLightOn", HTTP_GET, [](AsyncWebServerRequest *request) {
    Serial.println("# [OZNAMENI] Svetlo zapnuto");
    lightState = 1;
  });
  server.on("/buttonLightOff", HTTP_GET, [](AsyncWebServerRequest *request) {
    Serial.println("# [OZNAMENI] Svetlo vypnuto");
    lightState = 2;
  });
  server.begin();
}
```

obr. 41 Spuštění serveru

3.10.5. Sériová komunikace

Funkce *serialCommunication()* se stará o komunikaci se sériovým portem (Serial). Hlavním účelem této funkce je posílat data připojenému zařízení, které si tento kód čte přes Serial port. Konkrétně se nejprve vytvoří prázdná řetězcová proměnná *serialPeripheralData*, do které se postupně přidávají různé údaje o aktuálním stavu periferních zařízení, jako je stav dveří *doorState*, stav světla *lightState* a stav WiFi připojení *wifiState*. Tyto údaje jsou odděleny pomocí speciálního znaku „€“, který se používá jako oddělovač.Poté je pomocí příkazu *Serial.println()* tato řetězcová proměnná odeslána přes Serial port do Arduina. Funkce *serialCommunication()* se zaměřuje na přijímání dat z připojeného zařízení. Pokud jsou k dispozici nějaká data k přijetí, funkce nejprve vytvoří prázdnou řetězcovou proměnnou *serialDataPackage*, do které se postupně ukládají přijatá data, dokud není celý paket přijat. Poté jsou data zpracována pomocí funkce *splitString()*, která rozdělí řetězec na několik částí podle zadaného oddělovače „&“ a uloží je do proměnných *temperatureValue*, *humidityValue* a *waterValuePercent*, které jsou následně k dispozici pro další zpracování. V hlavní smyčce programu *loop()* se používá funkce *millis()*, která vrací počet milisekund od spuštění programu, a s tímto počtem milisekund se porovnává čas posledního čtení dat ze sériového portu. Hlavním účelem tohoto kódu je zajistit, aby se funkce *serialCommunication()* volala pouze v určitých intervalech, které jsou definovány konstantou *READ_INTERVAL_MS*. Pokud uplyne dostatečně dlouhá doba od

posledního volání této funkce, tak se tato funkce volá znovu. Toto opakování funkce je řízeno pomocí proměnné *previousReadMillis*, která uchovává čas posledního úspěšného volání funkce *serialCommunication()*.

```
void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousReadMillis >= READ_INTERVAL_MS) {
    previousReadMillis = currentMillis;

    serialCommunication();
  }
}
```

obr. 42 Zařízení sériové komunikace s Arduinem

3.11. Rozdělovač řetězců

Tento kód představuje funkci *splitString*, která slouží k rozdělení řetězců typu *String* na podřetězce pomocí určitého oddělovače. Funkce bere tři parametry tj. řetězec určený pro rozdělení, znak oddělovače a index podřetězce, který má být navrácen. V první řadě se funkce inicializuje pomocí proměnných *found*, *strIndex* a *maxIndex*. Poté se v cyklu prochází řetězec od prvního znaku do posledního a testuje se, zda se jedná o oddělovač nebo zda se nachází na konci řetězce. Pokud se jedná o oddělovač nebo poslední znak, zvyšuje se počet nalezených oddělovačů a určí se indexy začátku a konce podřetězce. Nakonec funkce vrátí podřetězec určený indexem, pokud byl nalezen, nebo prázdný řetězec.

```
String splitString(String data, char separator, int index) {
  int found = 0;
  int strIndex[] = { 0, -1 };
  int maxIndex = data.length() - 1;

  for (int i = 0; i <= maxIndex && found <= index; i++) {
    if (data.charAt(i) == separator || i == maxIndex) {
      found++;
      strIndex[0] = strIndex[1] + 1;
      strIndex[1] = (i == maxIndex) ? i+1 : i;
    }
  }
  return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}
```

obr. 43 Rozdělovač řetězců

Závěr

Cílem této práce bylo realizovat inteligentní systém pro ovládání a sledování průběhu v chlívku pro drůbež. Zadání se mi podařilo splnit, ale i přes to jsem narazil na mnoho komplikací. V práci jsem nejdříve uvedl co je cílem celého řešení a také jsem rozvedl problematiku, která je v dnešní době velice populárním tématem, a tím je vysoká pořizovací cena těchto hotových projektů. Dále jsem se zaměřil na mechanické provedení celé sestavy a na co je potřeba si dát pozor při vytváření podobné sestavy. V případě tohoto inteligentního systému jsem uvedl i uživatelské rozhraní, které je v tomto případě nezbytné a dovoluje tak uživateli celé zařízení ovládat i bez nutnosti připojení k internetu. Zhodnotil jsem rovněž co vše je možné udělat z hlediska technického vybavení a čemu celý systém může prospívat. Dále jsem se zaměřil na vysvětlení sériové komunikace, protože v tomto případě to bylo jedno z největších témat a bylo nezbytné pro fungování komunikace mezi Arduinem a ESP8266. Hned poté jsem teoreticky vysvětlil komunikaci přes I2C sběrnici mezi jednotlivými zařízeními a jak používat více I2C periferií. Při návrhu softwaru jsem se rozhodl navrhnout a realizovat celý systém hlavní nabídky, který bude sloužit pro snadné ovládání. Jako ovládací prvek jsem zvolil rotační enkodér s tlačítkem, který je snadný na ovládání a není nutné používat např. více tlačítek. Jedním z největších problémů při realizaci softwaru bylo zřízení sériové komunikace mezi Arduinem a ESP8266, kdy docházelo k neustálemu rušení v přijímání a odesílání dat, která se následně zobrazovala ve špatném pořadí na webovém rozhraní. Proto jsem navrhl oddělovač řetězců, který měl zajistit správné snímání přijatých dat a uložit je tak do proměnných pro další zpracování. Dalším tématem byl samotný hardware a zapojení jako takové. Blokové schéma zapojení jsem předvedl na obrázku č. 3. Zde jsem se poměrně zasekl, protože docházelo ke špatnému zobrazování na OLED displeji. Jednotlivé pixely neustále blikaly a někdy se celý systém zasekl. Nakonec jsem zjistil, že příčinou byla špatně zvolená ovládací logika pro I2C. V závěru celého řešení bylo pak samotné testování projektu a ověření správné funkčnosti, kdy jsem se zaměřil na otestování jednotlivých funkcí a také na zátěžový test. Výsledkem celé práce bylo jednak splnění zadání a stanovených podmínek ale pro mě osobně to byla bohatá zkušenost s deskou Arduino MEGA v kombinaci s ESP8266.

Seznam obrázků

Seznam použité literatury

[1] J. Perks, „Ibigroup,“ [Online]. Available: https://www.ibigroup.com/ibi-insights/my-smart-chicken-coop/.

[2] „Arduino,“ [Online]. Available: https://www.arduino.cc/reference/en/language/functions/communication/serial/.

[3] „Wikisofia,“ [Online]. Available: https://wikisofia.cz/wiki/U%C5%BEivatelsk%C3%A9_rozhran%C3%AD.

[4] cornelam, „Instructables,“ [Online]. Available: https://www.instructables.com/I2C-between-Arduinos/.

[5] F. Mazunga, „Sciencedirect,“ [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2468227622003040.

[6] Dejan, „Howtomechatronics,“ [Online]. Available: https://howtomechatronics.com/tutorials/arduino/how-i2c-communication-works-and-how-to-use-it-with-arduino/.

Příloha I.

```
/**
  Inteligentní systém PowerHatch
  Název: PowerHatch
  Účel: Hlavní třída pro inteligentní systém.

  @author David Pavlík
  @version 1.0 01/1/23
*/

const byte WATER_SENSOR = A3;
unsigned long waterValue = 0;
unsigned long waterValuePercent = 0;

unsigned long previousReadMillis = 0;
const unsigned long READ_INTERVAL_MS = 1000;

float temperatureValue = 0;
float humidityValue = 0;

String serialPeripheralData = "";
String serialDataPackage = "";

boolean doorOpened = false;
boolean lightTurned = false;

String openingModes[2] = {"Auto", "Manual"};
int selectedOpeningMode = 0;

String lightingModes[2] = {"Auto", "Manual"};
int selectedLightingMode = 0;

int motorStepCount = 1500;

int doorState = 2;
int doorRequest = 0;

int lightState = 2;
int lightRequest = 0;

int wifiState = 0;

int openDoorHour = 0;
int openDoorMinute = 0;
int closeDoorHour = 0;
int closeDoorMinute = 0;

int currentHour = 0;
int currentMinute = 0;
```

```
int currentYear = 0;
int currentMonth = 0;
int currentDay = 0;

int settingTimeState = 0;

void setup() {
  setupSerial();
  initializeLED();
  initializeMainMenu();
  initializeWaterSensor();
  initializeTemperatureSensor();
  initializeRTC();
  initializeStepperMotor();
}

void loop() {
  getMainMenu();
  systemControl();
}

/**
  Inteligentní systém PowerHatch
  Název: PowerHatch
  Účel: Systém pro kontrolu a aktualizaci periferií.

  @author David Pavlík
  @version 1.0 01/1/23
*/

/**
  Kontroluje systém a aktualizuje hodnoty.
*/
void systemControl() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousReadMillis >= READ_INTERVAL_MS) {
    previousReadMillis = currentMillis;

    readWaterSensor();
    readTemperatureSensor();
    getCurrentTime();
    serialCommunication();
    doorControl();
    lightControl();
  }
}
```

```
/**
  Ovládá otevírání a zavírání dveří.
*/
void doorControl() {
  if (selectedOpeningMode == 0) {
    if (!doorOpened && doorState == 2 && currentHour == openDoorHour &&
currentMinute == openDoorMinute) {
      openDoor();
    }
    else if (doorOpened && doorState == 1 && currentHour == closeDoorHour &&
currentMinute == closeDoorMinute) {
      closeDoor();
    }
  }
  else if (selectedOpeningMode == 1) {
    if (doorRequest == 1 && doorState == 2 && !doorOpened) {
      openDoor();
    }
    else if (doorRequest == 2 && doorState == 1 && doorOpened) {
      closeDoor();
    }
  }
}

/**
  Ovládá zapínání a vypínání světla.
*/
void lightControl() {
  if (selectedLightingMode == 0) {
    if (lightTurned && lightState == 1 && currentHour == openDoorHour &&
currentMinute == openDoorMinute) {
      lightOff();
    }
    else if (!lightTurned && lightState == 2 && currentHour == closeDoorHour &&
currentMinute == closeDoorMinute) {
      lightOn();
    }
  }
  else if (selectedLightingMode == 1) {
    if (lightRequest == 1 && lightState == 2 && !lightTurned) {
      lightOn();
    }
    else if (lightRequest == 2 && lightState == 1 && lightTurned) {
      lightOff();
    }
  }
}

/**
```

```
  Otevírá dveře.
*/
void openDoor() {
  doorOpened = true;
  doorState = 1;
  rotateStepperMotor(true, motorStepCount);
  Serial.println("# [OZNAMENI] Dvere otevreny");
}

/**
  Zavírá dveře.
*/
void closeDoor() {
  doorOpened = false;
  doorState = 2;
  rotateStepperMotor(false, motorStepCount);
  Serial.println("# [OZNAMENI] Dvere zavreny");
}

/**
  Zapíná světlo.
*/
void lightOn() {
  lightTurned = true;
  lightState = 1;
  setWhiteLED(true);
  Serial.println("# [OZNAMENI] Svetlo zapnuto");
}

/**
  Vypíná světlo.
*/
void lightOff() {
  lightTurned = false;
  lightState = 2;
  setWhiteLED(false);
  Serial.println("# [OZNAMENI] Svetlo vypnuto");
}

/**
  Inteligentní systém PowerHatch
  Název: SerialCommunicationManager
  Účel: Nastavení a řízení sériové komunikace.

  @author David Pavlík
  @version 1.0 01/1/23
*/

/**
```

```
  Nastavuje sériové připojení pro komunikaci pomocí sériových portů.
*/
void setupSerial() {
  Wire.begin();
  Serial.begin(9600);
  Serial3.begin(115200);

  while(!Serial && !Serial3) {
    delay(10);
  }
  Serial.println("# [OZNAMENI] Spousteni skriptu PowerHatch.ino");
}

/**
  Odesílá a přijímá data pomocí sériové komunikace.
*/
void serialCommunication() {
  serialPeripheralData = "";
  serialPeripheralData += "&";
  serialPeripheralData += temperatureValue;
  serialPeripheralData += "&";
  serialPeripheralData += humidityValue;
  serialPeripheralData += "&";
  serialPeripheralData += waterValuePercent;
  Serial3.println(serialPeripheralData);

  if (Serial3.available()) {
    serialDataPackage = "";

    while (Serial3.available()) {
      serialDataPackage += char(Serial3.read());
    }
    Serial.print(serialDataPackage);

    doorRequest = splitString(serialDataPackage, '€', 1).toInt();
    lightRequest = splitString(serialDataPackage, '€', 2).toInt();
    wifiState = splitString(serialDataPackage, '€', 3).toInt();
  }
}

/**
  Inteligentní systém PowerHatch
  Název: StringSplitter
  Účel: Rozděluje řetězce ze sériové komunikace.

  @author David Pavlík
  @version 1.0 01/1/23
*/
```

```
/**
  Rozděluje řetězce typu String na podřetězce pomocí oddělovače.

  @param data řetězec určený pro rozdělení
  @param separator znak, kde má být řetězec rozdělen
  @param index určuje který řetězec má být navrácen
  @return rozděluje získaný řetězec dat
*/
String splitString(String data, char separator, int index) {
  int found = 0;
  int strIndex[] = { 0, -1 };
  int maxIndex = data.length() - 1;

  for (int i = 0; i <= maxIndex && found <= index; i++) {
    if (data.charAt(i) == separator || i == maxIndex) {
      found++;
      strIndex[0] = strIndex[1] + 1;
      strIndex[1] = (i == maxIndex) ? i+1 : i;
    }
  }
  return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}

/**
  Inteligentní systém PowerHatch
  Název: MainMenu
  Účel: Třída pro hlavní nabídku na OLED displeji s rotačním enkodérem.

  @author David Pavlík
  @version 1.0 01/1/23
*/

#include "Adafruit_SH110X.h"
#include "Adafruit_GFX.h"
#include "ClickEncoder.h"
#include "TimerOne.h"

const byte I2C_ADDRESS = 0x3c;
const byte SCREEN_WIDTH = 128;
const byte SCREEN_HEIGHT = 64;
const byte OLED_RESET = -1;

const byte CLK = A0;
const byte DT = A1;
const byte SW = A2;

const int ITEMS_COUNT = 8;
const int FRAME_COUNT = 6;
int selectedMenuItem = 1;
```

```
int lastMenuItem = 1;
int currentFrame = 1;
int currentPage = 1;

boolean EncoderUp = false;
boolean EncoderDown = false;
boolean EncoderMiddle = false;

int16_t last, value;

Adafruit_SH1106G display = Adafruit_SH1106G(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);
ClickEncoder *encoder;

String menuItem1 = "Otevrit dvere";
String menuItem2 = "Zapnout svetlo";
String menuItem3 = "Rezim dveri";
String menuItem4 = "Rezim osvetleni";
String menuItem5 = "Nastaveni casovace";
String menuItem6 = "Nastaveni motoru";
String menuItem7 = "Systemove informace";
String menuItem8 = "Tovarni nastaveni";


/**
  Inicializuje hlavní nabídku a s tím spojené prvky.
*/
void initializeMainMenu() {
  encoder = new ClickEncoder(DT, CLK, SW);
  encoder-
>setAccelerationEnabled(false);
  delay(250);

  Timer1.initialize(1000);
  Timer1.attachInterrupt(timerIsr);

  display.begin(I2C_ADDRESS, true);
  display.clearDisplay();
  display.display();
  delay(2000);

  last = encoder->getValue();
}

/**
  Získává hodnoty z hlavní nabídky od uživatele.
*/
void getMainMenu() {
  openDoorHour = constrain(openDoorHour, 0, 23);
```

```
openDoorMinute = constrain(openDoorMinute, 0, 59);
closeDoorHour = constrain(closeDoorHour, 0, 23);
closeDoorMinute = constrain(closeDoorMinute, 0, 59);
drawMenu();
readRotaryEncoder();

ClickEncoder::Button button = encoder->getButton();
if (button == ClickEncoder::Clicked) {
  EncoderMiddle = true;

  if (currentPage == 2 && selectedMenuItem == 5) {
    settingTimeState += 1;
  }
}

if (EncoderUp) {
  if (currentPage == 1) {
    EncoderUp = false;
    lastMenuItem = selectedMenuItem;
    selectedMenuItem = max(selectedMenuItem - 1, 1);

    if (selectedMenuItem >= 2 && selectedMenuItem <= 6) {
      currentFrame--;
    }
  }
  else if (currentPage == 2) {
    EncoderUp = false;
    if (selectedMenuItem == 3) {
      selectedOpeningMode = (selectedOpeningMode - 1 + 2) % 2;
    }
    else if (selectedMenuItem == 4) {
      selectedLightingMode = (selectedLightingMode - 1 + 2) % 2;
    }
    else if (selectedMenuItem == 5) {
      if (settingTimeState == 0) {
        openDoorHour--;
      }
      else if (settingTimeState == 1) {
        openDoorMinute--;
      }
      else if (settingTimeState == 2) {
        closeDoorHour--;
      }
      else if (settingTimeState == 3) {
        closeDoorMinute--;
      }
    }
    else if (selectedMenuItem == 6) {
      motorStepCount -= 50;
```

```
      }
    }
  }

  if (EncoderDown) {
    if (currentPage == 1) {
      EncoderDown = false;
      lastMenuItem = selectedMenuItem;
      selectedMenuItem = min(selectedMenuItem + 1, ITEMS_COUNT);

      if (lastMenuItem == 2 && selectedMenuItem == 3
          || lastMenuItem == 3 && selectedMenuItem == 4
            || lastMenuItem == 4 && selectedMenuItem == 5
              || lastMenuItem == 5 && selectedMenuItem == 6
                || lastMenuItem == 6 && selectedMenuItem == 7 && currentFrame !=
FRAME_COUNT) {
        currentFrame++;
      }
    }
    else if (currentPage == 2) {
      EncoderDown = false;
      if (selectedMenuItem == 3) {
        selectedOpeningMode = (selectedOpeningMode + 1) % 2;
      }
      else if (selectedMenuItem == 4) {
        selectedLightingMode = (selectedLightingMode + 1) % 2;
      }
      else if (selectedMenuItem == 5) {
        if (settingTimeState == 0) {
          openDoorHour++;
        }
        else if (settingTimeState == 1) {
          openDoorMinute++;
        }
        else if (settingTimeState == 2) {
          closeDoorHour++;
        }
        else if (settingTimeState == 3) {
          closeDoorMinute++;
        }
      }
      else if (selectedMenuItem == 6) {
        motorStepCount += 50;
      }
    }
  }

  if (EncoderMiddle) {
    EncoderMiddle = false;
```

```
    if (currentPage == 1) {
      if (selectedMenuItem == 1) {
        doorOpened = !doorOpened;
        menuItem1 = doorOpened ? "Zavrit dvere" : "Otevrit dvere";
        doorOpened ? openDoor() : closeDoor();
      }
      else if (selectedMenuItem == 2) {
        lightTurned = !lightTurned;
        menuItem2 = lightTurned ? "Vypnout svetlo" : "Zapnout svetlo";
        lightTurned ? lightOn() : lightOff();
      }
      else if (selectedMenuItem >= 3 && selectedMenuItem <= 7) {
        currentPage = 2;
      }
      else if (selectedMenuItem == 8) {
        resetDefaults();
      }
    }
    else if (currentPage == 2) {
      if (selectedMenuItem <= 4
          || selectedMenuItem == 5 && settingTimeState == 4
            || selectedMenuItem >= 6 && selectedMenuItem <= 7) {
        currentPage = 1;
        settingTimeState = 0;
      }
    }
  }
}

/**
  Vykresluje hlavní nabídku na OLED displeji.
*/
void drawMenu() {
  if (currentPage == 1) {
    display.setTextSize(1);
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK);
    display.setCursor(0, 0);
    display.print(currentHour);
    display.print(":");
    display.print(currentMinute);
    display.print(" | ");
    display.print(temperatureValue, 1);
    display.print((char)247);
    display.print("C ");
    display.print(humidityValue, 1);
    display.print("%");
    display.drawLine(0, 10, 128, 10, SH110X_WHITE);
```

```
    if (currentFrame == 1) {
      displaySelectedItem(menuItem1, 15, selectedMenuItem == 1);
      displaySelectedItem(menuItem2, 25, selectedMenuItem == 2);
      displaySelectedItem(menuItem3, 35, selectedMenuItem == 3);
    }
    else if (currentFrame == 2) {
      displaySelectedItem(menuItem2, 15, selectedMenuItem == 2);
      displaySelectedItem(menuItem3, 25, selectedMenuItem == 3);
      displaySelectedItem(menuItem4, 35, selectedMenuItem == 4);
    }
    else if (currentFrame == 3) {
      displaySelectedItem(menuItem3, 15, selectedMenuItem == 3);
      displaySelectedItem(menuItem4, 25, selectedMenuItem == 4);
      displaySelectedItem(menuItem5, 35, selectedMenuItem == 5);
    }
    else if (currentFrame == 4) {
      displaySelectedItem(menuItem4, 15, selectedMenuItem == 4);
      displaySelectedItem(menuItem5, 25, selectedMenuItem == 5);
      displaySelectedItem(menuItem6, 35, selectedMenuItem == 6);
    }
    else if (currentFrame == 5) {
      displaySelectedItem(menuItem5, 15, selectedMenuItem == 5);
      displaySelectedItem(menuItem6, 25, selectedMenuItem == 6);
      displaySelectedItem(menuItem7, 35, selectedMenuItem == 7);
    }
    else if (currentFrame == 6) {
      displaySelectedItem(menuItem6, 15, selectedMenuItem == 6);
      displaySelectedItem(menuItem7, 25, selectedMenuItem == 7);
      displaySelectedItem(menuItem8, 35, selectedMenuItem == 8);
    }
    display.drawLine(0, 45, 128, 45, SH110X_WHITE);
    display.setTextColor(SH110X_WHITE, SH110X_BLACK);
    display.setCursor(0, 50);
    String str = "Hladina vody: ";
    str += waterValuePercent;
    str += "%";
    display.print(waterValuePercent <= 10 ? "[!] Nedostatek vody" : str);
    display.display();
  }
  else if (currentPage == 2 && selectedMenuItem == 3) {
    displayStringPage(menuItem3, openingModes[selectedOpeningMode]);
  }
  else if (currentPage == 2 && selectedMenuItem == 4) {
    displayStringPage(menuItem4, lightingModes[selectedLightingMode]);
  }
  else if (currentPage == 2 && selectedMenuItem == 5) {
    displayTimePage(menuItem5, openDoorHour, openDoorMinute, closeDoorHour,
closeDoorMinute);
```

```
  }
  else if (currentPage == 2 && selectedMenuItem == 6) {
    displayIntPage(menuItem6, motorStepCount);
  }
  else if (currentPage == 2 && selectedMenuItem == 7) {
    displaySystemInfoPage(menuItem7);
  }
}

/**
  Obnovuje nastavení hlavní nabídky do továrního nastavení.
*/
void resetDefaults() {
  openDoorHour = 0;
  openDoorMinute = 0;
  closeDoorHour = 0;
  closeDoorMinute = 0;
  selectedLightingMode = 0;
  selectedOpeningMode = 0;
  motorStepCount = 1500;
}

/**
  Zobrazuje nabídku pro nastavení času.

  @param menuItem položka z hlavní nabídky pod kterou se zobrazí tato funkce
  @param openHour nastavení hodin otevírání
  @param openMinute nastavení minut otevírání
  @param closeHour nastavení hodin zavírání
  @param closeMinute nastavení minut zavírání
*/
void displayTimePage(String menuItem, int openHour, int openMinute, int
closeHour, int closeMinute) {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SH110X_WHITE, SH110X_BLACK);
  display.setCursor(0, 0);
  display.print(menuItem);
  display.drawLine(0, 15, 128, 15, SH110X_WHITE);
  display.setCursor(5, 25);

  if (settingTimeState <= 1) {
    display.print("Cas otevirani:");
    display.setTextColor(settingTimeState == 0 ? SH110X_BLACK : SH110X_WHITE,
settingTimeState == 0 ? SH110X_WHITE : SH110X_BLACK);
    display.setTextSize(2);
    display.setCursor(5, 35);
    display.print(openHour);
    display.setTextColor(SH110X_WHITE, SH110X_BLACK);
```

```
    display.print(":");
    display.setTextColor(settingTimeState == 0 ? SH110X_WHITE : SH110X_BLACK,
settingTimeState == 0 ? SH110X_BLACK : SH110X_WHITE);
    display.print(openMinute);
  }
  else {
    display.print("Cas zavirani:");
    display.setTextColor(settingTimeState == 2 ? SH110X_BLACK : SH110X_WHITE,
settingTimeState == 2 ? SH110X_WHITE : SH110X_BLACK);
    display.setTextSize(2);
    display.setCursor(5, 35);
    display.print(closeHour);
    display.setTextColor(SH110X_WHITE, SH110X_BLACK);
    display.print(":");
    display.setTextColor(settingTimeState == 2 ? SH110X_WHITE : SH110X_BLACK,
settingTimeState == 2 ? SH110X_BLACK : SH110X_WHITE);
    display.print(closeMinute);
  }
  display.display();
}

/**
  Zobrazuje informace o systému.

  @param menuItem položka z hlavní nabídky pod kterou se zobrazí tato funkce
*/
void displaySystemInfoPage(String menuItem) {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SH110X_WHITE, SH110X_BLACK);
  display.setCursor(0, 0);
  display.print(menuItem);
  display.drawLine(0, 15, 128, 15, SH110X_WHITE);
  display.setCursor(5, 25);
  display.print("Datum: ");
  display.print(currentDay);
  display.print("/");
  display.print(currentMonth);
  display.print("/");
  display.print(currentYear);
  display.setCursor(5, 35);
  display.print("Wi-Fi: ");
  display.print(wifiState ? "Pripojeno" : "Odpojeno");
  display.setCursor(5, 45);
  display.print("Verze: 1.0-alpha");
  display.setCursor(5, 55);
  display.print("Autor: David Pavlik");
  display.display();
}
```

```
/**
  Zobrazuje nabídku pro nastavení hodnoty typu String.

  @param menuItem položka z hlavní nabídky pod kterou se zobrazí tato funkce
  @param value hodnota nastavené proměnné
*/
void displayIntPage(String menuItem, int value) {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SH110X_WHITE, SH110X_BLACK);
  display.setCursor(0, 0);
  display.print(menuItem);
  display.drawLine(0, 15, 128, 15, SH110X_WHITE);
  display.setCursor(5, 25);
  display.print("Pocet kroku motoru:");
  display.setTextSize(2);
  display.setCursor(5, 35);
  display.print(value);
  display.display();
}

/**
  Zobrazuje nabídku pro nastavení hodnoty typu String.

  @param menuItem položka z hlavní nabídky pod kterou se zobrazí tato funkce
  @param value hodnota nastavené proměnné typu String
*/
void displayStringPage(String menuItem, String value) {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SH110X_WHITE, SH110X_BLACK);
  display.setCursor(0, 0);
  display.print(menuItem);
  display.drawLine(0, 15, 128, 15, SH110X_WHITE);
  display.setCursor(5, 25);
  display.print("Zvoleny rezim:");
  display.setTextSize(2);
  display.setCursor(5, 35);
  display.print(value);
  display.display();
}

/**
  Označuje aktuálně zvolenou položku uživatelem.

  @param item položka která se označí
  @param position pozice vybrané položky
  @param itemSelected pokud je položka vybrána
```

```
*/
void displaySelectedItem(String item, int position, bool itemSelected) {
  display.setCursor(0, position);
  if (itemSelected) {
    display.setTextColor(SH110X_BLACK, SH110X_WHITE);
    display.print("> ");
  }
  else {
    display.setTextColor(SH110X_WHITE, SH110X_BLACK);
    display.print("  ");
  }
  display.print(item);
}

/**
  Čte hodnotu z enkodéru.
*/
void readRotaryEncoder() {
  value += encoder -> getValue();

  if (value / 2 > last) {
    last = value / 2;
    EncoderDown = true;
    delay(150);
  }
  else if (value / 2 < last) {
    last = value / 2;
    EncoderUp = true;
    delay(150);
  }
}

/**
  Zpracovává signály z enkodéru, určuje počet otáček a směr.
*/
void timerIsr() {
  encoder -> service();
}

/**
  Inteligentní systém PowerHatch
  Název: RTC
  Účel: Třída pro hodiny reálného času.

  @author David Pavlík
  @version 1.0 01/1/23
*/

#include <RTClib.h>
```

```
RTC_DS3231 rtc;

/**
  Inicializuje hodiny reálného času.
*/
void initializeRTC() {
  if (!rtc.begin()) {
    Serial.println("! [VAROVANI] Nepodarilo se najit hodiny realneho casu");
  }

  if (rtc.lostPower()) {
    Serial.println("! [VAROVANI] Automaticky se nastavuji hodiny realneho
casu");
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }
}

/**
  Získává aktuální čas z RTC.
*/
void getCurrentTime() {
  DateTime now = rtc.now();
  currentHour = now.hour();
  currentMinute = now.minute();

  currentYear = now.year();
  currentMonth = now.month();
  currentDay = now.day();

  Serial.print("# [OZNAMENI] Cas: ");
  Serial.print(currentHour);
  Serial.print(':');
  Serial.print(currentMinute);
  Serial.println();
}

/**
  Inteligentní systém PowerHatch
  Název: LEDController
  Účel: Třída pro řízení LED modulu.

  @author David Pavlík
  @version 1.0 01/1/23
*/

#include <Adafruit_NeoPixel.h>

#define LED_PIN 7
#define NUM_LEDS 8
```

```
Adafruit_NeoPixel rgb_leds = Adafruit_NeoPixel(NUM_LEDS, LED_PIN, NEO_GRB +
NEO_KHZ800);

/**
  Inicializuje LED modul.
*/
void initializeLED() {
  rgb_leds.begin();
  delay(10);
}

/**
  Funkce pro nastavení barvy všech LED diod.

  @param red hodnota pro nastavení červené barvy diody (0-255)
  @param green hodnota pro nastavení zelené barvy diody (0-255)
  @param blue hodnota pro nastavení modré barvy diody (0-255)
*/
void setColorLED(byte red, byte green, byte blue) {
  uint32_t color;
  color = rgb_leds.Color(red, green, blue);

  for (int led_number = 0; led_number < NUM_LEDS; led_number++) {
    rgb_leds.setPixelColor(led_number, color);
  }
  rgb_leds.show();
}

/**
  Funkce pro zapnutí/vypnutí bílé LED diody.

  @param enable hodnota true/false pro zapnutí/vypnutí bílé diody
*/
void setWhiteLED(bool enable) {
  byte red = enable ? 255 : 0;
  byte green = enable ? 255 : 0;
  byte blue = enable ? 255 : 0;

  setColorLED(red, green, blue);
}

/**
  Inteligentní systém PowerHatch
  Název: StepperMotor
  Účel: Třída pro krokový motor.

  @author David Pavlík
  @version 1.0 01/1/23
```

```
*/

const byte STEPPER_PIN1 = 9;
const byte STEPPER_PIN2 = 10;
const byte STEPPER_PIN3 = 11;
const byte STEPPER_PIN4 = 12;

unsigned int stepCount = 0;

/**
  Inicializuje krokový motor při spuštění programu.
*/
void initializeStepperMotor() {
  pinMode(STEPPER_PIN1, OUTPUT);
  pinMode(STEPPER_PIN2, OUTPUT);
  pinMode(STEPPER_PIN3, OUTPUT);
  pinMode(STEPPER_PIN4, OUTPUT);
}

/**
  Funkce pro otáčení krokového motoru.

  @param direction směr otáčení krokového motoru
  @param steps počet kroků krokového motoru
*/
void rotateStepperMotor(bool direction, int steps) {
  for (int i = 0; i < steps; i++) {
    oneStep(direction);
    delay(2);
  }
}

/**
  Funkce pro provedení kroku pro krokový motor.

  @param direction směr otáčení krokového motoru
*/
void oneStep(bool direction) {
  switch (stepCount % 4) {
    case 0:
      digitalWrite(STEPPER_PIN1, HIGH);
      digitalWrite(STEPPER_PIN2, LOW);
      digitalWrite(STEPPER_PIN3, LOW);
      digitalWrite(STEPPER_PIN4, LOW);
      break;
    case 1:
      digitalWrite(STEPPER_PIN1, LOW);
      digitalWrite(STEPPER_PIN2, HIGH);
      digitalWrite(STEPPER_PIN3, LOW);
```

```
      digitalWrite(STEPPER_PIN4, LOW);
      break;
    case 2:
      digitalWrite(STEPPER_PIN1, LOW);
      digitalWrite(STEPPER_PIN2, LOW);
      digitalWrite(STEPPER_PIN3, HIGH);
      digitalWrite(STEPPER_PIN4, LOW);
      break;
    case 3:
      digitalWrite(STEPPER_PIN1, LOW);
      digitalWrite(STEPPER_PIN2, LOW);
      digitalWrite(STEPPER_PIN3, LOW);
      digitalWrite(STEPPER_PIN4, HIGH);
      break;
  }
  stepCount += direction ? 1 : -1;
  stepCount = (stepCount + 4) % 4;
}
/**
  Inteligentní systém PowerHatch
  Název: TemperatureSensor
  Účel: Třída pro senzor teploty a vlhkosti.

  @author David Pavlík
  @version 1.0 01/1/23
*/

#include "Wire.h"
#include "Adafruit_SHT31.h"

Adafruit_SHT31 TemperatureSensor;

/**
  Inicializuje senzor teploty a vlhkosti při spuštění programu.
*/
void initializeTemperatureSensor() {
  if (!TemperatureSensor.begin()) {
    Serial.println("! [VAROVANI] Nepodarilo se najit senzor teploty a
vlhkosti");
  }
}

/**
  Čte hodnoty senzoru teploty a vlhkosti.
*/
void readTemperatureSensor() {
  temperatureValue = TemperatureSensor.readTemperature();
  humidityValue = TemperatureSensor.readHumidity();
```

```
  if (isnan(temperatureValue) || isnan(humidityValue)) {
    Serial.println("! [VAROVANI] Nelze ziskat hodnoty ze senzoru teploty a
vlhkosti");
    return;
  }

  Serial.print("# [OZNAMENI] Teplota: ");
  Serial.print(temperatureValue);
  Serial.write("\xC2\xB0");
  Serial.print("C");
  Serial.print("\t");
  Serial.print("Vlhkost: ");
  Serial.print(humidityValue);
  Serial.println("%");
}

/**
  Inteligentní systém PowerHatch
  Název: WaterSensor
  Účel: Třída pro senzor hladiny vody.

  @author David Pavlík
  @version 1.0 01/1/23
*/

/**
  Čte hodnotu senzoru hladiny vody.
*/

void initializeWaterSensor() {
  pinMode(WATER_SENSOR, INPUT);
}

void readWaterSensor() {
  waterValue = analogRead(WATER_SENSOR);
  waterValuePercent = map(waterValue, 0, 1023, 0, 100);

  Serial.print("# [OZNAMENI] Hodnota hladiny vody: ");
  Serial.print(waterValue);
  Serial.print("\t");
  Serial.print("Procentuální hodnota: ");
  Serial.println(waterValuePercent);
}

/**
  Inteligentní systém PowerHatch
  Název: ESP8266
  Účel: Třída pro spuštění serveru a vytvoření webového rozhraní.
  @author David Pavlík
```

```
  @version 1.0 01/1/23
*/

#include "ESP8266WiFi.h"
#include "ESPAsyncWebServer.h"

AsyncWebServer server(80);

float temperatureValue = 0;
float humidityValue = 0;
float waterValuePercent = 0;

const String ssid = "nazev_site";
const String password = "heslo";

const unsigned long READ_INTERVAL_MS = 1000;
unsigned long previousReadMillis = 0;

String serialPeripheralData = "";
String serialDataPackage = "";

int doorState = 0;
int lightState = 0;
int wifiState = 0;


/**
  Definuje index webového prostředí.
*/
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML>
<html>
<head>
  <meta charset="UTF-8" meta name="viewport" content="width=device-width,
initial-scale=1">
  <link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.7.2/css/all.css"
integrity="sha384-
fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr"
  crossorigin="anonymous">
  <style>
    html {
      color: #ffffff;
      font-family: Arial;
      background-color: #151515;
      display: inline-block;
      margin: 0px auto;
      text-align: center;
    }
```

```
    h2 {
      font-size: 3.0rem;
    }
    p {
      font-size: 3.0rem;
      text-align: left;
    }
    .units {
      font-size: 1.2rem;
    }
    .values {
      font-size: 1.5rem;
      vertical-align:middle;
      padding-bottom: 15px;
    }
    .buttonDoor {
      background-color: #4CAF50;
      border: none;
      color: white;
      padding: 15px 32px;
      text-align: center;
      text-decoration: none;
      display: inline-block;
      font-size: 24px;
      margin: 4px 2px;
      cursor: pointer;
      border-radius: 15px;
      float: left;
    }
    .buttonLight {
      background-color: #4CAF50;
      border: none;
      color: white;
      padding: 15px 32px;
      text-align: center;
      text-decoration: none;
      display: inline-block;
      font-size: 24px;
      margin: 4px 2px;
      cursor: pointer;
      border-radius: 15px;
      float: left;
    }
  </style>
</head>
<body>
  <h2>Webové rozhraní PowerHatch</h2>
  <p>
    <i class="fas fa-thermometer-half" style="color:#cc2121;"></i>
```

```
    <span class="values">Teplota:</span>
    <span id="temperature">%TEMPERATURE%</span>
    <sup class="units">&deg;C</sup>
  </p>
  <p>
    <i class="fas fa-random" style="color:#00add6;"></i>
    <span class="values">Vlhkost vzduchu:</span>
    <span id="humidity">%HUMIDITY%</span>
    <sup class="units">&percnt;</sup>
  </p>
  <p>
    <i class="fas fa-tint" style="color:#00add6;"></i>
    <span class="values">Hladina vody:</span>
    <span id="water">%WATER%</span>
    <sup class="units">&percnt;</sup>
  </p>
  <button class="buttonDoor">Otevřít dveře</button>
  <button class="buttonLight">Zapnout světlo</button>
</body>
<script>
setInterval(function() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("temperature").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/temperature", true);
  xhttp.send();
}, 10000);
setInterval(function() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("humidity").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/humidity", true);
  xhttp.send();
}, 10000);
setInterval(function() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("water").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/water", true);
  xhttp.send();
```

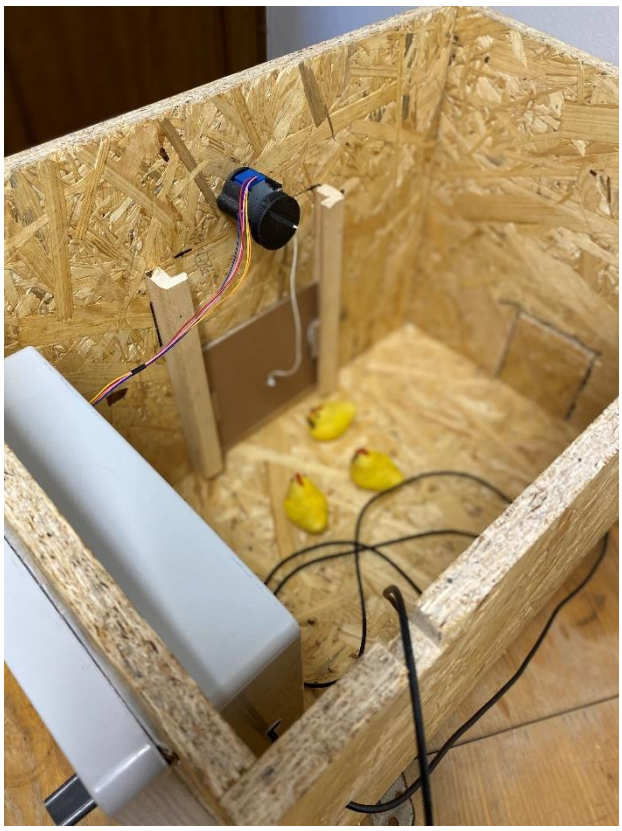
```
}, 10000);
document.querySelector('.buttonDoor').addEventListener('click', function() {
  var button = document.querySelector('.buttonDoor');
  var xhttp = new XMLHttpRequest();
  if (button.innerHTML == "Otevřít dveře") {
    button.style.backgroundColor = "red";
    button.innerHTML = "Zavřít dveře";
    xhttp.open("GET", "/buttonOpenDoor", true);
  }
  else {
    button.style.backgroundColor = "#4CAF50";
    button.innerHTML = "Otevřít dveře";
    xhttp.open("GET", "/buttonCloseDoor", true);
  }
  xhttp.send();
});
document.querySelector('.buttonLight').addEventListener('click', function() {
  var button = document.querySelector('.buttonLight');
  var xhttp = new XMLHttpRequest();

  if (button.innerHTML == "Zapnout světlo") {
    button.style.backgroundColor = "red";
    button.innerHTML = "Vypnout světlo";
    xhttp.open("GET", "/buttonLightOn", true);
  }
  else {
    button.style.backgroundColor = "#4CAF50";
    button.innerHTML = "Zapnout světlo";
    xhttp.open("GET", "/buttonLightOff", true);
  }
  xhttp.send();
});
</script>
</html>)rawliteral";

/**
  Čte hodnotu teploty z teplotního senzoru.
*/
String readTemperature() {
  if (isnan(temperatureValue)) {
    Serial.println("! [VAROVANI] Nepovedlo se ziskat hodnotu teploty");
    return "--";
  }
  else {
    Serial.println(temperatureValue);
    return String(temperatureValue);
  }
}
```

```
/**
  Čte hodnotu vlhkosti z teplotního senzoru.
*/
String readHumidity() {
  if (isnan(humidityValue)) {
    Serial.println("! [VAROVANI] Nepovedlo se ziskat hodnotu vlhkosti vzduchu");
    return "--";
  }
  else {
    Serial.println(humidityValue);
    return String(humidityValue);
  }
}

String readWater() {
  if (isnan(waterValuePercent)) {
    Serial.println("! [VAROVANI] Nepovedlo se ziskat hodnotu teploty");
    return "--";
  }
  else {
    Serial.println(waterValuePercent);
    return String(waterValuePercent);
  }
}

/**
  Provádí zpracování řetězce typu String a vrací hodnoty.

  @param var typ proměnné
  @return vrácení hodnot ze senzorů
*/
String processor(const String& var) {
  if(var == "TEMPERATURE"){
    return readTemperature();
  }
  else if(var == "HUMIDITY"){
    return readHumidity();
  }
  else if(var == "WATER"){
    return readWater();
  }
  return String();
}

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
```

```
    delay(1000);
    Serial.println("# [OZNAMENI] Nepodarilo se pripojit k Wi-Fi");
    wifiState = 0;
  }
  Serial.print("# [OZNAMENI] Pripojeno k Wi-Fi ");
  Serial.println(WiFi.localIP());
  wifiState = 1;

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/html", index_html, processor);
  });
  server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", readTemperature().c_str());
  });
  server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", readHumidity().c_str());
  });
  server.on("/water", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", readWater().c_str());
  });
  server.on("/buttonOpenDoor", HTTP_GET, [](AsyncWebServerRequest *request) {
    Serial.println("# [OZNAMENI] Dvere otevreny");
    doorState = 1;
  });
  server.on("/buttonCloseDoor", HTTP_GET, [](AsyncWebServerRequest *request) {
    Serial.println("# [OZNAMENI] Dvere zavreny");
    doorState = 2;
  });
  server.on("/buttonLightOn", HTTP_GET, [](AsyncWebServerRequest *request) {
    Serial.println("# [OZNAMENI] Svetlo zapnuto");
    lightState = 1;
  });
  server.on("/buttonLightOff", HTTP_GET, [](AsyncWebServerRequest *request) {
    Serial.println("# [OZNAMENI] Svetlo vypnuto");
    lightState = 2;
  });
  server.begin();
}

/**
  Rozděluje řetězce typu String na podřetězce pomocí oddělovače.

  @param data řetězec určený pro rozdělení
  @param separator znak, kde má být řetězec rozdělen
  @param index určuje který řetězec má být navrácen
  @return rozděluje získaný řetězec dat
*/
String splitString(String data, char separator, int index) {
  int found = 0;
```

```
  int strIndex[] = { 0, -1 };
  int maxIndex = data.length() - 1;

  for (int i = 0; i <= maxIndex && found <= index; i++) {
    if (data.charAt(i) == separator || i == maxIndex) {
      found++;
      strIndex[0] = strIndex[1] + 1;
      strIndex[1] = (i == maxIndex) ? i+1 : i;
    }
  }
  return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}

/**
  Odesílá data senzorů pomocí sériové komunikace.
*/
void serialCommunication() {
  serialPeripheralData = "";
  serialPeripheralData += "€";
  serialPeripheralData += doorState;
  serialPeripheralData += "€";
  serialPeripheralData += lightState;
  serialPeripheralData += "€";
  serialPeripheralData += wifiState;

  Serial.println(serialPeripheralData);

  if (Serial.available()) {
    serialDataPackage = "";

    while (Serial.available()) {
      serialDataPackage += char(Serial.read());
    }
    Serial.print(serialDataPackage);
    temperatureValue = splitString(serialDataPackage, '&', 1).toFloat();
    humidityValue = splitString(serialDataPackage, '&', 2).toFloat();
    waterValuePercent = splitString(serialDataPackage, '&', 3).toFloat();
  }
}

void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousReadMillis >= READ_INTERVAL_MS) {
    previousReadMillis = currentMillis;

    serialCommunication();
  }
}
```

Příloha II.




IP55