VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

REKONFIGUROVATELNÝ IOT UZEL NA BÁZI ESP8266/ESP32

RECONFIGURABLE ESP8266/ESP32-BASED NODE FOR IOT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETER DRAHOVSKÝ**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. VLADIMÍR JANOUŠEK, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20280/2017/xdraho11

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Drahovský Peter**
Obor: Informační technologie
Téma: **Rekonfigurovatelný IoT uzel na bázi ESP8266/ESP32**
Reconfigurable ESP8266/ESP32-Based Node for IoT
Kategorie: Vestavěné systémy

Pokyny:

1. Seznamte se s existujícími hardwarovými platformami na bázi ESP8266/ESP32 a s přístupy k tvorbě software pro tyto systémy.
2. Seznamte se s existujícími implementacemi PNOS (Petri Net Operating System) pro Raspberry Pi a pro Arduino Mega v C++. Prostudujte i alternativní možnosti tvorby rekonfigurovatelného sofware pro vestavěné systémy a IoT (MicroPython, JS, Lua, LISP).
3. Navrhněte a realizujte systém s podobnými možnostmi rekonfigurace jako má PNOS, ale bez Petriho sítí, s využitím vhodného interpretovaného jazyka, a to pro platformu ESP8266 nebo ESP32.
4. Proveďte testování za pomoci vhodné aplikace a vyhodnoťte dosažené výsledky.

Literatura:

- RICHTA Tomáš a JANOUŠEK Vladimír. Operating System for Petri Nets-Specified Reconfigurable Embedded Systems. In: Computer Aided Systems Theory - EUROCAST 2013. Berlin Heidelberg: Springer Verlag, 2013, s. 444-451. ISBN 978-3-642-53855-1.
- Domovská stránka projektu RENEW. Dostupná na URL http://www.renew.de

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a část návrhu.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Janoušek Vladimír, doc. Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Cieľom tejto práce je vytvoriť rekonfigurovateľný systém pre mikrokontroléry ESP8266 a ESP32 s použitím interpretovaného jazyka. Systém umožňuje nahrávať a spúšťať aplikácie, ktoré môžu komunikovať s ostatnými aplikáciami spustenými na akomkoľvek uzle v rámci danej siete. Na komunikáciu so systémom a medzi aplikáciami je použitý rovnaký protokol ako pri systéme PNOS (operačný systém na bázi Petriho sietí), pričom samotná komunikácia prebieha cez MQTT. Systém môže byť nasadený v rámci jednej siete v spolupráci s PNOS, ale aj samostatne.

Abstract

The purpose of this thesis is to create reconfigurable system for microcontrolers ESP8266 and ESP32 with usage of interpreted language. System enables to load and run aplications, which are allowed to comunicate with other aplications that are running on any of the nodes on the same network. A protocol used to communicate within aplications and the system is compatible with one used in PNOS (operating sysem based on Petri nets). System can cooperate with PNOS on the same network, or it can be run on its own.

Klíčová slova

ES8266, ESP32, Micropython, MPOS, IoT

Keywords

ES8266, ESP32, Micropython, MPOS, IoT

Citace

DRAHOVSKÝ, Peter. *Rekonfigurovatelný IoT uzel na bázi ESP8266/ESP32*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Vladimír Janoušek, Ph.D.

Rekonfigurovatelný IoT uzel na bázi ESP8266/ESP32

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Vladimíra Janouška Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Peter Drahovský
14. května 2018

Poděkování

Touto cestou by som chcel poďakovať doc. Ing. Vladimíru Janouškovi Ph.D. za odbornú pomoc a vedenie tejto bakalárskej práce.

Obsah

Kapitola 1

Úvod

V posledných rokoch sa vďaka znižujúcej cene mikrokontrolérov a komunikačných modulov začali rozširovať platformy pre internet vecí. Jedným z najvyužívanejších modulov v nepodnikovej sfére je modul ESP8266 a to najmä kvôli svojej cene. Aj napriek jeho relatívne malému výkonu sa na ňom podarilo umožniť beh niekoľkým interpretovaným jazykom, ktoré môžu značne uľahčiť vývoj rekonfigurovateľného operačného systému pre túto platformu.

Jedna z platforiem, ktorú je možné využiť pri nasadzovaní rekonfigurovateľného systému je operačný systém PNOS, ktorý k popisu aplikácií bežiacich v systéme využíva petriho siete. Konfigurácia v petriho sietiach môže byť výhodná, avšak niektoré aplikácie je jednoduchšie popísať v programovacom jazyku.

Cieľom tejto práce je vytvoriť operačný systém pre `ESP8266` alebo `ESP32`, ktorý umožní beh aplikácií popísaných v niektorom z podporovaných interpretovaných jazykov. Protokol tohto operačného systému by mal byť kompatibilný s protokolom použitým na komunikáciu vrámci PNOS, čo umožní ich spoluprácu vrámci jedného systému. V takomto prípade bude možné pre aplikácie, pre ktoré je to vhodnejšie, použiť Micropython a petriho siete využiť pre ostatné aplikácie. Micropython má oproti Petriho sietiam výhodu vo veľkosti komunity. Zatiaľ čo komunita okolo experimentálneho PNOS je veľmi malá, komunita Micropythonu vytvorila veľké množstvo knižníc a modulov. Spojením s PNOS by bolo možné využívať dostupnosti knižníc so zachovaním štruktúry a protokolu systému PNOS. Systém by malo byť možné využívať aj samostatne, bez PNOS.

Kapitola 2

Distribuované riadiace systémy

Distribuovaný riadiaci systém je zložený z niekoľkých podsystémov, ktoré spolu komunikujú využitím komunikačnej zbernice a spoločne sa podieľajú na riadení systému. Tento prístup sa začal rozvíjať a nasadzovať po vzniku komunikačných zberníc dosahujúcich požadované vlastnosti (rýchlosť a spoľahlivosť) a lepšej cenovej dostupnosti jednotlivých zariadení s procesorom – počítačov. Zlacnenie umožnilo nahradenie jedného výkonného centrálneho prvku niekoľkými menej výkonnými, ktoré sú schopné vykonávať rovnakú požadovanú prácu. [21, 13]

2.1 Internet vecí

IoT – z anglického „Internet of things“, je označenie pre sieť fyzických zariadení vybavených elektronikou, senzormi, softwarom a sieťovou konektivitou, ktorá umožňuje týmto zariadeniam prepojiť sa, komunikovať a vymieňať si dáta. Každé z týchto zariadení musí byť jednoznačne identifikovateľné. [19]

Pojem „vecí“ v oblasti IoT môže zahŕňať celú radu zariadení, ako sú napríklad telové implantáty so senzormi, inteligentné chladničky alebo senzory zabudované v automobiloch. Akékoľvek zariadenie, ktoré je schopné monitorovať alebo meniť stav svojho okolia a je schopné komunikovať s ostatnými zariadeniami patrí do tejto skupiny. Takéto namerané údaje je potom schopné samostatne rozosielať iným pripojeným zariadeniam.

Za prvé zariadenie zapojené do internetu vecí sa považuje modifikovaný stroj na výrobu Coca Coly, ktorýbol v roku 1982 pripojený na internet. Na diaľku dokázal informovať o stave surovín a o ich teplote. [19]

IoT sa ale začal rozširovať najmä v posledných rokoch s príchodom lacných mikroprocesorov a dostupnými možnosťami bezdrôtovej komunikácie. Dnes sa začínajú rozširovať najmä cloudové riešenia IoT, pričom sa nejedná len o produkty pre koncového spotrebiteľa. IoT sa využíva aj v komerčne, napríklad pri sledovaní zaplnenosti parkovísk, alebo pri inteligentných lampách sledujúcich a reagujúcich na svoje okolie.[1] Takéto riešenia typicky využívajú cloudové platfromy ako Amazon AWS, Google Cloud IoT alebo IBM Watson Internet of Things a riešenie je typicky prispôsobené presne potrebám zákazníka.

[1] https://prazsky.denik.cz/zpravy_region/potemni-praha-chytre-lampy-se-samy-zhasinaji-20161207.html

Možnosti komunikácie

Pre komunikáciu s ostatnými zariadeniami nemusí byť informácia nutne prenášaná cez internet i keď tomu tak vo väčšine prípadov je. Zariadenie môže ku komunikácii využi aj prenos dát cez iné zariadenie, ktoré je napojené na internet a môže komunikovať s ostatným prvkami. Pre IoT sa typicky používa niektorá z následujúcich technológií: [22]

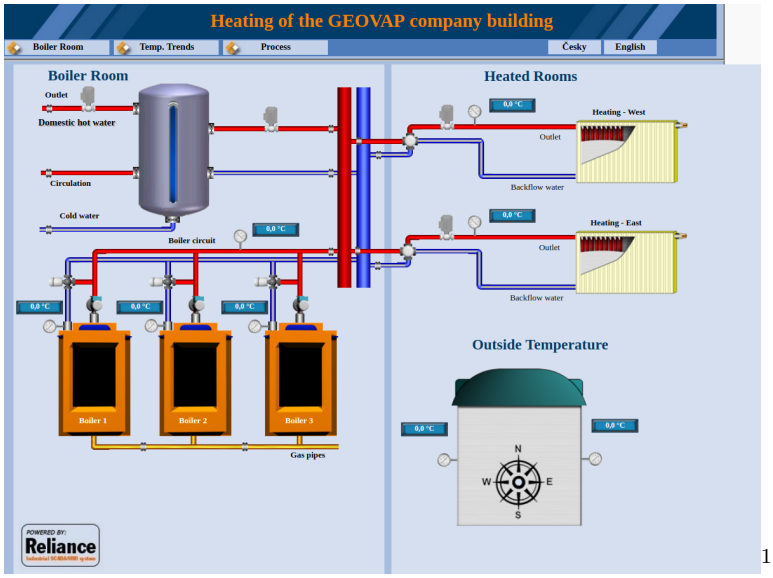
- Bluetooth
- WiFi
- ZigBee
- Z-Wave
- Mobilná sieť GSM
- Lora
- Sigfox
- Ethernet

Spôsoby komunikácie sa líšia v spotrebe, dosahu ale aj v rýchlosti alebo množstve odoslaných dát. Pre senzory, ktoré sú bez trvalého napájania sa hodia technológie s nižšími nárokmi na elektrickú energiu. Takéto pripojenia (Lora alebo Sigfox) sú ale pomalé a nemumožňujú prenos väčšieho množstva dát, ktoré potrebujú napríklad diaľničné kamery. Pokiaľ to spotreba a dosah dovoľuje, dá sa pre internet vecí použiť aj niektorá z už existujúcich technológií využívaná napríklad pre mobilné zariadenia. V takom prípade pri realizácii projektu odpadá nutnosť budovania pokrytia sieťou.

2.2 SCADA systémy

Supervisory Controll And Data Acquisition – v skratke SCADA, je architektúra riadiacich systémov, ktoré využívajú počítače, sieťovú komunikáciu a grafické rozhranie na poskytnutie vysoko úrovňovej kontroly nad systémom. Na koordinovanie riadeného systému využíva SCADA systém typicky iné podsystémy, ktoré komunikujú s low-level časťami systému. [20] Takýto systém umožňuje zobraziť dáta v takej podobe, v akej sú pre používateľa najprirodzenejšie a najjednoduchšie k interpretácii. Typicky sa rozhranie takýchto systémov približuje k nákresu kontrolovaného a riadeného úseku systému (kotolňa, časť továrne, výrobná linka, ...). Kontrolór daného úseku má pred sebou dáta, ktoré môže rýchlo a jednoznačne identifikovať. Medzi príklady SCADA systémov sa dá uviesť český produkt SCADA Reliance[2](na obrázku 2.1) alebo slovenský Ipesoft[3].

[2]https://www.reliance-scada.com/cs/main
[3]https://www.ipesoft.com



[1]

Obrázek 2.1: Ukážka SCADA systému určeného na riadenie a sledovanie stavu kotolne. Systém je od spoločnosti SCADA Reliance a jeho demo je dostupné na ich stránkach.

2.3 Home automation

Využitím prvkov IoT a SCADA systémov v domácnostiach vznikla možnosť centralizácie a automatizácie ovládania domácností. Takéto systémy umožňujú užívateľom ovládať v domácnosti osvetlenie, vykurovanie, chladenie, zabezpečenie ale aj sledovať spotrebu energií alebo dochádzky. Na základe získaných údajov umožňujú systémy vyhodnocovať a optimalizovať náklady na prevádzku domu, automaticky ovládať pripojené zariadenia, vypnúť osvetlenie alebo vykurovanie po odchode, prípadne ovládať celú domácnosť aj na diaľku. Ovládanie z mobilov a počítačov je dostupné pri väčšine riešení.

Systémy pre zautomatizovanie riadenia domácností sú väčšinou poskytované ako hotové a uzavreté riešenie od jedného výrobcu, no existujú aj opensourcové riešenia. Medzi tie patria napríklad OpenHab, Domoticz alebo HomeAssistant. U týchto riešení sú podporované zariadenia závislé na komunite.

[1] Prevzaté z: https://www.reliance-scada.com/cs/products/reliance-demo-applications

Kapitola 3

Vhodné hardwarové platformy a spôsoby tvorby softwaru

V následujúcej kapitole sú zhrnuté hardwarové platformy, ktoré sú vďaka svojím vlastnostiam vhodné pre vývoj rekonfigurovateľného IoT uzla. V závere kapitoly sú uvedené programovacie jazyky, v ktorých je možné vyvíjať software na týchto platformách.

3.1 Hardwarové platformy vhodné pre rekonfigurovateľný IOT uzol

Pri výbere platformy, ktorá bude použitá pri vývoji, boli kladené požiadavky nielen na výkon mikrokontrolérov obsiahnutých v moduloch, ale aj na ich cenu. Keďže pri nasadení systému sa očakáva veľké množstvo použitých modulov (napríklad v rádoch desiatok pri riadení domácnosti), malo by zvýšenie ceny jedného modulu výrazný dopad aj na celkovú cenu pri nasadení systému.

Pri výbere padla voľba na produkty firmy Espressif Systems, ktoré sú výrazne lacnejšie oproti iným riešeniam. Zrejme aj vďaka ich cene sú veľmi rozšírene používané a existuje množstvo komunitne vyvíjaných softwarových riešení určených na vývoj na týchto platformách. Nasleduje popis týchto modulov a modulu Pyboard pre lepšie porovnanie parametrov a cenovej dostupnosti modulov.

ESP8266

Mikročip ESP8266 [4, 8] je vyvinutý spoločnosťou Espressif Systems. Pôvodne bol mienený hlavne ako doplnkový modul k iným mikrokontrolérom, umožňujúci bezdrôtovú komunikáciu v pásme 2.4 Ghz. Medzi jeho hlavné výhody patrí cena a minimum vyžadovaných externých komponentov. Modul sa dostal do povedomia najmä po zahájení predaja modulu ESP-01, navrhnutého spoločnosťou Ai-thinker. Modul vo verzii ESP-12E je možné vidieť na obrázku 3.1. SoC obsahuje 32 bitový RISC procesor Tensilica L106, pracujúci na maximálnej frekvencii 160MHz (typicky 80MHz), doplnený o 96 kB pamäte RAM. Neobsahuje programovateľnú ROM pamäť, preto je pre uloženie programových inštrukcií nutné pripojiť externú SPI flash pamäť o veľkosti minimálne 512kB. Maximálna podporovaná veľkosť externej flash pamäte je 16MB. [9] Veľkosť osadenej pamäte sa na väčšine modulov pohybuje obvykle v rozpätí 512KB až 4MB.

[1] Prevzaté z: https://www.itead.cc/esp-12f.html

[1]

Obrázek 3.1: Modul ESP-12E

Periférie

ESP8266 má 17 vstupno-výstupných pinov, na ktorých je možné využiť následujúce periférie:

I²C Rozhranie je schopné pracovať v oboch režimoch („master“ aj „slave“) pri maximálnej frekvencii je 100 kHz. Mikrokontrolér nemá priamu podporu hardwarového I²C. Kedže je riešené softwarovo, je možné porty `SDA` a `SCL` nastaviť na ktorékoľvek dva voľné piny.

UART ESP8266 disponuje jedným plnohodnotným hardwarovým rozhraním `UART0`. Rozhranie `UART1` umožňuje iba odosielanie dát. Maximálna rýchlosť prenosu je cca. 4,5 Mb/s.

SPI Tento mikročip obsahuje dve hardwarové SPI rozhrania `SPI0` a `SPI1`. Rozhranie `SPI0` je použité na komunikáciu s externou flash pamäťou. Pre užívateľa tak typicky zostáva iba `SPI1`.

I²S Jedno vstupné `(GPIO 12, 13 a 14)` a jedno výstupné rozhranie `(GPIO 15, 3 a 2)`

A/D prevodník ESP8266 disponuje jedným 10 bitovým analogovo digitálnym prevodníkom na pine 6.

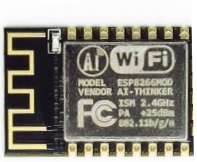
PWM Ktorýkoľvek z pinov je možné využiť na pulzne šírkovú moduláciu, najviac však dokopy 5 pinov.

Spotreba energie

Spotreba ESP8266 je závislá mimo iné hlavne na kvalite signálu pripojenej bezdrôtovej siete. Pri vysielaní dát do siete s dobrým signálom môže spotreba stúpnuť z kľudových 50-60 mA na 120 mA a pri zlej kvalite signálu dokonca aj na 170 mA. Na zníženie spotreby a umožnenie behu aj na batériu, umožňuje ESP8266 prepnutie do rôznych režimov spánku. Pri týchto režimoch sa dokáže spotreba stlačiť na prijateľných 15 mA, 0,5 mA alebo dokonca až na 0.5 μA. Pri režime spánku sú obmedzené funkcie MCU a to podľa zvoleného režimu (od vypnutého wifi modemu až po kompletné zastavenie CPU a periférií okrem časovača).

Verzie

Od vydania modulu `ESP-01` vytvorila Ai-thinker vyše 20 verzií modulov [3] obsahujúcich mikročip `ESP8266`. Líšia sa veľkosťou, počtom vyvedených pinov, anténou, dosahom ale aj cenou. (viz prehľad najrozšírenejších 3.1) [17, 3]

Názov	**ESP-01**	**ESP-12E**	**ESP07**
Obrázok	 1	 2 1	 1
Rozmery (mm)	14.3 x 24.8	24.0 x 16.0	21.2 x 16.0
Počet vývodov	8	22	16
Anténa	na plošnom spoji	na plošnom spoji	keramická + konektor
Cena	36 Kč	41 Kč	45 Kč
Tienenie	nie	áno	áno

Tabulka 3.1: Porovnanie ESP8266 modulov

Dostupné vývojové kity

V súčasnej dobe existuje niekoľko vývojových dosiek, ktoré obsahujú okrem samotného ESP8266 aj iné užitočné komponenty. Vo väčšine prípadov obsahujú 3,3V regulátor napätia, USB -> UART prevodník a majú vyvedené výstupy mikročipu v štandardnom rozostupe univerzálnych plošných spojov (2.54 mm). Príkladom môže byť opensource kit NodeMcu (obr.3.2).

3

Obrázek 3.2: NodeMcu

4

Obrázek 3.3: Sonoff BASIC

[1] Prevzaté z: https://www.esp8266.com/wiki/doku.php?id=esp8266-module-family
[2] Prevzaté z: https://www.itead.cc/esp-12f.html
[3] Prevzaté z: https://www.makerfabs.com/index.php?route=product/product&product_id=194
[4] Prevzaté z: http://sonoff.itead.cc/en/products/sonoff/sonoff

Okrem toho existujú aj hotové produkty pre bežného spotrebiteľa obsahujúce ESP8266. Zväčša majú pred-nahratý firmware umožňujúci napojenie iba na riešenia daného výrobcu, no vo väčšine prípadov je možné do nich nahrať firmware vlastný. Spomenúť sa dajú napríklad produkty Sonoff (obr. 3.3) od výrobcu Itead [12]. V závislosti od typu sú osadené rôznym počtom relé, diód, alebo spínačov. Takéto produkty sa hodia hlavne pre koncové nasadenie. Vývoj prebieha jednoduchšie na vývojových doskách typu NodeMcu, pretože pre ich používanie a programovanie postačuje prepojiť USB rozhrania modulu a počítača. Doska je z neho napájaná a umožňuje aj programovanie. Naproti tomu programovanie produktov typu Sonoff vyžaduje externé napájanie a externý USB → UART prevodník.

ESP32

V roku 2016 spoločnosť Espressif Systems oznámila príchod nového mikročipu `ESP32` (na obrázku 3.4), označovaného za nástupcu `ESP8266`. Narozdiel od ESP8266 bol už od začiatku primárne určený ako samostatne fungujúci mikrokontrolér, čo umožnilo zamerať sa pri vývoji na iné ciele. ESP32 rieši väčšinu problémov, s ktorými užívatelia bojovali pri ESP8266, ako napríklad hardwarovú podporu I2c, veľkosť pamäte RAM alebo problém s obsluhou bezdrôtového rozhrania na jednojadrovom procesore.[7] SoC obsahuje 32 bitový dvojjadrový procesor Xtensa LX6 pracujúci na frekvenci 160 alebo 240 MHz, 520 kB pamäte RAM a 448 kB ROM. Umožňuje bezdrôtovú komunikáciu v pásme 2,4 Ghz (802.11 b/g/n) a Bluetooth vo verzii 4.2.[6]

[1]

Obrázek 3.4: Modul ESP32

Periférie

ESP8266 má 34 vstupno-výstupných pinov, na ktorých je možné využiť následujúce periférie: [5]

I²C Obsahuje dve I²C rozhrania, ktoré sú schopné pracovať v oboch režimoch („master“ aj „slave“) pri maximálnej frekvencii 5 MHz.

UART ESP8266 disponuje tromi plnohodnotným rozhraniami `UART0`, `UART1` a `UART3`. Maximálna rýchlosť prenosu je 5 Mb/s.

SPI Tento mikročip obsahuje štyri hardwarové SPI rozhrania. Jedno z rozhraní je použité na komunikáciu s externou flash pamäťou. Pre užívateľa tak typicky zostávajú tri nevyužité SPI rozhrania.

[1] Prevzaté z: https://rlx.sk/sk/bezdratove-riesenia/5584-esp-wroom-32-esp32-wifi-bt-ble-mcu-module-er-wcw32110w.html

I²S Dve I²S rozhrania, obidve schopné pracovať ako vstup alebo ako výstup.

A/D prevodník ESP32 disponuje 8 kanálovým ADC1 a 10 kanálovým ADC2 prevodníkom. ADC1 je plne k dispozícii užívateľovi. ADC2 prevodník je dostupný obmedzene, pretože väčšinu kanálov využíva bezdrôtový modul.

PWM Ktorýkoľvek z pinov je možné využiť na pulzne šírkovú moduláciu, najviac však dokopy 16 nezávislých signálov.

Podpora pre šifrovanie Hardwarová podpora šifrovacích algoritmov ako napríklad AES, SHA, RSA alebo ECC.

Čítač pulzov K dispozícii je 7 rôznych režimov počítania. Čítač disponuje ôsmimi kanálmi, pričom každý kanál je schopný zaznamenávať až 4 signály.

SD/SDIO/MMC kontrolér ESP32 podporuje niekoľko režimov pripojenia pamäťových kariet.

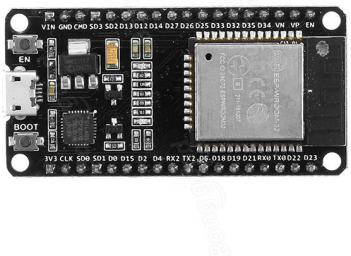
Integrovaná Hallova sonda V mikročipe je integrovaná hallova sonda, ktorej výstup je možné interne pripojiť interný zosilňovač a na A/D prevodník.

D/A prevodník Obsahuje dva osem bitové D/A prevodníky.

Dotykový senzor Mikročipu má 10 GPIO pinov, ktoré sú schopné detekovať zmeny spôsobené napríklad dotykom prsta na vstupe príslušného pinu.

Dostupné vývojové kity

Typov vývojových dosiek obsahujúcich modul `ESP32` je oproti modulu `ESP8266` oveľa viacej. Niektoré majú osadený displej, obvod pre nabíjanie batérie alebo rôzne senzory, u iných je zase možné pridávať funkcie jednoducho pripojením ďalšieho modulu (tzv. „shieldu“). Ceny týchto modulov sa pohybujú od 140 Kč (Geekcreit ESP32 3.5) kľudne až po 1500 Kč (napríklad modul M5Stack 3.6)



[1]

Obrázek 3.5: Geekcreit ESP32

[2]

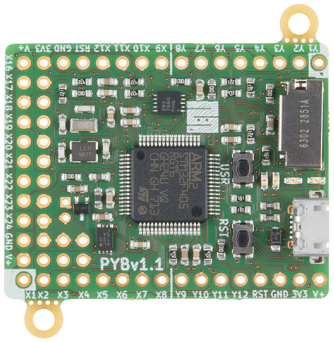
Obrázek 3.6: M5Stack

[1] Prevzaté z:
https://www.banggood.com/ESP32-Development-Board-WiFiBluetooth-Ultra-Low-Power-Consumption-Dual-Cores-ESP-32-ESP-32S-Board-p-1109512.html

[2] Prevzaté z:
https://www.amazon.co.jp/ILS-M5Stack-MPU9250-Development-Extensible/dp/B07912FXJ2

Pyboard

Modul Pyboard bol vydaný spolu s uvedením softwaru Micropython, po úspešnom financovaní na platforme Kickstarter. S jeho parametrami je možné ho zaradiť po boku modulov ESP. Obsahuje procesor Cortex M4 bežiaci na frekvencii 168 Mhz, 1024 kB flash pamäte ROM a 192 kB pamäte RAM. Pre užívateľa je dostupných 29 GPIO pinov, tri 12-bitové AD prevodníky, dva 12-bitové DA prevodníky, 4 integrované LED diódy alebo slot na Micro SD kartu. [15]



[2]

Obrázek 3.7: Modul Pyboard

Rozdiely v hardwardovej výbave niesu príliš odlišné od modulov ESP, najmä od modulu `ESP32`. Pyboard sa líši najmä cenou, ktorá je pri dnešnom kurze 660 Kč. Modul má výbornú softwarovú podporu. Keďže je už od začiatku vyvíjaný a určený pre beh Micropythonu, je v ňom implementovaných najviac funkcií a modulov spomedzi mikrokontrolérov podporujúcich beh Micropythonu.

3.2 Možnosti tvorby rekonfigurovateľného softwaru na ESP32 a ESP8266

Pre platformy ESP8266 a ESP32 je možné písať programy okrem staticky kompilovaného jazyka C aj v niektorom z interpretovaných jazykov. Túto možnosť je vhodné zvážiť pri tvorbe rekonfigurovateľného systému, ktorý má byť schopný spúšťať za behu definované aplikácie. V následujúcej kapitole budú popísané najrozšírenejšie z dostupných implementácií interpretovaných jazykov pre dané mikrokontroléry.

Medzi nesporné výhody písania programov v interpretovaných jazykoch, ako je napríklad Python, patrí aj možnosť spúšťania za behu vytvoreného programového kódu, čo je priamo využiteľné pri tvorbe rekonfigurovateľného softwaru.
Nasleduje stručný súhrn podporovaných programovacích jazykov:

Lua

Open source port Lua jazyka pre mikrokontroléry, využívajjúci port eLua. [16] Pre ESP8266 aj ESP32 existuje niekoľko implementácií. Pre ESP8266 je zrejme najznámejšia NodeMCU. Po štarte je pre užívateľa dostupných cca 40 kB pamäte RAM z celkových 128 kB. Port má dobrú dokumentáciu a množstvo knižníc.

[2] Prevzaté z: https://learn.sparkfun.com/tutorials/pyboard-hookup-guide

JavaScript

Espruino - Open source implementácia JavaScriptu pre mikrokontroléry. [10] Po štarte zostáva pre užívateľa dostupných 20 kB operačnej pamäte z celkových 128 kB. Programátor môže využiť Espruino Web IDE, ktoré umožňuje nahrávanie skriptov do zariadenia a komunikáciu s bežiacim programom.

Micropython

Micropython je opensourcová implementácia programovacieho jazyka Python 3, napísaná v C, optimalizovaná na beh v mikrokontroléroch. [1] Po spustení na ESP8266 je voľných cca 30 kB operačnej pamäte z celkových 128 kB.

Vývoj MicroPythonu začal v roku 2013, po úspešnom financovaní na platforme Kickstarter. Pôvodne bol určený pre beh na platforme Pyboard, no postupne sa rozšíril aj na ďalšie platformy. Financovanie portu na ESP8266 bolo opäť uskutočnené cez Kickstarter. Cieľom je čo najväčšia možná kompatibilita s CPython. Odlišnosti, ktoré bolo nutné spraviť z dôvodov optimalizácie sú zdokumentované v prehľadne spracovanej dokumentácii. [14]

MicroPython obsahuje aj nástroj uPip, ktorý je obdobou nástroja pip z klasického Pythonu. Vďaka nemu je možné jednoducho sťahovať dostupné moduly do zariadenia, bez nutnosti nahrávania cez počítač. Množstvo dostupných knižníc je zrejme najväčšie z porovnávaných jazykov. Nahrávanie a komunikácia s prostredím prebieha cez prostredie REPL. K dispozícii je na niektorých moduloch (napríklad `ESP8266`) aj webREPL.

Nástroje

Pre pracovanie s Micropythonom na moduloch `ESP8266` a `ESP32` existuje niekoľko nástrojov, ktoré sa líšia svojimi možnosťami, vzhľadom, ale aj kvalitou spracovania.

Prvou možnosťou komunikácie so zariadením je využitie akéhokoľvek univerzálneho programu pre komunikáciu cez sériovú linku. Príkladom sú: Putty, minicom, screen, Arduino IDE a ďalšie. Komunikácia prebieha bezproblémovo, ale pre pohodlnejšiu prácu je vhodnejšie využiť nástroje prispôsobené priamo na komunikáciu s Micropythonom. Slabina obecných aplikáciú sa ukáže napríklad pri prenášaní súborov z modulu alebo do modulu.

Ďalšou možnosťou sú nástroje priamo určené pre Micropython. Po vyberaní a testovaní niekoľkých nástrojov (`rshell`, `mpfShell` alebo `ampy`) som sa rozhodol použiť nástroj `mpfShell`.

Mpfshell[1] je konzolová utilita, určená pre komunikáciu s Micropython modulmi. Medzi jej hlavné výhody patrí, že v celom rozhraní je použité ovládanie rovnaké, ako v konzolovom prostredí linuxu. Po spustení je nutné určiť, na ktorom sériovom rozhraní je modul, s ktorým chceme komunikovať. Po pripojení môžeme zobrazovať súbory, ktoré sú nahraté v module príkazom `ls`. Zobrazenie modulov, súborov ktoré sa nachádzajú v aktuálnej zložke v počítači slúži príkaz `lls`. Pre kopírovanie z a do modulu slúži príkaz `put` a `get`. Po zadaní kľúčového slova `repl` sa dostaneme do komunikačného rozhrania s modulom.

Na inštalovanie firmwaru do modulu je určená utilita **esptool**. Podporuje `ESP8266` aj `ESP32`. Firmware vie do modulu nielen nahrávať, ale aj zálohovať. Pri provom nahrávaní a pri čistení modulov sa odporúča kompletne vymazať flash pamäť, na čo je určená funkcia `erase_flash`, .

[1] https://github.com/wendlers/mpfshell

Kapitola 4

Analýza a Návrh systému

Cieľom tejto práce je navrhnúť a vytvoriť systém s podobnými možnosťami rekonfigurácie ako má PNOS s využitím interpretovaného jazyka namiesto Petriho sietí. V prípade kompatibility komunikačného protokolu bude možné, aby v jednej sieti fungovali a komunikovali spolu zariadenia bežiace s PNOS aj zariadenia s mnou navrhnutým systémom (MPOS[1]). V tejto kapitole sú popísané požiadavky na systém a návrh samotného systému.

4.1 PNOS

PNOS je dynamicky rekonfigurovateľný operačný systém pre vstavané zariadenia, špecifikovaný s využitím petriho sietí. Špecifikácia systému sa skladá z popisov modulov systému, ich vnútorných procesov a komunikácie medzi modulmi a je popísaná pomocou referenčných Petriho sietí. Využitie petriho sietí pre špecifikáciu prináša zjednodušenie overovania správnosti aplikácií. Popis v petriho sietiach je reprezentovaný formou bytecodu, ktorý je následne reprezentovaný vo virtuálnom stroji (súčasť operačného systému).

Jeden z cieľov PNOS je umožniť dynamickú regonfigurovateľnosť nasadeného systému, v závislosti na zmenách v jeho špecifikácii. To umožňuje systému reagovať na zmenu požiadaviek za behu systému. Pre zmenu konfigurácie nie je nutné systém pozastaviť ani reštartovať. Dynamickej rekonfigurovateľnosti bolo dosiahnuté dekompozíciou celkovej funkcionality na relatívne malé interpretovateľné výpočetné časti. Tie časti, ktoré nie sú aktuálne rekonfigurované, ostanú bežať nedotknuté. [18]

Samotný beh takto špecifikovaného systému potom stojí na platforme PNOS, ktorá na interpretáciu hostovaných sietí využíva vistuálny stroj Petri Nets Virtual Machine (PNVM).

Systém PNOS využíva na komunikáciu medzi oddelenými interpretovanými časťami protokol, z ktorého vychádza aj komunikačný protokol MPOS. Pre študijné účely som mal k dispozícii poslednú, zatiaľ nepublikovanú verziu PNOS, v ktorej je použitý mierne odlišný protokol než v predchádzajúcich verziach. Správy v PNOS sa skladajú z troch častí.

Prvá časť správy určuje či sa jedná o unicastovú alebo broadcastovú správu. Keďže komunikácia prebieha cez protokol MQTT, ktorý neumožňuje odosielanie unicastových správ, toto rozlíšenie sa vykonáva až pri prijatí správy.

Druhá časť obsahuje buď cieľovú alebo zdrojovú adresu, podľa toho o aký typ správy sa jedná. Adresa je rozdelená na dve až tri polia podľa typu správy.

Tretia časť správy je jej samotné telo. Jej obsah a forma závisí na type správy.

Tento protokol je aj s rozšíreniami pre MPOS popísaný v kapitole 4.6.

[1] Micro Python Operating System

4.2 Analýza

Tak ako PNOS aj tento systém by mal byť distribuovaný, čo prináša niekoľko výhod. V prípade výpadku niektorého z uzlov môže byt zvyšok systému schopný pracovať naďalej, čo by v prípade centralizovaného riešenia mohlo byť problematické. Jednoduchšie je aj pridávanie nového alebo vymenenie už existujúceho uzla, ktorý je možné vymeniť bez zasiahnutia do zvyšných modulov zapojených do siete.

Systém by mal umožňovať inštalovanie, spúšťanie a zastavovanie užívateľských aplikácí. Keďže ostatné aplikácie, ktoré aktuálne nie sú upravované, môžu vykonávať dôležité úlohy a komunikovať so zvyškom systému, bolo by vhodné, aby pri týchto úkonoch nebol ich chod nijako obmedzený ani narušený.

Vstupy a výstupy aplikácií by mali byť smerovateľné do inej aplikácie bežiacej na rovnakom uzle, ako aj do aplikácie bežiacej na ktoromkoľvek z iných modulov vrámci danej siete. Smerovanie sa bude ukladať na uzloch, z ktorých bude príslušná správa vychádzať. Výstup jednej aplikácie bude možné presmerovať na viacero vstupov iných aplikácií. Toto smerovanie by taktiež malo byť možné meniť a nastavovať za behu bez nutnosti zastavenia aplikácií.

Systém by mal aplikáciam poskytovať jednoduché rozhranie pre komunikáciu s inými aplikáciami. Správy budú odosielané cez protokol MQTT a komunikácia by mala byť plne v réžii operačného systému uzla. Ďalej by mal systém poskytovať aplikáciam rozhranie pre uloženie jednoduchých parametrov. Tieto parametre budú uložené vo flash pamäti modulu a budú tak dostupné aj po jeho reštartovaní.

4.3 Prípad užitia

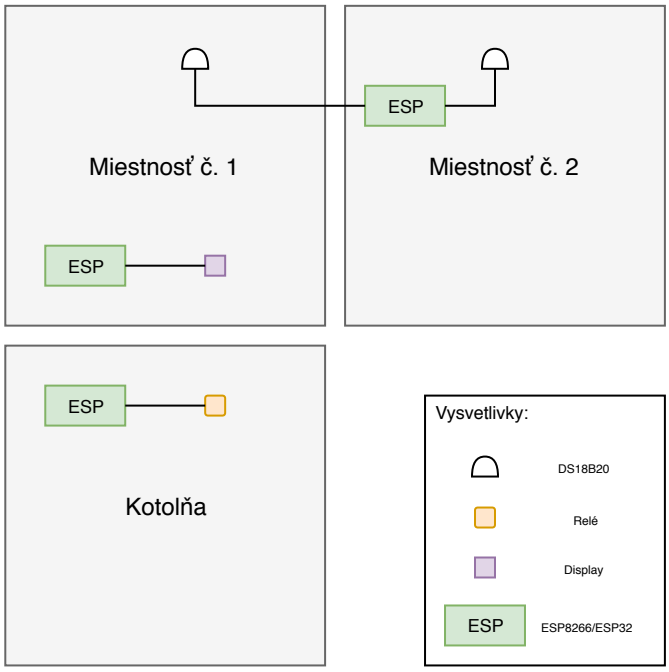
Pre lepšie pochopenie problematiky a nájdenie možného výskytu problémov som navrhol prípad užitia systému. Mnou navrhnutý systém bude pokusne nasadený v domácnosti, kde bude riadiť niekoľko komponentov.

Popis problému

Zjednodušená domácnosť sa skladá z dvoch obývaných izieb. V obidvoch miestnostiach sú radiátory s termostatickými hlavicami a v jednej z nich je termostat spínajúci kotol. Pokiaľ v prvej z miestnosti klesne izbová teplota pod nastavenú hranicu (napríklad po vyvetraní), izbový termostat zopne kotol a teplota sa postupne zvýši. Pokiaľ klesne teplota v druhej miestnosti a teplota v miestnosti s termostatom ostane nezmenená, kotol na túto zmenu nezareaguje a v miestnosti je zima.

Návrh riešenia

Navrhol som do oboch miestností umiestniť teplotné čidlá a spínať kotol, ak teplota klesne pod nastavenú hranicu v ktorejkoľvek z nich. Pri správnom nastavení termostatických hlavíc na radiátoroch by nemalo dôjsť k prekúreniu miestností nad požadovanú teplotu. Keďže miestnosti zdieľajú jednu stenu, navrhol som využiť jeden modul ESP8266 s dvomi pripojenými čidlami DS18B20. Modul s jedným z čidiel bude umiestnený v jednej miestnosti a druhé čidlo bude umiestnené do druhej miestnosti (pripojené cez prevŕtanú stenu). Ďalší z modulov bude umiestnený v blízkosti kotla, ktorý bude podľa potreby spínať cez relé. Pre sledovanie stavu teplôt a kúrenia bude slúžiť tretí modul, ku ktorému bude pripojený displej zobrazujúci aktuálny stav. Náčrt návrhu je možné vidieť nav obrázku 4.1.



Obrázek 4.1: Schéma miestností doplnená o navrhnuté riešenie

4.4 Štruktúra aplikácií

Kvôli obmedzenému výkonu zrejme nebude možné implementovať preemptívny multitasking. Preto je vhodné s týmto počítať už pri návrhu a navrhnúť vhodné riešenie prepínania aplikácií. Samotné užívateľské aplikácie sa budú skladať z následujúcich častí:

Inicializačná časť Tento úsek kódu bude spustený iba jeden krát pri spustení aplikácie. Môže slúžiť na inicializáciu premenných, nastavenie čidiel, nastavenie prerušenia, započatie komunikácie a podobne.

Programová sľučka Úsek kódu, ktorý je možné v nastavenom intervale spúšťať znovu. Vhodné napríklad na periodické čítanie teploty z teplotného čidla.

Ukončujúca časť Bude spustená pred ukončením aplikácie. Táto časť je určená napríklad na korektné ukončenie spojenia, ohlásenie vypnutia alebo uvoľnenie zdrojov

Prijímacia časť Voliteľná časť aplikácie, v ktorej bude možné spracúvať príchodzie správy a reagovať na ne. Táto časť bude spustená pri prijatí správy určenej pre príslušnú aplikáciu.

Obsluha prerušenia Funkcia, ktorá bude zavolaná v prípade obsluhy nastaveného prerušenia.

4.5 Výber implementačného jazyka a platformy

Výber vhodného modulu som zvažoval podľa ceny, výkonu, podporovaných programovacích jazykov a všeobecnej podpory daného modulu. Pri porovnaní `ESP32` a `ESP8266` vyhráva `ESP32` vo výkone a `ESP8266` vo zvyšných kategóriach. `ESP8266` stojí 50% ceny `ESP32` (pri

dnešných cenách 70 Kč a 140 Kč). `ESP8266` je tiež na trhu dlhšie a podporuje preto viacero programovacích jazykov a má pre ne lepšiu podporu a dokumentáciu. `ESP32` je výkonnejšie, má väčšiu pamäť, viacej GPIO pinov a periférií. Každé z nich má svoje výhody a preto som sa chcel pokúsi nájsť možnosť vyvinúť systém kompatibilný s obomi z nich, čo ovplyvnilo moje hľadanie vhodného programovacieho jazyka.

Pri výbere vhodného interpretovaného programovacieho jazyka som vyberal z troch, uvedených v kapitole 3.2. Pre jazyk Lua existujú porty pre oba moduly, no nie sú plne kompatibilné. To by vyžadovalo osobitný vývoj alebo aspoň prispôsobovanie systému pre obe zariadenia. Oproti tomu micropython aj JavaScript majú takmer rovnaké API pre obi dva moduly.

Micropython má oproti Javascriptu po spustení viacej voľnej operačnej pamäte pre užívateľa a lepšiu podporu pre kooperatívny multitasking. Pre vývoj som sa teda rozhodol použiť Micropython, s cieľom pokúsiť sa vytvoriť jeden systém, ktorý bude možné spustiť a prevádzkovať ako na `ESP32`, tak aj na `ESP8266`.

4.6 Navrh komunikačného protokolu

Pri návrhu protokolu som vychádzal z protokolu využívanom v PNOS (v jeho poslednej, zatiaľ nepublikovanej verzii). Tento protokol je rozšírený o niekoľko ďalších správ, ktoré uľahčujú prácu so systémom. V následujúcej sekcii je popísaný výsledný protokol.

Každá správa poslaná medzi modulmi sa skladá z troch častí, ktoré sú oddelené čiarkami. Ako celok je správa uzavretá v zložených zátvorkách, rovnako ako jej podčasti, ktoré tiež môžu byť zložené z viacerých častí (príklad 4.1).

Prvá časť určuje či sa jedná o správu určenú pre konkrétny uzol, alebo o správu odoslanú „broadcastovo“. V prípade, že sa v prvej časti nachádza `"."`, je správa určená pre jeden konkrétny uzol. Ak sa v nej ale nachádza `"*"`, správa nemá jednoznačne určeného prijímateľa a je určená pre všetky uzly. Väčšina správ je posielaných pre jeden konkrétny uzol. Broadcastovo sa typicky posielajú odpovede na servisné správy, chybové hlásenia alebo informatívne správy o prihlásení nového uzla. Aplikácie tiež majú možnosť odoslať „broadcastovú“ správu. Spracovanie broadcastových správ na uzloch je značne obmedzené. Typicky sa broadcastové správy nedajú smerovať aplikáciam a pokiaľ sa nejedná o niektorý z kľúčových príkazov, uzol na tieto správy ani nereaguje.

Druhá časť môže mať dva významy:

a) Pokiaľ sa jedná o správu určenú pre jeden uzol, obsahuje druhá časť údaje o cieľovom uzle. Tieto údaje sa skladajú z troch prvkov. Prvý určuje adresu cieľového uzla (IP adresa, prípadne meno). Druhý prvok určuje cieľovú aplikáciu, ktorá má správu dostať alebo sa jedná o jedno z kľúčových príkazov na ovládanie systému (napríklad pridávanie alebo odoberanie aplikácie – viacej v 4.7). V prípade, že sa jedná o správu určenú pre aplikáciu, správa musí obsahovať ešte tretí prvok – port aplikácie. Jedná sa o port, na ktorom cieľová aplikácia očakáva správu.

b) Pokiaľ správa nemá jasne určený cieľ (v prvej časti správy je `"*"`), v druhej časti sa nachádzajú údaje o odosielateľovi. V tomto prípade sa správa skladá z dvoch častí. V prvej časti je zapísaná IP adresa odosielateľa a v druhej je predmet správy.

Tretia časť obsahuje samotné telo správy. V prípade že správa je odosielaná medzi aplikáciami, forma správy je plne v ich réžii. Ak sa jedná o servisnú správu, je jej formát pevne daný (viac v časti 4.7).

```
{".",{"192.168.1.2","display","room1_temp"} ,"23.9"}
```

Pr. 4.1: Unicastová správa určená aplikácii `display` bežiacej na uzle s IP adresou `192.168.1.2`, doručená na port `room1_temp`. Obsah správy je `23.9`.

4.7 Servisné správy

V tejto časti sú popísané správy, ktorých cieľom nie je aplikácia, ale samotný systém bežiaci na danom module. Podľa ich názvu a obsahu môžu rôznym spôsobom zisťovať informácie o systéme a aplikáciach, ktoré na ňom bežia, upravovať ich alebo zastavovať.

Nasledujú príkazy s vysvetlením funkčnosti a príkladom správy:

load (príklad 4.2) Zaslaním správy typu `load` sa nahrávajú aplikácie do modulu. V tele správy musí byť samotný zdrojový kód. Formát zdrojového kódu správy je popísaný v kapitole 5.3.

```
{".",{"192.168.1.2","load"} ,{"""zdrojový kód"""} }
```

Pr. 4.2: Správa nahrávajúca do modulu nový zdrojový kód aplikácie. Obsahuje iba jeden parameter – zdrojový kód nahrávanej aplikácie.

unload (príklad 4.3) Zaslaním správy `unload` sa odstráni zdrojový kód z modulu. V tele správy je jediný parameter – názov zdrojového kódu určeného k odinštalovaniu

```
{".",{"192.168.1.2","unload"} ,{"názov"} }
```

Pr. 4.3: Správa odoberajúca z modulu zdrojový kód aplikácie. Obsahuje názov odoberaného zdrojového kódu.

activate (príklad 4.4) Po nahratí zdrojového kódu do modulu je možné spustiť ho pod zvoleným názvom. Jeden zdrojový kód môže byť spustený viacej krát s rôznymi názvami. V tele správy je názov, pod ktorým bude aplikácia bežať a názov zdrojového kódu, z ktorého bude aplikácia vytvorená. Správa môže byť doplnená aj o ďalšie parametre, ktoré budú predané do aplikácie pri jej spustení. Počet, forma a obsah týchto parametrov záleží na aktivovanej aplikácii.

```
{".",{"192.168.1.2","activate"} ,{"názov z. kódu","názov","prípadné ďalšie
                                  parametre"} }
```

Pr. 4.4: Správa aktivujúca novú aplikáciu v module. Povinne obsahuje názov aplikácie a názov zdrojového kódu, ktorý bude aplikácii pridelený. Správa môže obsahovať aj ďalšie parametre, ktoré budú predané aplikácii pri jej spustení.

deactivate (príklad 4.5) Aplikácia sa dá deaktivovať zaslaním príkazu `deactivate`

```
{".",{"192.168.1.2","deactivate"} ,{"názov"} }
```

Pr. 4.5: Správa odstraňujúca aplikáciu z modulu. Obsahuje názov aplikácie určenej k vypnutiu.

restart (príklad 4.6) Bezodkladne reštartuje modul. Telo správy môže byť prázdne.

```
{".",{"192.168.1.2","restart"} ,{ } }
```

Pr. 4.6: Správa reštartujúca dotazovaný modul.

status (príklad 4.7) Vyžiada zaslanie správy o stave modulu. Telo správy môže byť prázdne. Odpoveď správy je vo formáte popísanom na obrázku 5.3.

```
{".",{"192.168.1.2","status"} ,{ } }
```

Pr. 4.7: Správa vyžadujúca zaslanie informácií o stave modulu.

runningtime (príklad 4.8) Vyžiada zaslanie správy obsahujúcej dobu behu modulu. Telo správy môže byť prázdne. Odpoveď správy obsahuje jeden parameter – dobu behu.

```
{".",{"192.168.1.2","runningtime"} ,{ } }
```

Pr. 4.8: Správa vyžadujúca zaslanie doby behu systému.

addroute (príklad 4.9) Slúži na pridávanie záznamov do smerovacej tabuľky. Telo správy obsahuje zdroj a zoznam cieľov. Ciel aj zdroj majú tvar IP adresa, meno aplikácie, port.

```
{".",{"ESP_TEMP","addroute"} ,{{{"ESP_TEMP","srcapp","srcport"}
            ,{{"ESP_DST","dstapp","dstport"} } } } }
```

Pr. 4.9: Správa pridávajúca záznamy do smerovacej tabuľky.

removeroute (príklad 4.10) Slúži na odoberanie záznamov do smerovacej tabuľky. Telo správy obsahuje zdroj a zoznam cieľov. Ciel aj zdroj majú tvar IP adresa, meno aplikácie, port. Zo smerovacej tabuľky sú odobraté iba tie uzly, ktoré sú v tele správy. Pokiaľ v smerovacom pravidle neostane žiaden s cieľových uzlov, je záznam z tabuľku vymazaný komplente.

```
{".",{"ESP_TEMP","removeroute"} ,{{{"ESP_TEMP","srcapp","srcport"}
            ,{{"ESP_DST","dstapp","dstport"} } } } }
```

Pr. 4.10: Správa odoberajúca záznamy zo smerovacej tabuľky.

wifiapmode (príklad 4.11) Pomocou tejto správy sa dá zariadenie prepnúť do nastavovacieho režimu. Viac v 4.8

```
{".",{"192.168.1.2","wifiapmode"} }
```

Pr. 4.11: Prepína zariadenie do wifiap módu.

settingsweb (príklad 4.12) Ukončí beh aplikácií a spustí na module webové rozhranie. Viac v 4.8

```
{".",{"192.168.1.2","settingsweb"} }
```

Pr. 4.12: Prepína zariadenie do `settingsweb` módu.

discover (príklad 4.13) Príkaz discover je nutné odosielať v broadcastovom režime. Po prijatí tejto správy odpovie každé zariadenie v danej sieti.

```
{"*",{"192.168.1.2","discover"} }
```

Pr. 4.13: Broadcastová správe, ktorá vyžaduje odoslanie základných informácií od všetkých pripojených uzlov.

4.8 Nastavovací režim

Do modulov som navrhol aj nastavovací režim, v ktorom je možné meniť parametre potrebné k správnemu behu systému. V tomto režime sa dajú spravovať pripojenia k wifi sietiam a k ním príslušné adresy MQTT serverov. Ďalej je tu možné zmeniť názov zariadenia, reštartovať modul, zresetovať modul do počiatočného stavu a meniť aplikáciami uložené parametre. Tento modul sa spustí v prípade, že nebude dostupná žiadna z uložených wifi sietí, nebude dostupný MQTT server alebo užívateľ vyvolá `wifiapmode` príkazom poslaným cez MQTT. V prípade nezasiahnutia užívateľom sa po 5 minútach modul reštartuje a opäť sa pokúsi pripojiť k sieti a MQTT serveru. Pre zobrazenie režimu je nutné pripojiť sa k vytvorenej wifi sieti a otvoriť adresu `192.168.4.1` vo webovom prehliadači. Bezdrôtová sieť je prednastavene nezabezpečená heslom a pomenovaná podľa sériového čísla modulu. Meno aj heslo sa dá zmeniť po prihlásení do `wifiap` módu.

Pokiaľ už je systém funkčný a pripojený k bezdrôtovej sieti, je možné spustiť aj `settingsweb`, čo je rovnaké webové rozhranie aké je popísané v predchádzajúcom prípade. Táto stránka ale nie je dostupná na vytvorenom bezdrôtovom prístupovom bode, ale na sieti, ku ktorej je modul aktuálne pripojený. Detaily implementácie sú v kapitole 5.5.

Kapitola 5

Implementácia a Testovanie

Implementácia a testovanie prebiehali takmer výhradne na `ESP32`. Spustenie systému na `ESP8266` vyžaduje pribalenie zdrojových kódov do firmwaru Micropythonu pri kompilácii, čím sa tento proces výrazne predlžuje. Aj napriek tomu som sa snažil po implementovaní funkčného celku otestovať a odladiť funkčnosť aj na `ESP8266`. V tejto kapitole budú popísané detaily implementácie a predvedenie funkcionality na demopríklade.

5.1 Použité knižnice

Pri vývoji som využíval najmä štandardné knižnice dostupné v Micropythone. Knižnica `uasyncio`[1], ktorá implementuje časť knižnice asyncio z Pythonu, pomáha pri riadení kooperatívneho mutlitaskingu. Poskytuje možnosti, ako sa jednotlivé koprogramy môžu vzdávať procesoru pre prospech iných koprogramov a možnosti naplánovania behu za požadovaný čas. Ďalej sa v systéme využívajú knižnice pre tvorbu socketov, komunikáciu s hardwarom a knižnica `umqtt.simple`[2]. Jedná sa o podobnú knižnicu akou je mqtt knižnica v CPythone. Knižnice uasyncio a umqtt.simple bolo treba do modulov doinštalovať. Ostatné použité knižnice sa už všetky nachádzali v základe firmwaru Micropython.

5.2 Moduly systému

Systém pre moduly `ESP8266` a `ESP32` je rozdelený do dvoch spolu komunikujúcich modulov. Okrem týchto modulov je ešte malá časť zdrojových kódov aj v module `_boot.py`. Tento modul Micropython spustí ihneď po spustení systému. Sú v ňom naimportované potrebné moduly, inicializácie modulov `tasker` a `config` a je tu ošetrený prípadný pád systému. Keďže na platformách `ESP8266` a `ESP32` bohužiaľ nie je v Micropythone implementovaný Watchdog, je potenciálny pád ošetrený odchytávaním všetkým výnimiek, ktoré nastanú pri behu hlavnej slučky programu. Pri odchytení výnimky, ktorá nebola odchytená v moduloch tasker, config alebo v bežiacej aplikácii, je táto chyba odchytená v súbore `_boot.py` a je vynútený reštart systému. Pri vynútenom reštarte nie je systém vždy schopný odosielať správy (záleží od chyby, ktorá nastala), ale ak je to možné, systém o tomto reštarte odošle správu ešte pred jeho vykonaním. O reštarte sa môžu prípadné monitorovacie systémy dozvedieť aj zo správy typu `new_node`, ktorú systém odosiela pri každom štarte.

[1] https://github.com/micropython/micropython-lib/tree/master/uasyncio
[2] https://github.com/micropython/micropython-lib/tree/master/umqtt.simple

Modul config

Tento modul sa spúšťa z modulu `_boot.py` pred modulom `tasker`, aby mu pripravil potrebné nastavenia a pripojil modul k sieti. Funkčne sa dá rozdeliť do troch častí:

Prvá funkcia je ukladanie nastavení. Modul sa stará o čítanie, zapisovanie konfiguračného súboru do flash pamäte modulu a poskytuje rozhranie na čítanie a zapisovanie informácií do konfiguračného súbora. Pri prvom spustení modulu je vo flash pamäti vytvorený konfiguračný súbor vo formáte JSON, do ktorého sú uložené prednastavené hodnoty požadovaných parametrov (príklad na obrázku 5.1). Po uložení a pri každom ďalšom spustení modulu sú údaje z konfiguračného súbora uložené do asociatívneho poľa, kde s nimi môže modul ďalej pracovať. Zvyšku systému a aplikáciam umožňuje ukladať a načítať požadované parametre funkciami `put` a `get`. Pri čítaní sú údaje načítané z premennej, takže nie je nutné čítať zo súbora. Pri zmene dát sa aktuálne údaje uložia aj do konfiguračného súbora, aby boli dostupné aj po reštarte modulu.

```
{ 'wifi': [{'pass': '123456789', 'mqtt': '192.168.1.5', 'ssid':
'TP_LINK'}], 'ap_name' : 'ESP_15740b00', 'ap_password' : ' ', 'topic' :
                                'pnos'}
```

Pr. 5.1: Ukážka konfiguračného súbora vytvoreného pri prvom štarte systému.

Druhá funkcia je pripojenie modulu k wifi sieti. Po štarte systému sú vyhľadané bezdrôtove siete, ktoré sú v dosahu a sú porovnané so sieťami, ktorých prihlasovacie údaje sú uložené v konfiguračnom súbore. Pokiaľ je záznam o niektorej z dostupných sietí nájdený, modul sa k nej pokúsi pripojiť. Ak je pripojenie úspešné, v nastaveniach je vyplnená adresa `MQTT` servera, ktorá bola zadaná spolu s prihlasovacími údajmi o bezdrôtovej sieti, aby bola dostupná pre zvyšok systému. Pokiaľ je pripojenie neúspešné alebo modul nenašiel žiadnu bezdrôtovú sieť, ku ktorej by sa mohol pripojiť, prepne sa do wifiap módu.

Tretia funkcia modulu config je spustenie a obsluha wifiap módu. Pokiaľ je zavolaná funkcia `apMode`, je vytvorený bezdrôtový prístupový bod a spustený webový server čakajúci na pripojenie. Po pripojení sa užívateľovi zobrazí webová stránka, v ktorej má možnosť upravovať všetky parametre uložené v konfiguračnom súbore. Webová stránka (obrázok 5.1) je minimalistická a neposkytuje žiadne zložité dialógy, z dôvodu šetrenia pamäťových zdrojov modulu (pri zložitejších návrhoch bol problém s pamäťou na `ESP8266`). Na stránke sa nachádzajú dve tabuľky. V prvej z nich je možné upravovať nastavenia uložených bezdrôtových sietí. V druhej sú všetky ostatné nastavenia, ktorých hodnoty sú zobrazené v textovej podobe. Pre zmenu je potrebné vypísať príslušné políčka `edit` a stlačiť tlačidlo „Save“. Úspešnosť zmeny je možné poznať na zmenených údajoch. Pre reštartovanie modulu slúži tlačidlo reset a pre uvedenie do pôvodných nastavení (zmazanie konfiguračného súboru) slúži tlačidlo „Factory reset“.

To, že je modul prepnutý do wifiap módu sa dá zistiť pomocou dostupnosti bezdrôtového prístupového bodu s príslušným názvom a svietením integrovanej LED diódy na module. V prípade neúspešnosti pripojenia alebo neaktivity zo strany užívateľa sa systém po piatich minútach automaticky reštartuje a pokúsi sa opäť pripojiť k niektorej z preddefinovaných bezdrôtových sietí. Modul je tiež možné prepnúť do módu

`settingsweb`, ale až pri úspešnom pripojení k bezdrôtovej sieti. Tento režim je rovnaký ako režim `apmode` s jediným rozdielom. Modul pri prepnutí zostáva pripojený k sieti a spustí webový server na IP adrese, ktorú má pridelenú.

Okrem týchto troch hlavných úloh sú v tomto module implementované aj menej dôležité funkcie, akou je napríklad synchronizácia času. Tá prebehne vždy pri štarte systému a dostupnosti internetového pripojenia.

Modul sa spustí funkciou `init()`. Z tejto funkcie sa postupne spúšťajú ďalšie funkcie, v ktorých sa modul inicializuje. Ako prvé sa načítajú dáta z flash pamäte, nasleduje pripojenie k bezdrôtovej sieti a nakoniec synchronizácia času. Po úspešnom spustení sa pokračuje inicializáciou modulu `tasker`.

Modul tasker

Druhý z modulov je modul starajúci sa o beh systému – modul tasker. Tento modul sa spúšťa až ako druhý po module `config`, aby mal pripravené všetky prostriedky, ktoré potrebuje. Modul `tasker` očakáva, že pri jeho spustení je už modul úspešne pripojený k bezdrôtovému prístupovém u bodu a je možné sa dotazovať na všetky systémové nastavenia potrebné k behu. Implementácia v module sa dá rozdeliť do niekoľkých častí, ktoré sú popísané v následujúcich riadkoch:

Funkcia init

V module sa spúšťa ako prvá funkcia `init()`. V tejto funkcii sú definované niektoré globálne premenné získané po komunikácii s modulom config. Ďalej sa systém pokúsi o nadviazanie komunikácie s MQTT serverom. Pokiaľ je spojenie neúspešné, spustí funkciu `apmode` v module config. Pokiaľ je pripojenie úspešné, je odoslaná správa `new_node` (jej príklad – 5.2). Z tejto broadcastovej správy sa dá zistiť IP adresa modulu, názov modulu, typ zariadenia (`ESP8266` alebo `ESP32`) a počet sekúnd od štartu systému. Obsahom je správa podobná, ako odpoveď na príkaz `status`.

```
        {"*","{"192.168.1.2","new_node",{{"node_name","ESP_obyvak"}
,{"platform","esp8266"} ,{"IP","192.168.1.2"} {"running_time","5"} } } "}
```

Pr. 5.2: Ukážka správy typu `new_node` odosielanej po spustení systému. Správa obsahuje niekoľko informácií o systéme.

Funkcia loop

Po inicializácii už je riadenie predané knižnici `uasyncio`, ktorá postupne spúšťa naplánované úlohy – aplikácie a obsluhu systému. Obsluha systému je volaná z funkcie `loop` naplánovanej na čo najčastejšie spustenie, aké je možné. Toho je dosiahnuté volaním funkcie `asyncio.sleep(0)`. Pokiaľ čaká iná aplikácia na beh, je jej to umožnené. Pokiaľ nečaká, je spustená opäť funkcia `loop`. Vo funkcii je periodicky volaná obsluha knižnice `umqtt`, ktorá pri príchode novej správy zavolá príslušnú funkciu, kde prebieha jej ďalšie spracovanie. Ďalej táto funkcia kontroluje buffer `outcommingMessages`. Pokiaľ buffer obsahuje nejakú správu, je z bufferu vybratá a predaná na spracovanie. Vybraná a spracovaná je maximálne jedna správa pri jednom behu funkcie, z dôvodu urýchlenia obsluhy ostatných aplikácií.

Spracovanie odchodzích správ

Odchádzajúce správy sú spracovávané vo funkcii `sendMessage`. Spracovanie týchto správ môže prebiehať niekoľkými spôsobmi, v závislosti na ciele správy a obsahu smerovacích tabuliek. Správy odchádzajúce z aplikácií môžu byť určené pre aplikáciu vrámci rovnakého uzla, alebo pre aplikáciu bežiacu na inom uzle. Pri spracovaní je nahliadnuté do smerovacej tabuľky. Pokiaľ je v smerovacej tabuľke medzi zdrojmi nájdená odosielajúca aplikácia spolu s príslušným portom správy, sú postupne spracované všetky ciele smerovacieho pravidla. Ak je ciel vrámci rovnakého uzla, je mu správa doručená lokálne bez použitia protokolu `MQTT`. Pokiaľ sa jedná o vzdialený uzol správa je odoslaná cez protokol `MQTT`. Ak by sa zdroj správy nenachádzal v smerovacej tabuľke, správa typicky nebude odoslaná. Odoslaná by bola v prípade, že je zapnutá možnosť `broadcast_everything` v nastaveniach systému. Pri zapnutí tejto možnosti sú všetky správy (úspešne nájdené v smerovacej tabuľke, ale aj tie nenájdené) odosielané broadcastovo cez `MQTT` protokol. Využitie odosielania všetkej komunikácie je napríklad pri ladení alebo pri monitorovaní behu celého systému (viac v časti 5.9).

Spracovanie prichádzajúcich správ

Pri spracovaní prichádzajúcich správ sú v prijatom texte zamenené znaky „`{`" a „`}`" za „`[`" a „`]`" a text je prevedený na objekt pomocou vstavanej funkcie Micropythonu (s využitím funkcie `eval`). Pokiaľ sa text nedá previesť na objekt, je odoslaná broadcastová správa „`message_format`" s popisom chyby (príklad 5.3).

```
{"*",{"192.168.1.2","message_format"} ,{"Error","invalid syntax"} }
```

Pr. 5.3: Ukážka odpovede modulu pri chybnom formáte vstupnej správy.

Pokiaľ je formát správy v poriadku, následuje kontrola, či je daná správa unicastová alebo broadcastová, čo je zistené z prvej časti správy (popis protokolu je v časti 4.6). V prípade broadcastovej komunikácie sa môže jednať o jedinú správu, na ktorú môže systém reagovať a to o správu typu discover. Odpoveď na túto správu je unicastová s rovnakým predmetom („`discover`") (príklad 5.4).

```
{".",{"192.168.1.1","discover"} ,{"ESP_obyvak","192.168.1.2","21"} }
```

Pr. 5.4: Ukážka odpovede modulu na správu typu `discover`.

Pri prijatí unicastovej správy sa kontroluje či je správa určená uzlu, ktorý ju prijal. Kontroluje sa zhodnosť adresy v správe a IP adresy pridelenej modulu. Pokiaľ je v nastaveniach povolená možnosť „`route_by_name`", porovnáva sa adresa v správe aj s menom uzla. Ak je správa určená uzlu ktorý ju prijal, je ďalej porovnávaný predmet správy s kľúčovými slovami protokolu.
Nasleduje popis správania sa systému pri jednotlivých predmetoch správ:

status Pri prijatí správy status systém broadcastovo odošle správu obsahujúcu informácie o systéme. Informácie o spustených a nainštalovaných aplikáciach sú získané z globálnych polí. IP adresa, voľná pamäť, doba behu a platforma sú zistené volaním príslušných funkcií z modulov Micropythonu. Získané údaje sú predané funkcii `broadcastSend`, ktorá ich zformátuje a odošle.

load Prehľadá sa či názov inštalovanej aplikácie už nie je v zozname nainštalovaných aplikácií. Ak sa v systéme už nachádza zdrojový kód s rovnakým názvom, inštalovanie sa preruší a je odoslaná broadcastová správa s predmetom „`load`“ a textom „`'Error', ,'Redefinition'`“. Ak chyba nenastane, je spustený zdrojový kód, ktorý obsahuje definíciu triedy. Týmto je trieda pridaná do systému a je možné vytvárať jej inštancie. Meno triedy je uložené do zoznamu nainštalovaných aplikácií.

unload Ak sa názov odinštalovávanej aplikácie nachádza v zozname nainštalovaných, je spustený príkaz `del` s názvom aplikácie – triedy, čím je trieda odstránená.

activate V prípade, že je spúšťaná aplikácia nainštalovaná – v systéme existuje trieda s rovnakým menom ako zdrojové kódy aplikácie, je vytvorený objekt triedy so zaslaným menom a jeho meno je uložené do zoznamu spustených aplikácií. Po vytvorení objektu sú mu priradené parametre, ktoré mu boli zadané pri spustení a je naplánovaný beh metódy `loop` v danom objekte.

deactivate Ak v systéme beží aplikácia s dotazovaným menom = v systéme existuje objekt s dotazovaným menom, pristúpi sa k jeho ukončeniu. Ak neexistuje, je odoslaná broadcastová chybová správa s predmetom `deactivate` a telom správy popisujúcim chybu. Ukončenie prebieha zmenením parametra `running`, ktorý trieda objetku dedí od triedy Task, následkom čoho pri ďalšom behu aplikácie beh vyskočí zo sľučky `while`. Nakoniec je názov aplikácie je zmazaný zo zoznamu bežiacich aplikácií.

restart Broadcastovo je odoslaná správa potvrdzujúca reštart a je zavolaná funkcia `reset` z knižnice `machine`, ktorá vyvolá okamžitý reštart systému.

wifiapmode Broadcastovo je odoslaná správa potvrdzujúca spustenie `apmode` a je spustená funkcia `apMode` z modulu config.

settingseb Broadcastovo je odoslaná správa potvrdzujúca spustenie `settingsweb` a je spustená funkcia `settingsHttp` z modulu config.

Pokiaľ sa predmet správy nezhodoval ani z jedným s predchádzajúcich príkazov, je predmet porovnaný s menami bežiacich aplikácií. Pokiaľ taká aplikácia existuje, je v príslušnom objekte zavolaná metóda `recieve`. Ak neexistuje, je broadcastovo odoslaná chybová správa s predmetom `routing`.

Triedy v module

Pre zjednodušenie práce s aplikáciami, správami a smerovacou tabuľkou sú využité nasledujúce štyri triedy:

Point Obsahuje parametre `ip`, `taskName` a `port`. Umožňuje jednoznačnú identifikáciu zdroja alebo cieľa. Využíva s smerovacej tabuľke.

Route Predstavuje jedno pravidlo zo smerovacej tabuľky. Obsahuje jeden zdroj typu point a pole cieľov typu point. Sú v nej definované funkcie na pridávanie a odoberanie cieľových bodov.

Task Pomocná trieda, od ktorej musia dediť všetky triedy definujúce aplikácie. Je v nej definovaná metóda stop, ktorá odošle správu `deactivate` cez `MQTT` protokol, metóda `_stop`, ktorá nastaví premennú `running` na False a metóda `send`, pomocou ktorej

môžu aplikácie odosielať správy. Funkcia send pridáva objekty typu `Message` do globálneho bufferu na odoslanie (`outcommingMessages`).

Message Pomocná trieda na vytvorenie objektov, ktoré obsahujú zdroj a obsah, ktorý má byť odoslaný. Využíva sa pri odosielaní správ.

5.3 Užívateľské aplikácie

Pre nahratie a spustenie užívateľskej aplikácie je potrebné ju napísať v požadovanom formáte. Rozhodol som sa, že nahrávané zdrojové kódy pre aplikácie budú mať formu triedy. Pre uľahčenie spravovania a implementácie je nutné, aby táto trieda aplikácie dedila od triedy `Task`, pretože v tejto triede sú definované funkcie umožňujúce aplikácii komunikáciu s okolím a premenné, ktoré sú vyžadované na riadenie behu aplikácie.

V texte je uvádzaný pojem „aplikácia“, ktorý bol zvolený, pretože z pohľadu operačného systému sa jedná o aplikácie (je nutné ich inštalovať, aktivovať, majú vstupy a výstupy,...). V skutočnosti sa však jedná skôr o mikroaplikácie alebo mikroslužby, keďže celá požadovaná funkcionalita je rozdelená do niekoľkých menších logicky funkčných blokov, ktoré sú vykonávané samostatne.

Zápis zdrojového kódu aplikácií

Meno triedy zdrojového kódu aplikácie sa musí zhodovať s názvom zadaným pri príkaze `activate`. Táto trieda môže obsahovať ľubovoľný počet užívateľských metód, no musí obsahovať minimálne funkciu `loop`. Táto funkcia je zavolaná pri aktivácii aplikácie. Pre udržanie behu aplikácie je vnútri funkcie slučka while, závislá na premennej `self.running`. `Self.running` je premenná nastavená pri aktivácii aplikácie na `True`. Pri prijatí príkazu deactivate sa premenná nastaví na False, čo umožní vyskočenie z programovej slučky a ukončenie aplikácie. Vyvolať príkaz `deactivate` má aplikácia možnosť aj sama (detailnejšie v 5.3). Zdrojový kód aplikácie sa podľa návrhu skladá z piatich častí (ukážka 5.1):

Prvá časť sa nachádza pred programovou slučkou `while`. Táto časť je vykonaná iba raz a to pri spustení aplikácie. Je to vhodné miesto na inicializácie premenných alebo na importovanie knižníc.

Druhá časť je v samotnej slučke `while`. V aplikácii nie je vhodné používať funkciu `sleep()`, pretože je vhodné namiesto aktívneho čakania umožniť beh iným aplikáciam. Vzdanie sa procesoru a naplánovanie ďalšieho behu sa vykonáva zavolaním funkcie z knižnice uasyncio – `await asyncio.sleep(n)`, s parametrom ďalšieho naplánovaného času spustenia v sekundách. Existuje aj funkcia `sleep_ms(10)`, ktorá umožňuje aj naplánovanie behu za požadovaný čas v milisekundách. Pokiaľ aplikácia nevyžaduje čakať, ale chce umožniť ostatným aplikáciam pridelenie procesoru, môže to urobiť zavolaní `await asyncio.sleep(0)`. Týmto umožní beh prípadným aplikáciam čakajúcim na procesor, ale ak taká aplikácia neexistuje, vráti ho ihneď aplikácii, ktorá sa ho vzdala. Pre plynulý beh systému a aplikácií je vhodné volať niektorú z týchto funkciu čo najčastejšie. Aplikácie by nemali pracovať nepretržito dlhšie než pár stoviek milisekúnd. So zvyšovaním dĺžky behu jednej aplikácie sa predlžuje doba odozvy ostatných aplikácií a celkovo aj doba odozvy systému.

Ďalšia časť je voliteľná funkcia recieve. Metóda recieve je zavolaná pri prijatí unicastovej správy určenej pre príslušnú aplikáciu. Obsahuje dva parametre: `port` a `payload`. Je vhodné si prijaté údaje čo najrýchlejšie spracovať / poznačiť a umožniť beh ďalším aplikáciam.

Pokiaľ aplikácia nepotrebuje prijímať správy, nie je nutné túto funkciu implementovať. V takomto prípade ak je aplikácii poslaná správa, systém správu zahodí.

Voliteľné je aj spracovanie prerušenia, ktorého je možné nastaviť v prvej časti kódu. Jeho obsluha je potom typicky ďalšia definovaná metóda.

Poslednou časťou kódu aplikácie je časť za slučkou while. Táto časť je vykonaná iba raz a to pri deaktivovaní aplikácie.

Aplikácii je možné predať ľubovoľný počet parametrov. Tieto parametre sú uložené do poľa `self.params`. V tomto poli sú všetky parametre obsiahnuté v správe `activate`. Prvý je názov zdrojového kódu aplikácie, druhý názov pod akým je aplikácia spustená. Následujúce parametre sú všetky voliteľné a závislé na aplikácii. V prípade aplikácie na príklade 5.1 je voliteľný parameter `number_of_inputs`, ktorý určuje počet vstupov logického súčtu.

```
#[And, app_name,number_of_inputs]
class And(Task):
   async def loop(self):
      self.output=0
      interval=200
      self.number_of_inputs=2
      if len(self.params)>2 and self.params[2].isdigit():
         self.number_of_inputs=int(self.params[2])
      self.inputs=dict()
      while self.running:
         await asyncio.sleep(interval)

   def calculateOutput(self):
      if self.number_of_inputs<=0:
         self.output=0
         first = self.inputs.values()
         if all(val==0 for val in self.inputs.values()) or all(not val==0 for val
             in self.inputs.values()):
            self.output=1
         super().send("output",str(self.output))

   def recieve(self,port, payload):
      if not port in self.inputs:
         number_of_inputs-=1
      if payload.isdigit():
         self.inputs[port]=int(payload)
         self.calculateOutput()
```

Výpis 5.1: Aplikácia, na ktorej výstupe je logický súčet všetkých vstupov. Počet vstupov je dopredu definovaný pri spúšťaní aplikácie a svoj výstup odosiela až poprijatí hodnôt zo všetkých vstupov.

Systémové funkcie dostupné pre aplikácie

Nainštalované aplikácie môžu využívať následujúce funkcie

super.send('port','data') Slúži na odosielanie dát iným aplikáciam. Pre doručenie inej aplikácii je nutné správne nastavenie smerovania. Pokiaľ neexistuje smerovacie pravidlo pre danú aplikáciu s daným portom, je správa odoslaná ako broadcast.

super.stop() Naplánuje ukončenie behu aplikácie. Je realizované odoslaním MQTT deactivate správy.

config.put('name','data') Slúži na uloženie ľubovoľných dát do pamäte flash. Je možné prepisovať aj systémom uložené parametre.

config.get('name') Slúži na načítanie uložených dát z pamäte flash. Je možné čítať aj systémom uložené parametre, ako napríklad nastavenia smerovania, wifi, ... (viac v 5.4)

rtc.datetime() umožňuje získať čas z RTC modulu (po importovaí knižnice machine). Pre zlepšenie presnosti je nutné volať `ntp.settime()` z knižnice ntptime, čím dôjde k zosynchronizovaniu času v RTC module s internetovým časom. Systém túto synchronizáciu prevádza iba raz a to pri štarte systému.

Zistenie stavu aplikácií

Pre zistenie aktuálne bežiacich a nainštalovaných aplikácií je možné zaslať systému správu typu `status`. Z odpovede (príklad 5.5) na túto správu je možné získať zoznam bežiacich aplikácií, zoznam nainštalovaných zdrojových kódov v systéme, meno modulu, čas doby behu systému v sekundách, voľnú operačnú pamäť a platformu na ktorej systém beží.

```
{"*","192.168.1.2","status",{{"node_name","ESP_obyvak"}
,{"platform","esp8266"} ,{"IP","192.168.1.2"} ,{"running_time","364"}
,{"memory","15328"} ,{"installed",{"Sensor","Or","And"} }
,{"running",{"ds15b20","or1","And"} } } }
```

Pr. 5.5: Ukážka odpovede na správu typu `status`.

Preddefinované aplikácie

Pre ukázanie možností je k systému priložených niekoľko univerzálnych aplikácií. Niektoré z nich sú použité aj v demopríklade. Tieto aplikácie majú čo najmenšiu funkcionalitu. Poväčšinou majú jedinú úlohu – napríklad odosielať teplotu, robiť logický súčet alebo súčin, porovnávať dve hodnoty a podobne. Takýto návrh aplikácií zvyšuje možnosť ich znovupoužitia.
Funkčne by sa dali aplikácie rozdeliť do troch skupín:

Sensor Ich úlohou je periodicky sa dotazovať rôznych senzorov alebo reagovať na asynchrónne vstupy od čidiel alebo spínačov. Namerané údaje potom preposielajú ďalej na spracovanie. V demopríklade je príkladom aplikácie typu Sensor senzor teploty alebo aplikácia spracúvajúca asynchronne vstupy od tlačidla – button.

Logika Tieto aplikácie dostávajú správy od aplikácií typu Sensor, rôzne ich kombinujú a spracúvajú a na základe vstupov posielajú správy Actuatorom. V demoaplikácii je príkladom týchto typov aplikácií aplikácia Or alebo Setpoint.

Actuator Je typicky aplikácia, ktorá ovláda externe pripojený hardware (svetlo, relé, displej,...). V demopríklade pôsobí ako actuator aplikácia Kotol.

5.4 Formát a obsah nastavení

Systém ukladá dôležité nastavenia pre beh systému a užívateľsky uložených parametrov do pamäte typu flash, ktorej obsah zostane nezmenený aj po reštarte systému. Dáta sú uložené v súbore `settings.dat` v súborovom systéme vytvoreným Micropythonom. Tento súbor je textovom zápise formátu `JSON` (príklad súboru 5.2).

```
{
   "route_by_name":"1",
   "broadcast_everything":"1",
   "mqtt":"192.168.1.5",
   "topic":"pnos",
   "node_name":"ESP_OBYVAK",
   "ap_password":"",
   "wifi":
      [
         {
            "ssid":"NETGEAR"
            "pass":"password",
            "mqtt":"server.example.com",
         },
         {
            "ssid":"TP-link"
            "pass":"Password",
            "mqtt":"192.168.1.5",
         }
      ],
   "userdata":"123456"
}
```

Výpis 5.2: Príklad uložených parametrov v súbore `settings.dat`.

Systémové nastavenia

Medzi nastavenia uložené a využívané systémom patria následujúce:

node_name Meno príslušného uzla. Pod týmto menom sa bude uzol hlásiť `MQTT` serveru a používa sa aj pri umožnení smerovania podľa mena (`route_by_name`). Pri spustení `apmode` bude toto meno použité ako názov prístupového bodu.

route_by_name Môže byť 0 alebo 1, predvolená hodnota je 1. Určuje, či má daný uzol použiť na smerovanie okrem IP adresy aj meno uzla.

broadcast_everything Môže nadobúdať hodnoty 0 alebo 1. Pri zapnutí bude uzol preposielať všetky kópie správ broadcastovo, čo môže slúžiť napríklad na potreby monitorovania funkčnosti systému.

mqtt Nie je nastavované užívateľom v nastaveniach. Jedná sa o kópiu názvu aktuálne používaného servera zisteného z wifi nastavení.

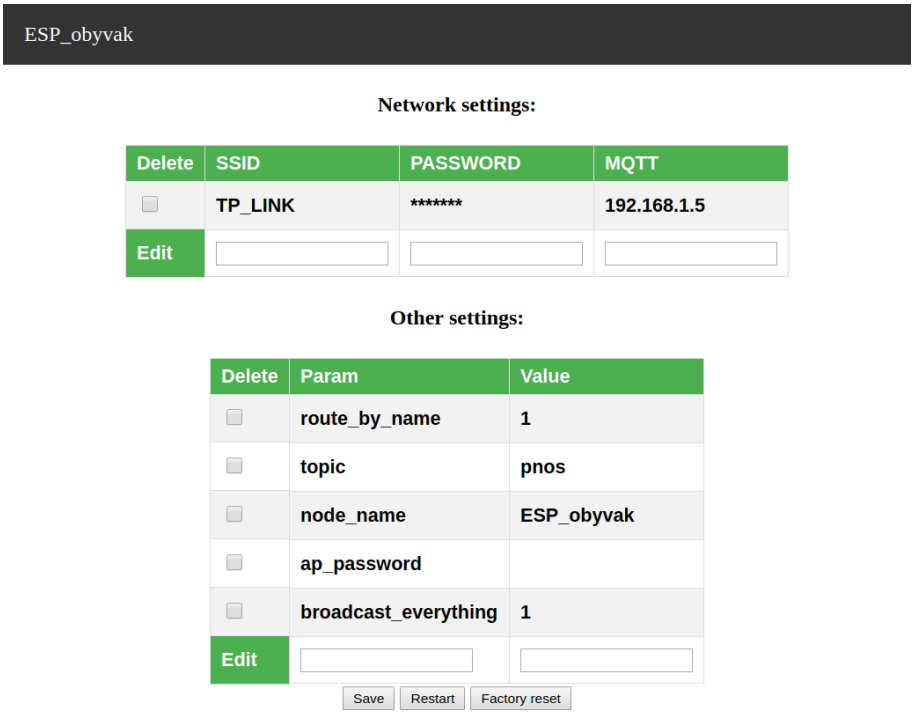
wifi Slúži na uloženie prihlasovacích údajov k bezdrôtovým prístupovým bodom. Je vo forme poľa objektov obsahujúcich tri prvky: `ssid` je názov bezdrôtovej siete, `pass` je heslo k bezdrôtovej sieti a `mqtt` je adresa `MQTT` servera použitá pri pripojení k tejto sieti.

Užívateľské nastavenia

Užívateľské aplikácie majú možnosť pristupovať k uloženým parametrom a aj ukladať svoje parametre do spoločného súbora `settings.dat`, pomocou funkcií `get` a `put` popísaných v 5.3. Obsah a názvy ukladaných parametrov sú plne v réžii tvorcu aplikácie. Užívateľ by si mal dať akurát pozor na nechcené prepísanie systémových nastavení (ak by zvolil rovnaký názov, akým je pomenovaný systémový parameter).

5.5 Webové rozhranie pre nastavenia

Podľa návrhu je v systéme implementované webové rozhranie pre modifikáciu nastavení. Do takéhoto rozhrania je možné systém prepnúť po zaslaní príkazu `appmode` alebo `settingsweb`. Systém sa do tohoto režimu prepne aj sám a to v prípade neúspešného pripojenia k bezdrôtovej sieti alebo `MQTT` serveru.



Obrázek 5.1: Webová stránka umožňujúca zmenu konfigurácie modulu

Webový server je implementovaný pomocou BSD schránok, ktoré poskytuje Micropython po importovaní modulu socket. Po zapnutí `apmode` sytém čaká na príchodzie spojenie. Po nadviazaní spojenia je klientovi odoslaná html šablóna doplnená o dáta získané z JSON objektu z nastaveniami. Pri aktualizácii alebo pridávaní nového parametra sa po stlačení

tlačidla `Save` zmenené parametre odošlú s využitím formulára metódou POST do modulu, ktorý ich spracuje pomocou regulárnych výrazov a aktualizuje uložené nastavenia. Pre zníženie veľkosti html súboru a kódu potrebného na obsluhu webového servera bolo pristúpené k takémuto minimalistickému návrhu bez využitia kontextových ponúk a množstva formulárov. Tieto prvky by síce mohli umožniť pohodlnejšie nastavovanie, no procedúra nastavovania nie je hlavnou časťou systému a preto je dôležitejšie zachovanie voľnej pamäte modulu.

V prípade, že so serverom nebude komunikované dlhšie ako 5 minút, bude systém reštartovaný, čo je realizované pomocou funkcie `settimeout()` na používanom sockete. Toto umožní modulu vyhľadať a pokúsiť sa pripojiť znovu k dostupným uloženým bezdrôtovým sietiam.

5.6 Nahrávanie knižníc do systému

Pre beh na operačného systému je nutné zdrojové súbory pribaliť k firmwaru Micropythonu z dôvodu malej pamäte modulu. Malá pamäť modulov znemožňuje aj použitie nástroja `upip` na inštaláciu knižníc do modulu. Preto som sa rozhodol do firmwaru pribaliť nielen zdrojové súbory systému, ale aj niektoré z najpoužívanejších knižníc pre prácu s externým hardwarom. Sú to knižnice:

encoder.py Knižnica na prácu s rotačnými enkódermi.
https://github.com/SpotlightKid/micropython-stm-lib/tree/master/encoder

pcd8544.py Knižnica poskytujúca nástroje pre komunikáciu s displejom pôvodne určeným pre mobilné telefóny NOKIA 5110.
https://github.com/mcauser/micropython-pcd8544

ssd1306.py Knižnica umožňujúca prácu s OLED displejmi s radičmi ssd1306, s využitím protokolom i2c a SPI.
https://github.com/adafruit/Adafruit_Python_SSD1306

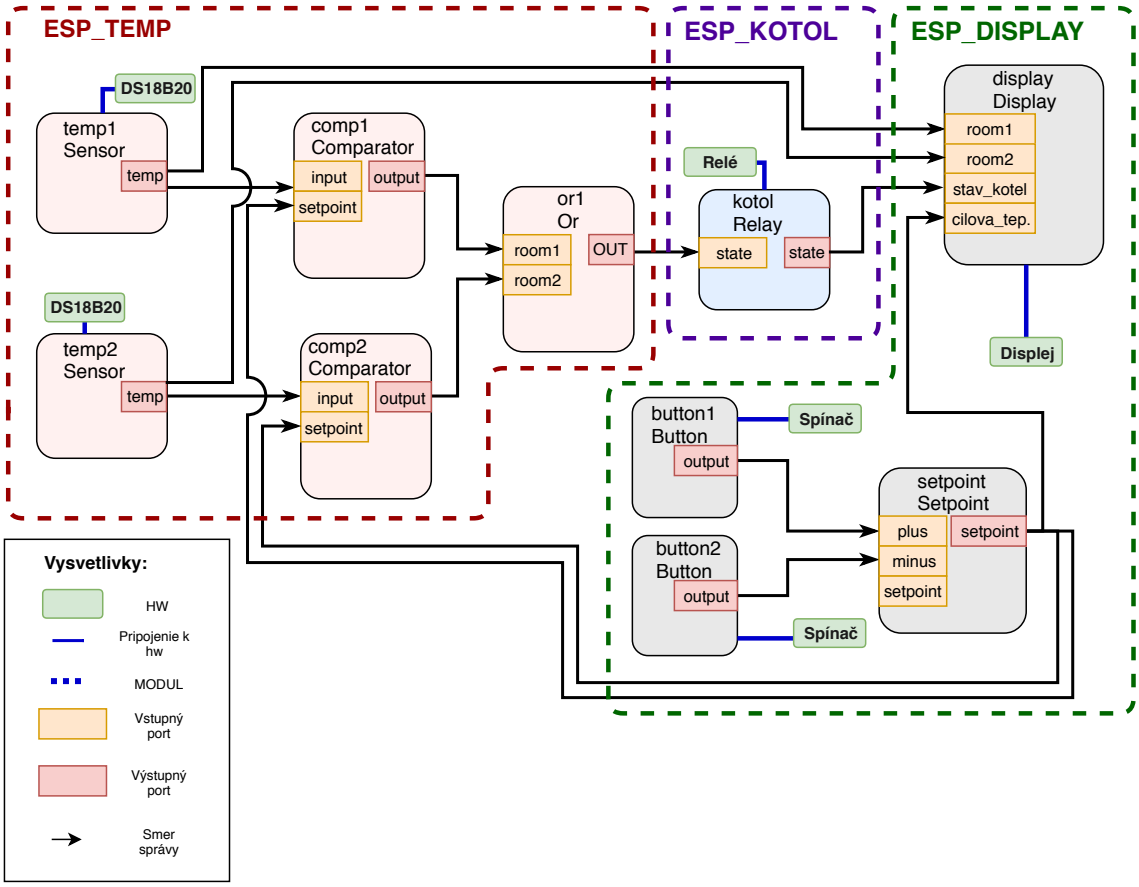
Ďalšie užitočné knižnice sú už obsiahnuté v pôvodnom firmware Micropythonu. Sú to napríklad knižnice umožňujúce komunikáciu s DHT senzormi, DS18B20 alebo knižnica Onewire.

V prípade, že by užívateľ chcel do systému nahrať ďalšiu knižnicu, existujú dva spôsoby. Ak by sa jednalo o knižnicu s malou veľkosťou a bola by nahrávaná do výkonnejšieho modulu `ESP32`, stačí do modulu knižnicu nahrať po pripojením k počítaču napríklad utilitou `mpfshell` (popísaná v 3.2). Ak je nutné nahrať knižnicu do modulu `ESP8266` alebo je knižnica rozsiahlejšia a nie je možné ju nahrať predchádzajúcim spôsobom ani do `ESP32`, je nutné ju pribaliť k firmwaru Micropythonu pri jeho kompilácii. V takom prípade je nutné stiahnuť kompletné zdrojové súbory Micropythonu. Pre uľahčenie tohoto postupu sú k dispozícii pokyny spolu s virtuálnym strojom, ktorý obsahuje všetky potrebné súbory Micropythonu.[2] K systému je potom nutné pribaliť nielen požadované knižnice, ale aj zdrojové súbory MPOS (jedná sa o súbory `_boot.py`, `tasker.py` a `config.py`).

Hotový firmware obsahujúci zdrojové súbory operačného systému, Micropythonu a knižníc je možné nahrať do modulu napríklad pomocou nástroja `esptool`, ktorý je popísaný v kapitole 3.2

5.7 Ukážka na demoaplikácii

Vytvorený systém bol otestovaný nasadením na riešenie problému s vykurovaním miestností, popísaného v sekcii 4.3. Na riešenie sú použité tri moduly, ktoré sú pripojené k rovnakej bezdrôtovej sieti a MQTT serveru. Všetky moduly majú povolené parametre `broadcast_everything` a `route_by_name`. Zjednodušený nákres so zobrazením modulov, aplikácií, vstupov a výstupov aplikácií je možné vidieť na obrázku 5.2. Riešenie je navrhnuté s použitím väčšieho množstva menších a univerzálnejších aplikácií. Vďaka tomu je možné jeden typ aplikácie použiť na viacerých miestach. Takéto dve aplikácie sa potom nemusia líšiť len svojím menom, ale môžu mať aj mierne odlišnú funkcionalitu vďaka parametrom pri ich spúšťaní. V samotnom module je týmto spôsobom možné ušetriť miesto, pretože namiesto dvoch zdrojových kódov aplikácií je potreba nahrať len jeden.



Obrázek 5.2: Ukážka zapojenia a komunikácie aplikácií spustených na moduloch. Jednotlivé moduly sú ohraničené v blokoch s prerušovanými čiarami.

Nastavenia modulov a bežiace aplikácie bežiace v moduloch sú nasledovné:

Modul 1

Názov:	ESP_TEMP
IP adresa:	192.168.1.2
Typ ESP:	ESP8266
Umiestnenie:	na spoločnej stene obytných miestností
HW doplnky:	2 x DS18B20
Aplikácie:	temp1 : Sensor
	temp2 : Sensor
	comp1 : Comparator
	comp2 : Comparator
	or1 : Or

Aplikácie `temp1` a `temp2` majú rovnaký zdrojový kód s názvom Sensor. Aplikácie sa líšia číslom pinu ku, ktorému je pripojené čidlo teploty. Tento pin ako aj ďalšie nastavenia (ako je napríklad typ senzora, interval odosielania teploty atd.) sú aplikácii predané pri aktivácii. Spustená aplikácia každých 100 sekúnd prečíta a odošle dáta prečítané z teplotného čidla. Čítanie teploty vyžaduje čakanie medzi jednotlivými príkazmi, čo je vyriešené pomocou `asyncio.sleepms(750)`. Aplikácie temp1 a temp2 sú nahrané a aktivované príkazmi 5.6.

```
{".",{"ESP_TEMP","load"} ,{"zdrojový kód Sensor A.1"} }
{".",{"ESP_TEMP","activate"} ,{"temp1","Sensor","100","DS18B20","5"} }
{".",{"ESP_TEMP","activate"} ,{"temp2","Sensor","100","DS18B20","4"} }
```

Pr. 5.6: Správy, ktoré nahrajú zdrojový kód `Sensor` a aktivujú dve aplikácie temp1 a temp2 s týmto z. kódom. Aplikáciam sú pri spustení predané parametre: interval odosielania správy, typ čidla a pin, na ktorom je čidlo pripojené.

Ďalej sú nahraté aplikácie `comp1` a `comp2`. Obidve majú spoločný zdrojový kód („šablónu“) `Comparator`. `Comparator` porovnáva svoje dva vstupy `input` a `setpoint` a na svoj výstup `output` dáva 0 alebo 1. Tieto aplikácie sú nahrané a aktivované príkazmi 5.7.

```
{".",{"ESP_TEMP","load"} ,{"zdrojový kód Comparator A.2"} }
{".",{"ESP_TEMP","activate"} ,{"comp1","Comparator"} }
{".",{"ESP_TEMP","activate"} ,{"comp2","Comparator"} }
```

Pr. 5.7: Správy, ktoré nahrajú zdrojový kód `Comparator` a aktivujú dve aplikácie comp1 a comp2 s týmto z. kódom.

Poslednou aplikáciou bežiacou v tomto module je aplikácia `or1`. Tá používa zdrojový kód `Or`. Aplikácia vykonáva logický súčet všetkých vstupov a výsledok dáva na výstup `output`. Jej nahratie a aktivovanie je realizované príkazmi 5.8.

```
{".",{"ESP_TEMP","load"} ,{"zdrojový kód Or A.4"} }
{".",{"ESP_TEMP","activate"} ,{"or1","Or"} }
```

Pr. 5.8: Správy, ktoré nahrajú zdrojový kód `Or` a aktivujú dve aplikáciu or1 s týmto z. kódom.

Pre správne fungovanie je treba do modulu odoslať ešte záznamy o smerovaniach podľa 5.9. Niektoré porty sú presmerované na aplikácie na rovnakom module, iné na aplikácie na iných moduloch. Smerovanie je možné vidieť aj na obrázku 5.2.

```
        {".",{"ESP_TEMP","addroute"} ,{{{"ESP_TEMP","temp1","temp"}
,{{"ESP_DISPLAY","display","room1"} ,{"ESP_TEMP","comp1","input"} } } } }
        {".",{"ESP_TEMP","addroute"} ,{{{"ESP_TEMP","temp2","temp"}
,{{"ESP_DISPLAY","display","room2"} ,{"ESP_TEMP","comp2","input"} } } } }
      {".",{"ESP_TEMP","addroute"} ,{{{"ESP_TEMP","comp1","output"}
                  ,{{"ESP_TEMP","or1","room1"} } } } }
      {".",{"ESP_TEMP","addroute"} ,{{{"ESP_TEMP","comp2","output"}
                  ,{{"ESP_TEMP","or1","room2"} } } } }
       {".",{"ESP_TEMP","addroute"} ,{{{"ESP_TEMP","or1","output"}
                 ,{{"ESP_KOTOL","kotol","state"} } } } }
```

Pr. 5.9: Správy, ktoré nahrajú smerovanie do modulu ESP_TEMP. Smerovanie je možné vidieť aj na obrázku 5.2.

Modul 2

Názov:	ESP_DISPLAY
IP adresa:	192.168.1.3
Typ ESP:	ESP32
Umiestnenie:	chodba spájajúca obytné miestnosti
HW doplnky:	1x NOKIA 5220 displej
Aplikácie:	display : Display
	button1 : Button
	button2 : Button
	setpoint : Setpoint

Aplikácia `display` so zdrojovým kódom `Display` využíva na komunikáciu s displejom knižnicu `pcd8544`. Periodicky každých 10 sekúnd aktualizuje obsah displeja. Jej zdrojový kód je možné vidieť v prílohe A.7. Na prijatie dát z modulov `ESP_TEMP` a `ESP_KOTOL` je v aplikácii implementovaná funkcia `recieve()`, ktorá prijaté dáta uloží do premenných, odkiaľ sú zobrazené na displeji pri najbližšom prekresľovaní. Nahratie a aktivovanie aplikácie `display` prebieha pomocou príkazov

```
{".",{"ESP_DISPLAY","load"} ,{"zdrojový kód A.7"} }
{".",{"ESP_DISPLAY","activate"} ,{"display","Display"} }
```

Pr. 5.10: Správy, ktoré nahrajú zdrojový kód `Display` a aktivujú aplikáciu `display` s týmto z. kódom.

V module sú ďalej spustené aplikácie `button1` a `button2`, ktoré zdieľajú rovnaký zdrojový kód `Button`. Aplikácie pomocou prerušení reagujú na stlačenie tlačidla na prednastavnom pine (poslanom ako parameter pri aktivácii). Ako reakcia na stlačenie je odoslanie správy na výstup `output`. Aplikácie sú nahrané do modulu pomocou príkazov 5.11.

```
{".",{"ESP_DISPLAY","load"} ,{"zdrojový kód Button A.5"} }
{".",{"ESP_DISPLAY","activate"} ,{"button1","Button","32"} }
{".",{"ESP_DISPLAY","activate"} ,{"button2","Button","33"} }
```

Pr. 5.11: Správy, ktoré nahrajú zdrojový kód `Button` a aktivujú aplikácie `button1` a `button2` s týmto z. kódom. Zadaný parameter je číslo pinu.

Posledná aplikácia spustená na module `ESP_DISPLAY` je aplikácia `setpoint`, ktorá používa zdrojový kód `Setpoint`. Aplikácia má prednastavenú hodnotu, ktorá je do aplikácie poslaná ako parameter. Pri prijatí správy na vstupe `plus` alebo `minus` aplikácia pripočíta alebo odpočíta hodnotu 0.5 od aktuálne uloženej premennej. Výsledok si uloží a zároveň prepošle na výstup setpoint. Aplikácia je nahratá a aktivovaná pomocou príkazov 5.12.

```
{".",{"ESP_DISPLAY","load"} ,{"zdrojový kód Setpoint A.6"} }
{".",{"ESP_DISPLAY","activate"} ,{"setpoint","Setpoint","25"} }
```

Pr. 5.12: Správy, ktoré nahrajú zdrojový kód `Setpoint` a aktivujú aplikáciu `setpoint` s týmto z. kódom. Zadaným voliteľným parametrom je prednastavená hodnota.

Do modulu je treba poslať aj nastavenie smerovania, na čo slúžia príkazy 5.13.

```
{".",{"ESP_DISPLAY","addroute"} ,{{{"ESP_DISPLAY","btn1","output"}
          ,{{"ESP_DISPLAY","setpoint","plus"} } } } } ]
{".",{"ESP_DISPLAY","addroute"} ,{{{"ESP_DISPLAY","btn2","output"}
          ,{{"ESP_DISPLAY","setpoint","minus"} } } } } ]
{".",{"ESP_DISPLAY","addroute"} ,{{{"ESP_DISPLAY","setpoint","setpoint"}
,{{"ESP_TEMP","comp1","setpoint"} ,{"ESP_TEMP","comp2","setpoint"} } } } }
```

Pr. 5.13: Správy, ktoré nahrajú smerovanie do modulu ESP_DISPLAY. Smerovanie je možné vidieť aj na obrázku 5.2.

Modul 3

Názov:	ESP_KOTOL
IP adresa:	192.168.1.4
Typ ESP:	ESP8266 (Sonoff BASIC)
Umiestnenie:	v kotolni
HW doplnky:	1x Relé
Aplikácie:	Kotol : Relay

Do modulu `ESP_KOTOL` je nahratá aplikácia `kotol`, používajúca zdrojový kód `Relay`. Aplikácia kotol spína relé pripojené na pine, ktorého číslo je poslané do aplikácie pri jej spúšťaní. Ďalším parametrom je interval, v ktorom aplikácia odosiela stav relé. Aktivácia a nahranie prebieha pomocou príkazov 5.14.

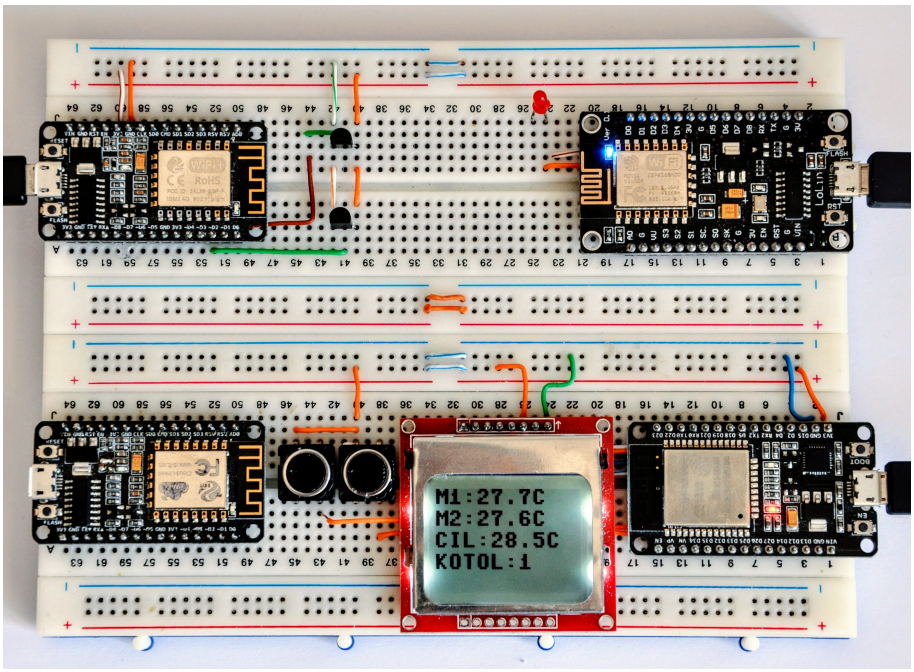
```
{".",{"ESP_KOTOL","load"} ,{"zdrojový kód Relay A.3"} }
{".",{"ESP_KOTOL","activate"} ,{"kotol","Relay","2","1"} }
```

Pr. 5.14: Správy, ktoré nahrajú zdrojový kód `Relay` a aktivujú aplikáciu `kotol` s týmto z. kódom. Zadané parametre sú pin a invertovanie hodnoty na pine.

Pre fungovanie je nutné do modulu nahrať aj záznam o smerovaní. Tento záznam je v prípade tohoto modulu len jeden (5.15).

```
{".",{"ESP_KOTOL","addroute"} ,{{{"ESP_KOTOL","kotol","output"}
      ,{{"ESP_DISPLAY","display","stav_kotel"} } } } } ]
```

Pr. 5.15: Správa, ktoré nahrajú smerovanie do modulu ESP_KOTOL. Smerovanie je možné vidieť aj na obrázku 5.2.



Obrázek 5.3: 4 moduly zapojené na nepájivom poli. V pravo dole je modul `ESP_DISPLAY` s pripojeným displejom a dvoma tlačidlami. Nad ním sa nachádza modul `ESP_KOTOL`, ktorý simuluje relé rozsvecovaní LED diódy. Vľavo hore je modul `ESP_TEMP`, s dvoma senzormi typu DS18B200. Modul v ľavo dole nie je aktuálne zapojený.

Všetky tri moduly boli pokusne zapojené a spustené na nepájivom poli (obr. 5.3). Namiesto pripojenia relé bola k modulu `ESP_KOTOL` pripojená LED dióda, ktorá simulovala zopnutie relé. Keďže systém sa skladá z 10 aplikácií, ktoré každá vykonávajú len jednoduché úlohy, je možné systém dynamicky prispôsobiť novo vzniknutým požiadavkom, čo sa dá jednoducho ukázať na príklade.

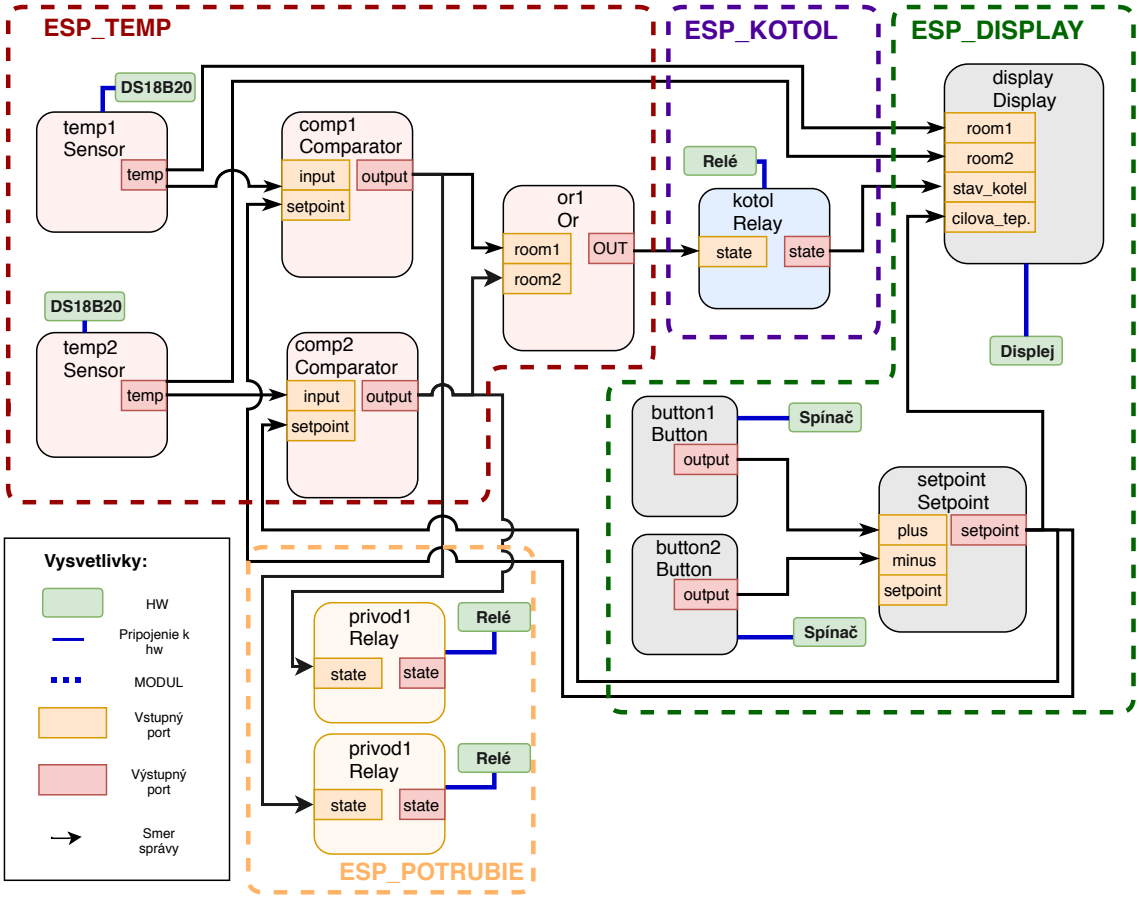
Rekonfigurácia systému

V prípade, že by sme mohli na prívody kúrenia do miestností nainštalovať elektricky ovládané ventily, dal by sa systém jednoducho zdokonaliť. Táto zmena by sa dala realizovať pridaním štvrtého modulu, ktorý bude ovládať ventily na potrubí do oboch miestností. Zmena sa dá celá vykonať po softwarovej stránke bez zastavenia systému, iba pozmenením smerovania a pridania aplikácií do nového modulu.
Zmeny by boli následujúce:

Do nového, štvrtého modulu s názvom ESP_POTRUBIE by bol nahraný zdrojový kód Relay a aktivované dve aplikácie `privod1` a `privod2`. Ďalej by bolo nutné pridať záznamy do smerovacej tabuľky modulu ESP_TEMP. Sekvencia príkazov je zobrazená na 5.16.

```
{".",{"ESP_POTRUBIE","load"} ,{"zdrojový kód A.3"} }
{".",{"ESP_POTRUBIE","activate"} ,{"privod1","Relay","2","1"} }
{".",{"ESP_POTRUBIE","activate"} ,{"privod2","Relay","4","1"} }
{".",{"ESP_KOTOL","addroute"} ,{{{"ESP_TEMP","comp1","output"}
        ,{{"ESP_POTRUBIE","privod1","state"} } } } } ]
{".",{"ESP_KOTOL","addroute"} ,{{{"ESP_TEMP","comp2","output"}
        ,{{"ESP_POTRUBIE","privod2","state"} } } } } ]
```

Pr. 5.16: Príkazy aktivujúce aplikácie `privod1` a `privod2` na module `ESP_POTRUBIE` a príkaz pridavajúci potrebné smerovania do modulu `ESP_TEMP`.



Obrázek 5.4: Ukážka zapojenia a komunikácie aplikácií spustených na moduloch po úprave. Jednotlivé moduly sú ohraničené v blokoch s prerušovanými čiarami.

Rovnako jednoduché je aj presunutie istej časti funkcionality na iný uzol. V demopríklade sa dá akákoľvek aplikácia, ktorá nekomunikuje priamo s hardwarom, presunúť na iný uzol. Takéto jednoduché presúvanie istej časti logiky sa dá využiť napríklad pri nedostatku výkonu alebo pamäte na niektorom z uzlov. Časť aplikácie sa jednoducho presunie na iný,

výkonnejší uzol. Všetky zmeny a presuny sa dajú vykonať bez nutnosti zastavenia alebo reštartovania systému.

5.8 Usnadnenie nahrávania aplikácií

Pre usnadnenie nahrávania aplikácií a smerovania do modulov je vhodné túto procedúru zautomatizovať skriptom. K systému MPOS je priložený skript, ktorý umožňuje toto automatické nahrávanie. Pracuje s dvomi typmi súborov.

Jedny z nich sa nachádzajú v zložke saves a slúžia na definovanie aplikácií a smerovania na moduloch. Názov súborov je `MENO_MODULU.save` alebo `IP_ADRESA_MODULU.save`. V súbore sú iba príkazy typu `activate` a `addroute`, oddelené čiarkami. Správy typu load systém generuje a posiela sám. V druhej zložke `templates` sa nachádzajú zdrojové kódy aplikácií, ktoré môžu byť nahraté do modulov.

Pri spustení skript odošle správu discover, čím získa zoznam všetkých pripojených modulov. Po zistení pripojenia nového modulu skript zistí, či má tento modul uložené nastavenia v zložke save. Pokiaľ nemá, skript ho uloží medzi nepoužívané moduly. Pokiaľ má k novo pripojenému modulu zoznam aplikácií k spusteniu v príslušnom súbore, zaháji posielanie príkazov zo súbora.

Ako prvé sú do modulu odoslané smerovacie príkazy. Ďalej je zistený zoznam už nainštalovaných a bežiacich aplikácií na module, s pomocou správy `status`. Tento zoznam sa porovná s aplikáciami, ktoré majú byť na module spustené (naplánované v súbore zo zložky save). Ak je už aplikácia spustená skript to rozpozná a aplikáciu znovu nespúšťa. Ak aplikácia zatiaľ nebeží, skript zistí či je v module nainštalovaný zdrojový kód príslušnej aplikácie. Pokiaľ nie je, je odoslaný príkaz `load` s požadovaným zdrojovým kódom zo zložky `templates`. Ak už je zdrojový kód v zariadení nainštalovaný, tento krok je preskočený. Posledným krokom je odoslanie príkazu `activate`. Táto sekvencia príkazov sa zopakuje pre všetky aplikácie, ktoré majú byť do modulu nainštalované. Každý s príkazov čaká na odpoveď modulu a overuje, či bol príkaz úspešne vykonaný. O úspešnosti príkazu sa je možné dozvedieť v logu vytváranom v konzole (príklad 5.3). Do logu sú taktiež zapisované novo pripojené uzly alebo reštartovania uzlov s počítadlom reštartov.

Skript je schopný reagovať aj na moduly pripojené za behu pretože monitoruje aj správy typu `new_node`. Monitorovacia funkcia skriptu je opísaná v následujúcej sekcii (5.9).

```
[10:21:19 10.05.2018] Connected to 192.168.1.50
[10:21:20 10.05.2018] New node found: ['ESP_DISPLAY', '192.168.1.51']
[10:21:20 10.05.2018] New node found: ['ESP_TEMP', '192.168.1.30']
[10:21:20 10.05.2018] New node found: ['ESP_KOTOL', '192.168.1.114']
[10:21:24 10.05.2018] Addroute to ['ESP_TEMP', '192.168.1.30']
[10:21:25 10.05.2018] Addroute to ['ESP_TEMP', '192.168.1.30']
[10:21:25 10.05.2018] Addroute to ['ESP_TEMP', '192.168.1.30']
[10:21:26 10.05.2018] Addroute to ['ESP_TEMP', '192.168.1.30']
[10:21:26 10.05.2018] Addroute to ['ESP_TEMP', '192.168.1.30']
[10:21:26 10.05.2018] Routing send
[10:21:27 10.05.2018] Loaded 'Sensor' to ['ESP_TEMP', '192.168.1.30']
[10:21:27 10.05.2018] Activated 'temp2' on ['ESP_TEMP', '192.168.1.30']
[10:21:28 10.05.2018] Loaded 'Comparator' to ['ESP_TEMP', '192.168.1.30']
[10:21:28 10.05.2018] Activated 'comp1' on ['ESP_TEMP', '192.168.1.30']
[10:21:29 10.05.2018] Activated 'temp1' on ['ESP_TEMP', '192.168.1.30']
[10:21:29 10.05.2018] Activated 'comp2' on ['ESP_TEMP', '192.168.1.30']
[10:21:30 10.05.2018] Loaded 'Or' to ['ESP_TEMP', '192.168.1.30']
```

```
[10:21:30 10.05.2018] Activated 'or1' on ['ESP_TEMP', '192.168.1.30']
[10:21:31 10.05.2018] FINISHED ['ESP_TEMP', '192.168.1.30']
[10:21:43 10.05.2018] New node found: ['ESP_TEMP', '192.168.1.30']
[10:21:44 10.05.2018] Reboot ['ESP_TEMP', '192.168.1.30'] Counter:1
[10:21:45 10.05.2018] Addroute to ['ESP_TEMP', '192.168.1.30']
[10:21:46 10.05.2018] Addroute to ['ESP_TEMP', '192.168.1.30']
[10:21:46 10.05.2018] Addroute to ['ESP_TEMP', '192.168.1.30']
[10:21:47 10.05.2018] Addroute to ['ESP_TEMP', '192.168.1.30']
[10:21:47 10.05.2018] Addroute to ['ESP_TEMP', '192.168.1.30']
[10:21:47 10.05.2018] Routing send
[10:21:48 10.05.2018] Loaded 'Sensor' to ['ESP_TEMP', '192.168.1.30']
[10:21:49 10.05.2018] Activated 'temp2' on ['ESP_TEMP', '192.168.1.30']
[10:21:49 10.05.2018] Loaded 'Comparator' to ['ESP_TEMP', '192.168.1.30']
[10:21:50 10.05.2018] Activated 'comp1' on ['ESP_TEMP', '192.168.1.30']
[10:21:51 10.05.2018] Activated 'temp1' on ['ESP_TEMP', '192.168.1.30']
[10:21:51 10.05.2018] Activated 'comp2' on ['ESP_TEMP', '192.168.1.30']
[10:21:52 10.05.2018] Loaded 'Or' to ['ESP_TEMP', '192.168.1.30']
[10:21:52 10.05.2018] Activated 'or1' on ['ESP_TEMP', '192.168.1.30']
[10:21:52 10.05.2018] FINISHED ['ESP_TEMP', '192.168.1.30']
```

Výpis 5.3: Časť logu skriptu `mpos` vypisovaného do konzoly. V logu je popísané nájdenie troch pripojených modulov, aktivácia `ESP_TEMP` a ukážka správania sa v prípade výpadku uzla `ESP_TEMP`. Aktivácie ostatných uzlova

5.9 Monitorovanie

Pre zaistenie nepretržitého chodu distribuovaného systému je vhodné monitorovať jeho chod a snažiť sa zasiahnuť v prípade výpadku niektorej z jeho častí. Na monitorovanie chodu aplikácií na moduloch sa dá využiť funkcia systému `broadcast_everything`. Zapína sa v nastaveniach systému a predvolene je zapnutá. Monitorovanie aplikácií je potom možné sledovaním broadcastovej komunikácie, preto je vhodné na monitorovanie myslieť už pri implementácii aplikácií. Pokiaľ aplikácia pravidelne nekomunikuje so zvyškom systému, je nemožné sledovať jej beh. Preto pre využitie takéhoto monitorovania je vhodné, aby aplikácia odoslala správu aspoň raz za určený čas. Táto správa môže byť potom prijatá monitorovacím skriptom, ktorý si poznačí, že aplikácia je funkčná. Pokiaľ sa mu niektorá z aplikácií neozve, môže ju reštartovať alebo upozorniť správcu systému.

Ďalšou možnosťou monitorovania chodu modulov je umiestnenie monitorovacej aplikácie do každého so zapojených modulov. Takáto aplikácia by odosielala pravidelne správu (je jedno či unicastovú alebo broadcastovú), ktorú by zase mohol spracovávať monitorovací skript. Takáto aplikácia beh systému nijako nevyťažuje, pretože je spustená len raz za niekoľko sekúnd (príklad implementácie 5.4).

```
class Monitoring(Task):
   async def loop(self):
   while self.running:
      super().send("STATUS","OK")
      await asyncio.sleep_ms(600)
```

Výpis 5.4: Príklad aplikácie odosielajúcej monitorovacie správy

Treťou možnosťou, ktorá nemonitoruje beh aplikácií, ale iba beh samotného systému je dotazovanie sa systému pravideľným zasielaním príkazu `runningtime`. Z tohoto príkazu je možné zistiť nielen či kontaktovaný uzol stále beží, ale aj to, či sa medzi kontaktovaniami nereštartoval. Tento spôsob monitorovania je použitý aj v skripte, ktorý pomáha nahrávať aplikácie do modulov (opísaný v časti 5.8). Každých 100 sekúnd skript skontroluje živosť každého z monitorovaných uzlov a v prípade, že nedostane odpoveď pokúsi sa modul reštartovať. Informáciu o reštarte zapíše aj do logu.

Pokiaľ by nastal v systéme neočakávaný stav (či už chybou v aplikácii alebo v implementácii systému), systém sa reštartuje. O tejto udalosti sa monitorovací skript môže dozvedieť prijatím broadcastovej správy `new_node`. Do takéhoto reštartovaného uzla je nutné znovu nahrať aplikácie, čo tiež môže obstarať monitorovací skript.

5.10 Ukladanie dát do databázy a ich zobrazovanie

Pre reálne nasadenie systému bude pravdepodobne nutné dáta zaznamenávať, ukladať do databáze a vedieť ich vhodne zobraziť. Pre predávanie dát inému systému bude zrejme nutné implementovať prevodník („bránu“), ktorý bude pracovať na pomedzi systémami a sprostredkovávať ich komunikáciu. Výstupy zo systému MPOS by mali byť dostatočné, keďže systém informuje o všetkých dôležitých udalostiach, ktoré v ňom môžu nastať (napríklad spustenia uzla alebo odpovede na príkazy). Možné odpovede na príkazy a správanie systému v rôznych stavoch sú popísané v sekcii pri popise komunikačného protokolu. Pomocou takejto brány je možné napojiť systém aj k hotovým riešeniam ako je Domoticz, OpenHab alebo Homeassistant. Ak by bolo možné v nejakom z týchto alebo im podobnom riešení presvedčiť systém na komunikáciu s protokolom popísaným v 4.6, nie je potrebné prevodník implementovať.

Kapitola 6

Záver

Cieľom práce bolo navrhnúť a realizovať systém s podobnými možnosťami rekonfigurácie ako poskytuje PNOS pre moduly `ESP8266` alebo `ESP32`, s využitím interpretovaného jazyka namiesto petriho sietí. Systém implementovaný v jazyku Micropython je možné napokon spustiť na oboch moduloch ESP. Modul `ESP32` umožňuje vďaka väčšej pamäti inštaláciu viacerých knižníc a beh väčšieho množstva aplikácií zároveň. Beh na module `ESP8266` je s určitými obmedzeniami tiež možný a stabilný. Typicky sa hodí na beh niekoľkých jednoduchých mikroaplikácií, ktoré merajú a odosielajú údaje z pripojených senzorov. Na problémy s nedostatočnou pamäťou na `ESP8266` som narazil až pri pokusoch s väčšími knižnicami, ako je napríklad knižnica na obsluhu displeja pripojeného cez rozhranie SPI.

Pri implementácii bol dodržaný protokol, ktorý využíva PNOS, vďaka čomu je možné dosiahnuť spoločného behu a spolupráce oboch systémov na jednej sieti. Pri implementácii aplikácie ju bude teda možné implementovať tým spôsobom, ktorý je pre ňu vhodnejší. Aplikácia spustená v Micropythone je schopná odosielať dáta aplikácii spustenej v PNOS a naopak.

Pridanie niektorých funkcií do systému znemožňuje neúplná implementácia Micropythonu na moduloch ESP (napríklad watchdog alebo OTA aktualizácie). Keďže má Micropython na moduloch ESP relatívne rozsiahlu komunitu, dá sa očakávať, že tieto časti systému budú s odstupom času doimplementované rovnako, ako už sú napríklad na module Pyboard.

Pre nasadenie systému do reálnych podmienok je vhodné systém doplniť o vyspelejší monitorovací systém, spôsob ukladania dát do databázy pre možnosti neskoršej analýzy a o dashboard. Na ukladanie a zobrazovanie dát by malo byť možné využiť niektoré z hotových riešení, ako je napríklad Home Assistant [11].

Literatura

[1] *Micropython*. Wikipedia, 2018, [Online; navštíveno 2.4.2018].
URL https://en.wikipedia.org/wiki/MicroPython

[2] DiCola, T.: *Building and running Micropython on ESP8266*. Adafruit, 2017, [Online; navštíveno 20.4.2018].
URL https://learn.adafruit.com/building-and-running-micropython-on-the-esp8266/build-firmware

[3] esp8266.com: *ESP8266 community wiki*. esp8266.com, 2018, [Online; navštíveno 10.4.2018].
URL https://www.esp8266.com/wiki/doku.php?id=esp8266-module-family

[4] Espressif: *ESP8266EX Technical Reference*. Espressif, 2017, [Online; navštíveno 10.4.2018].
URL https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf

[5] Espressif: *ESP32 Datasheet*. Espressif, 2018, [Online; navštíveno 11.4.2018].
URL https://www.espressif.com/en/products/hardware/esp32/resources

[6] Espressif: *ESP32 Overview*. Espressif, 2018, [Online; navštíveno 11.4.2018].
URL https://www.espressif.com/en/products/hardware/esp32/overview

[7] Espressif: *ESP32 Technical Reference*. Espressif, 2018, [Online; navštíveno 11.4.2018].
URL https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

[8] Espressif: *ESP8266EX Datasheet*. Espressif, 2018, [Online; navštíveno 10.4.2018].
URL https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

[9] Espressif: *ESP8266EX Overview*. Espressif, 2018, [Online; navštíveno 10.4.2018].
URL https://www.espressif.com/en/products/hardware/esp8266ex/overview

[10] Espruino: *Espruino Documentation*. Espruino, 2018, [Online; navštíveno 13.4.2018].
URL http://www.espruino.com/Quick+Start

[11] HomeAssistant: *HomeAssistant Documentation*. HASS, 2018, [Online; navštíveno 13.4.2018].
URL https://www.home-assistant.io/docs

[12] Itead: *Itead Overview*. Itead, 2018, [Online; navštíveno 29.4.2018].
URL https://www.itead.cc/sonoff-wifi-wireless-switch.html

[13] Koziorek, J.; aj.: Distribuované systémy řízení. VŠB-TU, 2011, ISBN 978-80-248-2599-1, str. 10.
URL http://www.person.vsb.cz/archivcd/FEI/DSR/Distribuovane%20systemy.pdf

[14] Micropython: *Micropython Documentation*. Micropython, 2018, [Online; navštíveno 10.2.2018].
URL http://docs.micropython.org

[15] Micropython: *Pyboard Documentation*. Micropython web, 2018, [Online; navštíveno 2.4.2018].
URL https://store.micropython.org/product/PYBv1.1

[16] NodeMcu: *NodeMCU Documentation*. NodeMcu, 2018, [Online; navštíveno 10.4.2018].
URL https://nodemcu.readthedocs.io/en/master/

[17] SQUIX78com: *ESP8266 modules comparison*. blog, 2015, [Online; navštíveno 10.4.2018].
URL https://blog.squix.org/2015/03/esp8266-module-comparison-esp-01-esp-05.html

[18] Tomáš Richta, V. J.: Operating System for Petri Nets-Specified Reconfigurable Embedded Systems. In *Computer Aided Systems Theory - EUROCAST 2013. EUROCAST 2013. Lecture Notes in Computer Science*, 2013, ISBN 978-3-642-53855-1, s. 444–451.
URL https://link.springer.com/chapter/10.1007%2F978-3-642-53856-8_56

[19] Wikipedia contributors: Internet of things — Wikipedia, The Free Encyclopedia. 2018, [Online; accessed 10-May-2018].
URL https://en.wikipedia.org/w/index.php?title=Internet_of_things&oldid=839979769

[20] Wikipedia contributors: SCADA — Wikipedia, The Free Encyclopedia. 2018, [Online; accessed 10-May-2018].
URL https://en.wikipedia.org/w/index.php?title=SCADA&oldid=840468422

[21] Wikipedie: Distribuovaný systém — Wikipedie: Otevřená encyklopedie. 2017, [Online; navštíveno 10. 05. 2018].
URL https://cs.wikipedia.org/w/index.php?title=Distribuovan%C3%BD_syst%C3%A9m&oldid=15069544

[22] Wikipedie: Internet věcí — Wikipedie: Otevřená encyklopedie. 2018, [Online; navštíveno 10. 05. 2018].
URL https://cs.wikipedia.org/w/index.php?title=Internet_v%C4%9Bc%C3%AD&oldid=16024848

Příloha A

Zdrojové kódy použité v demopříklade

```
#[Sensor, app_name, interval, sensor_type, pin,?pin2/correction ]
#[Sensor, app_name, interval, DS18B20, 0,-5 ]
class Sensor(Task):
   async def loop(self):
   self.correction=0
   print(len(self.params))
   if len(self.params)<5 or not self.params[2].isdigit():
      super().send("Error","Error in initialisation. Deactivating")
      super().stop()
   if self.params[3]=="DS18B20":
      import machine
      import onewire
      import ds18x20
      if len(self.params)>5:
         self.correction=float(self.params[5])
      dat = machine.Pin(int(self.params[4]))
      self.ds_sensor = ds18x20.DS18X20(onewire.OneWire(dat))
   elif self.params[3]=="DHT11":
      import machine
      import dht
      self.dht_sensor = dht.DHT11(machine.Pin(int(int(self.params[4]))))
   elif self.params[3]=="DHT22":
      import machine
      import dht
      self.dht_sensor = dht.DHT22(machine.Pin(int(self.params[4])))
      elif self.params[3]=="BMP180":
   import machine
      from bmp180 import BMP180
      self.bus = I2C(scl=machine.Pin(int(self.params[4])),
          sda=machine.Pin(int(self.params[5])))
      self.bmp180 = BMP180(self.bus)
      self.bmp180.oversample_sett = 2
      self.bmp180.baseline = 101325
   else:
      super().send("Error","Error in initialisation. Deactivating")
      super().stop()
```

```
  while self.running:
     try:
        if self.params[3]=="DS18B20":
           roms = self.ds_sensor.scan()
           self.ds_sensor.convert_temp()
           await asyncio.sleep_ms(750)
           for rom in roms:
           t=self.ds_sensor.read_temp(rom)
           t=float(t)+float(self.correction)
           super().send("temp",str(t))
        elif self.params[3][:3]=="DHT":
           self.dht_sensor.measure()
           super().send("temp",[str(self.dht_sensor.temperature())])
           super().send("hum",[str(self.dht_sensor.humidity())])
        elif self.params[3]=="BMP180":
           super().send("temp",[str(self.bmp180.temperature)])
           super().send("pressure",[str(self.bmp180.pressure)])
           super().send("altitude",[str(self.bmp180.altitude)])
     except Exception as ex:
        super().send("Error","Error reading")
        await asyncio.sleep(10)
```

Výpis A.1: Zdrojový kód aplikácie Sensor použitej v demopríklade.

```
#[Comparator, app_name, inverted]
class Comparator(Task):
   async def loop(self):
   interval=100
   if len(self.params)>2 and self.params[2].isdigit():
      self.inverted=int(self.params[2])
      self.output=0
      self.setPoint=0
      self.input=0
   while self.running:
      await asyncio.sleep(interval)

   def recieve(self,port, payload):
      if(port=="setpoint"):
         self.setPoint=float(payload)
      if(port=="input"):
         self.input=float(payload)
         self.actualise()

   def actualise(self):
      if float(self.input)<float(self.setPoint):
         self.output=1
      else:
         self.output=0
      super().send("output",str(self.output))
```

Výpis A.2: Zdrojový kód aplikácie Comparator použitej v demopríklade.

```
#[Relay, app_name, relay_pin, inverted]
```

```
class Relay(Task):
   async def loop(self):
      import machine
      pin=2
      interval=100
      self.inverted=0
      if len(self.params)>2 and self.params[2].isdigit():
         pin=int(self.params[2])
      if len(self.params)>3 and self.params[3].isdigit():
         self.inverted=bool(int(self.params[3]))
      self.relay=machine.Pin(pin,machine.Pin.OUT)
      self.state=0
      self.writeValue(self.state)
      while self.running:
         await asyncio.sleep(interval)

   def recieve(self,port, payload):
      if(port=="state"):
         if bool(int(payload))==True:
            self.state=1
         else:
            self.state=0
            self.writeValue(self.state)

   def writeValue(self,val):
      if self.inverted==True:
         self.relay.value(int(bool(not self.state)))
      else:
         self.relay.value(int(self.state))
      super().send("state",str(self.state))
```

Výpis A.3: Zdrojový kód aplikácie Relay použitej v demopríklade.

```
#[Or, app_name]
class Or(Task):
   async def loop(self):
      self.output=0
      interval=200
      self.inputs=dict()
      while self.running:
         await asyncio.sleep(interval)

   def calculateOutput(self):
      self.output=0
      for i in self.inputs.values():
         if int(i)>0:
            self.output=1
            break
      super().send("output",str(self.output))

   def recieve(self,port, payload):
      if str(payload).isdigit():
         self.inputs[port]=int(payload)
         self.calculateOutput()
```

Výpis A.4: Zdrojový kód aplikácie Or použitej v demopríklade.

```
#[Button, app_name, pin,debounce, inverted, interval]
class Button(Task):
   async def loop(self):
      import machine
      pin=0
      inverted=0
      debounce=40
      interval=60
      self.interruptOccurred=0
      if len(self.params)>2 and self.params[2].isdigit():
         pin=int(self.params[2])
      if len(self.params)>3 and self.params[3].isdigit():
         debounce=int(self.params[3])
      if len(self.params)>4 and self.params[4].isdigit():
         inverted=int(self.params[4])
      if len(self.params)>5 and self.params[5].isdigit():
         interval=int(self.params[5])
      if inverted==1:
         p = machine.Pin(pin, machine.Pin.IN, machine.Pin.PULL_UP)
         p.irq(trigger=machine.Pin.IRQ_FALLING, handler=self.process)
      else:
         p = machine.Pin(pin, machine.Pin.IN, machine.Pin.PULL_DOWN)
         p.irq(trigger=machine.Pin.IRQ_RISING, handler=self.callback)
      while self.running:
         if self.interruptOccurred>0:
            super().send("output","1")
            await asyncio.sleep_ms(debounce)
            self.interruptOccurred=0
         await asyncio.sleep_ms(interval)

   def callback(self,pin):
      self.interruptOccurred = self.interruptOccurred +1
```

Výpis A.5: Zdrojový kód aplikácie Button použitej v demopríklade.

```
#[Setpoint, app_name, default]
class Setpoint(Task):
   async def loop(self):
      interval=100
      self.setPoint=20.0
      if len(self.params)>2 and self.params[2].isdigit():
         self.setPoint=int(self.params[2])
      while self.running:
         await asyncio.sleep(interval)

   def recieve(self,port, payload):
      if(port=="plus"):
         self.setPoint+=0.5
         self.actualise()
      if(port=="minus"):
```

```
            self.setPoint-=0.5
            self.actualise()
        if(port=="setpoint"):
            self.setPoint=float(payload)
            self.actualise()

    def actualise(self):
        super().send("output",str(self.setPoint))
```

Výpis A.6: Zdrojový kód aplikácie Setpoint použitej v demopríklade.

```
class Display(Task):
    async def loop(self):
        import pcd8544
        from machine import Pin, SPI
        import framebuf
        spi=SPI(baudrate=8000000, mosi=Pin(19, Pin.OUT), sck=Pin(18, Pin.OUT),
            miso=Pin(15, Pin.IN))
        cs = Pin(22)
        dc = Pin(21)
        rst = Pin(23)
        self.bl = Pin(5, Pin.OUT, value=1)
        lcd = pcd8544.PCD8544(spi, cs, dc, rst)
        buffer = bytearray((lcd.height // 8) * lcd.width)
        framebuf = framebuf.FrameBuffer1(buffer, lcd.width, lcd.height)
        self.bl.value(0)
        self.temp1="-"
        self.temp2="-"
        self.status="?"
        self.target="-"
        while self.running:
            framebuf.fill(0)
            framebuf.text('M1:'+self.temp1+'C', 0, 0, 1)
            framebuf.text('M2:'+self.temp2+'C', 0, 10, 1)
            framebuf.text('CIL:'+self.target+'C', 0, 20, 1)
            framebuf.text('KOTOL:'+self.status, 0, 30, 1)
            lcd.position(0, 0)
            lcd.data(buffer)
            await asyncio.sleep\_ms(500)

    def recieve(self,port, payload):
        if(port=="room1"):
            self.temp1=payload[:payload.find(".")+2]
        if(port=="room2"):
            self.temp2=payload[:payload.find(".")+2]
        if(port=="stav\_kotel"):
            self.status=payload
        if(port=="cilova\_teplota"):
            self.target=str(payload)
```

Výpis A.7: Zdrojový kód aplikácie Display použitej v demopríklade.

Obsah priloženého média