VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

# BAKALÁŘSKÁ PRÁCE

Brno, 2020 Jan Vaverka


# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

## FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

## ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS


# VYTVOŘENÍ BLUETOOTH DATALOGGERU PRO ZÁZNAM A ZPRACOVÁNÍ NAMĚŘENÝCH DAT
CREATION OF BLUETOOTH DATA LOGGER FOR RECORDING AND PROCESSING OF MEASURED DATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS


AUTOR PRÁCE
AUTHOR
Jan Vaverka

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Petr Dejdar


BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

# Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

***Student:*** Jan Vaverka ***ID:*** 205963

***Ročník:*** 3 ***Akademický rok:*** 2019/20

**NÁZEV TÉMATU:**

## Vytvoření Bluetooth dataloggeru pro záznam a zpracování naměřených dat

**POKYNY PRO VYPRACOVÁNÍ:**

Cílem bakalářské práce je vytvoření zařízení s nízkou spotřebou, které bude využívat mikrokontroler ATmega pro kontinuální měření a ukládání dat. Tato data budou ukládána do paměti pro možné zobrazení na připojeném displeji a dále odesílána (na vyžádání) pomocí technologie Bluetooth do mobilního telefonu. Pro možnost zobrazení záznamů na mobilním telefonu bude vytvořena aplikace, která bude naměřená data přehledně zobrazovat v grafech. Bakalářská práce bude obsahovat kompletní funkční zařízení včetně aplikace pro mobilní telefony.

**DOPORUČENÁ LITERATURA:**

[1] TOWNSEND, Kevin, Robert DAVIDSON, AKIBA a Carles CUFÍ, 2014. Getting started with Bluetooth lowenergy: tools and techniquesforlow-power networking. RevisedFirstEdition. Sebastopol, CA: O'Reilly. ISBN 978-1491949511.

[2] MONK, Simon, 2016. Programming arduino: petting started with sketches. 2nd edition. New York, NY: McGraw-HillEducation. ISBN 978-125-9641-633.

***Termín zadání:*** 3.2.2020 ***Termín odevzdání:*** 8.6.2020

***Vedoucí práce:*** Ing. Petr Dejdar

**prof. Ing. Jiří Mišurec, CSc.**
předseda rady studijního programu

## ABSTRAKT

První část této práce se věnuje dataloggerům a jejich využití v dnešním světě. Ve druhé částí se práce zabývá mikrokontroléry, které se využívají jako řídící jednotky pro dataloggery. Byly popsány jednotlivé periferní zařízení MCU a rovněž jejich komunikační rozhraní. Další část je věnovaná Bluetooth Low Energy, které se využívá k přenosu naměřených dat k uživateli. Na základě získaných znalostí byl proveden výběr mikrokontroléru a senzorů. Z vybraných komponentů byl sestrojen a naprogramován datalogger, který bude měřit vlastnosti kvasu během kvašení ovoce. Na závěr byla vytvořená mobilní aplikace pro stahování dat z dataloggeru a následné zobrazení dat v grafech.

## KLÍČOVÁ SLOVA

Bluetooth Low Energy, datalogger, Flutter, mikrokontrolér, mobilní aplikace, senzor

## ABSTRACT

The first part of the theses deals with dataloggers and their use in today's world. The second part is dedicated to microcontrollers which are used as a control unit for dataloggers. Individual peripheral devices MCU and their communication interfaces have been described. The next part deals with Bluetooth Low Energy which is used for data broadcasting to the user. Based on the acquired knowledge the microcontroller and sensors were selected. A datalogger was constructed and programmed from selected components, which will measure the properties of ferment during fruit fermentation. Finally, a mobile application was created to download data from the datalogger and then display the data in charts.

## KEYWORDS

Bluetooth Low Energy, datalogger, Flutter, microcontroller, mobile application, sensor

VAVERKA, Jan. *Vytvoření Bluetooth dataloggeru pro záznam a zpracování naměřených dat*. Brno, 2020, 69 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Petr Dejdar

# PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Vytvoření Bluetooth dataloggeru pro záznam a zpracování naměřených dat“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ..... .....

podpis autora

# PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce panu Ing. Petru Dejdarovi, za účinnou metodickou, pedagogickou a odbornou pomoc při zpracování mé bakalářské práce.

# Obsah

# Seznam obrázků

# Úvod

Datalogger je malé zařízení, které umožňuje automatické měření a sběr dat v pravidelných intervalech. Tyto naměřené data jsou následně uložena na externí úložiště, nebo přenášena např. pomocí bezdrátové komunikace rovnou k uživateli. Mnohem podrobněji je datatlogger popsán v kapitole 1, kde jsou zmíněna jeho využití a další technologie související s dataloggery.

Jako řídící jednotka se u dataloggeru používají mikrokontroléry nejrůznějších druhů (Arduino, Raspberry Pi atd.). Kapitola 2 se ze začátku věnuje popisu různých architektur a instrukčních sad, které se dnes v procesorech a u mikrokontrolérů využívají. Převážná část kapitoly je pak věnována mikrokontrolérum AVR, konkrétně typu ATmega328P. V kapitole jsou popsány všechny základní periferní obvody a komunikační rozhraní, které ATmega328p obsahuje. Na konci kapitoly jsou vybrány a popsány různé druhy mikrokontrolérů, které lze využít jako řídící jednotku dataloggeru.

Protože je nutné aby datalogger komunikoval s okolním světem je potřeba využít některé z bezdrátové komunikační technologie. Tomuto problému se věnuje kapitola 3. Jelikož musí mít datalogger dlouhou výdrž na baterii, je nutné tomu přizpůsobit i komunikační technologii. Proto se tato kapitola podrobně věnuje technologie Bluetoot Low Energy, které je přesně k tomuto problému určeno. V kapitole je popsán způsob komunikace, jednotlivé kroky potřebné k navázaní spojení, i jednotlivé vrstvy protokolu.

Cílem bakalářské práce je vybrat senzory a řídící jednotku a z těchto komponentů sestavit funkční datalogger pro snímání veličin během procesu kvašení ovoce. Této problematice se věnují kapitoly 4 a 5. V poslední kapitole 6, je popsán vývoj aplikace pro mobilní zařízení, která bude stahovat data z dataloggeru a následně je zobrazovat.

# 1 Datalogger

Datalogger nebo také záznamník dat je elektronické zařízení, které umožňuje dlouhodobý sběr dat. Většinou se jedná o kompaktní mobilní zařízení, které obsahuje mikrokontrolér, vnitřní paměť pro ukládání naměřených dat, různé senzory a baterii pro napájení zařízení [1]. Jedna z největších výhod dataloggeru je schopnost nepřetržitého a automatického sběru dat a to i v extrémních podmínkách, které by mohly být pro člověka nebezpečné.

Datalogger používá buď vestavěné senzory, nebo umožňuje záznam vstupních signálů. Tyto signály můžou být digitální, nebo analogové které jsou před uložením převedené na diskrétní hodnoty. Naměřená data jsou následně ukládána do vnitřního úložiště nebo na přenosné medium např. paměťovou kartu. Datalogger muže také na žádost uživatele odesílat data do jeho mobilního telefonu, například s pomocí některé bezdrátové technologie např. Bluetooth, Wi-Fi [2]. Možností je také odeslat data přímo do databáze, kde mohou být například zpracována do grafů apod.

V dnešní době dochází k velkému rozmachu tzv. Internetu věcí (IoT- Internet of Things). Velkou výhodou je, že IoT umožňuje propojení všech zařízení do jednoho velkého celku. IoT přispívá k rozvoji průmyslu 4.0, kde umožňuje automatizaci jednoduchých pracovních úkonů. Velkou roli jak v IoT, tak v průmyslu 4.0 hrají právě dataloggery. Právě pro IoT vznikl nový typ sítí, tzv. low-power sítě (Sigfox, NB-IoT, LoRa). Tento typ sítí je vhodný pro přenos malých objemu dat a zařízení jsou tak schopná pracovat na baterii až několik let.

Některé dataloggery vyžadují pro svou činnost připojení k PC, který slouží jako úložiště pro data, nebo také prostředek pro jejich zpracování. Ovládání a programovaní zařízení je možné buď prostřednictvím ovládacích prvků přímo na zařízení, nebo prostřednictvím PC, či mobilního telefonu. Pro komunikaci se nejčastěji používá sériová sběrnice (USB), která je neoficiálním standardem pro mobilní zařízení. Lze také využít propojení přes Ethernet anebo sériový port RS-232. U průmyslových zařízení se často využívají speciální sběrnice např. průmyslový Ethernet, CAN, ProfiBus atd.

Jelikož se datalogger často nachází na odlehlých místech je nutné ho vybavit akumulátorem. Protože je od zařízení často požadováno, aby vydrželo v provozu v rozmezí několika měsíců až let, je nutné tomu přizpůsobit všechny aspekty zařízení a to jak na hardwarové, tak i na softwarové úrovni. Ze softwarového hlediska je nutné, aby byl datalogger co nejčastěji v režimu spánku a spouštěl se jen při nejnutnějších operacích, a to na velmi krátkou dobu. U hardwaru je nutné použit speciální součástky, které jsou přímo vyvinuty pro toto použití a mají nízkou spotřebu energie a vysokou efektivitu.

# 2 Mikrokontroléry

Mikrokontrolér (MCU) nebo také Jednočipový počítač je elektronická součástka nejčastěji v provedení integrovaného obvodu (IC). Mikrokontrolér lze programovat tak, aby plnil určité úlohy a řídil ostatní elektronické součástky. Mikrokontrolér obsahuje velké množství dalších obvodů, např. komparátor, pulzně šířková modulace (PWM), analogově-digitální nebo digitálně-analogový převodník, čítač/časovač, Synchronní/asynchronní sériové rozhraní (USART) atd.

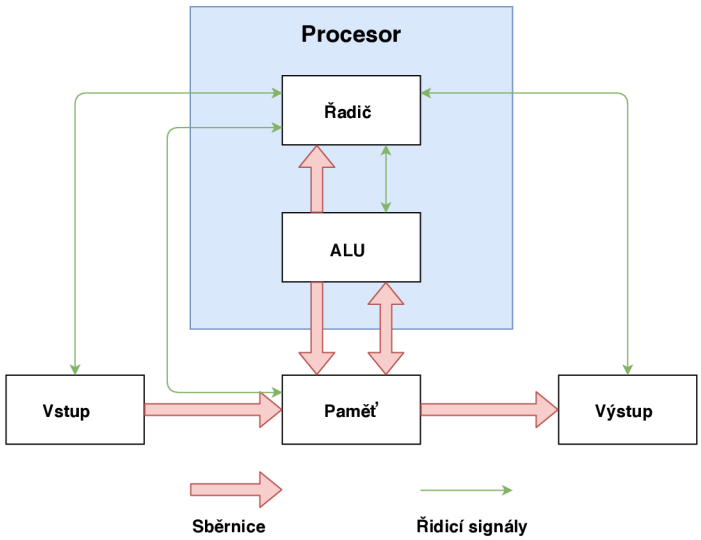
Existuje velké množství výrobců mikrokontrolérů, mezi nejrozšířenější patří Atmel se svojí rodinou mikrokontrolérů AVR (8 bitů), AVR32 (32 bitů) a firma ARM se svoji sérii Cortex-M (32 bitů).

S mikrokontroléry se dnes setkáváme v celé řadě aplikací: v automobilech, ve spotřební elektronice, součástech různých senzorů pro chytrou domácnost, nebo pro průmyslovou výrobu. Výhodou mikrokontrolérů je, že jsou velmi levné, lehce dostupné a oplývají velmi dobrým výpočetním výkonem.

## 2.1 Architektura mikropočítačů

### 2.1.1 Von Neumannova architektura

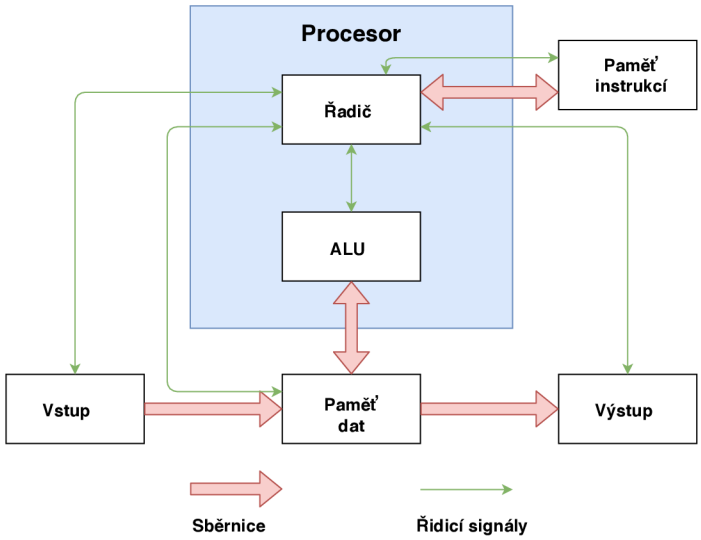
Skládá se z pěti bloků: paměti, řadiče, aritmeticko-logické jednotky (ALU) a vstupně-výstupních periferií. Tato koncepce popisuje počítač se společnou pamětí pro program i data. To znamená, že nelze pracovat s daty a programem najednou. Jedná se tak o sekvenční zpracovávání [3]. Celé blokové schéma je znázorněné na obr. 2.1.




Obr. 2.1: Blokové schéma Von Neumannovy architektury [4].

### 2.1.2 Harvardská architektura

Vznikla mírnou úpravou Von Neumannovy koncepce. Hlavní rozdíl je, že paměti programu a dat jsou fyzicky oddělené, jak jde vidět na obr. 2.2. To vede k tomu, že každá paměť muže být jiného typu a využívat jiného adresování. Výhodou je, že tato koncepce umožňuje paralelní přístup k oběma pamětím najednou, což podstatně zvyšuje rychlost zpracování. Lze tak číst nebo zapisovat do obou pamětí zároveň [5].




Obr. 2.2: Blokové schéma Harvardské architektury [6].

## 2.2 Typy instrukčních sad

### 2.2.1 CISC

Procesory typu Complex Instruction Set Computer (CISC) jsou charakterizovány velmi rozsáhlou sadou, mnohdy i složitých, strojových instrukcí (instrukce mají různou délku). Tyhle složité instrukce je možné naprogramovat pomocí jednodušších instrukcí. Na druhou stranu se tyto procesory vyznačuji relativně malým počtem registrů [7].

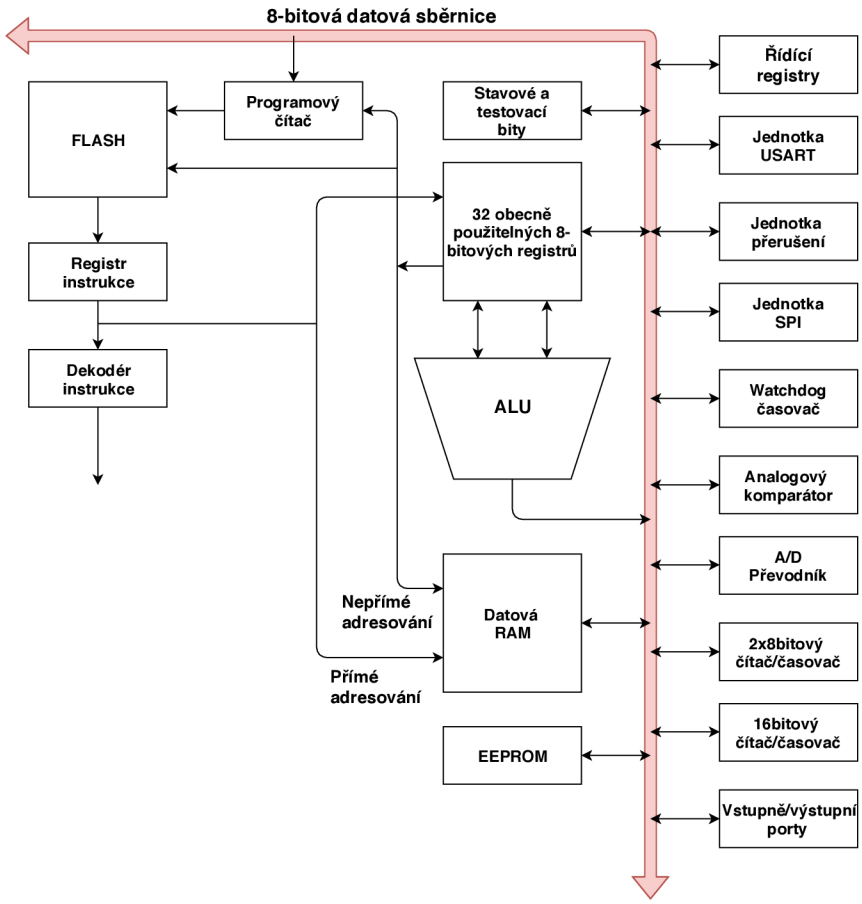
### 2.2.2 RISC

Bezpochyby jedna z výhod procesorů typy RISC (Reduced Instruction Set Computer) je malá instrukční sada. Naopak se vyznačují větším počtem registrů (oproti instrukční sadě CISC). Hlavním cílem bylo vytvořit vysoce optimalizovanou sadu strojových instrukcí. Je využíváno řetězení instrukcí tzv. pipelining. Další vlastností

je, že délka provádění jedné instrukce je většinou jeden hodinový cyklus a délka všech instrukcí je vždy stejná [8].

## 2.3 Mikrokontroléry AVR

AVR je rodina 8 nebo 32bitových mikrokontrolérů vyráběná firmou Atmel, která byla odkoupená v roce 2016 firmou Microchip Technology. Tyto mikrokontroléry využívají instrukční sadou RISC a jsou založený na upravené harvardské architektuře, tedy oddělené paměti programu a dat. Microchip v dnešní době vyrábí 3 základní řady: AVR Atiny, AVR ATmega a AVR ATXmega [9].




Obr. 2.3: Blokové schéma AVR [10].

Jádro AVR obsahuje 32 osmibitových registru, a toto jádro je propojené s aritmeticko-logickou jednotkou. K přístupu k těmto registrům stačí pouze jeden hodinový cyklus. Vnitřní propojení jádra AVR s periferními obvody je znázorněné na obr. 2.3. Posledních šest registrů lze také po dvojicích využít jako ukazatele pro nepřímé adresování do paměti [8].

**Adresovací módy AVR:**

- přímé adresování,
- nepřímé adresování,
- nepřímé adresovaní s 6bitovým posunem,
- nepřímé adresování s dekrementací ukazatele po zpravování instrukce,
- nepřímé adresování s dekrementací ukazatele před zpravováním instrukce.

## 2.4 Periferní obvody AVR

### 2.4.1 Čítač/časovač

ATmega328P obsahuje celkem 3 čítače/časovače, a to 2x8bitový (čítač/časovač 0 a čítač/časovač 2) a 1x16bitový (čítač/časovač 1) viz obr. 2.3. Čítače a časovače umožňují přesně časovat běh určité události, počítat impulsy z externího zdroje, generovat signál, dělit, měřit kmitočet nebo generovat signály PWM. Hlavní rozdíly mezi 8bitovým a 16bitovým čítačem/časovačem je že 16bitový čítač/časovač má k dispozici 4 nezávislé zdroje přerušení a muže měnit šířku PWM periody. Navíc obsahuje 16bitovou jednotku compare/capture, záchytný registr a dva komparační registry [11].

- Čítač – Obvod, který počítá impulsy vnějších signálů. Po dosažení určitého počtu impulsu se vykoná daná událost.
- Časovač – Obvod, který čítá pevný kmitočet, který udávají hodiny mikrokontroléru. Napočítáním určité hodnoty impulsů se určí časový úsek, který uběhl od posledního vynulování.

### 2.4.2 Watchdog

Jedná se o sledovací obvod (časovač), který hlídá správný běh programu. Zapíná se ihned po restartu MCU, nebo taktéž později z programu. Watchdog je svázán se zabudovaným RC oscilátorem, u kterého čítá jednotlivé impulsy [12].

Pokud dojde k jeho přetečení vyvolá obvod automatický restart MCU. Proto se musí watchdog nulovat speciální instrukcí. Reset MCU je pojistka proti tomu, aby program nezabloudil nebo neuvízl např. v nekonečné smyčce, ve které nedochází k nulování obvodu watchdog. Dobu, za kterou dojde k přetečení watchdogu lze ovládat předděličkou (na výběr je 8 různých intervalů) [11].

### 2.4.3 Reset a přerušení

Reset slouží k tomu, aby se MCU vrátil zpět na počáteční adresu v programové paměti. Mikrokontrolér ATmega328P má 4 zdroje resetu (reset po připojení napá-

jení, reset po poklesu napájení, vnější reset a watchdog reset). Po vykonaní jednoho z těchto resetů se u všech vstupně/výstupních portů nastaví jejich výchozí hodnota a program začne běžet od počáteční adresy [11].

Naproti tomu, přerušení funguje tak, že je běžící program přerušen a začne vykonávat jinou část programu (obsluhu přerušeni). Po dokončení této obsluhy přerušení se program vrátí zpět na místo, kde byl přerušen a pokračuje ve své činnosti. ATmega328P má celkem 26 vektorů přerušení. Každé přerušení má specifickou adresu, kterou nelze měnit. Nejnižší adresa (reset) má nejvyšší prioritu, naopak nejvyšší adresa přerušení má nejnižší prioritu. Přerušení může vyvolat jakýkoliv periferní obvod MCU (čítač/časovač, A/D převodník, USART, SPI atd.) [13].

### 2.4.4 A/D převodník

A/D (Analogově/Digitální) převodník je obvod mikrokontrolérů, který slouží k převodu spojitých signálu na signál diskrétní. Mikrokontroléry ATmega328P obsahují vnitřní 10bitový A/D převodník pracující s algoritmem postupné aproximace. U tohoto převodníku lze snímat až 8 vstupů, jelikož je připojený na analogový multiplexer. Doba jednoho převodu trvá přibližně 65 až 260 µs (15 000 vzorků za sekundu). A/D převodník má oddělené napájení a rovněž využívá analogovou zem. Zabudované referenční napětí je 1,22 V (nebo 2,56 V). Když převodník pracuje v režimu Idle, lze také využít potlačení šumu [11].

### 2.4.5 Analogový komparátor

Analogový komparátor porovnává dvě různá vstupní napětí vůči sobě. Využívají se k tomu dva vstupy a to invertující (záporný) a neinvertující (kladný). Pokud je ($U+ \leq U-$) je na výstup komparátoru nastavena logická 0. A jestli platí (U+ > U-) je výstupu nastaven na logickou 1. Na výstup komparátoru lze generovat speciální vektor přerušení. Analogový komparátor může využívat multiplexer od A/D převodníku za podmínky, že je převodník vypnutý [14].

### 2.4.6 Vstupně/výstupní porty

Veškeré vstupně/výstupní porty ATmega328P mají tzv. skutečnou funkcionalitu (Read, Modify, Write) za předpokladu, že se používají jako obecně číslované vstupy/výstupy. ATmega328P má celkově čtyři 8bitové porty (port A, port B, port C a port D). Pro práci s každým portem jsou k dispozici tři řídící registry (DDRx, PORTx, PINx) písmeno x zastupuje jméno portu v tomhle případě pro port [15].

- DDRx (Data Direction Register) – Tento registr definuje směr toku dat. Určuje, jestli se bude port chovat jako vstup nebo výstup.

- PORTx – Jedná se o datový registr, do něhož se zapisuji výstupní data z vyrovnávacího registru portu. Jednoduše řečeno se jedná o výstupní registr.
- PINx (Pins Input) – Tento registr umožňuje přístup k fyzické hodnotě vlastního portu a je určen pouze pro čtení. Jednoduše se tak jedná o vstupní registr.

### 2.4.7 EEPROM

Electrically Erasable Programmable Read-Only Memory (EEPROM) je nevolatilní paměť typu ROM-RAM (Read-Only Memory, Random-Access memory). Nevolatilní typ pamětí znamená, že data zůstávají v paměti uložena i po odpojení napájecího napětí. EEPROM má životnost přibližně 100 000 zapisovacích/mazacích cyklů. Kapacita paměti činí u ATmegy328P 1 KB [14].

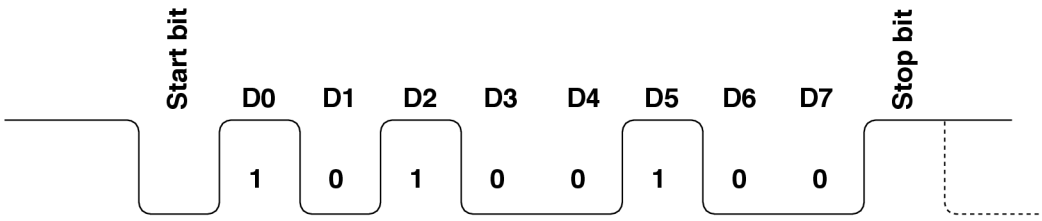
## 2.5 Sériová komunikační rozhraní

U mikrokontrolérů se sériová komunikační rozhraní používají ke dvěma účelům. První je ke komunikaci mezi jednotlivými integrovanými obvody. Druhým účelem je komunikace mezi dvěma mikrokontroléry.

### 2.5.1 USART

Jednotka USART u MCU AVR zpřístupňuje celou řadu nejpoužívanějších komunikačních protokolu např. RS232, RS485 atd. Pro přenos se používají rámce, které mohou mít 5 až 9 datových bitů. Každý rámec začíná tzv. start bitem a končí jedním nebo dvěma stop bity, jak jde vidět na obr. 2.4 na kterém je znázorněn přenos 8bitového rámce. Do rámce se muže ještě vložit tzv. paritní bit, který slouží ke kontrole rámce. USART podporuje dva módy přenosu a to synchronní (half-duplex) a více využívaný asynchronní (full-duplex). Mimo jiné také USART obsahuje filtraci šumu, detekce falešného start bitu, přetečení datového registru a detekci chybného znaku [12].

U asynchronní přenos USART se využívá převážně dvou pinů a to transmitter (TX) a receiver (RX). Modulační rychlost komunikace se uvádí v baudech za sekundu (Bd/s). Aby byla možná komunikace mezi dvěma zařízeními musí být na obou zařízeních nastavená stejná modulační rychlost a piny TX a RX musí být zapojené do kříže. K řízení se používá řídící registr UCR, registr přenosových rychlostí UBRR, datový registr UDR a stavový registr USR.




Obr. 2.4: USART asynchronní přenos.

### 2.5.2 SPI

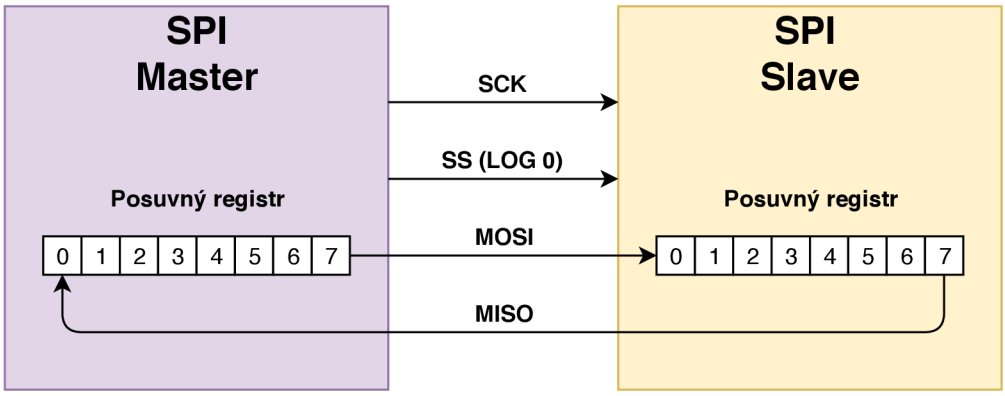
Serial Peripheral Interface (SPI) rozhraní je především určeno ke komunikaci vnitřních integrovaných obvodu MCU. U řady mikrokontrolérů se také SPI využívá k programování jejich vnitřní paměti Flash. Taktéž se toto rozhraní využívá ke komunikaci mezi jednotlivými mikrokontroléry [16].

SPI je full duplex komunikace typu Master-Slave. V tomhle typu komunikace je vždy jeden obvod v režimu Master (obvykle procesor) a další obvody v režimu Slave. Oba obvody obsahují posuvné registry. Pokud jsou propojeny dvě zařízení je potřeba 4 vodiče (SCK, SS, MISO a MOSI) [16].

- SCK (Serial Clock) – Přenáší hodinové impulsy.
- SS (Slave Select) – Slouží k výběru obvodu typu Slave, se kterým bude zařízení typu Master komunikovat.
- MISO (Master In Slave Out) – Jedná se o datový vstup zařízení typu Master, který je propojený se všemi porty MISO na zařízeních typu Slave.
- MOSI (Master Out Slave In) – Jedná se o datový výstup zařízení typu Master, který je propojený se všemi porty MOSI na zařízeních typu Slave.

Komunikace funguje tak, že Master vyšle signál logickou 0 na port SS obvodu s kterým chce komunikovat. Na všech ostatních obvodech typu Slave musí být nastavená logická 1, jelikož komunikace probíhá vždy mezi jedním obvodem typu Master a jedním obvodem typu Slave. Následně Master začne generovat hodinový signál (maximálně 2 MHz) a zařízení začnou vysílat svoje data. Hodinový signál slouží k řízení posuvu obou posuvných registrů. Délka rámce většinou bývá 8 nebo 16 bitů [17]. Na obrázku 2.5 je znázorněná komunikace mezi Masterem a Slave zařízením.

Kromě nastavení frekvence hodin musí Master také nakonfigurovat polaritu hodin a fázi s ohledem na data. K tomu slouží dva konfigurační bity CPOL a CPHA. CPOL určuje polaritu hodin, zatímco CPHA určuje načasování (Fázi) datových bitů vzhledem k hodinovým impulsům.




Obr. 2.5: Komunikace přes SPI [18].

### 2.5.3 I²C

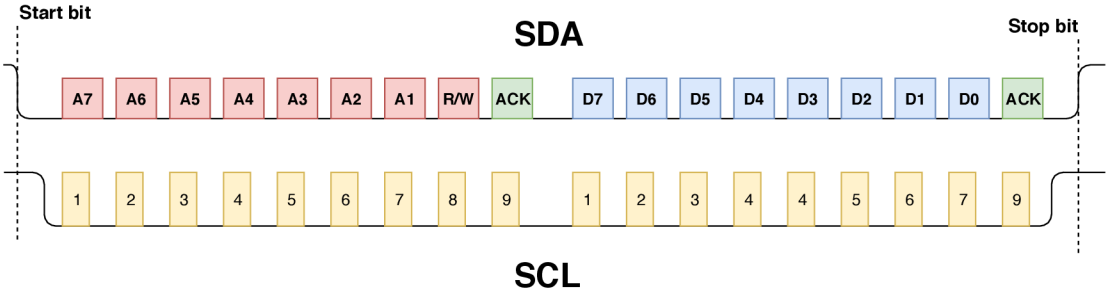
Inter-Integrated Circuit (I²C) je sériová sběrnice vyvinutá firmou Philips. Na rozdíl od SPI sběrnice je I²C typu multimaster což znamená, že obvodu typu Master muže být více jak jeden. Každý obvod má svojí vlastní adresu o velikosti 7 nebo 10 bitů. Tato adresa slouží k výběru zařízení. Jelikož lze mít v této koncepci více zařízení typu Master musí se zajistit, aby dvě nebo více zařízení nemohla vysílat současně. K tomu slouží tzv. arbitráž [16].

U I²C je zapotřebí pouze dvou obousměrných vodičů (maximálně až 128 různých zařízení), datového (SDA) a hodinového (SCL). Každý z těchto vodičů musí být připojený přes pull-up rezistor ke kladnému napětí. To má za následek, že v klidovém stavu (volná sběrnice) je na vodičích logická 1. Maximální délka vodiče je určená jejich nejvyšší povolenou kapacitou (400 pF). Co se týče maximální frekvence hodinového signálu SCL ta je dána podle verze I²C (100 kHz nebo 400 kHz). Pro obě tyto frekvence je dána minimální doba, kterou musí signál SCL setrvat v úrovni logická 0 a logická 1 [17].

- Synchronous Data (SDA) – Při komunikaci, když je na vodiči SCL logická 1, jsou po tomto vodiči synchronně přenášeny datové bity. Data mohou vysílat oba typy zařízení jak Master, tak i Slave.
- Synchronous Clock (SCL) – Slouží k přenášení hodinového signálu a může ho vysílat pouze zařízení typu Master.

Každá stanice může zahájit vysílaní, je-li předtím sběrnice v klidovém stavu. Před každým přenosem je nutné vyslat tzv. start bit, který je následován 7bitovou adresou příjemce, se kterým chce Master komunikovat. Poté je vyslán speciální bit R/W kterým se určí typ operace (čtení, zápis) kterou chce Master provést. Následně Slave, který zjistí, že se jeho adresa shoduje s přijatou adresou vyšle bit ACK, potvrzení o připravenosti na samotný přenos dat. Poté jsou již přenášená samotná

data, a to vždy když je signál SCL v logické 1. Každý bajt je následován jedním kontrolním bitem ACK. Po přenosu všech dat je přenos ukončen tzv. stop bitem [19]. Celý průběh komunikace pomoci 7bitové adresy je znázorněn na obr. 2.6.




Obr. 2.6: Průběh komunikace přes $I^2C$ [20].

### 2.5.4 TWI

Z důvodu patentu firmy Philips na technologii $I^2C$ používají některé firmy (Microchip Technology) označení Two Wire Interface (TWI). Jedná se o identickou sběrnici.

### 2.5.5 1-Wire

Jedná se o speciální sběrnici typu Master-Slave vyvinutou firmou Dallas Semiconductor. Koncepčně je podobná $I^2C$ sběrnici s tím rozdílem, že nepodporuje tak vysoké přenosové rychlosti. Sběrnice pro komunikaci potřebuje pouze jeden vodič. Datový vodič musí být přes pull-up rezistor 4,7 kΩ připojen ke kladnému napětí.

Komunikaci vždy zahajuje Master. Data jsou posílaná v oknech o délce 60 $\mu$s až 120 $\mu$s a během jednoho okna se přenese 1 bit. Po každém okně musí zůstat sběrnice v klidu a to minimálně na dobu 1 $\mu$s [21].

## 2.6 Arduino

Arduino je platforma, která vznikla v Itálii v roce 2005. Od začátku byla založena jako open source, což zapříčinilo velký pokles cen a nárůst obliby těchto desek. Základní deska Arduina se dá rozšířit o tzv. Shieldy, které Arduinu přidávají další funkce. Další velkou výhodou je, že software k programování Arduina lze používat na všech operačních systémech (Windows, MacOS, Linux) [22].

Každé Arduino obsahuje mikrokontrolér ATmega od firmy Atmel. Jednotlivé desky se liší jak t jeho typem, tak i periferiemi. Nejznámější deskou je Arduino Uno obsahující ATmegu328.

## 2.7 Firebeetle

DFRobot FireBeetle je MCU založený na platformě Arduino s nízkou spotřebou určený pro vývoj a aplikaci v oblasti IoT. Kombinuje mikrokontrolér Arduino Pro Mini s Bluetooth 4.1 low energy [23]. Jelikož je založen na platformě Arduino, lze využít veškeré nástroje a knihovny, které byly pro Arduino vytvořené.

**Specifikace:**

- Procesor: ATmega328P
- Napětí VCC I/O: 3.7 V  5.5 V
- Digitální porty: 14
- Analogové porty: 6
- Maximální přenosová rychlost: 4 Kbajtů/s
- SPI
- $I^2C$
- UART
- Rozměry: 58x29 mm

## 2.8 Raspberry Pi

Jedná se o jednodeskový počítač o velikostí přibližně platební karty. Obsahuje všechny základní porty pro připojení periferii (USB, ethernet, 3,5 mm jack konektor, GPIO). Pro napájení se nejčastěji používá MicroUSB (USB C), lze také použít GPIO piny nebo napájení přes ethernet pomoci PoE (Power over Ethernet). Dále podporuje nejužívanější bezdrátové standardy jako Wi-Fi a Bluetooth. Verze jednotlivých portů a bezdrátových standardu se liší s ohledem na verzi počítače. Nejnovějším modelem je v současné chvíli Raspberry Pi 4 [24].

Celá platforma Raspberry Pi je postavená na architektuře ARM. Na zařízení lze spustit několik operačních systému jako Raspbian nebo Linuxové distribuce založené na Debianu nebo Ubuntu. K programování se nejčastěji využívá jazyk Python, ale je možné využít i řadu dalších.

## 2.9 ESP32

MCU ESP32 je nástupcem velmi úspěšného MCU ESP8266 od firmy Tensilica, který si získal u komunity velkou přízeň. ESP32 přichází s řadou vylepšení jako je přidání BLE, dvou jádrovým procesorem a lepším úsporným režimem [25]. Všechny tyto funkce dělají z ESP32 skvělý MCU pro implementací v IoT segmentu.

**Specifikace:**

- Procesor: Xtensa LX6
- Napětí: 3.3 V
- Wi-Fi: 802.11 b/g/n
- Bluetooth: 4.2 a nižší
- GPIO: 36
- SPI: 4
- $I^2C$: 2
- UART: 2
- Rozměry: 58x25,4 mm

# 3 Komunikační protokoly

Komunikační protokoly jsou množina pravidel, která umožňuji dvěma nebo více entitám přenášet informace prostřednictvím přenosového média. Komunikační protokoly určují správný formát zprávy, velikost rámců, jejich potvrzování, rychlost komunikace atd. Výhodou je, že komunikační protokoly nejsou závislé na implementačním řešení. Proto lze propojit zařízení, která mají různou architekturu nebo odlišný operační systém. Protokoly lze realizovat jak hardwarově, tak i softwarově nebo jako kombinací obou těchto metod. Jednotlivé komunikační systémy obvykle nepoužívají jediný protokol ke zpracování přenosu. Místo toho používají sadu spolupracujících protokolů, které se navzájem doplňují.

Komunikační protokoly toho zajišťuji opravdu hodně a dělí se tedy do několika vrstev. Každá z vrstev poskytuje své služby vrstvě nadřazené. Proto má každá vrstva na starost jenom několik určitých úkolů. U vrstvového modelu lze jednotlivé vrstvy implementovat nezávisle na sobě a pro každou vrstvu muže existovat několik různých implementací. Tyto implementace se mohou lišit např. při použití různých přenosových medií [26].

## 3.1 Bluetooth

Bluetooth je otevřený standard pro bezdrátovou komunikaci na kratší vzdálenosti. Tato technologie je definována standardem IEEE 802.15.1, který spadá do kategorie osobních počítačových síti tzv. PAN (Personal Area network). Tuto technologii zaštiťuje organizace Bluetooth SIG (Bluetooth Special Interest Group). Tato organizace se stará o vývoj nových verzí této technologie.

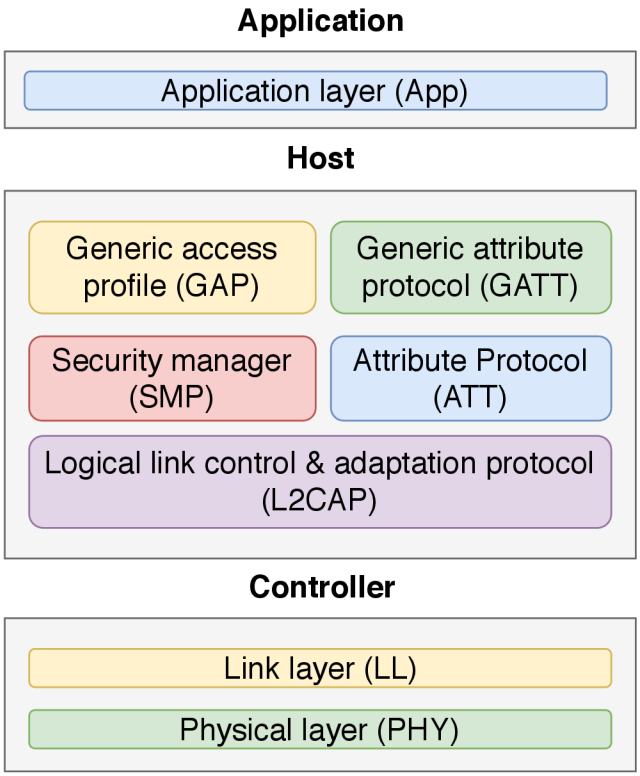
Bluetooth technologie prošla v průběhu času velkými změnami. Jedna z největších novinek je Bluetooth Low Energy (BLE) kterou přinesl standard Bluetooth verze 4.0 v roce 2010. Hlavní výhodu, kterou BLE přineslo, je nízkoenergetická náročnost pro zařízeni, která mají omezený výkon a nevyužívají vysoké datové toky. V roce 2016 byl vydán standard Bluetooth verze 5.0. Jednou z největších novinek bylo zvětšení dosahu, a to až 4x oproti předchozí verzi, rovněž došlo ke zlepšení účinnosti a výkonu. Zatím poslední verze 5.1 byla vydána v roce 2019. Tato poslední verze umožňuje lokalizaci vysílajícího zařízení z přesnosti na desítky až jednotky centimetrů [27] [28].

Bluetooth je velmi rozšířená technologie, kterou najdeme téměř v každém moderním zařízením. Po celém světě jsou aktivní miliardy zařízení s touto technologií. Každý člověk u sebe každý den nosí několik zařízení s Bluetooth (mobilní telefon, notebook, sluchátka, nositelná elektronika – smart wereables). S příchodem verze 4.0

a BLE se tato technologie začala rozšiřovat i do oblasti IoT a smart home zařízení (chytré žárovky, zámky dveří, chytré senzory atd.).

## 3.2 Bluetooth low energy

Jako každý moderní protokol tak i BLE je rozdělený do několika vrstev, jak znázorňuje obr. 3.1. Protokol je tvořený třemi základními vrstvami, každá obsahuje několik dalších podvrstev a protokolů. Jednotlivé protokoly jsou rozdělené do tří sekcí. Sekce Controller reprezentuje samotný integrovaný obvod Bluetooth. Sekce Host reprezentuje zařízení, které využívá Bluetooth (PC, mobil, datalogger). Poslední sekcí je Application, jedná se o aplikaci nebo program, který běží na zařízení, které využívá Bluetooth technologii.

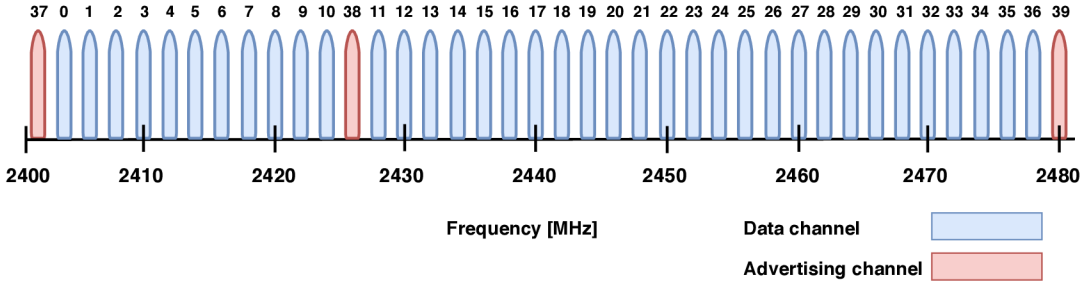



Obr. 3.1: Architektura BLE protokolu [30].

Zařízení s BLE lze realizovat s dosahem i 30 m nebo více (100 m při přímé viditelnosti). Standardní rozsah u BLE je okolo 3-5 m. Tento rozsah poskytuje jakýsi kompromis mezi spotřebou energie a spolehlivostí přenosu. BLE umožňuje dva druhy spojení: broadcast a unicast [29].

### 3.2.1 Fyzická vrstva

BLE využívá frekvenci 2,4 GHz z ISM (Industrial Scientific Medical) pásma. Toto pásmo je rozdělené na 40 menších kanálů od 2,400 GHz do 2,4835 GHz mezi každým kanálem je 2 MHz mezera viz obr. 3.2. 37 kanálů je vyhrazeno pro data a zbylé 3, konkrétně 37, 38 a 39, jsou určeny pro navázaní komunikace nebo broadcast [31].




Obr. 3.2: Pásmo a kanály BLE [31].

Komunikace neprobíhá pouze na jednom kanálu, ale využívá metody FHSS (frequency hopping spread spectrum). Jak už název napovídá jedná se o to, že komunikace postupně skáče vpřed po jednotlivých kanálech. Tato metoda využívá k vypočtu dalšího kanálu vzorec 3.1.

$$f_{n+1} = (f_n + hop) mod 37 \tag{3.1}$$

Tato metoda se využívá hlavně proto to, aby minimalizovala rušení a přeslechy během komunikace. V dnešní době je pásmo 2,4 GHz velmi zarušené, jelikož toto pásmo rovněž využívají Wi-Fi sítě a různá další zařízení [29].

### 3.2.2 Linková vrstva

Tato vrstva je přímo propojená s fyzickou vrstvou a stará se o nalezení, vytvoření a udržení spojení mezi zařízeními. BLE ve spolupráci s linkovou vrstvou umožňuje dva typy spojení a to unicast (peer-peer) a broadcast.

#### Unicast

Unicast komunikace se skládá ze dvou subjektů, Master a Slave. Tato komunikace se používá v případě, kdy je nutné data posílat v obou směrech. Jedná se o pravidelnou výměnu dat mezi dvěma subjekty. Jedná se o soukromou (zabezpečenou) komunikaci [29].

**S verzí Bluetooth 4.1 byly přidány speciální funkce:**

- Zařízení může být jak Master, tak i Slave ve stejnou chvíli.
- Master muže být propojen s více zařízeními typu Slave.
- Může být propojen s více Mastery.

### Broadcast

Broadcast komunikace se skládá ze dvou subjektů, vysílače a přijímače. U tohoto typu komunikace jsou data vysílaná všesměrově a jakékoliv jiné zařízení je může přijmout. Výhodou je, že se jedná o velmi jednoduchou komunikaci. Na druhou stranu se však nejedná o bezpečný typ komunikace, proto není vhodný pro citlivá a soukromá data. Rovněž nelze data posílat obousměrně, jedná se tedy pouze o směr od vysílače k přijímači [29].

### Adresa

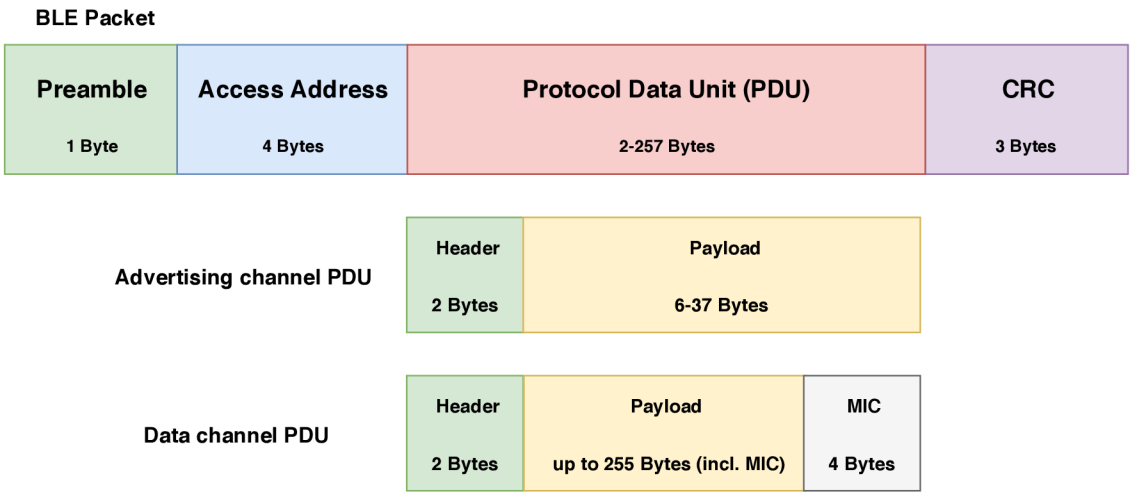
Každé BLE zařízení má svojí jedinečnou identifikační adresu, která je mu přiřazena při výrobě. Tato adresa má velikost 48 bitů (6 bajtů) a bývá nejčastěji zapisována v hexadecimálním formátu. Adresa zařízení může být dvojího typu, veřejná a náhodná [29].

- Veřejná adresa – jedná se o standardní MAC adresu zařízení, která musí být registrována u organizace IEEE a nikdy se nesmí změnit.
- Náhodná adresa – Tato adresa může být na zařízení přeprogramována nebo dynamicky generována při provozu. Náhodná adresa se ještě dále dělí na tři typy: statická adresa, rozpoznatelná soukromá adresa a nerozpoznatelná soukromá adresa.

### Paket

U BLE se využívá jeden formát paketu, který se skládá ze 4 částí viz obr. 3.3. Část PDU se mění v závislosti na tom, zda se jedná o přenos pomocí datového nebo advertising kanálu [32].

- Preamble – Tato část má vždy velikost 1 bajt a je používána přijímačem pro synchronizaci času a frekvence.
- Access Address - Pro všechny advertising pakety se používá pevná adresa 0x8E89BED6. Pro datové pakety je vygenerována nová náhodná adresa o velikosti 32 bitů.
- PDU – Jedná se o část paketu, kde jsou uložené data. V závislosti na kanálu, přes který se paket přenáší se mění i tato část. U datového PDU slouží část MIC (Message Integrity Check) k zabezpečení, tato část není povinná.
- CRC – Kontrolní součet se počítá z PDU a tvoří poslední 3 bajty.




Obr. 3.3: BLE paket [32].

### Připojení

Prvním krokem v procesu navázání komunikace je nalezení druhého zařízení (přijímače). K tomu jsou určeny dva procesy, advertising a scanning. Když o sobě obě zařízení vědí dostatek informací nastává čas pro poslední proces (connection), kdy dojde ke konečnému spojení obou zařízení. Všechny tři procesy jsou popsány níže [33].

### 1) Proces advertising

Tento proces používá zařízení Slave k tomu, aby dal zařízení Master vědět, že je připravený ke komunikaci. Slave vysílá v pravidelných intervalech advertising pakety. Tyto pakety mohou být odesílány v rozmezí od 20 ms až do 10,24 s. Celkově rozlišujeme 4 základní typy advertising paketů:

- General (ADV_IND) – Tímto paketem dává zařízení najevo, že se chce připojit. Je posílán přes advertising kanál všem zařízením, které na tomto kanále jsou. General paket se využívá nejčastěji ze všech zmíněných paketů.
- Directed (ADV_DIRECT_IND) – Tento paket je žádostí o připojení, ale je poslán pouze konkrétnímu zařízení.
- Nonconnectable (ADV_NONCONN_IND) – Tento paket vysílají pouze vysílače, které tímto informují, že se jedná o broadcast vysílaní.
- Discoverable (ADV_SCAN_IND) – Tímto paketem zařízení říká, že se k němu nelze připojit, ale lze ho pouze oskenovat.

### 2) Proces scanning

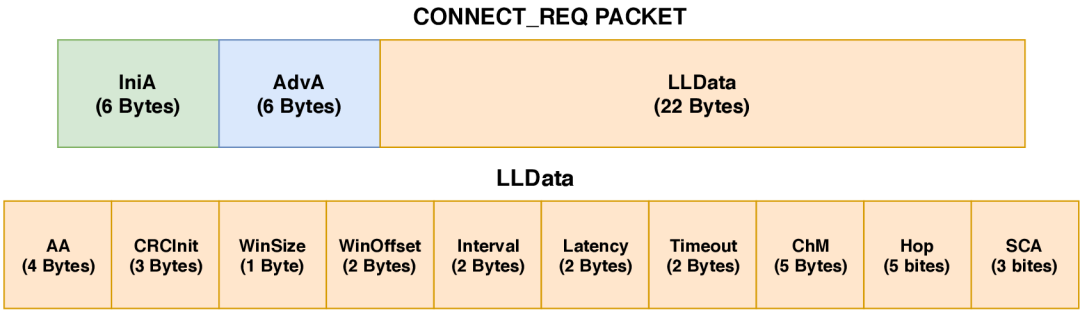
Skenování využívá zařízení Master k tomu, aby objevil zařízení typu Slave, která vysílají advertising pakety. Master může využít dvojího typu skenovaní, pasivní a aktivní.

- Pasivní skenování – Master pouze naslouchá na advertising kanálu a čeká na ADV_IND pakety, které vysílají Slave. Tento typ skenování je velmi jednoduchý a Master se od Slave dozví pouze základní informace.
- Aktivní skenování – Tento typ skenování využívá zařízení Master tehdy, když chce o Slave získat více informací jako např. jméno zařízení. Master opět naslouchá na advertising kanálu a čeká na ADV_IND paket. Po zachycení tohoto paketu si Master vyžádá speciálním paketem SCAN_REQ další informace. Slave odesílá odpověď v paketu SCAN_RSP.

### 3) Proces connection

Jakmile má Master MAC adresu a další informace, podle kterých se rozhodne, ke kterému Slavovi se připojí, zahájí proces připojení. Master posílá paket CONNECT_REQ, který se skládá ze tří částí a má celkovou velikost 34 bajtů.

- InitA – Adresa Master zařízení, které inicializuje připojení.
- AdvA – Adresa Slave zařízení.
- LLData – Dodatečné informace potřebné k navázaní komunikace.



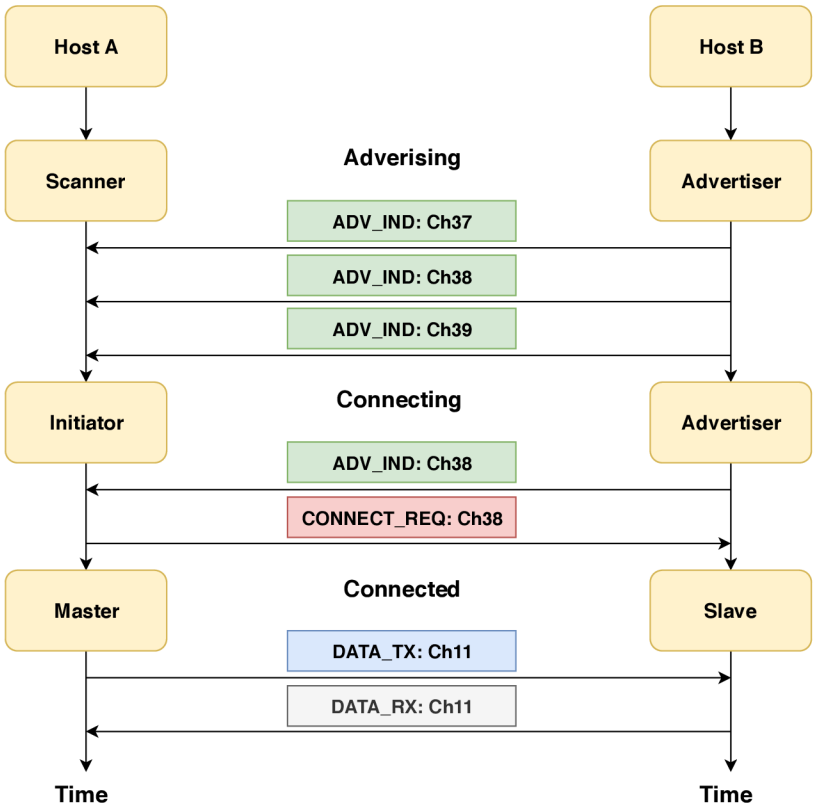

Obr. 3.4: CONNECT_REQ paket [34].

Část LLData je dále rozdělená na 10 menších částí. Pro navázaní komunikace jsou nejdůležitější:

- Interval (Connection interval) – Čas mezi začátkem dvou po sobě jdoucích událostí. Tato hodnota může být v rozsahu od 7,5 ms (vysoký výkon / velká spotřeba) až do 4 s (malý výkon / malá spotřeba).

- Timeout (Supervision timeout) – Maximální doba mezi dvěma platně přijatými datovými pakety. Po překročení tohoto času je spojení mezi zařízeními považováno za ztracené.
- (Frequency hopping) – Jedná se o hodnotu hop, která se využívá u FHSS ve vzorci 3.1.

Celý proces spojení a komunikace je znázorněn na obr. 3.5. Na obrázku jde vidět, jak jednotlivá zařízení prochází jednotlivými stavy během navázaní komunikace. Jedná se o situaci, kdy Master (Scanner) využívá pasivní skenování a pouze naslouchá na advertising kanálech (kanály 37, 38 a 39). Zatímco Slave (Advertiser) na těchto kanálech rozesílá ADV_IND pakety. Jakmile Scanner zachytí advertising paket přepne se do stavu Initiator a odesílá CONNECT_REQ paket. Jakmile je paket CONNECT_REQ doručen na obě zařízení, přepínají se do posledního stavu (Master/Slave) a přesouvají se na předem domluvený datový kanál, kde začne výměna dat. Na CONNECT_REQ paket Advetiser nijak neodpovídá, ani ho nijak nepotvrzuje. Pokud Master nenajde na datovém kanálu Slave zařízení, jednoduše se celý proces spojení zopakuje.




Obr. 3.5: Spojení a komunikace zařízení pomoci BLE [35].

### 3.2.3 Komunikace

Po úspěšném spojení si zařízení začnou vyměňovat data. Využívají k tomu jeden z 37 datových kanálů. Komunikace probíhá v pravidelných intervalech, na kterém se zařízení domluvila při procesu Connection pomocí parametru Connection interval viz 3.2.2. Komunikace probíhá neustále i v případě, kdy zařízení nemají data k výměně. V takovém případě má PDU velikost 0 bitů. Jedná se o funkci, která slouží k tomu, aby nedošlo k odpojení zařízení.

BLE poskytuje spolehlivé doručení každého paketu. Každý paket obsahuje CRC parametr, který kontroluje, zda není paket nějak poškozený. V případě, že paket nedojde z nějakého důvodu k příjemci, probíhá jeho opětovné odeslání během dalšího časového intervalu. Tento proces se opakuje do doby, než dojde ke správnému doručení paketu k příjemci.

### 3.2.4 L2CAP

Logical Link Control and Adaptation Protocol (L2CAP) má dvě hlavní funkce. Zaprvé funguje jako multiplexor protokolů, vezme několik protokolů z vrchních vrstev a zapouzdří je do paketu BLE (tento proces funguje i naopak). Druhou funkcí je segmentace paketů na menší části, které se vejdou do 27 bajtového paketu na straně odesílatele. Na straně příjemce se tyto pakety opět zahrnou do jednoho velkého, který je následně předáván vyšším vrstvám. Dalším úkolem L2CAP je správa QoS (Quality of Service) pro vyšší vrstvy [29].

### 3.2.5 SMP

Security Manager Protocol (SMP) je protokol a řada bezpečnostních algoritmů, které umožňuji zařízením generovat a vyměňovat si bezpečnostní klíče. Tyto klíče následně umožňuji oběma subjektům provádět šifrovanou komunikaci, ověřovat identitu druhého zařízení anebo skrýt veřejnou adresu zařízení. Proces zabezpečení komunikace má celkem tři části [29].

- Pairing – Dojde k vygenerování bezpečnostního klíče, díky kterému mohou obě zařízení přejít na šifrovanou komunikaci. Tento klíč se neukládá, a tudíž ho nelze použít během následujících připojení.
- Bonding – Jedná se o proces, kdy dojde k vytvoření a k výměně trvalého bezpečnostního klíče a následného uložení těchto klíčů.
- Encryption Re-establishment – Pokud byly klíče uloženy, tento proces definuje, jak použít klíče k obnovení šifrované komunikace, aniž by bylo potřeba znovu provádět proces párování.

BLE umožňuje dva způsoby párování, první je LE Legacy Pairing jedná se o starší a méně bezpečnou metodu. Velkou slabinou je, že všechny klíče lze velmi lehce odhalit. Druhou metodou je LE Secure Connections. K využití této metody musí obě zařízení podporovat Bluetooth ve verzi 4.2. Jedná se o novější metodu, která neobsahuje bezpečnostní problémy, kterými trpělo LE Legacy Pairing.

### 3.2.6 ATT

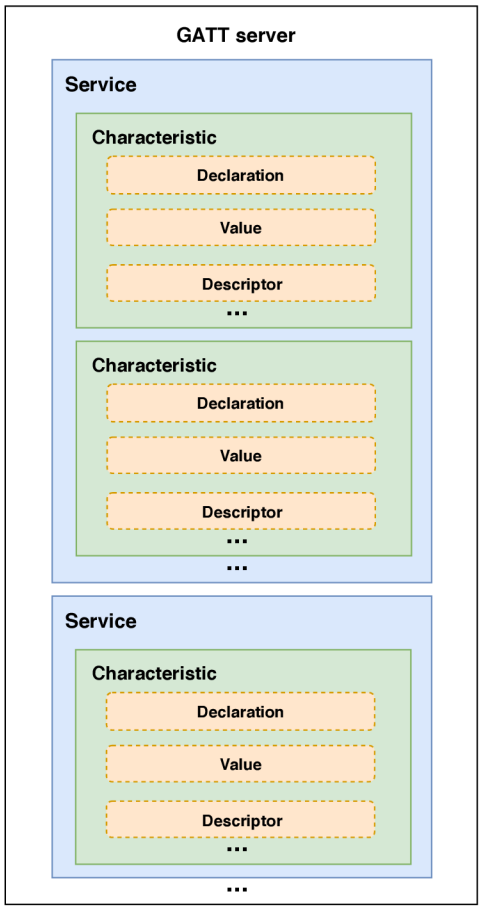
Attribute Protocol (ATT) slouží k výměně dat (atributů) mezi zařízeními. Protokol ATT tvoří nižší vrstvu protokolu GATT viz 3.2.7. a tudíž je s ním úzce svázán. Jedná se o jednoduchý bezstavový protokol typu klient/server. V BLE komunikaci může být zařízení typu server, klient nebo obojí zároveň. Protokol podporuje spolehlivé doručení, tzn. pokud odesilatel neobdrží odpověď na jeho předchozí zprávu, nemůže odesílat další. Jediným stavebním kamenem ATT protokolu je atribut, ten se skládá ze čtyř části (handle, UUID, value a permissions) [29] [36].

- Handle – Je jedinečný 16bitový identifikátor pro každý atribut na konkrétním GATT serveru, díky tomu je každý atribut adresovatelný. Každý GATT server může mít maximálně 0xFFFE (65535) handlerů i když většinou je jich jen pár desítek.
- UUID – (Universally unique identifier) je 128bitový univerzální identifikátor, u kterého je zaručeno, že je globálně jedinečný. Jelikož 16 bajtový identifikátor zabíral velkou část velikosti paketu, kterou udává linková vrstva, byly dále vytvořené další kratší formáty 16bitový a 32bitový. Tyto zkrácené formáty lze použít pouze s UUID, které jsou definované ve specifikaci Bluetooth.
- Value – Jedná se o skutečná data, která se přenáší a jejichž maximální délka může být až 512 bajtů.
- Permissions – Metadata, která určují, jaké operace (čtení, zápis atd.) lze provést u každého konkrétního atributu a s jakým zabezpečením.

### 3.2.7 GATT

Generic Attribute Profile (GATT) podrobně stanovuje, jak pomocí BLE komunikace vyměňovat celé profily a uživatelská data. GATT rovněž poskytuje referenční rámec pro všechny profily založené na GATT. Jsou to určitá pravidla, která musí všichni výrobci zařízení dodržet, aby jejich zařízení mohla mezi sebou správně komunikovat. Toto dělá z GATT klíčovou sekci celého BLE, protože každá jednotlivá položka dat musí být formátována, zabalena a odeslána v souladu s pravidly GATT. Jak již bylo zmíněno GATT využívá ke svoji práci ATT protokol, konkrétně jako přenosový protokol k výměně dat. Tato data jsou uspořádaná v sekci s názvem služby (services). Služby pak dále obsahují a seskupují koncepčně související části uživatelských dat

s názvem characteristics. Tato část nadále obsahují další tři sekce (Declaration, Value, Descriptor) [29] [36]. Celá hierarchie je znázorněná na obr. 3.6.




Obr. 3.6: Hierarchie GATT [37].

- Services – Seskupují funkčně související atributy do jedné společné sekce. Koncepčně se tato metoda velmi podobá objektově orientovanému programování, kde služba představuje třídu.
- Characteristics – Lze je chápat jako kontejnery pro uživatelská data. Vždy obsahují nejméně dva atributy, deklaraci a hodnoty. Deklarace poskytuje informace o uživatelských datech. Hodnota je atribut, který ve svém poli přenáší uživatelská data. Dále se zde mohou nacházet popisovače.
- Descriptors – Poskytuji rozšířené informace o metadatech, které se vyskytují v deklaraci. Popisovač je vždy zařazen za atributem hodnoty.

### 3.2.8 GAP

Generic Access Profile (GAP) je základním kamenem, který umožňuje BLE zařízením spolupracovat mezi sebou. GAP poskytuje nástroje pro interakce v různých

situacích, kterými mohou zařízení BLE projít (proces objevení, vysílání dat, navazovat zabezpečenou komunikaci atd.) [29].

### Role zařízení

Každé zařízení může mít jednu nebo více rolí v jednu chvíli. Každá role si vynucuje určité požadavky na chovaní zařízení. Některé kombinace rolí umožňuji zařízením komunikovat mezi sebou a GAP vytváří interakce mezi těmito rolemi. Role můžou být například vysílač, přijímač atd.

### Režimy a procedury

Režimy jsou jakýmsi zpřesněním role, ve kterém se zařízení nachází. Zařízení může zůstat v jedné roli a přepínat se na určitou dobu mezi různými režimy, aby dosáhlo konkrétního cíle nebo umožnilo vrstevníkovi provádět konkrétní postup. Přepínaní režimu může být automatické podle potřeb zařízení nebo se mohou přepnout pomocí uživatelského rozhraní. Procedura je posloupnost akcí (obvykle řídicí sekvence) linkové vrstvy, které umožňuji zařízením dosáhnout určitého cíle. Procedura je obvykle spojena s režimem na druhém zařízení, takže jejich provázanost je velmi úzká.

### Zabezpečení

GAP z velké části využívá k zabezpečení již dříve zmíněný SMP protokol viz 3.2.5 a nadále jej rozšiřuje. Samotný GAP definuje různé role a režimy související s navázáním šifrované komunikace. Tyto bezpečnostní postupy jsou většinou asymetrické, přičemž centrální a vedlejší zařízení při vytváření a výměně bezpečnostních klíčů prochází opět několika rolemi a režimy. GAP definuje dva režimy zabezpečení a několik dalších úrovní v každém režimu [38].

### Režim zabezpečení 1

Tento režim vyžaduje zabezpečení pomocí šifrování a obsahuje čtyři úrovně:

- Úroveň 1 – Bez zabezpečení (bez ověření a bez šifrování).
- Úroveň 2 – Neověřené párování se šifrováním.
- Úroveň 3 – Ověřené párování se šifrováním.
- Úroveň 4 – Autentizace LE Secure Connections spárování se šifrováním.

### Režim zabezpečení 2

Tento režim vyžaduje zabezpečení pomocí podepisování dat a obsahuje dvě úrovně:

- Úroveň 1 – Neověřené párování s podpisem dat.
- Úroveň 2 – Ověřené párování s podpisem dat.

# 4 Návrhy možných řešení dataloggeru

Cílem bakalářské práce je návrh a realizace dataloggeru, který bude zaznamenávat údaje, které budou ukládaný na paměťovou kartu. Odtud si je bude moci uživatel odeslat do mobilní aplikace, která je popsána v kapitole 6.

Datalogger je určený k měření veličin během procesu kvašení ovoce. Datalogger bude umožňovat měřit vnitřní a venkovní teplotu, kyselost, tlak a koncentraci alkoholu. Na datalogeru bude malý OLED displej, a tři tlačítka, které budou sloužit pro ovládaní dataloggeru. Celé zařízení bude napájené z baterie, tudíž je nutné ho uzpůsobit pro nízkou spotřebu. Datalogger bude přimontovaný na venkovní straně víka kvasné nádoby.

Byly vypracovány tři návrhy, které využívají rozličné MCU i senzory. První řešení se skládá z MCU Firebeetle viz kapitola 2.7 a tří teplotních senzorů DS18B20 viz 4.1.

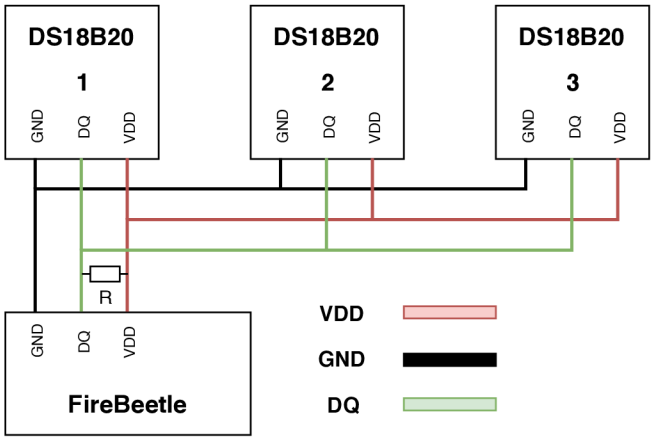
Pro druhé řešení bylo jako MCU vybráno Raspberry Pi viz kapitola 2.8. Pro měření teploty a vlhkosti byl vybrán senzor SHT31-DIS-F, který je podrobně popsán v kapitole 4.2. Pro eliminaci chyb a pro co možná nejlepší přesnost měření bude použit pětkrát stejný senzor. Pro senzor bylo rovněž nutno navrhnout a realizovat schéma zapojení viz kapitola 4.2.1 a desku plošných spojů (DPS) viz kapitola 4.2.2

Třetí řešení využívá MCU ESP32 viz 2.9. Toto řešení bylo vybráno jako finální a proto se mu věnuje celá kapitola 5.

## 4.1 FireBeetle a senzor DS18B20

Teplotní senzor DS18B20 od firmy Maxim (Dallas), lze zakoupit v pouzdře TO-92, které je velmi podobné klasickým tranzistorům. Vyvedené jsou tři nožky a to VDD, GND a DQ. Senzor se k MCU připojuje pomocí 1-Wire sběrnice viz kapitola 2.5.5. Senzor lze napájet pomocí vstupního napětí od 3,0 V do 5,5 V. Teplotní rozsah senzoru činí -55 °C až 125 °C, přičemž na rozsahu -10 °C až 85 ° je odchylka měření ±0,5 °C. U senzoru lze vybrat 4 úrovně rozlišení pro měření teploty (0,5 °C, 0,25 °C, 0,125 °C nebo 0,0625 °C) [39].

Jak již bylo zmíněno senzor byl použit třikrát pro eliminaci chyby a pro přesnější měření. Naměřené data lze zobrazovat na sériové lince a rovněž jsou ukládaná do paměti dataloggeru. Schéma propojení FireBeetlu a tří senzorů DS18B20 je znázorněno na obr. 4.1, kde lze rovněž vidět pull-up rezistor R s hodnotou odporu 4,7 kΩ, který propojuje vodič DQ a VDD.




Obr. 4.1: Schéma zapojení FireBeetlu a senzoru DS18B20.

## 4.2 Raspberry Pi a senzor SHT31-DIS-F

Digitální senzor vlhkosti a teploty SHT31-DIS-F od firmy Sensirion je vyráběn v podobě IC. Pro komunikaci s MCU využívá $I^2C$ sběrnici viz kapitola 4.2.3. Pouzdro senzoru má rozměry 2,5 x 2,5 x 0,9 mm. Senzor má 8 vývodu (SDA, ADDR, ALERT, SCL, VDD, nRESET, R, VSS). Rozsah napájecího napětí činí 2,15 V až 5,5 V. Průměrný odběr senzoru je 1,7 $\mu$A. Teplotní rozsah činí -40 °C až 125 °C, přičemž na rozsahu 0 °C až 90 °C je odchylka měření ±0,2 °C. Odchylka při měření relativní vlhkosti (RH) je ±2 % . Senzor podporuje dvě metody snímaní, u kterých jde nastavit doba trvání jednoho měření (2,5 ms, 4,5 ms a 12,5 ms), obě metody jsou popsány níže [40].


Obr. 4.2: Senzor SHT31-DIS-F [41].

### Jednotlivé měření

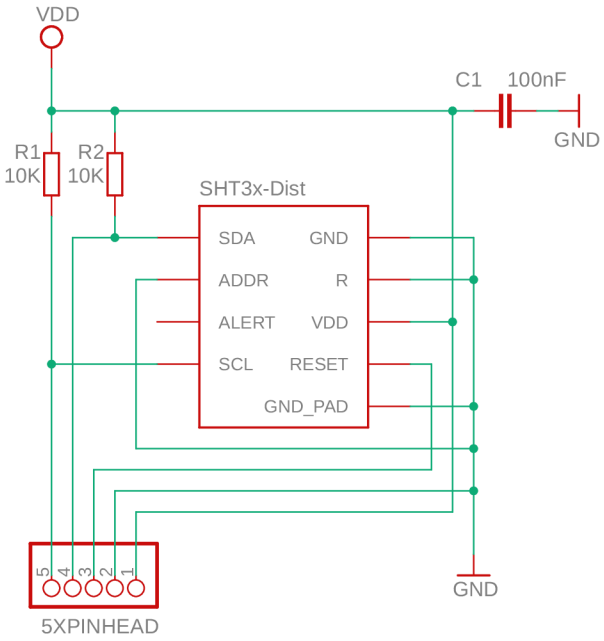
Senzor změří 16bitovou hodnotu teploty a následně 16bitovou hodnotu vlhkosti (přesně v tomto pořadí). Během přenosu je každá hodnota ověřena pomoci kontrolního součtu CRC.

### Periodické měření

Senzor změří 16bitovou hodnotu teploty a 16bitovou hodnotu vlhkosti (přesně v tomto pořadí). Mimo nastavené délky doby měření jde rovněž nastavit frekvenci měření (0,5, 1, 2, 4, 10) Measurements Per Second (MPS).

### 4.2.1 Návrh obvodového schématu

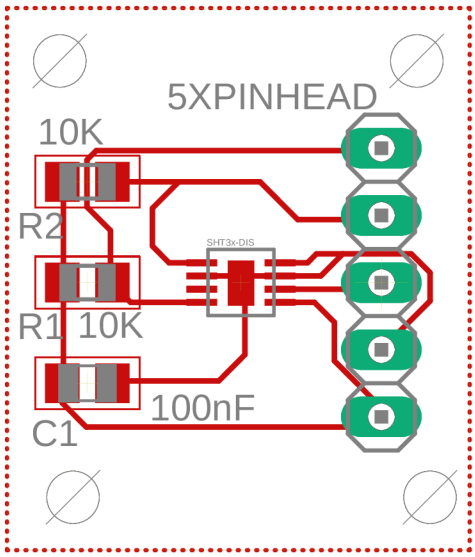
Pro návrh obvodového schématu a DPS byl použit program Eagle education. Schéma zapojení je vidět na obr. 4.3. Senzor SHT31-DIS-F používá pro komunikaci s MCU $I^2C$ sběrnici, proto bylo nutné připojit piny SCL a SDA přes pull-up rezistory R1 a R2 na kladné napětí. Rezistory mají hodnotu odporu 10 kΩ. Dále byl použit kondenzátor C1 s hodnotou 100 nF. Piny SCL, SDA, RESET, GND, VDD a jsou vyvedeny na GPIO hlavici, která slouží k propojení DPS s MCU. Pin ALERT nebude v zapojení použit a piny ADDR, R, GND a GND_PAD jsou připojeny na zem.



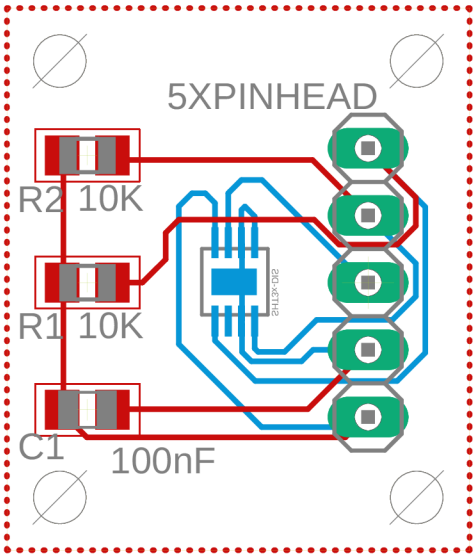

Obr. 4.3: Schéma zapojení senzoru SHT31-DIS-F.

### 4.2.2 Návrh desky plošných spojů

Na základě návrhu elektrického obvodu z kapitoly 4.2.1 jsou realizované dvě verze DPS. První verze je realizována jako jednovrstvá deska, kde všechny komponenty jsou umístěné na jedné straně, návrh desky je na obr. 4.4. Druhá verze je dvouvrstvá deska, kde je senzor SHT31-DIS-F vyveden na druhou stranu než všechny ostatní komponenty, jak jde vidět na obr. 4.5. Všechny pasivní součástky jsou v provedeni SMD řady 0805. Obě verze DPS mají rozměry 20,5 x 17,5 mm. Šířka všech spojů je 8 mils (0,2032 mm) a díry pro uchycení mají průměr 2 mm.




Obr. 4.4: Návrh jednovrstvé DPS pro SHT31-DIS-F.




Obr. 4.5: Návrh dvouvrstvé DPS pro SHT31-DIS-F.

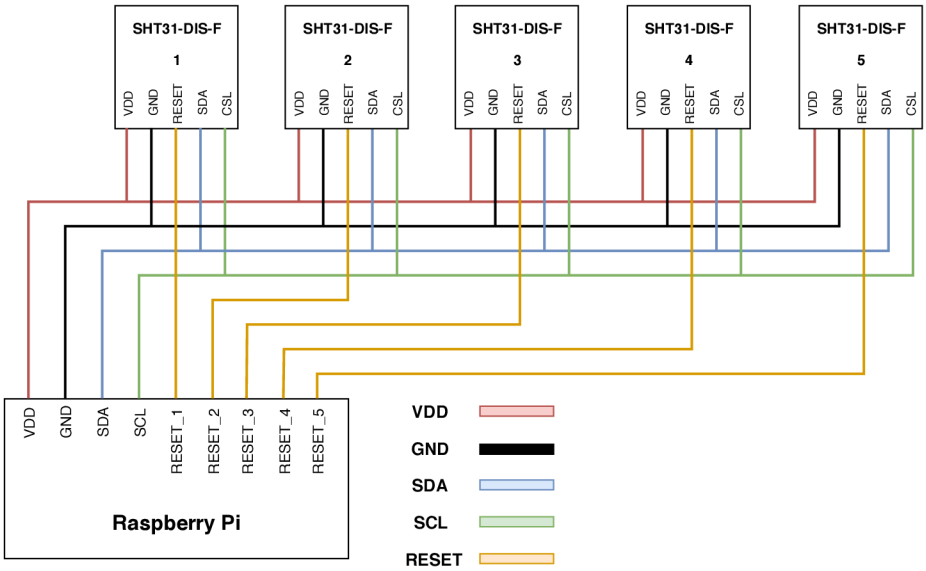
### 4.2.3 Propojení MCU a senzorů

Komunikace mezi MCU (Raspberry Pi) a senzorem (SHT31-DIS-F) bude realizováno pomoci $I^2C$ sběrnice viz kapitola 2.5.3. Jelikož je pin ADDR připojen na zem (logická nula) viz obr. 4.3 bude adrese 0x44 v hexadecimálním tvaru. Protože všech pět senzoru bude mít stejnou adresu, je nutné vymyslet řešení tohoto problému viz kapitoly 4.2.4 a 4.2.5.

### 4.2.4 Softwarové řešení

#### Využití resetu

Jedna z možnosti, jak vyřešit problém se stejnou adresou senzorů je držet všechny senzory v resetu. Jelikož všechny senzory budou mít vyvedený pin RESET na GPIO hlavici, bude možné ovládat reset každého senzoru jednotlivě. Resetu zařízení se dosáhne tak, že na pin RESET se přivede logická nula minimálně na dobu 1 $\mu$s. Z toho vyplývá, že pokud bude na pinu RESET neustále logická nula bude senzor neaktivní. Pokud bude potřeba, aby senzor změřil hodnotu a následně ji odeslal do MCU, přivede se na senzor logická jednička a po odeslání dat se znovu senzor přepne do resetu pomoci logické nuly. Schéma zapojení je znázorněné na obr. 4.6.




Obr. 4.6: Schéma zapojení Raspberry Pi s pěti resety.

#### Vytvoření více I²C sběrnic

Jelikož bude jako MCU využito Raspberry Pi nabízí se vytvořit softwarově další I²C sběrnice pomocí volných GPIO pinů. Tato metoda umožňuje vytvořit až 5 dalších I²C sběrnic tudíž lze využit až 6 sběrnic. Následně by komunikace probíhala s každým zařízením po vlastní sběrnici.

### 4.2.5 Hardwarové řešení

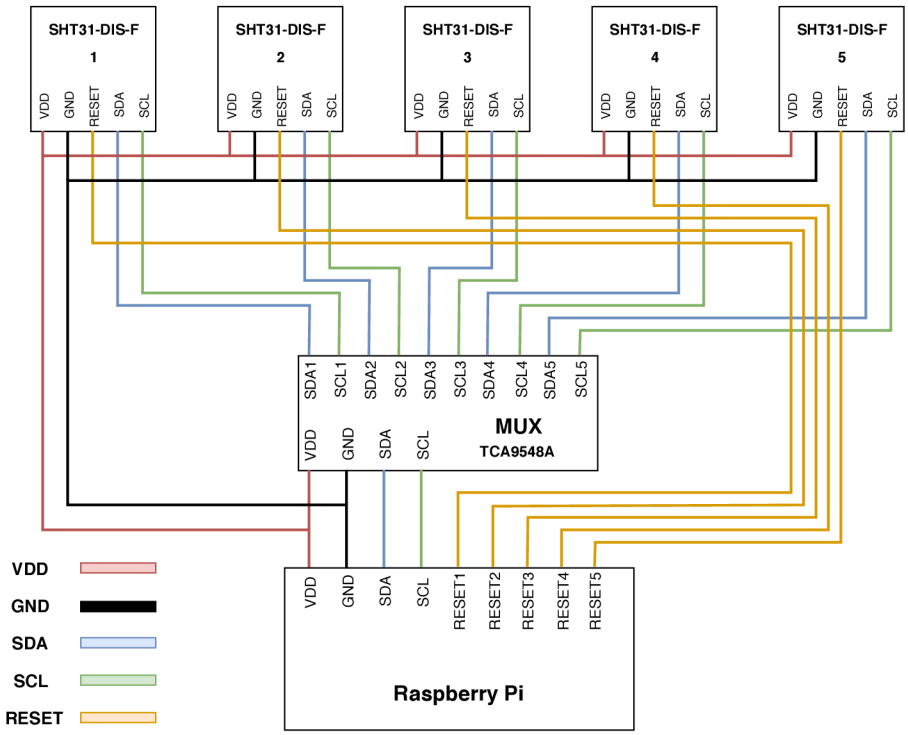
#### Multiplexer

Z hardwarového řešení se jednoznačně nabízí využít $I^2C$ multiplexer (MUX). Těchto multiplexerů je na trhu celá řada, nejčastěji se vyrábějí ve variantě, kdy je možné připojit až 8 zařízení se stejnou adresou. Multiplexer funguje tak, že každému páru pinů SDA a SCL je přiřazena nová adresa, která se programově volí. Velkou výhodou je jednoduchost zařízení. Nevýhodu je, že se jedná o další aktivní zařízení, které vyžaduje napájeni a tudíž klesá výdrž na baterii.

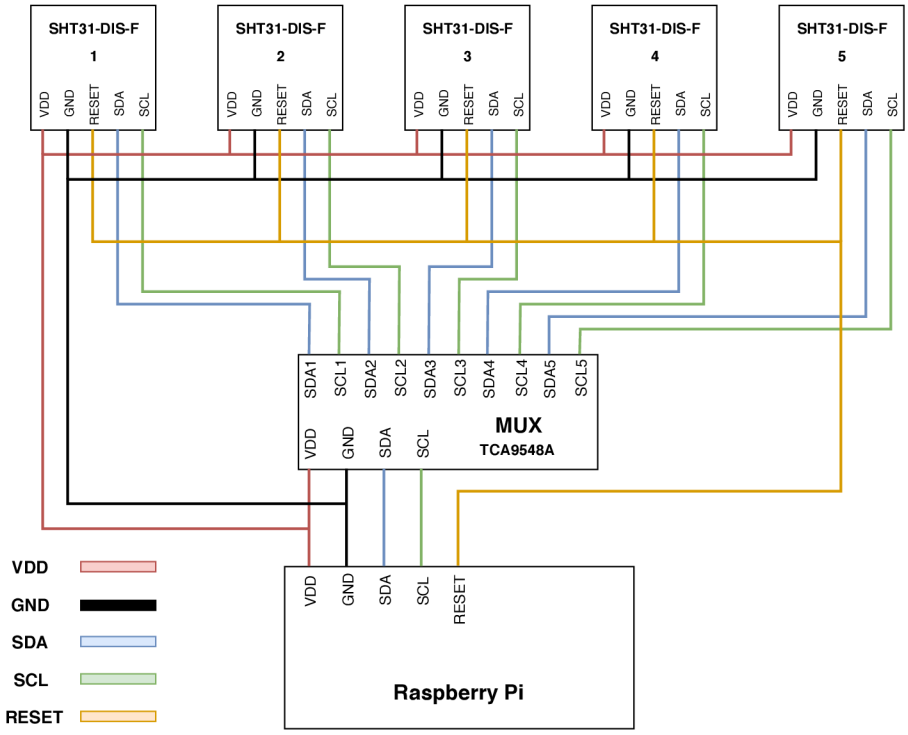
Pro realizaci lze využít například MUX TCA9548A, který umožňuje připojit až 8 zařízení se stejnou adresou. Cena zařízení se pohybuje okolo 70 Kč. Jednotlivé páry pinů SDA a SCL mají adresu 0x70 až 0x77 v hexadecimálním tvaru. Adresu lze volit buď pomocí třech pinů A1 až A3, nebo přímo pomocí programu. Multiplexer se k Raspberry připojí pomocí 4 vodičů a to VDD, GND, SDA a SCL.

Co se týče využiti pinů RESET existuje několik možností:

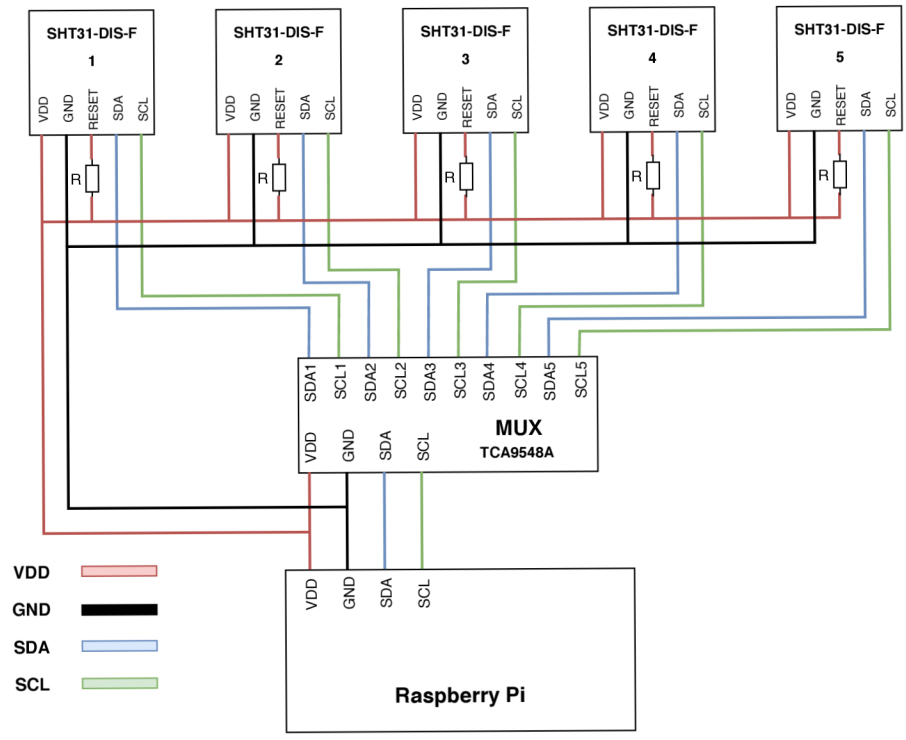
- Piny budou připojené jednotlivě na MCU jak jde vidět na obrázku 4.7
- Piny budou přípojeny přímo na MCU pouze pomocí jednoho vodiče, který ovládá všechny senzory, schéma je znázorněno na obr. 4.8.
- Poslední možností je, že piny se již nebudou vůbec využívat, bude na nich neustále logická jednička. Jedná z možnosti je nechat piny nepřipojené stejně jako u pinu ALERT. Druhá možnost je připojit piny sériově k napájení pomocí rezistorů R s hodnotou 2 kΩ. Schéma je znázorněné na obr. 4.9.




Obr. 4.7: Schéma zapojení Raspberry Pi a MUX s pěti resety.




Obr. 4.8: Schéma zapojení Raspberry Pi a MUX s jedním resetem.




Obr. 4.9: Schéma zapojení Raspberry Pi a MUX bez použití resetů.

# 5 Finální návrh Dataloggeru

Tato kapitola se věnuje finální realizaci dataloggeru. Kapitola je rozdělená na několik části. V první části 5.1 jsou popsány všechny funkce kterými datalogger disponuje. V další části jsou popsány všechny senzory, které se budou využívat pro měření všech stanovených veličin viz 5.2. Následuje část 5.3 s nejdůležitějšími částmi kódu dataloggeru. V Posledních dvou částí je popsáno testováni dataloggeru viz 5.4 a měření spotřeby viz 5.5.

## 5.1 Funkce dataloggeru

Jak již bylo zmíněno datalogger bude měřit vnitřní a venkovní teplotu, koncentraci alkohol, tlak a kyselost roztoku. Všechny tyto veličiny dávají uživateli jasný přehled o tom v jakém stádiu se kvašení nachází. Měření budou prováděna každou hodinu. Jelikož datalogger uživateli nezobrazuje real-time data, nemusí být interval měření tak malý, rovněž se s větším intervalem docílí lepší životnosti na baterii.

Všechna naměřená data se ihned po naměření uloží na paměťovou kartu. Pro interakci s uživatelem je na dataloggeru umístěn malý OLED displej a tři tlačítka, první je pro probuzení zařízení z režimu spánku a zahájení blueetooth komunikace. Druhé tlačítko je pro mazání dat z microSD karty a poslední slouží pro zobrazení aktuálních hodnot na displeji.

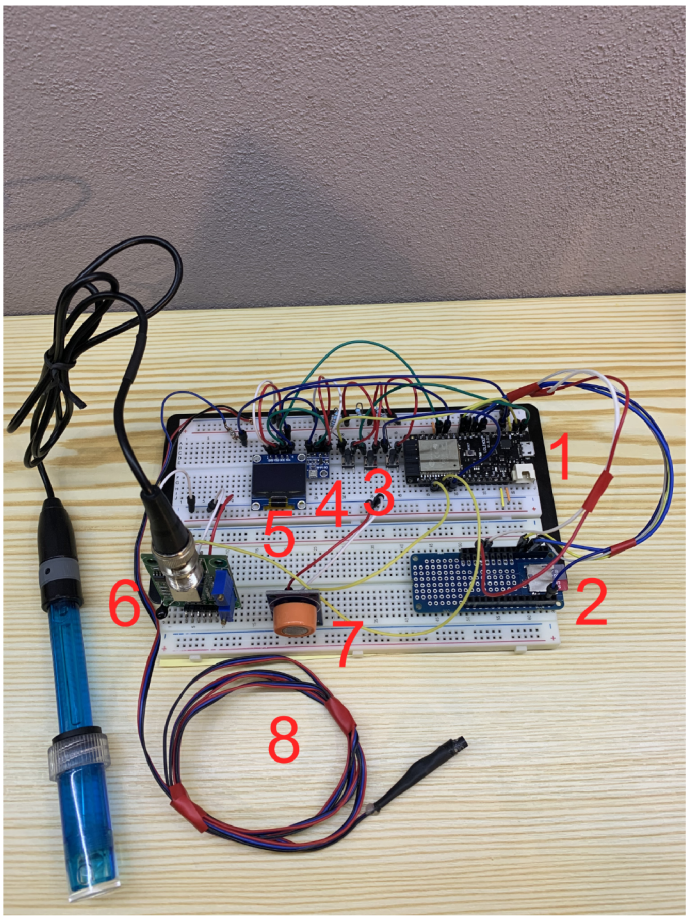
## 5.2 Senzory a moduly

### 5.2.1 Sonda E201 a modul PH-4502C

Sonda je určená k měření PH v tekutinách a to v rozmezí 0 až 14 $\pm$ 0,5 pH. Uvnitř sondy se nachází teplotní čidlo. Rozsah teplotního čidla je 0 °C až 60 °C což k měření teploty kvasu je postačující, jelikož se teploty kvasu pohybují mezi 10 °C až 30 °C. Sonda je s modulem PH-4502C propojena pomocí BNC konektoru a délka kabelu je 95 cm. Aby sonda fungovala správně musí byt celá špička ponořená v měřené kapalině. Jak pH tak teplota se měří pomocí analogových pinů na senzoru a vstupní napětí je 5 V. Sonda a modul jsou vidět na obr. 5.1 a jsou označeny číslem 6.

### 5.2.2 Senzor MQ-3

Teto senzor dokáže ze vzduchu detekovat alkohol, ethanol a v nižší citlivosti i benzín. Výsledek se získává měřením napětí na analogovém výstupu senzoru. Regulace




Obr. 5.1: Fotka dataloggeru na nepaájivém poli s popisem komponentů. 1 - ESP32, 2 - čtečka microSD karet, 3 - tlačítka, 4 - senzor BMP180, 5 - OLED displej, 6 - Sonda E201 a modul PH-4502C, 7 - Senzor MQ-3, 8 - senzor DS18B20.

citlivosti se provádí pomocí potenciometru na zadní straně senzoru. Vstupní napětí senzoru je 5 V. Senzor bude umístěn uvnitř nádrže kde bude měřit koncentraci alkoholu ve vzduchu. Senzor má popisové číslo 7 na obr. 5.1.

### 5.2.3 Senzor tlaku a teploty BMP180

Senzor BMP180 od firmy Bosch dokáže měřit tlak v rozmezí 300 až 1100 hPa s přesností -4 až +2 hPa. Teplotu vzduchu lze měřit v rozmezí 0 až 65 stupňů °C s přesností ± 2 °C. Senzor je s MCU propojen pomocí $I^2C$ sběrnice. Napájecí napětí je 3,3 V a senzor bude umístěný uvnitř nádrže kde bude měřit jak tlak, tak i teplotu vzduchu. Senzor je na obr. 5.1 s popisovým číslem 4.

### 5.2.4 Teplotní senzor DS18B20

Pro měření venkovní teploty se používá senzor DS18B20, tento senzor by již popsán v části 4.1. Senzor bude umístěn z venkovní strany nádrže. Senzor má číslo 8 na obr. 5.1.

### 5.2.5 Čtečka microSD karet

Čtečka microSD karet má vstupní napětí 3,3 nebo 5 V a k MCU je připojená pomocí SPI sběrnice. Na microSD kartu se ukládají jednotlivá měření. Čtečka microSD karet má číslo 2 na obr. 5.1.

### 5.2.6 OLED displej

OLED displej lze vidět na obr. 5.1 s číslem 5. Displej je k MCU připojen pomocí $I^2C$ sběrnice a má rozlišení 128x64 pixelů. OLED panel byl vybrán z důvodu spotřeby, jelikož se rozsvěcí jednotlivé pixely jde dosáhnout nižší spotřeby než u klasických LCD displejů, kde je nutné podsvítit celý displej. Displej má vstupní napětí 3,3 V. OLED displej lze vidět i na obr. 5.2, kde je zapájený na desku plošných spojů.

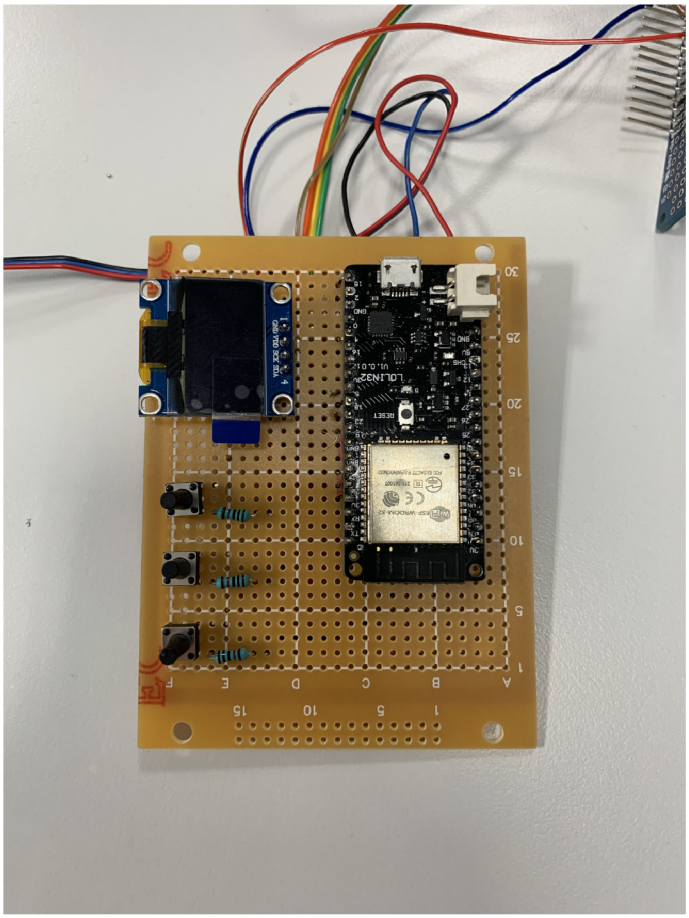
## 5.3 Kód dataloggeru

V této kapitole jsou popsané nejdůležitější funkce v kódu. Celý kód je pojat tak, aby datalogger co největší část svého provozu prospal v hlubokém spánku.

### 5.3.1 Hlavní část programu

Hlavní část programu se celá odehrává ve funkci **setup** odkud se volají další funkce, které obsluhují jednotlivé události. Na úplném začátku programu se zjistí důvod probuzení dataloggeru. Jsou tři hlavní důvody probuzení a to:

- Probuzení časovačem – Toto probuzení nastává po vypršení časovače, který je nastaven přes každým uvedením datalogeru do hlubokého spánku. Výchozí hodnota časovače je nastavená na jednu hodinu. To znamená, že pokud během spánku není zaregistrováno externí probuzení, datalogger tuto dobu prospí v hlubokém spánku. Po probuzení jsou zavolané jednotlivé funkce pro měření veličin a pro uložení naměřeních dat na microSD kartu. Po té datalogger opět přechází na jednu hodinu do režimu spánku.
- Externí probuzení 0 – Jedná se o probuzení z důvodu zmačknutí tlačítka pro bluetooth komunikaci. po zmačknutí tlačítka přechází datalogger do režimu



Obr. 5.2: Zapojení dataloggeru na desce plošných spojů.

párovaní, ve kterém zůstaví po dobu 30 sekund. Pokud během této doby nedojde k připojení k mobilnímu telefonu přechází datalogger opět do režimu spánku. Celé toto probuzení je zahrnuto do celkové doby spánku. To znamená, že když přijde probuzení ve 48 minutu spánku a doba připojení a následné komunikace trvá např. 35 sekund, tak datalogger bude spát již pouze 11 minut a 25 sekund. Po této době přijde další probuzení, ale tentokrát od časovače po kterém následuje měření veličin.

- Externí probuzení 1 – Toto probuzení lze využít pro více zdrojů, které jsou spjaty s jednotlivými tlačítky. V programu lze zjistit z jakého pinu přišlo probuzení a následně opět vykonat určité akce. V programu se využívají dvě tyto probuzení. První je spjato se žádosti o vymazání dat z mircoSD karty. Druhé probuzení se využívá pro zobrazení aktuálních dat na displeji. Obě tyto probuzení se opět počítají do celkové doby spánku, tak jako u externího probuzení 0.

Po tom, co je zjištěný důvod probuzení se v závislosti na tom volají jednotlivé

funkce, které vykonávají dílčí úlohy např. měření teploty, ukládaní dat na microSD kartu, odesílaní dat pomocí bluetooth atd.

### 5.3.2 Funkce connectionBLE

Funkce **connectionBLE** je zavolána v případě, když si chce uživatel vyžádat připojení k bluetooth. Ve funkci jsou vytvořené a nastavené všechny věci nutné pro BLE komunikaci viz 5.1.

Jako první se nastaví jméno zařízení, které bude viditelné pro všechny ostatní bluetooth zařízení. Dále je vytvořený server, který hlídá zda je připojeno nějaké zařízení. Následuje vytvoření service a characteristic, kterým je přiřazené jedinečné UUID, které jsou na výpisu 5.2. Rovněž je k characteristic přiřazená vlastnost notify, tato vlastnost definuje přenos ve směru od serveru (datalogger) ke klientovy (mobilní telefon). Notify nevyžaduje potvrzení o přenosu, tudíž je rychlejší a energeticky méně náročné než druhá možnost přenosu indicate, která vyžaduje potvrzení o doručení zprávy. Následně je zapnutá služba (service) a zapnutá viditelnost zařízení.

Jakmile je všechno nastaveno tak datalogger čeká 30 sekund na spojení. Pokud nedojde ke spojení, tak datalogger přechází do režimu spánku. Naopak pokud dojde ke spojení, tak program vyskočí z čekací smyčky a je zavolána funkce **sendDataBLE**, která je popsáná v části 5.3.3.

```
void BLEConnection() {
  BLEDevice::init("Datalogger_VUT");
  pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());
  BLEService *pService = pServer->createService(SERVICE_UUID);
  pCharacteristic = pService->createCharacteristic(
                      CHARACTERISTIC_UUID,
                      BLECharacteristic::PROPERTY_NOTIFY
                    );
  pCharacteristic->addDescriptor(new BLE2902());
  pService->start();
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->setScanResponse(false);
  pAdvertising->setMinPreferred(0x0);
  BLEDevice::startAdvertising();
  while ((timer + 30000) > millis()) {
    if (deviceConnected) {
      delay(3000);
      break;
    }
  }
  if (deviceConnected) {
```

```
    connectedOLED();
    BLESendData(SD, "/data.txt");
  }
}
```

Výpis 5.1: Funkce pro navázaní BLE komunikace.

```
#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
```

Výpis 5.2: UUID pro service a characteristic.

### 5.3.3 Funkce sendDataBLE

Funkce **sendDataBLE** je na výpisu 5.3. Funkce začíná vytvořením proměnné **file** pomocí které se přistupuje k souboru s daty na microSD kartě. Jako první se kontroluje zda soubor na microSD kartě opravdu existuje, pokud ano začne čtení dat ze souboru pomocí cyklu. Do proměnné **character** se postupně ukládají jednotlivé znaky, pokud je načten konec řádku, tak je splněna podmínka a celý řádek se muže odeslat pomocí funkce notify. Mezi jednotlivými vysíláními je pauza po dobu 10 ms aby nedošlo k zahlcení zásobníku u bluetooth.

Jakmile je čtení ze souboru dokončeno, je odeslán ukončovací znak, který aplikaci dává najevo, že byly odeslány všechny data. Na konec je uzavřen soubor a následuje třísekundová pauza, aby měla aplikace dostatek času na zpracování data a odpojení se od dataloggeru. Průběh bluetooth komunikace ze strany aplikace je popsána v části 6.3.3.

```
void BLESendData(fs::FS &fs, const char * path) {
  File file = fs.open(path);
  if (!file) {
    return;
  }
  char character;
  while (file.available()) {
    character = file.read();
    if (character == '\n') {
      pCharacteristic->setValue((char*)dataFromFile.c_str());
      pCharacteristic->notify();
      dataFromFile = "";
      delay(10);
    }
    else {
      dataFromFile += String(character);
    }
  }
```

```
  pCharacteristic->setValue((char*)dataEnd.c_str());
  pCharacteristic->notify();
  Serial.println(dataEnd);
  file.close();
  delay(3000);
}
```

Výpis 5.3: Funkce pro odesílaní dat přes BLE.

### 5.3.4 Funkce goSleep

Tato funkce se zavolá vždy na konci programu a jako parametr vyžaduje dobu po kterou má datalogger spát. Jako první funkce vypočítá čas, kdy by se měl datalogger probudit, tento čas je uložen do proměnné **wakeUpTime**, která je uložena do RTC paměti, aby ji bylo možné po probuzení načíst. Tato hodnota je důležitá v případě, kdy dojde k externímu probuzení, kdy se z této hodnoty počítá zbývající doba, kterou má datalogger prospat. Dále se kontroluje podmínka zda proměna **wakeUpTime** není menší jak proměna **timeNow**. Tahle situace může nastat v případě kdy datalogger externě probuzený těsně před probuzením časovačem. V tomto případě není použitá vypočítaná hodnota pro zbývající dobu spánku, ale použije se pevně definovaná doba spánku, což je jedna hodina. Na následujících řádcích se povolují jednotlivé probuzení (ext0, ext1 a timer). Poslední příkaz přepíná datalogger do hlubokého spánku. Funkce je na výpisu 5.4.

```
void goSleeep(unsigned long timeOfSleep) {
  unsigned long timeNow = (rtc_time_slowclk_to_us(rtc_time_get(),
  esp_clk_slowclk_cal_get()))/1000000);
  wakeUpTime = timeNow + timeOfSleep;
  if(wakeUpTime <= timeNow){
    timeOfSleep = TIME_TO_SLEEP;
    wakeUpTime = timeOfWakeUp + TIME_TO_SLEEP;
    }
  esp_sleep_enable_ext0_wakeup(GPIO_NUM_33, 1);
  esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK_EXT1,
  ESP_EXT1_WAKEUP_ANY_HIGH);
  esp_sleep_enable_timer_wakeup(timeOfSleep * uS_TO_S_FACTOR);
  esp_deep_sleep_start();
}
```

Výpis 5.4: Funkce pro přepnutí MCU do hlubokého spánku.

## 5.4 Testování dataloggeru

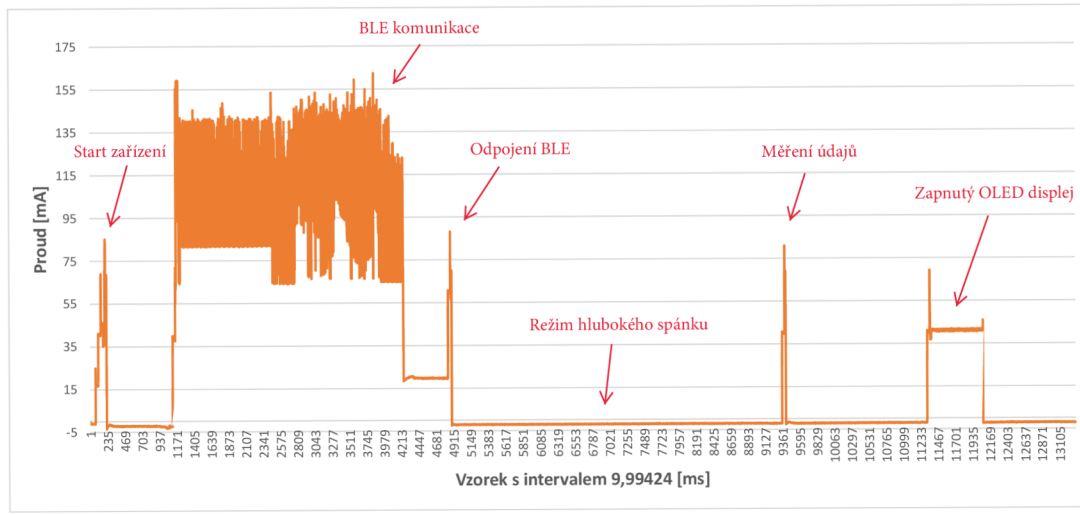
Všechny součástky byly otestovány nejdříve na nepájivém poli, kde se zkoušela jejich funkčnost. Jak mile byly všechny součástky otestovaný byl z nich složen finální datalogger. Celý datalogger byl testován v několik denním provozu. Během testování se vyskytl problém s pH sondou, který je nejspíše způsobený rozdílným napětím zařízení, kde ESP32 běží na 3,3 V a senzor na 5 V. Bohužel se nepovedlo najít 3,3 V alternativu senzoru proto byl ponechán 5 V varianta i za cenu zkreslených hodnot. Jelikož se vyskytla epidemie COVID-19, tak se datalogger nestihl otestovat v reálném provozu, kde by měřil veličiny během kvašení ovoce.

## 5.5 Měření spotřeby dataloggeru

Datalogger velkou část své činnosti stráví v režimu hlubokého spánku. Datalogger má během normálního provozu spotřebu mezi 160 až 260 mA. V režimu hlubokého spánku lze dosáhnout spotřeby až 10 $\mu$A. V tomhle režimu běží pouze ULP (Ultra Low Power) koprocesor a RTC paměť o velikosti 8 kB. Proměnné, které jsou do téhle paměti uloženy zůstávají perzistentní i během hlubokého spánku. Pokud ovšem dojde k restartu MCU, tak jsou data vymazána.

V rámci měření spotřeby jehož výsledky lze vidět na obr. 5.3 bylo zjištěno, že průměrná spotřeba v režimu hlubokého spánku je okolo 2,19 mA. Teoretické hodnoty 10 $\mu$A se nedosáhlo, jelikož na dataloggeru bliká LED dioda a rovněž byl k dataloggeru připojen senzor alkoholu na kterém je rovněž rozsvícená LED dioda indikující napájení. Kdyby byly tyto LED diody odstraněny mohlo být dosaženo lepšího výsledku.

Největší proudový odběr byl zaznamenán v průběhu BLE komunikace, kde se špičky pohybovali až okolo 160 mA. V průběhu měření byl proudový odběr okolo 80 mA a během rozsvíceni OLED displeje byla naměřená hodnota okolo 40 mA.




Obr. 5.3: Graf spotřeby během jednotlivých režimů.

# 6 Aplikace pro mobilní zařízení

Jedním z cílů v rámci bakalářské práce bylo navrhnout a vytvořit aplikaci pro mobilní zařízení, která bude přehledně zobrazovat naměřená data a bude komunikovat se samotným dataloggerem. Tato kapitola je rozdělena na několik částí.

V první části je popsán program Figma pro tvorbu UI (User Interface) designu a prototypu aplikace viz 6.1.1. Samotná struktura aplikace a tvorba UI je popsána v sekci 6.1.2.

V druhé části je popsán samotný vývoj aplikace a všech nástrojů, které byly pro vývoj použity. Je zde odůvodněn výběr a popis programovacího jazyku Dart a UI frameworku Flutter viz 6.2.1.

V neposlední řadě jsou zde popsány nejdůležitější části kódu aplikace viz sekce 6.3.

## 6.1 Design aplikace

Hlavním cílem této aplikace je přehledně a srozumitelně zobrazovat naměřená data, rovněž by měla uživateli přinést jednoduché ovládaní.

### 6.1.1 Program Figma

Pro tvorbu UI/UX designu byl využit program Figma. Tento program nabízí předplatné zdarma pro dvoučlenné týmy designerů. V případě více četných týmu nebo firem je nutné si zakoupit vyšší verzi. Program po startu nabízí mnoho rámečků, které kopírují rozměry reálných zařízeních (Android, iOS, web atd.). Po výběru správného rámečku je už jenom na uživateli zda si stáhne GUI (Graphical User Interface) kolekci jednotlivých operačních systémů, kde si z jednotlivých prvků sestaví UI design, nebo zda si všechny prvky vytvoří sám. Velká výhoda těchto GUI kolekcí je ta, že programátor může rychle vytvořit samotný layout a design, jelikož GUI kolekce obsahuji naprosto stejné prvky, jako knihovny v programovacích jazycích. Programátorovi tudíž stačí znát samotný název prvku, který následně může lehce pozměnit (barva, velikost písma, font atd.).

Jakmile je UI design aplikace hotový přichází čas na tvorbu prototypu. Tlačítkům nebo objektům se nastaví určité chování např. po stisku tlačítka nastavení se zobrazí obrazovka nastavení. Jednoduše řečeno se vytvoří aplikace se statickými daty, kterou může designer procházet, jako by měl v ruce již naprogramovanou aplikaci. Velkou výhodou prototypu je, že lze ihned odhalit chyby v UI a designer ji může ihned opravit. Jelikož jsou všechny chyby v UI designu opraveny rovnou v prototypu,

programátor se může soustředit pouze na logickou část aplikace a práci s daty. Prototyp aplikace lze vidět na obr. 6.1.


Obr. 6.1: První prototyp aplikace v programu Figma.

### 6.1.2 UI aplikace

Celá aplikace je rozdělená do čtyř sekcí dashboard, daily charts, weekly charts a settings, které jsou popsány dále v textu.

#### Dashboard

První co uživatel uvidí po spuštění aplikace je stránka dashboard, která je centrem celé aplikace. Na teto stránce se nacházejí následující widgety.

- Datum – Widget s datem, který si uživatel zvolil a ke kterému jsou svázaná všechny data na jednotlivých widgetech.

- Graf s teplotami – Na grafu se zobrazují tři teplotní údaje za zvolený den. První údaj je vnitřní teplota v nádrži, druhá je venkovní teplota a třetí je rozdíl těchto teplot. Uživatel si rovněž může zobrazit průměrné hodnoty za den, nebo si zobrazit jednotlivé hodnoty v daný čas přímo v grafu.
- Maximální a minimální teplota – Pod hlavní grafem se nacházejí dva widgety, které zobrazují minimální a maximální teplotu, které byly za zvolený den naměřeny. Tyhle widgety dávají uživateli jasný přehled o tom v jakých mezích se teploty na grafu nacházejí. Tyto hodnoty jsou svázané s vnitřní teplotou.
- Výběr data – Po kliknutí na widget se objeví kalendář, kde si uživatel zvolí datum měření, které si chce zobrazit v grafech.
- Dostupné dny – Widget zobrazuje všechny dostupné dny, které si uživatel muže vybrat k zobrazení.
- Aktualizace stránky - Jedná se o tlačítko v pravém horním rohu, které po stisknutí zobrazí nejnovější dostupná data.

### Daily charts

Aby měl uživatel co možná nejvíce informací o všech datech, bylo vytvořeno několik grafů, které tyto data zobrazují.

- Graf s teplotami – Jedná se o ten samý graf jako na hlavní stránce dashboard.
- Graf s min. a max. teplotami – Tento graf zobrazuje minimální a maximální teploty za posledních pět dnů od vybraného dne. Uživatel si opět může přímo v grafu zobrazit přesné teplotní údaje.
- Graf pH – Graf zobrazuje naměřené hodnoty pH tekutiny.
- Graf koncentrace alkoholu – Graf zobrazuje naměřené hodnoty koncentrace alkoholu v nádrži.
- Graf tlaku – Tento graf zobrazuje naměřené hodnoty tlaku v nádrži.

### Weekly charts

Stránka weekly charts zobrazuje ty samé grafy, jako stránka daily charts pouze s tím rozdílem, že se zobrazuje posledních 7 dní od vybraného data.

### Settings

Stránka settings umožňuje uživateli spravovat dvě věci a to bluetooth a úložiště.

- Bluetooth – V záložce bluetooth má uživatel v první řadě informace o tom, jak se zařízení jmenuje a rovněž zde má zobrazené datum a čas od kterého se data začnou zobrazovat. Toto datum musí korespondovat s časem, kdy bylo zahájeno měření. Poslední tlačítko na této stránce slouží k připojení a stažení dat z dataloggeru. Po stisku se uživatel dostane na obrazovku pro vyhledání

bluetooth zařízení. Samotné stažení dat proběhne po stisknuti tlačítka connect u příslušného zařízení.

- Úložiště – V záložce úložiště má uživatel možnost si zobrazit kolik místa zabírají naměřená data na jeho úložišti a rovněž zde může provést vymazaní dat.
- Datum a čas – Tato záložka slouží opět pro kontrolu a změnu data začátku měření.

Finální verze aplikace je zobrazena na obrázcích 6.2.


Obr. 6.2: Snímky z finální verze aplikace.

## 6.2 Vývoj aplikace

Od začátku se uvažovalo o možnosti vyvíjet aplikaci pro obě nejpoužívanější platformy současnosti (Android a iOS). Aby se celý program nemusel psát dvakrát, jak v programovacím jazyku Java pro Android tak ve Swiftu pro iOS, byla hledána možnost vyvíjet aplikaci pro obě platformy současně. Proto byl vybrán multiplatformní UI framework Flutter viz 6.2.1, který přesně řeší tento problém.

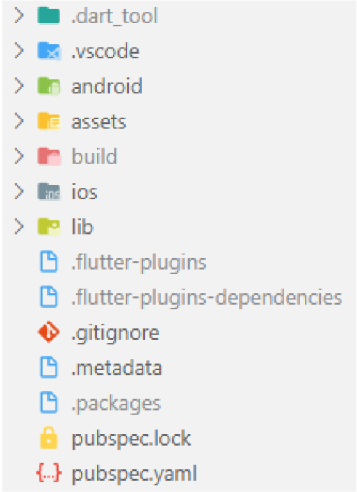
### 6.2.1 Flutter a Dart

Flutter je v porovnání s Javou nebo Swiftem velmi mladý, vznikl v roce 2017 ve společnosti Google. Jak již bylo zmíněno, Flutter dovoluje psát aplikace pro více

platforem současně (Android, iOS, web, desktop) a to vše v jednom programovacím jazyce s názvem Dart. Dart je o něco starší než Flutter vznikl v roce 2011 rovněž ve společnosti Google.

Velkou výhodou frameworku Flutter je téměř nativní výkon jelikož je celý engine systému napsaný v C++. Další populárni vlastností je hot-reloading, to znamená obnovení UI při vývoji bez potřeby nového sestavení celé aplikace. Tato funkce šetří spoustu času při psaní kódu aplikace, jelikož programátor nemusí neustále čekat na sestavení celého projektu.

Celá struktura projektu je zobrazená na obrázku 6.3. Struktura obsahuje několik adresářů a souborů, ty nejdůležitější jsou popsány níže.

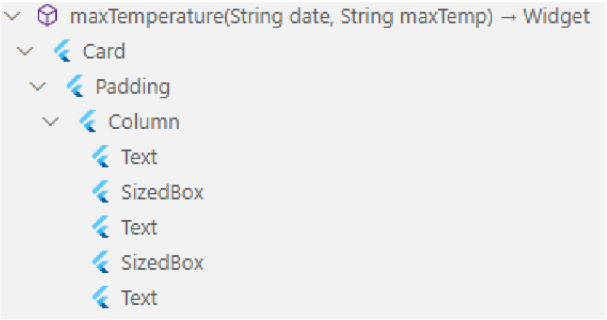



Obr. 6.3: Struktura projektu ve Flutter.

- Adresář android – V tomhle adresáři se nachází klasické soubory a adresáře z Android projektu. Soubory slouží k napojení Flutter aplikace na Android aplikaci. Rovněž se zde nachází podepisovací certifikáty, nebo minimálně podporované SDK.
- Adresář ios – V adresáři se nachází klasický iOS projetk, opět je zde několik vygenerovaných souborů, které slouží pro napojení Flutter aplikace na iOS aplikaci. Zde se rovněž provádí veškerá specifická konfigurace pro iOS platfromu.
- Adresář lib – Jedná se o adresář ve kterém probíhá většina práce na aplikaci. Právě zde jsou umísťovány jednotlivé soubory aplikace. Nachází se zde hlavní soubor main.dart, ve kterém se nachází funkce void main, která spouští celou aplikaci. Povšimněte si, že soubory mají příponu .dart, nikoliv .flutter a to z důvodu, že se celý program píše v jazyce Dart.
- Souboru pubspec.yaml – Soubor slouží pro konfiguraci Flutter aplikace. Rovněž se zde do projektu přidávají rozšiřující knihovny a externí soubory.

Součástí Flutter jsou dva balíčky, které obsahují všechny potřebné UI prvky (tlačítka, ikony, vyskakovací okna atd.). První se jmenuje Material a je především určený pro Android platformu. Druhý balíček je Cupertino a obsahuje UI prvky z iOS platformy. V případě potřeby je možné oba dva balíčky zkombinovat.

Specifikum pro Flutter je to, že všechno je widget. Vysvětleme si toto specifikum na funkci **MaxTemperature**. Tato funkce zobrazuje widget s maximalní teplotou. Funkce vrací typ **Widget**, který je v tomhle případě typu **Card**, u widgetu **Card** můžeme nastavit několik parametru jako barva, tvar, atd. Následně ve widgetu **Card** vytváříme potomka **Padding**, ve widgetu **Padding** je vytvořen další potomek typu **Column**, který má za úkol srovnat jednotlivé widgety, které jsou v něm zabaleny do sloupce. Dále se v **Column** nachází již pouze widgety **Text** ve kterých vypisujeme informace a widgety **SizeBox**, které vytvářejí vertikální mezery mezi widgety **Text**. Tímhle zabalováním jednoho prvku do druhého vzniká stromová struktura, která jde vidět na obr. 6.4.




Obr. 6.4: Stromová struktura widgetu.

## 6.3 Kód aplikace

V následující sekci budou popsány nejdůležitější části programu aplikace.

### 6.3.1 Funkce main

Ve funkci **main** je jako první volán widget **MaterialApp**, který obaluje všechny ostatní widgety. Dále widget **MaterialApp** poskytuje veškeré UI prvky pro platformu Android a umožňuje přecházet mezi jednotlivými stránkami. Dále zde jsou definované stránky mezi kterýma se může přecházet. Jako první se načte stránka **Loading**, která zobrazuje načítací kolečko dokud nejsou načtená a připravená všechna data z úložiště. Po načtení se načte stránka **Wrapper**,která sdružuje tři základní

stránky celé aplikace a to **Dashboard**, **Charts** a **Settings**. Zbylé čtyři stránky jsou dostupné ze stránky **Settings**. Celá funkce je zobrazená na výpisu 6.1

```
void main() => runApp(MaterialApp(
      initialRoute: '/',
      routes: {
        '/': (context) => Loading(),
        '/wrapper': (context) => Wrapper(),
        '/storage': (context) => StorageScreen(),
        '/time': (context) => TimeScreen(),
        '/bluetooth': (context) => Bluetooth(),
        '/checkAdapter': (context) => BluetoothCheckAdapter(),
      },
    ));
```

Výpis 6.1: Funkce main.

### 6.3.2 Načítaní dat ze souboru a knihovna Shared preferences

Celý algoritmus načítaní dat ze souboru funguje tak, že se data ze souboru můžou načíst pouze ve dvou případech. První případ je start aplikace a to vždy nejnovější den, který je k dispozici. Druhý případ je změna dne uživatelem, poté se načte uživatelem vybraný den.

Tyto data nejsou perzistentní to znamená, že po přechodu na jinou stránku a následném návratu na původní stránku by uživatel neviděl žádná data. Tento problém řeší knihovna Shared preferences. Po načtení dat ze souboru jsou data ihned uložená do Shared preferences, která zabezpečuje jejich persistenci. Ukládaní dat do Shared preferences je o dost rychlejší varianta než data načítat přímo ze souboru po každém načtení stránky.

### 6.3.3 Bluetooth komunikace

Bluetooth komunikace je zahájena v moment, kdy uživatel klikne na tlačítko **Connecet** u příslušného zařízení. První řadě se aplikace snaží k zařízení připojit viz 6.2. Připojení může trvat v rozmezí několika sekund.

```
await snapshot.data[index].device.connect();
```

Výpis 6.2: Připojení k zařízení.

Jakmile je aplikace k zařízení připojená začne proces vyhledávaní služeb (services). V jednotlivých službách se nacházejí characteristics, které obsahují již zasílaná data. První je ale nutné ještě najít správnou characteristic s vlastnosti notify. Notify je typ přenosu dat od serveru (datalogger) ke klientovi (mobilní telefon). Jakmile je nalezená správná characteristic je uložena do proměně **characteristicNotify** se

kterou se nadále pracuje. Celá procedura nalezení services a characteristics je na výpisu 6.3.

```
services = await snapshot.data[index].device.discoverServices();
services.forEach((service) {
    for (BluetoothCharacteristic characteristic in service.
        characteristics) {
        if (characteristic.properties.notify) {
            characteristicNotify = characteristic;
        }
    }
});
```

Výpis 6.3: Vyhledávaní services a characteristics.

Aby bylo možné do zařízení zasílat data prostřednictvím notify je nutné tuto možnost povolit u konkretní characteristic viz výpis 6.4.

```
await characteristicNotify.setNotifyValue(true);
```

Výpis 6.4: Povolení zasílaní dat do zařízení.

Jakmile začnou chodit zprávy, tak se zachytí pomocí funkce **listen**. V této funkci se nejdříve zpráva dekóduje a následně se ukládá do proměnné **dataList**. Rovněž se každá zpráva kontroluje zda neobsahuje ukončovací znak. Tento znak označuje poslední zprávu ve vysílaní. Pokud je ukončovací znak zachycen, tak se mobilní telefon odpojí od dataloggeru. Následně jsou z proměnné **dataList** odstraněny nepotřebné zprávy a data jsou uložená do JSON souboru. Celá operace je na výpisu 6.5.

```
characteristicNotify.value.listen((value) async {
    messageDecoded = utf8.decode(value);
    dataList.add(messageDecoded);
    if (messageDecoded == '*') {
        await snapshot.data[index].device.disconnect();
        services = null;
        characteristicNotify = null;
        downloading = false;
        deleteUnnecessaryThings();
        Formater().createJsonFile(dataList, startDate);
        setState(() {
            snapshot.data.clear();
        });
    }
});
```

Výpis 6.5: Příjem zpráv a odpojení od zařízení.

### 6.3.4 Třída Data

Třída Data slouží pro práci s naměřeními daty. Všechny entity, které se ve třídě používají jsou vypsané ve výpisu 6.6. Jelikož jsou data uložená v JSON souboru musí se při každém ukládaní data zakódovat a při načítaní zpátky dekódovat.

```
String date;
List<double> temps;
List<double> ph;
List<double> alcohol;
List<double> tempsOut;
List<double> pressure;
```

Výpis 6.6: Entity.

Struktura JSON souboru je znázorněna na výpisu 6.7. Jednoduše řečeno se jedná o List<Data>, kde objekt **Data** tvoří šest Map<String, dynamic>, každá mapa zastupuje jednu měřenou veličinu. První Mapa má klíč **date**, do které se ukládá datum měření. Následuje pět Map, které mají klíče **temps**, **ph**, **alcohol**, **tempsOut**, **pressure** a jako hodnotu nesou List<double>. Do každého Listu je uloženo 24 hodnot, které byly za den naměřeny.

```
[{
    "date": "20.04.2020",
    "temps": [25.2, 25.8, 26.0, 26.1, 26.2, ...],
    "ph": [4.8, 4.9, 5.1, 5.2, 5.2 ...],
    "alcohol": [7.2, 7.6, 7.7, 7.7, 7.7 ...],
    "tempsOut": [23.2, 23.8, 24.0, 24.1, 24.2, ...],
    "pressure": [982.9, 983.2 983.2, 983.2, 983.3, ...]
},
{
    "date": "21.04.2020",
    "temps": [26.2, 26.2, 26.2, 26.3, 26.3, ...],
    "ph": [5.5, 5.5, 5.6, 5.7, 5.7, ...],
    "alcohol": [7.7, 7.7, 7.7, 7.7, 7.8, ...],
    "tempsOut": [24.7, 24.8, 25.0, 25.0, 25.2, ...],
    "pressure": [984.2, 984.2 984.2, 984.3, 984.3, ...]
},
...]
```

Výpis 6.7: Struktura JSON souboru.

Samotná data jsou ze souboru načtená jako String, který se následně pomocí funkce **dataFromJson** viz 6.8, dekóduje na mapu String a dynamického datového typu. Dynamický typ je použit, protože za hodnotou v mapě lze použít libovolný primitivní datový typ jako int, String, double, nebo bool, rovněž lze použit i List těchto typů. Nadále je tato mapa předána speciálnímu factory konstruktoru, který je na výpisu 6.9. Přes tenhle Factory konstruktor lze entity inicializovat.

```
List<Data> dataFromJson(String str) =>
List<Data>.from(json.decode(str).map((x) => Data.fromJson(x)));
```

Výpis 6.8: funkce dataFromJson

```
factory Data.fromJson(Map<String, dynamic> json) => Data(
date: json["date"],
temps: List<double>.from(json["temps"].map((x) => x.toDouble())),
ph: List<double>.from(json["ph"].map((x) => x.toDouble())),
alcohol: List<double>.from(json["alcohol"].map((x) => x.toDouble())),
tempsOut: List<double>.from(json["tempsOut"].map((x) => x.toDouble())),
pressure: List<double>.from(json["pressure"].map((x) => x.toDouble())),
);
```

Výpis 6.9: Factory konstruktor.

Naopak při ukládaní dat do souboru je zavolána funkce **toJson**, která je na výpisu 6.10. Tato funkce vrací mapu String a dynamic, která je následně pomocí toString funkce převedení na String a tento String uložen do JSON souboru.

```
Map<String, dynamic> toJson() => {
"date": date,
"temps": List<dynamic>.from(temps.map((x) => x)),
"ph": List<dynamic>.from(ph.map((x) => x)),
"alcohol": List<dynamic>.from(alcohol.map((x) => x)),
"tempsOut": List<dynamic>.from(tempsOut.map((x) => x)),
"pressure": List<dynamic>.from(pressure.map((x) => x)),
};
```

Výpis 6.10: Funkce toJson.

# Závěr

Cílem bakalářské práce bylo shromáždit informace a udělat si přehled v oblasti dataloggerů, mikrokontrolérů a komunikačních protokolů vhodných pro nízkoenergetickou komunikaci. Nadále bylo nutné navrhnout a zrealizovat funkční datalogger, který bude měřit veličiny během procesu kvašení ovoce. Jako poslední cíl bakalářské práce bylo navrhnout a naprogramovat mobilní aplikací pro zobrazování dat z dataloggeru.

V jednotlivých kapitolách teoretické části byly popsány různé mikrokontroléry vhodné pro nízkoenergetický provoz. Byly popsány periferní obvody a komunikační rozhraní, které se mohou u MCU vyskytovat. Další část se zaměřovala na komunikační protokoly vhodné pro nízkoenergetickou komunikaci. V této části byl podrobně popsán komunikační protokol Bluetooth Low Energy, který se používá u finální realizace dataloggeru a byla vysvětlená funkčnost jednotlivých vrstev jeho modelu.

V rámci praktické části byly vytvořeno několik návrhů možných řešení dataloggeru, které využívají rozličné MCU i senzory pro snímaní veličin. MCU musí splňovat několik vlastností, které jsou důležité pro tenhle druh zařízení (datalogger). Za prvé musí být energeticky nenáročné to znamená, že by mělo disponovat ultra úsporným režimem, který se bude zapínat ve chvíli, kdy od zařízení nebude nic potřeba. Další vlastností která velmi úzce souvisí předchozím bodem je nízkoenergetická komunikace s ostatními zařízeními. V neposlední řadě by mělo být k zařízení dostatek dokumentace.

První návrh počítá s FireBeetlem jako řídící jednotkou dataloggeru, ke kterému jsou pomocí 1-Wire sběrnice připojené tři senzory DS18B20. Problém u tohoto návrhu se vykytl u MCU FireBeetle, ke kterému je velmi omezená dokumentace a velmi malý počet knihoven. Dalším problémem byla BLE komunikace, která nefungovala podle představ.

Druhá varianta využívá jako řídící jednotku Raspberry Pi. Jako teplotní senzory byly vybrány SHT31-DIS-F, pro které byly navrhnuty a realizovány dvě varianty DPS. Rovněž bylo navrhnuto několik možností zapojení využívající sběrnici $I^2C$. Tato varianta byla rovněž zavrhnuta jelikož Raspberry Pi se nehodí pro nízkoenergeticky provoz, který byl u návrhu jedním z důležitých faktorů.

Jako finální varianta dataloggru bylo vybráno řešení, které využívá MCU ESP32. Tento MCU splňuje všechny požadavky, které jsou kritické pro datalogger to znamená disponuje několika energetickými režimy, lze využít nízkoenergetickou komunikaci (BLE) a rovněž je k ESP32 spousta kvalitní dokumentace, která řeší veškeré aspekty zařízení. Datalogger je vybaven pěti senzory a pro snímaní kyselosti roztoku, koncentraci alkoholu v ovzduší, tlaku, venkovní a vnitřní teploty. Rovněž je vybaven ovládacími prvky a OLED displejem pro interakci s uživatelem. Celý program da-

taloggeru je koncipován tak, aby co největší část svého provozu strávil v hlubokém spánku a probouzel se jenom v nejnutnějších případech jako měření veličin, nebo při BLE komunikaci. Jelikož se vyskytl problém s nahráváním programu do dataloggeru, které bylo způsobené připájením dataloggeru na desku plošných spojů, je datalogger odevzdán na nepájivém poli.

Poslední z cílů bakalářské práce bylo navrhnout a naprogramovat aplikaci pro mobilní telefon, která umožní uživateli stahovaní dat z dataloggeru a následně je přehledně zobrazit v grafech. Od začátku bylo zamýšleno, že se aplikace bude vyvíjet pro obě mobilní platformy (Android, iOS) současně. Proto tento fakt byl vybrán programovací jazyk Flutter, který umožňuje psát aplikaci pro obě platformy současně. Aplikace uživateli umožňuje si zobrazit jednotlivé dny měření a to hned v několika grafech, které zastupují jednotlivé měřené veličiny (teplota, koncentrace alkoholu v ovzduší, kyselost roztoku a tlak). Rovněž lze přes aplikaci stahovat data z dataloggeru pomocí BLE.

# Literatura


[1] *Data logger* Wikipedie [online]. Wikipedie: Otevřená encyklopedie., 2019 [cit. 2019-10-23]. Dostupné z: <https://en.wikipedia.org/w/index.php?title=Data_logger&oldid=917531572>.

[2] Bc. DRÁB, Dominik. *Zařízení pro záznam dat* [online]. Brno, 2018 [cit. 2019-10-23]. Dostupné z: <https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=172660>. Diplomová práce. VUT. Vedoucí práce Ing. Jiří CTIBOR.

[3] *Von Neumannovo schéma*. Muni [online]. Brno: MUNI [cit. 2019-10-23]. Dostupné z: <https://www.fi.muni.cz/usr/pelikan/ARCHIT/TEXTY/VNEUM.HTML>

[4] *Von Neumannovo schéma*. GVPDOC.COMEHERE.CZ [online]. 2015 [cit. 2019-12-02]. Dostupné z: <http://gvpdoc.comehere.cz/doku.php?id=wiki:informatika:hardware:neumann>.

[5] *Architektura počítače*. Github [online]. [cit. 2019-10-23]. Dostupné z: <http://michaelkuty.github.io/ssz-ai-hk-3/tech/2.html>.

[6] *Harvardská architektura*. GVPDOC.COMEHERE.CZ [online]. 2015 [cit. 2019-12-02]. Dostupné z: <http://gvpdoc.comehere.cz/doku.php?id=wiki:informatika:hardware:harvard>.

[7] *CISC*. Wikipedie [online]. Wikipedie: Otevřená encyklopedie., 2019 [cit. 2019-10-23]. Dostupné z: <https://cs.wikipedia.org/wiki/CISC>.

[8] FÜHRER, Ondřej. *Architekura mikroprocesoru AVR ATMega (Pokročilé architektury počítačů)* [online]. Ostrava: Vysoká škola báňská [cit. 2019-10-23]. Dostupné z: <http://wh.cs.vsb.cz/mil051/images/c/ca/PAP_Architektura_procesoru_AVR_ATMega_(Ondřej_Führer).pdf>.

[9] *AVR microcontrollers*. Wikipedie [online]. Wikipedie: Otevřená encyklopedie., 2019 [cit. 2019-10-23]. Dostupné z: <https://en.wikipedia.org/w/index.php?title=AVR_microcontrollers&oldid=915418379>.

[10] *Blokové schema ATMega 32*. Souepl.cz [online]. [cit. 2019-12-02]. Dostupné z: <https://www.souepl.cz/wp-content/ucitele/valecka/at_mega_32_popis%20%28Opraveno%29.htm>.

[11] MATOUŠEK, David. *Práce s mikrokontroléry ATMEL ATmega16*. Praha: BEN, 2016. ISBN 80–7300-174-8.

[12] HAVLENA, Marek. *Datalogger s paralelním záznamem dat* [online]. Brno, 2012 [cit. 2019-10-25]. Dostupné z: <https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=55084>. Bakalářská práce. VUT. Vedoucí práce Ing. Daniel ZUTH, Ph.D.

[13] *AVR® Interrupts*. Microchip.com [online]. wikidot.com, 2019 [cit. 2019-10-28]. Dostupné z: <https://microchipdeveloper.com/8avr:int>.

[14] *Atmel ATmega328P datasheet* [online]. San Jose: Atmel Corporation, 2015 [cit. 2019-10–28]. ISBN 7810D–AVR–01/15. Dostupné z: <http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf>.

[15] *Digital Input/Output Ports on AVR*. Microchip.com [online]. wikidot.com, 2019 [cit. 2019-10-28]. Dostupné z:<https://microchipdeveloper.com/8avr:ioports>.

[16] DUDÁČEK, Karel. *Sériová rozhraní SPI, Microwire, I2 C a CAN* [online]. 2002 [cit. 2019-10-25]. Dostupné z:<http://home.zcu.cz/~dudacek/NMS/Seriova_rozhrani.pdf>.

[17] HONS, Viktor. *Záznamník* [online]. Brno, 2019 [cit. 2019-10-28]. Dostupné z: <https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=190498>. Bakalářská práce. VUT. Vedoucí práce Doc. Ing. Petr BENEŠ, Ph.D.

[18] *Serial Peripheral Interface*. Wikipedia [online]. Wikipedia, The Free Encyclopedia., 2019 [cit. 2019-12-02]. Dostupné z: <https://en.wikipedia.org/wiki/Serial_Peripheral_Interface#/media/File:SPI_8-bit_circular_transfer.svg>.

[19] *Stručný popis sběrnice I2C a její praktické využití k připojení externí eeprom 24LC256 k mikrokontroléru PIC16F877*. Vyvoj.hw.cz [online]. 2000 [cit. 2019-10-28]. Dostupné z:<https://1url.cz/CM7uZ>.

[20] *Komunikace po sériové sběrnici I2C*. Root.cz [online]. 2009 [cit.2019-12-02]. Dostupné z: <https://www.root.cz/clanky/komunikace-po-seriove-sbernici-isup2supc/>.

[21] *Sběrnice 1-Wire*. Vyvoj.hw.cz [online]. Redakce HW serveru, 2004 [cit. 2019-12-20]. Dostupné z: <https://vyvoj.hw.cz/navrh-obvodu/rozhrani/sbernice-1-wiretm.html>.

[22] *What is Arduino?* Arduino.cc [online]. [cit. 2019-12-10]. Dostupné z: <https://www.arduino.cc/en/Guide/Introduction>.

[23] *FireBeetle Board 328P.* Dfrobot [online]. [cit. 2019-10-28]. Dostupné z: <https://wiki.dfrobot.com/FireBeetle_Board-328P_with_BLE4.1_SKU__DFR0492#target_0>.

[24] *What is a Raspberry Pi?* Opensource.com [online]. [cit. 2019-12-10]. Dostupné z: <https://opensource.com/resources/raspberry-pi>.

[25] *ESP32 je tu. Co přinese nástupce ESP8266?* ROOT.cz [online]. [cit. 2020-05-08]. Dostupné z: <https://www.root.cz/clanky/esp32-je-tu-co-prinese-nastupce-esp8266/>.

[26] *Síťové architektury.* Earchiv [online]. 2005 [cit. 2019-11-20]. Dostupné z: <http://www.earchiv.cz/b05/b0500001.php3>.

[27] *What's the Difference Between Bluetooth Versions 2.x, 3.x, 4.x and 5.x. The Droid Guy* [online]. 2019 [cit. 2019-11-20]. Dostupné z: <https://thedroidguy.com/2019/11/whats-difference-between-bluetooth-versions-2-x-3-x-4-x-5-x-1065792>.

[28] *Bluetooth versions.* Pcmag [online]. [cit. 2019-11-20]. Dostupné z: <https://www.pcmag.com/encyclopedia/term/69982/bluetooth-versions>.

[29] TOWNSEND, Kevin, Robert DAVIDSON, AKIBA a Carles CUFI. *Getting started with Bluetooth low energy: tools and techniques for low-power networking.* Revised First Edition. Sebastopol, CA: O'Reilly, 2014. ISBN 978-1-491-94951-1.

[30] *Introduction to Bluetooth® Low Energy.* Microchip.com [online]. wikidot.com, 2019 [cit. 2019-12-02]. Dostupné z: <https://microchipdeveloper.com/wireless:ble-introduction>.

[31] *Bluetooth® Low Energy Physical Layer.* Microchip.com [online]. wikidot.com, 2019 [cit. 2019-11-20]. Dostupné z: <https://microchipdeveloper.com/wireless:ble-phy-layer>.

[32] *Bluetooth® Low Energy Packet Types.* Microchip.com [online]. wikidot.com, 2019 [cit. 2019-11-20]. Dostupné z: <https://microchipdeveloper.com/wireless:ble-link-layer-packet-types>.

[33] *Bluetooth® Low Energy Connection Process.* Microchip.com [online]. wikidot.com, 2019 [cit. 2019-11-20]. Dostupné z: <https://microchipdeveloper.com/wireless:ble-link-layer-connections>.

[34] *Bluetooth: Specification of the Bluetooth System.* Bluetooth Special Interest Group [online]. [cit. 2019-12-02]. Dostupné z: <https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=421043>.

[35] *Bluetooth® Low Energy Connection Process.* Microchip.com [online]. wikidot.com, 2019 [cit. 2019-12-02]. Dostupné z: <https://microchipdeveloper.com/wireless:ble-link-layer-connections>.

[36] *Bluetooth: ATT and GATT.* Epxx [online]. [cit. 2019-11-20]. Dostupné z: <https://epxx.co/artigos/bluetooth_gatt.html>.

[37] *Attribute and Data Hierarchy.* [cit. 2019-11-20]. Microchip.com [online]. wikidot.com, 2019 [cit. 2019-12-02]. Dostupné z: <https://microchipdeveloper.com/wireless:ble-gatt-data-organization>.

[38] *Bluetooth® Low Energy Security Modes and Procedures.* Microchip.com [online]. wikidot.com, 2019 [cit. 2019-11-20]. Dostupné z: <https://microchipdeveloper.com/wireless:ble-gap-security>.

[39] *DS18B20 Programmable Resolution 1-Wire® Digital Thermometer* [online]. dallas Semiconductor [cit. 2019-12-20]. Dostupné z: <https://cdn.sparkfun.com/datasheets/Sensors/Temp/DS18B20.pdf>.

[40] *Datasheet SHT3x-DIS: Humidity and Temperature Sensor* [online]. 6. SENSIRION, 2019 [cit. 2019-12-04]. Dostupné z:<https://cdn.sos.sk/productdata/60/8c/9dc263f2/sht-31-dis-f.pdf>.

[41] *Digital Humidity Sensor SHT3x (RH/T).* Sensirion.com [online]. [cit. 2019-12-04]. Dostupné z: <https://www.sensirion.com/en/environmental-sensors/humidity-sensors/digital-humidity-sensors-for-various-applications/>.

# Seznam symbolů, veličin a zkratek

| | |
|---|---|
| **A/D** | Analog Digital |
| **ACK** | Acknowledgement |
| **ALU** | Arithmetic Logic Unit |
| **ATT** | Attribute Protocol |
| **BLE** | Bluetooth Low Energy |
| **CAN** | Controller Area Network |
| **CISC** | Complex Instruction Set Computing |
| **CPHA** | Clock Phase |
| **CPOL** | Clock Polarity |
| **CRC** | Cyclic Redundancy Check |
| **DDRx** | Data Direction Register |
| **DPS** | Deska plošných spojů |
| **EEPROM** | Electrically Erasable Programmable Read-Only Memory |
| **FHSS** | frequency hopping spread spectrum |
| **GAP** | Generic Access Profile |
| **GATT** | Generic Attribute Profile |
| **GPIO** | General-purpose input/output |
| **GUI** | Graphical User Interface |
| **I2C** | Inter-Integrated Circuit |
| **IC** | Integrated Circuit |
| **IEEE** | Institute of Electrical and Electronics Engineers |
| **IoT** | Internet of Things |
| **ISM** | Institute of Electrical and Electronics Engineers |
| **L2CAP** | Logical Link Control and Adaptation Protocol |
| **LCD** | Liquid Crystal Display |
| **MAC** | Media Access Control |
| **MCU** | Microcontroller Unit |
| **MISO** | Master In Slave Out |
| **MOSI** | Master Out Slave In |
| **MPS** | Measurements Per Second |
| **MUX** | Multiplexer |
| **NB-IoT** | NarrowBand Internet of Things |
| **OLED** | Organic Light-Emitting Diode |
| **PAN** | Personal Area network |
| **PC** | Personal Computer |
| **PDU** | Protocol Data Unit |
| **PINx** | Pins Input |

| | |
|---|---|
| **PoE** | Power over Ethernet |
| **ProfiBus** | Process Field Bus |
| **PWM** | Pulse Width Modulation |
| **QoS** | Quality of Service |
| **R/W** | Read/Write |
| **RAM** | Random-Access-Memory |
| **RH** | Relative Humidity |
| **RISC** | Reduced Instruction Set Computer |
| **ROM** | Read-Only Memory |
| **RS232** | Recommended Standard 232 |
| **RS485** | Recommended Standard 485 |
| **RX** | Receiver |
| **SCK** | Serial Clock |
| **SCL** | Synchronous Clock |
| **SDA** | Synchronous Data |
| **SIG** | Bluetooth Special Interest Group |
| **SMD** | Surface Mount Device |
| **SMP** | Security Manager Protocol |
| **SPI** | Serial Peripheral Interface |
| **SS** | Slave Select |
| **TWI** | Two Wire Interface |
| **TX** | Transmitter |
| **UBBR** | UART Baud Rate Register |
| **UCR** | UART Control Register |
| **UDR** | UART Data Register |
| **UI** | User Interface |
| **USART** | Universal Synchronous/Asynchronous Receiver and Transmitter |
| **USB** | Universal Serial Bus |
| **USR** | UART status register |
| **UUID** | Universally unique identifier |
| **UX** | User Experience |
| **WiFi** | Wireless Fidelity |
| $\boldsymbol{C}$ | Kapacita $[F]$ |
| $\boldsymbol{f}$ | Frekvence $[Hz]$ |
| $\boldsymbol{I}$ | Proud $[A]$ |
| $\boldsymbol{R}$ | Odpor $[\Omega]$ |
| $\boldsymbol{T}$ | Teplota $[°C]$ |
| $\boldsymbol{U}$ | Napětí $[V]$ |