Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Přesné měření teploty s dálkovým přenosem

Plzeň 2016 Lukáš Lichý

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 30. června 2016

Lukáš Lichý

Abstract

This thesis is about designing a device for precise temperature measurement with remote transmission. The goal of this thesis is to create a device that will contain *CC3200* MCU from *Texas Instruments* company and other components allowing precise temperature measurement. This thesis also contains comparsion of devices for temperature measurement. For the temperature measuring device is further provided an additional circuit board which cooperates with the above-mentioned MCU. Another goal of the work is creating an application that stores and processes measured data on a remote server. The application includes a web server that allows end user to view measured data using a web browser.

Abstrakt

Tato práce se zabývá návrhem zařízení pro přesné měření teploty s bezdrátovým přenosem. Cílem práce je vytvořit zařízení, které bude obsahovat MCU *CC3200* od společnosti *Texas Instruments* a další komponenty, umožňující přesné měření teploty. Práce obsahuje také porovnání jednotlivých zařízení pro přesné měření teploty. Pro vybrané zařízení pro měření teploty je vytvořena přídavná destička, která spolupracuje s výše uvedeným MCU. Dále se práce zabývá vytvořením aplikace, která ukládá a dále zpracovává naměřená data na vzdáleném serveru. Součástí aplikace je i webový server, který umožňuje konečnému uživateli zobrazit naměřená data pomocí webového prohlížeče.

Obsah

1 Úvod

Práce se zabývá problematikou přesného měření teploty. Dalším jejím aspektem bylo vytvoření bezdrátového modulu, který je schopný jak měření teploty, tak vzdáleného přenosu naměřených hodnot. Jádrem modulu je MCU *CC3200* od společnosti *Texas Instruments*. Práce obsahuje mimo jiné popis samotného MCU a srovnání způsobů, kterými lze pro tento MCU vyvíjet aplikace.

Výsledkem práce je tedy zařízení pro přesné měření teploty s bezdrátovým přenosem. Součástí práce je srovnání jednotlivých typů průmyslových teplotních čidel, ze kterých bylo vybráno to, které se pro daný úkol nejvíce hodí.

Dalším krokem bylo vytvořit přídavnou destičku, která obsahuje vybrané čidlo a je schopna komunikace s výše uvedeným MCU. S tímto souvisel další bod práce, zabývající se metodami přesného měření teploty včetně kompenzace rušivých vlivů.

Pomocí výše zmíněného MCU jsou naměřená data z teplotního čidla vzdáleně přenášena na server. Dalším úkolem bylo tedy vytvoření aplikace na straně serveru, která tato data ukládá a dále zpracovává. Součástí aplikace je i webový server, umožňující uživateli zobrazit naměřená data pomocí webového prohlížeče. Webový server dynamicky vytváří webové stránky, obsahující data zanesená do přehledných grafů.

Bezdrátový modul je napájen z baterií, a proto se práce zabývá i možnostmi napájení.

Součástí práce je i měření spotřeby energie výsledného modulu. Naměřené hodnoty jsou použity pro výpočet teoretické výdrže výsledného zařízení v případě, kdy je napájeno z baterií.

2 Teorie

2.1 SimpleLink Wi-Fi CC3200 LaunchPad

Simplelink Wi-Fi CC3200 LaunchPad[21] je vývojový kit osazený mikrokontrolérem (dále MCU) *CC3200*[5]. Tento kit byl vybrán, protože již v základu nabízí bezdrátové připojení Wi-Fi. Tento předpoklad byl důležitý, jelikož se práce zabývá bezdrátovým přenosem. *CC3200*[5] je prvním průmyslově vyráběným MCU, který má zabudovanou funkci Wi-Fi konektivity.

Důležitým předpokladem byl také fakt, že vývoj pro tento MCU stále pokračuje a vycházejí stále nové verze sad vývojových nástrojů, stejně jako firmwarů pro samotné zařízení.

Nabízí také více možností připojení externích zařízení prostřednictvím rozhraní `SPI` (sériové periferní rozhraní) nebo `I2C` (sériová sběrnice). Jednoduché a pohodlné připojení dalších periférií je možné díky modulům, které jsou označovány jako tzv. *BoosterPack*[3] moduly (podrobněji v kapitole 2.1.4).

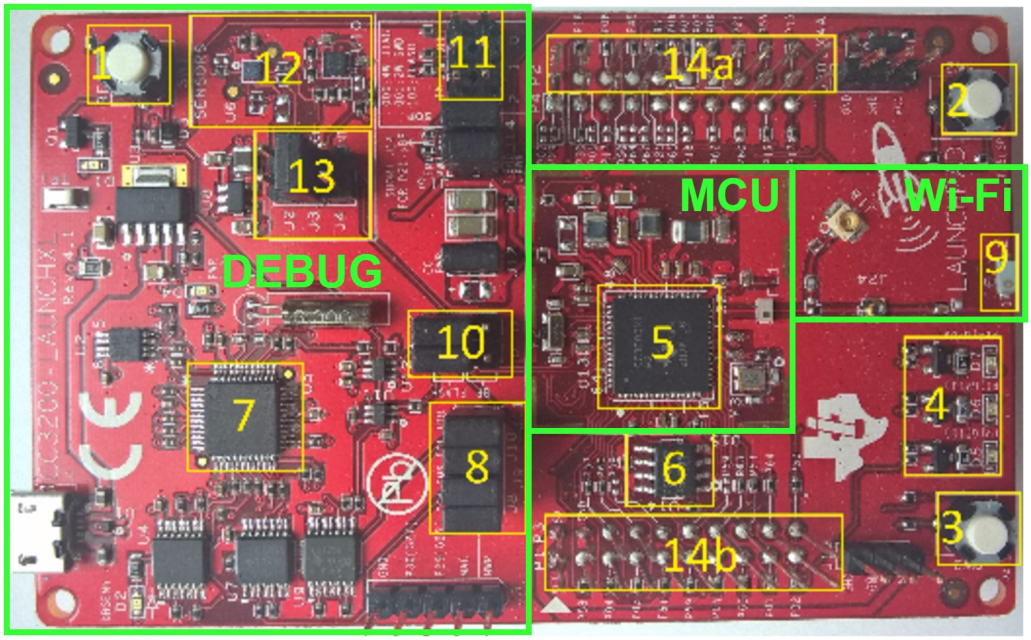
2.1.1 Základní popis

Na obrázku 2.1 je vývojový kit popisovaný v předchozí kapitole 2.1. Jedná se vlastně o konkrétní kus, který byl při vývoji používán. Kit je rozdělen na tři hlavní části (zeleně ohraničené), kterými jsou:

- **DEBUG** část obsahuje součástky a moduly, které slouží pro rozšíření funkcionality MCU *CC3200*. Obsahuje především zařízení pro jednodušší vývoj aplikací pro MCU, sadu senzorů, USB konektor pro připojení kitu k počítači a další. Konkrétní součásti jsou popsané níže.
- **MCU** část obsahuje samotný MCU *CC3200*.
- **Wi-Fi** část obsahuje integrovanou anténu pro Wi-Fi přenos a konektory pro připojení externí antény.

Dále jsou na obrázku očíslovány nejdůležitější části kitu (žlutě ohraničené), které jsou níže popsány:

1. Tlačítko pro resetování *cc3200* MCU. Pokud je k MCU připojen *BoosterPack*[3] modul, může taktéž využít signál tohoto tlačítka.



Obrázek 2.1: SimpleLink Wi-Fi CC3200 LaunchPad

2. Uživatelem programovatelné tlačítko. Při stisku je zvednuta hrana na pinu `GPIO_22`.

3. Uživatelem programovatelné tlačítko. Při stisku je zvednuta hrana na pinu `GPIO_13`.

4. Sada programovatelných LED diod.

5. Hlavní část vývojového kitu - *CC3200* MCU.

6. Sériová FLASH paměť pro uložení programu a uživatelských dat o velikosti 8 Mb.

7. *FTDI JTAG* rozhraní pro emulaci. Obsahuje sériový port pro programování FLASH paměti.

8. Sada pinů pro připojení *JTAG* rozhraní k *CC3200* MCU. Piny je možné rozpojit a připojit jiné externí emulační zařízení.

9. Integrovaná anténa pro bezdrátový přenos Wi-Fi.

10. Sada pinů pro připojení *UART* signálů. *UART* je zařízení pro sériovou komunikaci. V tomto případě slouží jako virtuální *COM* port připojený jako USB zařízení. Za běhu programu může být použito jako sériový vstup/výstup pro komunikaci s terminálem (nebo s počítačem).

11. Sada pinů konfigurující tři různé operační režimy. Jsou to:
    - *JTAG* se čtyřvodičovým režimem,
    - *JTAG* s dvouvodičovým režimem,
    - režim zápisu do sériové FLASH paměti.
12. Sada integrovaných senzorů, se kterými lze komunikovat přes `I2C` rozhraní. Jsou to:
    - teplotní sensor *TMP006*[29],
    - akcelerometr *BMA222*[2].
13. Sada pinů pro připojení integrovaných senzorů k `I2C` rozhraní MCU.
14. Dvojice 20-pinových konektorů, na které jsou přivedeny vybrané výstupy/vstupy MCU. Konektory dále slouží pro připojení *BoosterPack* modulů.

2.1.2 Komunikační rozhraní

Vývojový kit společně s MCU obsahuje tři základní rozhraní, přes které lze komunikovat buď s hostitelským počítačem, nebo s různými integrovanými či externími periférierni.

UART

V tomto případě nejde tak úplně o rozhraní, jako spíše o zařízení pro komunikaci. `UART` [1] je univerzální asynchronní přijímač/vysílač. Jde o zařízení, které převádí paralelní komunikaci na sériovou.

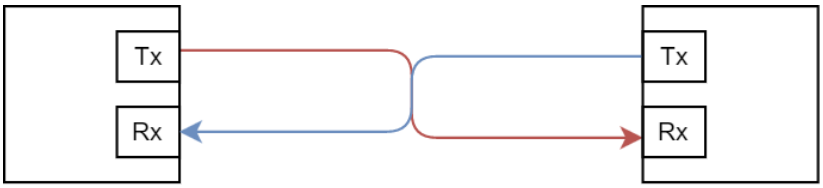
Pro komunikaci používá pouze dva vodiče (a navíc ještě uzemnění):

- **Tx** - vodič pro vysílání,
- **Rx** - vodič pro přijem.

Komunikace mezi dvěma zařízeními je zobrazeno na obr. 2.2. Výstupní signál jednoho zařízení (Tx) je vstupním signálem druhého zařízení (Rx) a naopak.

Výše zmiňovaný vývojový kit obsahuje konkrétní obvod *FT2232D*[15] od společnosti *FTDI* (viz obr. 2.1, oblast 7). Tento modul se stará o připojení MCU k počítači prostřednictvím virtuálního COM portu, který je realizován pomocí klasického USB připojení. Modul *FTDI FT2232D* obsahuje dva porty, a proto zastává ve vývojovém kitu dvě funkce. Jsou jimi:

[1] Universal asynchronous receiver/transmitter



Obrázek 2.2: UART komunikace

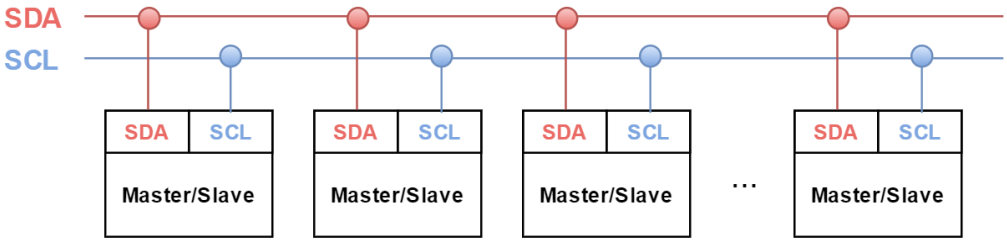
- Emulace pomocí `JTAG` rozhraní. Pomocí tohoto rozhraní lze přistupovat k debugovacím možnostem procesoru a zjednodušit tak vývoj pro tento MCU.
- Emulace virtuálního COM portu.

I2C

Mezi nejzákladnější a nejrozšířenější rozhraní patří `I2C` rozhraní (viz obr. 2.3). Jde o jedno z nejjednodušších komunikačních rozhraní v dané oblasti. Základ tvoří dva vodiče, ke kterým jsou připojeny veškeré periférie včetně samotného MCU. Vodiči jsou:

- **SDA** - Obousměrný datový vodič.
- **SCL** - Hodinový signál, který vysílá *master*.

Jedná se o sběrnici typu *multimaster*. Každý připojený obvod tedy může zastávat jak roli *master*, tak roli *slave*. Z toho důvodu je nutná arbitrace[2].



Obrázek 2.3: I2C rozhraní

Jednotlivé komponenty mají jednoznačně přidělenou adresu, kterou *master* používá pro adresaci, a tedy určení výsledné *slave* stanice, která bude zprávu přijímat.

[2]Proces přidělení sběrnice konkrétní master stanici.

Jednotlivé bajty jsou potvrzované již na úrovni sběrnice, jde tedy o potvrzovaný přenos. Vysílající vysílá ACK[3] bit v horní úrovni (logická hodnota 1) a přijímající potvrzuje příjem dat nastavením ACK bitu na nízkou úroveň (logická hodnota 0).

Existují tři rychlostní kategorie:

- **Standard Mode** - frekvence SCL je 100 kHz.
- **Fast Mode** - frekvence SCL je 400 kHz.
- **High Speed Mode** - frekvence SCL je 3400 kHz.

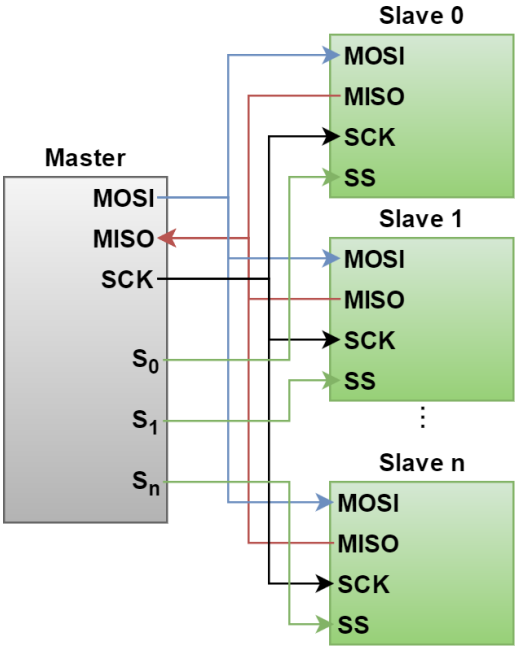
SPI

Dalším rozhraním je `SPI` rozhraní (viz obr. 2.4). V komunikačním okruhu se nachází pouze jedna *master* stanice a jedna nebo více *slave* stanic. Toto rozhraní si již nevystačí pouze s dvěma vodiči. Počet vodičů je proměnný, závisí totiž na počtu *slave* stanic. Vodiči jsou:

- **MOSI** - vodič typu *Master Out, Slave In*. Jde tedy o vodič pro komunikaci vedoucí z *master* stanice do *slave* stanice.
- **MISO** - vodič typu *Master In, Slave Out*. Je to vodič pro komunikaci vedoucí ze *slave* stanice do *master* stanice.
- **SCK** - hodinový signál, který vysílá *master*.
- $\mathbf{S_i}$ - slouží pro výběr konkrétní *slave* stanice, se kterou chce *master* komunikovat.

`SPI` rozhraní je obecně komplikovanější, nabízí však již při standardní hodinové frekvenci 2 MHz vyšší rychlost přenosu. Frekvence hodin může však být podle typu obvodu i 10 MHz.

[3] ACK(Acknowledgement) je potvrzovací signál.



Obrázek 2.4: I2C rozhraní

2.1.3 Integrované senzory

Vývojový kit je doplněn dvěma integrovanými senzory s komunikačním rozhraním `I2C`.

BMA222

BMA222[2] je digitální senzor od společnosti *Bosch*. Jde o tříosý akcelerometr s nízkou spotřebou, a proto je vhodný zejména pro použití v mobilních zařízeních.

Senzor dokáže komunikovat jak přes rozhraní `SPI`, tak přes rozhraní `I2C`. Vývojový kit používá pro komunikaci `I2C` rozhraní.

BMA222 umožňuje měřit akceleraci ve čtyřech možných rozsazích:

- -2g až 2g,
- -4g až 4g,
- -8g až 8g,
- -16g až 16g.

Výběr rozsahu je plně programovatelný.

TMP006

TMP0006[29] je infračervený teplotní senzor, který dokáže měřit infračervené záření až do vzdálenosti cca 8cm. Senzor je schopný měřit teplotu v rozsahu od $-40\,°\mathrm{C}$ do $125\,°\mathrm{C}$. V celém rozsahu měření má poměrně velikou výstupní chybu $\pm 1.5°C$. Při rozsahu od $0\,°\mathrm{C}$ do $60\,°\mathrm{C}$ se výstupní chyba sníží na $\pm 1°C$.

Pro přenos dat je k dispozici dvojice rozhraní. Prvním je `SMBus`[4] a druhým `I2C`. Vývojový kit opět využívá pro komunikaci rozhraní `I2C`.

Jelikož se tato práce zabývá měřením teploty a samotný vývojový kit již obsahuje teplotní senzor *TMP006*, vyvstává otázka, proč byl použit jiný externí teplotní senzor? Odpovědi jsou dvě. Zaprvé tento integrovaný senzor ve vývojovém kitu neměří teplotu dostatečně přesně a zadruhé je počítáno s tím, že výsledný výrobek nebude využívat vývojový kit, ale vytvoří se samostatný obvod s MCU *CC3200*, kde již nebudou tyto integrované senzory.

2.1.4 Nadstavba

BoosterPack slouží koncovým uživatelům vývojových kitů pro jednoduchou a univerzální rozšiřitelnost. Na obrázku 2.5 je zobrazen princip *BoosterPack* modulů. Firma *Texas Instruments* se tímto snaží motivovat uživatele a dovoluje jim tím vytvářet jednoduše nadstavby pro své vývojové kity.

Texas Instruments nabízí knihovny pro návrhové systémy (např. *Eagle*), umožňující vlastní návrh a vývoj *BoosterPack* modulů. Každý si tak může vytvořit vlastní modul a připojit ho ke svému vývojovému kitu.

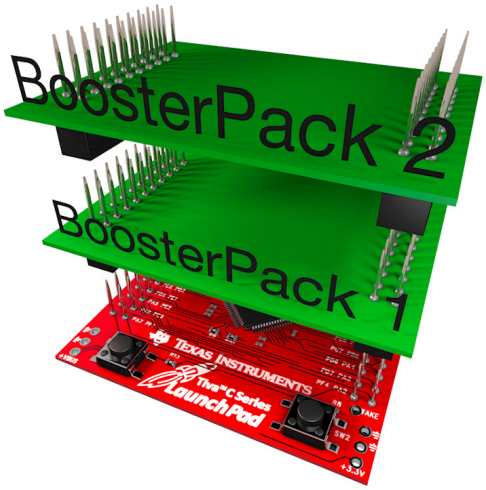
Na internetu jsou k dostání již hotové moduly, konkrétně na stránkách *Texas Instruments*[3].

Další nespornou výhodou těchto modulů je fakt, že je lze vrstvit na sebe a použít tedy hned několik rozšíření najednou. Ve většině případů totiž chceme v rámci jednoho modulu využít pouze několik vstupů/výstupů, které nabízí *BoosterPack* konektory. Ostatní piny konektoru zůstanou v tom případě nevyužity a není důvod, proč by je nemohl využít nějaký další modul.

2.1.5 Řízení spotřeby

MCU *CC3200* nabízí několik možností pro řízení spotřeby. Každá z možností je specifická a určená pro určitý případ použití. Je na koncovém uživateli, kterou možnost si zvolí.

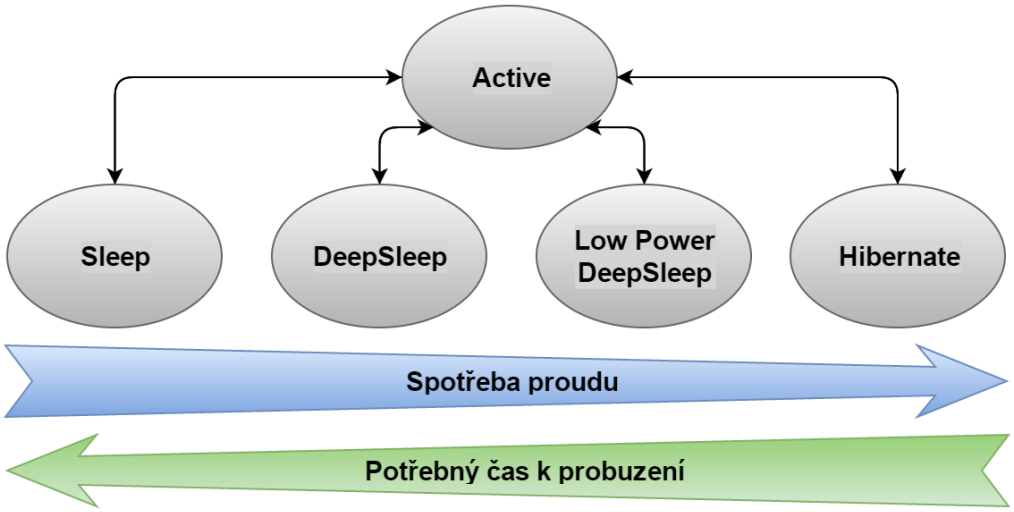
[4]SMBus je odvozen z I2C a je tedy opět založen na dvouvodičové sběrnici.



Obrázek 2.5: Propojení základní vývojové desky s BoosterPack moduly [3]

Na obrázku 2.6 lze vidět jednotlivé stavy, které MCU podporuje. Jsou uvedeny v takovém pořadí, aby je bylo možné porovnat podle výsledné spotřeby proudu a času, potřebného k probuzení z konkrétního stavu.

Každý ze stavů má specifické chování a vlastnosti. Na základě těchto informací je možné vybrat konkrétní stav, který se hodí pro náš případ využití. Někdy se může stát, že rozhodnutí není jednoznačné a bylo by možné použít dva režimy. V tom případě je třeba určit, zda je pro konkrétní případ stěžejní buď nízká spotřeba, a nebo čas, potřebný pro probuzení MCU.



Obrázek 2.6: Druhy spánku pro CC3200 MCU

Stavy a jejich vlastnosti:

- **Sleep**
  - **frekvence hodin:** 80 MHz,
  - **obsah RAM:** zůstává,
  - **kontext CPU:** zůstává,
  - **budící zdroj:** jakýkoliv zdroj přerušení.
- **DeepSleep**
  - **frekvence hodin:** 40 MHz,
  - **obsah RAM:** zůstává (je možné změnit a obsah RAM nezachovat),
  - **kontext CPU:** zůstává,
  - **budící zdroj:** jakýkoliv zdroj přerušení.
- **Low Power DeepSleep**
  - **frekvence hodin:** 32.768 kHz,
  - **obsah RAM:** zůstává (je možné změnit a obsah RAM nezachovat),
  - **kontext CPU:** ztracen (pro následnou obnovu je nutné kontext před vstupem do tohoto módu uložit),
  - **budící zdroj:** časovač, `GPIO_{2,4,11,13,17,24}` a síťové přerušení.
- **Hibernate**
  - **frekvence hodin:** 32.768 kHz,
  - **obsah RAM:** ztracen,
  - **kontext CPU:** ztracen,
  - **budící zdroj:** časovač (k dispozici pouze čítač pro pomalé hodiny na frekvenci 32.768 kHz), `GPIO_{2,4,11,13,17,24}`.

Dalším aspektem výběru vhodného stavu spánku je doba trvání spánku. Je zbytečné například vstupovat do hibernace, pokud je zařízení potřeba uspat pouze na pár sekund. V tom případě se hibernace nevyplatí, jelikož dlouho trvá, než se zařízení probudí, a je lepší tedy zvolit režim *Low Power DeepSleep*. Stejně tak se nevyplatí vstupovat pouze do režimu *Sleep*, pokud

by v něm mělo zařízení setrvat například několik sekund, či minut. Obecné rady k výběru režimu spánku na základě předpokládaného času v něm stráveného:

- **Hibernate** - v řádech minut a víc,
- **Low Power DeepSleep** - v řádech sekund,
- **Sleep/DeepSleep** - v řádech milisekund.

API

Texas Instruments nabízí zákazníkům framework[6] pro řízení spotřeby MCU. Framework nabízí kompletní API pro inicializaci, nastavení a vstup do jednotlivých stavů spánku.

Další možnosti snížení spotřeby

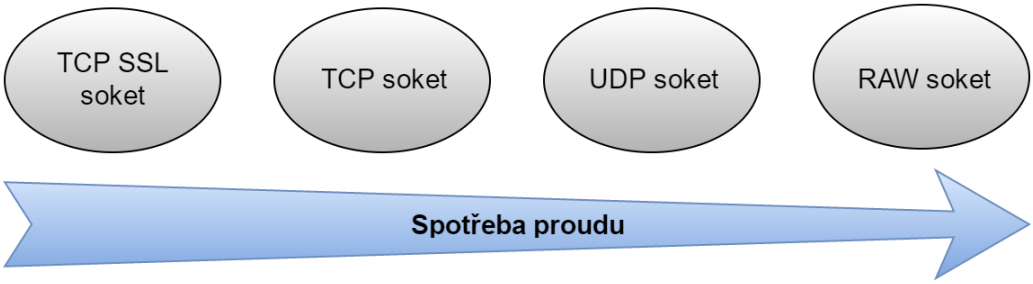
Kromě spánku existují i další možnosti, jak snížit spotřebu MCU. Samozřejmě záleží na konkrétním použití.

Tato práce se zabývá bezdrátovým přenosem využívajícím Wi-Fi. Pro datovou komunikaci musí tedy nejdříve MCU najít požadovanou bezdrátovou síť, přihlásit se k ní, počkat na přidělení IP adresy a dalších nutných informací od DHCP služby routeru, vytvořit paket, a teprve poté poslat data. Jde o zdlouhavý proces, který prodlužuje dobu, po kterou musí být MCU vzhůru, a tím pádem spotřebovávat energii.

Při využití bezdrátového přenosu existuje několik postupů, při kterých lze dosáhnout snížení spotřeby. První možností, jak snížit spotřebu, je snížit režii při tvorbě soketu. Knihovny vývojového kitu nabízejí tvorbu a manipulaci `TCP`, `UDP` a `RAW` soketů. Každý z nich má své výhody a nevýhody. Dále jsou srovnány typy soketů z hlediska spolehlivého přenosu a spojovaného přenosu (spojovaný ve smyslu nutnosti se nejdříve připojit k AP[5]). `TCP` soket nabízí spojovaný spolehlivý přenos, `UDP` nabízí spojovaný nespolehlivý přenos a `RAW` soket nabízí nespojovaný nespolehlivý přenos. Obrázek 2.7 nabízí přehled možných soketů seřazený podle spotřeby energie.

Speciálním případem je právě `RAW` soket, který nevyžaduje spojovaný přenos. MCU tedy nemusí vytvářet spojení mezi AP, ani nijak jinak konfigurovat přenos. Prakticky se vytvoří `RAW` soket, nastaví se mu manuálně typ rámce, číslo rámce, zdrojová adresa, adresa AP, cílová adresa a další atributy. Poté se nastaví manuálně vysílací výkon, vysílací rychlost a délka dat a soket je poslán. Samozřejmě je nutné s tímto počítat a udělat patřičná opatření

[5]Access Point - přístupový bod Wi-Fi sítě



Obrázek 2.7: Spotřeba proudu dle zvoleného soketu

na straně příjemce (AP), aby mohl být takový soket přijat a nebyl například zahozen. AP nemá totiž o zdrojovém zařízení tohoto soketu žádný záznam, a je tedy jen na programovém vybavení daného AP, jak s tímto případem naloží. Dalším úskalím tohoto typu soketu je fakt, že nevyužívá klasických mechanismů Wi-Fi sítí pro přidělení vysílacího pásma (RTS[6], CTS[7]), a proto není jasné, zda data nakonec k AP došla, nebo zda došlo ke kolizi. Faktem ovšem je, že tento typ soketu je nejšetrnější co se týče spotřeby energie.

Jak obrázek 2.7 napovídá, spotřebu proudu také ovlivňuje způsob zabezpečení soketu. Pokud je nutné přenos zabezpečit, je k dispozici několik revizí `SSL/TLS` protokolu. Při zabezpečení soketu je nutné dbát na to, že se spotřeba opět zvýší z důvodu zvýšené režie při navazování spojení pomocí `SSL/TLS` protokolu.

Dalším způsobem snížení energie může být například nastavení statické adresy MCU a vypnutí DHCP služby. Tím se opět sníží režie potřebná pro navázání spojení. S tímto případem je opět nutno počítat na straně přístupového bodu a přidělit statickou IP adresu dle MAC adresy MCU.

Pokud se připojujeme stále ke stejnému přístupovému bodu, je také možné vypnout automatické vyhledávání přístupových bodů v okolí.

Posledním zmíněným mechanismem snížení spotřeby je vypnutí mDNS služby, která se stará o převod IP adres na jména stanic uvnitř menších sítí.

Konkrétní výběr pro tuto práci

Typický pracovní cyklus výsledného řešení (rozuměj MCU s externím čidlem pro měření teploty) bude vypadat následovně:

1. Spuštění MCU (aktivace po hibernaci).

2. Změření teploty (případně dalších veličin).

[6]Ready to send
[7]Clear to send

3. Navázání spojovaného spolehlivého a zabezpečeného přenosu.
4. Odeslání naměřených hodnot.
5. Nastavení časovače hibernace.
6. Vstup do hibernace.

Konkrétní výběr mechanismů pro snížení spotřeby energie je tedy následující:

- Využití **hibernace**, tedy stavu, který nejvíce snižuje spotřebu energie při spánku.
- Využití **statické IP** adresy.
- **Vypnutí DHCP**.
- **Vypnutí mDNS**.

Vzhledem k zadání a požadavku zabezpečeného přenosu bude použit `TCP` soket, `SSL/TLS` protokol a klasické připojení k AP.

2.1.6 Operační systém reálného času

Obecně

Operační systémy reálného času (RTOS) jsou systémy, které zabezpečují spouštění úloh tak, aby dávaly správné výsledky v požadovaném čase. Existují tři základní skupiny úloh:

- **Hard Real Time** - úloha (výpočet) musí být provedena v požadovaném čase za všech okolností. Pokud je čas překročen, považuje se to za selhání s možnými fatálními následky.
- **Soft Real Time** - překročení požadovaného času není fatální, ale nepříjemné. Výsledky jsou stále použitelné, ale se zpožděním.
- **Firm Real Time** - překročení požadovaného času není fatální, ale výsledky jsou dále nepoužitelné.

Jádro RTOS tvoří plánovač, který zabezpečuje správu tasků, a moduly pro správu ostatních objektů RTOS. Základními objekty RTOS jsou:

- **Task** - představuje ucelený jasně definovaný úkol (výpočet). Typický task v RTOS je tvořen inicializační částí (provede se jen při prvním spuštění) a nekonečnou smyčkou (obsahuje blokující volání).

- **Semafory** - pro synchronizaci tasků.
- **Mailboxy** - pro předávání zpráv mezi tasky.
- **Události** - pro signalizaci definovaných událostí.
- **Časovače** - poskytují časové informace, např. pro periodické spouštění plánovače a přeplánování běžících tasků.

Typy RTOS pro MCU CC3200

V MCU *CC3200* je možné použít dva různé RTOS.

Prvním je *FreeRTOS*[14]. Tento software je volně šiřitelný i pro komerční účely. Poskytuje jednotné a nezávislé řešení pro použití RTOS pro mnoho různých architektur a vývojových zařízení. Poskytuje bohatou funkcionalitu, a přitom je jeho jádro tvořeno pouze třemi základními zdrojovými soubory, napsanými v programovacím jazyce `C`. Je stále aktivně vyvíjen, a na jeho domovských stránkách[14] je aktivní fórum, sloužící jako podpora pro vývojáře. Je také dostupná bohatá dokumentace v elektronické i tištěné podobě.

Druhým řešením je *TI-RTOS*[27]. Tento operační systém byl vyvinut přímo společností *Texas Instruments*. Už z tohoto důvodu je jasné, že je cílený právě na produkty od stejnojmenné společnosti. Systém se skládá z několika dílčích částí, které jsou jasně rozděleny do funkčních celků. Jsou jimi:

- **Kernel** - vestavěné jádro systému dříve nazývané *SYS/BIOS*.
- **Drivers and Board Initialization** - ovladače zařízení. Obsahuje ovladače pro Ethernet, GPIO (programovatelné vstupy/výstupy), I2C, I2S, SPI, UART, Wi-Fi, ...
- **Network Services** - implementuje rodinu protokolů TCP/IP, HTTP a TLS/SSL.
- **Interprocessor Communication** - API pro meziprocesorovou komunikaci.
- **Instrumentation** - API pro aplikace reálného času.
- **File System** - poskytuje API pro přístup k souborovým systémům typu FAT.

Postup vytvoření a konfigurace projektu pro použití *TI-RTOS* je popsán v příloze B.

Využití RTOS

Hlavní otázkou zůstává, ve kterém případě použít RTOS, a ve kterém je to zbytečné. Pokud se jedná o aplikaci řešící vícero dílčích funkcionalit, a je možné jednotlivé části logicky rozdělit a definovat jako tasky, je použití RTOS vhodné. Tasky lze poté jednoduše synchronizovat pomocí funkcí, které systém poskytuje, a obecně zjednodušují práci s komplexněji navrženými aplikacemi.

RTOS se na druhou stranu nehodí v případech, kdy je dopředu jasné, že jeho funkce vyjmenované výše nevyužijeme. V příloze C je například vzorový kód pro MCU *CC3200*, pro který by bylo použití RTOS naprosto zbytečné.

Výběr RTOS

V mém případě jsem zvolil *TI-RTOS* od společnosti *Texas Instruments*. Ve výsledné aplikaci pracuji s rozhraními I2C, SPI, UART, dále s TCP sokety a TLS protokolem. Využívám také API pro přístup k souborovému systému a odesílám data pomocí Wi-Fi. Všechny tyto aspekty mě vedly k využití RTOS. Důvod výběru *TI-RTOS* je jasný, je od stejné společnosti jako MCU. Logicky tedy předpokládám, že bude více odladěný a kompatibilní s MCU.

2.1.7 Vývoj pro SimpleLink Wi-Fi CC3200 LaunchPad

Přehled

Existuje více způsobů, kterými je možné vyvíjet aplikace pro tento konkrétní MCU. *Texas Instruments* nabízí a popisuje tři možnosti:

- Využití **Code Composer Studia**[7].
  - Jedná se o software vyvíjený přímo firmou *Texas Instruments*.
  - Je postaven na vývojovém prostředí *Eclipse*[9].
  - Nabízí rozšířené možnosti debugování.
  - Všechny vzorové aplikace pro MCU od *Texas Instruments* jsou pro tento software dostupné.
  - Celkový pocit z tohoto nástroje je bohužel smíšený, jelikož se jedná o obsáhlý a nepříliš přehledný nástroj. Celkově trvá déle, než se uživatel s nástrojem naučí pracovat.
  - Po registraci na stránkách *Texas Instruments* je tento nástroj volně ke stažení.

- Využití **IAR Workbench**[16].
  - Jedná se vývojové prostředí firmy *IAR Systems*.
  - Je zaměřený hlavně na širokou oblast působnosti, podporuje přes jedenáct tisíc zařízení.
  - Software je nabízen volně ke stažení, ale je bohužel časově omezený licencí na 30 dní.
- Využití **gcc** překladače.
  - *Texas Instruments* nabízí návod, jak zprovoznit vývoj aplikaci na platformě *Windows* při použití prostředí *Cygwin*[8].
  - Nabízí pouze kompilaci na příkazové řádce.
  - Absence GUI.
  - Složitá instalace, kromě klasických nástrojů, jako jsou `gcc`, `g++`, `make`, `autoconf`, `automake` a další, je nutné stáhnout i další nástroje, jako speciálně upravený `gcc` pro ARM procesory a *OpenOCD* nástroj pro zpřístupnění debugu pro MCU.

Kromě nabízených prostředků firmy *Texax Instruments* existuje ještě jeden způsob, a to využití volně dostupného softwaru jménem *Energia*[10]. Tento projekt chce přiblížit vývoj pro MCU od společnosti *Texas Instruments* širší veřejnosti. Odstraňuje neduhy *Code Composer Studia*, jako jsou nepřehlednost, robustnost a těžkopádné zacházení. Obecně je *Energia* daleko více uživatelsky přívětivá a dbá především na:

- jednoduchý vývoj,
- intuitivní ovládání,
- zabudované příklady aplikaci pro MCU přímo v prostředí,
- širokou přispívající komunitu vývojářů,
- rychlý začátek vývoje.

Texas Instruments si tyto kladné stránky nástroje *Energia* uvědomuje, a nejspíše proto do svého nástroje *Code Composer Studio* zabudoval možnost tvorby projektů pro *Energii*. Samozřejmě je nutné mít *Energii* nainstalovanou, a poté určit cestu k její instalaci.

Z výše uvedených nástrojů jsem vybral dva, které jsou pro vývoj aplikací pro daný MCU použitelné a zároveň poskytují uživatelsky přívětivé prostředí. Kompilaci v příkazové řádce jsem zavrhl z důvodu složité konfigurace a absence grafického rozhraní.

Code Composer Studio

Prvním vybraným je *Code Composer Studio* od společnosti *Texas Instruments*. Nástroj je dle mých zkušeností nepřehledný, ale na druhou stranu jde o nejvíce odladěný program, který je cílený především na vývoj aplikací pro použitý MCU, jelikož oba pocházejí od stejné firmy.

Jak už bylo psáno výše, nástroj vychází z vývojového prostředí *Eclipse*[9]. Pokud má tedy někdo s tímto prostředím zkušenosti například z předešlého vývoje, bude mít nejspíše jednodušší rozhodování při výběru nástroje, který nakonec použije.

Tvorba projektu s tímto nástrojem pro použitý MCU je ovšem poměrně složitá a těžkopádná. Prakticky existují dvě možnosti jak začít s projektem. Zde jsou uvedeny kladné a záporné stránky obou možností:

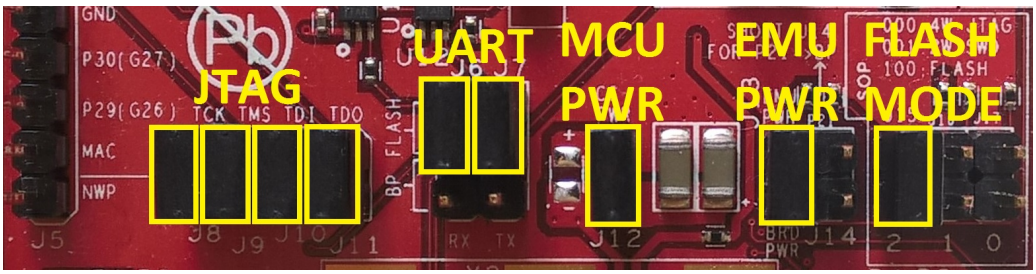
- Vytvoření nového projektu.
  - \+ Začínáme s čistým projektem.
  - \+ Během tvorby projektu si zvolíme, zda chceme vytvořit aplikaci, která bude používat operační systém reálného času (Free-RTOS, TI-RTOS), či ne.
  - \+ Zvolíme si jen ty zdrojové složky pro include, které chceme.
  - \+ Poznáme lépe prostředí použitého nástroje.
  - \+ Pochopíme lépe provázanost projektů a jednotlivé volby konfigurace.
  - − Je to zdlouhavý proces.
  - − Někdy je možné narazit na rozdílné nastavení z důvodu jiné verze nástroje.
- Importování již existujícího projektu (některý z příkladů poskytnutých přímo společností *Texas Instruments*).
  - \+ Rychlost - s projektem můžeme začít v podstatě okamžitě po importování.
  - \+ Možnost využití části kódu.
  - − V projektu máme zbytečné includy, které ani nevíme k čemu slouží.
  - − Používáme konfiguraci, kterou nastavil někdo před námi, a proto ani nevíme, zda vyhovuje našim požadavkům.

V příloze B je uveden postup vytvoření nového projektu pro vývoj aplikace pro MCU *CC3200* od instalace nástroje *Code Composer Studio*[8] až po kompilaci zdrojového kódu. Návod je poměrně rozsáhlý, proto je uveden jako příloha.

Vytvořený projekt, podle postupu uvedeném v příloze B, je zkonfigurován pro použití operačního systému reálného času *TI-RTOS*, který nabízí multitasking, prioritní spouštění úloh, řízení spotřeby a další výhody. Podrobnější informace naleznete v kapitole 2.1.6, nebo na stránkách *Texas Instruments*[26].

Jakmile je provedeno nastavení projektu, je již vše připraveno pro následný vývoj. V momentě, kdy je zdrojový kód projektu připraven a máte v úmyslu jej spustit a vyzkoušet aplikaci, stačí již udělat tři poslední kroky.

Prvním krokem je kontrola, zda jsou správně nastaveny propojky na vývojovém kitu. Nastavte propojky podle obrázku 2.8. Touto konfigurací propojek je zajištěno, že vývojový kit správně používá `JTAG` rozhraní, dále jsou správně připojeny `UART` signály, poté jsou připojeny přívody proudu jak do MCU, tak do emulační části vývojového kitu a poslední propojky zpřístupní FLASH paměť pro zápis.



Obrázek 2.8: Nastavení pinů vývojového kitu pro správnou funkčnost v CCS

Druhým krokem je samotné připojení vývojového kitu k počítači prostřednictvím USB kabelu. Kit se zobrazí ve správci zařízení mezi `COM` porty jako **CC3200LP Dual Port (COM3)**. Konkrétní číslo `COM` portu se může lišit.

Třetím a posledním krokem je samotné zavedení aplikace do vývojového kitu. To lze udělat prostřednictvím menu **Run** → **Debug**. Pokud se v paměti vývojového kitu již nachází nějaká aplikace, není potřeba se obávat, že by byla přepsána. Zavedením aplikace přímo v nástroji CCS v Debug módu tuto dříve uloženou aplikaci nepřepíše. Po rozpojení posledních pinů (viz obr. 2.8 - piny s titulem FLASH MODE) a resetování MCU (viz obr. 2.1 -

[8]Návod byl vytvořen ve verzi 6.1.1.00022.

tlačítko označené číslem 1) MCU nahraje a spustí původní program uložený v sériové FLASH paměti.

Tvorba projektu pro vývojový kit s MCU *CC3200* je tedy poměrně zdlouhavá záležitost. Nicméně jakmile je jednou vytvořený, je možné si tento čistý projekt zálohovat a použít jako šablonu při vytváření dalšího projektu.

Na konci práce v příloze C je přiložen kód, který je určen právě pro takto vytvořený nový projekt. Tímto kódem je možné ověřit alespoň základní funkčnost vývojového kitu. Slouží především pro ověření, že byl projekt správně zkonfigurován a vytvořený program lze spustit.

Energia

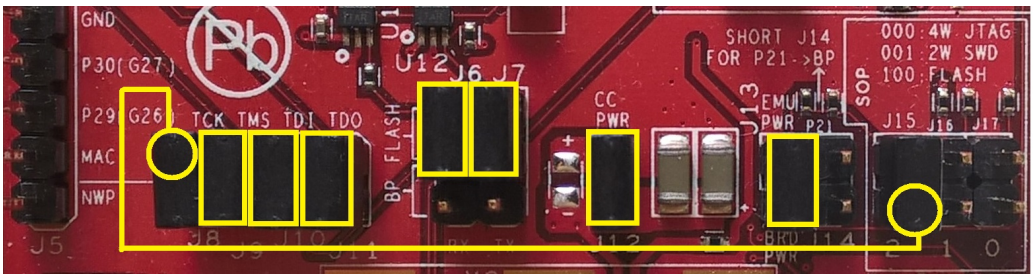
Druhým nástrojem je projekt *Energia*, který přistupuje k řešení problému zcela odlišným způsobem než firma *Texas Instruments* a jejich nástroj *Code Composer Studio*. *Energia* je synonymem především rychlého začátku vývoje, jednoduché instalace a uživatelsky jednoduchého rozhraní.

Přesto je potřeba udělat několik kroků:

1. Prvním krokem je stažení samotného softwaru *Energia* z webových stránek[10].

2. Dalším krokem je pozměnit nastavení konfiguračních pinů na vývojovém kitu (viz obr. 2.9).

   – Horní pin **J8** je potřeba spojit s dolním pinem **J15**.



Obrázek 2.9: Nastavení pinů vývojového kitu pro správnou funkčnost v nástroji Energia

3. Poté je již možné připojit samotný vývojový kit pomocí USB.

   – Je potřeba se ujistit, že vývojový kit je rozpoznán ve správci zařízení jako `COM` port podobný tomuto názvu **CC3200LP Dual**

Port (COM3). Pokud tomu tak není, je nutné nainstalovat ovladače vývojového kitu ze stránek projektu *Energia*[13], věnující se dodatečné instalaci ovladačů vývojových kitů.

- (Dle mých zkušenosti již nebylo třeba instalovat dodatečné ovladače. Pokud se nemýlím, při instalaci **SDK** z kapitoly 2.1.7 byly již ovladače vývojového kitu automaticky nainstalovány.)

4. Posledním krokem je spuštění samotného nástroje *Energia* a jeho konfigurace pro konkrétní vývojový kit.
   - V menu **Tools** → **Board** zvolte položku **"LaunchPad w/ cc3200 (80MHz)"**.
   - Dále připojte vývojový kit k počítači.
   - Pokud je kit správně rozpoznaný ve správci zařízení jako `COM` port, zbývá již jen zvolit v menu **Tools** → **Serial Port** patřičný `COM` port, který představuje vývojový kit.

A tímto je hotová veškerá instalace a konfigurace pro použití nástroje *Energia*, jako vývojového prostředí pro aplikace určené pro MCU *CC3200*. Vytváření nových projektů je totiž na rozdíl od *Code Composer Studia* otázkou několika sekund. V menu stačí vybrat **File** → **New** a nastavení pro konkrétní vývojový kit zůstává stejné.

Stavba projektu v nástroji *Energia* má jiné pojetí, než je tomu u nástroje *Code Composer Studio*. Při vytvoření nového projektu v *Energii* se automaticky vygenerují dvě hlavní funkce. Funkce `void setup()` a funkce `void loop()`. Tyto funkce jsou základním kamenem každého projektu *Energie*.

Funkce `void setup()` zajišťuje, jak již název napovídá, počáteční konfiguraci zařízení, inicializaci proměnných, inicializaci periférií, nastavení terminálového výstupu a dalších věcí. Tato funkce se spustí na začátku aplikace pouze jednou.

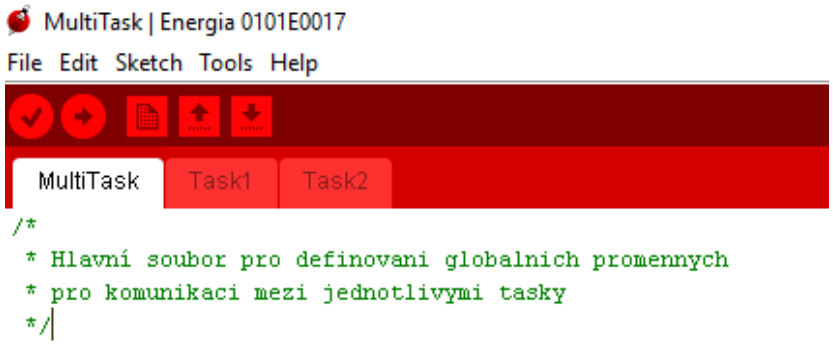
Kdežto funkce `void loop()` obsahuje kód aplikace, který se opakuje. Je to v podstatě `while()` smyčka. Veškerá logika aplikace má tedy místo právě v této funkci. Samozřejmě je možné vytvářet další pomocné funkce.

Takto vypadá nově vytvořený projekt v *Energii*:

```
void setup()
{
  // zde vlozte kod, ktery se provede jednou pri spusteni
}

void loop()
```

```
{
  // zde vlozte hlavni kod aplikace, ktery se opakuje
}
```

Tvůrci *Energie* představili další projekt jménem *Energia MT*[12], který přináší možnost, jak využít výhod RTOS. Autoři projektu v podstatě zakomponovali *TI-RTOS* od společnosti *Texas Instruments* do svého projektu a nabídli svým zákazníkům jednoduchý způsob jeho využití. Projekt je možné zatím využít společně s 6 různými zařízeními (včetně MCU *CC3200*) a autoři s každou další vydanou verzí přidávají podporu pro další zařízení. Vytvoření projektu využívající TI-RTOS je jednoduché. Každá záložka v *Energii* odpovídá jednomu tasku (viz obr. 2.10). Záložka `MultiTask` představuje hlavní soubor, který obsahuje globální proměnné pro komunikaci mezi tasky. Záložky `Task1` a `Task2` představují samotné tasky.



Obrázek 2.10: Vytvoření RTOS projektu v Energii

Názvy hlavních funkcí `setup()` a `loop()` se musí upravit tak, aby odpovídaly jednotlivým taskům. `Task1` bude tedy obsahovat následující definici výše uvedených funkcí:

```
void setupTask1() {
  //...
}

void loopTask1() {
  //...
}
```

A obdobně `Task2` bude obsahovat následující definice:

```
void setupTask2() {
  //...
}
```

```
void loopTask2() {
  //...
}
```

Energia nabízí také bohatou nabídku knihoven, které jsou již součástí nástroje a lze je jednoduše naimportovat prostřednictvím menu **Sketch** → **Import Library...**. Nabídka knihoven, které lze v projektu použít, se mění dle typu použitého zařízení. Zařízení lze změnit v menu **Tools** → **Board**. Knihovny poskytují jak základní funkcionalitu, např. pro jednoduché použití SPI rozhraní, tak i složitější, např. nástroj *Temboo* pro práci s internetovými službami. Dále je uveden seznam vybraných knihoven, které lze společně s MCU *CC3200* použít (kompletní seznam knihoven a jejich podrobnější popis lze nalézt na stránkách *Energie*[11]):

- **SPI** - slouží pro komunikaci pomocí SPI rozhraní.
- **Serial** - slouží pro komunikaci mezi PC a UART zařízením.
- **WiFi** - slouží pro ovládání Wi-Fi modulu. Knihovna obsahuje také třídy, které přemění zařízení ve Wi-Fi klienta nebo Wi-Fi server.
- **Wire** - slouží pro komunikaci pomocí I2C rozhraní.
- **aJson** - parsovací nástroj pro formát JSON[9].
- **Temboo** - knihovna, poskytující data z různých internetových služeb. Jde například o služby poskytující počasí (Yahoo! Weather), datová úložiště (Dropbox) nebo sociální sítě (Facebook). Více informací lze nalézt na stránkách *Temboo*[23].
- **BMA222** - knihovna pro akcelerometr, který se nachází na vývojovém kitu.
- **Adafruit_TMP006** - knihovna pro teplotní čidlo, které se nachází na vývojovém kitu.

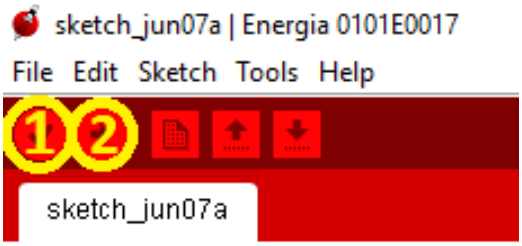
Na konci práce (v příloze D) je opět ukázka kódu tentokrát určeného pro nástroj *Energia*. Pokud je tedy nástroj správně zkonfigurován a vývojový kit taktéž, po vložení kódu do nového projektu a zkompilování je možné výslednou aplikaci nahrát do vývojového kitu. Tentokrát však výsledný program přepíše dříve nahranou aplikaci. I po resetování MCU tedy zůstává jako hlavní nově nahraný program.

Na obrázku 2.11 je vidět strohá nabídka tlačítek rozhraní nástroje *Energia*. Nejdůležitější jsou dvě čísly označená tlačítka:

[9] JavaScript Object Notation - formát pro výměnu dat

1 - slouží pro kontrolu a kompilaci kódu,

2 - slouží pro kompilaci kódu a následné nahrání aplikace do sériové FLASH paměti.



Obrázek 2.11: Rozhraní nástroje Energia

Bohužel samotný nástroj nepodporuje debugování aplikace. Existuje však řešení, které dokáže zpřístupnit debugování vašemu kódu psanému v *Energii*. Řešením je importování projektu *Energie* do nástroje *Code Composer Studio*. Jak už jsem dříve podotkl, firma *Texas Instruments* do svého nástroje přidala možnost importování projektů psaných právě v *Energii*. Tímto způsobem zpřístupníte klasické debugovací nástroje, na které jste zvyklí. Toto řešení není zrovna ideální, jelikož požaduje instalaci obou nástrojů současně. Na druhou stranu je skvělé, že existuje alespoň nějaké řešení.

Srovnání kódů pro oba nástroje při stejné finální funkcionalitě

Na konci práce jsou v přílohách C a D zdrojové kódy pro nástroje *Energia* a *Code Composer Studio*. Výsledná funkcionalita po nasazení aplikace je shodná. Tyto příklady kódů mají především demonstrovat způsob vývoje v jednotlivých nástrojích. Způsob, kterým se programuje v nástroji *Energia*, je velice intuitivní a uživatelsky přívětivý. Dokazuje to už ten fakt, že zdrojový kód *Energie* má 30 řádků, kdežto kód nástroje *Code Composer Studio* má řádků 70 při stejné funkcionalitě. Mezi demonstrované funkce obou kódů patří:

- inicializace MCU,
- konfigurace pinů,
- inicializace terminálového výstupu,
- manipulace s piny (LED dioda je v podstatě připojená k pinu MCU),

- výpis na terminálový výstup.

Oba zdrojové kódy obsahují funkci `void blickLed(unsigned int led)`, která podle předaného parametru rozsvítí příslušnou LED diodu vývojového kitu, poté počká 300 ms, LED diodu vypne a opět počká 300 ms. Oba zdrojové kódy samozřejmě využívají knihovních funkcí k dosažení výsledné funkcionality. Další srovnatelnou funkcí je funkce `void setup()`, která v obou případech zkonfiguruje použité piny a inicializuje terminálový výstup. Rozdíl je v tom, že se tato funkce v *Energii* volá automaticky, kdežto v druhém nástroji se musí volat manuálně z funkce `main()`. Funkce se samozřejmě liší konkrétními prostředky, kterými dosahují výsledné funkcionality. Verze pro *Code Composer studio* obsahuje navíc ještě funkci `printMessage()`, která se stará o výpis zprávy na terminálový výstup. V *Energii* je funkce pro výpis na terminál již obsažena v základním programovém vybavení.

Výsledný výběr nástroje pro vývoj

Přestože jsem nástroj *Energia* v tomto přehledu nástrojů ve většině případů chválil a vyzdvihoval jeho klady, pro vývoj jsem zvolil raději nástroj *Code Composer Studio*.

Výběr je odůvodnitelný tím, že jde o oficiální nástroj společnosti *Texas Instruments* pro vývoj na tomto MCU. Vycházejí aktualizované `SDK`, stejně jako vycházejí aktualizace pro samotný nástroj.

Další výhodou je zabudovaný debugovací systém a obecně bohatší uživatelské prostředí, které je zatím v *Energii* velmi strohé. *Energia* nenabízí v podstatě ani tak základní prostředek, jako je automatické doplňovaní při psaní kódu, na které je většina vývojářů zvyklá z moderních vývojových prostředí.

2.2 Teplotní čidla

V zadání práce je požadavek na přesné měření teploty. Bylo tedy potřeba vybrat takové teplotní čidlo, které by vyhovovalo zadání a zároveň by bylo možné použít čidlo společně s MCU *CC3200*. Pro tento účel byla vyvinuta přídavná destička, do které je připojeno externí teplotní čidlo. Destička je v podstatě *BoosterPack* modulem (více kapitola 2.1.4). Obsahuje A/D převodník *ADS1247*, pomocné obvody (filtry) pro kompenzaci chyb při měření, čidlo pro měření tlaku, napěťový převodník DC/DC a další. Podrobný seznam a popis všech použitých prvků je uveden v kapitole 3.1.1.

2.2.1 Základní popis

V průmyslu se běžně používají čtyři skupiny teplotních čidel:

- termočlánky,
- termistory,
- RTD senzory,
- IC[10] teplotní senzory.

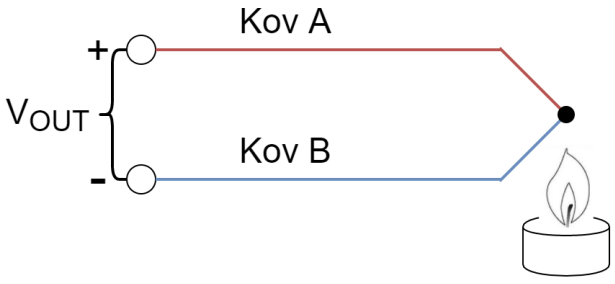
Každá skupina má samozřejmě své klady, ale i zápory. Stejně tak se liší místem, kde mohou být aplikovány. Obecné parametry, které lze použít při srovnání jednotlivých skupin jsou: cena čidla, rozsah měření, rychlost odezvy při změně teploty, přesnost měření, lineárnost měření a konstrukce čidla.

2.2.2 Typy čidel

Termočlánky

První skupinou jsou termočlánky. Termočlánek se skládá ze dvou různorodých kovů spojených v jednom bodě. Jednotlivé skupiny kovů jsou předem definované a jsou popsány níže. Výběrem různých skupin lze dosáhnout různých rozsahů měření, různých oblastí působnosti a různých přesností měření.

Na obrázku 2.12 je obecné schéma termočlánku. Termočlánek produkuje proud, a tím vzniká na koncích vodičů rozdílové napětí V_{OUT}, které se úměrně mění podle okolní teploty.



Obrázek 2.12: Schéma termočlánku

Jednotlivé typy termočlánků podle použitých kovů a jejich vlastnosti jsou zaneseny v tabulce 2.1.

[10]IC = integrated circuit (integrovaný obvod)

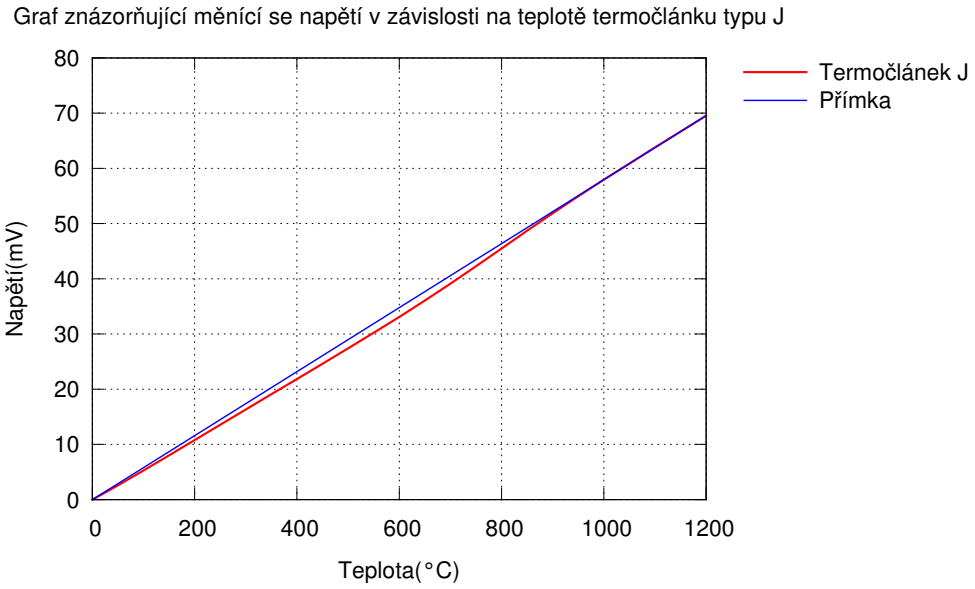
Typ	Typ E	Type K	Type J	Typ R	Typ T
Kov A	nikl(10%)-chrom	nikl(10%)-chrom	železo	platina (13%)-rhodium	měď
Kov B	konstantan	nikl(5%)-hliník-křemík	konstantan	platina	konstantan
Změna napětí na 1 °C	62 µV	40 µV	62 µV	7 µV	40 µV
Teplotní rozsah ve °C	-100 až 1000	0 až 1370	0 až 760	0 až 1450	-160 až 400
Místa použití	kryogenika	obecné účely	důraz na citlivost	kalibrační účely, odolnost vůči oxidaci a korozi	diferenciální měření

Tabulka 2.1: Jednotlivé typy termočlánků podle použitých kovů

Typický průběh změny generovaného napětí v závislosti na měnící se teplotě znázorňuje graf na obrázku 2.13. Hodnoty byly převzaty z webových stránek společnosti *National Instruments*[24].

Výhody termočlánku:

+ velký teplotní rozsah měření od $-200\,°C$ do $2000\,°C$ (konkrétní hodnoty záleží na použité skupině kovů),
+ senzor nepotřebuje žádné externí napájení, jelikož samotný senzor generuje napětí,
+ rychlá odezva při změně teploty,
+ jednoduchá a odolná konstrukce,
+ je levný,
+ existují různé skupiny termočlánku (větší možnost výběru pro konkrétní použití).



Obrázek 2.13: Graf měnícího se napětí v závislosti na teplotě pro termočlánek typu J (železo/konstantan)

Nevýhody termočlánku:

- generované napětí vůči teplotě není lineární,
- generované napětí je poměrné nízké v řádech mV,
- nepřesný: $\pm 2\,°C$,
- nízká citlivost,
- napětí na koncích vodičů je úměrné rozdílu teplot (ne absolutní teplotě). Proto je nutné "studený" konec udržovat na známé teplotě (to bývá problém v případě přesného měření).

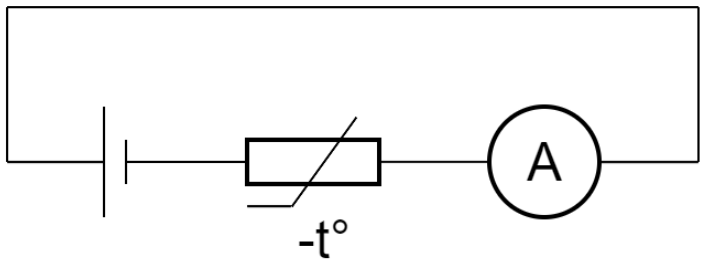
Termistory

Další skupinou jsou termistory. Na rozdíl od předchozí skupiny termistory negenerují žádné napětí, ale mění odpor v závislosti na měnící se teplotě. Materiál, ze kterého jsou termistory vyrobeny, je především keramika nebo polymer. Existují dva základní typy termistorů, a to:

- **NTC termistor** má negativní teplotní koeficient. Při zahřívání NTC termistoru jeho odpor klesá.

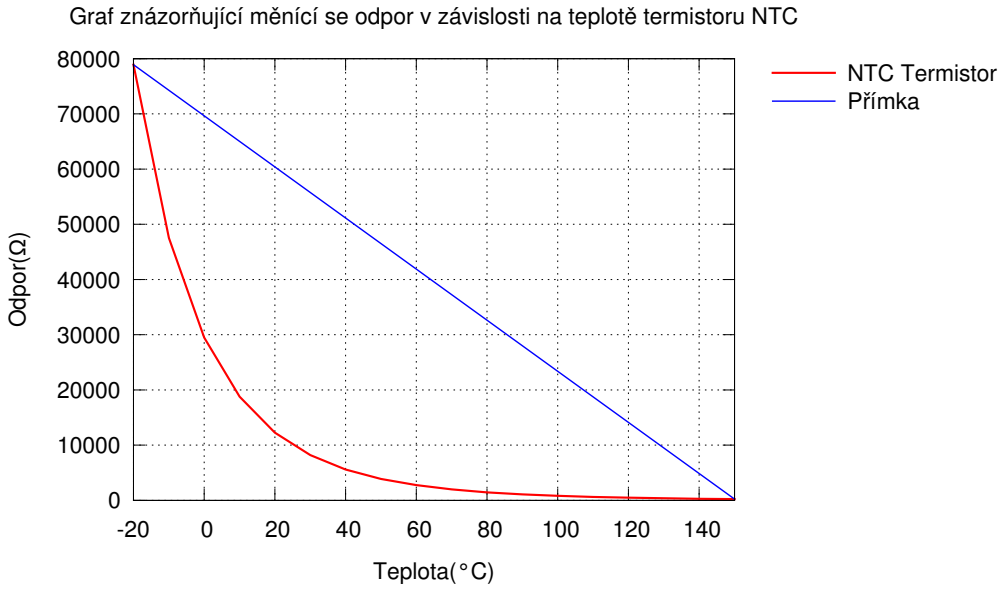
- **PTC termistor** má pozitivní teplotní koeficient. Při zahřívání PTC termistoru jeho odpor roste.

Na obrázku 2.14 je nejjednodušší zapojení takového termistoru do obvodu. Obvod obsahuje zdroj konstantního napětí, dále obsahuje samotný termistor a poslední komponenta obvodu je ampérmetr. V obvodu tedy měříme procházející proud. Poté dle jednoduché rovnice $R = U/I$, kde U je konstantní napětí a I je měřený proud, zjistíme odpor termistoru.



Obrázek 2.14: Jednoduché schéma zapojení termistoru do obvodu

Typický průběh změny odporu v závislosti na měnící se teplotě znázorňuje graf na obrázku 2.15. Hodnoty představují skutečný průběh pro zařízení s označením *TH-10-44006*[25] od společnosti *Omega*.



Obrázek 2.15: Graf odporu v závislosti na teplotě pro termistor TH-10-44006

Výhody termistoru:

+ je levný,
+ je k dostání v různých rozsazích odporů,
+ vysoká přesnost ($< \pm 0.2\,°C$),
+ volba typu termistoru (NTC/PTC) pro konkrétní použití,
+ vysoká citlivost na změny teploty.

Nevýhody termistoru:

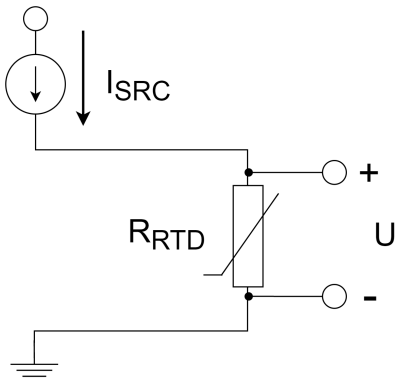
− omezený teplotní rozsah měření od $-100\,°C$ do $150\,°C$,
− měřený odpor je vůči teplotě silně nelineární,
− je nutný externí zdroj napětí/proudu,
− je časově nestálý, tzn. jeho vlastnosti se v čase mění (částečně možno kompenzovat umělým stárnutím pro ustálení vlastností),
− při měření se zahřívá.

RTD

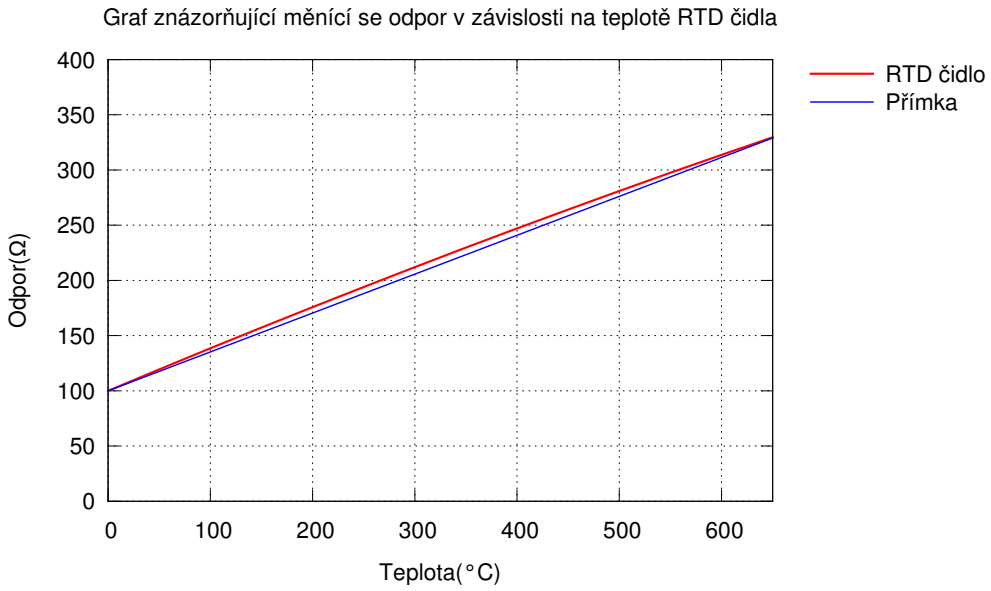
Další skupinou teplotních senzorů jsou RTD[11] senzory. Tato čidla mění opět odpor v závislosti na měnící se teplotě. Čidla tohoto typu jsou vyráběna především z platiny. Výslednou teplotu lze zjistit z tabulek pro převod naměřeného odporu na skutečnou teplotu. Tabulky jsou na internetu volně dostupné a liší se podle třídy RTD čidla.

Nejznámějším představitelem této skupiny je teplotní čidlo *Pt100*. Je charakteristické především nominální hodnotou odporu $100\,\Omega$ při $0\,°C$. Jde o nejpoužívanější senzor v průmyslové sféře a pomalu nahrazuje termočlánky v oblastech, kde je rozsah měřených teplot do $600\,°C$. Obdobou *Pt100* je čidlo *Pt1000*, které má nominální hodnotu odporu $1\,k\Omega$ při $0\,°C$.

Na obrázku 2.16 je obvod, ve kterém je zapojené RTD čidlo. Jde o zjednodušené zapojení. Obvod obsahuje konstantní zdroj proudu, dále obsahuje samotné RTD čidlo a svorky pro měření napětí v obvodu. V obvodu lze poté změřit napětí na svorkách U a podle jednoduchého vztahu $R_{RTD} = U/I_{SRC}$, kde I_{SRC} je konstantní předem známý proud, dopočítat výsledný odpor čidla.



Obrázek 2.16: Jednoduché schéma zapojení RTD čidla do obvodu



Obrázek 2.17: Graf odporu v závislosti na teplotě pro RTD čidlo Pt100

Do grafu na obrázku 2.17 je zanesená závislost měnícího se odporu teplotního čidla Pt100 na okolní teplotě.

RTD čidla se vyrábějí v různých třídách přesnosti měření. Obecně existují 3 základní třídy *A*, *B* a *C*. Podle přání zákazníka lze ale vyrobit i individuální RTD čidlo, které bude ještě přesnější. Tabulka 2.2 představuje jednotlivé třídy a jejich charakteristické přesnosti.

Výhody RTD čidla:

\+ vysoká přesnost (při použití individuální třídy i $< \pm\ 0.1\,°C$),

[11]RTD = Resistance Temperature Detector

Třída	± přesnost [°C]	± přesnost při 25°C [°C]	± přesnost při 200°C [°C]
A	$0.15 + 0.002 \cdot \|t\|$	0.2	0.55
B	$0.3 + 0.005 \cdot \|t\|$	0.425	1.3
C	$0.6 + 0.01 \cdot \|t\|$	0.85	2.6
Individuální	$0.03 + 0.0005 \cdot \|t\|$	0.0425	0.13

$|t|$ je numerická hodnota teploty ve °C

Tabulka 2.2: Třídy RTD čidel a jejich přesnosti

+ velký teplotní rozsah měření od $-200\,°\mathrm{C}$ do $600\,°\mathrm{C}$,
+ mírně nelineární průběh měřeného odporu vůči teplotě,
+ stabilní měření odporu v čase.

Nevýhody RTD čidla:

− je drahé,
− je nutný externí zdroj proudu,
− při měření se zahřívá,
− pomalu reaguje na změnu teploty,
− poměrně malá změna odporu ($0.385\,\Omega$) na $1\,°\mathrm{C}$ (platí pro *Pt100*).

IC teplotní senzory

Poslední skupinou jsou IC[12] teplotní senzory. Jde o typ zařízení, který generuje elektrický proud úměrný absolutní teplotě. Většinou jde o malý prvek, který nachází uplatnění především v mobilní elektronice. Typické oblasti použití jsou:

- mobilní zařízení (mobilní telefon, PDA, ...),
- počítačová odvětví (teplotní senzory v jednotlivých částech PC: CPU, GPU, ...),
- obecně plošné spoje pro monitorování jejich teplot.

Výhody IC teplotního senzoru:

[12]IC = Integrated Circuit (Integrovaný Obvod)

+ nejlineárnější průběh měřené veličiny vůči teplotě ze všech typů čidel,

+ je levný.

Nevýhody IC teplotního senzoru:

– omezený teplotní rozsah měření (<250 °C),

– nepřesný: ±2 °C,

– je nutný externí zdroj energie,

– pomalu reaguje na změnu teploty,

– při měření se zahřívá,

– limitovaná konfigurace pro vývojáře (senzor již obsahuje obvody včetně zkonfigurovaného A/D převodníku).

2.2.3 Výběr teplotního čidla

Z výše uvedených skupin teplotních čidel bylo zapotřebí vybrat jednu, která se pro tento úkol hodí nejlépe. V zadání je kladen důraz především na přesnost měření a vhodný teplotní rozsah, použitelný v běžných podmínkách v našem klimatickém prostředí. Z tohoto důvodu nejsou vhodné termočlánky a IC teplotní senzory, jelikož nejsou dostatečně přesné. Termistor je časově nestálý (jeho vlastnosti se v čase mění) a měřený odpor vůči teplotě silně nelineární. Tento typ čidla tedy také není vhodný. Pro účel přesného měření teploty byla nakonec vybrána skupina RTD čidel.

Dále bylo potřeba vybrat konkrétní typ čidla z této skupiny. Nejrozšířenějším a nejdostupnějším představitelem RTD čidel je čidlo s označením *Pt100*. Čidlo splňuje obě předsevzatá kritéria a je dostupné u místních prodejců elektronických součástek. Čidlo bylo zakoupeno u společnosti *Farnell*[20].

2.2.4 Platinové čidlo Pt100

Pro přesné měření teploty bylo tedy vybráno čidlo *Pt100*. Jde o čidlo vyrobené společností *Innovative Sensor Technology*[17] s přesným označením *P0K1.202.3FW.B.007*. Charakteristické znaky konkrétního čidla, se kterým probíhalo měření jsou:

- Základní hodnota odporu při 0 °C je rovna 100 Ω.
- Patří do třídy *B* (viz kapitola 2.2.2).

- Doporučená hodnota proudu, kterou lze aplikovat z obvodu se zapojeným čidlem je $1mA$.
- Čas, za který se čidlo přizpůsobí okolní teplotě je:
  - **0.16 s** - při aplikaci ve vodě a současném proudění vody $0.4\,\mathrm{m\,s^{-1}}$.
  - **4.9 s** - při aplikaci na vzduchu a současném proudění vzduchu $1\,\mathrm{m\,s^{-1}}$.
- Operační rozsah teplot je od $-200\,^{\circ}\mathrm{C}$ do $300\,^{\circ}\mathrm{C}$.
- Rozměry: $2 \times 2 \times 1.3$mm (délka x šířka x hloubka).
- Kontakty jsou z pozlaceného niklu o rozměrech 0.2×0.4mm a délce 7 mm.
- Patří do skupiny čidel s charakteristickou křivkou 3850 ppm/K.

Poslední bod vlastností představuje charakteristickou křivku 3850 ppm/K. Nejde tak úplně o zažitý pojem, dále bude tedy vysvětlen a ukázán na konkrétním příkladu. Zkratka *ppm* znamená *"parts per milion"*. 3850 ppm/K tedy znamená, že odpor se změní o $\frac{3850}{1000000}$ při změně teploty o 1 K. Rovnice 2.1 představuje, jak takový výpočet probíhá. $\mathbf{R_T}$ představuje odpor čidla. $\mathbf{R_0}$ je odpor čidla při $0\,^{\circ}\mathrm{C}$, což je, jak už bylo psáno v charakteristice, $100\,\Omega$. Proměnná $\delta\mathbf{t}$ představuje změnu teploty o určitý počet stupňů.

$$\mathbf{R_T} = \mathbf{R_0}[\mathbf{1} + (\delta\mathbf{t}\frac{\mathbf{3850}}{\mathbf{1000000}})]) \tag{2.1}$$

Výpočet odporu pro teplotu například $10°C$ by poté vypadal takto:

$$R_T = 100[1 + (10 \cdot \frac{3850}{1000000})]\Omega$$
$$R_T = 103,85\Omega$$

Tento výpočet se však v praxi nedá použít, jelikož závislost odporu na teplotě čidla *Pt100* není lineární. Proto se tato charakteristika používá v podstatě jen pro kategorizaci těchto teplotních čidel typu *Pt100*. Již tento výsledek se liší oproti tabulkovým hodnotám o $0.05\,\Omega$, a to byl výpočet proveden pouze pro teplotní rozdíl $10\,^{\circ}\mathrm{C}$.

V praxi se proto používá následující aproximační funkce daná rovnicí 2.2. Rovnice zavádí základní vztah mezi teplotou a odporem *Pt100*. **Tuto rovnici lze ovšem použít pouze pro teploty vyšší nebo rovno $0\,^{\circ}\mathrm{C}$.** Proměnná **t** představuje měřenou teplotu, tato proměnná nás tedy zajímá

nejvíce. **A** a **B** jsou konstanty, které jsou specifické pro konkrétní typ čidla. Pro tento konkrétní typ čidla *Pt100* jsou hodnoty konstant následující:

$$A = 3.9083 \cdot 10^{-3}$$
$$B = 5.7750 \cdot 10^{-7}$$

$$\mathbf{R_T = R_0(1 + At + Bt^2)} \tag{2.2}$$

Pro vyjádření samotné proměnné **t** z rovnice 2.2 je potřeba nejdříve rovnici upravit. Z upravené rovnice je zřejmé, že se jedná o kvadratickou rovnici a lze tedy vypočítat kořen rovnice, tedy požadovanou proměnnou **t**.

$$R_T = R_0 + R_0At + R_0Bt^2$$
$$0 = R_0Bt^2 + R_0At + R_0 - R_T$$

Nejdříve je potřeba spočítat diskriminant.

$$D = b^2 - 4ac$$
$$D = R_0^2A^2 - 4[R_0B(R_0 - R_T)]$$
$$D = R_0^2A^2 - 4(R_0^2B - R_0BR_T)$$
$$D = R_0^2A^2 - 4R_0^2B + 4R_0R_TB$$

Posledním krokem je dosazení do rovnice a výpočet kořenu kvadratické rovnice. Rovnice 2.3 je finální rovnice pro výpočet teploty podle změřeného odporu čidla $\mathbf{R_T}$.

$$t = \frac{-b + \sqrt{D}}{2a}$$
$$t = \frac{-R_0A + \sqrt{R_0^2A^2 - 4R_0^2B + 4R_0R_TB}}{2R_0B}$$
$$t = \frac{-R_0A + \sqrt{R_0^2(A^2 - 4B + 4B\frac{R_T}{R_0})}}{2R_0B}$$
$$t = \frac{-R_0A + R_0\sqrt{A^2 - 4B(1 - \frac{R_T}{R_0})}}{2R_0B}$$
$$t = \frac{R_0(-A + \sqrt{A^2 - 4B(1 - \frac{R_T}{R_0})})}{2R_0B}$$

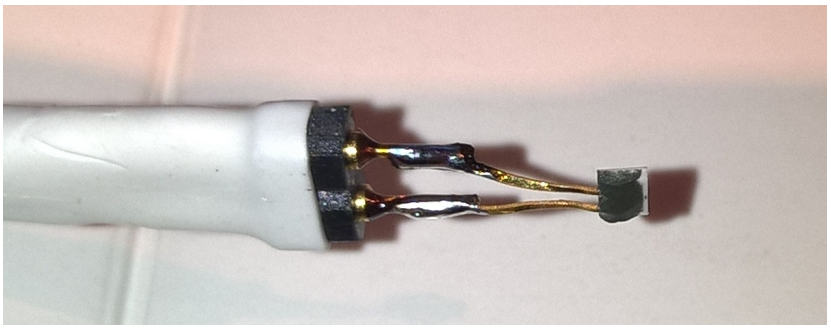
$$\mathbf{t} = \frac{-\mathbf{A} + \sqrt{\mathbf{A^2} - \mathbf{4B}(\mathbf{1} - \frac{\mathbf{R_T}}{\mathbf{R_0}})}}{\mathbf{2B}} \tag{2.3}$$

Z rovnice 2.3 je zřejmé, že nebude úplně praktické pro každé měření teploty počítat takto složitý výraz. Jelikož jsou $\mathbf{A}$, $\mathbf{B}$ i $\mathbf{R_0}$ konstanty, je možné si jednotlivé hodnoty vyčíslit dopředu a uložit. Ostatně již existují tabulky pro čidla *Pt100*, které tento proces aplikují. Jedním z nejjednodušších a výpočetně nenáročných principů převodu odporu čidla na teplotu je postup, při kterém jsou hodnoty vypočteny dopředu (dle konkrétních požadavků) a uloženy do některé z datových struktur podobných *HashMapě*. Klíčem v této mapě bude samozřejmě změřený odpor čidla a hodnotou požadovaná teplota.

Rovnice 2.2 byla definovaná pouze pro výpočet teploty vyšší než $0°C$. Pro tepoty pod touto hranicí je definována rovnice 2.4. Na první pohled je zřejmé, že se jedná o polynomiální rovnici třetího řádu. Pro vyjádření proměnné $\mathbf{t}$ by bylo nutné provést aproximaci a výsledkem by opět byla polynomiální rovnice. Čím vyššího řádu by byla výsledná aproximovaná rovnice, tím menší by byla výsledná chyba výpočtu. V předchozím odstavci byla již vysvětlena a odůvodněna nevhodnost použití takto složitých výpočtů pro převod hodnot mezi teplotou a odporem. Rovnice 2.2 ani 2.4 pro výpočet použity nebudou, a místo toho bude použit princip popsaný v předchozím odstavci (zanesení hodnot do *HashMapy*).

$$\mathbf{R_T} = \mathbf{R_0}[\mathbf{1} + \mathbf{At} + \mathbf{Bt^2} + \mathbf{C}(\mathbf{t} - \mathbf{100})\mathbf{t^3}] \tag{2.4}$$

Na obrázku 2.18 je čidlo s označením *P0K1.202.3FW.B.007* od společnosti *Innovative Sensor Technology*[17]. S tímto konkrétním čidlem byla prováděna veškerá měření. Jeho popis je uveden na začátku této kapitoly.

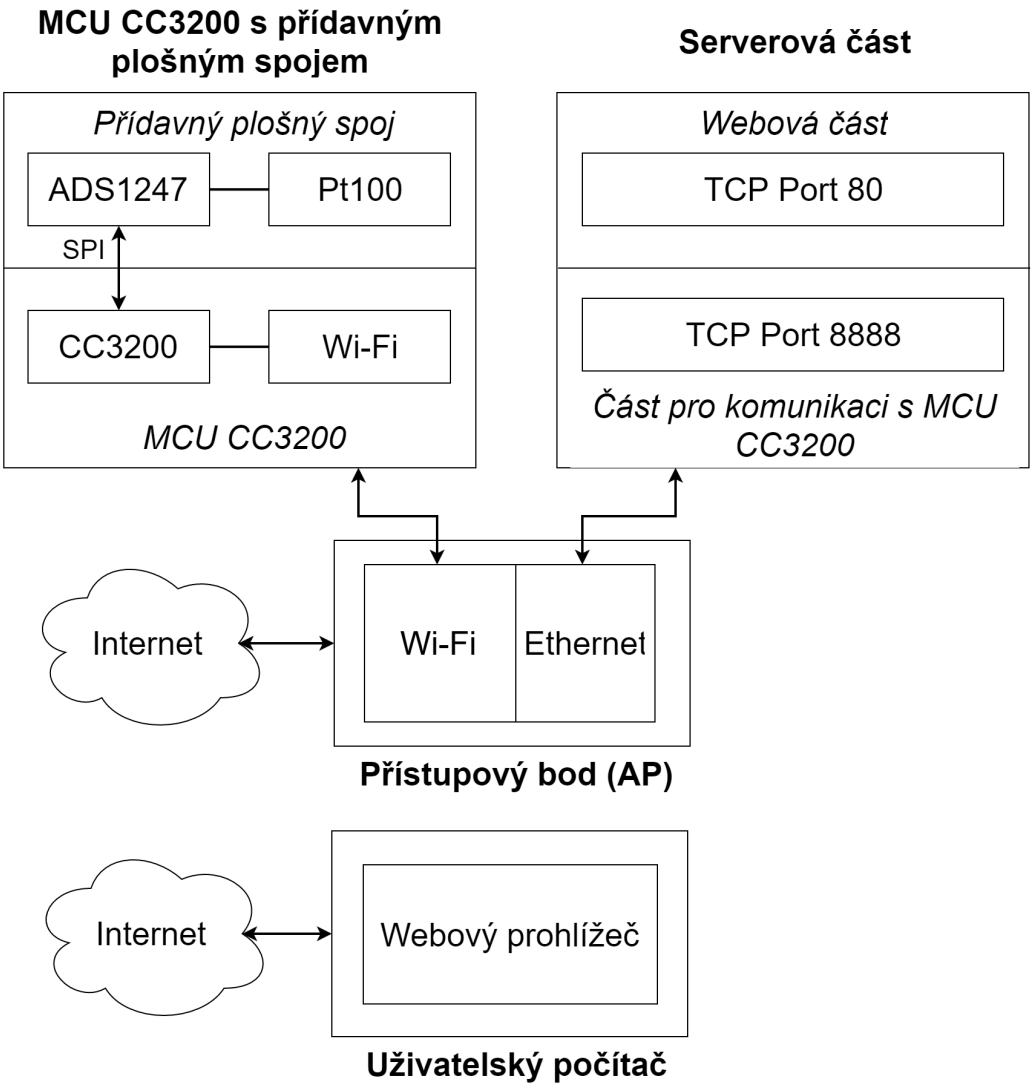


Obrázek 2.18: Fotografie konkrétního čidla Pt100, se kterým probíhaly měření

3 Praktická část

Praktická část práce se zabývá tvorbou přídavného plošného spoje pro vývojový kit a samotnou implementací programového vybavení jak pro MCU *CC3200*, tak pro serverovou část, která bude s MCU komunikovat, zpracovávat data a prezentovat data pomocí jednoduchého webového serveru.

Zjednodušený modelový případ komunikace mezi MCU *CC3200*, serverovou částí a uživatelským počítačem je zachycen na obrázku 3.1.



Obrázek 3.1: Model komunikace

Na přídavném plošném spoji je zapojen teplotní senzor *Pt100*. Hodnoty jsou ze senzoru čteny pomocí A/D převodníku *ADS1247*. MCU s převodníkem komunikuje přes rozhraní *SPI* a čte naměřené hodnoty.

MCU je připojen k AP pomocí Wi-Fi spojení a server pomocí Ethernetového připojení. MCU vytváří TCP soket s cílovou IP adresou serverové části a portem 8888[1] a odesílá data na server. Server naslouchá na portu 8888 a vytváří příchozí spojení mezi MCU a serverem. Dále přijme data, nově vytvořené spojení zavře a data zpracuje. Mezitím si uživatel ve webovém prohlížeči zadá IP adresu serveru pro zobrazení webové stránky se statistikami teplot. Server naslouchá také na portu 80 a vytváří příchozí spojení mezi uživatelským počítačem a serverem. Server přijme HTTP požadavek, zpracuje ho, odešle požadovaná data zpět uživateli a uzavírá spojení.

Uživatelský počítač a server samozřejmě nemusí být ve stejné síti. Pokud je patřičně nakonfigurován síťový prvek, ke kterému je server připojen, a tím pádem je server viditelný zvenčí, je možné na webový server přistupovat i vzdáleně z jiné sítě.

Konkrétní způsoby komunikace a připojení jednotlivých prvků budou detailněji rozebrány a vysvětleny v dalších kapitolách práce.

3.1 Tvorba přídavného plošného spoje

Prvním krokem práce byla právě tvorba přídavného plošného spoje, na kterém bude zapojeno teplotní čidlo. Z čidla samozřejmě nelze rovnou číst hodnoty přes nějaké rozhraní, jelikož jediné co čidlo umí, je to, že mění odpor v závislosti na okolní teplotě. Bylo tedy potřeba nalézt zařízení, které by dokázalo nějakým způsobem změřit hodnotu odporu čidla, dále je převést do digitální podoby, a tato digitální data poté poslat přes datové rozhraní do MCU *CC3200*. Pro tyto účely se nejlépe hodí A/D[2] převodníky. Samotný MCU *CC3200* obsahuje tyto převodníky, ty však nedosahují takové přesnosti, aby mohly být použity pro tento konkrétní účel měření. Proto byl vybrán převodník *ADS1247*, který je primárně určen pro přesné měření teploty pomocí termočlánků nebo RTD platinových čidel, díky jeho vysokému rozlišení 24 bitů. Převodník je podrobněji popsán v kapitole 3.1.1.

Dále bylo potřeba nějakým způsobem připojit plošný spoj k samotnému MCU. Pro tyto účely slouží již dříve rozebíraný BoosterPack konektor v kapitole 2.1.4.

Základní seznam součástek na plošném spoji byl tedy následující:

[1] Čísla portů byla vybrána náhodně.

[2] Analogově digitální

- teplotní senzor Pt100,
- A/D převodník,
- BoosterPack konektor.

Po domluvě a s pomocí vedoucího práce byly do schématu navíc ještě přidány následující součástky:

- senzor pro měření atmosférického tlaku a nadmořské výšky,
- napěťový regulátor,
- doky pro 2 AAA baterie,
- časovač pro spínání napětí.

Prvním přidaným zařízením je senzor pro měření atmosférického tlaku a nadmořské výšky. Bylo vybráno konkrétní zařízení, které má již implementovanou schopnost komunikace přes dvě standardní rozhraní SPI a $I2C$.

Druhým zařízením je napěťový regulátor, který slouží pro případ, kdy je výsledný modul napájen z přiložených baterií. Regulátor převádí vstupní napětí, které je dáno součtem napětí dvou AAA baterií, na výstupní napětí, které slouží pro napájení veškerých komponent plošného spoje včetně MCU *CC3200*.

Posledním zařízením, které bylo přidáno do schématu, je časovač pro spínání napětí. Jde o zařízení, které lze konfigurovat pomocí připojených odporů. Hodnoty odporů určí časový interval, po kterém se bude časovač budit. Po probuzení časovač sepne napájecí napětí ostatních obvodů včetně MCU. Tím se výsledný modul zapne a začne pracovat, jako kdyby byl zapnut manuálně. Poté, co MCU ukončí svoji činnost, nastaví patřičný výstupní pin, který je spojen právě s časovačem. Tímto pinem MCU sdělí, že se časovač může uspat a čekat další časový interval. Tento princip spínání napětí v podstatě simuluje hibernaci MCU. V tomto případě se lze tedy rozhodnout, který ze způsobů pro spínání MCU použít. Nejrozumnějším důvodem pro výběr je spotřeba energie jednotlivých řešení v období klidu.

3.1.1 Použité součástky

P0K1.202.3FW.B.007

Jde o konkrétní typ platinového čidla, které je použito ve výsledném modulu. Veškeré charakteristiky a vlastnosti byly již popsány v kapitole 2.2.4.

ADS1247

Jde o A/D převodník. Tento konkrétní typ s označením *ADS1247*[1] je od společnosti *Texas Instruments*. Jedná se o analogově-digitální převodník typu delta-sigma. Typickými vlastnostmi tohoto převodníku jsou:

- vysoké rozlišení - 24 bitů,
- programovatelný zesilovač vstupu (PGA), možné hodnoty:
  - 1, 2, 4, 8, 16, 32, 64, 128,
- obsahuje interní oscilátor,
- zdroj interního referenčního napětí o hodnotě 2.048 V,
  - zdroj je schopný generovat proud o hodnotě až 10 mA,
- programovatelná rychlost vzorkování dat až $2000 SPS$[3],
- analogový multiplexor se čtyřmi nezávisle volitelnými vstupy,
- dvojice programovatelných proudových zdrojů,
  - programovatelné hodnoty proudu jsou: 50 µA, 100 µA, 250 µA, 500 µA, 750 µA, 1 mA, 1.5 mA,
- čtveřice programovatelných obecně použitelných I/O pinů,
- interní teplotní senzor,
- *SPI* rozhraní,
- automatická kalibrace,
- manuální kalibrace pomocí *OFC* a *FSC* registru,
  - $Výstup = (Vstup - OFC) \cdot \frac{FSC}{0x400000}$,
- operační teplota od −40 °C do 125 °C,
- typické oblasti použití jsou:
  - měření teplot pomocí RTD čidel/termočlánků/termistorů,
  - měření tlaku,
  - měření průtoků,

[3] SPS = Samples Per Second (vzorků za sekundu)

– průmyslová automatizace.

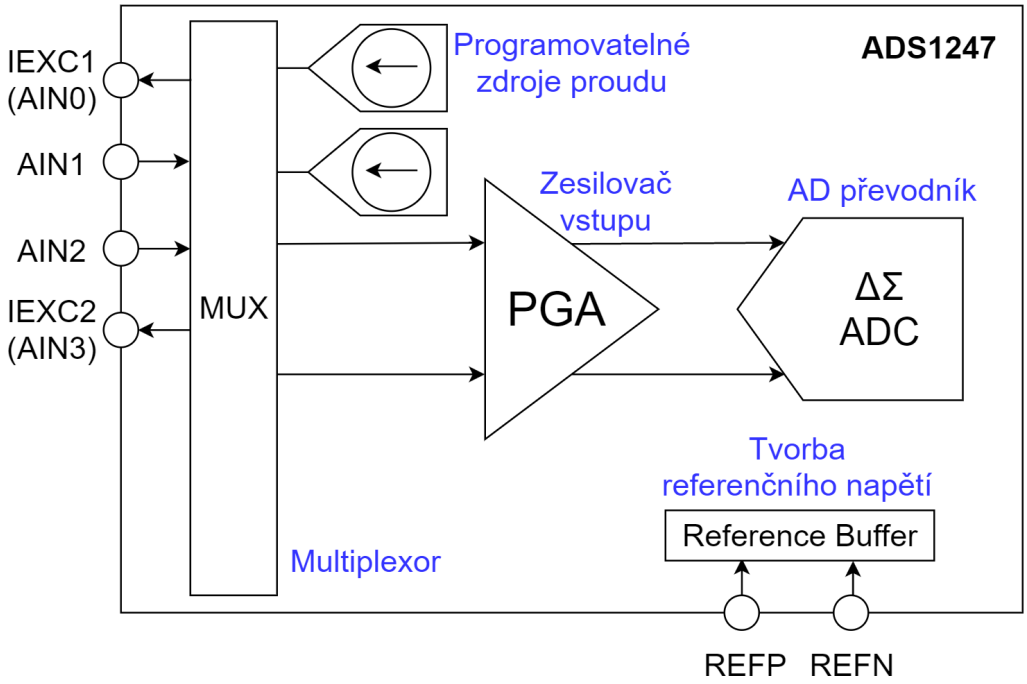
AD převodník je důležitým prvkem v celkovém návrhu výsledného modulu. Bez A/D převodníku by nebylo možné měřit odpor teplotního senzoru *Pt100*. Konkrétní konfigurace a výsledná funkcionalita převodníku závisí na hodnotách konfiguračních registrů, kterých je celkem 15. V tabulce 3.1 je uvedeno konkrétní nastavení konfiguračních registrů včetně názvu, adresy a hodnoty.

Název registru	Adresa registru	Hodnota registru
MUX0	0x00	0x0A
VBIAS	0x01	0x00
MUX1	0x02	0x20
SYS0	0x03	0x22
OFC0	0x04	xx
OFC1	0x05	xx
OFC2	0x06	xx
FSC0	0x07	xx
FSC1	0x08	xx
FSC2	0x09	xx
IDAC0	0x0A	0x96
IDAC1	0x0B	0x03
GPIOCFG	0x0C	0x00
GPIODIR	0x0D	0x00
GPIODAT	0x0E	0x00

Tabulka 3.1: Nastavení konfiguračních registrů převodníku ADS1247

Na obrázku 3.2 je zaneseno zjednodušené schéma převodníku *ADS1247* s nedůležitějšími vstupy/výstupy pro samotné měření. Dále je uvedeno, jak jednotlivé registry ovlivňují výsledné chování A/D převodníku (při nastavených hodnotách uvedených v tabulce 3.1):

- **MUX0:** nastaví vstupní pin `AIN1` jako pozitivní vstup pro měření a `AIN2` jako negativní vstup A/D převodníku.
- **VBIAS:** nastavení konstantního napětí BIAS, vypnuté (konstantní napětí v tomto případě není potřeba, jelikož právě napětí je proměnná, která se mění a díky tomu lze dopočítat hodnotu odporu teplotního senzoru).



Obrázek 3.2: Zjednodušené schéma A/D převodníku ADS1247

- **MUX1:** interní referenční napětí je zapnuté a zdrojem jsou piny `REFP` (pozitivní) a `REFN` (negativní).
- **SYS0:** programovatelný zesilovač vstupu (*PGA*) je nastaven na hodnotu 4 a rychlost vzorkování dat je nastaveno na hodnotu 20 vzorků za sekundu.
- **OFC, FSC:** tyto registry lze použít pro kalibraci výstupu převodníku.
- **IDAC0:** nastaví hodnotu proudu generovanou proudovými zdroji na 1 mA.
- **IDAC1:** výstupním pinem prvního proudového zdroje je `IEXC1(AIN0)` a výstupním pinem druhého proudového zdroje je `IEXC2(AIN3)`.
- **GPIOCFG:** všechny obecně použitelné I/O piny jsou vypnuté.
- **GPIODIR:** nastaví všechny obecně použitelné I/O piny jako výstupní.
- **GPIODAT:** nastaví všechny obecně použitelné I/O piny do nízké úrovně.

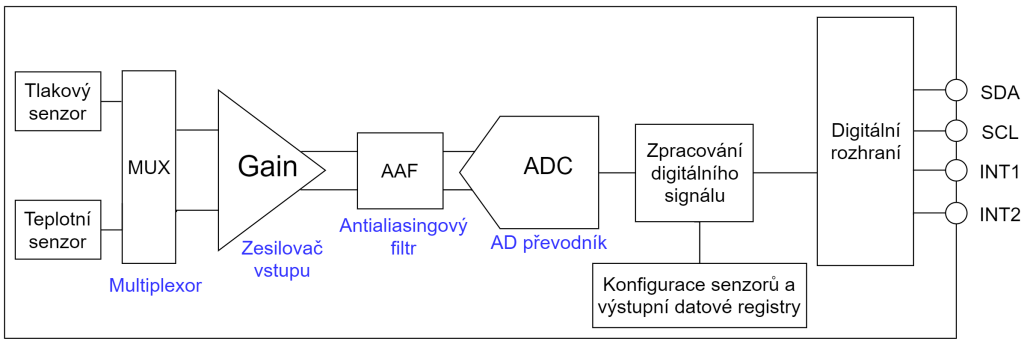
MPL3115A2LGA8

Jedná se o multifunkční čidlo pro měření atmosférického tlaku, nadmořské výšky a také teploty. Teplotu samozřejmě neměří s dostatečnou přesností a pro účely této práce je nepoužitelné. Čidlo bylo přidáno do výsledného modulu pro jeho ostatní dvě funkce, které společně s teplotním čidlem *Pt100* již nabízí rozumné řešení pro vytvoření například menší meteorologické stanice.

Technické specifikace čidla:

- Rozlišení se liší dle typu měření:
  - atmosférický tlak: 20 bitů,
  - nadmořská výška: 20 bitů,
  - teplota: 12 bitů.
- Měřitelný rozsah hodnot tlaku je od 20 kPa do 110 kPa.
- Kalibrovaný rozsah hodnot tlaku je od 50 kPa do 110 kPa.
- Programovatelné přerušení.
- *I2C* rozhraní.
- Autonomní sběr dat:
  - vestavěná FIFO fronta pro 32 vzorků dat,
  - logování dat a uložení dat až po dobu 12 dní ve FIFO frontě,
  - programovatelný sběr dat po určitém časovém intervalu (1 sekunda až 9 hodin).
- Typické oblasti použití jsou:
  - vysoce přesné měření tlaku a nadmořské výšky,
  - mobilní zařízení (mobilní telefony, tablety, chytré hodinky, ...),
  - GPS,
  - meteorologické stanice.

Zjednodušené schéma tohoto multifunkčního čidla je na obrázku 3.3. Digitální rozhraní čidla implementuje rozhraní *I2C*. Piny *SDA a SCL* slouží tedy pro komunikaci přes *I2C*, a zbývající piny *INT1 a INT2* jsou programovatelné piny pro přerušení.



Obrázek 3.3: Zjednodušené schéma multifunkčního čidla MPL3115A2LGA8

TPL5110DDC

Jde o programovatelný časovač pro spínání napětí od společnosti *Texas Instruments*. Je vhodný pro použití zejména v případech, kdy je nutné opakovat nějakou činnost pravidelně, v zadaných časových intervalech. Svou nízkou spotřebou 35 nA je také vhodný pro použití v systémech, které jsou napájeny bateriemi. Mezi hlavní vlastnosti tohoto modulu patří:

- Spotřeba proudu při napájení 2.5 V je 35 nA.
- Volitelný časový interval:
  - od 100 ms až po 7200 s.
- Přesnost časovače je ±1%.
- Obsahuje MOSFET pro manuální sepnutí napětí.
- Typické oblasti použití:
  - bateriemi napájené systémy,
  - IoT,
  - termostaty,
  - konzumní elektronika.

Na obrázku 3.4 lze vidět zjednodušené schéma zapojení časovače společně s MCU. Kromě samotného časovače a MCU obsahuje schéma další komponenty označené číslicí, které jsou dále popsány.

1. Zdroj napájení.

2. Tlačítko pro manuální sepnutí časovače pomocí vnitřního MOSFET tranzistoru, který je součástí časovače.

3. MOSFET tranzistor pro sepnutí napájení do MCU (tranzistor je sepnut výstupním pinem časovače *DRV*, který generuje signál po každém uběhlém časovém intervalu).

4. Odpor, jehož hodnota nastavuje délku časového intervalu, za kterou generuje pin *DRV* signál a tím sepne napětí do MCU.

    - Hodnota odporu se vypočítá podle rovnice 3.1.

$$R_{EXT} = 100 \cdot \left(\frac{-b + \sqrt{b^2 - 4a(c - 100T)}}{2a} \right) \tag{3.1}$$

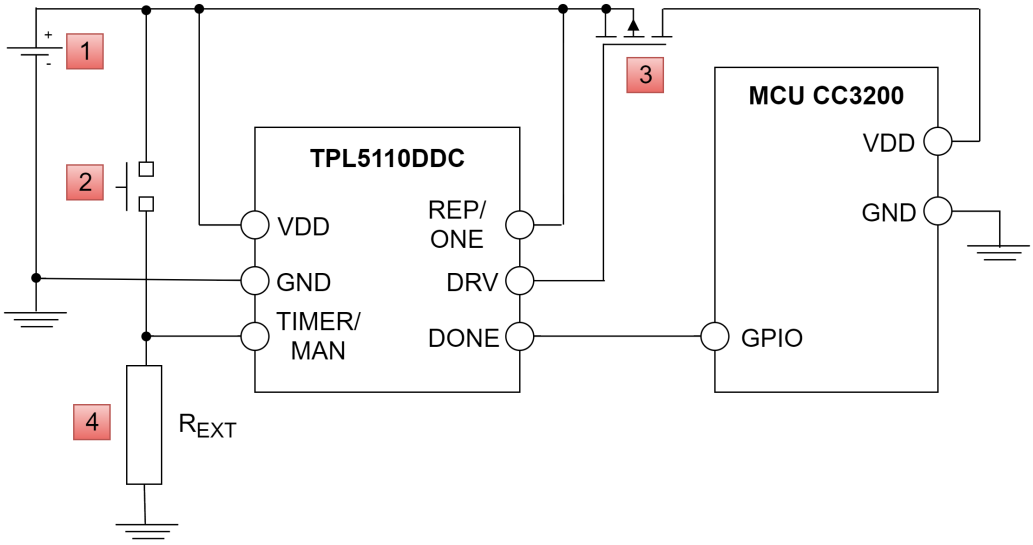
R_{EXT} představuje výslednou hodnotu odporu, T je požadovaná délka časového intervalu v sekundách a zbylé proměnné a, b, c jsou koeficienty, které se volí dle požadované délky časového intervalu. Hodnoty koeficientů podle hodnoty časového intervalu jsou uvedeny v tabulce 3.2.

Časový interval [s]	a	b	c
$(1, 5\rangle$	0.2253	-20.7654	570.5679
$(5, 10\rangle$	-0.1284	46.9861	-2651.8889
$(10, 100\rangle$	0.1972	-19.3450	692.1201
$(100, 1000\rangle$	0.2617	-56.2407	5957.7934
> 1000	0.3177	-136.2571	34522.4680

Tabulka 3.2: Hodnoty konstant pro výpočet hodnoty odporu pomocí vzorce 3.1

Jednotlivé piny časovače mají funkci:

- **VDD:** pin pro napájení časovače.
- **GND:** pin pro uzemnění.
- **TIMER/MAN:** odpor mezi tímto pinem a uzemněním nastavuje délku časového intervalu. Dále je k pinu také připojen vnitřní MOSFET tranzistor pro manuální sepnutí časovače.
- **REP/ONE:**



Obrázek 3.4: Zjednodušené schéma zapojení časovače TPL5110DDC do obvodu společně s MCU

  - pokud je tento pin připojen na úroveň odpovídající napájecímu napětí, funguje časovač jako klasický periodický spínač napětí,
  - pokud je pin naopak připojen na úroveň odpovídající uzemnění, časovač sepne MOSFET (komponenta s číslem 3 na obrázku 3.4) pouze jednou na dobu danou časovým intervalem, a další sepnutí je povoleno pouze manuálním sepnutím (pomocí tlačítka - komponenta 2 na obrázku 3.4).

- **DRV:** výstupní pin generující signál, který spíná MOSFET (komponenta s číslem 3 na obrázku 3.4) a sepne napětí do MCU.
- **DONE:** vstupní signál, kterým MCU indikuje, že dokončil svou práci. Časovač může vypnout napětí do MCU a započít další časový interval, než znovu sepne MCU.

TPS61029DRCR

Další komponentou je měnič napětí s označením *TPS61029* od společnosti *Texas Instruments*. Jedná se o synchronní napěťový převodník s efektivitou 96%. Výstupní napětí zůstává regulované i v případě, kdy vstupní napětí měniče překročí požadované výstupní napětí. Základní vlastnosti tohoto přístroje jsou:

- Možné vstupní napětí je v rozmezí od 0.9 V do 6.5 V.

- Výstupní napětí je volitelné.
- Obsahuje spořící mód pro zvýšení efektivity při nízkém výstupním výkonu.
- Detekuje nízkou kapacitu baterií.
- Obsahuje ochranu proti přehřátí.

Typickými místy, kde je tento převodník použit, jsou produkty napájené jedním, dvěma nebo třemi bateriovými články, přenosné hudební přehrávače, mobilní zařízení (mobilní telefon, PDA, ...) a LED diody u fotoaparátů.

Na obrázku 3.5 je zjednodušené schéma zapojení měniče v obvodu společně s MCU. Dále jsou popsány funkce jednotlivých pinů převodníku a MCU:

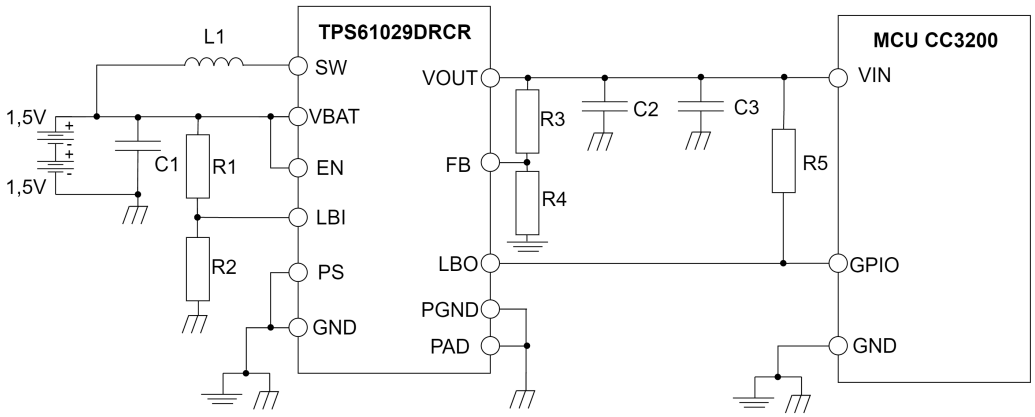
- **VBAT:** vstupní napětí pro měnič (nejčastějším zdrojem jsou baterie).
- **VOUT:** výstupní napětí měniče.
- **VIN:** vstupní napětí MCU.
- **SW:** posiluje a upravuje vstupní napětí.
- **EN:** zapnutí/Vypnutí měniče (pro zapnutí je nutno spojit s *VBAT*, pro vypnutí s *GND*).
- **PS:** zapnutí/vypnutí spořivého módu měniče (pro zapnutí je nutno spojit s *VBAT*, pro vypnutí s *GND*).
- **FB:** zpětná vazba.
- **LBI:** vstupní pin pro konfiguraci indikace nízkého vstupního napětí (vybité baterie), pokud není použit, je vhodné pin připojit na *VBAT* nebo *GND*.
- **LBO:** výstupní pin pro indikaci nízkého napětí (pin je aktivní v nízké úrovni).
- **PAD:** pouzdro měniče (je vhodné ho připojit k *PGND*).
- **PGND:** kostra zařízení.
- **GND:** uzemnění.

Kromě samotného převodníku a MCU obsahuje obvod několik dalších elektronických součástek, které slouží pro konfiguraci a zlepšení elektrických vlastností měniče:

- **L1:** cívka mezi kladným pólem vstupního napětí a vstupním pinem *SW* posiluje a upravuje vstupní napětí.
  - Doporučená hodnota je 6.8 µH.
- **C1:** kondenzátor slouží pro zlepšení přechodného chování měniče a ke snížení elektromagnetické interference celého obvodu.
  - Doporučená hodnota je nejméně 10 µF.
- **R1,R2:** odpory sloužící pro nastavení spodní hranice, která indikuje nízké vstupní napětí (vybité baterie).
  - **R2:** doporučená hodnota je 390 kΩ.
  - **R1:** hodnotu odporu je nutné dopočítat dle požadované spodní hranice pro indikaci pomocí rovnice 3.2.
- **R3,R4:** odpory sloužící pro nastavení výstupní hodnoty napětí.
  - **R4:** doporučená hodnota je 180 kΩ.
  - **R3:** hodnotu odporu je nutné dopočítat dle požadované velikosti výstupního napětí pomocí rovnice 3.3.
- **C2,C3:** kondenzátory sloužící pro redukci vlnění výstupního napětí.
  - **C3:** tantalový kondenzátor, doporučená hodnota je v intervalu od 47 µF do 100 µF.
  - **C2:** z důvodu použití tantalového kondenzátoru pro *C3* je nutné přidat ještě paralelní obyčejný keramický kondenzátor, jehož doporučená hodnota je 2.2 µF.
- **R5:** pull up rezistor, který zajišťuje, že na výstupním pinu *LBO* je při nesepnutém stavu vysoká úroveň.

$$R_1 = R_2 \cdot \left(\frac{V_{BAT}}{V_{LBI}} - 1\right) = 390k\Omega \cdot \left(\frac{V_{BAT}}{500mV} - 1\right) \tag{3.2}$$

$$R_3 = R_4 \cdot \left(\frac{V_O}{V_{FB}} - 1\right) = 180k\Omega \cdot \left(\frac{V_O}{500mV} - 1\right) \tag{3.3}$$



Obrázek 3.5: Zjednodušené schéma zapojení regulátoru napětí TPS61029 do obvodu společně s MCU

BoosterPack konektor

BosterPak konektor obsahuje celkem 40 pinů, které lze využít na přídavném plošném spoji. Většina z nich je spojena přímo s MCU *CC3200*. Některé ovšem slouží pro obecné funkce, jako jsou napájení, uzemnění a podobně.

Tabulka 3.3 obsahuje seznam všech pinů BoosterPack konektoru, které jsou ve výsledném modulu použity. Tabulka též obsahuje sloupce *MCU pin* a *GPIO pin*, které obsahují odpovídající označení pinu. Z tohoto důvodu je značení pinů poněkud zmatené. Pro každý pin BoosterPack modulu je nutné dohledávat odpovídající pin MCU, a v případě inicializace a práce s pinem ve výsledném aplikačním kódu je nutné ještě k tomu dohledávat označení *GPIO* pinu.

Tabulka 3.3 také obsahuje funkcionalitu, kterou jednotlivé piny zastávají ve výsledném modulu. Většina těchto funkcí je vysvětlena a uvedena v kapitole 3.1.1 u jednotlivých komponent výsledného modulu.

Booster Pack pin	MCU pin	GPIO pin	Vsutpní(I) / Výstupní(O)	Funkce
1	-	-	-	Vstup napájení
4	3	12	O	TPL DONE
5	61	06	I	TPS LBO
6	59	04	I	TPL DRV
7	5	14	O	ADS SPI CLK
8	62	07	I	MPL INT1
9	1	10	O	MPL I2C SCL
10	2	11	I/O	MPL I2C SDA
11	15	22	O	ADS START
12	55	01	O	ADS RESET
13	21	25	O	ADS SPI CS
14	6	15	I	ADS SPI MISO
15	7	16	O	ADS SPI MOSI
17	45	31	I	ADS DRDY
20	-	-	-	Uzemnění
22	-	-	-	Uzemnění

Tabulka 3.3: Převodní tabulka použitých pinů pro BoosterPack konektor, MCU a GPIO včetně funkcionality

3.1.2 Kompenzace rušivých jevů při měření

Všechna následující schémata měření předpokládají konstantní zdroje proudu generující 1 mA, dle doporučení výrobce teplotního čidla *Pt100*.

Všechna schémata také obsahují schodný A/D převodník a fungují na stejném principu měření:

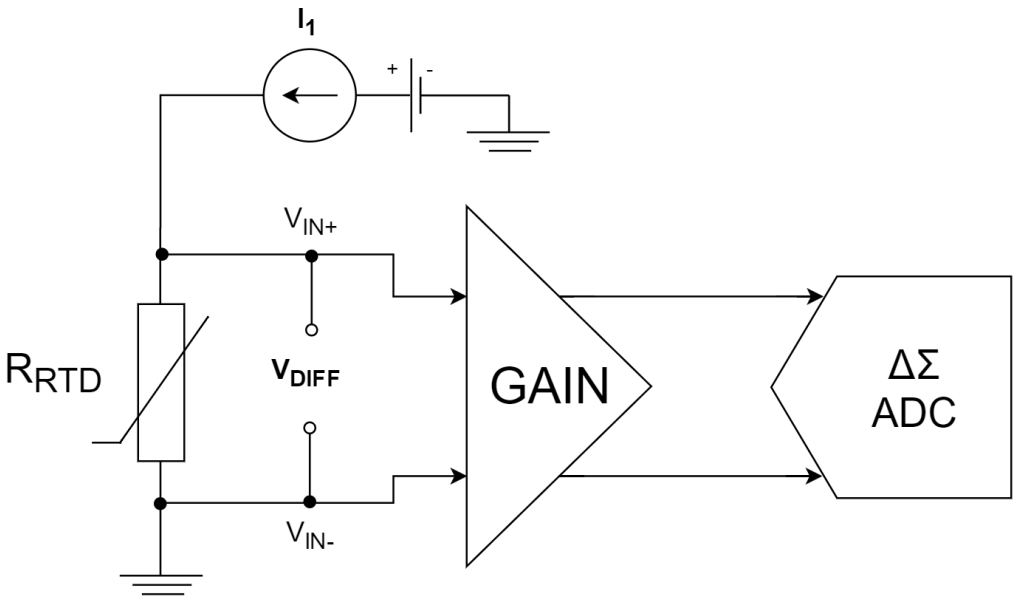
- v obvodu je konstantní zdroj proudu,
- proud prochází odporovým teplotním senzorem,
- napětí, které je na koncích teplotního senzoru, je poté zesíleno pomoci zesilovače,
- zesílené napětí je poté převáděno A/D převodníkem do digitální podoby.

Každé schéma, kromě prvního, se snaží kompenzovat některou z chyb, která vzniká při měření.

Základní schéma

První schéma na obrázku 3.6 obsahuje základní zapojení čidla *Pt100* do obvodu společně s A/D převodníkem. Jde o nejjednodušší zapojení, které lze realizovat. Proud generovaný proudovým zdrojem I_1 prochází skrz teplotní čidlo (v obvodu značené jako R_{RTD}). Dále je měřen rozdíl napětí na teplotním čidle (v obvodu značen jako V_{DIFF}), který je posléze zesílen a slouží jako vstup A/D převodníku. A/D převodník poté změřené napětí převede do digitální podoby. Výsledný odpor čidla je možné vypočítat podle rovnice 3.4.

$$V_{DIFF} = V_{IN+} - V_{IN-} = I_1 \cdot R_{RTD} \tag{3.4}$$

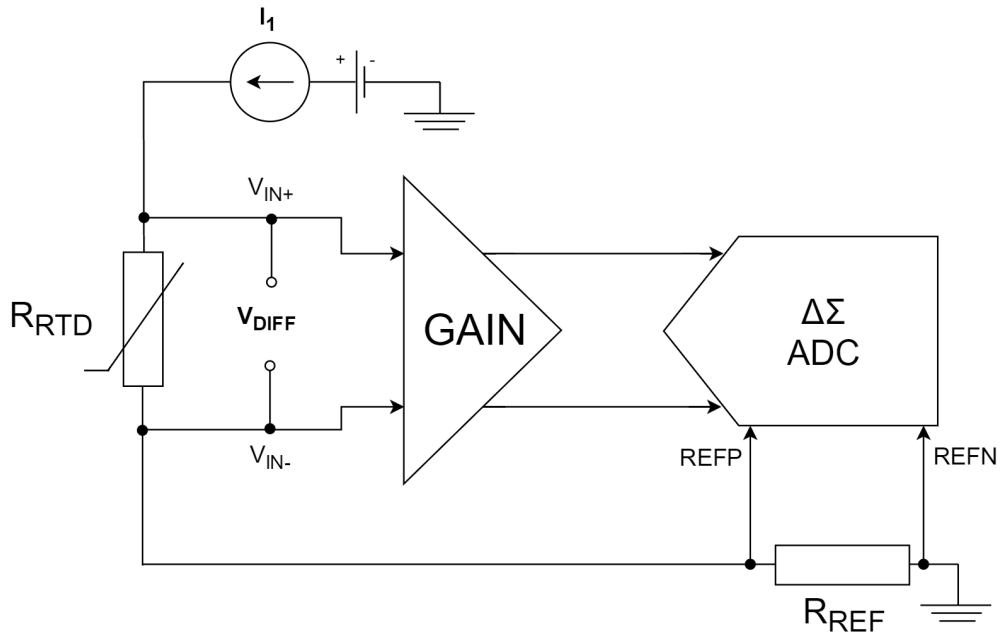


Obrázek 3.6: Základní schéma použití RTD čidla a A/D převodníku

Referenční napětí

Druhé schéma na obrázku 3.7 zavádí do obvodu tzv. referenční napětí. Použití A/D převodníku vyžaduje právě referenční napětí pro úspěšné převedení vstupního napětí do digitální podoby. Ve většině případů (i v tomto) je tato reference fixní a je generována buď interně nebo externě.

Schéma představuje tzv. 3-vodičové zapojení RTD čidla. Tento způsob zapojení používá "ratiometric configuration" pro vytvoření referenčního napětí a tím zvyšuje přesnost měření.



Obrázek 3.7: Schéma RTD čidla a A/D převodníku s přidaným referenčním napětím

Proud I_1 teče opět RTD čidlem, ale tentokrát teče ještě referenčním odporem R_{REF}, než je sveden do uzemnění. Referenční napětí V_{REF}, vznikající na tomto odporu, je poté přivedeno na pozitivní a negativní referenční pin (REFP, REFN) A/D převodníku. Napětí V_{REF} slouží A/D převodníku hned k několika účelům:

- nastavuje vstupní napětí V_{CM}[4] A/D převodníku ($V_{CM} = V_{REF}$),
- definuje rozsah vstupního napětí A/D převodníku (typicky $\pm V_{REF}$),
- je využito při konverzi vstupního napětí čidla do digitální podoby.

Hodnota V_{REF} by měla být zvolena tak, aby představovala střední hodnotu napětí z intervalu, který předpokládáme jako vstupní napětí čidla. Tím je zajištěna optimální funkcionalita A/D převodníku. Hodnotu V_{REF} přímo ovlivňuje hodnota odporu R_{REF}. Tento odpor by měl být pečlivě vybrán s důrazem na nízkou toleranci chyb, jelikož jakákoliv chyba, která se projeví v referenčním napětí V_{REF}, se projeví ve výsledné konverzi hodnot napětí teplotního čidla.

Výstup A/D převodníku je přímo závislý na vstupním napětí čidla a referenčním napětí, a proto finální výsledek konverze je jednoduše poměrem

[4] "common-mode" napětí

odporu R_{RTD} a R_{REF}, jak je ukázáno v rovnici 3.5. RES_{FINAL} je výsledná hodnota, která je čtena z A/D převodníku a RES_{PART} je mezivýsledek měření.

$$V_{REF} = V_{CM} = I_1 \cdot R_{REF}$$
$$V_{DIFF} = V_{RTD} = I_1 \cdot R_{RTD}$$

$$RES_{FINAL} = RES_{PART} \cdot \left(\frac{V_{DIFF}}{2 \cdot V_{REF}}\right) = RES_{PART} \cdot \left(\frac{R_{RTD}}{2 \cdot R_{REF}}\right) \quad (3.5)$$

Dva proudové zdroje

Využití dvou proudových zdrojů má uplatnění při použití 3-vodičového zapojení, představeného v předchozí podkapitole. Zapojením dvou proudových zdrojů (každý na jednom konci RTD čidla) lze lehce eliminovat chybu měření, vznikající vlivem odporů odporům vodičů RTD čidla, které jsou v dalším schématu na obrázku 3.8 zakresleny jako R_{LEAD}. Nejlepších výsledků lze docílit pouze tehdy, když jsou oba proudové zdroje stejné a stejně přesné.

Základním předpokladem je, že proudové zdroje generují proud o stejné hodnotě.

$$I_1 = I_2 = I$$

Poté je možné vypočítat jednotlivé hodnoty napětí V_{IN+} a V_{IN-}.

$$V_{IN+} = I \cdot (R_{LEAD} + R_{RTD}) + 2 \cdot I \cdot (R_{LEAD} + R_{REF})$$
$$V_{IN+} = 3 \cdot I \cdot R_{LEAD} + 2 \cdot I \cdot R_{REF} + I \cdot R_{RTD}$$

$$V_{IN-} = I \cdot R_{LEAD} + 2 \cdot I \cdot (R_{LEAD} + R_{REF})$$
$$V_{IN-} = 3 \cdot I \cdot R_{LEAD} + 2 \cdot I \cdot R_{REF}$$

Z těchto dvou hodnot napětí je možné dopočítat výslednou hodnotu napětí V_{DIFF} uvedenou v rovnici 3.6.

$$V_{DIFF} = V_{IN+} - V_{IN-} = I \cdot R_{RTD} \quad (3.6)$$

S dvěma proudovými zdroji se hodnota referenčního napětí V_{REF} zdvojnásobí, jak je tomu ukázáno v rovnici 3.7.

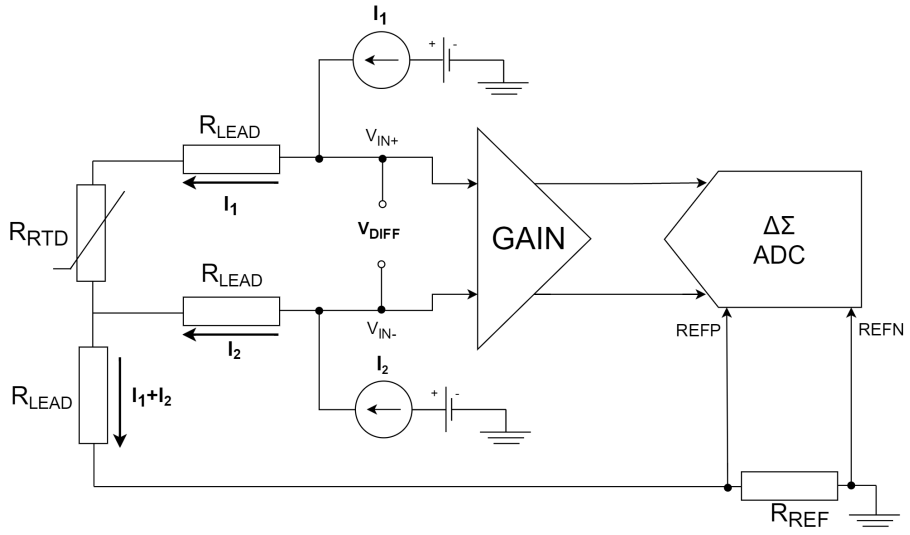
$$V_{REF} = 2 \cdot I \cdot R_{REF} \tag{3.7}$$

V tomto kroku lze již vypočítat hodnotu odporu R_{REF}, jelikož základní koncept obvodu se již nezmění. A/D převodník ve výsledném modulu je napájen napětím 3.3 V. Hodnota odporu R_{REF} byla tedy zvolena tak, aby referenční napětí převodníku V_{REF} bylo 1.64 V. Tato hodnota je zvolena záměrně tak, aby výsledný rozsah napětí vstupujícího do A/D převodníku byl od -1.64 V do 1.64 V (vstupní napětí 3.3 V rozděleno na dvě poloviny) a byl tak využit celkový rozsah měření, který A/D převodník nabízí. Jelikož jsou v obvodu dva proudové zdroje, každý generující proud o hodnotě 1 mA, hodnota odporu R_{REF} se vypočte podle rovnice 3.8.

$$R_{REF} = \frac{V_{REF}}{I_1 + I_2} = \frac{1.64}{0.001 + 0.001}\Omega = 820\Omega \tag{3.8}$$

Zdvojnásobením referenčního napětí se změní i finální výsledek konverze (viz rovnice 3.11).

$$RES_{FINAL} = RES_{PART} \cdot \left(\frac{R_{RTD}}{4 \cdot R_{REF}}\right) \tag{3.9}$$



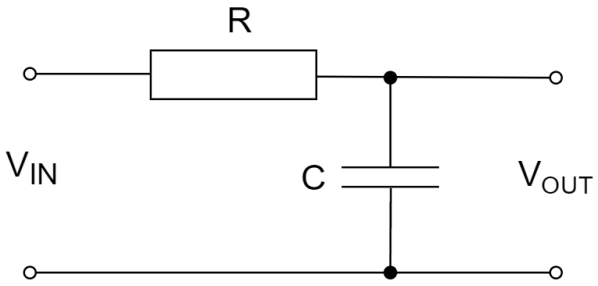
Obrázek 3.8: Schéma RTD čidla a A/D převodníku s dvěma proudovými zdroji

Dolní propust

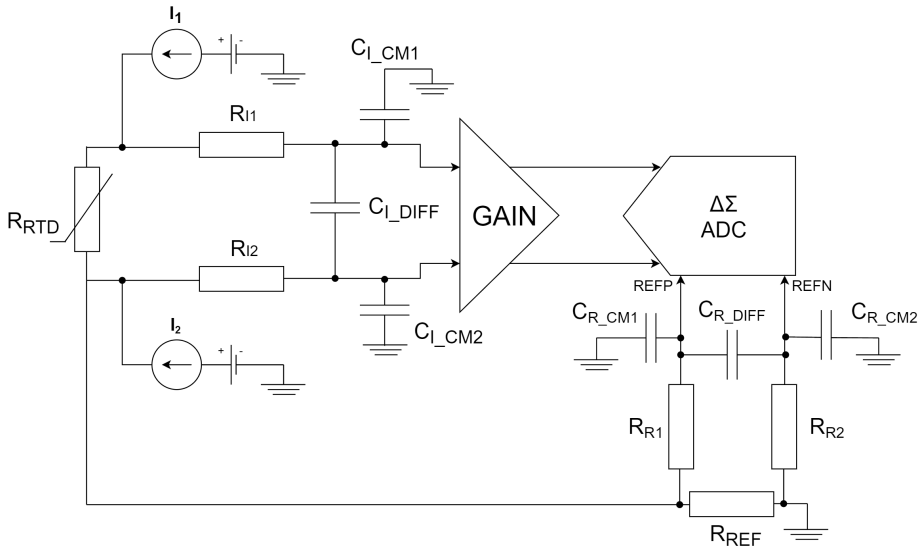
Posledním a finálním schématem zapojení RTD čidla společně s A/D převodníkem do jednoho obvodu je zobrazeno na obrázku 3.10. Toto schéma

zavádí, kromě výše uvedených vylepšení, navíc ještě filtraci pomocí dolní propusti. Dolní propustí je označován lineární filtr složený ze sériově zapojeného rezistoru a paralelně zapojeného kondenzátoru (viz obr. 3.9), jehož funkce je odfiltrování signálů vyšších frekvencí.

Dolní propusti jsou použity u vstupního napětí teplotního čidla a dále u referenčního napětí. Umístěním těchto filtrů do obvodu se zmírní vliv rušení od budících proudových zdrojů a také šumů z okolního prostředí.



Obrázek 3.9: Schéma dolní propusti



Obrázek 3.10: Výsledné schéma RTD čidla a A/D převodníku s přidanými lineárními filtry

Další obecné vylepšení

Prvním vylepšením je využití předzesilovače signálu A/D převodníku. Tento článek je označován v předchozích čtyřech schématech jako *GAIN*. Jde o pro-

gramovatelný zesilovač, který zesílí vstupní hodnoty ještě před tím, než jsou vpuštěny do samotného procesu převodu na digitální hodnotu. Vstupní rozsah hodnot, které lze vpustit do převodníku, je spjat s referenčním napětím V_{REF}, které je dále spjato s hodnotou proudových zdrojů. Pokud se zvolí nižší hodnota proudu proudových zdrojů, aby nedocházelo například k chybám způsobených zahříváním RTD čidla a přívodních vodičů, zmenší se i výsledný rozsah a změna v napětí produkovaném na RTD čidle a nebude využito celé rozsahové spektrum, které A/D převodník nabízí. Z toho důvodu je zde právě programovatelný zesilovač vstupu, který vstup zesílí tak, aby lépe odpovídal plnému rozsahu A/D převodníku a tím maximalizoval rozlišení a přesnost měření. Vstupní napětí převodníku lze vypočítat pomocí rovnice 3.10, kde *GAIN* je hodnota zesílení. Více podrobností o převodníku v kapitole 3.1.1.

$$V_{IN\_ADC} = I \cdot R_{RTD} \cdot GAIN \tag{3.10}$$

Druhým vylepšením by se mohlo jevit zvýšení hodnot proudů proudových zdrojů a tím zvýšení přesnosti měření. Tento předpoklad má dvě úskalí. Prvním úskalím je, že hodnota proudů proudových zdrojů přímo ovlivňuje velikost napětí na RTD čidle a velikost napětí generovaného proudovými zdroji je limitovaná topologií a skutečným dodávaným napětím do A/D převodníku. Z toho důvodu nelze bezhlavě zvyšovat hodnoty proudů proudových zdrojů, jinak by se mohlo stát, že napětí produkované na RTD čidlu by se vyrovnalo maximálnímu napětí, které lze proudovými zdroji vygenerovat. Druhým úskalím při zvyšování hodnot proudu, proudících RTD čidlem, je zahřívání samotného čidla včetně vodičů. Jelikož se jedná o přesné měření teploty, je logické podobnému scénáři zamezit.

Chyba měření daná referenčním odporem, proudovými zdroji a RTD senzorem

Výslednou hodnotu čtenou z A/D převodníku představuje RES_{FINAL} v rovnici 3.11. Tato rovnice platí v ideálním případě, kdy nejsou zohledněny jednotlivé tolerance ve výrobě odporu a RTD čidla.

$$RES_{FINAL} = RES_{PART} \cdot \left(\frac{R_{RTD}}{4 \cdot R_{REF}} \right) \tag{3.11}$$

Pokud bychom definovali toleranci, se kterou je odpor R_{REF} vyráběn jako Δ_{REF}, pak po dosazení hodnoty $(\Delta_{REF} \cdot R_{REF})$ do rovnice 3.11 bychom dostali rovnici 3.12.

$$RES_{R_{REF}Error} = RES_{PART} \cdot \left(\frac{R_{RTD}}{4 \cdot (R_{REF} \pm \Delta_{REF} \cdot R_{REF})} \right) \tag{3.12}$$

Chyba, kterou by tolerance v referenčním odporu způsobila, se poté vypočítá podle rovnice 3.13.

$$Chyba_{R_{REF}} = \frac{RES_{R_{REF}Error} - RES_{FINAL}}{RES_{FINAL}} \tag{3.13}$$

Po úpravách rovnice vyjde výsledná chyba v rovnici 3.14 pro případ, kdy se za referenční odpor dosadí výraz $(R_{REF}+\Delta_{REF}\cdot R_{REF})$, a 3.15 pro případ, kdy se dosadí výraz $(R_{REF} - \Delta_{REF} \cdot R_{REF})$. Δ_{REF} je rovna toleranci, se kterou je referenční odpor vyroben.

$$Chyba_{R_{REF}} = -\frac{\Delta_{REF}}{1 + \Delta_{REF}} \tag{3.14}$$

Chybu danou výrobní tolerancí referenčního odporu lze ovšem kompenzovat kalibrací pomocí *OFC* a *FSC* registrů v A/D převodníku *ADS1247* (více v kapitole 3.1.1).

$$Chyba_{R_{REF}} = \frac{\Delta_{REF}}{1 - \Delta_{REF}} \tag{3.15}$$

Z rovnice 3.6 v kapitole 3.1.2 je zřejmé, že pokud nebudou proudy, které proudové zdroje generují stejné, odpory představující přívodní vodiče RTD čidla nebudou vyrušeny. Z toho důvodu by docházelo k chybě v měření a rovnice 3.6 by již neplatila. Naštěstí A/D převodník *ADS1247* používá techniku, při které prohodí proudové zdroje mezi dvěma měřeními a výsledky zprůměruje. Tato technika zredukuje chybu, která vzniká v případě, kdy se generované proudy obou proudových zdrojů částečně liší.

Konkrétní RTD čidlo, které je použito ve výsledném modulu, patří do třídy *B*, která definuje možnou toleranci jako:

$$Chyba_{R_{RTD}} = \pm 0.3 + 0.005 \cdot |t|^{\circ}C$$

Tuto chybu lze kompenzovat stejným způsobem, jako tomu je u referenčního odporu. Konstantní složku chyby lze kompenzovat nastavením *OFC* registru, a druhou, proměnnou složku chyby, lze kompenzovat nastavením *FSC* registru. Tímto způsobem lze u čidla nižší třídy získat přesnost čidla z vyšší.

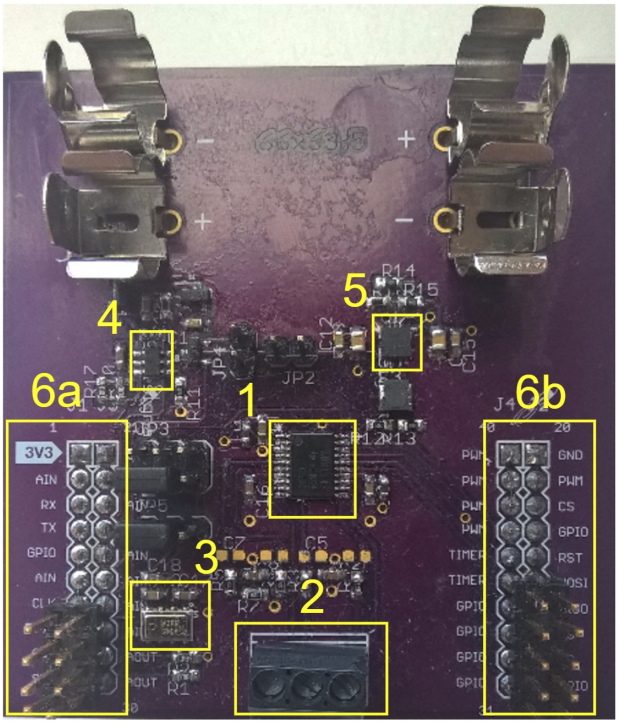
3.1.3 Schéma plošného spoje

Vytvořené schéma plošného spoje je uvedeno v příloze E. V návrhu jsou popsány jednotlivé použité součástky včetně jejich hodnot. Zdrojové soubory plošného spoje pro nástroj *Eagle* jsou součástí přiloženého CD.

3.1.4 Výsledný plošný spoj

Výsledný plošný spoj je na obrázku 3.11. Na obrázku jsou vyznačeny hlavní komponenty destičky. Jsou jimi:

1. A/D převodník *ADS1247*,
2. konektor pro připojení teplotního čidla *Pt100*,
3. multifunkční čidlo *MPL3115A2LGA8*,
4. programovatelný časovač *TPL5110DDC*,
5. napěťový převodník *TPS61029DRCR*,
6. BoosterPack konektor.



Obrázek 3.11: Výsledný plošný spoj

Všechny zmíněné prvky jsou podrobně popsány v kapitole 3.1.1.

3.2 Implementace

Při implementaci aplikace pro MCU *CC3200* byl využit nástroj *Code Composer Studio*. Vytvoření projektu a jeho patřičné nastavení pro tento nástroj je popsán v kapitole 2.1.7. Veškeré zdrojové kódy aplikačního vybavení pro MCU jsou psány v jazyce `C`.

Při implementaci serverové aplikace byl využit nástroj *Microsoft Visual Studio Community 2015*. Aplikace je napsána v programovacím jazyce `C++` a byla vyvíjena s důrazem na multiplatformní využití. Proto je možno výsledný program využít i pod operačním systémem *Linux*. Pro jednoduché přeložení pod tímto systémem je samozřejmě přiložen `Makefile`.

3.2.1 Aplikace pro MCU

Pro ovládání jednotlivých komponent byly vytvořeny samostatné zdrojové soubory.

Soubory `ads1247drv.c` a `ads1247.h` obsahují funkce pro manipulaci s A/D převodníkem *ADS1247*. Jsou jimi:

- `ADS1247DrvInit()` - tato funkce se stará o počáteční inicializaci A/D převodníku. Zároveň konfiguruje i *SPI* rozhraní. Po provedení této funkce je převodník připraven a je možné z něj číst hodnoty.
- `ADS1247DrvGetValue(int * value)` - funkce slouží k přečtení výsledné hodnoty napětí. Samotná funkce provádí měření několikrát, načež výslednou hodnotu zprůměruje a uloží do proměnné `value`, která byla předaná jako ukazatel.
- `ADS1247DrvGetPins()` - tato funkce zobrazí aktuální hodnoty pinů (vysoká/nízká úroveň), které jsou použity při komunikaci s A/D převodníkem.
- `ADS1247DrvStop()` - funkce zastaví činnost A/D převodníku nastavením `START` pinu na nízkou úroveň. Ekvivalentní možností je použití příkazu `SLEEP`. Nastavení pinu je ovšem jednodušší a rychlejší, proto byl zvolen právě tento způsob.
- `enableCS()` - funkce pro aktivaci pinu, který slouží pro výběr *Slave* stanice. Více v kapitole 2.1.2.
- `disableCS()` - funkce pro deaktivace pinu, který slouží pro výběr *Slave* stanice.

- `startADS()` - tato funkce slouží pro probuzení a aktivaci převodníku nastavením `START` pinu na vysokou úroveň. Ekvivalentní možností je použití příkazu `WAKEUP`.
- `resetADS()` - funkce pro resetování převodníku. Resetování se provede rychlým nastavením `RESET` pinu do nízké a zpět do vysoké úrovně, jelikož je tento pin aktivní v nízké úrovni.
- `readADS(unsigned int * value)` - tato funkce slouží pro jednorázové přečtení hodnoty z převodníku.
- `waitUntilReady()` - voláním této funkce dojde k aktivnímu čekání, dokud není pinem `DRDY` notifikována připravenost nově naměřených hodnot.
- `configureADS()` - funkce pro zkonfigurování převodníku. Nastavuje konfigurační registry do stavu, který odpovídá způsobu měření. Jednotlivé hodnoty jsou uvedeny a vysvětleny v kapitole 3.1.1.
- `syncADS()` - funkce, která pošle převodníku příkaz `SYNC`. Ten resetuje digitální filtry převodníku a započne novou konverzi.
- `rdataADS()` - funkce, která pošle převodníku příkaz `RDATA`. Ten načte nejnovější výsledek konverze do výstupního registru, a poté je možné hodnotu přečíst posláním 24 taktů.
- `sdatacADS()` - funkce, která pošle převodníku příkaz `SDATAC`. Ten vypne mód, při kterém jsou hodnoty z převodníku čteny kontinuálně.

Soubory `mpl3115a2drv.c` a `mpl3115a2drv.h` obsahují funkce pro práci s multifunkčním čidlem *MPL3115A2LGA8*. Jsou jimi:

- `MPL3115A2DrvInit()` - fato funkce inicializuje jednotlivé příkazy pro komunikaci s čidlem. Funkce také inicializuje *I2C* rozhraní.
- `MPL3115A2DrvStop()` - uzavírá *I2C* rozhraní.
- `MPL3115A2DrvGetPress(float * pressure)` - funkce pro přečtení hodnoty atmosférického tlaku. Výsledek uloží do proměnné `pressure`, která je předaná ukazatelem.
- `MPL3115A2DrvGetAlti(float * altitude)` - funkce pro přečtení hodnoty nadmořské výšky. Výsledek uloží do proměnné `altitude`, která je předaná ukazatelem.

- `MPL3115A2DrvGetTemp(float * temperature)` - funkce pro přečtení hodnoty teploty. Výsledek uloží do proměnné `temperature`, která je předaná ukazatelem.
- `MPL3115A2DrvWaitUntilReady()` - voláním této funkce dojde k aktivnímu čekání, dokud nejsou připraveny naměřené hodnoty ke čtení.
- `MPL3115A2DrvReadResults(unsigned char *bytes,int mode)` - funkce pro čtení hodnot z čidla. Pro výsledek alokuje paměť a odkaz na ni uloží do ukazatele `bytes`. Proměnná `mode` určuje, v jakém módu má čidlo pracovat (mód pro měření tlaku nebo mód pro měření nadmořské výšky).

Soubory `config.c` a `config.h` slouží pro konfiguraci výsledného programu. Soubory obsahují definici dvou struktur:

- `Config` - struktura pro uložení výsledné konfigurace,
- `DateTime` - struktura pro uložení a nastavení času samotného MCU.

Funkce `loadConfiguration(Config * config, DateTime * dateTime)` slouží pro načtení konfigurace. Proměnné `config` a `dateTime` jsou ukazatele pro uložení výsledných hodnot načtených z konfiguračního souboru. Cesta ke konfiguračnímu souboru je určená staticky a nelze ji změnit. Podrobnější popis konfigurace MCU je uveden k kapitole 3.4.

Soubory `pinmux.c` a `pinmux.h` obsahují funkce pro konfiguraci a inicializaci pinů MCU. Společnost *Texas Instruments* nabízí velice přehledný nástroj *TI Pin Mux Tool*[19], kterým je možné tyto soubory vygenerovat. V nástroji stačí vybrat MCU, pro který je třeba vygenerovat zdrojové kódy, a jednoduchým způsobem přidat potřebné piny. Zdrojový kód obsahuje dvě funkce. První je `PinMuxConfig()`, která byla vygenerovaná výše zmíněným nástrojem. Funkce slouží k nastavení veškerých pinů MCU, které aplikace při svém běhu využívá. Druhou funkcí je `PinMuxDisable()`, která slouží pro deaktivaci pinů pro *I2C* rozhraní.

Aplikace dále využívá zdrojové kódy, které vytváří a poskytuje společnost *Texas Instruments*. Tyto zdrojové kódy jsou součástí `SDK`[22] a poskytují širokou funkcionalitu pro ovládání periférií, komunikaci přes datová rozhraní, správu síťového rozhraní a další. Následuje výčet použitých kódů včetně popisu jejich funkcionality:

- `gpio_if.c` - toto rozhraní nabízí funkce pro manipulaci s `GPIO`[5] piny, což jsou softwarově programovatelné piny pro obecné použití. Dále

[5]General Purpose Input/Output

obsahuje funkce pro manipulaci s třemi LED diodami nacházejícími se na vývojovém kitu.

- `i2c_if.c` - toto rozhraní slouží pro práci s datovým rozhraním *I2C*. Nabízí funkce pro inicializaci a ukončení *I2C* rozhraní a funkce pro čtení a zápis přes *I2C* rozhraní.
- `network_if.c` - toto rozhraní poskytuje funkce pro správu síťového rozhraní.
- `uart_if.c` - toto rozhraní nabízí funkce, které slouží pro konfiguraci `UART` pro terminálový výstup připojený k externímu PC. Poskytuje funkce pro výpis na terminál, ale i funkce pro načtení vstupu, který uživatel zadá na terminálu v PC.
- `utils_if.c` - toto rozhraní obsahuje funkce pro uspání některých částí MCU a tím snižuje spotřebu energie v úsporných režimech, jako je např. hibernace.

Posledním a hlavním souborem obsahujícím zdrojový kód je `main.c`. Vytváří a konfiguruje hlavní *task*, který se stará o připojení k AP, měření hodnot ze senzorů a odeslání dat na serverovou aplikaci. Obsahuje následující funkce:

- `StartPin()` - nastavuje `GPIO00` pin do vysoké úrovně. Tímto pinem je možné měřit délku pracovního cyklu výsledného modulu.
- `StopPin()` - nastavuje `GPIO00` pin do nízké úrovně.
- `Connect()` - tato funkce se stará o připojení MCU k WiFi AP. Veškeré hodnoty potřebné pro připojení jsou konfigurovatelné konfiguračním souborem.
- `EnterHibernate()` - funkce pro vstup do hibernace. Délku hibernace je možné měnit v konfiguračním souboru.
- `MainTask()` - hlavní task programu. Stará se o volání jednotlivých funkcí.
- `BoardInit()` - tato funkce slouží pro inicializaci MCU a *TI-RTOS*.
- `SetTime()` - funkce pro nastavení času MCU.
- `ConfigureSimpleLinkToDefaultState()` - funkce pro konfiguraci MCU. MCU nastaví jako klienta, vypne DHCP službu, nastaví statickou IP adresu, nastaví sílu signálu WiFi a vypne mDNS službu.

- `PrintConfiguration()` - vypíše načtenou konfiguraci z konfiguračního souboru na terminál.
- `ConnectMeasureAndSend()` - funkce vytváří a konfiguruje soket pro spojení se serverovou aplikací. Dále volá funkce pro naměření hodnot ze senzorů. Poté se připojí k AP, vytvoří spojení mezi MCU a serverovou aplikací a nakonec pošle naměřená data na server.
- `main()` - hlavní funkce aplikace. Volá jednotlivé funkce pro inicializaci MCU, pinů a senzorů. Poté vytvoří hlavní task programu `MainTask` a spustí *TI-RTOS*.

3.2.2 Aplikace pro server

C++

Aplikace pro server je napsána v programovacím jazyce `C++` ve standardu `C++11`.

C++11 vlákna

Ve standardu `C++11` byla představena standardizovaná knihovna[4] pro práci s vlákny objektově orientovaným způsobem. Knihovna nabízí také třídy představující mechanismy pro vzájemné vyloučení, jako jsou *mutexy* a *podmínkové proměnné*.

Serverová aplikace využívá výše zmíněnou knihovnu pro paralelizaci výsledného programu.

OpenSSL

Pro zabezpečený přenos byla použita knihovna *OpenSSL*[18] ve verzi *1.0.2d*. Konkrétně aplikace využívá protokolu `TLS 1.2`. Podrobnější popis včetně návodu na vygenerování certifikátů potřebných pro komunikaci je uveden v kapitole 3.3.2.

Vytvořené třídy

- `Config` - třída pro konfiguraci aplikace. Obsahuje dvě statické funkce. Funkce `parseConfiguration()` nahraje a parsuje konfigurační soubor. Načtené hodnoty ze souboru jsou uloženy do statických proměnných třídy `Config`. Druhá funkce `printConfiguration()` slouží pouze pro zobrazení konfigurace.

- `Record` - třída pro uložení hodnot přijímaných z výsledných bezdrátových modulů. Instance třídy obsahuje proměnné pro uložení teploty z čidla *Pt100* a atmosférického tlaku, nadmořské výšky a teploty z čidla *MPL3115A2LGA8*. Třída obsahuje `get` metody pro všechny vyjmenované proměnné a funkci pro standardní výstup pomocí operátoru `<<`.

- `Device` - instance této třídy představuje jeden bezdrátový modul. V instanci jsou uložené veškeré záznamy o modulu, včetně všech typů naměřených hodnot za jeden den. Jednotlivé moduly jsou rozlišeny pomocí proměnné `id`, kterou lze nastavovat v konfiguračním souboru výsledného modulu. Obsahuje metody pro přidání nového záznamu měření, pro smazání všech záznamů (v případě uložení do souboru), pro přidání nového souboru se záznamy, pro získání uložených záznamů a další.

- `DeviceRequest` - tato třída slouží pro komunikaci s bezdrátovým modulem. Jedna instance třídy slouží pro jeden konkrétní přenos mezi modulem a aplikací. Třída obsahuje hlavní metodu `recvReq()`, ve které se přijímají a parsují data od modulu. K pasrovaným hodnotám lze poté přistupovat pomocí příslušných `get` metod.

- `HttpRequest` - tato třída slouží pro komunikaci mezi aplikací a webovým prohlížečem uživatele. Vyřizuje jednotlivé HTTP požadavky přicházející z webového prohlížeče uživatele. Nejdříve přijme požadavek pomocí metody `recvReq`, poté požadavek parsuje v metodě `parseReq` a nakonec podle požadovaného dokumentu vytvoří finální webovou stránku a pošle zpět uživateli.

- `DeviceInterface` - hlavní třída pro příjem spojení z bezdrátových modulů. Třída naslouchá na předem zkonfigurovaném globálním soketu a přijímá jednotlivé spojení, pro které vytváří nové sokety. Dále vytváří instance třídy `DeviceRequest`, kterým nově vytvořené sokety předává ke zpracování. Třída `DeviceInterface` vytváří určitý počet vláken (volitelný v konfiguračním souboru), které čekají na semaforu, až jim bude poskytnuto nové spojení ke zpracování. Třída dále vytváří a načítá soubory, kam jsou ukládány záznamy z jednotlivých zařízení po každém skončeném dnu.

- `WebInterface` - hlavní třída pro příjem spojení s webovými prohlížeči. Třída naslouchá na předem zkonfigurovaném globálním soketu a přijímá jednotlivé spojení, pro které vytváří nové sokety. Dále vytváří instance třídy `HttpRequest`, kterým nově vytvořené sokety předává

ke zpracování. Třída opět vytváří určitý počet vláken (volitelný v konfiguračním souboru), které čekají na semaforu, až jim bude poskytnuto nové spojení ke zpracování.

- `GraphMaker` - třída pro vytváření grafů z hodnot přijatých od bezdrátových modulů. Grafy jsou generovány pomocí HTML `svg` tagu. V aplikaci běží jedna instance této třídy ve zvláštním vlákně a stará se o pravidelné generování aktuálních grafů (rozuměj grafů pro aktuální den). Časový úsek, po kterém jsou grafy generovány, je volitelný v konfiguračním souboru.

- `Platform` - tato třída se stará o uvedení patřičných `#include` hlaviček podle platformy. Jde o síťové knihovny, které se podle použité platformy liší. Srovnává tedy rozdíly mezi systémy v použití síťových knihoven.

- `Utilities` - jde o pomocnou třídu, která obsahuje pouze statické funkce, jejichž funkcionalita se nehodila do žádné jiné třídy, nebo jejichž funkcionalitu používá více tříd. Implementuje například funkce pro získání formátovaného času v proměnné typu `std::string` nebo funkci pro vytvoření složky nezávislou na platformě.

- `Semaphore` - třída implementující semafor. Jedná se o synchronizační mechanismus pro jednotlivá vlákna aplikace.

Další pomocné soubory

Mezi další zdrojové kódy, které již ale neimplementují třídy patří:

- `constants.h` - soubor obsahuje statické proměnné, výchozí názvy konfiguračních souborů, klíče pro parsování konfiguračního souboru a další.

- `htmlcode.*` - soubory obsahují staticky definované části webových stránek, které se nemění. Poté jsou společně s dynamicky tvořenými částmi (například grafy) spojovány dohromady a odesílány podle příchozích požadavků webového prohlížeče.

- `temperatureInterpolation.*` - tyto soubory obsahují deklaraci a definici proměnné `resistToTemp` typu `std::map`. Proměnná obsahuje hodnoty pro převod naměřeného odporu teplotního čidla *Pt100* na výslednou teplotu.

- `dirent.h` - tento soubor obsahuje implementaci knihovny `dirent` pro použití v systému *Windows*. Na linuxových systémech je běžnou součástí

základních knihoven. Sjednocuje práci se složkami a soubory v obou systémech.

3.3 Zabezpečený přenos

3.3.1 TLS 1.2

Pro zabezpečení je použita nejnovější verze `TLS` kryptografického protokolu `TLS 1.2`. Protokol zabraňuje falšování a odposlouchávání jednotlivých zpráv, které si mezi sebou vyměňují komunikující strany. Dále šifruje jednotlivé zprávy a nabízí tedy soukromí při komunikaci. Navazování spojení pomocí tohoto protokolu se skládá ze tří základních kroků:

- Obě strany se dohodnou na podporovaných algoritmech pro šifrování.
- Výměna klíčů obou stran a autentizace pomocí certifikátů.
- Šifrování vzájemné komunikace symetrickou šifrou, která byla dohodnuta v prvním kroku.

3.3.2 Certifikáty

Pro zabezpečení přenosu mezi bezdrátovými moduly a serverovou aplikací je použita *OpenSSL*[18] knihovna. Pro správné fungování je nutné poskytnout oběma stranám certifikáty, kterými jsou ověřeny totožnosti při navazování spojení.

3.3.3 Generování certifikátů

Pro náš případ postačí vlastnoručně podepsaný certifikát, který bude představovat certifikát certifikační autority. Certifikační autorita poté může vydávat certifikáty, které podepíše svým privátním klíčem. Takto vydané a podepsané certifikáty je poté možno jednoduše ověřit, pokud vlastníte certifikát certifikační autority, který obsahuje její veřejný klíč.

Dále je uveden postup pro generování certifikátů na Linuxovém systému s nainstalovaným nástrojem *OpenSSL*. Konfigurační soubory použité při tvorbě certifikátů jsou uvedeny v příloze F. `CAConf.cnf` obsahuje konfiguraci pro certifikační autoritu a `ServerConf.cnf` pro server.

V dalším postupu je výchozí pracovní složkou `~/cert/`. V této složce je nutné vytvořit následující soubory a složky:

- `~/cert/CA/` - hlavní složka pro uložení certifikátů a konfiguračních souborů.
- `~/cert/CA/signedcerts/` - složka obsahující kopie všech podepsaných certifikátů.
- `~/cert/CA/private/` - složka obsahující privátní klíč certifikační autority.
- `~/cert/CA/serial` - soubor pro uložení pořadového čísla podepsaného certifikátu.
- `~/cert/CA/index.txt` - databázový soubor certifikační autority.

Vytvoření certifikátu a privátního klíče pro certifikační autoritu je možné v těchto krocích:

- `cd ~/cert/CA/ && echo '01' > serial`
  - Změní pracovní složku a nastaví výchozí pořadové číslo.
- `export OPENSSL_CONF=~/cert/CA/CAConf.cnf`
  - Nastaví výchozí konfigurační soubor pro nástroj *openssl*.
- `openssl req -x509 -newkey rsa:2048 -out cacert.pem -outform PEM -days 1825`
  - Vygeneruje privátní klíč a certifikát certifikační autority podle konfiguračního souboru.
- `openssl x509 -outform der -in cacert.pem -out cacert.der`
  - Převede certifikát z formátu `.pem` na formát `.der`, který podporuje MCU.

Dalším krokem je vygenerování privátního klíče a certifikátu pro server:

- `export OPENSSL_CONF=~/cert/CA/ServerConf.cnf`
  - Nastaví výchozí konfigurační soubor pro nástroj *openssl*.
- `openssl req -newkey rsa:1024 -keyout tempkey.pem -keyform PEM -out tempreq.pem -outform PEM`
  - Vygeneruje dočasný privátní klíč a certifikát pro server.

- `openssl rsa < tempkey.pem > server_key.pem`
  - Převede dočasný privátní klíč na nezašifrovanou formu. Pokud by klíč zůstal zašifrovaný zvoleným heslem, které se zadává při jeho generování, bylo by nutné při každém spuštění serverové aplikace zadávat právě toto heslo.

Posledním krokem je podepsání serverového certifikátu certifikační autoritou:

- `export OPENSSL_CONF=~/cert/CA/CAConf.cnf`
  - Příkaz nastaví výchozí konfigurační soubor pro nástroj *openssl*.
- `openssl ca -in tempreq.pem -out server_crt.pem`
  - Příkaz podepíše a vytvoří serverový certifikát `server_crt.pem`.

Výsledné soubory `server_crt.pem` a `server_key.pem` slouží pro serverovou aplikaci a soubor `cacert.der` pro bezdrátový modul.

3.3.4 CC3200 datum a čas

Platnost certifikátu je kontrolována pomocí systémového času zařízení. MCU *CC3200* si systémový čas nikde neukládá, a proto je nutné při každém zapnutí tento čas znovu nastavit, aby kontrola platnosti certifikátu byla úspěšná. Nastavit čas je možné přes konfigurační soubor (více v kapitole 3.4). Konfigurační soubor není nutné samozřejmě neustále měnit podle změny času a data. Výsledný čas v konfiguračním souboru bezdrátového modulu stačí zvolit tak, aby nahraný certifikát `cacert.der` prošel kontrolou platnosti.

3.4 Uživatelská příručka

3.4.1 Konfigurační soubory

MCU

Konfigurační soubor pro MCU má následující strukturu:

```
DEVICE_ID=3
DEVICE_DESC=Device with ID 3
AP_SSID=SsidOfAccessPoint
AP_PASS=PasswordOfAccessPoint
AP_SEC_TYPE=WPA2
```

```
CA_FILE=/cert/cacert.der
HIB_DURATION=60
SERVER_PORT=9999
SERVER_IP_ADDRESS=192.168.1.140
MCU_IP_ADDRESS=192.168.1.130
GATEWAY_ADDRESS=192.168.1.2
SUBNET_MASK=255.255.255.0
DNS_SERVER=192.168.1.2
DATE_YEAR=2016
DATE_MONTH=3
DATE_DAY=23
```

Vysvětlení jednotlivých položek konfiguračního souboru:

- `DEVICE_ID` - id bezdrátového modulu. Při použití několika modulů zároveň, je potřeba je odlišit pomocí identifikátoru.
- `DEVICE_DESC` - popis zařízení.
- `AP_SSID` - SSID přístupového bodu.
- `AP_PASS` - heslo k přístupovému bodu.
- `AP_SEC_TYPE` - typ zabezpečení přístupového bodu. Na výběr je z možnosti `WEP`, `WPA` a `WPA2`.
- `CA_FILE` - cesta k certifikátu certifikační autority.
- `HIB_DURATION` - doba hibernace mezi vysíláním modulu. Udává se v sekundách.
- `SERVER_PORT` - port serverové aplikace.
- `SERVER_IP_ADDRESS` - IP adresa serveru, kde běží aplikace.
- `MCU_IP_ADDRESS` - staticky přidělená IP adresa pro MCU.
- `GATEWAY_ADDRESS` - IP adresa výchozí brány (AP).
- `SUBNET_MASK` - maska sítě.
- `DNS_SERVER` - adresa DNS serveru (AP).
- `DATE_YEAR`, `DATE_MONTH` a `DATE_DAY` - položky pro nastavení systémového času MCU. Důvod je popsán v kapitole 3.3.4.

Server

Konfigurační soubor pro serverovou aplikaci má následující strukturu:

```
MCU_PORT=9999
WEB_PORT=8889
GRAPH_RATE=5
LOG_LEVEL=3
LOG_DIR=D:\logs\
DEVICE_INTERFACE_THREADS=5
WEB_INTERFACE_THREADS=5
SERVER_CERT_FILE=..\certs\server_crt.pem
SERVER_KEY_FILE=..\certs\server_key.pem
```

Výše uvedenou konfiguraci lze použít v operačních systémech *Windows*. V případě použití *Linuxového* systému je nutné dbát na odlišný způsob zápisu adresářových cest. V konfiguračním souboru pro *Linux* se tedy změní položky obsahující adresářovou cestu na:

```
LOG_DIR=/home/mint/logs/
SERVER_CERT_FILE=../certs/server_crt.pem
SERVER_KEY_FILE=../certs/server_key.pem
```

Ostatní položky zůstávají stejné. **Hlavně je nutné dbát na správné konce řádku v konfiguračním souboru.** V systému *Windows* je nutné použít `DOS` konce řádků a v *Linuxovém* systému naopak `UNIX` konce řádků.

Vysvětlení jednotlivých položek konfiguračního souboru:

- `MCU_PORT` - globální port, který přijímá spojení z bezdrátových modulů. Port je možné zvolit v rozmezí 8000 až 10000.
- `WEB_PORT` - globální port, který přijímá HTTP požadavky. Port je možné zvolit v rozmezí 8000 až 10000.
- `GRAPH_RATE` - časový úsek, po kterém jsou pravidelně generovány grafy pro webové stránky. Hodnotu je možné volit v rozmezí 1 až 180 sekund.
- `LOG_LEVEL` - úroveň výpisů aplikace. Na výběr jsou tři možnosti:
  - `1` pro vypnutí výpisů,
  - `2` pouze pro chybový výpis,
  - `3` vypisuje vše.
- `LOG_DIR` - složka pro uložení souborů, které obsahují jednotlivé přijaté hodnoty od bezdrátových modulů. Tyto soubory se tvoří jednou denně pro každý modul zvlášť. Jsou uloženy do zvláštních složek pojmenovaných podle `DEVICE_ID` zařízení. Název souborů obsahuje datum. Cesta ke složce musí končit lomítkem.

- `DEVICE_INTERFACE_THREADS` - počet vláken, které zpracovávají spojení z bezdrátových modulů. Hodnotu je možné volit v rozmezí 1 až 20.
- `WEB_INTERFACE_THREADS` - počet vláken, které zpracovávají HTTP požadavky. Hodnotu je možné volit v rozmezí 1 až 20.
- `SERVER_CERT_FILE` - cesta k serverovému certifikátu.
- `SERVER_KEY_FILE` - cesta k serverovému privátnímu klíči.

3.4.2 Aplikace pro MCU

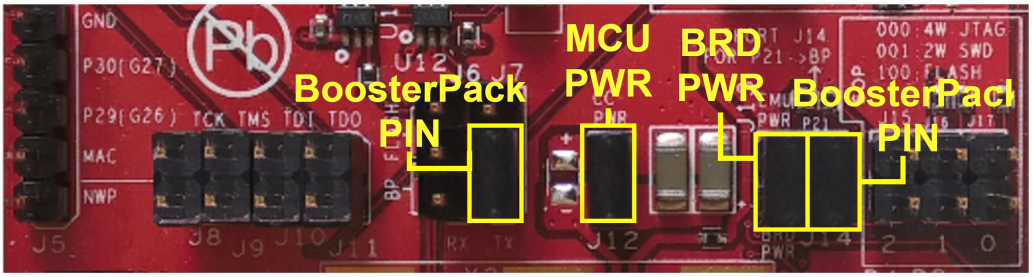
Nahrání binárního souboru aplikace, konfiguračního souboru a certifikátu do sériové FLASH paměti je nejjednodušší pomocí nástroje *CCS Uniflash*[28]. Propojky na vývojovém kitu je potřeba nastavit podle obrázku 2.8 v kapitole 2.1.7. Pokud je vše připraveno, postup je následující:

- Po spuštění nástroje *CCS Uniflash* vyberte z menu **File** → **New Configuration**.
- V poli **Connection:** vyberte **CC3x Serial(UART) Interface** a v poli **Board or Device:** vyberte **SimpleLink Wifi CC3200**
- V levém menu vyberte odrážku **/sys/mcuimg.bin**. V poli **Url** zadejte cestu k binárnímu souboru pro MCU (vyberte vlastní soubor nebo přiložený soubor na CD, cesta k souboru je uvedena v příloze G). Zaškrtněte položky **Erase**, **Update** a **Verify**.
- Pomocí menu **Operation** → **Add File** přidejte dva nové soubory. V levém menu v položce **User Files** se vytvoří dvě nové odrážky s náhodně vygenerovanými názvy.
  - Vyberte první z nich a do pole **Name** zadejte **/conf/cc3200.conf**. V poli **Url** zadejte cestu ke konfiguračnímu souboru pro MCU (vyberte vlastní soubor nebo přiložený soubor na CD, cesta k souboru je uvedena v příloze G). Zaškrtněte položky **Erase**, **Update** a **Verify**.
  - Vyberte druhý soubor a do pole **Name** zadejte cestu, kam se má uložit certifikát. Cesta k certifikátu je konfigurovatelná konfiguračním souborem, který je popsán v kapitole 3.4.1. Vložíme tedy hodnotu **/cert/cacert.der** podle vzorového konfiguračního souboru. V poli **Url** zadejte cestu k samotnému certifikátu certifikační autority (vyberte vlastní soubor nebo přiložený soubor

na CD, cesta k souboru je uvedena v příloze G). Zaškrtněte položky **Erase**, **Update** a **Verify**.

- Vyberte menu **Operation** → **Program** a vyčkejte na úspěšné dokončení. Pokud došlo k chybě, překontrolujte, zda máte správně zkratované propojky na vývojovém kitu.

Poté je potřeba znovu přenastavit propojky podle obrázku 3.12. Propojky značené nadpisem *BoosterPack PIN* slouží pro přivedení patřičných pinů MCU do *BoosterPack* konektoru. Propojky označené *MCU PWR* a *BRD PWR* slouží pro napájení MCU.



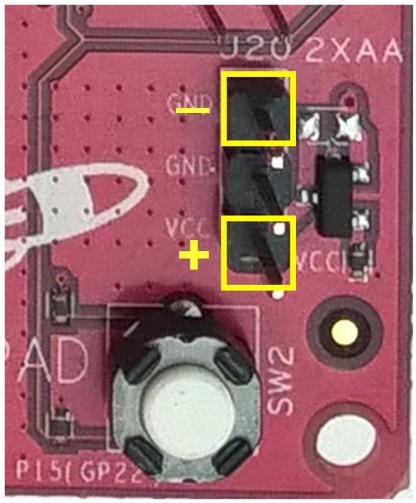
Obrázek 3.12: Nastavení propojek vývojového kitu pro použití s přídavnou destičkou

Po nastavení propojek je již vše připraveno pro samotné nasazení přídavné destičky na *BoosterPack* konektor a připojení napájení. Napájení v podobě dvou do série zapojených 1.5 V baterií lze připojit na piny podle obrázku 3.13, nebo lze využít napájení z USB kabelu, připojenému k vývojovému kitu. Toto řešení je pouze dočasné, jelikož výsledný zdroj napájení bude pocházet z přídavné destičky, kde se nachází napěťový převodník *TPS61029DRCR* a dále dvojice patic pro umístění 1.5 V baterií.

3.4.3 Aplikace pro server

Aplikace pro server je konfigurovatelná souborem uvedeném a popsaném v kapitole 3.4.1. Spuštění aplikace je tím pádem jednoduché a provede se z příkazového řádku následovně:

```
/* spusteni aplikace na systemech Widnows */
D:\>server.exe serverConf.conf

/* spusteni aplikace na systemech Linux */
mint@mint ~ $ ./server serverConfLinux.conf
```



Obrázek 3.13: Piny pro připojení napájení z baterií

Pokud nejsou uvedeny konfigurační soubory při spuštění, aplikace se pokusí načíst konfigurační soubor z výchozí cesty `.\serverConf.conf` pro systém *Windows* respektive `./serverConfLinux.conf` pro systém *Linux*.

Do složky s aplikací je možné vložit soubor `favicon.ico`. Aplikace soubor načte a bude posílat webovým prohlížečům, pokud si o něj požádají.

V případě, kdy aplikace vypíše následující chybu:

```
15.5.2016 09:53:25 - [Accepted new device client]
8396:error:1408A0C1:SSL routines:ssl3_get_client_hello:no
  shared cipher:.\ssl\s3_srvr.c:1413:
15.5.2016 09:53:25 - [INVALID SOCKET] Error setting client
  socket!
```

je to nejspíše vinou chybně uvedené cesty k serverovému certifikátu nebo serverovému klíči. Chyba pochází přímo z *OpenSSL* knihovny.

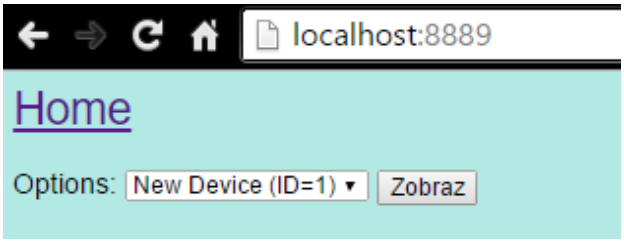
3.5 Webové rozhraní

Při načtení hlavní webové stránky je možné vybrat zařízení podle popisu uvedeném v konfiguračním souboru bezdrátového modulu (viz obr. 3.14).

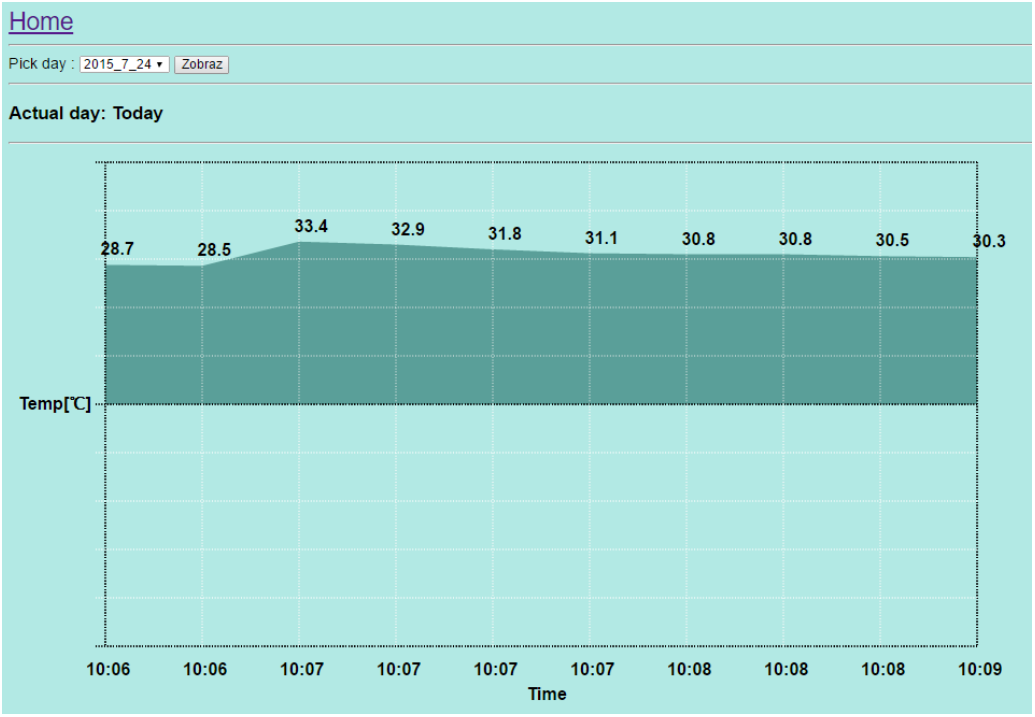
Po kliknutí na tlačítko `Zobraz` je načtena stránka s grafy, které obsahují hodnoty vybraného zařízení z aktuálního dne (viz obr. 3.15). Na této stránce je dále uveden select box s daty, které odpovídají názvům uložených souborů vybraného modulu s hodnotami z avizovaného data.

Výběrem konkrétního data a kliknutím na tlačítko `Zobraz` bude načtena stránka s hodnotami z vybraného dne (viz obr. 3.16). Hodnoty zanesené

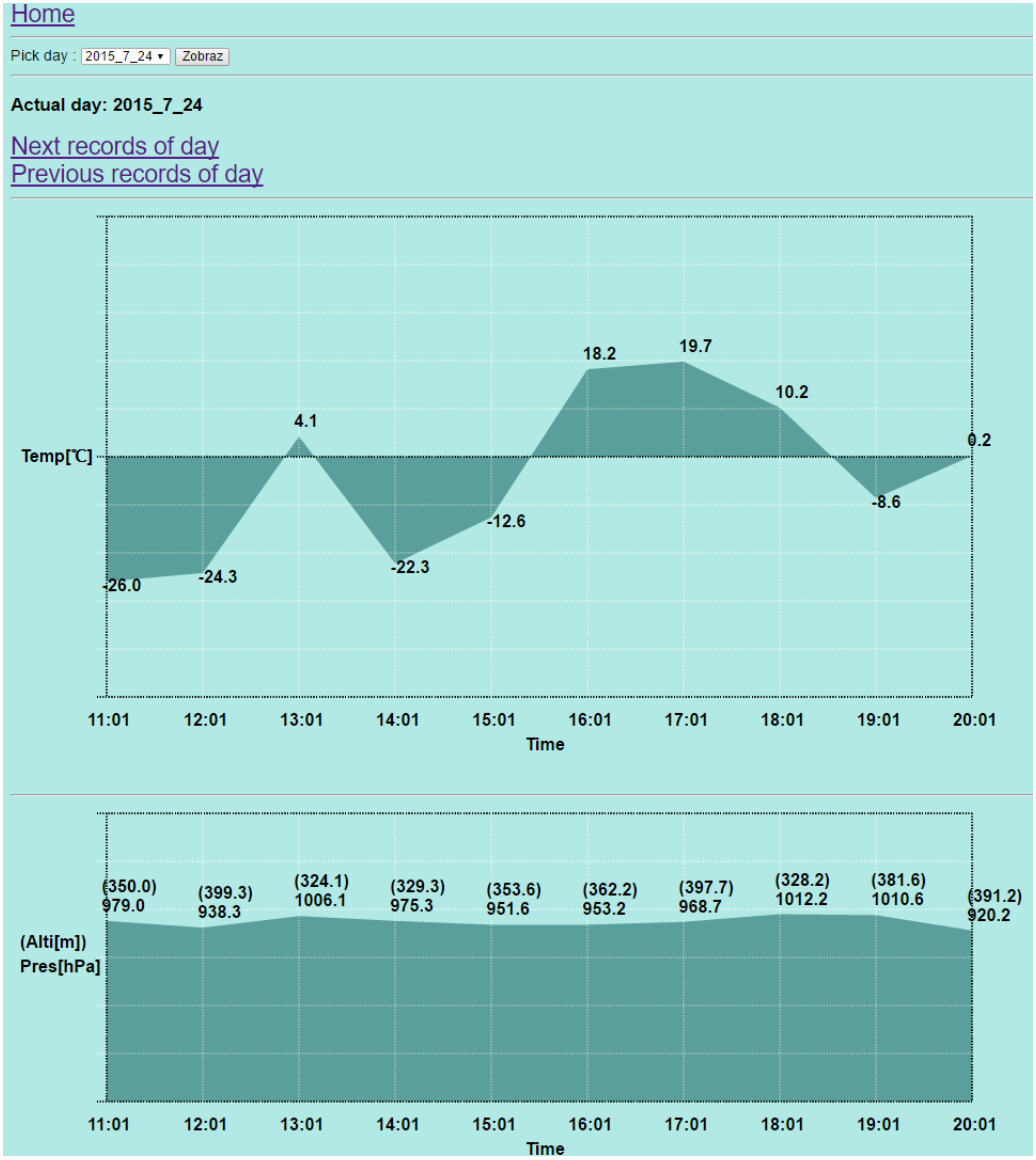
do grafu jsou náhodně vygenerované. Odkazy označenými *Next records of day* a *Previous records of day* je možné listovat jednotlivými záznamy z vybraného dne.



Obrázek 3.14: Hlavní webová stránka



Obrázek 3.15: Webová stránka zobrazující graf s hodnotami z aktuálního dne



Obrázek 3.16: Webová stránka zobrazující graf s hodnotami z konkrétního dne

4 Měření

4.1 Měření přesnosti ADS

Měření přesnosti A/D převodníku bylo provedeno tak, že místo RTD čidla byly do přídavné destičky připojeny klasické odpory. Hodnoty odporů byly voleny v rozsahu, který je možný dosáhnout i s RTD čidlem. Jednotlivé hodnoty odporů byly předem pečlivě změřeny multimetrem *Agilent U1253A*. Odpory byly měřeny v rozsahu do 500 Ω. V tomto rozsahu dosahuje multimetr rozlišení 0.01 Ω a přesnosti 0.05 %. Tabulka 4.1 obsahuje porovnání naměřených hodnot odporů pomocí multimetru s hodnotami odporů, získaných výsledným zařízením pro měření přesné teploty.

Hodnota odporu změřená multimetrem [Ω]	Hodnota odporu získaná ze zařízení [Ω]	Rozdíl naměřených hodnot [Ω]
107.4	107.36	0.04
100.5	100.413	0.087
99.8	99.73	0.07
90.1	90.03	0.07

Tabulka 4.1: Srovnání hodnot odporů měřených multimetrem s hodnotami získanými výsledným zařízením

4.2 Měření spotřeby

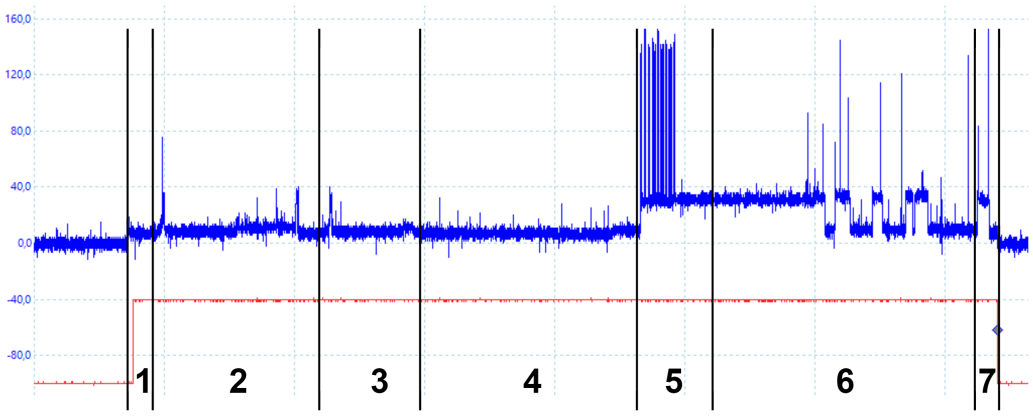
Měření spotřeby celkového modulu probíhalo ve školní laboratoři pomocí digitálního osciloskopu. Mezi rozpojené konce napájecího obvodu byl napájen odpor o hodnotě 0.47 Ω a na koncích odporu bylo měřeno osciloskopem napětí. Pomocí vztahu $I = \frac{U}{R}$ je poté možné dopočítat elektrický proud, který obvodem teče v určitou chvíli.

Všechny grafy obsahují dvě křivky. Modrá křivka představuje průběh měřeného napětí na odporu. Červená křivka značí aktivitu výsledného zařízení. Zapnutí (probuzení z hibernace) zařízení je vidět na červené křivce v okamžiku, kdy je hrana nastavena do vysoké úrovně. Vstup do hibernace na druhou stranu demonstruje nastavení hrany do nízké úrovně. Důvodem zašuměných měření (modrá křivka) je nízká hodnota vybraného odporu.

Vyšší hodnota odporu nemohla být použita, a to z důvodu velkého poměru odebíraných proudů v hibernaci a aktivním režimu. Zvýšení hodnoty odporu na desetinásobek vedlo k opakovanému restartu při zapnutí napájení. Důvodem byl příliš velký pokles napětí na procesoru.

Graf na obrázku 4.1 zobrazuje průběh odběru proudu MCU. Průběh byl měřen na pinech konektoru `J12` vývojového kitu (s odstraněnou propojkou a napájeným odporem) a zahrnuje pouze průběh odběru proudu samotného MCU. Celková doba pracovního cyklu (doba, po kterou je červená křivka ve vysoké úrovni) je 5.5 s. V grafu jsou označeny číslicemi jednotlivé fáze, které představují určité části programu. Fáze programu, odděleny značkami vytvořenými změnou výstupní hodnoty pinu `GPIO00`, jsou (v závorkách jsou uvedeny přibližné časy trvání jednotlivých fází):

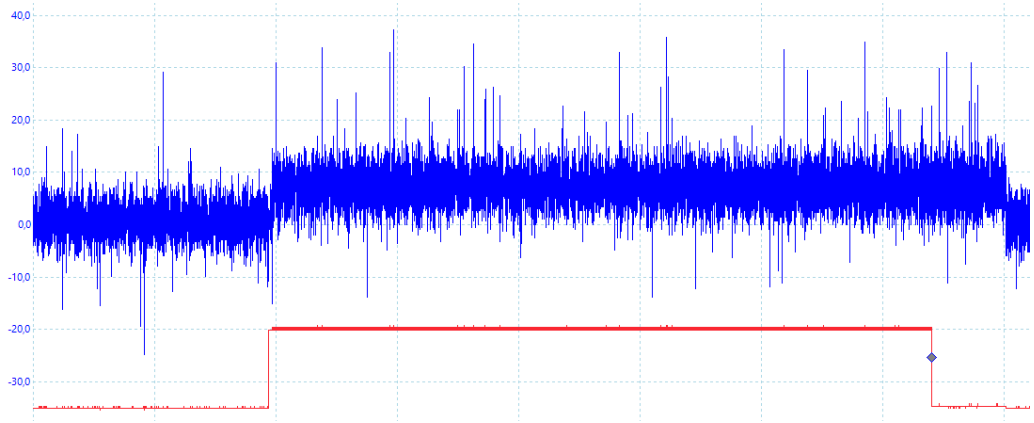
1. - inicializace čidla *MPL3115A2LGA8* a A/D převodníku *ADS1247* (40 ms).
2. - čtení konfiguračního souboru (1400 ms).
3. - zapnutí síťového rozhraní včetně konfigurace soketu pro komunikaci se serverem (740 ms).
4. - čtení hodnot ze senzorů (1470 ms).
5. - připojení k AP (530 ms).
6. - navázání spojení s aplikací běžící na serveru (1310 ms).
7. - odeslání hodnot na server (10 ms).



Obrázek 4.1: Průběh odběru proudu MCU

Z uvedeného grafu je vidět, že spotřeba MCU ve fázi, kdy se nevysílá, je poměrně malá. Největší množství energie je spotřebováno ve fázi 5 a ve fázi 6. Vlastní přenos dat představuje zanedbatelnou část spotřebované energie.

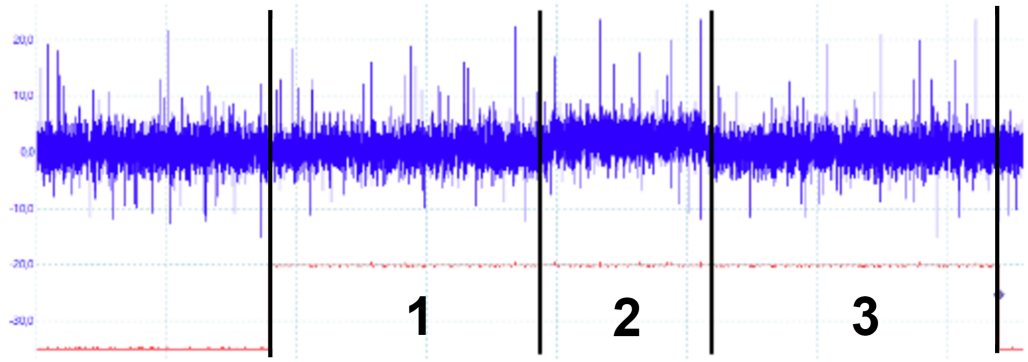
Graf na obrázku 4.2 zobrazuje průběh odběru proudu A/D převodníku *ADS1247*. Průběh byl měřen na pinech 1 a 3 konektoru `JP3` (viz schéma v příloze E). Celková doba pracovního cyklu je 5.6 s. Křivka, reprezentující odběr proudu obvodem, je v tomto případě konstantní po celý pracovní cyklus. Z grafu je jasně vidět, že převodník odebírá proud i po vstupu procesoru do hibernace. Tento fakt je způsobený nejspíše tím, že převodníku trvá déle vypnout napájení. Tento problém bude eliminován poté, kdy bude spínání napětí ovládáno programovatelným časovačem *TPL5110DDC*. Ten napájení vypne pro všechny obvody ve stejnou dobu.



Obrázek 4.2: Průběh odběru proudu A/D převodníku

Průběh odběru proudu čidlem *MPL3115A2LGA8* zobrazuje graf na obrázku 4.3. Průběh byl měřen na pinech 1 a 2 konektoru `JP5` (viz schéma v příloze E). Celková doba pracovního cyklu je 5.6 s. Ve vyznačeném intervalu 2 lze pozorovat mírné zvýšení odběru proudu, které je způsobeno probíhajícím měřením čidla. V intervalu 1 probíhá inicializace procesoru, čtení konfiguračního souboru a zapnutí síťového rozhraní. Odebíraný proud je tedy nulový. V intervalu 3 jsou již hodnoty naměřeny a odběr čidla je opět nulový. Tento interval odpovídá připojení k AP, navazování spojení a odesílání hodnot.

Posledním bodem měření bylo zjištění hodnoty proudu, který MCU spotřebovává v hibernaci. Pro toto měření již nebyl použit osciloskop, ale multimetr *Agilent U1253A*, jelikož doba, po kterou je procesor v režimu hibernace, je dostatečně dlouhá jak pro ustálení výchylky multimetru, tak i pro odečtení údaje. Proud se měřil na pinech rozpojeného konektoru `J12` na vývojovém



Obrázek 4.3: Průběh odběru proudu čidla MPL3115A2

kitu. Hodnota výsledného proudu odebíraného v hibernaci je konstantní a je rovna 10 µA.

4.3 Výpočet doby života

Hodnoty naměřené osciloskopem v kapitole 4.2 lze exportovat do souboru ve formátu `csv`. Tento soubor lze poté zpracovat pomocí tabulkového procesoru.

Hodnoty ze všech tří měření z kapitoly 4.2 byly vyexportovány a zvlášť nahrány do tabulkového procesoru. Dále byly vymazány ty hodnoty, které odpovídaly průběhu napětí v době, kdy byl MCU v hibernaci. Ze zbylých hodnot napětí byl vypočítán určitý integrál, který udává průměrnou hodnotu napětí. Kladné a záporné odchylky dané zašuměním se při výpočtu z větší části vykompenzovaly. Průměrná hodnota napětí se poté jednoduše podělí hodnotou odporu, pomocí kterého probíhalo měření s osciloskopem, a vyjde výsledná průměrná spotřeba proudu.

V tabulce 4.2 jsou výsledné průměrné hodnoty proudů.

Zařízení	Průměrná hodnota proudu [mA]
MCU CC3200	27.96
ADS1247	15.67
MPL3115A2	2.19
Celkem (I_a)	45.82

Tabulka 4.2: Průměrné hodnoty proudů

V tabulce 4.2 je definován součet průměrných hodnot odebíraných proudů

v době aktivity zařízení jako I_a. Dále definujme hodnotu proudu odebíraného v době hibernace jako I_h, jejíž hodnota je 10 µA. Dobu aktivity definujme jako t_a s hodnotou 6 s (při měření byla maximální doba rovna 5.6 s). Doba hibernace bude definována jako t_h a její hodnota bude proměnná. Průměrná spotřeba, definována jako I_p, za dobu $t_a + t_h$ se vypočte podle vztahu 4.1.

$$I_p = \frac{t_h \cdot I_h + t_a \cdot I_a}{t_h + t_a} \tag{4.1}$$

Výsledné doby života jsou uvedeny v tabulce 4.3. Hodnoty jsou vypočteny při použití dvou 1.5 V baterií s kapacitou 2700 mAh zapojených do série. Doba života je vypočtena podle vztahu 4.2. Výsledné hodnoty jsou orientační a vychází přibližně, jelikož na baterie působí různé vlivy prostředí, které snižují jejich kapacitu.

$$t_v = \frac{kapacita\ baterií}{I_p} \tag{4.2}$$

Čas strávený v hibernaci (t_h) [min]	Průměrná spotřeba (I_p) [µA]	Výsledná výdrž modulu (t_v) [dny]
10	463.5	242
30	162.2	693
60	86.2	1304
120	48.1	2336

Tabulka 4.3: Výsledná doba života modulu dle času stráveného v hibernaci

5 Dosažené výsledky

Vzhledem k použití platinového čidla *Pt100* třídy B, není možné bez kompenzace obecně dosáhnout požadované přesnosti měření tepoty 0.1 °C. A/D převodník *ADS1247* ale obsahuje dva registry, kterými lze upravit výsledné hodnoty měření. Po vytvoření odpovídajícího programu a prostředí pro přesné měření teploty, by bylo možné obsahy těchto registrů nastavit tak, aby byla dosažena požadovaná přesnost.

Při realizaci výsledného plošného spoje se navíc podařilo zadání rozšířit přidáním dalších integrovaných obvodů. Jedná se o multifunkční čidlo *MPL3115A2LGA8*, programovatelný časovač *TPL5110DDC* a DC/DC převodník *TPS61029DRCR*.

Multifunkční čidlo se podařilo úspěšně zakomponovat do řešení. Funkcionalita výsledného modulu byla rozšířena o možnost měřit atmosférický tlak a nadmořskou výšku.

Na ostatní dva obvody již bohužel nezbyl při realizaci práce čas. Konkrétně programovatelný časovač by mohl v budoucnu nahradit funkci hibernace. Odběr tohoto obvodu v klidu je podle katalogu pouze 35 nA. Na druhou stranu, podle provedeného měření, má procesor v hibernaci odběr 10 µA, což je asi 300x více. Použitím časovače by se tedy snížila průměrná spotřeba proudu, a tím by se prodloužila doba života modulu.

Z časových důvodů se nepodařilo uvést do provozu DC/DC převodník. Problém byl zejména s jeho připájením, které se s prostředky dostupnými na katedře nepodařilo. Zařízení bylo tedy napájeno z USB konektoru.

5.1 Návrh na úpravu

V původním návrhu je pro signál *chip select* u rozhraní SPI použit pin MCU číslo 21 (`GPIO25`). Tento pin není výchozím pinem pro použití této funkce. Z toho důvodu s ním neumí zacházet knihovní funkce pro SPI rozhraní, obsažená v `SDK`, a pin je nutné ovládat dodatečně manuálním nastavením v programu. Pokud by se místo tohoto pinu použil pin MCU číslo 8 (`GPIO17`), nebylo by nutné s pinem manipulovat manuálně.

Dále byl při návrhu přiřazen nevhodným způsobem pin sloužící jako signál `RESET` pro A/D převodník. Pro tento signál byl zvolen pin MCU číslo 55 (`GPIO01`). Ten ovšem ve vývojovém kitu slouží jako výchozí pin pro `TX` signál rozhraní `UART`. Proto bylo nutné signál `TX` přemapovat na pin MCU číslo 53 (`GPIO30`) a přesunout propojku `J7` na vývojovém kitu do druhé polohy, kdy

je přiveden signál pinu 55 (`GPIO01`) na *BoosterPack* konektor. Pro signál `RESET` A/D převodníku bych tedy v upraveném návrhu volil například pin MCU číslo 18 (`GPIO28`).

Další zajímavou úpravou by bylo upravit osazení modulu a k měření použít teplotní čidlo *Pt1000* s nominálním odporem 1000 Ω při 0 °C. Hodnoty proudů proudových zdrojů A/D převodníku by byly v tom případě zmenšeny na 100 µA. Tím by se dosáhlo snížení průměrné spotřeby a zároveň vyššího rozlišení měření.

Při návrhu byla přehlédnuta šířka destičky (je větší než šířka vývojového kitu). Dále byl přehlédnut průměr AAA baterií, takže jejich držáky jsou příliš blízko sebe. Bylo by také vhodné přidat vypínač (nebo propojku), který by byl schopen rozpojit napájení baterií. Nyní je pro tento efekt nutné baterie vyjmout z držáku.

Dalším možným vylepšením by bylo využití jiného principu měření hodnot. Například by bylo možné naměřit více hodnot, které by se hned neodesílaly, ale jen uložily společně s časovou značkou. Perioda měření by byla např. 10 minut a hodnoty by se odesílaly na server jednou za 2 hodiny. Tím by se ušetřila energie, potřebná pro navazování spojení a odesílání hodnot. Bylo by ovšem nutné přidat obvod reálného času (RTC), který by sloužil pro uchování aktuálních hodnot data a času. Ty by byly potřeba pro vytváření časových značek k naměřeným hodnotám. Odpadla by tím i nutnost manuálně nastavovat čas procesoru, který v tuto chvíli slouží pouze pro kontrolu platnosti certifikátu.

6 Závěr

S takovouto úlohou jsem se setkal poprvé a její zvládnutí bylo složité. Úloha pokrývá širokou oblast problematiky od hardwaru až po webový software. Musel jsem se seznámit s podrobným hardwarovým popisem a charakteristikou jednotlivých použitých obvodů, dále se seznámit s širokou škálou nástrojů, od návrhových systémů (*Eagle*) až po vývojové prostředí (*CCS*), a v neposlední řadě vytvořit softwarové vybavení pro MCU *CC3200* a webový server.

Do zprávy jsem se snažil napsat i takové informace, které jsem získal z nastudovaných materiálů s cílem, aby pomohly ostatním lidem, kteří se budou zabývat podobnou problematikou. Zprávu jsem také doplnil o pracovní postupy, např. vytvoření projektu v CCS. Z těchto důvodu je rozsah práce poměrně velký.

V práci jsem se zabýval metodami přesného měření teploty. Srovnal jsem různá dostupná řešení v podobě průmyslových teplotních čidel. Pro výsledný modul jsem vybral platinové čidlo *Pt100* a zakomponoval ho do obvodu v podobě přídavné destičky pro vývojový kit *SimpleLink Wi-Fi CC3200 LaunchPad*.

Dále jsem vyvinul softwarové vybavení pro výsledný modul s procesorem *CC3200*. Ten je schopný bezdrátového přenosu naměřených hodnot z teplotního čidla. Také jsem implementoval aplikaci na straně serveru, která naměřené hodnoty zpracovává, ukládá a prezentuje v podobě grafu.

Ověřil jsem funkčnost realizovaného řešení. Na realizovaném vzorku jsem provedl měření spotřeby v různých fázích činnosti. Z naměřených hodnot jsem spočítal dobu života modulu při napájení z bateriového zdroje.

Literatura

[1] *ADS1247* [online]. Texas Instruments Incorporated, 2016. [cit. 2016/05/30]. Dostupné z: http://www.ti.com/product/ADS1247.

[2] *BMA222 Digital, triaxial acceleration sensor* [online]. Bosch Sensortec GmbH, 2016. [cit. 2016/05/15]. Dostupné z: http://www.mouser.com/ds/2/783/BST-BMA222-FL000-02-786483.pdf.

[3] *TI LaunchPad - BoosterPacks* [online]. Texas Instruments Incorporated, 2016. [cit. 2016/05/15]. Dostupné z: http://www.ti.com/ww/en/launchpad/boosterpacks.html.

[4] *Thread support library* [online]. 2016. [cit. 2016/05/29]. Dostupné z: http://en.cppreference.com/w/cpp/thread.

[5] *CC3200* [online]. Texas Instruments Incorporated, 2016. [cit. 2016/05/15]. Dostupné z: http://www.ti.com/product/CC3200.

[6] *CC32xx Power Management Framework* [online]. Texas Instruments Incorporated, 2016. [cit. 2016/05/15]. Dostupné z: http://processors.wiki.ti.com/index.php/CC32xx_Power_Management_Framework.

[7] *Code Composer Studio (CCS) Integrated Development Environment (IDE) for Wireless Connectivity* [online]. Texas Instruments Incorporated, 2016. [cit. 2016/05/16]. Dostupné z: http://www.ti.com/tool/ccstudio-wcs.

[8] *Cygwin* [online]. Red Head Inc., 2016. [cit. 2016/05/16]. Dostupné z: http://cygwin.com/.

[9] *Eclipse* [online]. The Eclipse Foundation, 2016. [cit. 2016/05/16]. Dostupné z: https://eclipse.org.

[10] *Energia - Prototyping Software to Make Things Easy* [online]. Energia, 2016. [cit. 2016/05/16]. Dostupné z: http://energia.nu/.

[11] *Energia Libraries* [online]. Energia, 2016. [cit. 2016/05/16]. Dostupné z: http://energia.nu/reference/libraries/.

[12] *Energia MT* [online]. Energia, 2016. [cit. 2016/05/16]. Dostupné z: http://energia.nu/tag/energia-mt/.

[13] *How to set up Energia on Windows* [online]. Energia, 2016. [cit. 2016/05/16]. Dostupné z: http://energia.nu/guide/guide_windows/.

[14] *FreeRTOS* [online]. Real Time Engineers Ltd., 2016. [cit. 2016/05/25]. Dostupné z: http://www.freertos.org/index.html.

[15] *FT2232D Dual USB to Serial UART/FIFO IC* [online]. Future Technology Devices International Ltd., 2016. [cit. 2016/05/15]. Dostupné z: http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT2232D.pdf.

[16] *IAR Embedded Workbench* [online]. IAR Systems, 2016. [cit. 2016/05/16]. Dostupné z: https://www.iar.com/iar-embedded-workbench/.

[17] *IST AG Innovative Sensor Technology* [online]. Innovative Sensor Technology IST AG, 2016. [cit. 2016/05/25]. Dostupné z: http://www.ist-ag.com/.

[18] *OpenSSL* [online]. OpenSSL Software Foundation, 2016. [cit. 2016/05/29].

[19] *Pin Mux Tool* [online]. Texas Instruments Incorporated, 2016. [cit. 2016/05/29]. Dostupné z: http://www.ti.com/tool/PINMUXTOOL.

[20] *ST INNOVATIVE SENSOR TECHNOLOGY P0K1.202.3FW.B.007 RTD Senzor, Plochý Kabel, Třída B* [online]. Premier Farnell plc., 2016. [cit. 2016/05/25]. Dostupné z: http://cz.farnell.com/ist-innovative-sensor-technology/p0k1-202-3fw-b-007/sensor-temp-100ohm-flat-wire-cl/dp/2191841.

[21] *SimpleLink Wi-Fi CC3200 LaunchPad* [online]. Texas Instruments Incorporated, 2016. [cit. 2016/05/15]. Dostupné z: http://www.ti.com/tool/cc3200-launchxl.

[22] *SimpleLink Wi-Fi CC3200 Software Development Kit (SDK)* [online]. Texas Instruments Incorporated, 2016. [cit. 2016/05/16]. Dostupné z: http://www.ti.com/tool/cc3200sdk.

[23] *How to Temboo with your TI LaunchPad* [online]. Temboo, Inc, 2016. [cit. 2016/05/15]. Dostupné z: https://temboo.com/hardware/ti/getting-started.

[24] *Tables of Thermoelectric Voltages for All Thermocouple Types* [online]. National Instruments Corporation, 2016. [cit. 2016/05/25]. Dostupné z: http://www.ni.com/white-paper/4231/en/.

[25] *Thermistor Resistance vs. Temperature* [online]. OMEGA Engineering inc., 2016. [cit. 2016/05/25]. Dostupné z: http://www.omega.com/temperature/Z/pdf/z256-257.pdf.

[26] *TI-RTOS: Real-Time Operating System (RTOS)* [online]. Texas Instruments Incorporated, 2016. [cit. 2016/05/16]. Dostupné z: `http://www.ti.com/tool/ti-rtos`.

[27] *TI-RTOS: Real-Time Operating System (RTOS)* [online]. Texas Instruments Incorporated, 2016. [cit. 2016/05/25]. Dostupné z: `http://www.ti.com/tool/ti-rtos`.

[28] *Uniflash Standalone Flash Tool for TI Microcontrollers (MCU), Sitara Processors & SimpleLink devices* [online]. Texas Instruments Incorporated, 2016. [cit. 2016/05/30]. Dostupné z: `http://www.ti.com/tool/uniflash`.

[29] *TMP006 Infrared Thermopile Contactless Temperature Sensor in WCSP Package* [online]. Texas Instruments Incorporated, 2016. [cit. 2016/05/15]. Dostupné z: `http://www.ti.com/product/TMP006`.

A Seznam použitých zkratek

MCU microcontroller

AP access point

CCS Code Composer Studio

RTS ready to send

CTS clear to send

API application programming interface

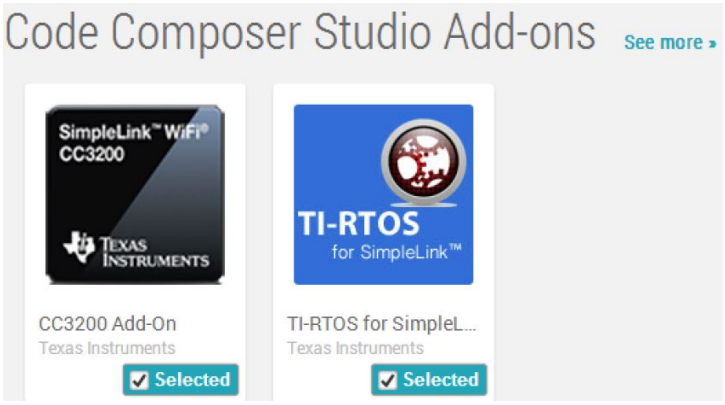
RTD resistance temperature detector

GPIO general purpose input/output

RTOS real time operating system

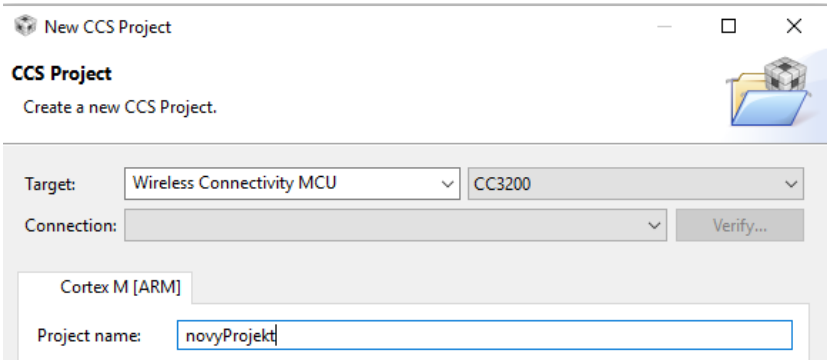
B Postup instalace a konfigurace nástroje Code Composer Studio

1. Nejdříve je potřeba stáhnout aktuální `SDK` pro MCU *CC3200* ze stránek *Texas Instruments*[22].
2. Poté je nutné stáhnout samotný nástroj *Code Composer Studio*(CCS) ze stránek *Texas Intruments*[7].
3. Spusťte CCS.
4. Dalším krokem je instalace doplňků (add-on) pro správnou funkcionalitu s MCU *CC3200* (viz obr. B.1):
   - Spusťte *CCS App Center* z menu **Help** → **CCS App Center**.
   - Vyhledejte následující dva add-ony a nainstalujte: *CC3200 Add-On, TI-RTOS for Simplelink.*



Obrázek B.1: Instalace add-onů do CCS

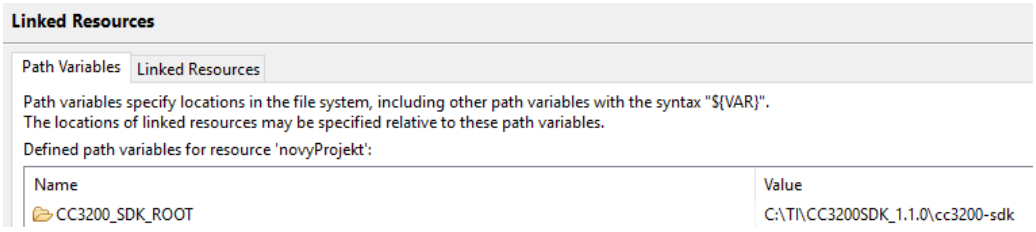
5. Dalším krokem je vytvoření nového projektu z menu **File** → **New** → **CCS Project** (viz obr. B.2):
   - Položku **Target** nastavte na `Wireless Connectivity MCU`.



Obrázek B.2: Vytvoření nového projektu v CCS

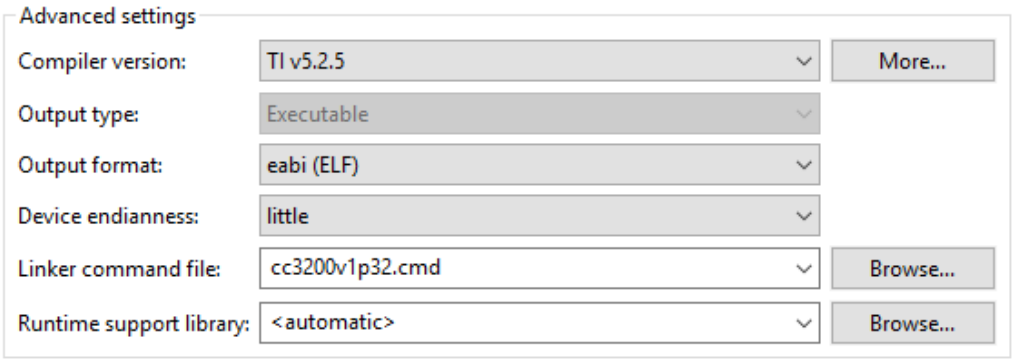
- Položku **Device** nastavte na `CC3200`.
- Pojmenujte projekt dle vlastního uvážení.

6. Dále je vhodné nastavit v projektu proměnnou, která bude reprezentovat cestu k aktuálnímu umístění `SDK` (viz obr. B.3).
   - Vlastnosti projektu.
   - **Resource** → **Linked Resorces** → **Path Variables** → **Add**.
   - Jméno proměnné není důležité, v mém případě jsem zvolil název `CC3200_SDK_ROOT`.
   - Poté již zvolte cestu ke složce s nainstalovaným aktuálním `SDK`.



Obrázek B.3: Nastavení proměnné umístění SDK v CCS

7. Další bodem je nastavení kompilátoru (viz obr. B.4).
   - Vlastnosti projektu.
   - **CCS General** → **Advanced settings**.



Obrázek B.4: Výběr kompilátoru v CCS

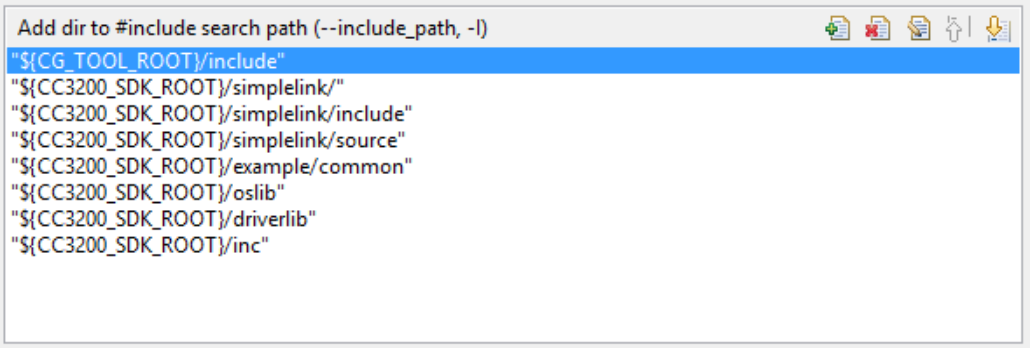
- Položku **Compiler version** nastavte na `TI v5.2.5` (nebo pokud máte již novější verzi, tak na novější).
- Položku **Linker command file** nastavte na `cc3200v1p32.cmd`.

8. Dále je nutné nastavit vlastnosti procesoru (viz obr. B.5):
   - Vlastnosti projektu.
   - **CCS Build** → **ARM Compiler** → **Processor options**.
   - Položku **Target processor version** nastavte na `7M4`.
   - Položku **Designate code state** natavte na `16`.
   - Položku **Specify floating point support** nastavte na `vfplib`.

Obrázek B.5: Vlastnosti procesoru v CCS

9. Dalším krokem je nastavení cest, kde má projekt hledat includované hlavičkové soubory ve zdrojových kódech projektu (viz obr. B.6):
   - Vlastnosti projektu.

- **CCS Build → ARM Compiler → Include options → Add dir to #include search path**.
- Přidejte následující položky (je použita dříve definovaná proměnná `CC3200_SDK_ROOT` pro umístění `SDK`):
- `"${CC3200_SDK_ROOT}/simplelink/"`,
- `"${CC3200_SDK_ROOT}/simplelink/include"`,
- `"${CC3200_SDK_ROOT}/simplelink/source"`,
- `"${CC3200_SDK_ROOT}/example/common"`,
- `"${CC3200_SDK_ROOT}/oslib"`,
- `"${CC3200_SDK_ROOT}/driverlib"`,
- `"${CC3200_SDK_ROOT}/inc"`.

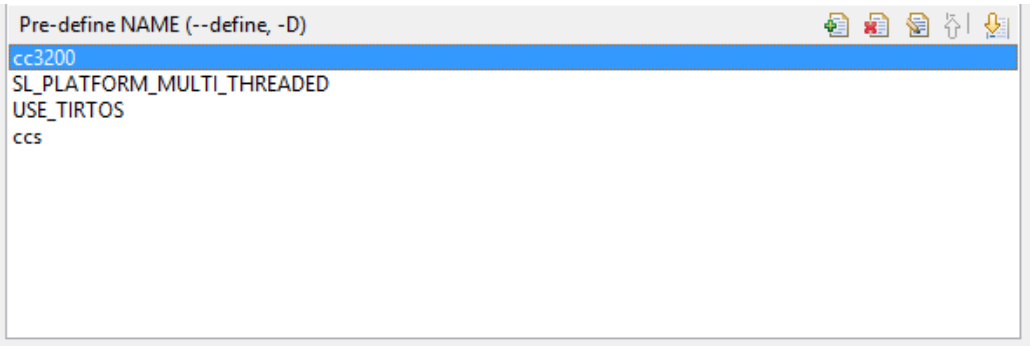


Obrázek B.6: Definice složek pro hledání hlavičkových souborů v CCS

10. Dále je potřeba zadefinovat symboly a tím zkonfigurovat chování aplikace (viz obr. B.7):
    - Vlastnosti projektu.
    - **CCS Build → ARM Compiler → Advanced Options → Predefined Symbols**.
    - Do **Pre-define NAME** je nutné vložit následující možnosti kompilátoru:
    - `cc3200` - definuje použití MCU *CC3200*,
    - `USE_TIRTOS` - definuje použití operačního systému reálného času *TI-RTOS*,

– `SL_PLATFORM_MULTI_THREADED` - tuto možnost je nutné uvést při použití kteréhokoliv operačního systému reálného času,

– `ccs` - definuje typ projektu založený nástrojem *Code Composer Studio*.



Obrázek B.7: Definice symbolů kompilátoru v CCS

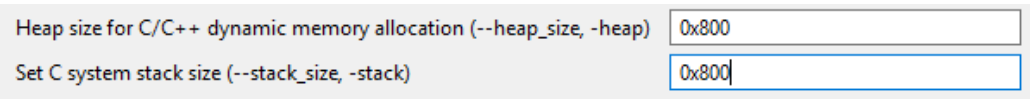
11. Dalším bodem je nastavení velikostí zásobníku a haldy, které bude aplikace používat (viz obr. B.8):

    – Vlastnosti projektu.

    – **CCS Build → ARM Linker → Basic Options**.

    – Položky **Heap size** a **Set C system stack** nastavte na hodnotu `0x800`.

    – (Hodnotu lze v případě nedostatku paměti zvýšit.)



Obrázek B.8: Nastavení velikostí zásobníku a haldy v CCS

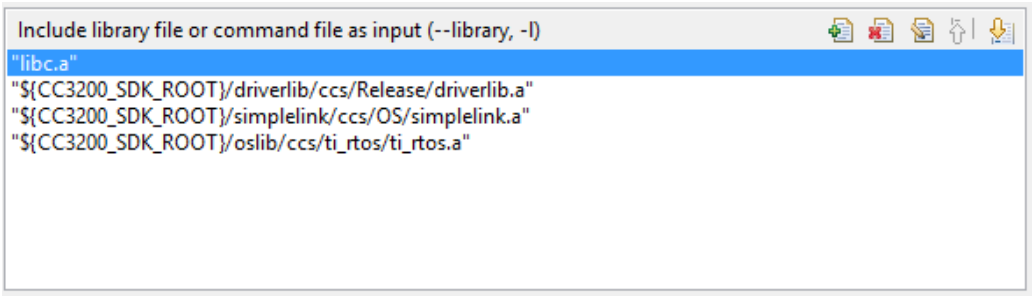
12. Dále je potřeba v nastavení projektu zahrnout knihovny (viz obr. B.9):

    – Vlastnosti projektu.

    – **CCS Builds → ARM Linker → File Search Path**.

    – Do **Include library file or command file as input** je potřeba vložit následující položky:

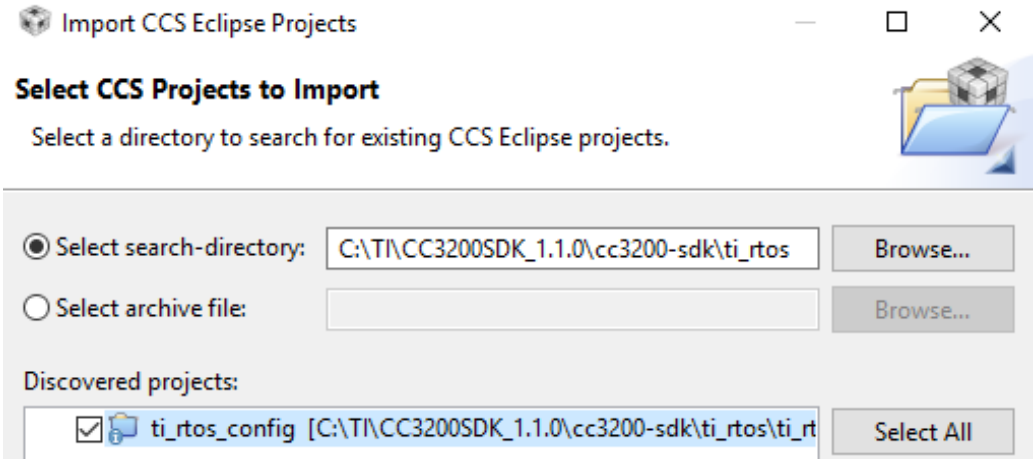
– `"${CC3200_SDK_ROOT}/driverlib/ccs/Release/driverlib.a"`,
– `"${CC3200_SDK_ROOT}/oslib/ccs/ti_rtos/ti_rtos.a"`,
– `"${CC3200_SDK_ROOT}/simplelink/ccs/OS/simplelink.a"`.



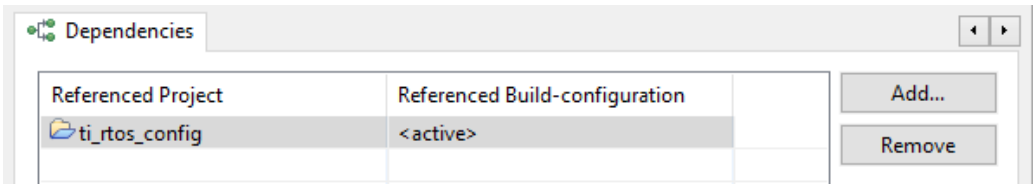
Obrázek B.9: Přidání knihoven do projektu v CCS

13. Pro využití operačního systému reálného času *TI-RTOS* je nutné naimportovat projekt `ti_rtos_config` (viz obr. B.10) a odkázat se na tento projekt v našem novém projektu (viz obr. B.11):
    – **Projects** → **Import CCS Projects...**
    – Vyberte složku `${CC3200_SDK_ROOT}/ti_rtos` (místo `${CC3200_SDK_ROOT}` vložte cestu k vašemu umístění `SDK`).
    – Zaškrtněte nalezený projekt `ti_rtos_config`.
    – Potvrďte a poté proveďte build projektu.
    – Poté vyberte vlastnosti nového projektu.
    – **CCS Build** → záložka **Dependencies** → **Add**.
    – Zvolte projekt `ti_rtos_config` a potvrďte.

14. Pro vytvoření finálního binárního souboru, který lze posléze nahrát přímo do sériové FLASH paměti je potřeba přidat příkaz, který se provede vždy na konci kompilace projektu (viz obr. B.12):
    – Vlastnosti projektu.
    – **CCS Build** → záložka **Steps**.
    – Do pole **Post-build steps** přidejte následující příkaz.
    – **(Příkaz je nutné vložit jako jeden řádek!)**



Obrázek B.10: Importování projektu *ti_rtos_config* v CCS



Obrázek B.11: Přidání odkazu na projekt v CCS

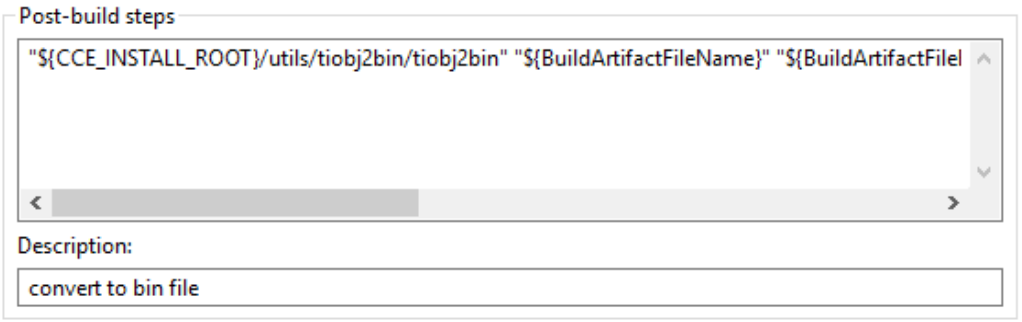
```
/* Prikaz, ktery po provedeni kompilace vytvori
 * finalni .bin soubor pro MCU CC3200
 * Prikaz se vklada do pole "Post-build steps"
 * v nastaveni projektu
 * Prikaz se vklada jako jeden radek
 */
"${CCE_INSTALL_ROOT}/utils/tiobj2bin/tiobj2bin"
"${BuildArtifactFileName}" "${BuildArtifactFileBaseName}.bin"
"${CG_TOOL_ROOT}/bin/armofd" "${CG_TOOL_ROOT}/bin/armhex"
"${CCE_INSTALL_ROOT}/utils/tiobj2bin/mkhex4bin"
```

15. Posledním krokem je vložení a nastavení konfiguračního souboru samotného zařízení (viz obr. B.13):

    – Projekt je nutné nastavit jako aktivní.

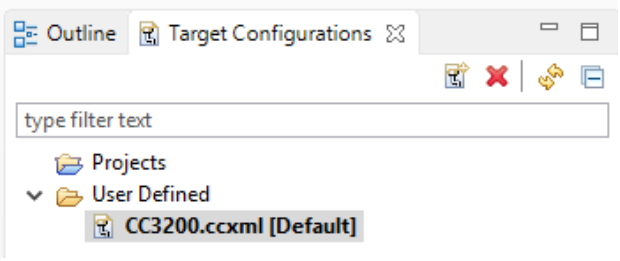
    – Z menu vyberte **View** → **Target Configuration**.

    – Vpravo se otevře nové menu, pokud složka **User Defined** obsahuje již konfigurační soubor `CC3200.ccxml` a je nastavený jako výchozí, další 2 kroky přeskočte.



Obrázek B.12: Přidání příkazu volaného po každé kompilaci v CCS

– Po kliknutí pravým tlačítkem na složku **User Defined** vyberte **Import Target Configuration**.
– Ze složky `"$CC3200_SDK_ROOT/tools/css"` vyberte soubor `cc3200.ccxml` a nastavte jako výchozí.
– Otevřete konfigurační soubor, klikněte na záložku **Advanced**, vyberte připojení **Cortex_M4_0** a v položce **Port** změňte hodnotu na **JTAG**.



Obrázek B.13: Vložení konfiguračního souboru zařízení v CCS

C Ukázkový zdrojový kód pro CCS

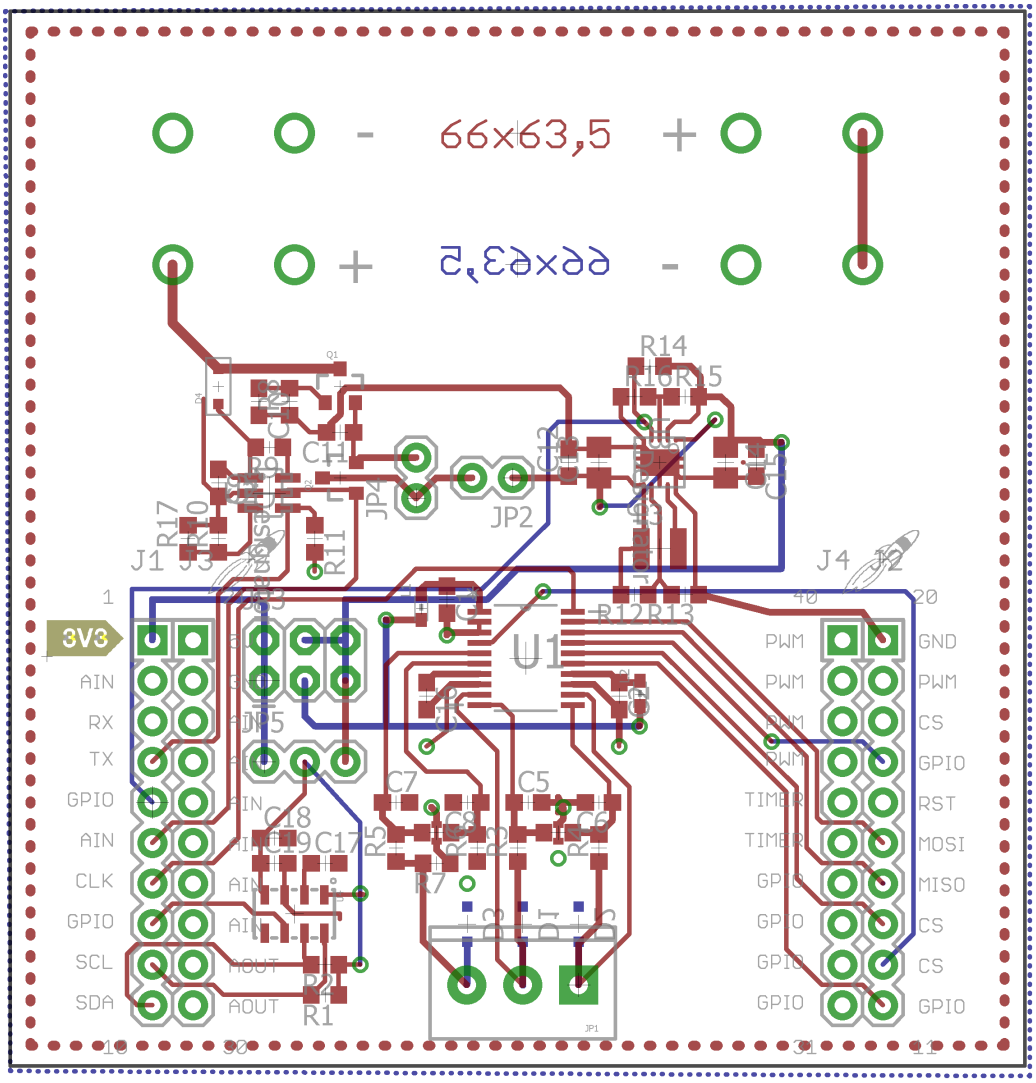
```
#include "hw_types.h"
#include "rom_map.h"
#include "prcm.h"
#include "gpio.h"
#include "hw_memmap.h"
#include "utils.h"
#include "hw_gpio.h"
#include "pin.h"
#include "uart.h"

#define RED 1
#define YELLOW 2
#define GREEN 3

// zkonfiguruje pouzite piny a UART terminal
static void setup()
{
  MAP_PRCMPeripheralClkEnable(PRCM_GPIOA1, PRCM_RUN_MODE_CLK);
  MAP_PRCMPeripheralClkEnable(PRCM_UARTA0, PRCM_RUN_MODE_CLK);

  MAP_PinTypeGPIO(PIN_64, PIN_MODE_0, false);
  MAP_GPIODirModeSet(GPIOA1_BASE, 0x2, GPIO_DIR_MODE_OUT);

  MAP_PinTypeGPIO(PIN_01, PIN_MODE_0, false);
  MAP_GPIODirModeSet(GPIOA1_BASE, 0x4, GPIO_DIR_MODE_OUT);

  MAP_PinTypeGPIO(PIN_02, PIN_MODE_0, false);
  MAP_GPIODirModeSet(GPIOA1_BASE, 0x8, GPIO_DIR_MODE_OUT);

  MAP_PinTypeUART(PIN_53, PIN_MODE_9);
  MAP_PinTypeUART(PIN_57, PIN_MODE_3);

  MAP_UARTConfigSetExpClk(UARTA0_BASE,
    MAP_PRCMPeripheralClockGet(PRCM_UARTA0),
    115200, (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
    UART_CONFIG_PAR_NONE));
}

// vypise zpravu na UART
static void printMessage(const char * message) {
  while(*message!='\0')
```

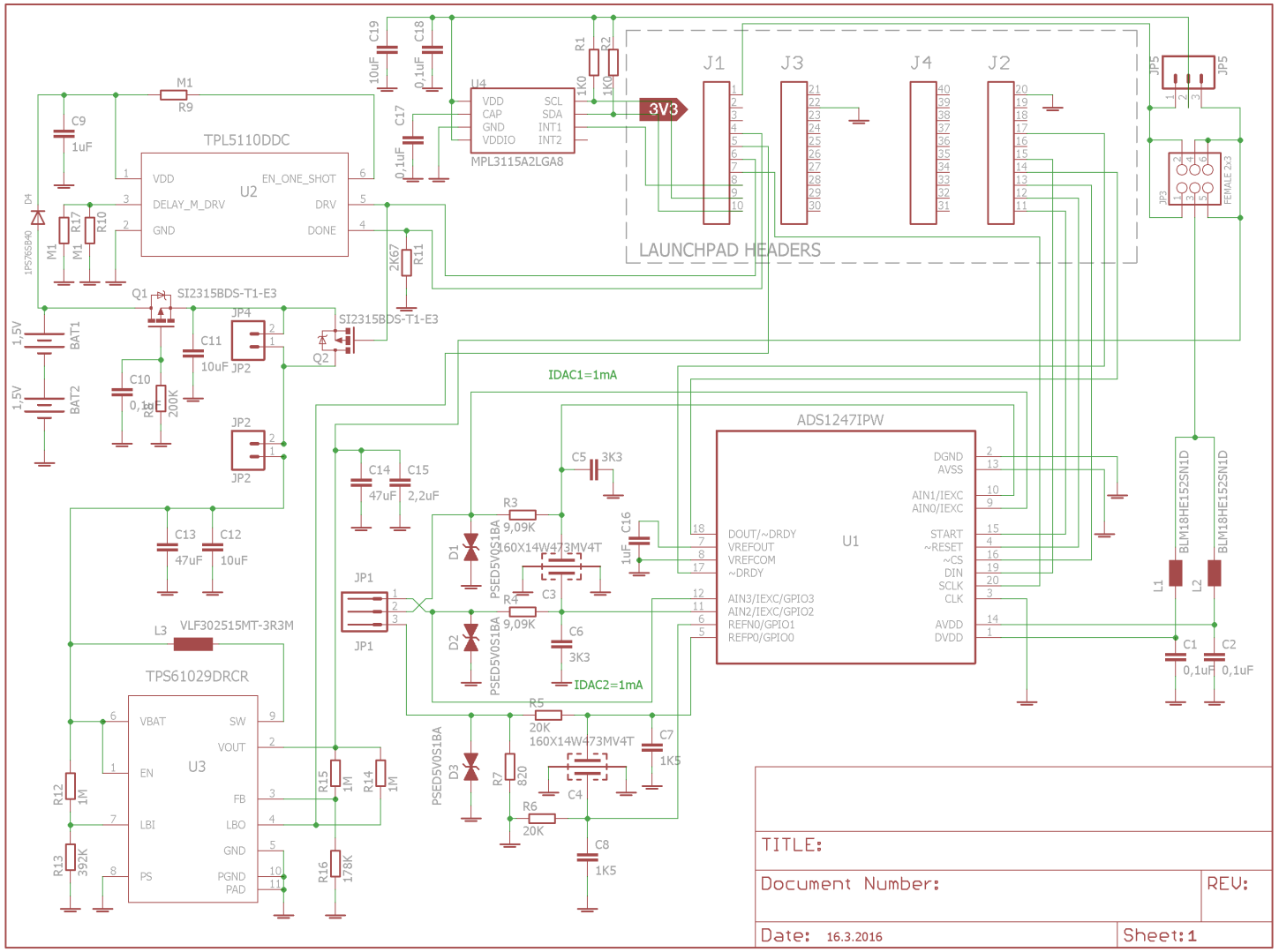
```
  {
    MAP_UARTCharPut(UARTA0_BASE,*message++);
  }
}

// rozsviti LED, pocka, zhasne LED, pocka
static void blickLed(unsigned int led) {
  MAP_GPIOPinWrite(GPIOA1_BASE, 1 << led, 1 << led);
  MAP_UtilsDelay(4000000);
  MAP_GPIOPinWrite(GPIOA1_BASE, 1 << led, 0 << led);
  MAP_UtilsDelay(4000000);
}

// funkce main
main()
{
  // inicializace MCU
  PRCMCC3200MCUInit();
  // konfigurace pinu a UART terminalu
  setup();

  while(1) {
    blickLed(RED);
    blickLed(YELLOW);
    blickLed(GREEN);
    printMessage("All␣LEDs␣blicked␣once!\r\n");
  }
}
```

D Ukázkový zdrojový kód pro Energii

```
#define RED RED_LED
#define YELLOW YELLOW_LED
#define GREEN GREEN_LED

// setup smycka (probehne jednou na zacatku programu)
void setup() {
  // inicializace LED diod
  pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT);
  pinMode(YELLOW, OUTPUT);
  // inicializace UART terminalu
  Serial.begin(115200);
}

// rozsviti LED, pocka, zhasne LED, pocka
static void blickLed(unsigned int led) {
  digitalWrite(led, HIGH);
  delay(300);
  digitalWrite(led, LOW);
  delay(300);
}

// hlavni smycka pro beh programu
void loop() {

  blickLed(RED);
  blickLed(YELLOW);
  blickLed(GREEN);
  Serial.print("All␣LEDs␣blicked␣once!\r\n");
}
```

E Plošný spoj a elektrické schéma navržené destičky




TPL5110DDC
U2
VDD
EN_ONE_SHOT
DELAY_M_DRV
DRV
GND
DONE
MPL3115A2LGA8
U4
VDD
SCL
CAP
SDA
GND
INT1
VDDIO
INT2
3V3
LAUNCHPAD HEADERS
J1
J3
J4
J2
JP5
JP3
FEMALE 2X3
SI2315BDS-T1-E3
Q1
Q2
BAT1
BAT2
1,5V
IDAC1=1mA
IDAC2=1mA
ADS1247IPW
U1
DGND
AVSS
AIN1/IEXC
AIN0/IEXC
DOUT/~DRDY
VREFOUT
VREFCOM
~DRDY
START
~RESET
~CS
DIN
SCLK
CLK
AIN3/IEXC/GPIO3
AIN2/IEXC/GPIO2
REFN0/GPIO1
REFP0/GPIO0
AVDD
DVDD
TPS61029DRCR
U3
VBAT
SW
VOUT
EN
FB
LBI
LBO
GND
PS
PGND
PAD
VLF302515MT-3R3M
L3
L1
L2
BLM18HE152SN1D
PSED5V0S1BA
160X14W473MV4T
TITLE:
Document Number:
REV:
Date: 16.3.2016
Sheet: 1

F Konfigurační soubory pro generování certifikátů

Konfigurační soubor pro certifikační autoritu `CAConf.cnf`:

```
[ ca ]
default_ca      = local_ca

[ local_ca ]
dir             = /home/mint/cert/CA
certificate     = $dir/cacert.pem
database        = $dir/index.txt
new_certs_dir   = $dir/signedcerts
private_key     = $dir/private/cakey.pem
serial          = $dir/serial

default_crl_days        = 365
default_days            = 1825
default_md              = sha1

policy          = local_ca_policy
x509_extensions = local_ca_extensions

copy_extensions = copy

[ local_ca_policy ]
commonName              = supplied
stateOrProvinceName     = supplied
countryName             = supplied
emailAddress            = supplied
organizationName        = supplied
organizationalUnitName  = supplied

[ local_ca_extensions ]
basicConstraints        = CA:false

[ req ]
default_bits    = 2048
default_keyfile = /home/mint/cert/CA/private/cakey.pem
default_md      = sha1

prompt                  = no
distinguished_name      = root_ca_distinguished_name
x509_extensions         = root_ca_extensions
```

```
[ root_ca_distinguished_name ]
commonName               = Self Signed CA
stateOrProvinceName      = Czech Republic
countryName              = CZ
emailAddress             = lichy@students.zcu.cz
organizationName         = ZCU
organizationalUnitName   = FAV

[ root_ca_extensions ]
basicConstraints         = CA:true
```

Konfigurační soubor pro server `ServerConf.cnf`:

```
[ req ]
prompt                   = no
distinguished_name       = server_distinguished_name
req_extensions           = v3_req

[ server_distinguished_name ]
commonName               = Server certificate
stateOrProvinceName      = Czech Republic
countryName              = CZ
emailAddress             = lichy@students.zcu.cz
organizationName         = ZCU
organizationalUnitName   = FAV

[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
```

G Obsah přiloženého CD

- `readme.txt` ... stručný popis obsahu CD
- `bin` ... adresář se spustitelnou formou implementace
  - `server` ... adresář se serverovou aplikací
    - `Windows` ... adresář se spustitelným programem pro Windows
    - `Linux` ... adresář se spustitelným programem pro Linux
  - `mcu` ... adresář s aplikací pro MCU CC3200
    - `mcuimng.bin` ... binární soubor pro MCU
    - `cc3200.conf` ... vzor konfigurace pro MCU
- `certs` ... adresář se vzorovými certifikáty
  - `server` ... adresář se serverovým certifikátem
  - `mcu` ... adresář s certifikátem pro MCU
- `src`
  - `impl` ... zdrojové kódy implementace
  - `thesis` ... zdrojová forma práce ve formátu LaTeX
  - `schema` ... schéma přídavné destičky pro nástroj *Eagle*
- `text` ... text práce
  - `DP_Lichý_Lukáš_2016.pdf` ... text práce ve formátu PDF
- `Poster` ... adresář obsahující poster
- `LICENSE.txt` ... soubor s licencemi použitých knihoven