Západočeská univerzita v Plzni
Fakulta elektrotechnická
Katedra aplikované elektroniky a telekomunikací

# Bakalářská práce

# Implementace rozšíření KETCube pro Arduino IDE

Plzeň 2021 Tomáš Hák

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta elektrotechnická

Akademický rok: 2019/2020

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Tomáš HÁK** |
| Osobní číslo: | **E17B0072P** |
| Studijní program: | **B2612 Elektrotechnika a informatika** |
| Studijní obor: | **Elektronika a telekomunikace** |
| Téma práce: | **Implementace rozšíření KETCube pro Arduino IDE** |
| Zadávající katedra: | **Katedra aplikované elektroniky a telekomunikací** |

## Zásady pro vypracování

1. Seznamte se s dokumentací open-hardware projektů Arduino a KETCube.
2. Navrhněte rozšiřující balíček pro Arduino IDE, jež umožní tvorbu softwarových rozšíření KETCube v Arduino IDE.
3. Po dohodě s vedoucím práce navhněte rozšiřující desku (DPS) pro KETCube.
4. Demonstrujte funkčnost navrženého řešení, ovládací software pro DPS implementujte v Arduino IDE.

Rozsah bakalářské práce: **30 – 40 stran**
Rozsah grafických prací: **podle doporučení vedoucího**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:


1. BOSWELL, Dustin; FOUCHER, Trevor: The Art of Readable Code: Simple and Practical Techniques for Writing Better Code. „O'Reilly Media, Inc.", 2011.
2. ZÁHLAVA, Vít: Návrh a konstrukce desek plošných spojů: principy a pravidla praktického návrhu. BEN-technická literatura, 2010.
3. Vývojářská dokumentace projektu Arduino (dostupné online): https://github.com/arduino/Arduino/wiki
4. Dokumentace projektu KETCube (dostupné online): https://github.com/SmartCAMPUSZCU/KETCube-docs


Vedoucí bakalářské práce: **Ing. Jan Bělohoubek**
Katedra technologií a měření

Datum zadání bakalářské práce: **4. října 2019**
Termín odevzdání bakalářské práce: **11. června 2020**

Západočeská univerzita v Plzni
Fakulta elektrotechnická
1

L.S.

**Prof. Ing. Zdeněk Peroutka, Ph.D.**
děkan

**Doc. Dr. Ing. Vjačeslav Georgiev**
vedoucí katedry

V Plzni dne 4. října 2019

# Prohlášení

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

V Plzni dne 26. května 2021

Tomáš Hák

# Abstract

This thesis is about the support of the KETCube device in the Arduino IDE development environment. The second part shows the practical usage of the implementation on a physical device. This thesis is concerned with creating a software package for further usage in Arduino IDE. This package is used for software development focused on the KETCube device. The purpose of the KETCube device and its software package is to enable easier and user-friendly development software for IoT devices.



# Abstrakt

Tato práce se zabývá softwarovou implementací zařízení KETCube do vývojového prostředí Arduino IDE. Součástí práce je názorná ukázka použití dané implementace na fyzickém zařízení. Implementace zahrnuje vytvoření softwarového balíčku pro vývojové prostředí Arduino IDE. Tento balíček se používá k vývoji nového softwaru zařízení KETCube. Zařízení KETCube s přiloženým softwarovým balíčkem slouží k snadnějšímu programování nových aplikací ve vývojovém prostředí Arduino IDE, které je uživatelsky přívětivější. Zařízení se uplatňuje nejvíce v oblasti Internetu věcí (IoT).

# Obsah

# Seznam obrázků

# Seznam symbolů a zkratek

| | |
|---|---|
| **KET** | Katedra aplikované elektroniky a telekomunikací |
| **IDE** | Integrated Development Environment |
| **IoT** | Internet of Things (česky Internet věcí) |
| **RTC** | Real Time Clock |
| **ADC** | Analog to Digital Converter |
| **I²C** | Inter-Integrated Circuit (nebo IIC) |
| **SPI** | Serial Peripheral Interface |
| **USART** | Universal Synchronous / Asynchronous Receiver and Transmitter (česky Univerzální synchronní / asynchronní příjmač a vysílač) |
| **DMA** | Direct Memory Access |
| **AES** | Advanced Encryption Standard |
| **OS** | Operační systém |
| **RHT** | Relative Humidity and Temperature |
| **DPS** | Deska plošného spoje (angl. PCB) |
| **PWM** | Pulse Width Modulation |
| **GPIO** | General Purpose Input/Output |
| **MSB** | Most Significant Bit/Byte |
| **LSB** | Least Significant Bit/Byte |

# 1 Úvod

V posledních letech se vývoj v oblasti nových technologií velice zrychlil. Tato skutečnost zahrnuje i rychlý vývoj v oblasti Internetu věcí. Tyto technologie umožňují uživatelům propojit více senzorů v oblasti fyzického zabezpečení objektů, měřících zařízení a zařízení používaná k automatizaci práce člověka a jejich vzdálené kontrole.

Do oblasti Internetu věcí zahrnujeme např. zabezpečení objektů, inteligentní senzory, měření vlastností půdy v zemědělství, inteligentní spotřebiče, inteligentní domácnosti, ovládání hlasem, drony a jejich přidružené technologie a mnoho dalších zařízení. Tato technologie umožňuje člověku získat potřebné znalosti v rozhodování a větší kontrolu nad věcmi, které jsou kolem nás, prostřednictvím chytrých technologií jako je např. mobilní telefon, chytré hodinky a podobně. Zároveň tato technologie nabízí možnosti automatizace opakujících se procesů, které jsou do IoT sítě připojeny vzdáleně.

S tvorbou těchto chytrých technologií přichází také požadavek na nový software, který bude mozkem použitého zařízení. Lidí, kteří se do tohoto odvětví začínají angažovat, přibývá, a proto je důležité klást větší důraz na uživatelskou přívětivost vývojových prostředí. Tyto softwarové technologie jsou využívány čím dál tím větší skupinou lidí, kteří se s programováním setkávají poprvé. Proto vznikají nové projekty, které se snaží učit veřejnost širokému využití těchto technologií a umožnit růstu jejich kreativity. A jednou z těchto technologií se zabývám i v této práci.

Tato práce se věnuje tvorbě softwarového balíčku zařízení KETCube pro použití ve vývojovém prostředí Arduino IDE. Účelem tohoto balíčku je snadnější a přívětivější psaní nových softwarových aplikací pro zařízení KETCube. Funkční softwarový balíček je první částí této práce a jeho ověření se bude provádět aplikací na fyzickém zařízení. Výběru a návrhu fyzického zařízení a ověření funkčnosti balíčku se věnuje druhá část této práce.

Cílem této práce je umožnit novým uživatelům programování zařízení KETCube vyvinutého na Západočeské univerzitě v Plzni ve vývojovém prostředí Arduino IDE.

# 2 Prerekvizity

## 2.1 Prostředí Arduino

Vývojové prostředí Arduino je softwarová platforma, která byla vyvinuta v institutu Ivrea Interaction Design. Tato platforma je Open Source a její záměr je jednoduché a rychlé programování, prototypování a testování zařízení s mikrokontroléry [1].

Především je zaměřeno na jednoduché užívání, které je přívětivé jak nováčkům v oboru programování a elektrotechniky, tak může sloužit k prototypování složitějších aplikací.

Dříve bylo toto prostředí využíváno k vyvíjení softwaru pro desky platformy Arduino, ale později se vývojové prostředí Arduino IDE s narůstajícím množstvím nových desek od jiných výrobců rozrostlo a nyní podporuje mnoho zařízení s mikrokontroléry.

Prostředí Arduino IDE umožňuje uživatelům psát kód, který mohou posléze zkompilovat. Pokud se v napsaném kódu nevyskytují chyby, je možné zkompilovaný kód nahrát do zařízení. Pro debugování[1] lze využít vestavěnou funkci Serial monitor, která odposlouchává sériovou komunikaci na portu, ke kterému je zařízení s mikrokontrolérem připojeno.

Standardní metoda debugování zahrnuje výčet paměti zařízení v daném okamžiku chodu programu, který je definován bodem zastavení programu tzv. *breakpoint*. To je pro znalejší uživatele samozřejmě informativnější, ale záměrem vývojového prostředí je především jednoduchost a přehlednost, takže tuto funkcionalitu Arduino IDE neobsahuje. To ztěžuje hledání chyb v programu.

Součástí prostředí Arduino IDE je možnost automatického vyhledávání aktualizací instalovaných balíčků, které obsahují knihovny souborů s definicemi použitelných funkcí. Zároveň dodávají Arduino IDE také funkce a informace jak napsaný kód zkompilovat a do daného zařízení také nahrát.

[1] debugování (z *angl.* slova *debug*) - metoda používaná k hledání chyb v kódu

## 2.2 Platforma KETCube

KETCube je prototypovací platforma vyvinutá na Katedře aplikované elektroniky a telekomunikací Fakulty elektrotechnické (*zkr.* KET FEL) Západočeské univerzity v Plzni [2]. Tato platforma se skládá z hlavní desky, bateriové desky a přídavných modulů, kterými lze přidat další senzory a čidla. Celé zařízení lze umístit do boxu (obr. 2.2).

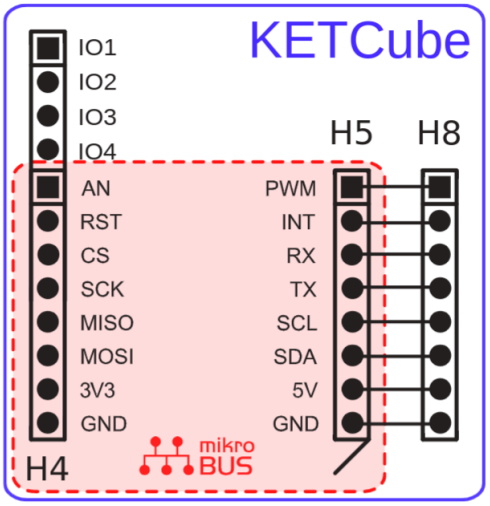
Hlavní deska je osazena procesorem řady STM32L0 [6], který je energeticky nejúspornější řadou procesorů od společnosti ST. Procesor je vybaven vnitřním oscilátorem, díky němuž není potřeba vnějšího krystalového oscilátoru. Hlavními parametry procesoru jsou 32-bitová architektura, až 192 kB Flash paměti, až 20 kB RAM paměti, až 6 kB EEPROM paměti a napájecí napětí snížené na 1,65 V. Procesor je dále vybaven 5 zdroji hodin, RTC s kalibrací, 12-bit ADC převodníkem, širokou škálou komunikačních rozhraní (např.: $I^2C$, SPI, USART), několika 16-bitovými časovači, DMA přenosem a šifrováním AES-128.

Na hlavní desce KETCube lze rovněž nalézt senzor vlhkosti a teploty HDC1080 RHT [7] a anténu (popř. konektor SMA), kterou lze vysílat naměřená data. Samotný modul s procesorem a anténou lze napájet micro USB konektorem nebo deskou s baterií.

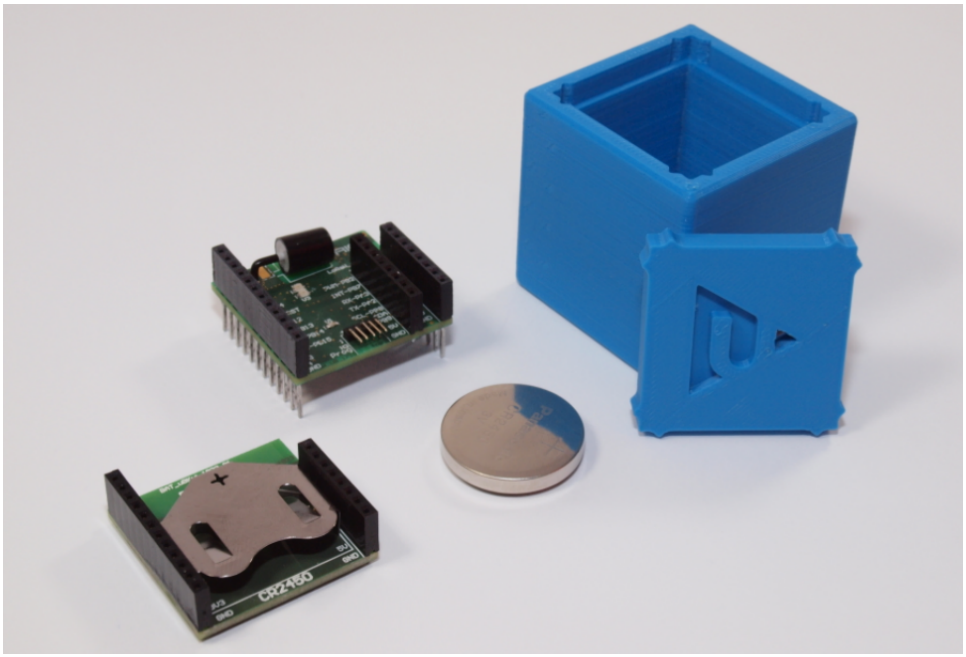
Anténa využívá frekvence 868 MHz a 915 MHz sítě LoRaWAN (Class A a Class C), kterými lze komunikovat až na desítky kilometrů s širokou sítí přijímacích zařízení. Nejčastěji se tato síť používá k zachycování dat od různých senzorů a čidel, která zajišťují bezpečnost i pohodlí člověka.

Koncept celé platformy KETCube je v jeho široké škálovatelnosti. Hlavní deska je osazená konektorem standardu mikroBUS™ [4] a proprietárním konektorem KETCube (obr. 2.1). K těmto rozšiřujícím konektorům lze připojit vlastní desky, které splňují rozměrové standardy kontaktů.

Zařízení KETCube je vybaveno sériovým terminálem s možností zapínat / vypínat různé moduly a přiřazovat jim závažnost vypisovaných informací (NONE, ERROR, INFO, DEBUG). KETCube terminál je připojen na sériové lince USART1. Tento terminál slouží k zjišťování stavu modulů, jejich chyb či k nastavení periodicity odesílaných dat. Všechny příkazy je možno zobrazit napsáním příkazu *help* v terminálu.




Obrázek 2.1: KETCube pinout diagram [2]



Obrázek 2.2: Platforma KETCube s boxem [2]

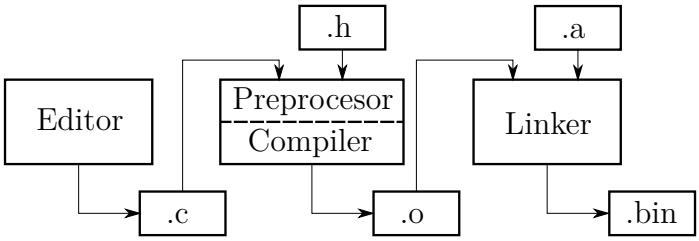
## 2.3 Proces sestavení zdrojových kódů

Každý program začíná v editoru jako zdrojových kód, který je potřeba přeložit do formátu, kterému rozumí dané zařízení. V programovacím jazyce C končí soubor programu koncovkou **.c** a přidružené hlavičkové soubory koncovkou **.h**. Tyto soubory jsou čitelné a přenositelné mezi všemi operačními systémy. Pro daný procesor či mikrokontrolér je potřeba tyto soubory přelo-

žit patřičným předkladačem. Postup překladu je znázorněn na obrázku č. 2.3.

Překladač provádí překlad zdrojového kódu do strojového kódu zařízení. V první fázi překladu ještě nejsou přesně známy všechny absolutní adresy proměnných a použitých funkcí, protože mohou být definovány v knihovním souboru **.a** (nebo **.lib**) a nebo v jiném objektu **.o**. Tyto soubor připojuje tzv. Linker, který dodá všem relativním adresám jejich absolutní adresy. Výsledkem linkeru je výsledný program v požadovaném formátu. Tento soubor je veliký blok hexadecimálních čísel, které se zapisují přímo do programové paměti mikrokontroléru.

V případě mikrokontroléru STM32L0 použitého v zařízení KETCube není pokaždé kompilaci ve vývojovém prostředí Arduino IDE potřeba znovu kompilovat celý zdrojový kód tzv. core, a proto je vhodné předkompilovat celé jádro KETCube do knihovního archivu **.a**. Tento soubor se při provedení kompilace připojí k novému objektovému souboru **.o** (nebo **.obj**) a po přilinkování knihoven se vytvoří soubor **.bin**, který obsahuje blok paměti a je přímo nahrán do paměti mikrokontroléru.




Obrázek 2.3: Postup kompilace zdrojových kódů v jazyce C [3]

# 3 Implementace rozšíření pro Arduino IDE

Aby uživatel mohl využívat k programování zařízení KETCube vývojové prostředí Arduino IDE je potřeba vytvořit balíček, který se do prostředí Arduino IDE nainstaluje. Tento balíček musí splňovat kritéria, která vývojové prostředí vyžaduje.

Kompletní balíček lze nalézt ve veřejném repositáři[1]. Balíček byl testován v operačním systému Ubuntu 20.04 s vývojovým prostředím Arduino IDE verze 1.8.13.

## 3.1 Implementace balíčku

Balíček vývojového prostředí nazvaný **KETCube package** je rozdělen na dvě části. Tyto části tvoří složky *hardware/* a *tools/* (obr. 3.1). Složka *hardware/* obsahuje definice cest, příkazů a jejich argumentů. Zatímco složka *tools/* obsahuje software potřebný ke kompilaci, linkování a sestavení binárního souboru **.bin**.

Ve složce *KETCube/harware/* lze nalézt odstupňované podsložky *stm32/* a v ní *0.2.0/*. Tyto vnořené složky specifikují architekturu použitého procesoru *stm32* a verzi balíčku KETCube *0.2.0*. Toto členění není nezbytně nutné, ale umožňuje přidání dalších desek s jinou architekturou a případné nové verze jejich definic. Dělení složek udržuje přehlednost celého balíčku.

Další členění je samotným jádrem balíčku KETCube. Toto jádro dále zahrnuje složku *cores/*, která obsahuje hlavičkové soubory využívané při programování v Arduino IDE. Dále složky *include/* a *system/* obsahující zbylé hlavičkové soubory potřebné při kompilaci celého projektu. Zde jsou především deklarace funkcí, které využívá softwarové jádro *libKETCube.a*.

V balíčku je obsažen také program *stm32flash* pro nahrání našeho projektu (souboru **.bin**) do fyzického zařízení KETCube. Tento program je ve

[1] `https://github.com/Tomhauscz/KETCube_package`

složce *tools/linux/* a *tools/win/*. Toto dělení je důležité z hlediska použitého operačního systému. Poslední složka *variants/* obsahuje zmíněné softwarové jádro KETCube *libKETCube.a* a linkovací skript *STM32L072CZYx_FLASH.ld*.

Podle dokumentace vývojového prostředí Arduino IDE [1] musí balíček obsahovat textové soubory *boards.txt* a *platform.txt*. Obsah těchto souborů bude dále rozebrán.

### 3.1.1 Soubor boards.txt

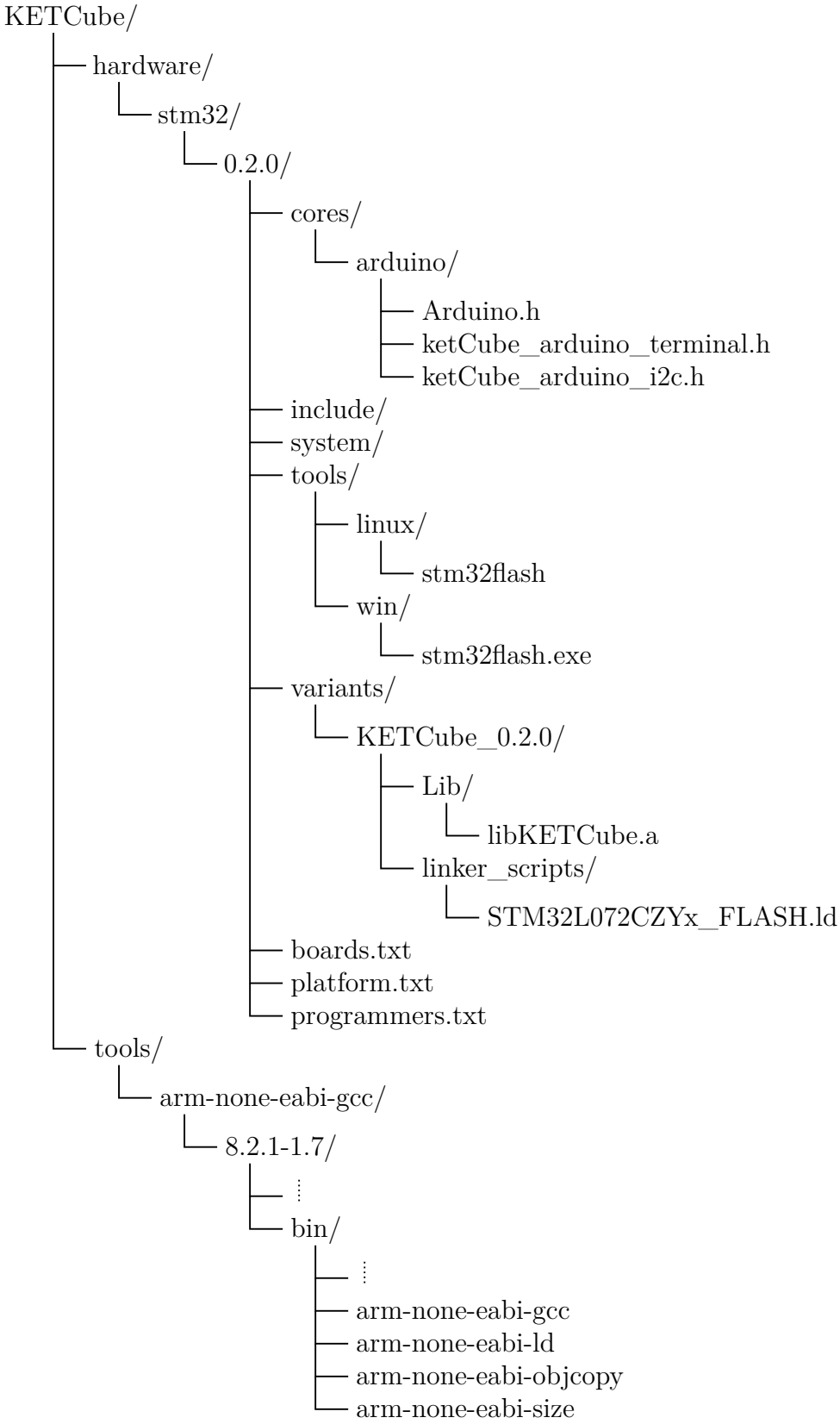
V souboru **boards.txt** (obr. 3.2) jsou definice proměnných desky KETCube. Tyto definice mají hierarchicky strukturované proměnné, které jsou odděleny tečkou. Většina proměnných začíná názvem desky, ke které se vztahují. V tomto případě je to hlavní deska platformy KETCube nazvaná `mainboard`. Dalším dělením je oblast, kde bude daná proměnná použita.

Množina proměnných `upload` definuje proměnnou `file_type`, nebo-li požadovaný typ souboru (v tomto případě binární soubor). Dále také `maximum_size` a `maximum_data_size`, které potřebuje prostředí Arduino IDE k výpočtu procentuálního využití paměti programu a paměti dat.

V množině `build` je definice proměnných týkajících se např. varianty desky, architektury procesoru, endianity[2], umístění a název linkovacího skriptu a také všechny cesty k hlavičkovým souborům, potřebných při kompilaci projektu.

Poslední množina proměnných odstupňovaných `menu.upload_method.stm32flashMethod` definuje metodu nahrávání zkompilovaného projektu do zařízení. Zde je možné definovat několik metod, které dané zařízení používá pro nahrávání kódu do paměti. V případě zařízení KETCube jsou k dispozici metody `stm32flash` a `st-flash`. Protože metoda `st-flash` potřebuje pro nahrávání externí zařízení ST-LINK, není v balíčku uvedena jako jedna z možností.

[2] endianita - pořadí bytů vícebytových proměnných




Obrázek 3.1: Stromový výpis složek a souborů balíčku KETCube

```
menu.upload_method=Upload Method

mainboard.name=KETCube_mainboard
mainboard.build.core=arduino

mainboard.upload.file_type=bin
mainboard.upload.maximum_size=196608
mainboard.upload.maximum_data_size=217088

mainboard.build.variant=KETCube_0.2.0
mainboard.build.mcu=cortex-m0plus
mainboard.build.march=armv6-m
mainboard.build.endianness=little-endian
mainboard.build.board=mainboard
mainboard.build.vect_flags=-DUSER_ADDR_ROM=0x08000000
mainboard.build.extra_flags=
mainboard.build.ldscript=
   linker_scripts/STM32L072CZYx_FLASH.ld
mainboard.build.libdir=Lib
mainboard.build.variant_system_include=
   -I{runtime.platform.path}/include/Drivers/
   STM32L0xx_HAL_Driver/Inc
...

...
   -I{runtime.platform.path}/include/Drivers/KETCube/core
   -I{runtime.platform.path}/include/Drivers/KETCube/modules
   -I{runtime.platform.path}/include/Projects/inc

mainboard.menu.upload_method.stm32flashMethod=
   STM32Flash (Serial bootloader)
mainboard.menu.upload_method.stm32flashMethod.upload.
   protocol=0
mainboard.menu.upload_method.stm32flashMethod.upload.
   options=-i "-dtr&-rts,rts," -v -g 0x0
mainboard.menu.upload_method.stm32flashMethod.upload.tool=
   stm32flash
```

Obrázek 3.2: Výpis souboru **boards.txt**

### 3.1.2 Soubor platform.txt

Soubor **platform.txt** (obr. 3.3 a 3.4) obsahuje informace o použitém kompilátoru. Příkaz samotného kompilátoru vyžaduje mnoho parametrů, které jsou zde také definovány. Tyto parametry zahrnují informace např. o standardu překladače (std=gnu99) a jsou zde použity proměnné ze souboru **boards.txt**. Je zde také uvedena cesta ke kompilátoru C++ programů s patřičnými parametry použití. Avšak projekt KETCube je převážně napsán v jazyce C.

Dále lze zde nalézt příkaz pro výpočet výsledné velikosti zkompilovaného programu a příkazy pro převod souboru **.elf** na binární (**.bin**) či hexadecimální (**.hex**) formát. Následují šablony pro sestavení příkazů kompilace, linkování a tvorby výsledného souboru formátu **.bin**. Tyto šablony jsou rozsáhlé, a proto nejsou v ukázce uvedeny.

Na tyto šablony navazuje nastavení preferovaného výstupu zkompilovaného kódu. Zde je použit binární formát výstupu. Výstupní soubor je vhodné před nahráváním dočasně uložit. Cesta k dočasně uloženému souboru je v proměnné `recipe.output.tmp_file`. Soubor je také uložen do složky s definovanou cestou v `recipe.output.save_file`. Nakonec je spočítána výsledná velikost zkompilovaného projektu, který lze nahrát do zařízení KETCube.

Na konci tohoto souboru se nachází způsob nahrání projektu do KETCube zařízení. Jedinou možností je nahrávání pomocí STM32Flash[3] programu. Proměnná `tools.stm32flash.upload.pattern` ukazuje vyvolání příkazu při nahrávání. Parametr *-e 1535* před nahráním vymaže obsah flash paměti zařízení. Za parametrem *-w* následuje cesta k binárnímu souboru vytvořeném při kompilaci projektu. Následují další volitelné parametry, jako je např. komplexnější výstup příkazu. Posledním povinným parametrem je komunikační port, na kterém je zařízení KETCube připojeno k počítači.

[3] `https://sourceforge.net/p/stm32flash/wiki/Home/`

```
name=KETCube (ARM 32-bit) boards
version=0.2.0

# ================ Compile variables ===============

compiler.warning_flags=-w
compiler.warning_flags.none=-w
compiler.warning_flags.default=
compiler.warning_flags.more=-Wall
compiler.warning_flags.all=-Wall -Wextra

compiler.path={runtime.tools.arm-none-eabi-gcc.path}/bin/
compiler.c.cmd=arm-none-eabi-gcc
compiler.c.flags=-x c -std=gnu99 -c -Wall -Wno-missing-braces
   -g -mthumb -mcpu={build.mcu} -Os -march={build.march}
   -m{build.endianness} "-T{build.variant.path}/
   {build.ldscript}" -g3 -DSTM32L082xx -DUSE_B_L082Z_KETCube
   -DUSE_HAL_DRIVER -DREGION_EU868
compiler.c.elf.cmd=arm-none-eabi-gcc
compiler.c.elf.flags=-mcpu=cortex-m0 -march=armv6-m
   "-T{build.variant.path}/{build.ldscript}" -Wl,
   -Map={build.path}/{build.project_name}.map,
   --gc-sections -mthumb -mfloat-abi=soft -specs=nano.specs
   -specs=nosys.specs -lc -lrdimon -u _printf_float

compiler.cpp.cmd=arm-none-eabi-g++
compiler.cpp.flags=-x c -std=gnu99 -c -Wall
   -Wno-missing-braces -g -mthumb -mcpu={build.mcu} -Os
   -march={build.march} -m{build.endianness}
   "-T{build.variant.path}/{build.ldscript}" -g3
   -DSTM32L082xx -DUSE_B_L082Z_KETCube
   -DUSE_HAL_DRIVER -DREGION_EU868
compiler.ar.cmd=arm-none-eabi-ar
compiler.ar.flags=rcs
compiler.size.cmd=arm-none-eabi-size


compiler.elf2hex.bin.flags=-O binary
compiler.elf2hex.hex.flags=-O ihex
compiler.elf2hex.cmd=arm-none-eabi-objcopy
...
```

Obrázek 3.3: Počáteční část souboru **platform.txt**

```
...
build.preferred_out_format=bin

## Save output
recipe.output.tmp_file={build.project_name}.
   {build.preferred_out_format}
recipe.output.save_file={build.project_name}.
   {build.variant}.{build.preferred_out_format}

## Compute size
recipe.size.pattern="{compiler.path}{compiler.size.cmd}"
   -A "{build.path}/{build.project_name}.elf"
recipe.size.regex=\.text\s+([0-9]+).*

# Uploader tools
# --------------

# ======== STM32FLASH ========

tools.stm32flash.cmd=stm32flash
tools.stm32flash.cmd.windows=stm32flash.exe
tools.stm32flash.path={runtime.platform.path}/tools/linux
tools.stm32flash.path.windows=
   {runtime.platform.path}/tools/win
tools.stm32flash.path.macosx=
   {runtime.platform.path}/tools/macosx
tools.stm32flash.port={serial.port.file}
tools.stm32flash.port.linux=/dev/{serial.port.file}
tools.stm32flash.port.macosx=/dev/{serial.port.file}
tools.stm32flash.upload.params.verbose=
tools.stm32flash.upload.params.quiet=
tools.stm32flash.upload.pattern={path}/{cmd} -e 1535
   -w {build.path}/{build.project_name}.bin
   {upload.options} {port}
```

Obrázek 3.4: Koncová část souboru **platform.txt**

## 3.2 Implementace KETCube knihovny

Při vytvoření nového projektu ve vývojovém prostředí Arduino IDE se automaticky vytvoří funkce `setup()` a `loop()`. Funkce `setup()` se spouští jen jednou při spuštění programu a může obsahovat libovolná nastavení prováděna právě jednou. Funkce `loop()` se spouští po dokončení funkce `setup()` a provádí se cyklicky. V této funkci se často nacházejí např. opakovaná měření z periferních obvodů. Program v této funkci setrvává, dokud není zařízení restartováno. Tyto funkce je potřeba propojit s firmwarem platformy KETCube.

Do platformy KETCube byl začleněn modul Arduino, který lze zapínat/vypínat pomocí terminálu KETCube a přiřadit mu severitu[4]. Každý modul má automaticky vytvořenou inicializační funkci a periodicky spuštěnou funkci. Na tyto funkce je nutné tzv. namapovat funkce `setup()` a `loop()` používané prostředím Arduino IDE. Toto mapování je provedeno klíčovým slovem `extern` ve firmwaru platformy KETCube. Po slinkování se zkompilovaným souborem z prostředí Arduino IDE se tyto funkce spojí s použitými funkcemi `setup()` a `loop()`.

Dále je nutné definovat funkce, které lze použít při programování v prostředí Arduino IDE. Funkce jsou definovány v hlavičkových souborech **ketCube_arduino_terminal.h** a **ketCube_arduino_i2c.h** (obr. 3.5, 3.6 a 3.7).

Funkce jsou rozděleny podle oblasti využití na funkce vypisující do terminálu KETCube a funkce, které komunikují s externími zařízeními pomocí I2C komunikace. Výpis na terminál je zajištěn funkcemi, které jsou členěny podle již zmíněných priorit závažnosti vypisované informace. Všechny funkce využívají univerzální funkci firmwaru KETCube `ketCube_terminal_ModSeverityPrintln()`, která vyžaduje parametry:

- `KETCUBE_CFG_SEVERITY_<severity>`, kde `<severity>` je nahrazena klíčovým slovem závažnosti,
- `KETCUBE_LISTS_MODULEID_<moduleId>`, kde `<moduleId>` je v tomto případě ARDUINO,
- `formated_string`, formátovaný řetězec (jako klasická funkce `printf()`) vypisovaný do terminálu KETCube a

[4]severita - závažnost výpisů do terminálu

- `args`, argumenty formátovaného řetězce.

Druhým hlavičkovým souborem je **ketCube_arduino_i2c.h** (obr. 3.7), kde jsou definované funkce pro komunikaci s externími I2C zařízeními přímo z prostředí Arduino IDE.

Nejprve se musí komunikace I2C inicializovat, a proto je na začátku hlavičkového souboru funkce `ketCube_I2CInit(void)`. Tato funkce je typicky volaná pouze jednou a to ve funkci `setup()`. Pro případ potřeby zrušení inicializace I2C komunikace je zde také funkce `ketCube_I2CUnInit(void)`. Obě tyto funkce vracejí návratovou hodnotu typu `ketCube_cfg_DrvError_t`, která může nabývat hodnot:

- `KETCUBE_CFG_DRV_OK`, což odpovídá numerické hodnotě 0, nebo-li příkaz byl proveden v pořádku nebo
- `KETCUBE_CFG_DRV_ERROR`, což odpovídá numerické hodnotě 1, nebo-li nastala chyba při vykonávání příkazu.

Po úspěšném provedení inicializace I2C komunikace lze volat funkce, které zapisují nebo čtou do/z daného I2C zařízení. Pro určení I2C zařízení potřebujeme jeho adresu, na kterou zařízení bude přijímat resp. vysílat data z/do mikrokontroléru. Tato hodnota se z pravidla nachází v datasheetu daného I2C zařízení a je definovaná výrobcem.

Při volání funkcí `ketCube_I2CReadData` a `ketCube_I2CWriteData` z Arduino IDE prostředí potřebujeme znát tyto parametry:

- `uint8_t Addr`, zpravidla hexadecimální adresa daného I2C zařízení,
- `uint8_t Reg`, číslo registru v I2C zařízení, do kterého budeme zapisovat resp. číst data,
- `uint8_t* pBuffer`, ukazatel na pole osmibitových čísel, které bude mikroprocesor vysílat resp. do kterých bude zapisovat a
- `uint16_t Size`, velikost pole osmibitových čísel.

Protože I2C zařízení mohou mít specifické registrové mapy, jsou zde také definované funkce `ketCube_I2CReadRawData` a `ketCube_I2CWriteRawData` umožňující generický přístup k I2C sběrnici. Tyto funkce nevyžadují číslo registru, ale čtou resp. zapisují data ve formátu, ve kterém byla vyslána resp. přijata I2C zařízením.

Zmíněné funkce pracující na I2C komunikaci vrací hodnotu obdobně jako u inicializačních funkcí této komunikace. Návratová hodnota umožňuje ověřování úspěšnosti vyvolaného příkazu za běhu programu. To bohužel neověřuje správnost vyslaných resp. přijatých dat.

Pro ověření správnosti přijatých dat doplňují některá zařízení I2C data o kontrolní součet typu CRC8. Parametry kontrolního součtu CRC8 jsou uvedeny v datasheetu použitého senzoru SHT31 [5]. Stejný kontrolní součet lze spočítat v mikrokontroléru a porovnat s přijatým. Pokud se oba shodují, data jsou přijata v pořádku a lze s nimi nadále pracovat.

```
#ifndef KETCUBE_ARDUINO_TERMINAL_H
#define KETCUBE_ARDUINO_TERMINAL_H

#include <stdarg.h>
#include "ketCube_terminal.h"

static inline void ketCube_printfInfo
                   (char* formated_string, ...)
{
    va_list args;
    va_start(args, formated_string);
    ketCube_terminal_ModSeverityPrintln(
        KETCUBE_CFG_SEVERITY_INFO,
        KETCUBE_LISTS_MODULEID_ARDUINO,
        formated_string,
        args);
    va_end(args);
}

static inline void ketCube_printfError
                   (char* formated_string, ...)
{
    va_list args;
    va_start(args, formated_string);
    ketCube_terminal_ModSeverityPrintln(
        KETCUBE_CFG_SEVERITY_ERROR,
        KETCUBE_LISTS_MODULEID_ARDUINO,
        formated_string,
        args);
    va_end(args);
}

...
```

Obrázek 3.5: Začátek výpisu souboru **ketCube_arduino_terminal.h**

```
...

static inline void ketCube_printfWarning
                   (char* formated_string, ...)
{
    va_list args;
    va_start(args, formated_string);
    ketCube_terminal_ModSeverityPrintln(
        KETCUBE_CFG_SEVERITY_DEBUG,
        KETCUBE_LISTS_MODULEID_ARDUINO,
        formated_string,
        args);
    va_end(args);
}

static inline void ketCube_printfNone
                   (char* formated_string, ...)
{
    va_list args;
    va_start(args, formated_string);
    ketCube_terminal_ModSeverityPrintln(
        KETCUBE_CFG_SEVERITY_NONE,
        KETCUBE_LISTS_MODULEID_ARDUINO,
        formated_string,
        args);
    va_end(args);
}

#endif /* KETCUBE_ARDUINO_TERMINAL_H */
```

Obrázek 3.6: Pokračování výpisu souboru **ketCube_arduino_terminal.h**

```
#ifndef KETCUBE_ARDUINO_I2C_H
#define KETCUBE_ARDUINO_I2C_H

#include <stdarg.h>
#include "ketCube_i2c.h"

static inline ketCube_cfg_DrvError_t ketCube_I2CInit(void)
{
    return ketCube_I2C_Init();
}

static inline ketCube_cfg_DrvError_t ketCube_I2CUnInit(void)
{
    return ketCube_I2C_UnInit();
}

static inline ketCube_cfg_DrvError_t ketCube_I2CReadData
    (uint8_t Addr, uint8_t Reg,
    uint8_t * pBuffer, uint16_t Size)
{
        return ketCube_I2C_ReadData(Addr, Reg,
                                    pBuffer, Size);
}

static inline ketCube_cfg_DrvError_t ketCube_I2CWriteData
    (uint8_t Addr, uint8_t Reg,
    uint8_t * pBuffer, uint16_t Size)
{
        return ketCube_I2C_WriteData(Addr, Reg,
                                     pBuffer, Size);
}

static inline ketCube_cfg_DrvError_t ketCube_I2CWriteRawData
    (uint8_t Addr, uint8_t * pBuffer, uint16_t Size)
{
        return ketCube_I2C_WriteRawData(Addr, pBuffer, Size);
}

static inline ketCube_cfg_DrvError_t ketCube_I2CReadRawData
    (uint8_t Addr, uint8_t * pBuffer, uint16_t Size)
{
        return ketCube_I2C_ReadRawData(Addr, pBuffer, Size);
}

#endif /* KETCUBE_ARDUINO_I2C_H */
```

Obrázek 3.7: Výpis souboru **ketCube_arduino_i2c.h**

# 4 Návrh hardware a ověření funkčnosti

## 4.1 Analýza

Tato práce je zaměřena především na tvorbu sofwarového balíčku. Softwarový balíček se všemi jeho funkcemi je také potřeba otestovat na reálném zařízení. V implementované knihovně balíčku jsou funkce využívající I2C komunikaci. Zařízení KETCube je vybaveno dostupnými piny pro I2C komunikaci, ke kterým je nutné vybrat a připojit vhodné externí zařízení.

Zařízení KETCube disponuje standardizovaným konektorem mikroBUS™ [4], na který lze připojit libovolné externí zařízení vyhovující tomuto standardu. Pro ověření funkčnosti sestaveného balíčku je v této práci použit externí senzor relativní vlhkosti a teploty (RHT senzor). Tento senzor využívá plnou funkcionalitu balíčku a spolehlivě otestuje jeho funkčnost.

## 4.2 Návrh hardware

Deska plošného spoje je osazena RHT senzorem s označením SHT31-DIS-B [5]. Tento senzor měří relativní vlhkost a teplotu ovzduší. Ke komunikaci s vnějšími obvody využívá I2C komunikaci. To umožňuje nastavení senzoru a spuštění měření. Následně je komunikace využita k vyslání naměřených dat ze senzoru.

K hlavní desce platformy KETCube je sonzor připojen přes standardizovaný konektor mikroBUS™ (obr. 4.1). Tento konektor disponuje napájecími kontakty a potřebnými kontakty pro fyzické připojení na I2C sběrnici. Jsou zde také využity kontakty pro restartování senzoru, nastavení I2C adresy a případný vstup požadavku na přerušení vyžádaný senzorem.

Na konektoru mikroBUS™ se nachází dva napájecí kontakty s dostupným napětím 3,3V (kontakt 3V3) a 5V (kontakt 5V). Obě zdrojová napětí mají přidružené zemnící kontakty označené GND. V případě použití senzoru SHT31-DIS-B je možné použít rozsah vstupního napájecího napětí od 2,15V do 5,5V na kontaktu VDD. Senzor je tedy napájen napětím 3,3V dostupným

na kontaktu 3V3 konektoru mikroBUS™. K napájecímu kontaktu senzoru je připojen paralelně blokovací kondenzátor C1 s hodnotou 100nF umožňující stabilizaci napětí.

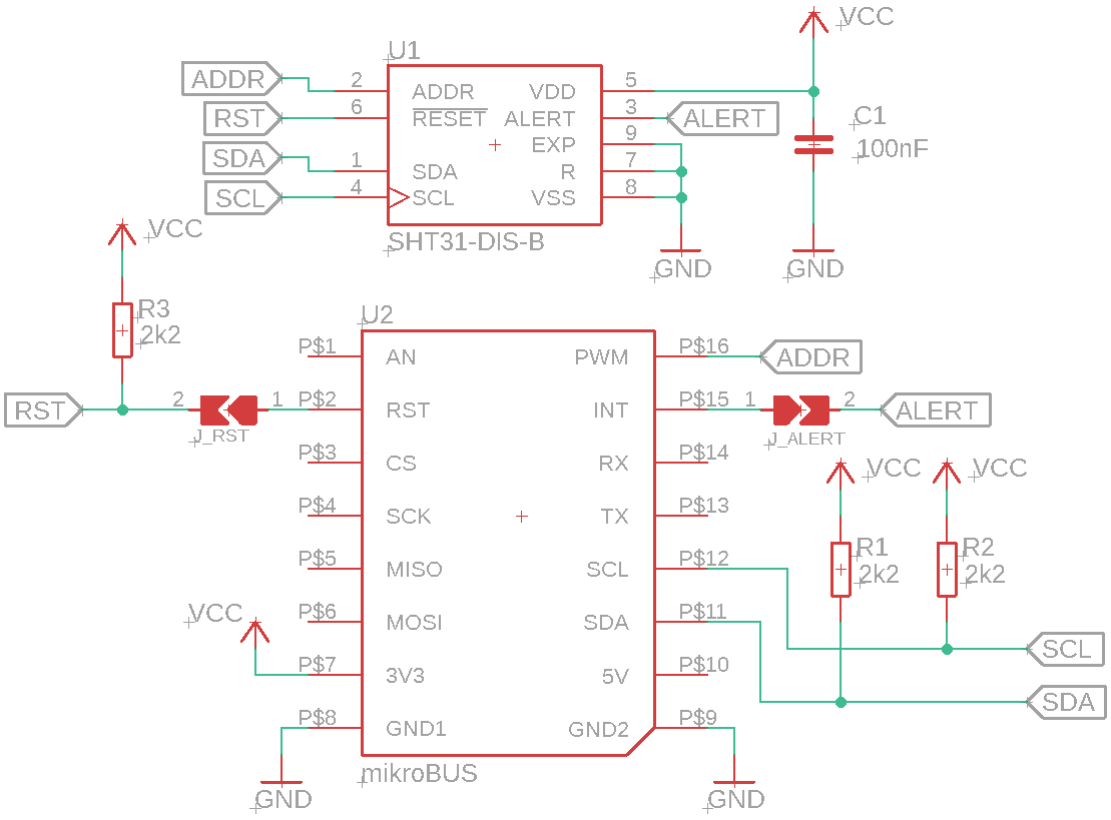
Sběrnice komunikace I2C disponuje standardně kontaktem SDA, po kterém se sériově přenášejí data a kontaktem SCL, který slouží jako zdroj hodinových signálů korespondující s taktem přenášených dat. Kontakt SDA senzoru je tedy přímo připojen na kontakt SDA konektoru mikroBUS™. Obdobně je tomu tak i u kontaktu SCL. Oba vodiče sběrnice I2C mají ještě připojené *tzv.* pull-up rezistory (označené R1 a R2).

Při restartu zařízení KETCube dojde i k restartu senzoru SHT31. To je zajištěno propojením kontaktu RST konektoru mikroBUS™ a konektoru RST senzoru. Toto propojení je standardně rozpojené, ale lze jej připojit pomocí pájecí propojky J_RST. Propojující vodič na plošném spoji je připojen k pull-up rezistoru R3 obdobně jako tomu bylo u sběrnice I2C.

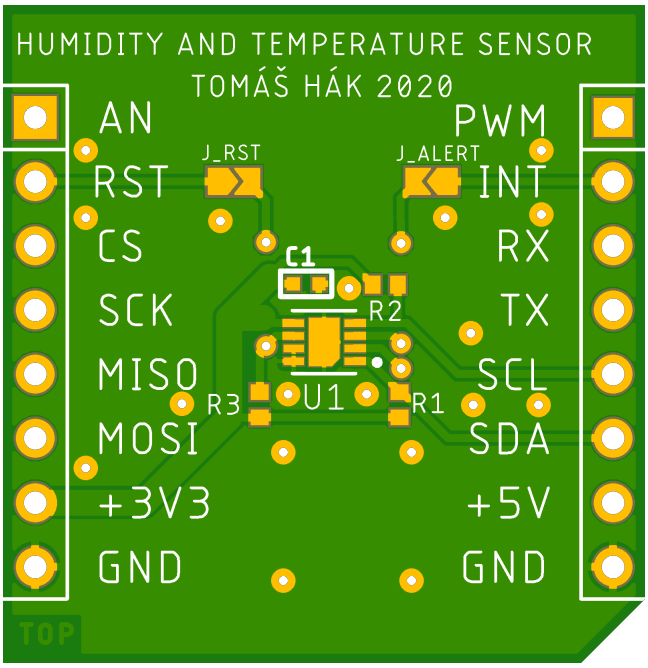
Senzor SHT31 umožňuje změnu I2C adresy pomocí kontaktu ADDR. Pokud by uživatel potřeboval adresu změnit, je tento kontakt vyveden ke konektoru mikroBUS™ na kontakt označený jako PWM. V mikrokontroléru lze nastavit alternativní funkci tohoto kontaktu na GPIO. To umožní programové nastavení kontaktu na hodnotu log. 0 nebo log. 1 a tím změnu požadované I2C adresy na hodnotu `0x44` resp. `0x45`.

Poslední kontakt vyvedený ke konektoru mikroBUS™ je ALERT, neboli signál požadavku na přerušení aktuálního programu mikrokontroléru. Tato funkcionalita je připojena přes pájecí propojku J_ALERT, kterou lze tento kontakt odpojit. Dokumentace senzoru SHT31-DIS-B [5] doporučuje kontakt ALERT při nepoužívání nechat odpojený od dalších obvodů.

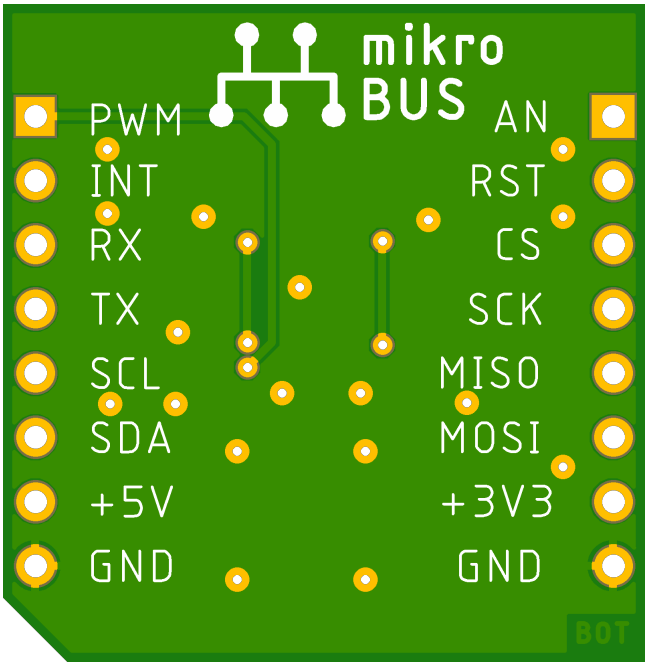
Konektor mikorBUS™ disponuje dalšími kontakty na připojení SPI a USART komunikace, které nejsou v tomto projektu využity. Na obrázcích 4.2 a 4.3 je vyobrazena vrchní resp. spodní strana DPS senzoru SHT31. Pro jednoduší montáž jsou všechny součástky na vrchní straně desky. Na spodní straně jsou jen propojovací vodiče. Obě strany desky mají *tzv.* rozlitou zem [8]. Pro jednoduchou lokalizaci kontaktů a součástek na DPS lze využít tištěný popis přímo na desce. Výsledná velikost desky je 25,4 x 26,04 mm, a tedy odpovídá standardu mikroBUS™.




Obrázek 4.1: Schéma desky se senzorem SHT31




Obrázek 4.2: Pohled na vrchní stranu desky senzoru SHT31




Obrázek 4.3: Pohled na spodní stranu desky senzoru SHT31

## 4.3 Návrh software

Pro ověření funkčnosti balíčku KETCube a desky s RHT senzorem je potřeba vytvořit testovací program ve vývojovém prostředí Arduino IDE. Tento program periodicky čte data ze senzoru a vysílá je na terminál.

V první části testovacího programu (obr. 4.4) jsou vložené knihovny z balíčku ketCube, kde jsou definované funkce použitelné v Arduino IDE projektu. Knihovna `ketCube_arduino_terminal.h` obsahuje funkce pro výpis na terminál a `ketCube_arduino_i2c.h` funkce ke komunikaci s externím I2C zařízením, v tomto případě s RHT senzorem. Je zde také definována adresa I2C senzoru. Adresu lze nalézt v datasheetu použitého senzoru a je závislá na přivedeném napětí na kontakt ADDR součástky senzoru. Protože se adresy používané v I2C komunikaci zapisují 7-bitově, je potřeba 8-bitovou hodnotu `0x44` posunout o 1 bit doleva.

Následuje standardní funkce `setup()`, ve které je nutné I2C komunikaci inicializovat. To se provádí příkazem `ketCube_I2CInit()`. Pokud je příkaz proveden v pořádku, návratová hodnota odpovídá `KETCUBE_CFG_DRV_OK`, což je ekvivalentní numerické hodnotě 0. V opačném případě je vrácena hodnota `KETCUBE_CFG_DRV_ERROR`, což je ekvivalentní numerické hodnotě 1. Vrácená hodnota je uložena do proměnné `outputInit`, která je následně vyhodnocena a stav úspěšnosti příkazu je vypsán na terminál.

Po inicializaci komunikace je potřeba stanovit parametry měření. Tyto parametry se podle dokumentace senzoru SHT31 nazývají *command* neboli příkaz. Tento příkaz se skládá ze dvou 8-bitových čísel, kde první číslo (command MSB) povoluje/zakazuje použití tzv. clock stretchingu[1]. Druhé číslo (command LSB) nastavuje úroveň opakovatelnosti (*Low*, *Medium* nebo *High*). Při zvýšení opakovatelnosti se zvyšuje doba měření a přesnost obou veličin. Zde je *clock stretching* povolen a opakovatelnost nastavena na hodnotu *High*. Před samotným měřením jsou ještě připraveny proměnné, do kterých se budou data zapisovat.

Periodicky volaná funkce `loop()` obsahuje samotné měření a zpracování dat. Nejprve je datové pole `dataBuffer` vynulováno pomocí cyklu `for`. Poté je vyslán zmíněný příkaz společně s adresou cílového I2C zařízení. O odeslání příkazu se postará funkce `ketCube_I2CWriteRawData()`, která má za vstupní parametry adresu, daný příkaz a délku příkazu v bajtech. Po přijetí

[1] clock stretching - umožňuje pozdržet příchod dalšího bytu

tohoto příkazu senzor SHT31 spustí měření obou veličin. Program dále pokračuje ke čtení dat pomocí funkce `ketCube_I2CReadRawData()` ze sběrnice I2C, na kterou momentálně senzor zapisuje naměřená data. Odeslaná data jsou opatřena kontrolním součtem CRC8. Senzor vyšle celkem 6 bajtů dat, přičemž první dva bajty obsahují 16-bitovou hodnotu teploty a třetí bajt obsahuje kontrolní součet CRC8 prvních dvou bajtů. Obdobně je rozdělena druhá polovina bajtů, první dva bajty této poloviny nesou hodnotu relativní vlhkosti a poslední bajt kontrolní součet CRC8 předchozí 16-bitové hodnoty. Tyto přijaté bajty jsou zapsány do připraveného pole `dataBuffer`.

Oba provedené příkazy zapsaly návratovou hodnotu do výstupní proměnné `outputWrite` resp. `outputRead`. Před zpracováním dat jsou tyto návratové hodnoty překontrolovány, zda byly příkazy správně provedeny. Pokud by nastala během volání těchto funkcí chyba, program vypíše informaci o jejich neprovedení na terminál a případná přijatá data nebude zpracovávat. Při správném provedení operací s I2C sběrnicí se v dalším kroku kontroly provede na základě přijatých dat lokální součet CRC8 a je porovnán s patřičnými bajty přijaté zprávy, která také nese kontrolní součet. Pokud je i tento stupeň kontroly v pořádku, program pokračuje ke zporacování přijatých dat. V opačném případě je uživatel informován o nerovnosti kontrolních součtů pomocí terminálu a data se dále nezpracovávají.

Samotné zpracování dat spočívá v sjednocení dvou 8-bitových čísel do jednoho 16-bitového a pomocí vzorce přepočítat tuto hodnotu na reálnou fyzikální veličinu. Vzorce pro přepočet lze nalézt v dokumentaci daného senzoru. Protože jsou fyzikální veličiny vyjádřeny reálnými čísly, je potřeba celočíselnou hodnotu přepočítat. Přepočet je proveden optimalizovaným postupem, který je ekvivalentní následujícím vzorcům:

$$T[^{\circ}C] = -45 + 175 \cdot \frac{S_T}{2^{16} - 1}$$

$$RH[\%] = 100 \cdot \frac{S_{RH}}{2^{16} - 1}$$

, kde $S_T$ a $S_{RH}$ jsou přijaté celočíselné hodnoty převedené do desítkové soustavy.

Obě naměřené a fyzikálně vyjádřené hodnoty jsou nyní uloženy v proměnných `temp` a `humidity`. S těmito hodnotami lze dále pracovat. Informace o těchto hodnotách je vypsána na terminál pomocí funkce `ketCube_printfInfo()` s prioritou INFO. Případné chybové hlášky mají

prioritu ERROR, aby se zobrazily na terminálu i když je informativní výpis vypnut. K tomu je využita funkce `ketCube_printfError()`.

```
#include "ketCube_arduino_terminal.h"
#include "ketCube_arduino_i2c.h"

#define ADDR 0x44 << 1

void setup() {
  ketCube_cfg_DrvError_t outputInit = ketCube_I2CInit();

  ketCube_printfInfo("Init I2C ... %s",
                     (outputInit) ? "ERROR" : "OK");
}

uint8_t cmd[2] = {0x2C, 0x06};     // Clock stretching enabled
                                   // Repeatibility = High
uint8_t dataBuffer[6];

uint8_t i = 0;

float temp = 0, humidity = 0;
```

Obrázek 4.4: Ověřovací program napsaný v Arduino IDE - funkce setup()

```
void loop() {
  for (i = 0; i < sizeof(dataBuffer); i++)
    dataBuffer[i] = 0x0;

  ketCube_cfg_DrvError_t outputWrite =
              ketCube_I2CWriteRawData(ADDR, cmd,
                                      sizeof(cmd));

  ketCube_cfg_DrvError_t outputRead =
              ketCube_I2CReadRawData(ADDR, dataBuffer,
                                     sizeof(dataBuffer));

  if (outputWrite == KETCUBE_CFG_DRV_OK &&
      outputRead == KETCUBE_CFG_DRV_OK) {

    if (dataBuffer[2] != crc8(dataBuffer, 2) ||
        dataBuffer[5] != crc8(dataBuffer + 3, 2)) {

      ketCube_printfError("Data received ... OK,
                           CRC8 ...ERROR\t
                           Data will not be used!");

    }
    else {
      int32_t _temp = (int32_t)
      (((uint32_t)dataBuffer[0] << 8) | dataBuffer[1]);
      _temp = ((4375 * _temp) >> 14) - 4500;
      temp = (float)_temp / 100.0f;

      uint32_t _humidity = ((uint32_t)dataBuffer[3] << 8) |
                           dataBuffer[4];
      _humidity = (625 * _humidity) >> 12;
      humidity = (float)_humidity / 100.0f;

      ketCube_printfInfo("temp = %.2fC, humidity = %.2f%%",
                         temp, humidity);
    }
  }
  else {
    ketCube_printfError("Writing command or reading data ...
                         ERROR");
  }

}
```

Obrázek 4.5: Ověřovací program napsaný v Arduino IDE - funkce loop()

## 4.4 Ověření funkčnosti

Pro ověření funkčnosti je potřeba desku plošného spoje se senzorem SHT31-DIS-B připojit ke korespondujícímu konektoru mikroBUS$^{TM}$ na hlavní desce platformy KETCube. Tento konektor zároveň přivádí napájecí napětí k externí desce, takže není potřeba externího zdroje. Po propojení desek je nutné do FLASH paměti mikroprocesoru nahrát zkompilovaný a sestavený kód napsaný ve vývojovém prostředí Arduino IDE. Nahrávání do mikroprocesoru umožňuje další externí deska se převodníkem USB na UART. Tato deska je propojená s hlavní deskou pomocí proprietárního konektoru platformy KETCube. Zároveň je osazená konektorem micro USB, kterým lze pomocí USB kabelu připojit celé zařízení k počítači.

Po nainstalování balíčku, který podporuje desky platformy KETCube, lze ve vývojovém prostředí Arduino IDE vybrat v záložce `Nástroje->Vývojová deska->KETCube (ARM 32-bit) boards` položku `KETCube_mainboard`. Následně se automaticky vybere nahrávací metoda `STM32Flash (Serial bootloader)`. Po připojení KETCube zařízení přes převodník USB na UART lze vybrat v záložce `Nastavení->Port` označení USB portu, ke kterému je zařízení právě připojeno. Dále je nutné stiskem tlačítek BOOT a RESET aktivovat STM bootloader z paměti ROM KETCube.

Následně je možné pomocí dostupného tlačítka ve vývojovém prostředí Arduino IDE *Ověřit* aktuální kód zkompilovat a tím ověřit správnost syntaxe. Vedlejším tlačítkem *Nahrát* se kód znovu zkompiluje a výstup Linkeru (binární soubor) je přes připojený převodní USB na UART nahrán do vnitřní paměti zařízení KETCube. Po nahrání celého projektu do paměti zařízení KETCube je mikroprocesor restartován a automaticky spustí právě nahraný program.

Ve výstupním terminálu je nutné zmíněný modul Arduino povolit. To se provede pomocí příkazu `enable Arduino 2`, přičemž posledním číselným parametrem nastavujeme závažnost vypisovaných výstupů. Číselný parametr `2` signalizuje výpis výstupu úrovně *INFO*. Nakonec je potřeba zařízení znovu načíst příkazem `reload`. Vypnutí modulu lze provést příkazem `disable Arduino` a opakovaném načtení zařízení.

Námi vypisované hodnoty teploty a relativní vlhkosti mají závažnost úrovně *INFO*, a proto by se po uplynutí počátečního zpoždění `KETCube startDelay` měly začít na terminálu objevovat (obr. 4.6).

Na terminálu je možné si také povšimnout, zda byla inicializace z funkce `setup()` a tím i inicializace I2C komunikace správně provedena. Prováděné měření je ve funkci `loop()`, která se spouští opakovaně s nastavenou periodou v `KETCube core basePeriod`. Zde je tato hodnota 1000 ms nebo-li 1 sekunda.

```
KETCube core base period set to: 1000 ms
KETCube start delay set to: 2000 ms
KETCube core severity level: INFO
KETCube driver severity level: NONE
--- "core" Init() END ---

--- "Arduino" Init() START ---
Module severity level: INFO
Arduino :: Init I2C ... OK
--- "Arduino" Init() END ---

>>
Arduino :: temp = 25.20°C, humidity = 46.06%
Arduino :: temp = 25.20°C, humidity = 46.10%
Arduino :: temp = 25.18°C, humidity = 45.80%
Arduino :: temp = 25.18°C, humidity = 45.27%
Arduino :: temp = 25.17°C, humidity = 44.79%
Arduino :: temp = 25.12°C, humidity = 44.65%
Arduino :: temp = 25.14°C, humidity = 44.99%
Arduino :: temp = 25.10°C, humidity = 45.21%
Arduino :: temp = 25.08°C, humidity = 45.28%
Arduino :: temp = 25.10°C, humidity = 45.51%
Arduino :: temp = 25.08°C, humidity = 45.98%
>>
```

Obrázek 4.6: Výpis terminálu při spuštěném periodickém měření teploty a relativní vlhkosti

# 5 Závěr

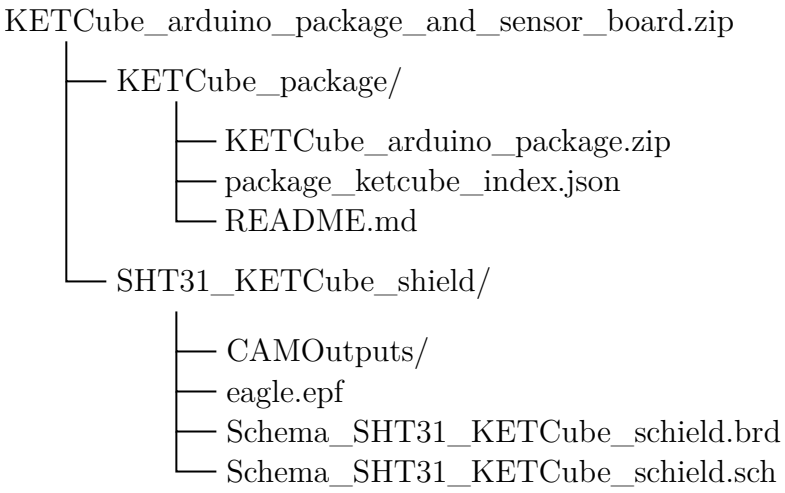
Tato práce má umožnit novým uživatelům programování zařízení KETCube ve vývojovém prostředí Arduino IDE. Tato možnost je zajištěna vytvořeným balíčkem, který lze do vývojového prostředí importovat. Hlavní výhodou je výsledná jednoduchost programování a nahrávání kódu do zařízení.

Funkčnost balíčku je ověřena pomocí externího zařízení, které je připojeno k hlavní desce platformy KETCube. Toto zařízení bylo vybráno tak, aby vyhovovalo technickým možnostem platformy KETCube a plně otestovalo funkce definované v knihovních souborech balíčku. Externím zařízením je senzor teploty a relativní vlhkosti SHT31, který je umístěný na vlastní desce plošného spoje.

V prostředí Arduino IDE byl vyvinut program, který nastaví externí zařízení a periodicky čte měřené hodnoty. Uživatel si může naměřené hodnoty zkontrolovat v terminálu KETCube a případně je dále využívat k dalším účelům.

Testovací program ověřil, že implementovaný balíček společně s externím zařízením lze použít pro práci s platformou KETCube. Další uživatelé tak mají volný prostor k vývoji vlastních externích zařízení a skrze prostředí Arduino IDE je jim umožněn programový přístup k platformě KETCube.

# A Obsah přiloženého archivu




Obrázek A.1: Stromový výpis obsahu přiloženého archivu **KETCube_package** a výrobních souborů desky se senzorem SHT31

V tomto archivu lze nalézt kompletní balíček společně s adresářem výrobních souborů desky senzoru SHT31. Adresář *KETCube_package/* obsahuje archiv balíčku pro vývojové prostředí **KETCube_arduino_package.zip**, soubor s informacemi o balíčku a jeho potřebných programů **package_ketcube_index.json** a informační soubor **README.md**.

Druhá část archivu obsahuje výrobní soubory desky senzoru SHT31. Zde lze nalézt výstupy pro tvorbu šablon, vrtání otvorů, samotné schéma desky a konečné rozmístění součástek na plošném spoji.

# Literatura


[1] *Arduino IDE Documentation* [online]. Ivrea Interaction Design Institute, 2021. [cit. 1. 5. 2021]. Arduino IDE Documentation. Dostupné z: https://www.arduino.cc/en/Guide/Introduction.

[2] Bělohoubek, J. *KETCube documentation* [online]. The SmartCAMPUS Team, 2019. [cit. 1. 5. 2021]. Dostupné z: https://github.com/SmartCAMPUSZCU/KETCube-docs/blob/master/KETCubeDatasheet.pdf.

[3] Herout, P. *Učebnice jazyka C.* nakladatelství KOPP, 2007. ISBN 80-7232-220-6.

[4] mikroElektronika. *mikroBUS standard specification* [online]. mikroElektronika, 2015. [cit. 1. 5. 2021]. Dostupné z: http://download.mikroe.com/documents/standards/mikrobus/mikrobus-standard-specification-v200.pdf.

[5] Sensirion. *SHT31-DIS-B relative humidity and temperature sensor* [online]. Sensirion, 2019. [cit. 1. 5. 2021]. Dostupné z: https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/2_Humidity_Sensors/Datasheets/Sensirion_Humidity_Sensors_SHT3x_Datasheet_digital.pdf.

[6] STMicroelectronics. *STM32L0-cortex-M0+ documentation* [online]. ST microelectronics, 2016. [cit. 1. 5. 2021]. STM32L082CZ-cortex-M0+. Dostupné z: https://www.st.com/en/microcontrollers-microprocessors/stm32l0-series.html.

[7] TI. *HDC1080 RHT senzor documentation* [online]. Texas Instruments, 2014. [cit. 1. 5. 2021]. Dostupné z: https://www.ti.com/document-viewer/HDC1080/datasheet.

[8] Záhlava, V. *Návrh a konstrukce desek plošných spojů.* BEN - Technická literatura, 2010. ISBN 978-80-7300-2.