VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

PODPORA MIKROKONTROLERŮ NXP VE VÝVOJOVÉM PROSTŘEDÍ ARDUINO IDE

COMPATIBILITY OF NXP MICROCONTROLLERS WITH ARDUINO IDE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. JAN TOMEČEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VÁCLAV ŠIMEK**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

154644

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Tomeček Jan, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Vývoj aplikací
Název:	**Podpora mikrokontrolerů NXP ve vývojovém prostředí Arduino IDE**
Kategorie:	Vestavěné systémy
Akademický rok:	2023/24

Zadání:

1. Detailně se seznamte s open-source vývojovým prostředím Arduino IDE. Zaměřte se především na strukturu dostupných knihoven a podporovanou funkcionalitu
2. Prostudujte strukturu, uživatelské rozhraní a praktické aspekty použití podpůrného SDK balíčku pro vybrané mikrokontrolery od společnosti NXP.
3. Prozkoumejte možnosti rozšíření prostředí Arduino IDE o podporu mikrokontrolerů NXP, přičemž se soustřeďte zejména na typy zvolené v bodě 2.
4. Navrhněte infrastrukturu potřebnou k funkčnímu propojení systému Arduino IDE a NXP SDK balíčků.
5. Proveďte implementaci sady základních funkcí dostupných v prostředí Arduino IDE tak, aby je bylo možné využít v kombinaci s NXP SDK balíčky pro zvolené typy mikrokontrolerů.
6. Vytvořené řešení pečlivě otestujte na vybraných exemplářích vývojových desek pro příslušné mikrokontrolery a zdokumentujte detailně jeho použití.
7. Zhodnoťte dosažené výsledky a diskutujte možná rozšíření či další pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:
Splnění bodů 1 až 4 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Šimek Václav, Ing.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2023
Termín pro odevzdání:	17.5.2024
Datum schválení:	30.10.2023

Abstrakt

Tato práce se zabývá návrhem podpory mikrokontrolérů od společnosti NXP Semiconductors ve vývojovém prostředí Arduino IDE. Práce analyzuje jednotlivé prvky Arduino IDE, jeho rozšiřovacími balíky a jednotlivé konfigurační soubory. Také popisuje prostředí MCUXpresso IDE, na základě kterého je navržen balík pro podporu mikrokontrolérů od NXP.

Abstract

This thesis deals with the design of support for microcontrollers from NXP Semiconductors in the Arduino IDE development environment. The thesis analyses the individual elements of the Arduino IDE, its extension packages and individual configuration files. It also describes the MCUXpresso IDE environment and based on the analysis the package for supporting NXP microcontrollers is designed.

Klíčová slova

Arduino, Arduino IDE, Mikrokontroléry, MCUXpresso, LPC845, LPC860

Keywords

Arduino, Arduino IDE, Microcontrollers, MCUXpresso, LPC845, LPC860

Citace

TOMEČEK, Jan. *Podpora mikrokontrolerů NXP ve vývojovém prostředí Arduino IDE.* Brno, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Václav Šimek

Podpora mikrokontrolerů NXP ve vývojovém prostředí Arduino IDE

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Václava Šimka Další informace mi poskytli Petr Hradský a Robert Havránek ze společnosti NXP Semiconductors. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jan Tomeček
17.5.2024

Poděkování

Děkuji vedoucímu práce, Ing. Václavu Šimkovi, za cenné rady a konzultace v průběhu vypracování práce. Dále děkuji Petru Hradskému a Robertu Havránkovi ze společnosti NXP Semiconductors za odbornou pomoc a konzultace během řešení problematiky práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

V dnešní době rychlého rozvoje technologií se mikrokontroléry stávají nedílnou součástí našeho každodenního života. Můžeme je nalézt v mnoha elektronických zařízeních, od běžných domácích spotřebičů po komplexní průmyslové systémy. Jedním z významných producentů těchto mikrokontrolérů je společnost NXP Semiconductors, která nabízí velké portfólio různě výkonných mikrokontrolérů pro všelijaké aplikace. Společnost NXP Semiconductors spíše cílí na průmyslové aplikace, ale mnoho svých mikrokontrolérů nabízí i široké veřejnosti.

Pro vývoj vestavěných aplikací existuje velké množství IDE (integrovaných vývojových prostředí). Zpravidla každý výrobce poskytuje buď své IDE, nebo zabezpečí podporu pro vývoj svých mikrokontrolérů v jiných IDE. Kromě vývoje v IDE je také možnost používat běžné editory zdrojového kódu, které ale v základu nenabízí tolik možností a nástrojů pro vývoj jako IDE.

Tato diplomová práce se zaměřuje na integraci a podporu vybraných mikrokontrolérů od společnosti NXP Semiconductors ve vývojovém prostředí Arduino IDE. Arduino IDE je velmi populární platforma pro vývoj projektů s mikrokontroléry. Známá je je hlavně svojí jednoduchostí, takže je přívětivá jak pro začátečníky, tak pro pokročilé uživatele. Integrace mikrokontrolérů od společnosti NXP přináší několik výhod, jako je zjednodušení programování, snadná konfigurace a rychlý vývoj prototypů.

Cílem této práce popsat jednotlivé kroky, které jsou nutné udělat pro zajištění podpory mikrokontrolérů od společnosti NXP Semiconductors ve vývojovém prostředí Arduino IDE. Práce se dále zaměří na vybrané mikrokontroléry pro které bude zajištěna podpora v Arduino IDE.

V rámci této práce bude popsáno vývojové prostředí Arduino IDE, jeho součásti a způsob jakým probíhá překlad kódu v tomto prostředí. Následuje kapitola, která se zabývá podporou nových desek v Arduino IDE. V té budou hlavně popsány principy a možnosti rozšiřovacích balíků v tomto prostředí. Dále bude popsáno vývojové prostředí MCUXpresso IDE od společnosti NXP Semiconductors včetně SDK (sady vývojářských nástrojů) pro zvolené mikrokontroléry. Následně bude představen návrh řešení pro zajištění podpory vybraných mikrokontrolérů v Arduino IDE. Podle popsaného návrhu pak bude implementována podpora pro jeden z vybraných mikrokontrolérů. Následně bude popsán způsob otestování podpůrného balíku. Získané postupy, pak budou ověřeny při tvorbě podpory pro další mikrokontrolér.

Kapitola 2

Vývojové prostředí Arduino IDE

Pro zajištění podpory nových nebo doposud nepodporovaných mikrokontrolérů ve vývojovém prostředí Arduino IDE je nutné nejprve analyzovat samotné vývojové prostředí a jeho součástí. V této kapitole se seznámíme s jednotlivými částmi prostředí, především se strukturou projektů, způsobem jakým je kód překládán a nahráván do mikrokontrolérů. Popsán bude také jazyk Arduino, ve kterém se programují projekty v tomto vývojovém prostředí.

2.1 Arduino IDE

Arduino je open-source projekt, který zahrnuje hardware i software určený především pro jednoduché použití. Součástí projektu je několik modelů Arduino desek a vývojové prostředí Arduino IDE. Pro programování desek projekt zahrnuje i jazyk Arduino. Díky tomu tak Arduino poskytuje platformu, která je vhodná i pro úplné začátečníky s mikrokontroléry. Užití těchto produktů můžeme najít v řadě edukačních, hobby, ale profesionálních sférách.

Díky vysoké popularitě k vývoji Arduina přispívá široká komunita lidí. Do Arduino IDE lze nainstalovat mnoho knihoven a také zle v Arduino IDE vyvíjet i mikrokontroléry od jiných firem.

2.2 Prostředí a Funkcionalita

Prostředí Arduino poskytuje mnoho funkcí pro kompletní vývoj. Oproti komerčním IDE může obsahovat méně funkcí, ale díky jednoduchosti a velké komunitě je velmi populární.

Projekty se v Arduino IDE nazývají **skeče**. Uživatel při tvorbě skeče pouze edituje samotnou jím definovanou funkcionalitu. Nemá tedy přístup k ostatním souborům, které jsou součástí výsledného programu při překladu. Tyto ostatní soubory jsou součástí „**jádra**“, které jé je specifikované pro každou desku. Tedy každá deska, co má podporu v Arduino IDE má specifikovanou strukturu souborů pro překlad, což Arduino nazývá jako core (jádro).

Součástí každé skeče jsou funkce `void setup()` a `void loop()`. Funkce **setup** je pak volána jednou, po restartu mikrokontroléru, a funkce **loop** je volána periodicky v nekonečné smyčce.

2.2.1 Prostředí

Obrázek 2.1: Ukázka hlavního pohledu v Arduino IDE 2 popisující i jednotlivé části uživatelského rozhraní[1]

Hlavní součástí prostředí je textový editor a nástroje pro kompilaci a nahrání kódu na mikrokontrolér. Součástí Arduino IDE 2 jsou také tyto funkce:[1]

- Arduino Sketchbook - manažer projektů, zobrazuje historii posledních editovaných projektů
- Boards Manager - manažer desek, umožňuje stahovat a aktualizovat balíky podpory pro jednotlivé mikrokontroléry
- Library Manager - manažer knihoven, zabezpečuje instalaci a aktualizaci dostupných knihoven
- Serial Monitor - sériový monitor, zobrazuje data, která deska posílá přes sériovou linku
- Serial Plotter - sériový plotr, vizualizuje průběh dat v čase
- Example sketches - sada ukázek kódu, prostředí obsahuje mnoho ukázek kódu od základní funkcionality (rozblikání LED diody) až po komplexnější kód. Zahrnují také ukázkový kód z knihoven.
- Debugger tool - nástroj pro ladění kódu
- Autocompletion - Automatické dokončování, novinka v IDE verze 2.x
- Remote Sketchbook - manažer vzdálených projektů, umožňuje práci s Arduino Cloudem
- Firmware & Certificate Uploader - Nástroj pro nahrávání firwaru a certifikátů pro desky s WIFI s NINA modulem

[1]Obrázek převzat z https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2.

2.3 Jazyk Arduino

Jazyk Arduino je založen na C++. Oproti C++ je rozšířen o funkcionalitu pro snazší vývoj vestavěných aplikací.

Při popisu tohoto jazyka se můžeme soustředit na jeho 3 hlavní části:[2]:

- Funkce
- Hodnoty
- Struktura

Funkce, které jsou součásti tohoto jazyka, implementují základní funkcionalitu a konfiguraci mikrokntrolérů. Ačkoliv je většina detailních konfigurací před uživatelem skryta, může uživatel základní konfigurace, jako například nastavení pinů na vstup/výstup, provézt pomocí těchto funkcí.

Hodnoty se v tomto jazyce ve většině shodují s jazykem C++. Navíc ale jazyk dodává konstanty a hodnoty související s vestavěnými aplikacemi, jako například logické hodnoty pro digitální výstup pinů. Zde definuje konstantu „HIGH“ pro logickou 1 a „LOW“ pro logickou 0.

Struktura je také totožná s jazykem C++. Hlavním dodatkem oproti C++ jsou funkce `setup()` a `loop()`. Tyto funkce implementují základní tok programu. Funkce `setup()` je volána jednou po restartu mikrokontroléru a funkce `loop()` je periodicky volána v nekonečné smyčce.

2.3.1 Funkce

Nejpoužívanějšími funkcemi, které Arduino jazyk navíc poskytuje jsou funkce pro řízení GPIO (Univerzálních vstupních nebo výstupních pinů). Ty dále můžeme dělit na funkce pro obsluhu digitálních a analogových vstupů a výstupů. Dalšími, často využívanými, funkcemi jsou metody třídy `Serial`, která zabezpečuje sériovou komunikaci (hlavně využíváno k ladící komunikaci s terminálem).

Pro obsluhu digitálních vstupů a výstupů to jsou funkce:

- `pinMode(pin, mód)` - inicializace pinů, nastavuje daný pin[1] buď jako vstup, vstup s pullup rezistorem, nebo výstup
- `digitalRead(pin)` - přečtení logické hodnoty na daném pinu[1]
- `digitalWrite(pin, hodnota)` - nastavení logické hodnoty daného pinu[1]

Pro obsluhu analogových vstupů a výstupů pak funkce:

- `analogRead(pin)` - přečtení analogové hodnoty na daném pinu[1]
- `analogReference(hodnota)` - nastavení referenčního napětí pro A/D převodník
- `analogWrite(pin, hodnota)` - nastavení analogové hodnoty na daný pin[1], využívá PWM, $hodnota \in \langle 0, 255 \rangle$

[1] parametr „pin“ je číslo odpovídající číslu pinu

Často používané metody třídy `Serial`:

- `Serial.print(zpráva)` - odešle zprávu přes sériové rozhraní
- `Serial.println(zpráva)` - odešle zprávu přes sériové rozhraní, po zprávě jsou poslány znaky nového řádku `\r\n`

Obecně jsou ale funkce podle jazyka Arduino děleny do kategorií podle jejich funkčnosti.[2] Jednotlivé kategorie, které dosud nebyly popsány, a jejich funkce budou pak probrány v kapitole 7, která se věnuje jejich implementaci. Jediná nepopsaná je kategorie USB, jejíž funkčnost není na vybraných mikrokontrolérech od firmy NXP dostupná.

Výčet kategorií funkcí podle jejich funkčnosti:

- Digital I/0 – obsluha digitálních vstupů a výstupů
- Analog I/O – obsluha analogových vstupů a výstupů
- Advanced I/O – rozšíření obsluhy vstupů a výstupů
- Time – zpoždění a čas
- Math – matematické funkce
- Trigonometry – trigonometrické funkce
- Characters – práce se znaky
- Random numbers – náhodná čísla
- Bits and Bytes – práce s bity
- External Interrupts – externí přerušení
- Interrupts – přerušení
- Communication – komunikace
- USB – Emulace USB periferií (klávesnice, myši). Není součástí řešení.

2.4 Arduino CLI a překlad kódu

Arduino CLI je nástroj spustitelný z příkazové řádky, který nabízí veškeré funkce jako celé Arduino IDE. Tedy Arduino IDE je založeno nad tímto nástrojem. Kromě správce desek/knihoven Arduino CLI zabezpečuje překlad a nahrávání kódu do mikrokontrolérů.

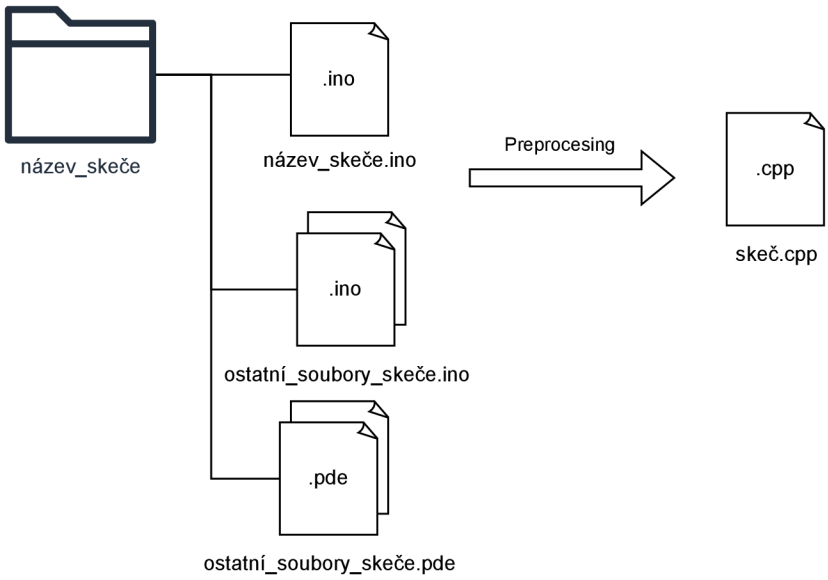
Překlad skeče probírá v následujících krocích:

1. Předzpracování
2. Vyřešení závislostí
3. Překlad kódu

2.4.1 Předzpracování

Nejprve dojde ke spojení všech `.ino` a `.pde` souborů do jednoho. V rámci tomto kroku se spojí soubor se stejným jménem jako je adresář skeče s ostatními soubory v adresáři v abecedním pořadí. Výsledný soubor má poté změněnou koncovku na `.cpp`. Dále je doplněn
o `#include <Arduino.h>`, pokud již není součástí skeče. Dále jsou přidány prototypy pro všechny funkce, které nemají prototyp. V neposlední řadě je soubor doplněn o direktivy `#line` , které slouží kompilátoru aby v případě chyby v kódu věděl, kde se chyba nacházela v původním rozložení skeče. Preprocesor nezpracovává žádné jiné soubory, než `.ino` a `.pde`.[4]

Popis struktury souborů skeče a tedy vstupních souborů pro předzpracování se nachází na obrázku 2.2.



Obrázek 2.2: Schéma popisující které soubory jsou součástí předzpracování preprocesoru. Výsledkem předzpracování je soubor „skeč.cpp“, kde „skeč“ je konkrétní pojmenování skeče a je totožné s pojmenováním vstupního souboru „název_skeče.ino“

2.4.2 Vyřešení závislostí

Dalším krokem je vyřešení závislostí. Pro nalezení všech závislostí je adresář skeče rekurzivně prohledávaný. Kromě toho se také prohledává: adresář jádra, adresář, který slouží pro popis variací desky (viz kapitola 3), adresář standardní systémové knihovny a cesty, kde byli nalezeny předchozí závislosti.

Pokud se stále nevyřešili všechny závislosti, dochází k prohledávání nainstalovaných knihoven.

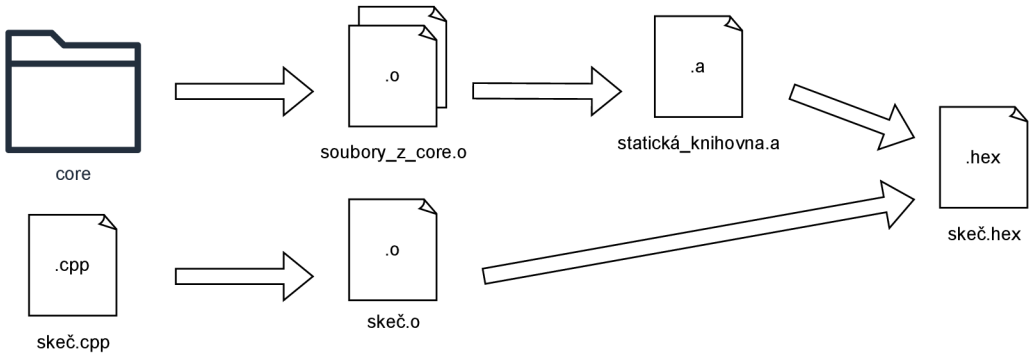
2.4.3 Překlad kódu

Skeč je kompilována pomocí překladače specifického pro architekturu daného mikrokontroléru. Popis mikrokontroléru se nachází v souboru `boards.txt` a obsahuje mimo jiné taky

popis čipu na desce. Překlad kódu na daný čip je popsán v souboru `platform.txt`. Detailní popis těchto souborů se nachází v kapitole 3.

Překlad pak probíhá standardním způsobem. Tedy všechny zdrojové soubory jsou kompilovány do `.o` souborů. Takto vytvořené soubory jsou pak, až na hlavní soubor skeče (vzniklý překladem výsledku pre-procesu), jsou následně spojeny do statické knihovny. Která je poté spojena s hlavním souborem skeče.

Výsledkem překladu je `.hex` soubor, který je později nahrán do mikroprocesoru.



Obrázek 2.3: Schéma znázorňující kompilaci v Arduino IDE. Vstupní soubor „skeč.cpp“ je výsledkem preprocesoru. Soubory z adresáře „core“, neboli jádra, jsou definovány pro každý mikrokontrolér zvlášť a jsou součástí podpůrného balíku v Arduino IDE. Výsledkem překladu je soubor „skeč.hex“, který se po kompilaci nahraje do mikrokontroléru.

Kapitola 3

Podpora mikrokontrolérů a vývojových desek v Arduino IDE

V této kapitole bude popsán způsob, jak je v prostředí Arduino IDE zajištěna podpora desek. Klíčovým krokem pro rozšíření podpory je pochopení struktury rozšiřovacích balíků a konfiguračních souborů, které jsou klíčovými prvkami pro správné fungování nově přidaných zařízení.

Nejprve bude popsán obsah a struktura rozšiřovacích balíků. Tedy jaké soubory a komponenty jsou nezbytné pro vytvoření rozšiřovacího balíku. Dále se bude věnovat jednotlivým konfiguračním souborům. U nich se zaměříme na jednotlivé parametry, které souvisí jak s deskami, tak i konkrétními mikrokontroléry.

3.1 Obsah rozšiřovacího balíku

Rozšiřovací balíky jsou určeny pro přidání nové funkcionality do prostředí Arduino IDE. Tyto balíky mohou být přidány do prostředí pomocí správce desek. Každý takovýto balík musí být popsaný souborem ve formátu JSON, aby byl dostupný ve správci desek ke stažení z internetu. Balíky lze také nainstalovat manuálně nakopírováním balíku do adresáře balíků (ve Windows se nachází na `C:\Users\user_name\AppData\Local\Arduino15\packages` , kde `user_name` je název uživatelského profilu, na kterém je Arduino IDE nainstalováno.).

Balík má hlavní 2 hlavní části:

- nástroje
- popis hardware

Nástroje, které balík může obsahovat závisí na tvůrci balíku. Nejčastěji se zde ale nachází:

- překladače pro mikrokontroléry
- nástroje pro nahrání kódu na čip

Popis Hardware pak obsahuje konfigurační soubory, které jsou nutné aby prostředí rozeznalo nové mikrokontroléry a vědělo jak přeložit kód pro daný čip. Dále je zde přítomen core (jádro) pro daný mikrokontrolér, který je součástí překladu viz kapitola 2.4.

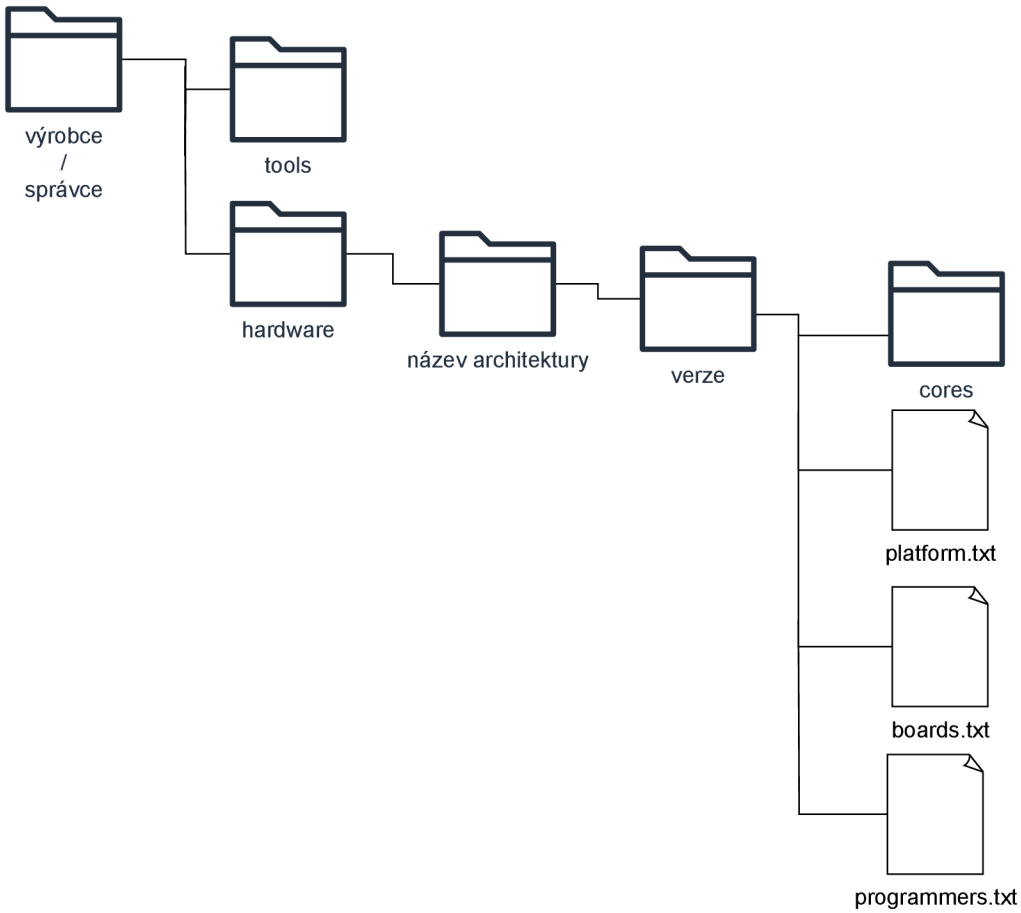
Součástí popisu hardware jsou 3 konfigurační soubory:

- platform.txt - popis CPU architektury osazené na desce
- boards.txt - popis parametrů desky
- programmers.txt - popis externích programátorů (využívá se pro nahrání bootloaderu na mikrokontroler)

3.2 Struktura rozšiřovacího balíku

Rozšiřovací balík má stanovenou strukturu, kterou je nutno dodržet, aby ho prostředí Arduino IDE rozpoznalo jednotlivé části rozšiřovacího balíku. Adresáře tools, hardware, cores musí být pojmenovány takto.Názvy ostatních adresářů se však musí dodržovat pouze pro další reference v popisu - tedy samotné prostředí tyto názvy nijak samo neinterpretuje.

Názvy architektury jsou navíc „case sensitive“ (záleží u nich na velikosti písma).



Obrázek 3.1: Schéma popisující strukturu a části rozšiřovacího balíku do prostředí Arduino IDE. Schéma bylo vytvořeno na základě struktury vestavěného balíku podpory pro mikrokontroléry Arduino. Adresář cores obsahuje jednotlivá jádra, která jsou součástí překladu. Soubory platform.txt, boards.txt a programmers.txt jsou konfigurační soubory.

3.3 Konfigurační soubory

Jednotlivé konfigurační soubory jsou navrženy jako seznam hodnot typu „klíč=hodnota“, které jsou odděleny novým řádkem. Při definování hodnot lze použít referenci na jiné hodnoty. Na ty se pak odkazujeme pomocí jejich klíče ve tvaru „{klíč}“.

```
compiler.path=/tools/g++_arm_none_eabi/bin/
compiler.c.cmd=arm-none-eabi-gcc
[....]
recipe.c.o.pattern={compiler.path}{compiler.c.cmd}
```

Obrázek 3.2: Ukázka konfiguračního souboru pro ilustraci formátu. Konfigurační soubory jsou mají formát seznamu hodnot klíč=hodnota, které jsou odděleny novým řádkem. Reference „{compiler.path}“ bude nahrazena hodnotou „/tools/g++_arm_none_eabi/bin/“.[2]

3.3.1 Globálně předdefinované hodnoty

Některé hodnoty jsou v konfiguračních souborech již předdefinovány. Jedná se hlavně o obecné konstanty. Tyto hodnoty jsou dostupné ve všech konfiguračních souborech.

V tabulce 3.1 je přehled vybraných globálních hodnot.

Klíč vlastnosti	**Popis**
{runtime.platform.path}	Absolutní cesta k adresáři s platformou. Tedy adresář „verze“ v obrázku 3.1
{runtime.hardware.path}	Absolutní cesta k adresáři s popisem hardware. Tedy adresář „název architektury“ v obrázku 3.1
{build.source.path}	Absolutní cesta k adresáři se zdrojovými soubory skeče
{build.path}	Absolutní cesta k adresáři se dočasnými soubory překladu
{build.project_name}	Název skeče/projektu

Tabulka 3.1: Popis globálně definovaných hodnot podle jejich klíče.[3]

3.3.2 boards.txt

Tento konfigurační soubor slouží pro definování nových desek. Desky definované v tomto souboru se zobrazí v Arduino IDE při výběru cílové desky. Tento soubor slouží pouze pro deklaraci základních vlastností a o tom jakým čipem je deska osazena - neboli jakou platformu deska využívá. Také zde lze definovat konstanty pro překlad, ale ostatní informace o kompilaci a informace o nahrávání kódu jsou definovány v souboru platform.txt.

Toto členění umožňuje snazší zjištění podpory pro různé desky osazené stejným mikrokontrolérem. Pokud bychom tedy chtěli přidat desku osazenou čipem, který již má v prostředí Arduino IDE zajištěnou podporu, zadefinujeme pouze parametry konkrétní desky a parametr „ID.build.core“ nastavíme na hodnotu odpovídající core (jádru) pro daný čip. Můžeme tedy využívat i podporu čipu z jiných balíku, ale před název core musí začínat názvem výrobce/právce(viz obrázek 3.1) ve tvaru „výrobce:core“ (například arduino:avr[3]).

[2]Převzato z https://arduino.github.io/arduino-cli/0.36/platform-specification/.

Při instalaci nového balíku, ale není zaručeno, že se v cílovém IDE nachází balík, z kterého by se využívala definice čipu, který je osazen na nové desce.

Všechny parametry v tomto souboru začínají ID desky. Toto ID slouží k odkazování se na desku v ostatních hodnotách. Parametry jsou tedy ve tvaru:

```
IDdesky.vlastnost1 = xxxx
IDdesky.vlastnost3.parametr = yyyy
```

V této ukázce má deska s ID = „IDdesky“ nastavenou vlastnost s identifikátorem „vlastnost1“ nastavenou na „xxxx“

Parametry pro definici desky:

Klíč vlastnosti	**Popis**
ID.name	Název desky, který se zobrazí uživateli (například v prostředí Arduino IDE).
ID.build.board	Hodnota, která se převede na makro v procesu překladu. Makro pak má tvar „ARDUINO_{build.board}“. Toto makro se pak dá využít k podmíněnému větvení kódu pomocí #ifdef.
ID.build.core	Specifikace jádra. Odpovídá názvu adresáře ve kterém se jádro nachází.
ID.build.mcu	Název mikrokontroléru, kterým je deska osazena. Během překladu je pak tato vlastnost dostupná pod „build.mcu“
ID.build.f_cpu	Frekvence jádra. Během překladu je pak tato vlastnost dostupná pod „build.f_cpu“
ID.build.variant	Varianta sestavení. Během překladu je pak tato vlastnost dostupná pod „build.variant“
ID.vid.x	VID (vendor ID) zařízení, sloužící k automatické detekci připojené desky. $x \geq 0$ - můžeme definovat více hodnot, kdy každá hodnota má jiné x.
ID.pid.x	PID (product ID) zařízení, sloužící k automatické detekci připojené desky. $x \geq 0$ - můžeme definovat více hodnot, kdy každá hodnota má jiné x.
ID.upload.maximum_size	Velikost dostupného úložiště pro program
ID.upload.maximum_data_size	Velikost dostupné dynamicky alokované paměti pro globální proměnné

Tabulka 3.2: Popis základních parametrů konfiguračního souboru boards.txt pro popis podporovaných desek v prostředí Arduino IDE.[3]

3.3.3 platform.txt

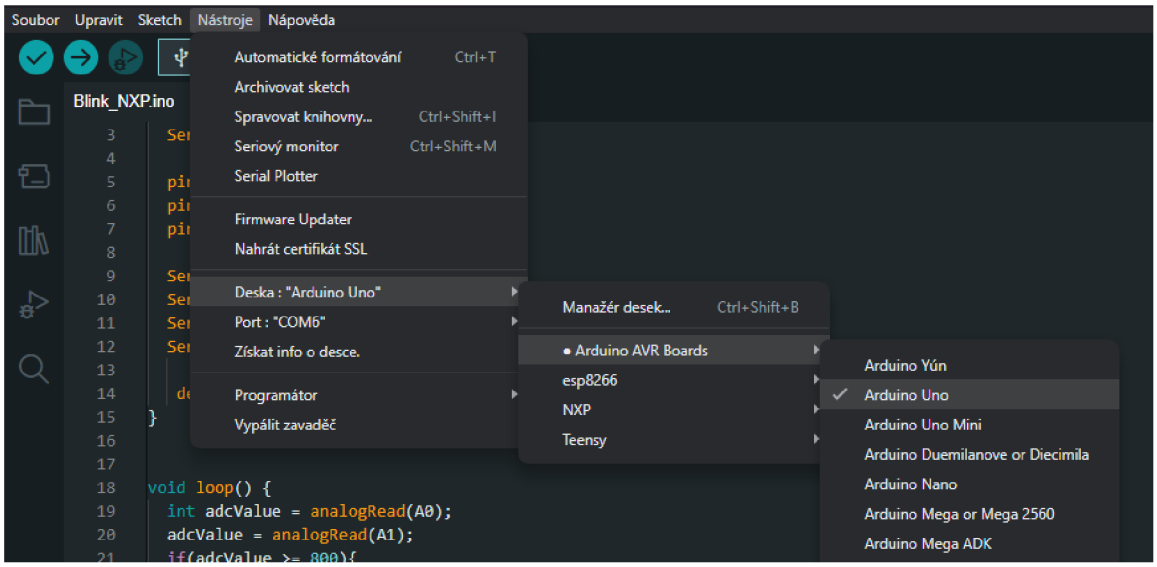
Tento konfigurační soubor slouží k definici platformy. Hlavní část tohoto souboru jsou parametry specifikující překlad a nahrání kódu. Ty se v prostředí Arduino IDE nazývají recepty.[3]

Každý recept tedy reprezentuje konkrétní příkazy, které Arduino CLI volá. Autor tohoto dokumentu tak má plnou kontrolu nad tím, jak se budou zdrojové kódy z core a ze skeče překládat včetně všech parametrů. Také má kontrolu nad parametry sestavení výsledného binárního souboru, který bude nahrán do mikrokontroléru.

V tomto souboru může využívat všech pomocných nástrojů, které se nachází v balíku v adresáři tools viz obrázek 3.1

Soubor platform.txt musí mít definované hodnoty „name“ a „version“. Hodnota „name“ se pak zobrazí v menu pro vybrání desek (bude se tak jmenovat kategorie v menu, pod kterou budou zobrazeny všechny přidané desky). Hodnota „version“ je zatím nevyužita, ale v budoucím vývoji by měla být použita.[3] Využití této hodnoty je ilustrováno na obrázku 3.3.

```
platform.txt:
    name=Arduino AVR Boards
    version=1.8.6
```



Obrázek 3.3: Na obrázku lze vidět, jak se parametr „name“ zobrazí v prostředí Arduino IDE. Hodnoty v ukázce jsou převzaty z konfiguračního souboru platform.txt z vestavěného podpůrného balíku od Arduino.

Parametry konfiguračního souboru pro překlad:

Klíč vlastnosti	**Popis**
recipe.c.o.pattern	Obsah příkazu pro překlad souborů v jazyce C.
recipe.cpp.o.pattern	Obsah příkazu pro překlad souborů v jazyce C++. Pokud je potřeba soubory s příponou .cpp, .cxx a .cc překládat jinými příkazy, lze dodefinovat i recipe.cxx.o.pattern, recipe.cc.o.pattern
recipe.S.o.pattern	Obsah příkazu pro překlad souborů v assembleru.
recipe.xxx.cmd	Příkaz pro volání překladače pro daný jazyk („xxx“ je nahrazeno zkratkou příslušného jazyka - c, cpp, cxx, cc, S).
{includes}	Nahrazeno za seznam cest, které mají být zahrnuty v kompilaci. Cesty mají tvar „-I/cesta“
{source_file}	Absolutní cesta k překládanému zdrojovému souboru.
{object_file}	Absolutní cesta k výstupnímu souboru překladu.
compiler.ar.cmd	Příkaz pro volání překladače pro překlad statické knihovny z core (jádra).
compiler.ar.flags	Příznaky pro překlad statické knihovny z core (jádra).
recipe.ar.pattern	Kompletní příkaz pro sestavení statické knihovny z core (jádra).

Tabulka 3.3: Popis vybraných parametrů pro překlad v souboru platform.txt[3]

Parametry konfiguračního souboru pro sestavení programu:

Klíč vlastnosti	Popis
compiler.xxx.elf.cmd	Příkaz pro volání linkeru (sestavovacího programu) pro daný jazyk („xxx“ je nahrazeno zkratkou příslušného jazyka - c, cpp, cxx, cc, S).
compiler.xxx.elf.flags	Příznaky pro sestavení programu pro daný jazyk („xxx“ je nahrazeno zkratkou příslušného jazyka - c, cpp, cxx, cc, S).
recipe.xxx.combine.pattern	Obsah celého příkazu pro sestavení programu daného jazyka („xxx“ je nahrazeno zkratkou příslušného jazyka - c, cpp, cxx, cc, S).

Tabulka 3.4: Popis vybraných parametrů pro sestavení programu v souboru platform.txt[3]

Parametry konfiguračního souboru pro překlad objektových souborů do binárních:

Klíč vlastnosti	Popis
recipe.objcopy.xxx.pattern	Obsah celého příkazu pro vytvoření binárních souborů pro danou příponu souboru („xxx“ je nahrazeno příponou souboru - například .hex, nebo .eep).

Tabulka 3.5: Popis vybraných parametrů v souboru platform.txt pro překlad objektových souborů na binární soubory, které se budou nahrávat na mikrokontrolér.[3]

Parametry konfiguračního souboru pro výpočet velikosti binárních souborů:

Klíč vlastnosti	Popis
recipe.size.pattern	Obsah celého příkazu pro výpočet velikosti binárních souborů.
recipe.size.regex	Regulární výraz pro zjištění využitého úložiště.
recipe.size.regex.data	Regulární výraz pro zjištění velikosti dynamicky alokované paměti, kterou využívají globální proměnné.

Tabulka 3.6: Popis parametrů v souboru platform.txt pro výpočet velikosti binárních souborů.[3] Dostupné velikosti pamětí jsou definovány v boards.txt (viz tabulka 3.2)

Parametry konfiguračního souboru pro nahrání kódu na mikrokontrolér:

Klíč vlastnosti	Popis
tools.avrdude.cmd.path	Absolutní cesta k programu, pomocí kterého se bude kód nahrávat. V popisu příkazu pak bude dostupné pod „{cmd.path}“
tools.avrdude.config.path	Absolutní cesta ke konfiguračnímu souboru pro program, pomocí kterého se bude kód nahrávat. V popisu příkazu pak bude dostupné pod „{config.path}“
tools.xxx.upload.pattern	Obsah celého příkazu pro nahrání kódu na mikrokontrolér. „xxx“ odpovídá názvu adresáře, kde se nachází program pomocí, pomocí kterého se kód nahraje.

Tabulka 3.7: Popis parametrů v souboru platform.txt pro nahrání kódu na mikrokontrolér.[3]

3.3.4 programmers.txt

Tento konfigurační soubor slouží pro definici externích programátorů. Ty jsou použity pro přehrání bootloaderu, nebo pro nahrání skeče prostřednictvím těchto programátorů. Tento soubor má podobnou strukturu jako `boards.txt`.

```
[......]
arduinoasisp.name=Arduino as ISP
arduinoasisp.protocol=stk500v1
arduinoasisp.program.speed=19200
arduinoasisp.program.tool=avrdude
arduinoasisp.program.extra_params=-P{serial.port} -b{program.speed}
[......]
```

Obrázek 3.4: Ukázka obsahuj souboru programmers.txt.[3]

[3]Převzato z https://arduino.github.io/arduino-cli/0.29/platform-specification/

Kapitola 4

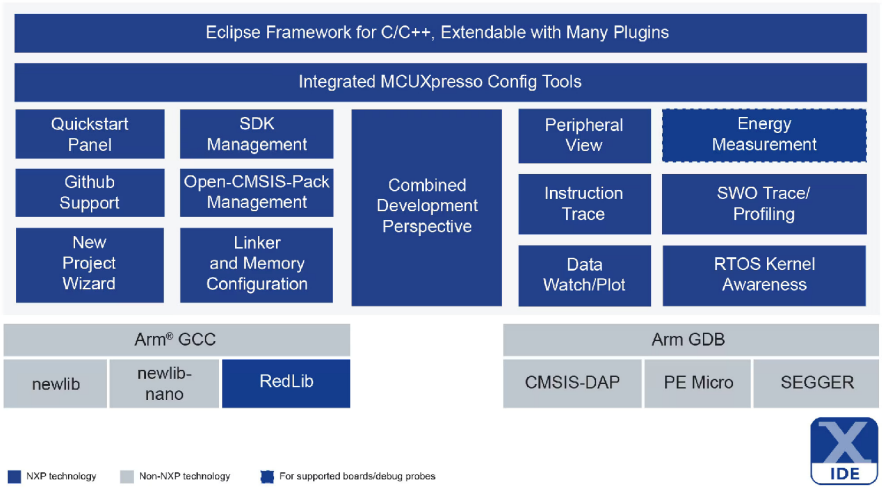
Vývojové prostředí MCUXpresso

V této kapitole se seznámíme s vývojový prostředí MCUXpresso IDE a jeho SDK (sadami vývojářských nástrojů) pro jednotlivé desky. Také bude popsána struktura projektů v MCUXpresso IDE a způsob překladu projektů. Právě struktura projektů a způsob, jakým se projekty překládají je klíčové pro zajištění podpory mikrokontrolérů od firmy NXP Semiconductors ve vývojovém prostředí Arduino IDE.

4.1 MCUXpresso IDE

MCUXpresso IDE je vývojové prostředí od společnosti NXP. Je založeno na Eclipse IDE a lze v něm vyvíjet všechny desky od firmy NXP osazené mikrokontroléry s jádrem Arm Cortex-M.[7]

Prostředí tedy nenabízí jen samotný editor kódu, ale také například konfigurační a ladící nástroje. Pro kompilaci používá ARM GNU toolchain (kolekce nástrojů k programování a překlad kódu) [7]. Všechny součásti vývojového prostředí jsou ukázány v blokovém schématu na obrázku 4.1.



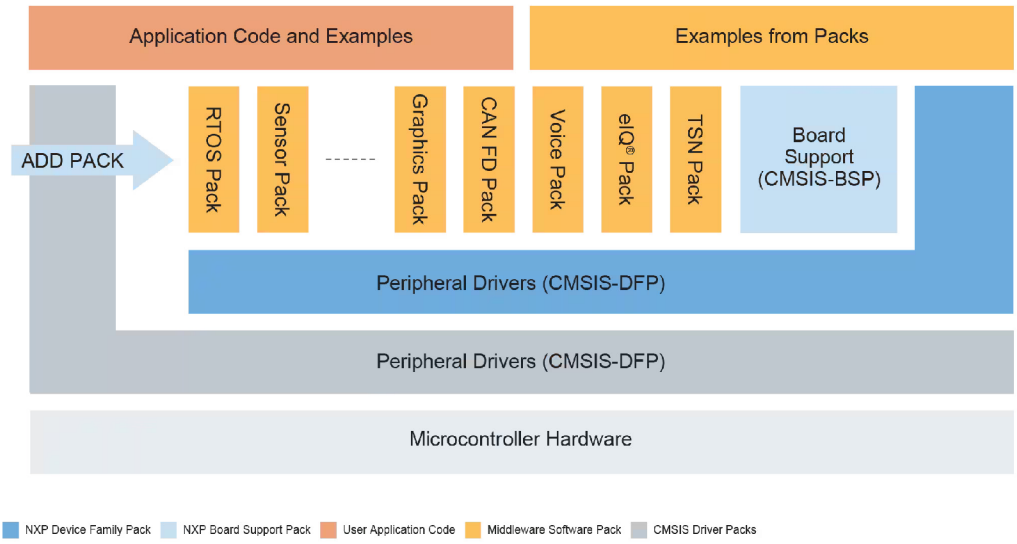
Obrázek 4.1: Blokové schéma znázorňující jednotlivé součásti MCUXpresso IDE.[1]

[1]Obrázek převzat z https://www.nxp.com/assets/images/en/block-diagrams/MCUXpressoIDE_BD_HR2.jpg.

Jednotlivé projekty mohou být vyvíjeny v jazyce C nebo C++. Pro podporu jednotlivých desek lze do prostředí nainstalovat SDK (sady vývojářských nástrojů).

SDK jsou dodávány ke každé desce od společnosti NXP. Jedná se o soubor nástrojů pro efektivní vývoj. SDK jsou kompatibilní s řadou toochainů a jsou otestovány pomocí statické analýzy. Součástí této sady jsou:[6]

- nástroje pro podporu desky
- ovladače periférií
- ukázky kódu
- integrovaný operační systém reálného času (RTOSes)
- další nástroje pro snazší vývoj a efektivnější práci s deskou



Obrázek 4.2: Blokové schéma znázorňující jednotlivé součásti sady vývojářských nástrojů MCUXpresso.[3]

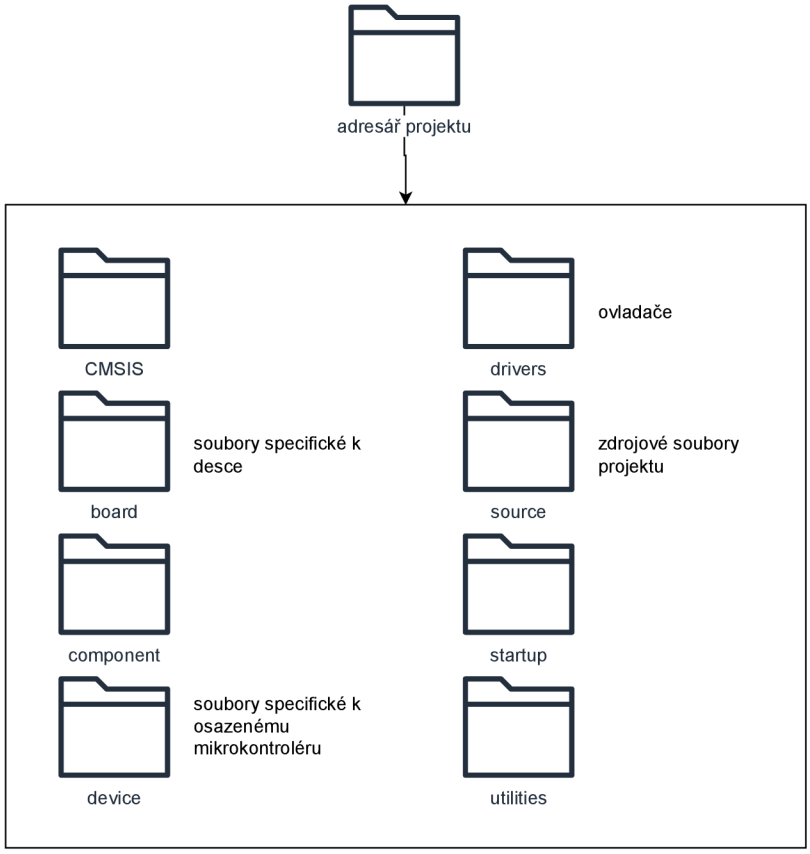
[3]Obrázek převzat z https://www.nxp.com/assets/images/en/block-diagrams/MCUXPRESSO-SDK-BD1.png.

4.2 Struktura projektů v MCUXpresso IDE

Projekty v MCUXpresso IDE mají oproti skečím v Arduino IDE komplexnější strukturu. Projekt obsahuje několik adresářů, které obsahují například:

- ovladače pro periferie
- hlavičkové a konfigurační soubory specifické pro danou vývojovou desku
- hlavičkové a konfigurační soubory specifické pro daný mikrokontrolér
- soubory s kódem pro start mikrokontroléru
- adresář s kódem, který je určen pro editaci uživatelem

Tato struktura je zavedená v každém projektu již při vytvoření daného projektu v prostředí MCUXpresso IDE. Struktura projektu je zobrazena na obrázku 4.3.

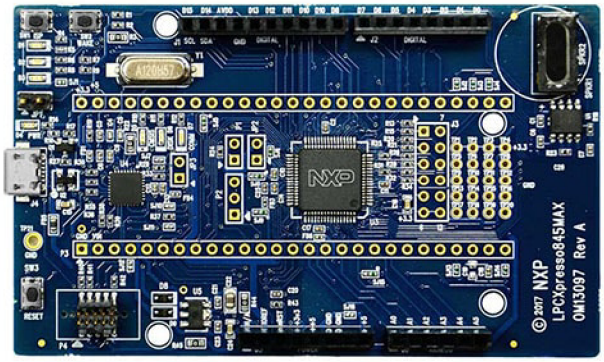


Obrázek 4.3: Schéma znázorňující strukturu projektů v MCUXpresso IDE. Každý projekt vytvořený v MCUXpresso IDE má tuto strukturu. Zdrojové soubory, které uživatel edituje jsou v adresáři source. Ostatní adresáře obsahují soubory specifické k danému projektu. Obsah jednotlivých adresářů se liší podle vyvíjené desky, nebo mikrokontroléru a také podle použitých periférií. Pomocí SDK lze naimportovat ukázky kódů, které ukazují obsluhu jednotlivých periférií.

4.3 Vybrané mikrokontroléry

4.3.1 LPCXpresso845-MAX

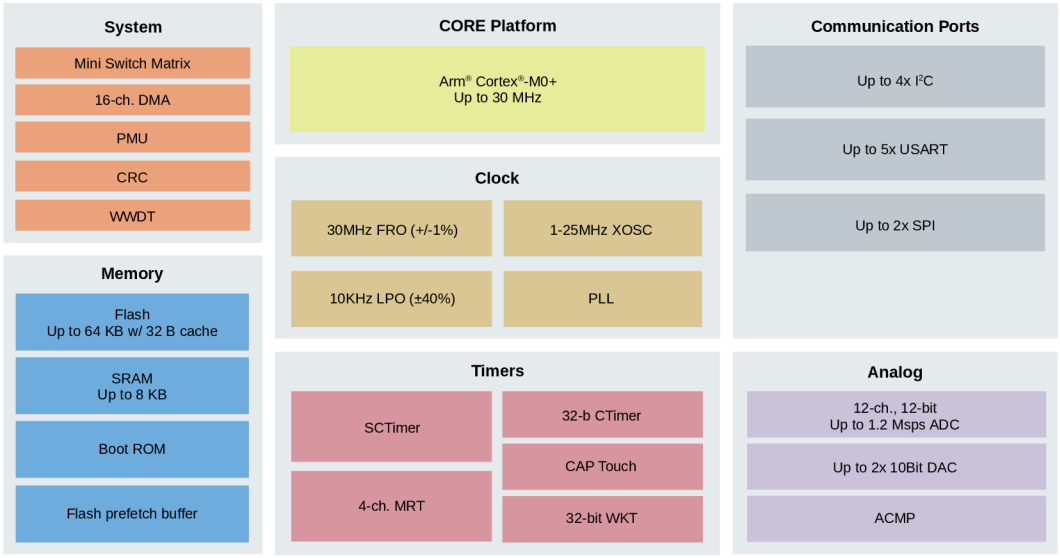
Hlavním důvodem pro zvolení této desky byla kompatibilita rozložení portů s deskou Arduino UNO. Deska LPCXpresso845-MAX je osazená čipem z rodiny LPC84x, které jsou založeny na 32 bitovém jádru Arm Cortex-M0+. Deska je dále osazena čipem LPC11U35, který zabezpečuje ladění kódu - rozhraní CMSIS-DAP (debug probe).



Obrázek 4.4: Vybraný mikrokontrolér LPCXpresso845-MAX, pro který bude v této práci vytvořena podpora do Arduino IDE.[5]

Mikrokontrolér LPC845 je z rodiny mikrokontrolérů LPC84x, což jsou nízkoenergetické čipy, taktované až na 30 MHz. Mají dostupnou 64kB Flash paměť a 16kB RAM paměti. Nabízí 12-ti bitový ADC převodník s 12 kanály a 10-ti bitový DAC převodník s 2 kanály. Podporují sériovou komunikaci, I^2C, SPI, UART a 54 GPIO.[8]

[5]Obrázek převzat z https://www.nxp.com/design/software/development-software/mcuxpresso-software-and-tools-/lpcxpresso845-max-board-for-lpc84x-mcu-family:OM13097.



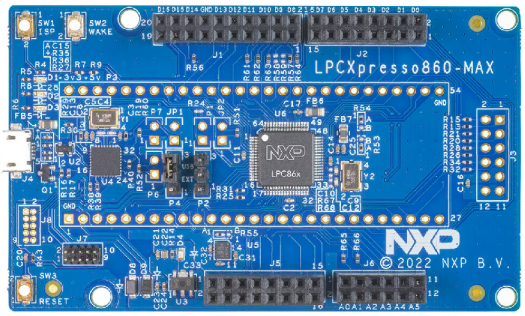
Obrázek 4.5: Blokové schéma jednotlivých součástí mikrokontrolérů z rodiny LPC84x.[6]

4.3.2 LPCXpresso860-MAX

Tato deska byla vybrána pro ověření postupů získaných při tvorbě podpůrného balíku pro desku LPCXpresso845-MAX. Jedná se totiž o podobnou desku s novějším čipem. Má také kompatibilní rozložení portů s deskou Arduino UNO.

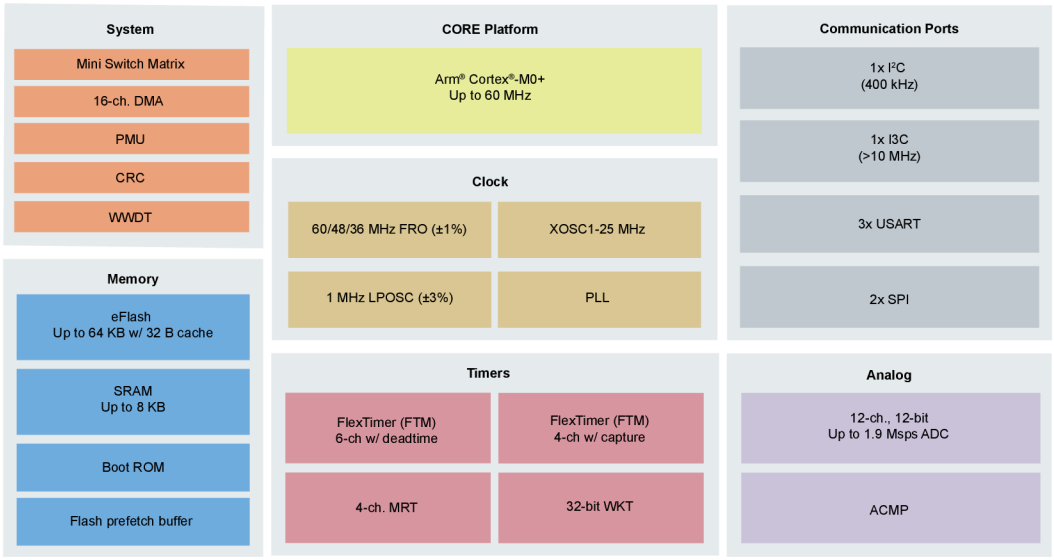
Deska LPCXpresso860-MAX je osazená čipem z rodiny LPC86x, které jsou založeny na 32 bitovém jádru Arm Cortex-M0+. Deska je dále osazena čipem LPC11U35, který zabezpečuje ladění kódu - rozhraní CMSIS-DAP (debug probe).

[6] Obrázek převzat z https://www.nxp.com/assets/images/en/block-diagrams/LPC84x-Fam-BD1-SVG.svg.



Obrázek 4.6: Vybraný mikrokontrolér LPCXpresso860-MAX, pro na kterém bude v této práci ověřen postup vytváření podpory do Arduino IDE.[8]

Mikrokontrolér LPC845 je z rodiny mikrokontrolérů LPC86x, což jsou nízkoenergetické čipy, taktované až na 60 MHz. Mají dostupnou až 64kB Flash paměť a až 8kB RAM paměti. Nabízí 12-ti bitový ADC převodník s 12 kanály. Podporují I^3C, I^2C, SPI, UART a 54 GPIO.[8]



Obrázek 4.7: Blokové schéma jednotlivých součástí mikrokontrolérů z rodiny LPC86x.[9]

[8]Obrázek převzat z https://www.nxp.com/design/design-center/software/development-software/mcuxpresso-software-and-tools-/lpcxpresso-boards/lpcxpresso860-max-development-board-for-lpc860-mcus:LPCXPRESSO860-MAX.

[9]Obrázek převzat z https://www.nxp.com/assets/images/en/block-diagrams/LPC86X-BD.svg.

Kapitola 5

Koncepce podpory mikrokontrolérů NXP

V této kapitole budou nejprve popsány stávající mikrokontroléry, které mají podporu v prostředí Arduino IDE. Dále bude popsáno jak bude vytvořen rozšiřovací balík pro zabezpečení podpory desky LPCXpresso845-MAX v prostředí Arduino. Bude také popsána navržená struktura a také které funkce je nutno implementovat pro základní funkcionalitu.

5.1 Rešerše stávajících řešení

Společnost NXP Semiconductors aktuálně nenabízí přímou podporu svých mikrokontrolérů a vývojových desek v prostředí Arduino IDE. Na trhu ale existují desky Teensy, které jsou osazené mikrokontrolérem od společnosti NXP, které podporu v prostředí Arduino IDE nabízí.

Společnost Teensy vyrábí několik vývojových desek se stejným jménem, u kterých má zabezpečenou podporu v Arduino IDE, kterou nazývá Teensyduino.[10]

Konkrétně vývojové desky od verze Teensy LC jsou osazeny mikrokontroléry od Společnosti NXP Semiconductors.

5.1.1 Teensyduino

Teensyduino je rozšiřovací balík od společnosti Teensy pro podporu vývojových desek Teensy v prostředí Arduino IDE. Toto rozšíření také nabízí kompatibilitu s některými knihovnami, které jsou dostupné v prostředí pomocí správce knihoven.

Konkrétně se jedná o knihovny zabezpečující:[9]

- podporu displejů
- komunikaci
- podporu senzorů a dalších vstupních prvků
- řízení a podporu výstupních signálů
- čas a časování
- práci s daty

Podrobný výpis podporovaných knihoven a kompatibilita s jednotlivými vývojovými deskami je dostupný na adrese: https://www.pjrc.com/teensy/td_libs.html.

Nástroje Součástí tohoto rozšiřovacího balíku je řada nástrojů od společnosti Teensy pro překlad, nahrání a monitorování kódu. Po analýze rozšiřovacího balíku lze pozorovat, že k překladu kódu využívá GNU Arm Embedded Toolchain (kolekce nástrojů k programování a překlad kódu). Tedy stejně jako MCUXpresso IDE. Navíc ale obsahuje již zmiňované pomocné nástroje.

Jádra Pro jednotlivé mikrokontroléry Teensy jsou jsou vyvíjena od této společností. Po jejich analýze nepřipomínají svojí strukturou ani obsahem, projekty z MCUXpresso IDE, které jsou popsány v kapitole 4.

5.2 Core

Pro implementaci core (jádra) bude použitý projekt vygenerovaný v MCUXpresso IDE. Jelikož prázdný projekt neobsahuje žádné ovladače, ani konfigurace desky, bude se dále využívat ukázek kódu z SDK pro daný mikrokontrolér. Z těchto SDK příkladů se budou brát jednotlivé ovladače a jiné potřebné soubory SDK. Pomocí konfiguračního nástroje pro namapování pinů mikrokontroléru na GPIO desky se namapují piny tak, aby odpovídali rozložení a funkcionalitě pinů desky Arduino UNO.

Navíc musí být vytvořen celý nový soubor s funkcí `main`. V těle funkce `main` se musí na začátku zavolat funkce `setup` a následně v nekonečné smyčce volat funkce `loop`. Těla těchto funkcí se při překladu doplní ze skeče.

Dalšími funkcemi, které musí být v core implementované jsou funkce jazyka Arduino pro zajištění kompatibility. Jednotlivé funkce budou popsány v kapitole 7, která se věnuje implementaci jádra.

5.3 Konfigurační soubory

Pro zajištění kompatibility vybraných desek musíme vytvořit konfigurační soubory `boards.txt` a `platform.txt`. Obsah souboru `boards.txt` bude vytvořen podle specifikací mikrokontroléru na deskách. Obsah `platform.txt` bude vytvořen pomocí výpisů z terminálu v MCUXpresso IDE.

Překladač, který využívá MCUXpresso IDE bude samostatně umístěn v adresáři tools. Zde bude také umístěn program pro nahrávání kódu na desku.

Příkazy, včetně jejich parametrů, budou pak získány z podrobného výpisu z konzole v MCUXpresso IDE při procesu překladu a sestavení. Tyto příkazy pak budou rozděleny do jednotlivých parametrů v souboru platform.txt.

Obdobným způsobem budou získány a zpracovány příkazy pro nahrání kódu na mikrokontrolér.

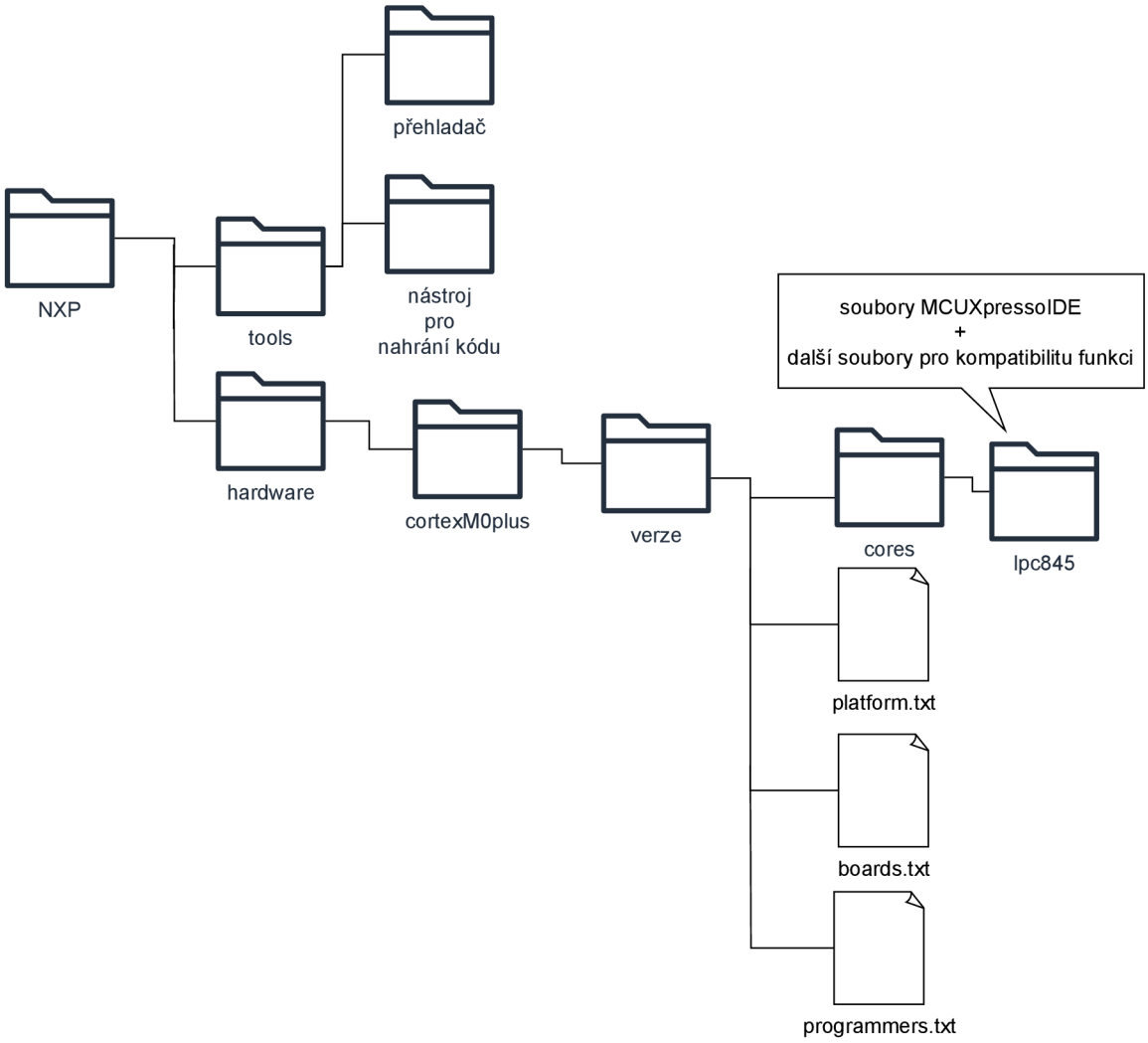
5.4 Navržená struktura rozšiřovacího balíku

Rozšiřovací balík pro podporu mikrokontrolérů a vývojových desek v prostředí Arduino IDE bude používat strukturu popsanou na obrázku 3.1.

Pro překlad bude v adresáři tools umístěn překladač, který používá prostředí MCUXpresso IDE. V tomto adresáři bude také umístěn nástroj pro nahrání kódu na desku/mikrokontrolér. Oba tyto nástroje pak budou použity v konfiguračních souborech.

Pro každý mikrokontrolér nebo vývojovou desku bude vytvořen adresář obsahující jádro a konfigurační soubory pro toto zařízení. Tento adresář bude umístěn v adresáři hardware. Soubory které budou součástí jádra budou převzaty z projektů z prostředí MCUXpresso IDE.

Celé navržené schéma je popsáno schématem na obrázku 5.1.



Obrázek 5.1: Navržené schéma výsledné struktury rozšiřovacího balíku.

Kapitola 6

Sestavení podpůrného balíku

Tato kapitola popisuje postupy pro složení podpůrného balíku pro mikrokontroléry od NXP. Nejprve budou popsány postupy vytvoření balíku pro desku LPCXpresso845-MAX. Tyto postupy budou pak znovu aplikovány pro podporu mikrokontroléru LPCXpresso860-MAX, tomu se věnuje kapitola 9.

Součástí popisu bude získání nástrojů pro překlad kódu a nástroje pro nahrání kódu na desku. Dále bude napsáno z čeho vychází jádra. Detailní popis implementace jádra se nachází v kapitole 7.

Také bude popsán způsob zahrnutí testovacích skečí k jádru. Jednotlivé testy jsou pak popsány v kapitole 8.

6.1 Vytvoření balíku

Podpůrný balík musí být vytvořen v adresáři Arduina IDE, který je určen právě pro podpůrné balíky. Ve Windows je to, po výchozí instalaci, adresář:
`C:\Users\user_name\AppData\Local\Arduino15\packages` , kde `user_name` je název uživatelského profilu, na kterém je Arduino IDE nainstalováno.

V tomto adresáři byla tedy vytvořen adresář se stejnou strukturou jako v návrhu na obrázku 5.1. V následujících kapitolách pak budou popsány jednotlivé části tohoto balíku.

Při vytváření jádra byly některé soubory převzaté z prototypu podpůrného balíku od Roberta Havránka ze společnosti NXP. Tento prototyp sloužil k nastínění řešení pro základní podporu. Každý převzatý soubor byl ale upraven v řešení této práce. Původní řádky jsou v souborech označeny komentářem. Konkrétně byly převzaty a poupraveny tyto soubory:

- boards.txt
- platform.txt
- keywords.txt
- Arduino.h
- wiring_analog.c
- wiring_digital.c
- arduino_pins.h

6.2 Nástroje

V této části budou popsány nástroje, které jsou důležité pro zajištění podpory, jsou překlad kódu a nástroj pro nahrání kódu na desku.

6.2.1 Překlad kódu

Prvním nástrojem, který je nutný zajistit je překladač kódu. Jelikož je řešení založeno na SDK balíku z MCUXpresso, využívá řešení stejný překladač jako MCUXpresso. Tedy do adresáře `tools` (viz obrázek 5.1) byl umístěn adresář `arm-none-eabi-gcc`, který obsahuje překladač z MCUXpresso.

6.2.2 Nahrání kódu na desku

Dále je nutné zajistit nahrání kódu na desku. K tomu MCUXpresso využívá nástroj redlink. Ten slouží nejen pro nahrání kódu, ale i pro ladění kódu. Tedy do adresáře `tools` (viz obrázek 5.1) byl umístěn adresář `redlink`.

Nástroj redlink využívá pomocný ladící čip umístěný na desce. Tento čip zabezpečuje jak USB komunikaci s deskou, ale i komunikaci s redlinkem. Pokud bychom ale nechtěli využívat redlink nabízí mikroprocesor LPC845 nahrání kódu pomocí technologie ISP.

ISP neboli In-System Programming umožňuje přehrát program ve flash paměti přes sériovou komunikaci s mikrokontrolérem po vyvolání ISP režimu pomocí programu v bootloaderu.[5] Tento způsob nahrání kódu na desku však nebyl využit po neúspěšných pokusech o navázání spojení s deskou v tomto režimu.

Při zkoušení této technologie jsem postupoval podle pokynů v manuálu desky LPCXpresso845-MAX pro uvedení desky do režimu ISP. Po uvedení do ISP režimu by pak po zaslání znaku „?“ přes UART rozhraní měla deska provést auto konfiguraci baudratu a odpovědět „Synchronized<CR><LF>“.[5] Avšak deska neodpovídá. Tento test jsme provedli se zástupci firmy NXP na více deskách LPCXpresso845-MAX i na deskách LPCXpresso860-MAX a žádná neodpovídala.

Po prozkoumání manuálů desek bylo pak zjištěno, že vodiče po kterých ISP komunikuje nejsou připojeny k ladícímu obvodu, tedy nejsou připojeny k USB konektoru na desce a bylo by nutné použít externí sériový převodník. Proto v práci nebyl použit tento způsob nahrání kódu na desku.

6.3 Konfigurační soubory

V této části bude popsán postup vytvoření jednotlivých konfiguračních souborů. Co je obsahem konfiguračních souborů, je popsáno v kapitole 3.3.

Všechny konfigurační soubory jsou vytvořeny na základě konfiguračních souborů vestavěného podpůrného balíku Arduino.

6.3.1 boards.txt

Většina hodnot tohoto konfiguračního souboru byla vytvořena na základě parametrů desky získaných z vývojového prostředí MCUXpresso, jeho konfiguračních nástrojů, také uživatelského manuálu[1]a parametrů čipu na desce[2]. Obsah tohoto souboru je ukázán na obrázku 6.1.

Název desky, který se ukáže ve vývojovém prostředí byl zvolen stejný jako na obalu vývojové desky. Parametry `vid` a `pid` byly získány ze správce zařízení v operačním systému Windows.

Parametry `upload.tool` a `upload.tool.default` odpovídají názvu nástroje pro nahrávání kódu – tento název bude použit v souboru `platform.txt`. Parametr `upload.protocol` není používán, ale bez jeho uvedení hlásí Arduino IDE, při pokusu o nahrání kódu na desku, chybovou hlášku „`A programmer is required to upload`“.

Všechny parametry `build` (viz 6.1), jako například frekvence procesoru, název jádra mikrokontroléru, a hodnoty pod klíčem `lpc_845_max.build.extra_flags`, které jsou důležité při překladu kódu, byly získány z vývojového prostředí MCUXpresso a z manuálu desky. Kromě tedy parametru `lpc_845_max.build.core`, který odpovídá názvu jádra – tedy názvu adresáře, ve kterém se soubory jádra nachází. Všechny hodnoty parametru `lpc_845_max.build.extra_flags` byly získány z vývojového prostředí MCUXpresso z výpisu v terminálu při překladu kódu.

[1]Dostupné na https://www.nxp.com/webapp/Download?colCode=UM11057.
[2]Dostupné na https://www.nxp.com/part/LPC845M301JBD64#/.

```
# LPC 845 MAX
# ---------------------------------------
lpc_845_max.name=LPCXpresso845MAX board
lpc_845_max.vid.0=0x1FC9
lpc_845_max.pid.0=0x0132
# upload
lpc_845_max.upload.tool=redlink
lpc_845_max.upload.name=LPC845
lpc_845_max.upload.tool.default=redlink
lpc_845_max.upload.protocol=redlink
lpc_845_max.upload.maximum_size=65536
lpc_845_max.upload.maximum_data_size=16352
# build
lpc_845_max.build.mcu=cortex-m0plus
lpc_845_max.build.f_cpu=18000000L
lpc_845_max.build.board=LPC845MAX
lpc_845_max.build.extra_flags=-D__NEWLIB__ -DCPU_LPC845M301JBD64 ...
lpc_845_max.build.core=lpc845
lpc_845_max.build.ld_flags=-T "{runtime.platform.path}\
        LPC845_ProjectC++_Debug.ld"
```

Obrázek 6.1: Obsah konfiguračního souboru pro vývojovou desku LPCXpresso845-MAX. Hodnota `lpc_845_max.build.extra_flags` byla zkrácena pro lepší přehlednost tohoto psaného textu (zbytek hodnoty nahrazeno „...“). Parametr `build.ld_flags` byl přidán pro přidání správné verze ladící knihovny při sestavení programu pro danou desku. Popis hodnot jednotlivých parametrů je v kapitole 6.3.1. Parametr `upload.name` byl přidán pro identifikaci desky při nahrávání kódu pomocí redlink.

6.3.2 platform.txt

Většina hodnot tohoto konfiguračního souboru byla vytvořena na základě výpisu hlášek z překladu zdrojových souborů z MCUXpresso IDE. Tedy jednotlivé hodnoty parametrů pro překladač a i použitý překladač pro jednotlivé soubory byly převzaty z těchto výpisů. Tyto parametry byly pak rozděleny do jednotlivých hodnot.

Překlad

Parametry, které se vykytovaly u všech příkazů jsou umístěny v souboru `boards.txt` do hodnoty `lpc_845_max.build.extra_flags`.

Parametry pro překlad jednotlivých souborů jazyka C (přípona souboru `.c`), C++ (přípona souboru `.cpp`) jsou téměř identické, liší se hlavně v použitém překladači. Přehled příkazů se nachází v tabulkách 3.3 a 3.4 v kapitole 3.3.3

Důležitou součástí receptů pro jejich překlad je ale použití parametru `-D`, kterým se definují konstanty důležité pro překlad. Hlavně jsou to parametry v tabulce 6.1.

Parametr	Popis
-DF_CPU=build.f_cpu	Makro F_CPU s hodnotou frekvence CPU zadanou v souboru `boards.txt`.
-DARDUINO=runtime.ide.version	Makro s hodnotou verze Arduino IDE.
-DARDUINO_build.board	Makro začínající „ARDUINO_“ následováno hodnotou názvu desky zadanou v souboru `boards.txt`. Například „ARDUINO_LPC845MAX“ pro překlad na desku s konfiguračním souborem na obrázku 6.1.
-DARDUINO_ARCH_build.arch	Makro začínající „ARDUINO_ARCH_“ následováno hodnotou názvu desky zadanou v souboru `boards.txt`. Obdobně jako předchozí makro. Toto řešení tuto hodnotu nemá definovanou. Makro tedy bude mít tvar `ARDUINO_ARCH_`.

Tabulka 6.1: Popis vybraných parametrů pro překladač v souboru platform.txt. Konkrétně parametry pro definice maker preprocesoru.

Nahrání kódu

Parametry pro nahrání kódu jsou v principu stejné jako parametry pro překlad. Jedná se o definici „receptu“ podle kterého se bude spouštět nástroj pro nahrání kódu. Přehled příkazů se nachází v tabulce 3.7 v kapitole 3.3.3

Hodnoty těchto parametrů byly opět získány z MCUXpresso. Konkrétně z výpisu příkazu pro nahrání programu na terminálu.

6.4 Jádro

Jádro pro mikrokontrolér bylo vytvořeno na základě 2 hlavních zdrojů:

- Vestavěný podpůrný balík Arduino IDE
- Příklady kódů z SDK balíku MCUXpresso

Dalším přínosem byl repositář ArduinoCore-API, který je dostupný na adrese https://github.com/arduino/ArduinoCore-API/tree/master. Ten obsahuje všechny definice Arduino funkcí bez implementace na konkrétní platformu. Při vytváření balíku ale bylo spíše použit vestavěný podpůrný balík Arduino IDE kvůli detailům implementace, které nejsou uvedeny ani na manuálových stránkách Arduino IDE (dostupných na https://www.arduino.cc/reference/en/).

Popis implementace jádra se nachází v kapitole 7.

6.5 Testovací příklady

Pro ověření implementované funkcionality jsou součástí jádra testovací skeče. Jelikož Arduino IDE nepodporuje ukázky přímo z podpůrného balíku, byla v něm vytvořena pomocná knihovna `lpc845test`, která obsahuje sadu testovacích skečí. Ty mají hlavně ověřit, že u nich dojde k úspěšnému překladu a nahrání na desku. Některé ale i ověřují základní funkcionalitu. O testování a jednotlivých testovacích skečích bude uvedeno v kapitole 8.

Kapitola 7

Implementace jádra pro LPCXPresso845-MAX

V této kapitole bude popsána implementace jádra pro podporu mikrokontroléru LPCXpresso845-MAX. Na základě tohoto postupu byla pak implementována podpora pro mikrokontrolér LPCXpresso860-MAX, jako ověření tohoto postupu (popsáno v kapitole 9).

Nejprve bude popsán postup vytváření jádra. Následně obecná struktura jádra. Dále pak budou popsány jednotlivé části a k ním příslušící soubory. U každé části bude také uvedeno na základě jakých příkladů z balíku SDK MCUXpresso byla daná kategorie implementována. Tyto části budou strukturovány podle dělení funkcí do skupin podle Arduino jazyka[2].

7.1 Postup vytvoření jádra

Při implementaci jádra byl uplatněn jednotný postup pro implementaci jeho částí. Části byly postupně implementovány po kategoriích podle rozdělení funkcí Arduino jazyka[2].

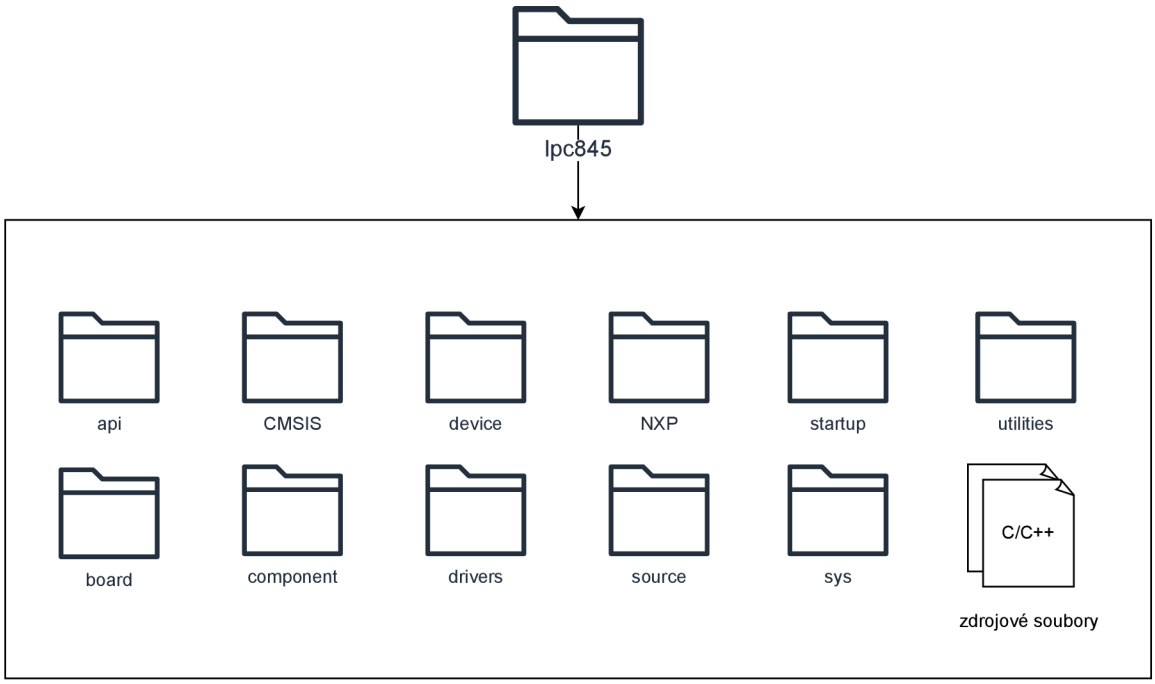
Nejprve jsem analyzoval funkcionalitu funkcí Arduino na webu, ale i jejich implementaci ve vestavěném balíku Arduino. Následně jsem prošel příklady z SDK balíku MCUXpresso. Následně jsem zkopíroval ovladače a další potřebné soubory z MCUXpresso projektu do vytvářeného jádra. Následně jsem implementoval funkce jazyka Arduino ve vytvářeném jádru. Funkcionalitu jsem ověřil na vestavěných příkladech, nebo jsem je ověřil na vlastních skečích.

Celý postup vytvoření jádra lze shrnout do následujících kroků:

1. Analýza implementované funkce / funkcionality a její API
2. Analýza implementace vestavěného balíku Arduino
3. Analýza příkladů z podpůrného SDK balíku MCUXpresso
4. Přesun potřebných souborů z MCUXpresso projektu do vyvíjeného jádra (ovladače, knihovny, atd.) a oprava cest v `#include`
5. Implementace dané funkce / funkcionality
6. Oveření funkcionality na příkladech v Arduino IDE, nebo na vlastních skečích

7.2 Obecná struktura jádra

Jádro bude mít podobnou strukturu jako projekty v MCUXpresso, jelikož je založena na SDK balících z MCUXpresso. Hlavním rozdílem je, že většina zdrojových souborů s implementací funkcí Arduino se bude nacházet v kořenovém adresáři jádra kvůli snazšímu překladu. Celková struktura projektů prostředí MCUXpresso je ukázána na obrázku 4.3.



Obrázek 7.1: Struktura výsledného jádra pro mikrokontrolér LPCXpresso845-MAX. Adresář `api` obsahuje některé soubory z repositáře ArduinoCore-API, který je dostupný na adrese https://github.com/arduino/ArduinoCore-API/tree/master. Ostatní adresáře jsou stejné jako v projektech MCUXpresso (viz obrázek 4.3). „Zdrojové soubory“ reprezentují všechny zdrojové soubory v jazyce C/C++, které obsahují implementaci jádra.

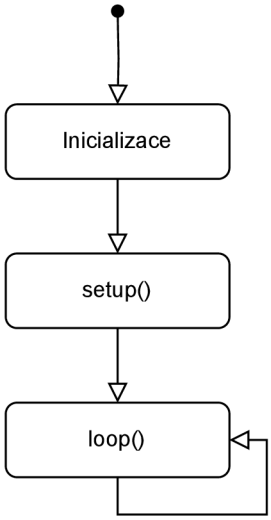
7.3 Základní funkcionalita

V této kapitole budou popsány obecné koncepty, struktury a části, které byly použity při tvorbě jádra. Také zde budou popsány soubory, ve kterých se nenachází implementace konkrétních funkcí, ale obsahují důležité součásti jádra.

7.3.1 Potřebné soubory

Důležitý soubor, který je součástí jádra je **`Arduino.h`**. Tento soubor je při předzpracování přidán k překládané skeči. Měl by tedy obsahovat veškeré nutné závislosti, které by mohly být pro skeč důležité.

V jádru byl také vytvořen soubor **`main.cpp`**, který obsahuje důležité prvotní inicializace. Dále obsahuje základní tok programu – zavolá funkci `void setup()` a dále ve smyčce volá funkci `void loop()`. Hlavní tok programu je zobrazen na obrázku 7.2.



Obrázek 7.2: Diagram hlavního toku implementovaném v souboru `main.cpp`.

Další součástí jádra je ladící knihovna, která se v procesu sestavení programu připojí k překládanému kódu. Knihovna se nachází v kořenovém adresáři podpůrného balíku. Konkretně to jsou soubory:

- `LPC845_ProjectC++_Debug.ld`
- `LPC845_ProjectC++_Debug_memory.ld`
- `LPC845_ProjectC++_Debug_library.ld`

Poslední součástí jádra je soubor `string_helper_func.h`, který obsahuje implementaci některých funkcí pro práci s řetězci. Jedná se zejména o využité funkce v jiných modulech z důvodu potíže přidáním standardních knihoven, které je implementují. Funkce v tomto souboru byly vygenerovány pomocí Copilot AI.

7.3.2 Správa pinů

Další důležitou součástí je struktura pro uchování informací o pinech. Konkrétně se jedná o pole struktur, které obsahují informace o všech pinech. Tyto informace jsou použity v ostatních částech jádra. Čísla pinů ve funkcích pak odpovídají indexu v tomto poli. Tato struktura se nachází v souboru **`arduino_pins.h`**. Navíc také obsahuje definici maker pro snazší indexaci pinů. Definovány byly názvy pinů D0 – D13, A0 – A5, LED_BUILTIN, LED_RED, LED_GREEN, LED_BLUE a SPEAKER_BUILTIN, jelikož deska obsahuje 3 led diody a i reproduktor. Implementace této struktury se nachází na obrázku 7.3.

```
typedef struct {
    uint8_t port; // GPIO port number
    uint8_t pin; // GPIO pin number
    bool canBeGPIO;  // true if pin can be GPIO
    bool configurable; // for mapping other boards if all pins are numbered
    swm_port_pin_type_t swmPinType; // used for referencing pin in SDK functions
    uint8_t ioconIndex; // used for referencing pin in SDK functions
    hal_gpio_handle_t gpioHandle; // used for GPIO external interrupts
    void (*gpioCallback)(void *); // used for GPIO external interrupts
    voidFuncPtr userFunc; // used for GPIO external interrupts
} PortPinConfig;
```

Obrázek 7.3: Struktura pro uchování informace o pinu v souboru `arduino_pins.h`. V jádru je pak definované pole těchto struktur, které obsahuje tuto strukturu pro každý konfigurovatelný pin. Funkce které pak využívají číslo pinu získají dané informace použitím čísla pinu jako index do tohoto pole. Pro použití názvu pinů byly definovány makra, které mají hodnotu odpovídajícího indexu.

7.4 Digital I/O

V této kategorii se nachází funkce pro práci s GPIO. Konkretně funkce pro vstup a výstup logických hodnot 0 a 1. Implementace těchto funkcí se nachází v souboru `wiring_digital.c`. Implementace založena na SDK příkladu `gpio_led_output`.

Nachází se zde funkce:

- `void pinMode(pin_size_t ulPin, PinMode ulMode)`
- `void digitalWrite(pin_size_t ulPin, PinStatus ulVal)`
- `PinStatus digitalRead(pin_size_t ulPin)`

void pinMode(pin_size_t ulPin, PinMode ulMode) – Nastavení pinu jako GPIO, pin je identifikovaný číslem (parametr `pin_size_t ulPin`) a podle parametru `PinMode ulMode` je buď nastaven jako vstup, nebo výstup. Navíc lze u vstupu nastavit pull-up, nebo pull-down rezistor.

void digitalWrite(pin_size_t ulPin, PinStatus ulVal) – Nastavení logické hodnoty na daný pin. Pin je identifikovaný číslem (parametr `pin_size_t ulPin`) a podle parametru `PinStatus ulVa` je na něj nastavena buď logická 1, nebo logická 0.

PinStatus digitalRead(pin_size_t ulPin) – Přečtení logické hodnoty z daného pinu. Pin je identifikovaný číslem (parametr `pin_size_t ulPin`). Funkce vrací hodnotu typu `PinStatus`, která reprezentuje buď logickou 1, nebo logickou 0.

7.5 Analog I/O

V této kategorii se nachází funkce pro práci s GPIO. Konkretně funkce pro vstup a výstup analogových hodnot. Tedy použití ADC a PWM. Implementace těchto funkcí se nachází v souboru `wiring_analog.c`. Implementace založena na SDK příkladech `lpc_adc_basic` a `sctimer_pwm_with_dutycyle_change`.

Nachází se zde funkce:

- `int analogRead(pin_size_t pinNumber)`
- `void analogReadResolution(int bits)`
- `void analogReference(uint8_t mode)`
- `void analogWrite(pin_size_t pinNumber, int value)`
- `void analogWriteResolution(int bits)`

int analogRead(pin_size_t pinNumber) – Přečtení analogové hodnoty na daném pinu. Pin je identifikovaný číslem (parametr `pin_size_t pinNumber`). Touto funkcí lze přečíst hodnoty pouze z analogových pinů A0 až A4. A5 slouží pouze jako výstup. Vrací analogovou hodnotu na daném pinu v přesnosti N bitů. Přesnost lze změnit pomocí funkce `analogReadResolution`, výchozí hodnota je 10 bitů.

void analogReadResolution(int bits) – Nastavení přesnosti analogového vstupu. Nastaví přesnost na `bits` bitů.

void analogReference(uint8_t mode) – Změna referenční hodnoty pro ADC modul na desce. V této implementaci nebylo použito.

void analogWrite(pin_size_t pinNumber, int value) – Nastavení analogové hodnoty na zadaný pin. Pin je identifikovaný číslem (parametr `pin_size_t pinNumber`). Využívá PWM, konkrétně SCTIMER modul. Analogový výstup je dostupný pouze na stejných pinech jako na deske Arduino Uno. Konkrétně jde o piny D3, D5, D6, D9, D10, D11. Přesnost lze změnit pomocí funkce `analogWriteResolution`, výchozí hodnota je 8 bitů. Parametr `int value` určuje střídu (duty cycle) na daném rozsahu (určený přesností), pokud jeho hodnota není v rozsahu, je upravena na minimální, nebo maximální korektní hodnotu.

void analogWriteResolution(int bits) – Nastavení přesnosti analogového výstupu. Nastaví přesnost na `bits` bitů.

7.6 Advanced I/O

V této kategorii se nachází funkce pro pokročilejší práci s GPIO. Využívají se zde časovače pro generování tónů, ale i pro měření délky pulzů. Implementace těchto funkcí se nachází v souboru `advanced_IO.cpp`. Implementace těchto funkcí je založena na SDK příkladech pro ctimer a na příkladu ctimer capture (dostupný na https://community.nxp.com/t5/LPC-Microcontrollers-Knowledge/LPC845-Pulse-width-Measurement-Using-CTIMER/ta-p/1254941).

Nachází se zde funkce:

- `void tone(pin_size_t pin, unsigned int frequency, unsigned long duration)`
- `void noTone(pin_size_t pin)`
- `unsigned long pulseIn(pin_size_t pin, uint8_t state, unsigned long timeout)`
- `unsigned long pulseInLong(uint8_t pin, uint8_t state, unsigned long timeout)`
- `uint8_t shiftIn(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder)`
- `void shiftOut(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t val)`

void tone(pin_size_t pin, unsigned int frequency, unsigned long duration) – Generuje na daném pinu signál se zadanou frekvenci (s 50% střídou). Pin je identifikovaný číslem (parametr `pin_size_t pinNumber`). Dokud je specifikovaná délka tónu, tón hraje jen po určitou dobu (parametr `duration`). Pokud není doba uvedena, tón hraje až do volání funkce `noTone`. Na rozdíl od referenční implementace je blokující. K fungování využívá periferii ctimer. Před opětovným voláním této funkce je nutné zavolat `noTone`.

void noTone(pin_size_t pin) – Zastaví generování signálu na daném pinu. Pin je identifikovaný číslem (parametr `pin_size_t pinNumber`). Dále de-inicializuje periferii ctimer.

unsigned long pulseIn(pin_size_t pin, uint8_t state, unsigned long timeout) – Změří délku trvání zadané logické hodnoty na daném pinu. Pin je identifikovaný číslem (parametr `pin_size_t pinNumber`). Měřená logická hodnota je zadána parametrem `state`. Parametr `timeout` odpovídá maximální možné době, po kterou se signál měří - funkce je totiž blokující. Tato funkce využívá ctimer pro zaznamenání délky signálu. Vrací naměřenou hodnotu.

unsigned long pulseInLong(uint8_t pin, uint8_t state, unsigned long timeout) – Funkce pro měření delších signálů (oproti funkci `pulseIn`). Implementace této funkce vychází z implementace ve vestavěném Arduino balíků. Funkce využívá `funkci micros()`. Vrací naměřenou hodnotu.

uint8_t shiftIn(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder) – Načtení dat z daného pinu. Pin je identifikovaný číslem (parametr `dataPin`). Generuje hodinový signál na pinu, který je identifikovaný číslem (parametr `clockPin`). Data jsou načítána s náběžnou hranou. Parametr `bitOrder` určuje pořadí bitů, ve kterém budou přijaty - hodnota LSBFIRST, nebo MSBFIRST. Implementace převzata z vestavěného balíku Arduino. Vrací načtenou hodnotu.

void shiftOut(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t val) – Zápis hodnoty na daný pinu (data odpovídají parametru `val`). Pin je identifikovaný číslem (parametr `dataPin`). Generuje hodinový signál na pinu, který je identifikovaný číslem (parametr `clockPin`). Data jsou zasílána s náběžnou hranou. Parametr `bitOrder` určuje pořadí bitů, ve kterém budou přijaty - hodnota LSBFIRST, nebo MSBFIRST. Implementace převzata z vestavěného balíku Arduino.

7.7 Time

V této kategorii se nachází funkce spojené s časem. Zejména pak funkce pro čekání. Implementace těchto funkcí se nachází v souboru `delay.c`. Všechny tyto funkce využívají systick a jeho přerušení. Systick generuje jednou za 100ms přerušení, které zaznamenává počet uběhlých milisekund a mikrosekund od startu.

Nachází se zde funkce:

- `void delay(unsigned long ms)`
- `void delayMicroseconds(unsigned int usec)`
- `unsigned long millis()`
- `unsigned long micros()`

void delay(unsigned long ms) Pozastaví program na zadaný počet milisekund (parametr `ms`). Využívá přerušení systick. Pro nižší hodnoty využívá čekání na daný počet cyklů. Jedná se o aktivní čekání.

void delayMicroseconds(unsigned int usec) Pozastaví program na zadaný počet mikrosekund (parametr `usec`). Využívá přerušení systick. Pro nižší hodnoty využívá čekání na daný počet cyklů. Jedná se o aktivní čekání.

unsigned long millis() Vrátí počet milisekund od startu mikrokontroléru. Využívá zaznamenané hodnoty milisekund (z přerušení, které generuje systick) a hodnoty registru systick pro přesnější výsledky. Po přetečení 32 bitové hodnoty zaznamenaných milisekund, dojde k resetování hodnoty.

unsigned long micros() Vrátí počet mikrosekund od startu mikrokontroléru. Využívá zaznamenané hodnoty mikrosekund (z přerušení, které generuje systick) a hodnoty registru systick pro přesnější výsledky. Po přetečení 32 bitové hodnoty zaznamenaných mikrosekund, dojde k resetování hodnoty.

7.8 Math

V této kategorii jsou matematické funkce a makra. Ty nebylo nutné implementovat, protože jsou buď dostupné ve vestavěné knihovně `math.h`, nebo v souboru `api/Common.h` a `api/Common.cpp`. Tyto soubory v adresáři `api` jsou převzaty z univerzálního Arduino API (dostupné na .https://github.com/arduino/ArduinoCore-API/tree/master).

Nachází se zde funkce a makra:

- `abs(x)` – absolutní hodnota x
- `contrain(x, a, b)` – omezení hodnoty x na interval $\langle a; b \rangle$
- `map(x, a, b, m, n)` – přepočet hodnoty x z rozsahu $\langle a; b \rangle$ na $\langle m; n \rangle$
- `max(a, b)` – maximum z čísel a, b
- `min(a, b)` – minimum z čísel a, b
- `pow(x, e)` – umocnění čísla x na expoment e
- `sq(x)` – druhá mocnina čísla x
- `sqrt(x)` – druhá odmocnina čísla x

7.9 Trigonometry

V této kategorii se nachází trigonometrické funkce. Ty nebylo nutné implementovat, protože jsou dostupné ve vestavěné knihovně `math.h`

Nachází se zde funkce:

- `cos(x)` – kosinus x
- `sin(x)` – sinus x
- `tan(x)` – tangens x

7.10 Characters

V této skupině se nacházejí funkce pro práci se znaky. Konkrétně funkce pro testování znaků. Funkce v této kategorii nebylo nutné implementovat, protože nejsou závislé na cílové platformě. Jejich implementace se nachází v převzatém souboru z vestavěného balíku Arduino – `WCharacter.h`.

Nachází se zde funkce:

- `isAlpha(c)` – Analyzujte, zda je znak c písmeno.
- `isAlphaNumeric(c)` – Analyzujte, zda je znak c alfanumerický (písmeno nebo znak).
- `isAscii(c)` – Analyzujte, zda je znak c znakem Ascii.
- `isControl(c)` – Analyzujte, zda je znak c řídicím znakem. (např.: `\n`, `\t`, atd.)
- `isDigit(c)` – Analyzujte, zda je znak c číslice.
- `isGraph(c)` – Analyzujte, zda je znak c tisknutelný ale není mezera.
- `isHexadecimalDigit(c)` – Analyzujte, zda je znak c hexadeximální číslice.
- `isLowerCase(c)` – Analyzujte, zda je znak c malé písmeno.
- `isPrintable(c)` – Analyzujte, zda je znak c tisknutelný.
- `isPunct(c)` – Analyzujte, zda je znak c interpunkcí. (např.: čárka, vykřičník, atd.)
- `isSpace(c)` – Analyzujte, zda je znak c prázdným znakem. (tj. mezera, `\f`, `\n`, `\r`, `\t`, `\v`)
- `isUpperCase(c)` – Analyzujte, zda je znak c velké písmeno.
- `isWhitespace(c)` – Analyzujte, zda je znak c mezerou. (tj. mezera a `\t`)

7.11 Random Numbers

V této skupině se nachází funkce pro generování náhodných čísel. Funkce v této kategorii nebylo nutné implementovat, protože nejsou závislé na cílové platformě. Jejich implementace se nachází v převzatém souboru z vestavěného balíku Arduino – `WMath.cpp`. Tento soubor byl ale doplněn o funkce `random()` a `srandom()`, jelikož funkce `rand()` a `srand()` z standardní knihovny vedly k zamrznutí programu.

Nachází se zde funkce:

- `void randomSeed(unsigned long seed)` – Nastavení generátoru pseudonáhodných čísel.
- `long random(long x)` – Generuje náhodné číslo intervalu $\langle 0; x \rangle$.
- `long random(long a, long b)` – – Generuje náhodné číslo intervalu $\langle a; b \rangle$.

int random(void) – Funkce pro generování pseudonáhodných čísel. Jedná se o implementaci funkce `rand()` ze standardní knihovny.

void randomSeed(unsigned long seed) – Funkce pro nastavení počáteční hodnoty pro generování pseudonáhodných čísel. Jedná se o implementaci funkce `srand()` ze standardní knihovny.

7.12 Bits and Bytes

V této skupině se nachází makra pro práci s bity a byty. Implementace těchto maker je také součástí souboru `api/Common.h`, takže je nebylo nutné implementovat. Tento soubor je převzat z univerzálního Arduino API (dostupné na https://github.com/arduino/ArduinoCore-API/tree/master).

Nachází se zde makra:

- `bit(x)` – Hodnota bitu na pozici x. (např. `bit(2) = 4`)
- `bitClear(x, n)` – V proměnné x nastaví bit na pozici n na 0.
- `bitRead(x, n)` – Je rovno hodnotě bitu na pozici n v proměnné x.
- `bitSet(x, n)` – Nastaví n-tý bit v proměnné x na 1.
- `bitWrite(x, n, b)` – V proměnné x nastaví n-tý bit na hodnotu b. ($b \in \{0, 1\}$)
- `highByte(x)` – Vrátí horních 8 bitů z čísla x. (tedy bity 8 až 16)
- `lowByte(x)` – Vrátí dolních 8 bitů z čísla x. (tedy bity 1 až 8)

7.13 External Interrupts

V této skupině se nachází funkce a makra pro nastavení externích přerušení. Konkretně pro přerušení změnou logické hodnoty na pinu. Implementace těchto funkcí se nachází v souboru `external_interrupt.cpp`. Implementace těchto funkcí byla založena na kódu, který byl vygenerován pomocí nástroje MCUXpresso – peripherals.

Nachází se zde funkce a makra:

- `void attachInterrupt(pin_size_t interruptNumber, voidFuncPtr callback, PinStatus mode)`
- `void detachInterrupt(uint8_t interruptNum)`
- `digitalPinToInterrupt(p)`

void attachInterrupt(pin_size_t interruptNumber, voidFuncPtr callback, PinStatus mode) – Funkce, která zapne přerušení odpovídající danému pinu. V této implementaci se shodují číslo přerušení (parametr `interruptNumber`) s číslem digitálních pinů. Obecně by se hodnota tohoto parametru měla získat pomocí makra `digitalPinToInterrupt(p)`. Přerušení tedy lze nastavit na digitálních pinech D0 až D13. Přerušení zavolá funkci, která je předána parametrem `callback`. Poslední parametr `mode` určuje s jakou logickou hodnotou se má přerušení vyvolat.

Parametr `mode` může nabývat hodnot:

- `LOW` – logická 0
- `HIGH` – logická 1
- `CHANGE` – vzestupná i sestupná hrana signálu
- `FALLING` – sestupná hrana
- `RISING` – vzestupná hrana

void detachInterrupt(uint8_t interruptNum) – Vypnutí přerušení na daném pinu. V této implementaci se shodují číslo přerušení (parametr `interruptNumber`) s číslem digitálních pinů. Obecně by se hodnota tohoto parametru měla získat pomocí makra `digitalPinToInterrupt(p)`.

digitalPinToInterrupt(p) – Makro, které převede číslo pinu na číslo přerušení. V této implementaci se shodují číslo přerušení (parametr `interruptNumber`) s číslem digitálních pinů.

7.14 Interrupts

V této skupině se nacházejí makra pro zapnutí a vypnutí všech přerušení mikrokontroléru. Tyto makra jsou v souboru `Arduino.h`.

Jedná se o makra:

- `interrupts()` – Zapnutí všech přerušení.
- `noInterrupts()` – Vypnutí všech přerušení.

7.15 Communication

V této kategorii se nachází objekty zabezpečující komunikaci. Nachází se zde objekty zabezpečující sériovou komunikaci, I^2C, SPI. Dále jsou zde pomocné třídy na kterých jsou komunikační objekty založeny.

Nachází se zde třídy:

- `Print`
- `Stream`
- `Serial`
- `Wire`
- `SPI`

7.15.1 Třída Print

Jedná se o abstraktní třídu zabezpečující výpis hodnot. Poskytuje metody pro výpis hodnot v různých formátech. Implementace této třídy nezávisí na konkretní platformě, proto byla její implementace převzatá z vestavěného balíku Arduino. Nachází se v souborech `Print.cpp` a `Print.h`.

Jejími hlavními metodami jsou:

- `write(data)` – Zápis dat od periférií obdržených jako odpověď, nebo zařazení dat pro odeslání do periférie. Vrací počet zapsaných bajtů.
- `print(data)` – Tisk dat v čitelném formátu. (Převádí čísla na znaky atd.)
- `println(data)` – Stejná jako funkce `print()`, navíc k datům doplní znaky `\r\n`.

Pro více detailů implementace vizte https://www.arduino.cc/reference/en/language/functions/communication/print/

7.15.2 Třída Stream

Třída reprezentující komunikační kanál. Je na ní založena většina komunikačních tříd. Zabezpečuje funkce pro obecnou komunikaci. Dědí z třídy `Print`. Implementace této třídy nezávisí na konkretní platformě, proto byla její implementace převzatá z vestavěného balíku Arduino. Nachází se v souborech `Stream.cpp` a `Stream.h`.

Jejími hlavními metodami jsou:

- `available()` – Vrátí počet bajtů, které se dají přečíst.
- `read()` – Přečte znak z kanálu do paměti kanálu.
- `flush()` – Zašle data a vyčistí paměť pro odesílaná data.
- `find()` – Čte data z kanálu dokud nenarazí na hledané znaky.
- `findUntil()` – Čte data z kanálu dokud nenarazí na hledané znaky, nebo na ukončovací řetězec.
- `peek()` – Vrací hodnotu znaku na kanálu bez toho, aniž by je přečetl.
- `readBytes()` – Přečte určitý počet bajtů / znaků.
- `readBytesUntil()` – Čte bajty / znaky dokud nenačte určitou délku, nebo dokud nenarazí na ukončovací řetězec.
- `readString()` – Načtení znaku do řetězce.
- `readStringUntil()` – Načtení znaku do řetězce. Funkce se zastaví pokud narazí na ukončovací řetězec.
- `parseInt()` – Vráti hodnotu celého čísla (`long int`) na daném kanálu.
- `parseFloat()` – Vráti hodnotu desetinného čísla (`float`) na daném kanálu.
- `setTimeout()` – Nastavení maximální hodnoty po kterou čekat na data na danám kanálu.

Pro více detailů implementace vizte https://www.arduino.cc/reference/en/language/functions/communication/stream/

7.15.3 Objekt Serial

Zabezpečuje sériovou komunikaci. Třída je pojmenovaná `HardwareSerial` a tedy její objekty se jmenují `Serial`. V této implementaci se nachází 2 objekty této třídy – `Serial` a `Serial1`. Objekt `Serial` zabezpečuje komunikaci pro ladící výpisy přes USB a `Serial1` zabezpečuje komunikaci přes piny D0 (Rx) a D1 (Tx). Tato třída nebyla implementována s úplnou podporou jako referenční implementace z vestavěného balíku Arduino. Nedědí tedy ani z třídy `Stream`, ani z třídy `Print`. Implementována byla na základě SDK příkladu z MCUXpresso – `usart_polling_example`. Implementace této třídy se nachází v souborech `Serial.cpp` a `Serial.h`.

Třída obsahuje tyto metody:

- `void begin(unsigned long baud)` – Inicializace UART komunikace. Parametr `baud` určuje rychlost komunikace.
- `void end(void)` – Ukončení UART komunikace.
- `int available(void)` – Vrací true. Funkce přidána kvůli kompatibilitě.
- `int peek(void)` – Přečte a vrátí byte. Liší se od referenčního řešení.
- `int read(void)` – Přečte a vrátí byte.
- `int availableForWrite(void)` – Vrací 4 pokud je UART dostupný pro zasílání dat. Liší se od referenčního řešení.
- `void flush(void)` – Prázdná funkce. Funkce přidána kvůli kompatibilitě, liší se od referenčního řešení.
- `write()` – Zašle data. Liší se od referenčního řešení.
- `print()` – Zašle data v čitelné podobě.
- `println()` – Zašle data v čitelné podobě, ke kterým přidá znaky `\r\n`.

7.15.4 Objekt Wire

Objekt `Wire` zabezpečuje komunikaci pomoci I^2C. Je instancí třídy `TwoWire`, která dědí z třídy `Stream`. Ve vestavěném balíku se tato třída i objekt nachází ve vestavěné knihovně. V řešení práce je ale součástí jádra kvůli jednodušší správě závislostí a kompatibility. Implementace vychází z výchozí implementace Arduino a je založena na MCUXpresso SDK příkladech – `i2c_pooling_b2b_master`, `i2c_pooling_b2b_slave`, `i2c_interrupt_b2b_transfer_slave`. Její implementace se nachází v souborech `Wire.cpp` a `Wire.h`.

Třída obsahuje tyto metody:

- `begin()` – Inicializace komunikace v režimu master. Namapování pinů.
- `begin(adresa)` – Inicializace komunikace v režimu slave. Namapování pinů.
- `setClock()` – Nastavení zdroje hodinového signálu. V této implementaci nevyužita. Funkce přidána pouze pro kompatibilitu.

- `setWireTimeout()` – Nastavení časového limitu pro komunikaci. V této implementaci nevyužita. Funkce přidána pouze pro kompatibilitu.
- `getWireTimeoutFlag()` – Vrací true, pokud byla komunikace ukončena vypršením limitu. V této implementaci nevyužita. Funkce přidána pouze pro kompatibilitu.
- `clearWireTimeoutFlag()` – Vyčistí z registu příznak, že došlo k přerušení komunikace vypršením časového limitu. V této implementaci nevyužita. Funkce přidána pouze pro kompatibilitu.
- `beginTransmission(adresa)` – Započetí komunikace. Nastaví adresu cílového zařízeni. Nic neodesílá.
- `endTransmission()` – Ukončení komunikace. Zašle adresu, data a příznak konce. Funkce má i variantu, kde lze parametrem zvolit zda se má příznak konce odeslat či ne.
- `requestFrom()` – Zažádá zařízeni na sběrnici o data. Zašle adresu na sběrnici, dále přečte určitý počet dat a zašle příznak konce.
- `write()` – Zápis dat do interní paměti pro odeslání. Pokud je mikrokontrolér v režimu slave, ihned zašle data zpět.
- `available()` – Vrátí počet volné paměti pro čtení.
- `read()` – Vrací znak z interní paměti pro čtení.
- `peek()` – Vrátí znak, který se má přečíst, ale nepřečte ho.
- `flush()` – V této implementaci nevyužita. Funkce přidána pouze pro kompatibilitu.
- `void onReceive(void (*)(int))` – Nastavení funkce, která se má zavolat po přijetí dat.
- `void onRequest(void (*)(void))` – Nastavení funkce, která se má zavolat při přijetí požadavku na data (v režimu slave).

7.15.5 Objekt SPI

Objekt zabezpečuje komunikaci pomoci SPI. Je instancí třídy SPIClass. Objekt také využívá třídu SPISettings pro nastavení parametrů komunikace. Implementace je založena na vestavěné knihovně SPI z balíku Arduino a na MCUXpresso příkladu `spi_polling_master`. Implementace se nachází v souborech `SPI.cpp` a `SPI.h`.

Třída `SPISettings` obsahuje tyto metody:

- `SPISettings(uint32_t clock, uint8_t bitOrder, uint8_t dataMode)`
- `SPISettings()` – Jako volání `SPISettings(4000000, MSBFIRST, SPI_MODE0)`

SPISettings(uint32_t clock, uint8_t bitOrder, uint8_t dataMode) – Konstruktor, parametr `clock` je v této implementaci nevyužit. Parametr `bitOrder` určuje v jakém pořadí se budou bity z bajtu odesílat (jsou zde definovaná makra LSBFIRST a MSBFIRST). Parametr `dataMode` určuje mód SPI.

Mode	Clock Polarity (CPOL)	Clock Phase (CPHA)	Output Edge	Data Capture
SPI_MODE0	0	0	Falling	Rising
SPI_MODE1	0	1	Rising	Falling
SPI_MODE2	1	0	Rising	Falling
SPI_MODE3	1	1	Falling	Rising

Tabulka 7.1: Přehled jednotlivých módu SPI. Převzato z https://docs.arduino.cc/learn/communication/spi/

Třída `SPIClass` (objekt `SPI`) obsahuje tyto metody:

- `begin()` – Inicializace SPI komunikace. Namapování pinů.
- `end()` – Ukončení SPI komunikace.
- `usingInterrupt()` – Funkce není v tomto řešení implementována. Přidána z důvodu kompatibility.
- `notUsingInterrupt()` – Funkce není v tomto řešení implementována. Přidána z důvodu kompatibility.
- `beginTransaction(SPISettings settings)` – Započetí transakce. Jelikož parametr `settings` obsahuje parametry spojení, ukončí a inicializuje komunikaci s požadovanými nastaveními.
- `endTransaction(void)` – Ukončení transakce. Funkce není v tomto řešení implementována. Přidána z důvodu kompatibility.
- `uint8_t transfer(uint8_t data)` – Odeslání a přijetí bajtu dat.
- `uint16_t transfer16(uint16_t data)` – Odeslání a přijetí 2 bajtů dat.
- `void transfer(void *buf, size_t count)` – Odeslání a přijetí dat o velikosti `count` z paměti `buf`.

Kapitola 8

Testování

V této kapitole bude popsáno, jak byly jednotlivé části jádra pro LPCXpresso845-MAX otestovány. Dále bude popsána sada testovacích skečí, na kterých lze balík ověřit. Tyto testovací skeče jsou součástí podpůrného balíku.

8.1 Testování

Testování jednotlivých funkcí z podpůrného balíku probíhalo již při implementaci jednotlivých funkcí. Pro lepší přehlednost a snazší ověření všech funkcí naráz byly ale vytvořeny testovací skeče. Na nich pak lze ověřit buď celé skupiny funkcí naráz, nebo alespoň skupiny funkcí, které spolu souvisejí.

Nejdůležitějšími funkcemi byly funkce z kategorie Time. Ty byly testovány jako jedny z prvních, jelikož je využívá většina dalších skečí.

Další důležitou částí byla sériová komunikace, která poskytuje ladící výpisy přes terminál. Ta tedy také byla testována jako první, jelikož ladících výpisů využívá většina testovacích skečí.

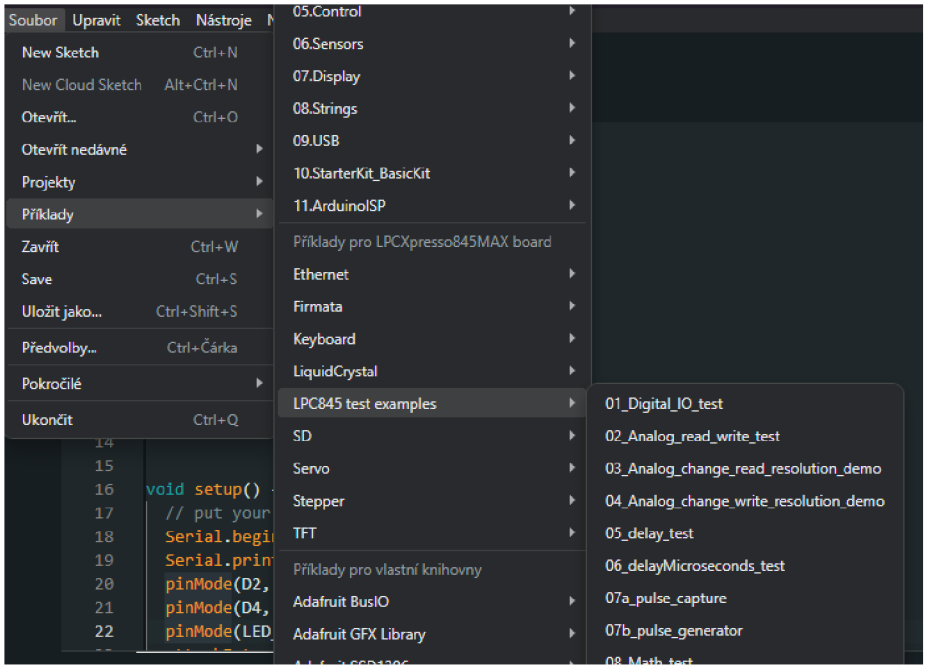
Jednotlivé skeče jsou pak popsány v kapitole 8.2.

8.2 Testovací skeče

Součástí podpůrného balíku je knihovna, která obsahuje testovací skeče. Jelikož Arduino IDE nepodporuje ukázky kódu přímo v podpůrném balíku, byla tedy vytvořena tato knihovna, která obsahuje testovací skeče (jako ukázky kódu). Jak se knihovna zobrazí v Arduino IDE je ukázáno na obrázku 8.1. Kromě těchto skečí byla funkcionalita otestována i na vestavěných příkladech v Arduino IDE. Pro každou kategorii bude tedy uvedeno na jakých skečích byla otestována.

Sériová komunikace byla otestována během vytváření celého balíku, protože poskytuje ladící výpisy, proto není uvedena v tabulce 8.1.

Funkce `delay` a `delayMicroseconds` byly otestovány ve skečích každá samostatně. Spolu s nimi byly ověřeny i funkce `millis` a `micros`, jejíž hodnoty byly použity k testování. Funkce `delay` se při testech jeví jako dostatečně přesná, avšak funkce `delayMicroseconds` podle testů přidává k zadané pauze navíc okolo 20 mikrosekund.



Obrázek 8.1: Na obrázku lze vidět jak se knihovna „LPC845 test examples“ s testovacími skečemi zobrazí v Arduino IDE.

Otestovány byly tyto kategorie, nebo funkce:

Skupina funkcí	Název skečí knihovny	Název vestavěných skečí	Komentář
Digital_IO	Digital_IO_test		
Analog_IO	Analog_read_write_test		funkce analogRead() a analogWrite()
	Analog_change_read_resolution_demo		funkce analogReadResolution()
	Analog_change_write_resolution_demo		funkce analogWriteResolution()
Time	delay_test		funkce delay() a millis()
	delayMicroseconds_test		funkce delayMicroseconds() a micros()
Advanced I/O		2.digital toneMelody	funkce tone() a noTone()
	pulse_generator a pulse_capture		test funkce pulseIn(), signál byl generován (skeč \texttt{pulse_capture}) pomocí Arduin Uno
Math	Math_test		
Trigonometry	Trigonometry_test		
Characters	Characters_test		
Random Numbers	Random_test		
Bits and Bytes	Bits_and_bytes_test		
External Interrupts	Ex_interrupt_demo		
Communication	I2C_Controller_Reader I2C_Peripheral_Sender		I2C test. Testováno s Arduino UNO jako druhým zařízením.
	I2C_Controller_Writer I2C_Peripheral_Receiver		I2C test. Testováno s Arduino UNO jako druhým zařízením.

Tabulka 8.1: Přehled testovacích skečí. U každé kategorie funkcí je popsáno na jakých skečích byla otestovaná popřípadě které funkce byly v dané skeči otestovány.

8.3 Popisy testovacích skečí

Digital_IO_test – Bliká vestavěnou LED diodou. Při stisknutí vestavěného tlačítka SW1 se blikání pozastaví v aktuálním stavu LED (buď zustane zapnutá, nebo vypnutá) dokud se tlačítko neuvolní.

Analog_read_write_test – Vyžaduje připojení LED diody a potenciometru. Přečte napětí na potenciometru, namapuje přečtenou hodnotu na hodnotu pro PWM výstup a nastaví PWM výstup pro LED diodu. Tedy nastavením potenciometru lze měnit jas diody.

Analog_change_read_resolution_demo – Podobná skeč jako `Analog_read_write_test`, ale místo mapování hodnot nastaví bitovou přesnost analogového čtení stejnou jako je přesnost pro analogový výstup (PWM). (8 bitů) Tedy nastavením potenciometru lze měnit jas diody bez nutnosti mapování hodnoty.

Analog_change_write_resolution_demo – Podobná skeč jako `Analog_read_write_test`, ale místo mapování hodnot nastaví bitovou přesnost analogového výstupu (PWM) stejnou jako je přesnost pro analogové čtení. (10 bitů) Tedy nastavením potenciometru lze měnit jas diody bez nutnosti mapování hodnoty.

delay_test – Ověřeni přesnosti funkce delay(). Využívá funkci millis() pro měření.

delayMicroseconds_test – Ověřeni přesnosti funkce delayMicroseconds(). Využívá funkci micros() pro měření.

pulse_generator a pulse_capture – Ověření měření délky pulzů. Využívá další mikrokontrolér pro generování pulzů (skeč `pulse_generator`). Mikrokontrolér, který pak měří délku pulzu (skeč `pulse_capture`) vypisuje na terminál naměřenou délku v mikrosekundách.

Math_test – Skeč pro ověření matematických funkcí. Hlavním cílem je úspěšný překlad, protože matematické funkce nemuseli být implementované. Vypisuje na terminál úspěšnost testů.

Trigonometry_test – Skeč pro ověření trigonometrických funkcí. Hlavním cílem je úspěšný překlad, protože trigonometrické funkce nemuseli být implementované. Vypisuje na terminál úspěšnost testů.

Characters_test – Skeč pro ověření funkcí pro práci se znaky. Hlavním cílem je úspěšný překlad, protože funkce pro práci se znaky nemuseli být implementované. Vypisuje na terminál úspěšnost testů.

Random_test – Skeč pro ověření funkcí pro generování náhodných čísel. Hlavním cílem je úspěšný překlad ale kontroluje, zda jsou generované hodnoty v zadaném rozsahu. Vypisuje na terminál úspěšnost testů.

Bits_and_bytes_test – Skeč pro ověření funkcí pro práci s bity a bajty. Hlavním cílem je úspěšný překlad, protože funkce pro práci s bity a bajty nemuseli být implementované. Vypisuje na terminál úspěšnost testů.

Ex_interrupt_demo – Demonstrace přerušení na pinech. Nastaví piny D2 a D4 jako vstupy (se zapnutým pulldown rezistorem) a aktivuje na nich přerušení. Pin D2 vyvolá přerušení s náběžnou hranou a pin D4 se sestupnou hranou. Přerušení změní stav LED diody na desce a vypíše na terminál zprávu. Zpráva obsahuje informaci o tom, který z pinů vyvolal přerušení a kolikrát ho už vyvolal.

I2C_Controller_Reader a I2C_Peripheral_Sender – Ukázkový skeč pro I^2C komunikaci.[1]Kontrolér periodicky čte z periferního zařízení. Tedy požádá o data a následně je vypíše na terminál (skeč `I2C_Controller_Reader`). Periferní zařízení zase při obdržení požadavku na data zašle data kontroléru (skeč `I2C_Peripheral_Sender`).

I2C_Controller_Writer a I2C_Peripheral_Receiver – Ukázkový skeč pro I^2C komunikaci.[1] Kontrolér periodicky zasílá data do periferního zařízení (skeč `I2C_Controller_Writer`). Periferní zařízení obdržená data vypíše na terminál (skeč `I2C_Peripheral_Receiver`).

[1]Převzato z https://docs.arduino.cc/learn/communication/wire/

Kapitola 9

Ověření postupů

V této kapitole bude popsáno přidání podpory pro další mikrokontrolér – LPCXpresso860-MAX. Přidání této podpory slouží jako ověření získaných postupů při tvorbě podpory prvního mikrokontroléru LPCXpresso845-MAX.

9.1 Doplnění podpůrného balíku

Jelikož mikrokontrolér LPCXpresso860-MAX využívá stejné nástroje, nebude pro něho vytvořen samostatný podpůrný balík. Ve velké většině by byl totiž identický jako podpůrný balík pro LPCXpresso845-MAX. Pro přidání podpory bude tedy doplněn balík pro LPCXpresso845-MAX.

9.1.1 Konfigurační soubory

Hlavním doplněním projde soubor `boards.txt`, který bude doplněn o definice pro mikrokontrolér LPCXpresso860-MAX. Doplněný obsah je na obrázku 9.1.

Soubor `platform.txt` nebylo potřeba nijak měnit. Recepty pro překlad a jednotlivé hodnoty byly při překládání kódů na obě desky stejné až na parametry, které jsou uvedeny v souboru `boards.txt`. Pokud by však nastal konflikt v těchto příkazech, muselo by dojít k přesunu parametrů ze souboru `platform.txt` do `boards.txt`, kde lze definovat pro každou desku argumenty zvlášť.

```
# LPC 860 MAX
# ------------------------------------
lpc_860_max.name=LPCXpresso860MAX board
lpc_860_max.vid.0=0x0D28
lpc_860_max.pid.0=0x0204
# upload
lpc_860_max.upload.tool=redlink
lpc_860_max.upload.name=LPC865
lpc_860_max.upload.tool.default=redlink
lpc_860_max.upload.protocol=redlink
lpc_860_max.upload.maximum_size=65536
lpc_860_max.upload.maximum_data_size=8192
# build
lpc_860_max.build.mcu=cortex-m0plus
lpc_860_max.build.f_cpu=60000000L
lpc_860_max.build.board=LPC860MAX
lpc_860_max.build.extra_flags=-D__NEWLIB__ -DCPU_LPC865M201JBD64 ...
lpc_860_max.build.core=lpc860
lpc_860_max.build.ld_flags=-T "{runtime.platform.path}\
        LPC865_Project_pins_Debug.ld"
```

Obrázek 9.1: Doplněný obsah konfiguračního souboru pro vývojovou desku LPCXpresso860-MAX. Hodnota `lpc_845_max.build.extra_flags` byla zkrácena pro lepší přehlednost tohoto psaného textu (zbytek hodnoty nahrazeno „...“). Obsah vychází z řešení podpory pro desku LPCXpresso860-MAX (viz obrázek 6.1)

9.1.2 Jádro

Jádro bylo pro desku LPCXpresso860-MAX vytvořeno zvlášť. Obsahem podpůrného balíku jsou tak 2 jádra. Jádro pro tuto desku bylo pojmenováno `lpc860`. Tento název byl doplněn do souboru `boards.txt` (viz obrázek 9.1).

V rámci této práce byla pro tento druhý mikrokontrolér LPCXpresso860-MAX přidaná pouze podpora funkcí z kategorií Digiatal I/O a Time (pro jednotlivé funkce viz kapitolu 7.4 a 7.7).

Testování jádra

Jelikož byla pro mikrokontrolér LPCXpresso860-MAX přidána podpora pouze funkcí z kategorie Digiatal I/O a Time, byla jejich funkčnost ověřena stejným způsobem jako u mikrokontroléru LPCXpresso845-MAX. Nebyly tedy vytvořeny samostatné testy pro tento mikrokontrolér.

Kapitola 10

Závěr

Při práci jsem se seznámil s vývojovým prostředím Arduino IDE a jeho součástmi. V rámci tohoto prostředí jsem analyzoval jak jsou navrženy projekty, které se nazývají skeče. Podstatnou částí byl i způsob, jakým probírá překlad kódu a vyřešení závislostí. Také jsem analyzoval možnosti rozšíření tohoto prostředí pro podporu jiných, než výchozích mikrokontrolérů. Analyzoval jsem strukturu rozšiřovacích balíků a obsah jednotlivých konfiguračních souborů.

Také jsem se seznámil s vývojovým prostředím MCUXpresso IDE od společnosti NXP Semiconductors. U něj jsem analyzoval strukturu a obsah projektů, což je klíčové pro návrh rozšiřovacího balíku.

Na základě těchto znalostí jsem navrhl jak zajistit podporu vývojových desek a mikrokontrolérů od společnosti NXP v prostředí Arduino IDE.

Podpůrný balík jsem implementoval pro mikrokontrolér LPCXpresso845-MAX, na kterém jsem vyzkoušel navržené postupy. Podporu tohoto mikrokontroléru jsem ověřil na sadě testovacích skečí. Tyto skeče jsem také zahrnul do podpůrného balíku ve formě knihovny. Otestována byla většina podpůrného balíku až na SPI komunikaci.

Postupy a zkušenosti, které jsem získal při vytváření podpůrného balíku pro mikrokontrolér LPCXpresso845-MAX jsem následně znovu ověřil při tvorbě balíku pro LPCXpresso860-MAX. U něj jsem ale implementoval jen omezenou podporu pro funkce Digital I/O a Time.

Dalším rozšířením této práce by mohlo být ověření podpory knihoven v Arduino IDE, případně zajištění této podpory (např. u populárních knihoven pro senzory, nebo výstupní zařízení).

Výsledek této práce bude publikován ve spolupráci se společností NXP jako open-source projekt.

Literatura

[1] Arduino. *Getting Started with Arduino IDE 2* [online]. 2023 [cit. 2023-10-27]. Dostupné z: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2.

[2] Arduino. *Language Reference* [online]. 2023 [cit. 2023-10-27]. Dostupné z: https://www.arduino.cc/reference/en/.

[3] Arduino. *Platform specification* [online]. 2023 [cit. 2024-10-1]. Dostupné z: https://arduino.github.io/arduino-cli/0.36/platform-specification/.

[4] Arduino. *Sketch build process* [online]. 2023 [cit. 2023-11-3]. Dostupné z: https://arduino.github.io/arduino-cli/0.35/sketch-build-process/.

[5] Semiconductors, N. *LPC84x User manual* [online]. 2021 [cit. 2023-4-4]. Dostupné z: https://www.nxp.com/webapp/Download?colCode=UM11029.

[6] Semiconductors, N. *FACT SHEET MCUXpresso SOFTWARE DEVELOPMENT KIT (SDK)* [online]. 2023 [cit. 2024-10-1]. Dostupné z: https://www.nxp.com/docs/en/fact-sheet/MCUXPRESSOSDKFS.pdf.

[7] Semiconductors, N. *MCUXpresso IDE User Guide* [online]. 2023 [cit. 2023-12-23]. Dostupné z: https://www.nxp.com/design/design-center/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDE.

[8] Semiconductors, N. *LPC840: 32-Bit Arm® Cortex®-M0+-Based Low-Cost MCU* [online]. 2024 [cit. 2024-11-1]. Dostupné z: https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpc800-arm-cortex-m0-plus-/lpc840-32-bit-arm-cortex-m0-plus-based-low-cost-mcu:LPC84X.

[9] Teensy. *Libraries* [online]. 2024 [cit. 2024-23-1]. Dostupné z: https://www.pjrc.com/teensy/td_libs.html.

[10] Teensy. *Teensyduino* [online]. 2024 [cit. 2024-23-1]. Dostupné z: https://www.pjrc.com/teensy/teensyduino.html.

Příloha A

Obsah přiloženého paměťového média

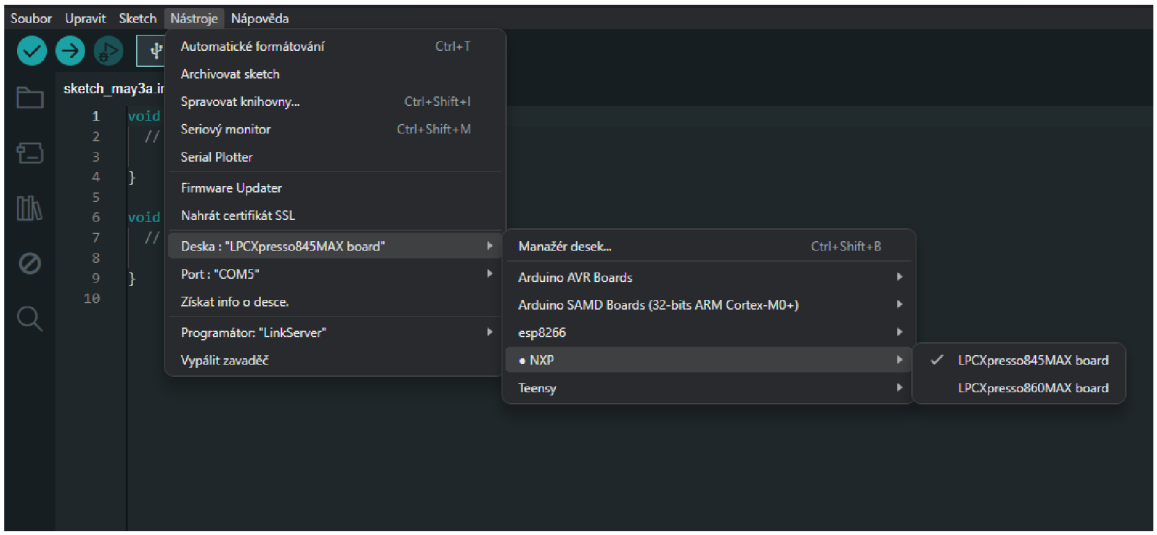
```
xtomec09.zip
├── text.pdf (tento dokument)
├── text_tisk.pdf (tento dokument - verze pro tisk)
├── text_latex (zdrojové soubory tohoto dokumentu)
└── src (zdrojové soubory)
    ├── NXP (podpůrný balík)
    ├── doc (doxygen dokumentace jader podpůrného balíku)
    └── readme.txt (návod k instalaci)
```

Příloha B

Manuál

Pro zprovoznění je nutné mít nainstalované Arduino IDE[1]. Dále je nutné zkopírovat adresář `NXP` z paměťového média (viz příloha A) do adresáře `packages` ve zdrojových souborech Arduino IDE. Při výchozí instalaci se adresář `packages` nachází na `C:\Users\user_name \AppData\Local\Arduino15\packages` , kde `user_name` je název uživatelského profilu, na kterém je Arduino IDE nainstalováno.

Následně stačí spustit Arduino IDE (pokud je již spuštěno, je nutné okno zavřít a program znovu spustit) a desky LPCXpresso845-MAX a LPCXpresso860-MAX se objeví v kontextovém menu na výběr.



Obrázek B.1: Zobrazení kontextového menu pro výběr desek po instalaci rozšiřovacího balíku.

[1]Dostupné na https://www.arduino.cc/en/software.