FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Grafické uživatelské rozhraní (GUI) pro definování funkcionality vývodů mikrokontroléru a generování kostry kódu v jazyku C
Student:	Dmitriy Tamarkov
Vedoucí:	Ing. Pavel Kubalík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

1. Prozkoumejte existující řešení.
2. Pomocí metod softwarového inženýrství navrhněte vlastní řešení vyhovující níže uvedeným požadavkům.
3. Navržené řešení naprogramujte, řádně ho zdokumentujte a otestujte.

Požadavky:
- aplikace bude napsána v jazyce Python
- uživatelské rozhraní bude napsáno v anglickém jazyce
- uživatelské rozhraní umožní nastavení funkcionality vývodů mikrokontroléru
- všechna možná nastavení přiřazení v mikrořadiči budou popsána v JSON formátu
- uživatelské rozhraní umožní čtení a zápis uživatelem zvolené konfigurace ve formátu JSON
- uživatelské rozhraní umožní vygenerovat kostru v jazyku C, která bude implementovat funkcionalitu (přiřazení funkce k vývodu obvodu), která byla nastavena uživatelem v uživatelském rozhraní

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 10. února 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Grafické uživatelské rozhraní (GUI) pro definování funkcionality vývodů mikrokontroléru a generování kostry kódu v jazyku C

Dmitriy Tamarkov

Katedra softwarového inženýrství
Vedoucí práce: Ing. Pavel Kubalík, Ph.D.

16. května 2019

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Pavlu Kubalíkovi, Ph.D. za nasměrování a pomoc při psání této bakalářské práce. Zvláštně bych chtěl poděkovat mé rodině, která mě podporovala a pomáhala během studia a při psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 16. května 2019 ……

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Tamarkov, Dmitriy. *Grafické uživatelské rozhraní (GUI)
pro definování funkcionality
vývodů mikrokontroléru
a generování kostry kódu v jazyku C.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato práce se zabývá návrhem, implementací a testováním grafického nástroje pro definování funkcionality vývodů mikrokontroléru a generování kostry kódu v jazyku C. Výsledkem dané práce je plně funkční desktopová aplikace implementovaná v jazyce Python s využitím grafického frameworku PySide2. Nástroj má jednoduché a intuitivní grafické uživatelské rozhraní a dokáže podle uživatelem zvolených nastavení vygenerovat kostru kódu v jazyku C.

Klíčová slova GUI, grafické uživatelské rozhraní, desktopová aplikace, mikrokontrolér, kostra kódu, programovací jazyk C, Python, PySide2, Qt

Abstract

This thesis covers design, implementation and testing of a graphical tool for assignment microcontroller pad functionality and generation C language skeleton code. The result of this thesis is a fully functional desktop application implemented in Python using graphical framework PySide2. The tool has a simple and intuitive graphical user interface and can generate a C code skeleton according to user's settings.

Keywords GUI, graphical user interface, desktop application, microcontroller, skeleton code, Python, programming language C, PySide2, Qt

Obsah

Seznam obrázků

Seznam tabulek

Úvod

S rozvojem průmyslu informačních technologií se neustále zvyšuje složitost softwaru. Vznikají produkty se stále rostoucím množstvím zdrojového kódu, počtem vazeb mezi komponentami atd. Vývoj těchto produktů se stává časově náročnějším a vyžaduje více zdrojů.

V IT průmyslu tak došlo k problému se zjednodušením a urychlením vývoje softwaru. V dnešní době se výše uvedený problém řeší několika způsoby. Jedním z nich je rozvoj organizace vývoje softwaru a metodik řízení projektů. Neméně důležité je ale zdokonalení nástrojů pro vývoj softwaru, což umožnuje významně snížit časové náklady a urychlit dodání produktu na trh. Mezi tyto nástroje patří editory kódu, systémy pro správu verzí, systémy pro automatizaci procesů vývoje a samozřejmě integrovaná vývojová prostředí (IDE).

V poslední době se začaly objevovat nástroje, které zjednodušují vývoj v oblasti firmware. Největší výrobci mikrokontrolérů, jako jsou *Microchip Technology*, *NXP Semiconductors* a *STMicroelectronics*, vydali grafické nástroje pro pohodlnou konfiguraci mikrokontrolérů a platforem a následné generování inicializačního kódu. Tyto nástroje poskytují konkurenční výhodu oproti těm, kteří takový software nemají. Výhodou je, že tyto programy jsou kompatibilní pouze s produkty tohoto výrobce. S tím ale nastává problém pro firmy jako Eaton, které vyrábějí své vlastní platformy. Problém nevyužitelnosti těchto řešení a neexistenci vlastního softwaru pro konfiguraci vývodu mikrokontroléru a generování kostry kódu bude řešit tato bakalářská práce.

Kapitola 1

Cíl práce

Cílem dané práce je návrh a implementace desktopové aplikací pro definování funkcionality vývodů mikrokontroléru a generování kostry kódu v jazyku C. Nástroj nabídne lehké, intuitivní a informatické uživatelské rozhraní, které umožní rychlou prací s periferií zařízení a provádění potřebných nastavení. Aplikace umožní vytváření, uložení a načtení projektů. Všechna možná nastavení přiřazení v mikrořadiči budou popsána ve formátu *JSON (JavaScript Object Notation)* a nástroj umožní čtení a zápis uživatelem zvolené konfigurace v tomto formátu. Mezi hlavní funkce aplikace patří i generování inicializačního kódu v jazyku C. Kostra kódu bude vygenerována na základě uživatelem vybrané šablony a provedených nastavení.

V rámci rešeršní části bude provedena analýza existujících řešení, nalezení a zvážení jejích výhod a nevýhod. Následně bude provedena analýza požadavků a modelování případů užití.

V praktické části práce se přistoupí k návrhu aplikace a její implementaci. Bude navržená a popsána architektura programu, zvolen způsob uložení dat a vytvořen prototyp grafického uživatelského rozhraní. Následně bude provedena implementace. Posledním krokem bude testování pro měření kvality a ověření funkčnosti aplikace.

Kapitola 2

Rešerše

V rámci bakalářské práce byly analyzovány grafické konfigurační nástroje umožňující generování kódu v jazyku C na základě nastavené funkcionality. Byly vybrány tří nástroje od známých výrobců mikrokontrolérů jako jsou STMicroelectronics, NXP Semiconductors a Microchip Technology.

2.0.1 STM32CubeMX

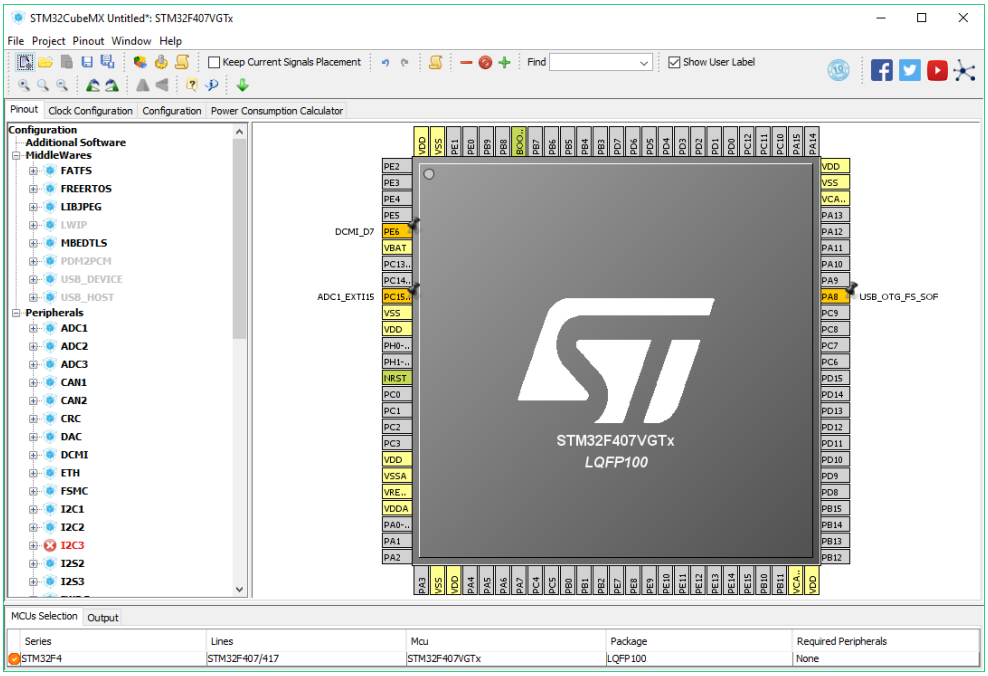
STM32CubeMX [1] je grafický nástroj společnosti STMicroelectronics, který umožňuje snadnou konfiguraci mikrokontrolérů STM32 a generování odpovídajícího inicializačního kódu v jazyku C. Je realizován v programovacím jazyku Java.

Mezi výhody patří:

- bezplatná distribuce,
- nezávislost na platformě,
- intuitivní a lehké uživatelské rozhraní,
- dobrá vizuální viditelnost obsazených pinů,
- automatické řešením konfliktů vývodů mikrokontroléra,
- je k dispozici jako samostatný software.

Mezi nevýhody patří:

- dostupnost jen po registraci,
- požaduje instalaci Java Runtime Environment (JRE),
- podporuje jenom platformy a mikrokontroléry vlastní výroby.



Obrázek 2.1: Uživatelské rozhraní STMCubeMX

2.0.2 MPLAB Code Configurator

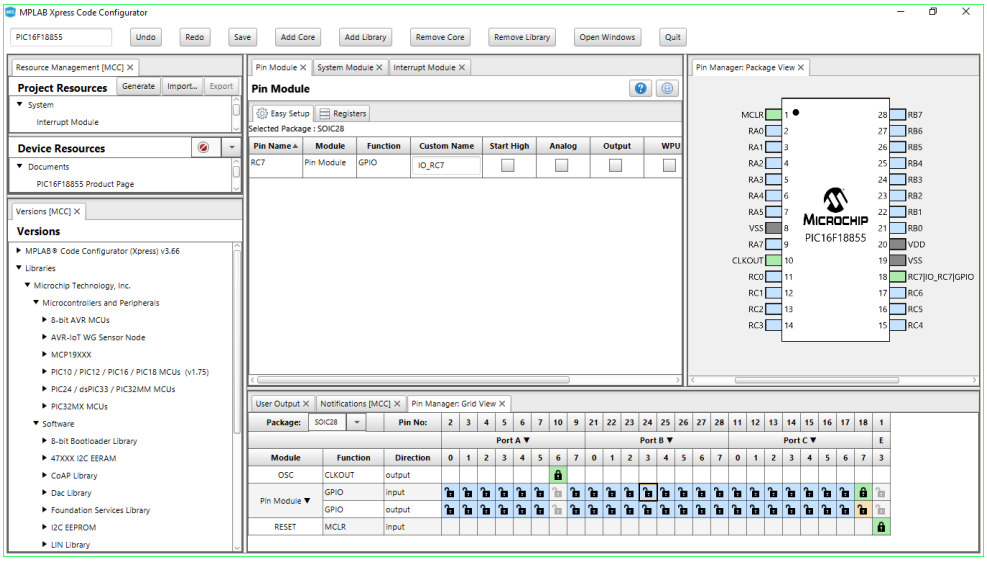
MPLAB Code Configurator [2] je grafické programovací prostředí společnosti Microchip Technology, které umožňuje vývojářům rychle a vizuálně konfigurovat periferní moduly mikrokontroléru a vytvářet konfigurační soubory v jazyku C. Tento nástroj je součástí integrovaného vývojového prostředí *MPLAB X IDE* a cloud-based *MPLAB Xpress IDE*. Podporuje 8bitové, 16bitové a 32bitové mikrokontroléry PIC (Peripheral Interface Controller).

Mezi výhody patří:

- bezplatná distribuce,
- nezávislost na platformě,
- možnost využiti ve vývojovém prostředí MPLAB Xpress IDE, které nepožaduje instalaci ani registraci,
- vizuálně viditelné obsazené a prázdné piny,
- nabízí několik pohledu pro nastavení pinů.

Mezi nevýhody patří:

- složité uživatelské rozhraní s velkým množstvím ovládacích prvků,
- požaduje instalaci JRE,
- není samostatný software,
- použitelné jenom pro platformy a mikrokontroléry společnosti Microchip Technology.



Obrázek 2.2: Uživatelské rozhraní MPLAB Code Configurator

2.0.3 MCUXpresso Config Tools

MCUXpresso Config Tools [3] je sada konfiguračních nástrojů společnosti NXP Semiconductors pro mikrokontroléry sérií Kinetis a LPC. Obsahuje mimo jiné nástroje *Pins tool* a *Peripherals tool* pro prací s piny, výběr požadovaných periferií a generací inicializačního kódu.

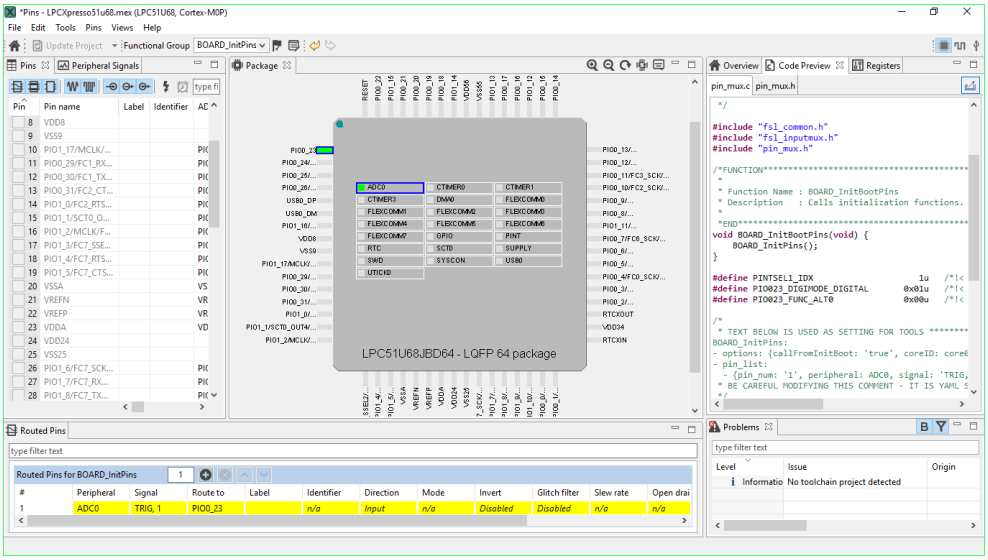
Mezi výhody patří:

- bezplatná distribuce,
- nezávislost na platformě,
- velmi informativní uživatelské rozhraní,

- vizuálně viditelné obsazené a prázdné piny,
- generace C kódu ihned po nastavení,
- možnost náhledu kódu,
- řešení I/O konfliktu,
- je k dispozici jako samostatný software.

Mezi nevýhody patří:

- složité uživatelské rozhraní s velkým množstvím nastavení,
- dostupnost jen po registraci,
- použitelné jenom pro platformy a mikrokontroléry výrobce.



Obrázek 2.3: Uživatelské rozhraní MCUXpresso

2.0.4 Shrnutí existujících řešení

Hlavní nevýhodou existujících řešení je to že jsou použitelné jenom pro platformy a mikrokontroléry výrobců těchto nástrojů. Dalšími nevýhody jsou složité a neintuitivní GUI (Graphical User Interface) a potřebnost instalaci zvláštních program jakožto JRE. Taktéž většina výrobců analyzovaných nástrojů pro stáhnutí instalačních balíčků nutí uživatele k registraci na vlastních webových stránkách.

Společnými výhodami jsou bezplatná distribuce softwaru a nezávislost na platformě. I když každý z analyzovaných nástrojů nabízí skoro stejnou funkcionalitu, *STM32CubeMX* má intuitivní a lehké uživatelské rozhraní, a proto je inspiraci pro návrh GUI aplikace, která je napsaná v rámci této bakalářské práce.

Kapitola 3

Analýza

Tato kapitola se věnuje analýze funkčních a nefunkčních požadavků, specifikací a popisu případů užití systému a také volbě technologií.

3.1 Analýza požadavků

Vývoj softwaru je komplikovaný proces, který se skládá z postupných, určitým způsobem seřazených fází. Tento proces zahrnuje nejen samotné psaní kódu, ale také sběr a analýzu požadavků, přípravu cílů, návrh architektury a testování pro potvrzení toho, že to, co je vyvíjeno, splňuje cíle. Pochopení potřeb a očekávání zákazníka od výsledného systému je jednou z klíčových podmínek produktivní spolupráce a dodání užitečného produktu splňujícího jeho požadavky. V této fázi se popisuje funkčnost softwaru, definuje se hranice a omezení systému.

3.1.1 Funkční požadavky

Funkčním požadavkem je formulace toho, co by měl systém dělat – popisuje požadovanou funkci systému [4]. U každého požadavku je uvedená jeho priorita.

F1 – Vytváření nového projektu a jeho uložení
Aplikace umožní uživateli založit nový projekt, a uložit všechny změny provedené v rámci tohoto projektu. Místo uložení a název projektu budou specifikovány uživatelem. Bude existovat možnost uložení aktuálního projektu v jiném umístění a pod jiným názvem. Při vytváření projektu bude navržen výběr obvodu a odpovídajícího pouzdra mikrokontroléru.

Priorita: vysoká

F2 – Otevření a editace existujícího projektu
Uživatel bude mít možnost otevřít minule uložený projekt a pokračovat v práci nad ním bez ztráty dat.

Priorita: vysoká

F3 – Čtení a zápis uživatelem zvolené konfigurace ve formátu JSON
Aplikace zajistí uložení uživatelem zvolené konfigurace ve formátu JSON a také její načtení.

Priorita: vysoká

F4 – Grafické zobrazení schématu vybraného mikrokontroléru
GUI aplikace zobrazí schéma uživatelem vybraného microkontroléru.

Priorita: vysoká

F5 – Automatické zobrazení změn v pinech mikrokontroleru
Všechny změny v pinech, které byly provedeny uživatelem, musí být automatické zobrazeny ve schématu.

Priorita: vysoká

F6 – Zvýraznění pinů, kterým lze vybranou funkci přiradit
Uživatel bude mít možnost vybrat položku ze seznamu funkcí a aplikace barevně zvýrazní piny, kterým lze danou funkci přiřadit.

Priorita: střední

F7 – Generování kostry kódu v jazyku C
Aplikace dokáže vyenerovat kostru kódu v jazyku C na základě vybraných šablon a nastavení, provedených uživatelem v grafickém uživatelském rozhraní.

Priorita: vysoká

F8 – Výběr šablon pro generování kódu
Systém umožní uživateli výběr šablon, na jejichž základě bude provedeno generování výsledného kódu.

Priorita: vysoká

3.1.2 Nefunkční požadavky

N1 – Aplikace v jazyku Python
Software musí být realizován v programovacím jazyku Python.

N2 – GUI
Grafické uživatelské rozhraní musí být v anglickém jazyce a mít následující vlastnosti:

- jednoduchost,
- intuitivnost,
- informativnost zobrazení dat.

N3 – Nezávislost na platformě
Aplikace musí fungovat na operačních systémech Windows a Linux.

N4 - Out-of-the-box
Uživatel musí mít možnost spustit aplikaci ihned bez nutnosti instalaci.

3.2 Případy užití

V této kapitole jsou uvedeny a popsány případy užití. Případ užití je popis interakcí mezi systémem a aktérem, který je externí pro tento systém. Případ užití popisuje všechny kroky, které musí aktér a navrhovaný systém provést pro dosazení požadovaného cíle. V podstatě se jedná o detailní specifikaci funkčních požadavků. Každý případ obsahuje krátký popis, seznam účastníků, kterým se říká aktéři, scénář a prioritu. Diagram případů užití je k dispozici v [3.1].

3.2.1 Scénáře případů užití

UC1 – Založení nového projektu
Krátký popis:

Use-case umožňuje založit nový projekt.

Aktéři:

- Uživatel,
- Systém.

Hlavní scénář:

1. Případ užití začíná, když uživatel potřebuje založit nový projekt.
2. Ve vítacím okně uživatel stiskne tlačítko *New Project*.
3. Zobrazí se okno vytvoření nového projektu, kde uživatel vyplní položku *Location*, vybere ze seznamu obvod, pouzdro a stiskne tlačítko *Create*.
4. Systém zkontroluje, jestli zadané umístění již neobsahuje projekt.
5. Pokud obsahuje, aplikace se zeptá, jestli uživatel opravdu chce založit projekt v daném umístění a tím smazat existující.

6. Uživatel potvrdí vytvoření projektu.
7. Systém zobrazí schéma mikrokontroléru a uživatel bude mít možnost začít pracovat nad projektem.

Alternativní scénář:

2.1 Uživatel má otevřený projekt a chce založit nový.
2.2 Uživatel klikne na tlačítko *File* a vybere *New Project*.

Kroky 3–7 se opakují

Priorita: vysoká

UC2 – Otevření existujícího projektu

Krátký popis:

Use-case umožňuje otevřít existující projekt.

Aktéři:

- Uživatel,
- Systém.

Hlavní scénář:

1. Případ užití začíná, když uživatel potřebuje otevřít existující projekt.
2. Ve vítacím okně uživatel stiskne tlačítko *Open Project*.
3. Zobrazí se okno výběru adresáře obsahujícího projekt, uživatel provede výběr a stiskne tlačítko *Open*.
4. Systém zkontroluje, jestli zadaná složka obsahuje projekt.
5. Pokud neobsahuje, aplikace zobrazí chybové hlášení.
6. Systém zobrazí schéma mikrokontroléru a uživatel bude moci pokračovat v práci na projektu.

Alternativní scénář:

2.1 Uživatel má otevřený projekt a chce otevřít jiný.
2.2 Uživatel klikne na tlačítko *File* a vybere *Open Project* nebo stiskne kombinaci kláves *Ctrl+O*.

Kroky 3–7 se opakují

Priorita: vysoká

UC3 – Uložení projektu

Krátký popis:

Use-case umožňuje uložit změny v projektu.

Aktéři:

- Uživatel,
- Systém.

Hlavní scénář:

1. Případ užití se začíná, když uživatel chce uložit projekt.
2. Uživatel klikne na tlačítko *File* a vybere *Save Project* nebo stiskne kombinaci kláves *Ctrl+S*.
3. Systém uloží projekt.

Priorita: vysoká

UC4 – Uložení existujícího projektu do jiného umístění a/nebo pod jiným názvem

Krátký popis:

Use-case umožňuje uložit existující projekt do jiného umístění a/nebo pod jiným názvem.

Aktéři:

- Uživatel,
- Systém.

Hlavní scénář:

1. Případ užití začíná, když uživatel chce vytvořit kopii aktuálního projektu, se kterým pracujete, s jiným názvem nebo do jiného umístění.
2. Uživatel klikne na tlačítko *File* a vybere *Save Project As* nebo použije zkratku *Ctrl+A*.
3. Zobrazí se okno, kde uživatel vybere umístění a název projektu a stiskne tlačítko *Save*.
4. Systém vytvoří adresář s názvem projektu a uloží projekt.

Priorita: vysoká

UC5 – Přirazení funkcionality pinu

Krátký popis:

Use-case umožňuje přiřadit pinu funkci.

Aktéři:

- Uživatel,
- Systém.

Hlavní scénář:

1. Případ užití začíná, když uživatel chce přiřadit pinu funkce.
2. Uživatel klikne levým tlačítkem na pinu.
3. Zobrazí se seznam funkcí pro daný pin.
4. Uživatel klikem levým tlačítkem vybere funkci ze seznamu.
5. Systém přiřadí pinu vybranou funkcí a graficky znázorní změny.

Priorita: vysoká

UC6 – Aktivace a deaktivace Pull-up rezistoru

Krátký popis:

Use-case umožňuje aktivovat a deaktivovat Pull-up rezistor na pinu.

Aktéři:

- Uživatel,
- Systém.

Hlavní scénář:

1. Případ užití začíná, když uživatel chce aktivovat nebo deaktivovat Pull-up rezistor na pinu.
2. Uživatel klikem pravým tlačítkem na pinu zavolá kontextové menu a vybere ze seznamu *Pull-up*.
3. Systém aktivuje (případně deaktivuje) Pull-up resistor pro daný pin a graficky znázorní změny.

Priorita: vysoká

UC7 – Aktivace a deaktivace Drive Strength

Krátký popis:

Use-case umožňuje aktivovat a deaktivovat Drive Strength pro daný pin.

Aktéři:

- Uživatel,
- Systém.

Hlavní scénář:

1. Případ užití začíná, když uživatel chce aktivovat/deaktivovat Drive Strength na pinu.
2. Uživatel klikem pravým tlačítkem na pinu zavolá kontextové menu a vybere ze seznamu *Drive Strength*.
3. Systém aktivuje (případně deaktivuje) Drive Strength pro daný pin a graficky znázorní změny.

Priorita: vysoká

UC8 – Nastavení názvu pinu

Krátký popis:

Use-case umožňuje nastavit název pinu.

Aktéři:

- Uživatel,
- Systém.

Hlavní scénář:

1. Případ užití začíná, když uživatel chce nastavit název pinu.
2. Uživatel kliknutím pravým tlačítkem na pinu zavolá kontextové menu a vybere ze seznamu *Enter User Device Name*.
3. Aplikace zobrazí okno pro zadání názvu. V případě prvního nastavení názvu bude nabídnut název stejný jako název přiřazené funkci.
4. Uživatel zadá vlastní nebo příjme navrhovaný název stisknutím klávesy *Enter*.
5. Systém nastaví název pro daný pin a graficky znázorní změny.

Priorita: vysoká

UC9 – Přídávání komentáře

Krátký popis:

Use-case umožňuje zadat komentář pro příslušný pin.

Aktéři:

- Uživatel,
- Systém.

Hlavní scénář:

1. Případ užití začíná, když uživatel chce okomentovat pin.
2. Uživatel kliknutím pravým tlačítkem na pinu zavolá kontextové menu a vybere ze seznamu *Enter User Comment*.
3. Aplikace zobrazí okno pro zadání komentáře.
4. Uživatel zadá komentář a stisknutím klávesy *Enter* potvrdí nastavení.
5. Systém nastaví komentář pro daný pin.

Priorita: vysoká

UC10 – Resetování stavu pinu

Krátký popis:

Use-case umožňuje resetovat nastavení, která byla provedena nad příslušným pinem.

Aktéři:

- Uživatel,
- Systém.

Hlavní scénář:

1. Případ užití začíná, když uživatel potřebuje smazat všechna nastavení pinu.
2. Uživatel kliknutím pravým tlačítkem na pinu zavolá kontextové menu a vybere ze seznamu *Reset State*.
3. Systém smaže všechna nastavení pinu a graficky znázorní změny.

Priorita: vysoká

UC11 – Výběr funkce pro nápovědu

Krátký popis:

Use-case umožňuje graficky zvýraznit piny, kterým lze přiřadit příslušnou funkci.

Aktéři:

- Uživatel,
- Systém.

Hlavní scénář:

1. Případ užití začíná, když uživatel chce dozvědět kterým pinům lze přiřadit vybranou funkcionalitu.
2. Uživatel klikne na funkci ze seznamu.
3. Systém graficky zvýrazní příslušné piny.

Priorita: střední

UC12 – Nastavení barev

Krátký popis:

Use-case umožňuje provedení nastavení barev schématu mikrořadiče.

Aktéři:

- Uživatel,
- Systém.

Hlavní scénář:

1. Případ užití začíná, když uživatel chce provést nastavení barev mikrokontroléru.
2. Uživatel klikne na tlačítko *Project* a vybere *Settings* nebo použije zkratku *Alt+S*.
3. Systém otevře okno nastavení.
4. Uživatel vybere záložku *Colors*.
5. Uživatel bude mít možnost zadat barvy pinů mikrokontroléru v hexadecimální podobě.
6. Uživatel potvrdí provedená nastavení stisknutím tlačítka *Apply*.
7. Systém zkontroluje správnost uvedených kódů barev.

8. V případě neplatností kódů, aplikace zobrazí chybové hlášení, zvýrazní políčka s chybami a povolí uživateli provést opravení.
9. Aplikace uloží nastavení a zavře okno.

Alternativní scénář:

7.1 Uživatel stiskne tlačítko „Cancel“ nebo tlačítko zavření okna.
7.2 Systém smaže uživatelem provedená nastavení a nechá dříve uložená nastavení beze změny.
7.3 Aplikace zavře okno nastavení.

Priorita: střední

UC13 – Nastavení generování kódu

Krátký popis:

Use-case umožňuje provést nastavení procesu generování výsledného kódu projektu.

Aktéři:

- Uživatel,
- Systém.

Hlavní scénář:

1. Případ užití začíná, když uživatel chce provést nastavení procesu generování výsledného kódu projektu.
2. Uživatel klikne na tlačítko *Project* a vybere *Settings* nebo použije zkratku *Alt+S*.
3. Systém otevře okno nastavení.
4. Uživatel vybere záložku *Code Generation*.
5. Uživatel bude mít možnost vybrat vlastní šablony pro generování zdrojového a hlavičkového souborů nebo vybrat použití standardních šablon.
6. Uživatel bude moct zapnout nebo vypnout automatické otevírání vygenerovaných souborů v standardním editoru kódu v jazyku C.
7. Uživatel potvrdí provedená nastavení stisknutím tlačítka *Apply*.
8. Systém zkontroluje existenci vybraných šablon.
9. V případě výběru neexistujících souborů aplikace zobrazí chybové hlášení a nabídne uživateli provést výběr znovu.
10. Aplikace uloží nastavení a zavře okno.

Alternativní scénář:

7.4 Uživatel stiskne tlačítko *Cancel* nebo tlačítko zavření okna.
7.5 Systém smaže uživatelem provedená nastavení a nechá dříve uložená nastavení beze změny.
7.6 Aplikace zavře okno nastavení.

Priorita: vysoká

UC14 – Generování kódu

Krátký popis:

Use-case umožňuje generování zdrojového a hlavičkového souboru v jazyku C na základě vybraných šablon a provedených nastavení.

Aktéři:

- Uživatel,
- Systém.

Hlavní scénář:

1. Případ užití začíná, když uživatel chce vygenerovat inicializační kód projektu.
2. Uživatel klikne na tlačítko *Project* a vybere *Generate Code* nebo použije zkratku *Ctrl+G*.
3. Systém vygeneruje zdrojový a hlavičkový soubor v jazyku C na základě vybraných šablon a provedených nastavení. Soubory budou uloženy ve vybraném repozitaře a otevřou se ve standardním editoru C kódu, pokud tato akce byla zapnuta v nastavení.

Priorita: vysoká

3.3 Volba technologií

3.3.1 Programovací jazyk

Jedním z nefunkčních požadavků zákazníka, popsaných v této kapitole, byla implementace aplikace v programovacím jazyku Python. Prvním důvodem výběru tohoto jazyka byla nezávislost výsledného programu na platformě. Aplikace napsaná v Pythonu se dokáže spustit na operačních systémech Windows, Linux a macOS. Dalším důvodem byla jednoduchost oproti jazyku C++, protože v Pythonu se vývojář nemusí starat o správu paměti. Tento problém se řeší v Javě garbage collectorem, ale její nevýhodou je nutnost instalace Java Virtual Machine (JVM) na uživatelském počítači. To je ale v rozporu s dalším nefunkčním požadavkem zákazníka, který spočívá v tom, že software musí být spustitelný ihned, bez nutnosti instalace. Z výšeuvedeného vyplývá, že Python je nejvhodnějším programovacím jazykem pro vývoj dané aplikace.

3.3.2 Grafický framework

Vybraný programovací jazyk ovlivňuje výběr frameworku pro tvorbu uživatelského rozhraní. Python má celou řadu nástrojů pro tyto účely. V rámci této práce byly porovnány nejznámější dva: *TkInter* [5] a *PyQt* [6].

TkInter je de-facto standardní GUI knihovna v Pythonu [5]. Důvodem je to, že tento balíček se dodává společně s instalací Pythona, a proto je ihned k využití. Další výhodou je jednoduchost použití TkInteru. Nevýhodou je to, že ovládací prvky (widgety) nabízené Tkinterem, mohou působit poněkud

zastarale a nedodržují zvyklosti panující na konkrétním operačním systému a desktopovém prostředí, kde je aplikace spuštěna. To například znamená, že na Linuxu se budou widgety chovat odlišně než na Microsoft Windows či na Mac OS. To vede k tomu že Tkinter se většinou používá pro jednoduché aplikace.

PyQt spojuje multiplatformový aplikační framework Qt a programovací jazyk Python. Qt je více než sada nástrojů pro tvorbu GUI. Zahrnuje abstrakce síťových soketů, vláken, regulárních výrazů, databází, plně funkčního webového prohlížeče, systému nápovědy, multimediálního frameworku a bohaté kolekce GUI widgetů [6]. Qt také zahrnuje Qt Designer [7], grafický návrhář uživatelského rozhraní. Tento nástroj umožnuje poměrně jednoduše vytvořit základní kostru GUI a zobrazení její náhledu. Výhodou je to, že PyQt je schopen generovat kód v Pythonu z Qt Designeru. Je také možné přidat do tohoto nástroje nové ovládací prvky GUI napsané v Pythonu. Další výhodou je velmi dobrá dokumentace a aktivní komunita, což značně zjednodušuje seznámení s danou technologií a umožnuje dostat pomoc od komunity. Grafické uživatelské rozhraní vytvořené v PyQt vypadá nativně. Je také vhodné zmínit licencování. Na rozdíl od samotného Qt je licencovaná pod GNU GPL v3 [8], která (stručně řečeno) vyžaduje, aby programy napsané s použitím PyQt byly šířeny pod stejnou licencí a se zdrojovým kódem. To znamená, že ten, kdo dostane kopii programu, musí mít možnost dostat odpovídající zdrojový kód a má možnost tento kód dál šířit pod stejnou licencí. To ale není moc vhodné pro daný projekt, proto výběr padl na framework *PySide2* [9], který je licencován pod GNU LGPL v3 [10]. Díky tomu lze PySide2 bez problémů použít i v uzavřených komerčních aplikacích. Z výšeuvedených důvodů byl pro vývoj grafického uživatelského rozhraní aplikací vybrán PySide2.

Pro lepší přehlednost jsou výhody a nevýhody TkInteru a PySide2 krátce uvedeny v tabulkách [3.1, 3.2].

3.3.3 Vývojové prostředí

Když se jedná o vývoj jakéhokoli softwaru, jeho kvalita a čas dodání velmi záleží na dokonalosti používaných nástrojů, zejména vývojového prostředí. Čím je prostředí kvalitnější, tím je vývoj pro vývojáře pohodlnější, a tedy i rychlejší.

Na vývoj pro jazyk Python existuje několik nejrůznějších nástrojů. Jako vývojové prostředí byl zvolen *PyCharm* [13]. Jedná se o moderní a inteligentní komerčně vyvíjený nástroj od společnosti JetBrains. Daný produkt nabízí nápovědu kódu, debagování a spoustu dalších velmi užitečných funkcionalit. Jedním z výhod je podpora zvoleného pro danou aplikaci grafického frameworku PySide2. Nelze neuvést i integraci s verzovacím systémem Git. Je k dispozici komunitní bezplatná verze, která má omezenou funkcionalitu, ale je dostačující pro vývoj daného systému.

3.3.4 Verzovací systém

Správa verzí je systém, který zaznamenává změny souboru nebo sady souborů v průběhu času, a uživatel tak může kdykoli obnovit jeho/jejich konkrétní verzi (tzv. verzování) [14]. Hlavním důvodem použití takového systému pro danou práci je zabezpečení zdrojového kódu. Verzovací systém zajistí vrácení jednotlivých souborů nebo celého projektu do předchozího stavu při ztrátě nebo vzniku neočekávaných problémů a umožní vrátit změny, které způsobily tyto problémy. Pro verzování aplikace bude použit verzovací systém Git, konkrétně repozitář hostovaný na serveru GitLab [15].

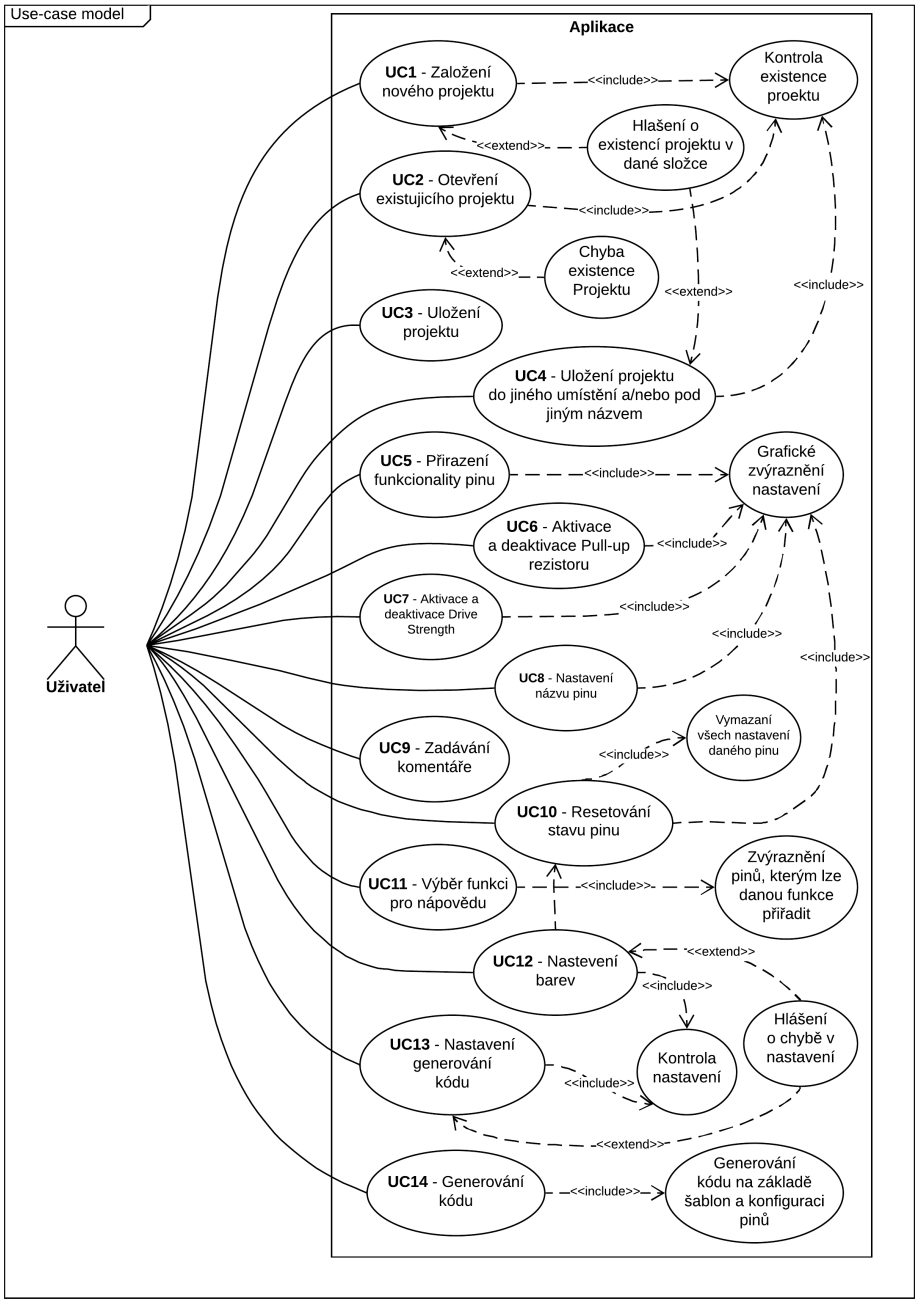
Během implementace zákazníkovi budou průběžně dodávány demo verze aplikace. Pro zpětnou vazbu bude použit nabízený GitLabem issue tracker.

Tabulka 3.1: výhody a nevýhody knihovny TkInter

Pro	Proti
Standardní knihovna (nepožaduje instalaci)	Používá se především v jednoduchých aplikacích
	Aplikace vypadá jinak v různých operačních systémech

Tabulka 3.2: výhody a nevýhody frameworku PySide2

Pro	Proti
Licencováno pod LGPL – je možné použit pro komerční projekty	Vyžaduje instalaci
Možnost vytvářet vlastní widgety	Knihovna je dost velká
Možnost načíst UI, který byl vytvořen v QtDesigneru	
Dobrá dokumentace	
Aktivní komunita	
Větší počet dostupných widgetů	
Aplikace vypadá nativně	



Obrázek 3.1: Use-case model

Kapitola 4

Návrh

Tato kapitola se věnuje návrhu řešení a popisuje jeho jednotlivé kroky.

4.1 Návrh GUI

Grafické uživatelské rozhraní je součástí téměř každé desktopové aplikace a je to první věc, kterou uživatelé uvidí a se kterou budou neustále přicházet do styku. Proto je třeba velmi pečlivě dbát na to, aby se toto GUI snadno ovládalo, aby úkony, které chce uživatel provést, daly udělat co nejefektivněji, aby bylo intuitivní, lehké a z estetického hlediska přijatelné.

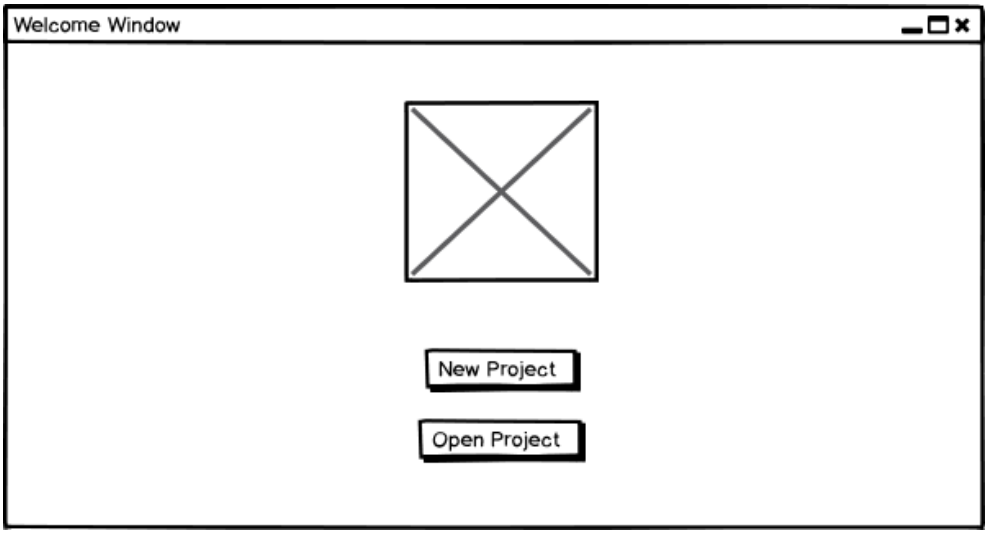
Návrh prototypu uživatelského rozhraní je velmi užitečné dělat po provedení fáze sběru požadavků. Grafický návrh obrazovky velmi usnadňuje pochopení případů užití a velmi pomáhá při komunikaci se zákazníkem [11].

Wireframes jsou standardní technologií, která se používá k návrhu uživatelského rozhraní. Jedná se o zjednodušený návrh jednotlivých obrazovek GUI aplikace. Jejich hlavním účelem není zobrazit výsledný vzhled aplikace, ale spíše rozložení funkčních komponent tak, aby odpovídaly požadavkům zadavatele. Wireframe vítacího okna (viz obrázek 4.1) obsahuje logotyp aplikace a dvě tlačítka.

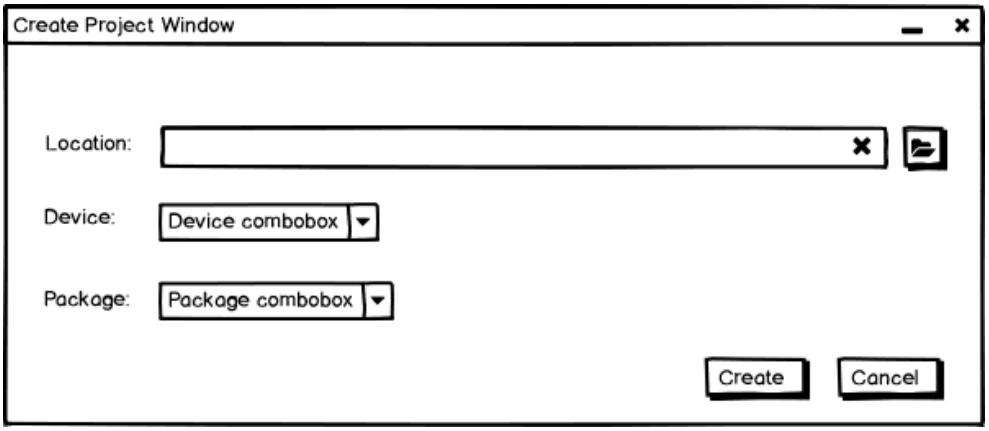
Stisknutím tlačítka *Open Project* lze otevřít existující projekt. Stisknutím tlačítka *New Project* bude otevřeno okno pro vytváření nového projektu (viz obrázek 4.2), které obsahuje pole pro zadání umístění projektu, dvě pole se seznamem dostupných obvodech a pouzdrech a dvě tlačítka.

Stisknutím tlačítka *Create* bude otevřeno hlavní okno aplikace (viz obrázek 4.3). Wireframe hlavního okna obsahuje schéma vybraného pouzdra a seznam funkcí, které podporujou piny. V horní části se nachází panel menu, přes které se uživatel bude mít možnost dostat do okna nastavení.

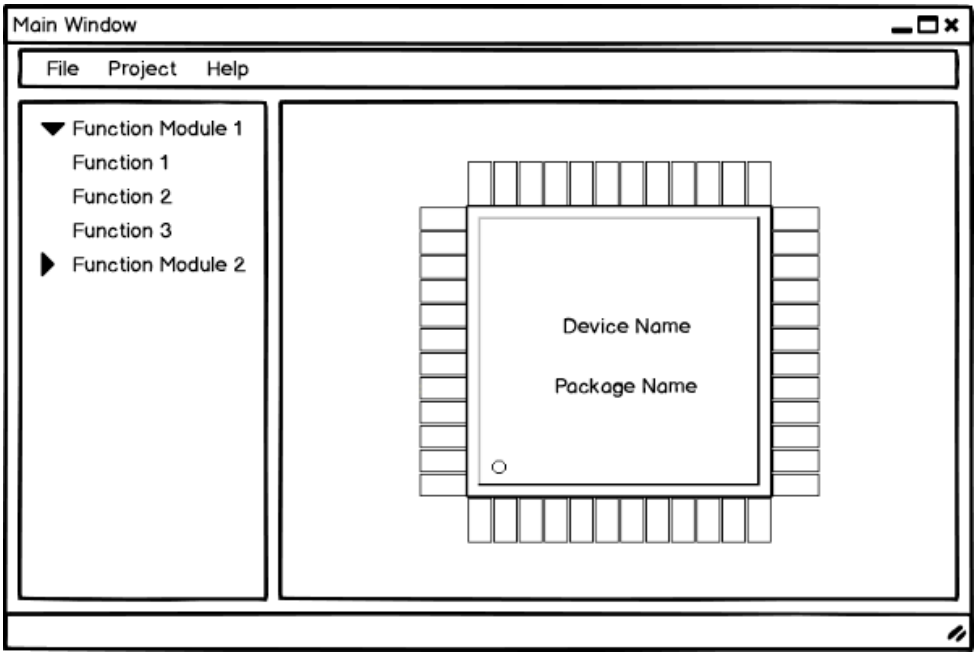
Wireframe okna nastavení se skládá ze tří záložek a dvou tlačítek pro přijmutí a smazání provedených změn. Wireframe okna s otevřenou záložkou pro nastavení generování kódu (viz obrázek 4.4) obsahuje dvě pole pro zadání umístění šablon, na jejichž základě bude provedeno generování výsledného kódu, čtyři zaškrtávací políčka, zaškrtnutím kterých budou použity standardní šablony a vytvořené soubory budou otevřeny v editoru kódu a políčko pro specifikaci místa pro uložení těchto souborů. Prototyp okna s otevřenou záložkou pro nastavení barev (viz obrázek 4.5) obsahuje políčka pro zadání barev v hexadecimálním formátu a předběžný náhled na tato nastavení.



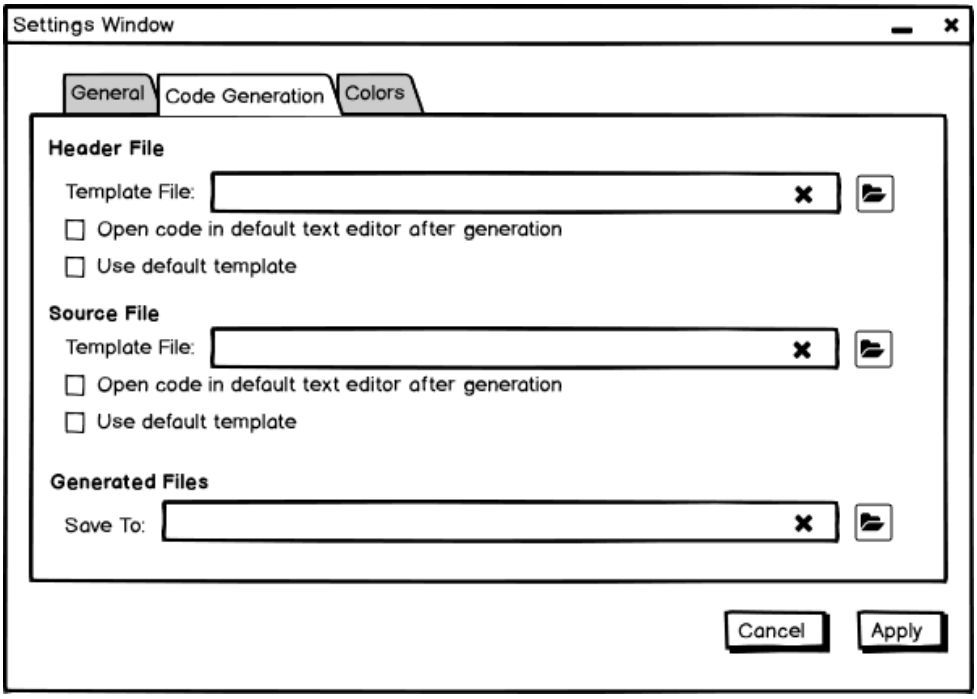
Obrázek 4.1: Wireframe vítacího okna



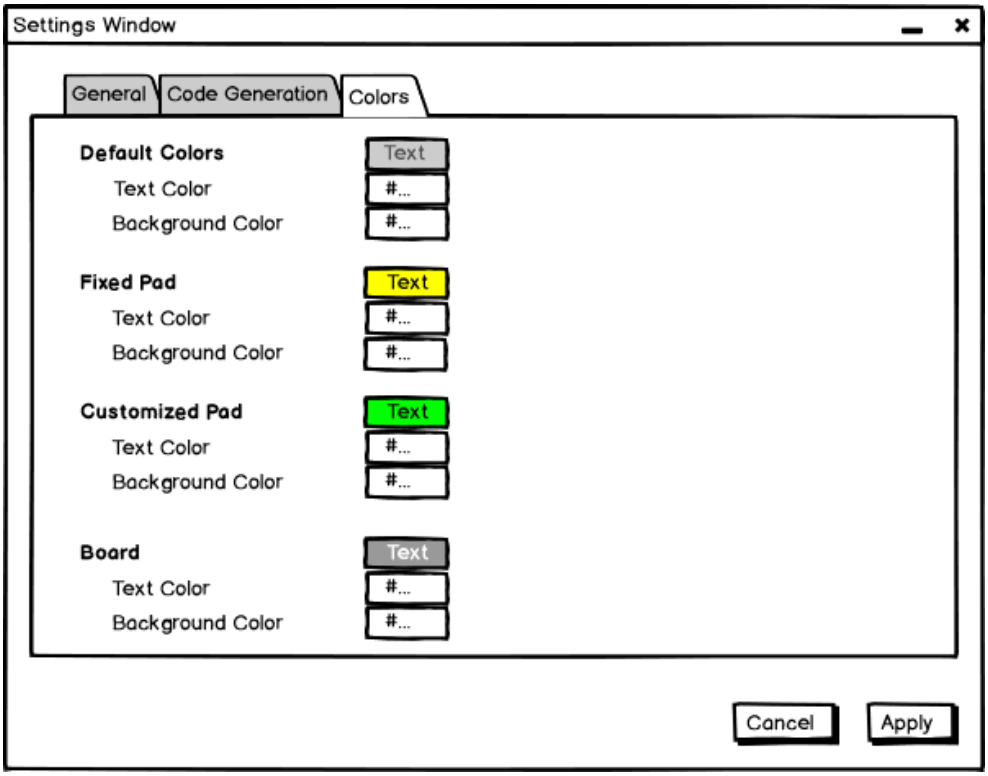
Obrázek 4.2: Wireframe okna pro vytváření nového projektu



Obrázek 4.3: Wireframe hlavního okna



Obrázek 4.4: Wireframe okna nastavení generování kódu



Obrázek 4.5: Wireframe okna nastavení barev

4.2 Návrh architektury

Architektura aplikace je v dostatečné míře ovlivněna vybranými technologiemi. V případě aplikačního rámce PySide2 je doporučovanán vzor *Model/View*, sestávající ze dvou propojených částí. Tento návrhový vzor používá termín *View* pro část softwaru, která se stará o zobrazení v uživatelském rozhraní. Pojem *Model* odpovídá části provádějící výpočty a starající se o data aplikace. Model poskytuje delegáty, pomocí kterých View poskytuje data v podobě vhodné k jejich zobrazení a upozorňuje na jejich změnu.

Hlavní výhodou tohoto architektonického vzoru je dělení kódu do menších celků, což umožňuje snadnější provádění změn a každá část aplikace se může vyvíjet odděleně. Kód je pak snadno udržovatelný, přehledný a rozšiřitelný. Tím je naplněn požadavek o snadné rozšiřitelnosti softwaru.

Při návrhu a implementaci aplikace byl dodržován tento architektonický vzor. Každý takzvaný pohled vždy načítá data z určitého modelu. Interakce s uživatelem je zorganizována pomocí ovládacích prvků uživatelského rozhraní, které komunikují s modely pomocí signálů a všechny změny dat provádí příslušný model.

Pro zobrazení navržené architektury byl vytvořen doménový model [4.6], krerý je náčrtem základních entit systému a vztahů mezi nimi. Třídy v doménovém modelu neobsahují metody a mají pouze důležité atributy. U atributů nejsou uvedeny jejich typy, jelikož doménový model je na platformově nezávislý. V [4.2.1] a [4.2.2] jsou popsány vrstvy architektury aplikace.

4.2.1 Vrstva Model

Celá vrstva (viz 4.6) je tvořena těmito třídami: `MainModel`, `SettingsModel`, `ProjectModel`, `PinModel` a `CustomizablePinModel`. Každá z těchto tříd se stará o část logiky aplikace.

`MainModel` – třída reprezentující celý backend aplikace. Slouží pro vytváření a zajištění přístupu k `SettingsModel` a `ProjectModel`. Navíc udržuje informaci o funkcích, které podporuje vybraný obvod.

`SettingsModel` – třída spravující nastavení aplikace. Udržuje stavy checkboxů, cesty k šablonám pro generování, umístění vygenerovaných souborů, hexadecimální kódy barev atd.

`ProjectModel` – třída reprezentující projekt aplikaci. Spravuje informace tykající se projektu jakožto název, místo uložení, údaje vybrané při vytváření projektu (název obvodu a pouzdra) a obsahuje instanci tříd `PinModel` a `CustomizablePinModel`. Navíc obsahuje informaci o obvodech a pouzdrech, které podporuje aplikace. Tato třída se stará i o generování výsledného kódu na základě informací projektu a vybraných šablon.

`PinModel` – třída reprezentující vývod mikrokontroléru. Spravuje informace o názvu, čísle a straně mikrokontroléru, ve které se tento vývod nachází.

`CustomizablePinModel` – třída reprezentující vývod, kterému lze přiřadit funkcionalitu, nastavit uživatelské jméno, komentář a provést další nastavení. Tato třída je rozšířením třídy PinModel.

4.2.2 Vrstva View

Z analýzy existujících řešení bylo odvozeno, že pro aplikace bude potřeba čtyř oken: vítací okno pro počáteční komunikaci s uživatelem, okno vytváření nového projektu, hlavní okno aplikace a okno s nastavením. Navíc tato vrstva obsahuje třídy reprezentující grafické vyjádření vývodů mikrokontroléru.

`MainWindow` – hlavní okno aplikace. Třída se stará o zobrazení schématu mikrokontroléru a seznamu funkcí, které podporují vývody daného obvodu. Hlavní okno zajištuje komunikaci uživatele s výšeuvedenými objekty a spravuje pomocná okna aplikace.

`WelcomeWindow` – vítací okno aplikace, které zajištuje počáteční komunikaci s uživatelem.

`CreateProjectWindow` – okno sloužící pro vytváření nového projektu. Zobrazuje informaci z `ProjectModel` o podporovaných obvodech a pouzdrech a obsahuje pole pro zadání umístění projektu.

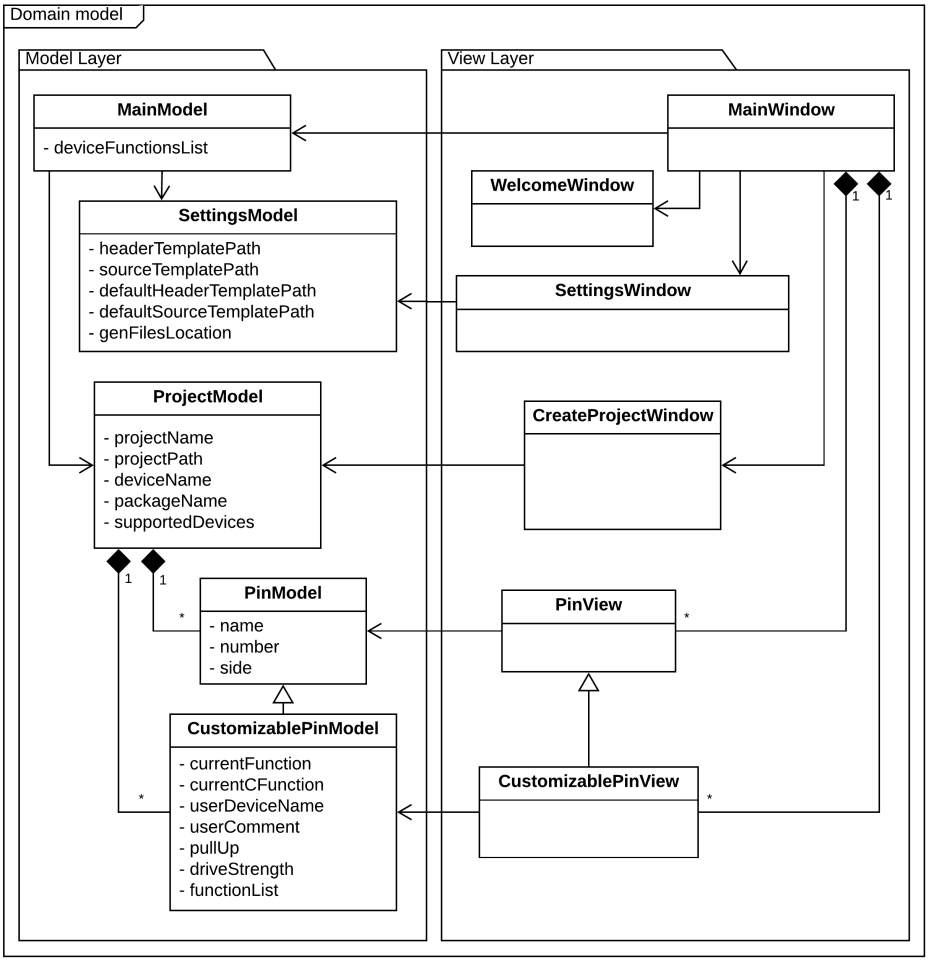
`SettingsWindow` – okno nastavení aplikace. Reprezentuje informace uložené ve třídě `SettingsModel` a slouží pro zadání jednotlivých nastavení uživatelem.

`PinView` – třída, která je grafickým znázorněním jednotlivých pinů mikrokontroléru a odpovídající informaci uložené v třídě `PinModel`.

`CustomizablePinView` – rozšíření třídy `PinView`, které reprezentuje informace uložené v třídě `CustomizablePinModel`. Jedná se o grafické vyjádření vývodů, nad kterými lze provádět nastavení. Tato třída graficky znázorňuje změny v nastavení.

4.3 Persistence dat

Jednou z nejdůležitějších částí jakéhokoliv softwaru je takzvaná persistence dat. Jedná se o způsob zachování dat mezi jednotlivými běhy aplikace. Vzhledem k tomu, že pro daný systém není potřebný přenos dat po síti, bylo rozhodnuto ukládat informace na disku. Existují dvě možnosti uložení dat na disku. První je databáze a je vhodná především při dotazování, když během použití programu je potřeba získávat jednotlivé části dat. Druhá možnost je uložení v souborech. Ten přístup je vhodný, když je nutné jednotlivé načtení všech uložených dat. Vzhledem k povaze vytvářeného softwaru, a proto, že během použití programu nebude potřeba dotazování a počet dat pro načtení není velký, byl vybrán způsob zachování informací v soubou. Kvůli malému počtu dat bude probíhat načtení dostatečně rychle. Důležité je to, že tento způsob není v rozporu s požadavkem zákazníka, který spočívá v tom, že aplikace musí zajistit uložení uživatelem zvolené konfigurace ve formátu JSON.



Obrázek 4.6: Doménový model

Kapitola 5

Realizace

V této kapitole je popsán postup implementace aplikací pro definování funkcionality vývodů mikrokontroléru a generování kostry kódu v programovacím jazyku C. Při implementaci byl použit návrh popsaný v předchozí kapitole. Tato kapitola obsahuje popis procesu tvorby oken grafického uživatelského rozhraní a implementaci grafického vyjádření schématu mikrokontroléru, vytváření konfiguračního souboru, realizaci prací s projekty a nastavení aplikace, generování kódu a balení výsledného softwaru. Na konci kapitoly jsou k dispozici ukázky výsledné aplikace.

5.1 Tvorba oken GUI

Tvorba jednotlivých oken aplikace se skládala z následujících kroků:

1. Kostra okna včetně ovládacích prvků byla navržena v *Qt Designeru*, což je nástroj pro navrhování a vytváření grafických uživatelských rozhraní (GUI) [7]. Tento program generuje XML soubor s příponou `ui`.

2. Následně z toho XML souboru byl vygenerován Python modul pomoci utility *pyuic5* [16]. Pro spuštění v konzole byl volán následující příkaz:

   ```
   python -m PyQt5.uic.pyuic -x file.ui -o file.py
   ```

3. Dalším krokem byla implementace třídy okna. Dle vzoru v [16] byla využita vícenásobná dědičnost. V konstruktoru, použitím metody `super()`, nejprve volí konstruktor třídy `QMainWindow` a voláním metody `setupUi()` do okna proběhne načtení designu. V [1] je uvedená část automaticky vygenerované třídy `Ui_MainWindow` a v [2] část konstruktoru třídy `MainWindow`.

```
class Ui_MainWindow(object):
def setupUi(self, MainWindow):
    MainWindow.setObjectName("MainWindow")
    MainWindow.resize(800, 500)
    MainWindow.setMinimumSize(QtCore.QSize(800, 500))
    icon = QtGui.QIcon()
    icon.addPixmap(QtGui.QPixmap("./inc/icons/proj_logo.png"),
                   QtGui.QIcon.Normal,
                   QtGui.QIcon.Off)
    MainWindow.setWindowIcon(icon)
    self.centralwidget = QtWidgets.QWidget(MainWindow)
    self.centralwidget.setObjectName("centralwidget")
```

Výpis kódu 1: část automaticky vygenerované třídy Ui_MainWindow

```
from src.UIs.ui_MainWindow import Ui_MainWindow
from PySide2 import QtWidgets

class MainWindow(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self, model):
        super(MainWindow, self).__init__()
        self.setupUi(self)
```

Výpis kódu 2: část konstruktoru třídy MainWindow

5.2 Schéma mikrokontroléru

Jednou z nejdůležitějších částí GUI vytvořené aplikace je grafické vyjádření schématu mikrokontroléru. Pro její implementací byly využity třídy, reprezentující grafické objekty jakožto `QGraphicsRectItem` pro zobrazení obdélníku a `QGraphicsEllipseItem` pro elipsu.

Aby grafické objekty byly zobrazeny, musí být přidány do instancí třídy `QGraphicsScene`, poskytující povrch pro správu velkého počtu 2D grafických objektů [17]. Byla navržena vlastní třída `GraphicsScene` rozšiřující funkcionalitu dané třídy. Byly implementovány metody pro vytváření schématu mikrokontroléru, seskupení grafických objektů, změnu jejích barev a další.

Následně byla rozšířena třída `QGraphicsView` poskytující widget pro zobrazení obsahu `QGraphicsScene` [18]. Byla přidána funkcionalita, umožňující přiblížení a tažení schématu mikrořadiče.

Zobrazení vývodu mikrokontroléru bylo realizováno rozšířením třídy `QGraphicsRectItem`. Bylo přidáno zobrazení informace o pinech, detekce kliknutí levé a pravé klávesy myši na daný objekt a další funkce. Výsledné zobrazení je k dispozici v hlavním okně aplikace (viz obrázek 5.1)

5.3 Konfigurační soubor

Důležitou částí práce byl návrh struktury a vytvoření konfiguračního souboru, který obsahuje popis obvodu a funkčnost pinů. Byl využit format JSON, protože s daným formátem lze snadno pracovat v Pythonu.

Navržená struktura je jednoduchá a na její základě je možné snadno vytvořit odpovídající konfigurační soubory pro jiné obvody. To je v souladu s požadavkem o rozšiřitelnosti programu, protože umožnuje zvýšit počet podporovaných obvodů. Ukázka malé části vytvořeného konfiguračního souboru je k dispozici v [3].

```
{ "ASIC2": {
    "PACKAGES": {
      "QFN48": [],
      "QFN64": []
    },
    "PIN_FUNCTIONS": {
      "p0": [
        ["GPIO_g0[0]", "GPIO"],
        ["HIAD_ncs", "HIAD"]
      ]
    },
    "FUNCTIONS": {
      "UART": [
        "OE",
        "RX"
      ],
      "HISPI": [
        "CS",
        "SCLK"
      ]
    }
  }
}
```

Výpis kódu 3: příklad konfiguračního souboru

5.4 Projekty

Během použití aplikace bude uživatel vytvářet nové a otevírat existující projekty. Projekt se skládá z jednoho souboru ve formátu JSON, který má přesně zadanou strukturu. Požadavek o vytváření nového projektu může být odeslán z vítacího nebo hlavního okna. Uživatel může zadat umístění projektu ručně v příslušném poli, která je realizována pomoci speciální třídy `QLineInput`,

nebo volbou cílového adresáře. Tato volba byla umožněna pomocí třídy `QFileDialog` [19], která implementuje samostatné dialogové okno, v němž si uživatel může pohodlně otevřít existující soubory nebo vytvořit nový soubor či adresář. Pak se v této složce vytváří soubor s názvem projektu do kterého se zapisuji informace o zvolených zařízení a pouzdře.

Při otevírání projektu nejprve pobíhá kontrola obsahu adresáře a projektového souboru. Použití třídy `QFileInfo` umožnilo velmi snadnou kontrolu existence souborů, jejích typů a přípon. Pokud kontrola proběhne bez problému bude otevřeno hlavní okno aplikace, zobrazí se projekt a uživatel bude moct pokračovat v práci nad projektem. V opačném případě se zobrazí chybové hlášení, které je instancí třídy `QMessageBox`.

Pro uložení dat projektu byla navřena speciální struktura souboru ve formatu JSON, jehož příklad je k dispozici v [4].

```
{ "Device": "ASIC2",
  "Package": "QFN48",
  "Pins": {
    "p27": {
      "Func": "HIAD_adr[6]",
      "CFunc": "HIAD",
      "UserDeviceName": "HIAD_adr",
      "UserComment": "comment",
      "PullUp": true,
      "DriveStrength": false
    }
  }
}
```

Výpis kódu 4: Ukázka uloženého projektu

5.5 Nastavení aplikace

Okno nastavení umožnuje uživateli definovat barvy schématu mikrokontroléru, vybrat šablony, na jejichž základě bude provedeno generování kódu, místo uložení a název vygenerovaných souborů (viz obrázek 5.2). Navíc byla realizována možnost automatického otevírání vygenerovaných souborů v defaultním editoru kódu v jazyce C.

V záložce nastavení barev (viz obrázek 5.3) se nachází speciální pole pro zadání hexadecimálních kódů barev a obdélníky s textem, reprezentující příslušný grafický objekt, jakožto pin nebo tělo mikrokontroléru. Tato pole jsou propojena s metodami pomocí takzvaných signálů a slotů [20]. Při změně obsahu těchto polí se posílá signál do určitého slotu (viz 5). Tento slot je metodou, která provádí kontrolu vstupu. Pokud není zadán celý kód barvy, obsah vstupního pole je obarven červeně, pro upozornění uživatele.

Po zadání celého kódu proběhne automatické znázornění změn v obdélnících, reprezentujících piny nebo tělo mikrokontroléru. Zadání nesprávných symbolů do vstupního pole znemožňuje speciální validátor, který je instancí třídy `QRegExpValidator` [21]. Pro fungování validátoru byl vytvořen regulární výraz, popisující správný hexadecimální kód barev (viz výpis kódu 6).

Persistence nastavení aplikace byla umožněna pomocí třídy `QSettings`, která umožňuje ukládat a obnovovat nastavení aplikací přenosným způsobem [12]. Použitím flagu `QSettings.IniFormat`, byla vybrána varianta uložení nastavení v speciálním inicializačním souboru, což umožnuje přenos aplikace spolu s nastavením. V [7] je uvedena ukázka uložení a získávání nastavení pomocí instancí třídy `QSetings`.

Navíc byla implementována možnost, aby při spuštění aplikace namísto vítacího okna bylo otevřeno hlavní okno s posledně otevřeným projektem a uživatel mohl ihned pokračovat v práci.

```
self.defaultTextColorLine.textChanged.connect(
    self.defaultTextColorWasChanged)
```

Výpis kódu 5: ukázka propojení signálu a metody

```
colorRegExp = QRegExp("^#([A-Fa-f0-9]{6})$")
colorValidator = QRegExpValidator(colorRegExp)
self.defaultTextColorLine.setValidator(colorValidator)
```

Výpis kódu 6: validátor hexadecimálního kódu barev

```
self._settings = QSettings(
    "./config/settings.ini", QtCore.QSettings.IniFormat)

def setLastOpenedProjectPath(self, path):
    self._settings.setValue(LAST_OPENED_PROJECT_PATH, path)
    self._settings.sync()

def getLastOpenedProjectPath(self):
    if self._settings.contains(LAST_OPENED_PROJECT_PATH):
        return self._settings.value(
            LAST_OPENED_PROJECT_PATH)
```

Výpis kódu 7: ukázka prací s nastavením aplikace

5.6 Generování kódu

Pro generování zdrojového kódu byly navřeny defaultní šablony. Jedná se o soubory v jazyku C, obsahující kostru kódu, který bude vygenerován, a klíčová slova. Tyto šablony může uživatel použít jako inspiraci pro tvorbu vlastních šablon a v okně nastavení pak má možnost si vybrat šablony na základě kterých bude provedeno generování výsledného kódu.

Klíčová slova jsou rozdělena do dvou skupin. První obsahuje slova signalizující začátek nebo konec generování kódu. Algoritmus začíná tak, že prochází šablonu řádek po řádku a hledá slovo signalizující začátek generování.

Dokud nebylo nalezeno klíčové slovo signalizující začátek generování, algoritmus kopíruje jednotlivé řádky do výsledného souboru. Po nalezení tohoto slova začíná samotné generování. Algoritmus analyzuje řádek a hledá nějaké klíčové slovo z druhé skupiny, která obsahuje slova, označují informace o pinu, jako například jeho název, funkce, komentář, Pull-up rezistor a Drive Strength. Klíčová slova nahrazuje odpovídající informací a zapisuje celý změněný řádek do výsledného souboru. Například klíčové slovo `$USER_DEVICE_NAME$` nahradí názvem, který uživatel zvolil pro příslušný vývod. Pokud bylo nalezeno nějaké klíčové slovo, pak se prochází všechny piny mikrokontroléru a pro každý z nich generuje řádek s odpovídající informací. Výjimka nastává, pokud bylo nalezeno klíčové slovo `$PAD_FUNC$`. V tomto případě budou řádky generovány jen pro ty vývody, kterým byla přiřazena funkcionalita. To se opakuje, dokud algoritmus nenarazí na slovo signalizující konec generování kódu. Slova označující začátek a konec se mohou vyskytovat v šabloně několikrát, v tomto případě výšeuvedený algoritmus se bude opakovat. Řádky neobsahující klíčová slova se kopírují do souboru. Příklad vygenerovaného souboru je k dispozici v [8]

```
#include "hardware.h"

// BEGIN_CODE_GENERATION
const device_pinout HIAD_ncs = IOMUX_PAD3_STRUCT;
const device_pinout p2 = IOMUX_PAD4_STRUCT; ///< comment
// END_CODE_GENERATION

void Init_pads(void)
{
// BEGIN_CODE_GENERATION
    io_ctrl_api_set_sel_func(HIAD, HIAD_ncs, true, true)
    io_ctrl_api_set_sel_func(p2, UART_RX, false, true)
// END_CODE_GENERATION
}
```

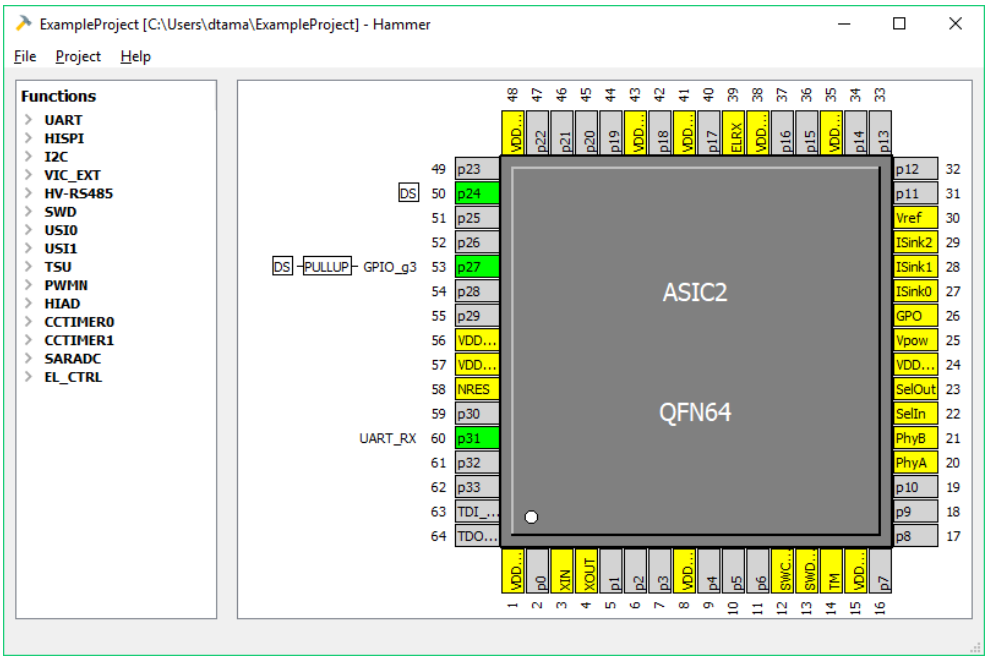
Výpis kódu 8: příklad kódu vygenerovaného souboru

Hledání klíčových slov je realizováno pomocí regulárních výrazu, pro které má Python speciální modul `re` [22]. Metoda `re.compile(pattern, flags=0)` kompiluje vzor regulárního výrazu do objektu regulárního výrazu, který lze použít pro nalezení shody pomocí `match()`, `search()` a dalších metod [22]. Nahrazení nalezených klíčových slov se provádí pomoci speciální metody `re.sub(pattern, repl, string, count=0, flags=0)`.

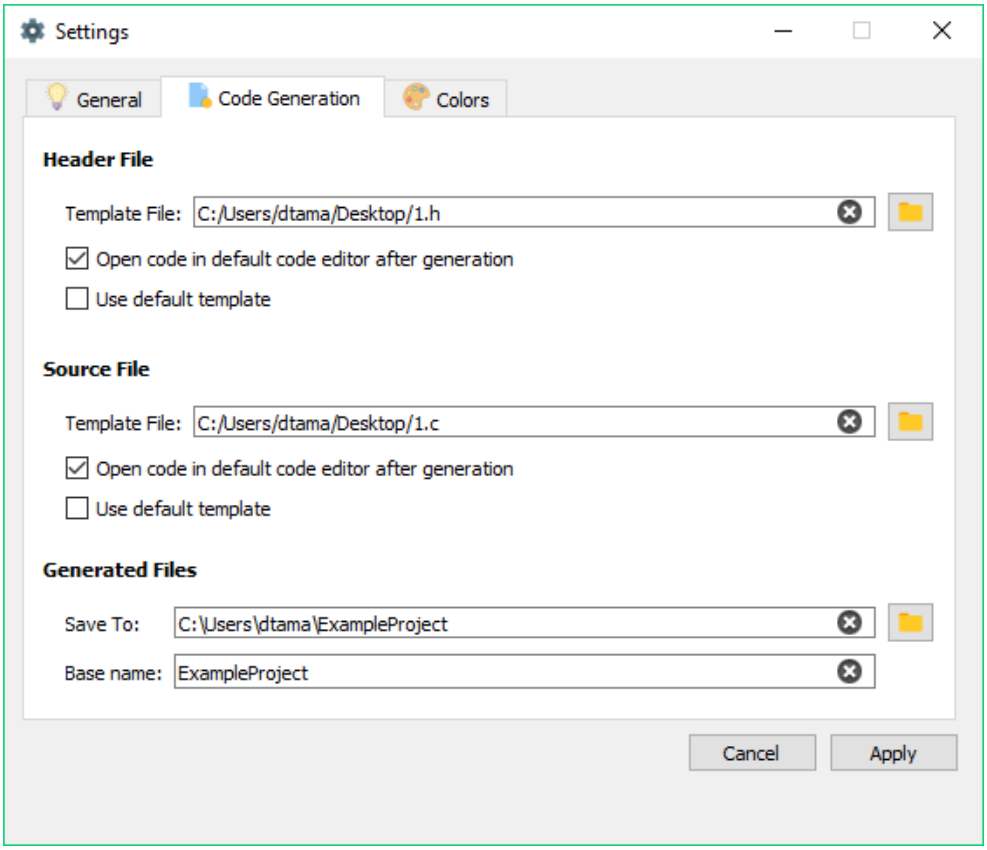
Uživatel má možnost zvolit v nastaveních aplikace automatické otevírání výsledného kódu v defaultním editoru ihned po jeho vygenerování. Daná funčknost byla realizována pomocí třídy `QDesktopServices`, která poskytuje metody přístupu ke společným službám na ploše [23].

5.7 Balení výsledné aplikace

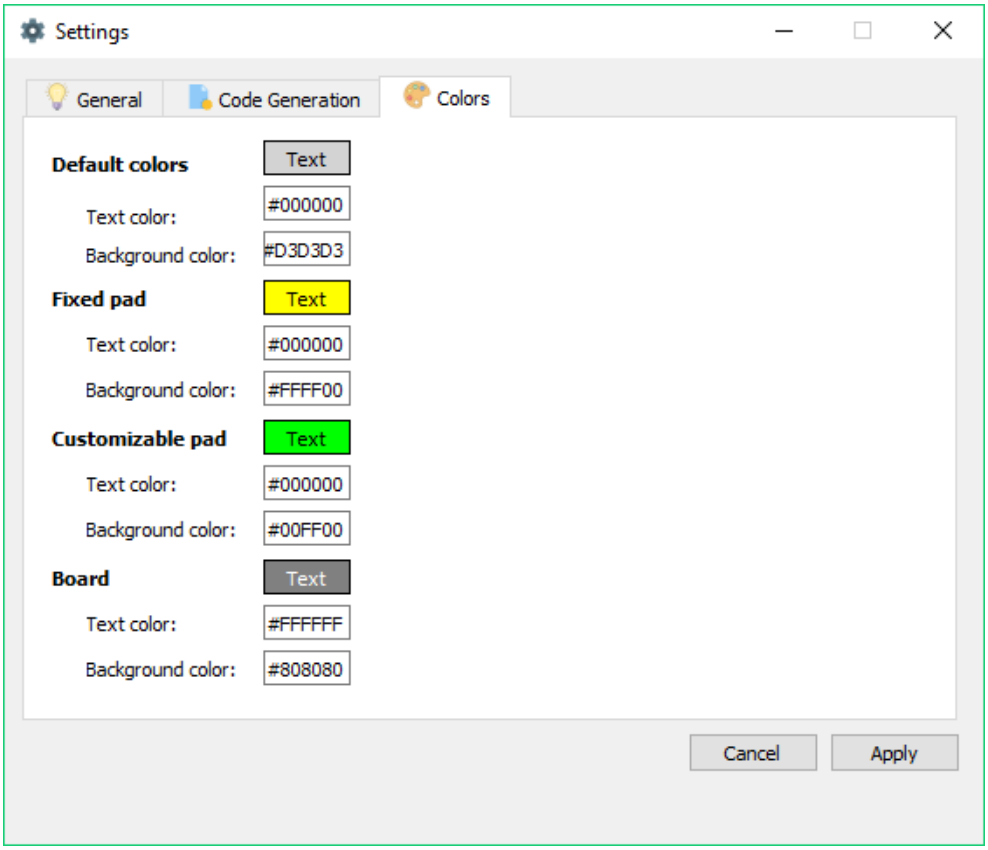
Nezbytnou částí vývoje softwaru je jeho balení. Jedná se o proces vázání relevantních souborů a komponent softwaru k vytvoření aplikace, přizpůsobené pro zákazníka. Pro tyto účely byl využit nástroj *PyInstaller* [24], který čte a analyzuje program, napsaný v Pythonu, hledá všechny ostatní moduly a knihovny, které kód potřebuje. Pak sbírá kopie všech těchto souborů, interpretu Pythonu a vloží je do skriptu v jedné složce nebo v jediném spustitelném souboru [25]. Na výstupu je out-of-box aplikace, kterou uživatel může spustit na vlastním počítači bez nutnosti instalace. Tímto byl splněn příslušný požadavek zákazníka.



Obrázek 5.1: Hlavní okno výsledné aplikace



Obrázek 5.2: Okno nastavení generování kódu



Obrázek 5.3: Okno nastavení barev

Kapitola 6

Testování

Testování je velice důležitou součástí vývoje softwaru, jejíž hlavním účelem je zjišťování, zda výsledný produkt odpovídá uživatelským požadavkům a neobsahuje závady. Pomáhá odhalit chyby a chybějící požadavky již během vývoje, ověřit správnost chování systému a ukázat jeho funkčnost.

6.1 Testování programátorem

Během vývoje aplikace v rámci této bakalářské práci bylo průběžně využiváno testování programátorem. Jedná se o postup, když ihned po vytvoření programového kódu, je tento kód prověřen programátorem. V praxi jsou tyto testy označovány jako *Assembly tests*. Program je v této časti testování kontrolován na úrovni zdrojového kódu [26]. Hlavním přínosem tohoto postupu jsou nejmenší časové náklady na opravy chyb. Nezbytnou částé byla i kontrola čitelnosti a srozumitelnosti zdrojového kódu, dodržováni stylistik a jmenných konvencí.

6.2 Testování jednotek

Po ověření kódu programátorem přichází na řadu test jednotek. U objektově orientovaného programování se jedná o testování jednotlivých tříd a metod. Testovanou jednotkou v tomto případě rozumějme samostatně testovatelnou část aplikačního programu. Testy těchto jednotek se zapisují ve formě programového kódu [26]. V rámci této části byly testovány třídy a metody zabývající se uložením, otevřením projektů a také generováním výsledného kódu. Pro testování byl použit Python framework *unittest* [12].

Pro kontrolu uložení projektu byla testována metoda `saveProject()`. Bylo provedeno nastavení v aplikaci a ručně vytvořen JSON soubor odpovídající tomuto nastavení. Tento soubor byl porovnán se souborem, který byl vytvořen použitím metody `saveProject()`.

Podobný postup byl využit i pro testování metody `openProject()`, které byl předán jako parametr JSON soubor s nastavením a pak proběhla kontrola obsahu třídy `ProjectModel` a jednotlivých pinů jako například název obvodu a pouzdra mikrořadiče, uživatelských názvů jednotlivých pinů, přiřazená funkce atd.

Posledním krokem této části testování byla kontrola vygenerovaného souboru. Bylo provedeno nastavení funkcionality pinů a ručně vytvořen soubor v jazyku C, odpovídající tomuto nastavení a vybrané šabloně. Obsah tohoto souboru byl následně porovnán s kódem, který byl vygenerován voláním metody `generateCode()`.

6.3 Systémové testy

Dalším použitým typem testů jsou takzvané systémové testy, které testují aplikace z pohledu zákazníka. Tyto testy slouží pro ověření softwaru jako funkčního celku a jako výstupní kontrola, před předáním aplikace zákazníkovi [26]. Byla prováděna simulace použití systému v reálném životě podle scénářů případů užití.

Následně bylo provedeno testování nezávislosti na platformě. Aplikace dokázala funkčnost a kvalitu uživatelského rozhraní na operačních systémech Windows a Linux, konkrétně byly použity Ubuntu 16.04 a Windows 10. Jediným nalezeným bugem byl rozdíl v umístění zobrazení některých komponent pinů. Tato chyba spadá do skupiny nekritických a byla velmi rychle a jednoduše opravena.

Po provedení a splnění těchto testů byla aplikace předáná zákazníkovi.

6.4 User acceptance testing

Posledním krokem testování softwaru byly akceptační testy. Jedná se o testy prováděné na straně zákazníka [26]. V rámci této fáze zákazníkem bylo provedeno testování výsledné aplikace v reálném použití. Výsledky testování ukázali, že aplikace je plně funkční, splňuje požadavky a byla přijata zákazníkem.

Kapitola 7

Budoucí práce

V budoucnu by bylo možné aplikaci rozšířit o další podporované obvody a pouzdra. Struktura konfiguračního souboru, která byla navržena v této práci, umožnuje snadné vytváření odpovídajících souborů. Bylo by dobré umožnit uživateli nahrávat vlastní konfigurační soubory přes grafické rozhraní.

Dalším rozšířením by mohla být realizace automatického řešení konfliktů při přiřazení funkcionality vývodům mikrořadiče.

Zákazník má také zájem o přidání možnosti nastavení hodinových stromů v mikrokontroléru s následným generováním zdrojového kódu.

Závěr

Cílem této práce bylo navrhnout a poté implementovat desktopovou aplikaci pro definování funkcionality vývodů mikrokontroléru, která zajistí automatické generování zdrojového kódu v jazyku C na základě nastavení a předem vytvořených šablon.

V rešeršní části této práce byla provedena analýza současného stavu řešení problému, nalezeny a zváženy výhody a nevýhody existujících řešení. Následně byla provedena analýza požadavků na software, modelování případů užití a byly zvoleny technologie pro implementaci. Praktická část se věnovala návrhu aplikace a její implementaci. Byla navržena a popsána architektura programu, způsob uložení dat a vytvořen prototyp grafického uživatelského rozhraní. Následně byla provedena implementace a testování pro měření kvality a ověření funkčnosti aplikace.

Výsledkem dané práce je plné funkční desktopová aplikace implementována v jazyce Python s využitím grafického frameworku PySide2. Výsledný software splňuje všechny funkční i nefunkční požadavky, které byly na tuto aplikaci kladeny a to znamená, že cíl práce byl splněn. Nástroj má jednoduché a intuitivní grafické uživatelské rozhraní, které zajišťuje rychlou práci s periferii zařízení a provádění potřebných nastavení. Na základě těchto nastavení a šablon umí vygenerovat zdrojový kód v jazyku C. Aplikace umožnuje vytváření, zachování stavu a načtení již uložených projektů. Všechna možná nastavení přiřazení v mikrořadiči jsou popsána v JSON formátu a nástroj umožňuje čtení a zápis uživatelem zvolené konfigurace v tomto formátu.

Literatura

[1] STMicroelectronics. *STM32Cube initialization code generator* [online]. [cit. 2019-05-16]. Dostupné z: https://www.st.com/en/development-tools/stm32cubemx.html

[2] Microchip Technology Inc. *MPLAB Code Configurator* [online]. [cit. 2019-05-16]. Dostupné z: https://www.microchip.com/mplab/mplab-code-configurator

[3] NXP Semiconductors. *MCUXpresso Config Tools - Pins, Clocks, Peripherals* [online]. [cit. 2019-05-16]. Dostupné z: https://www.nxp.com/support/developer-resources/software-development-tools/mcuxpresso-software-and-tools/mcuxpresso-config-tools-pins-clocks-peripherals:MCUXpresso-Config-Tools

[4] ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky.* 2., aktualiz.a dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

[5] *TkInter* [online]. [cit. 2019-05-16]. Dostupné z: https://wiki.python.org/moin/TkInter

[6] *PyQt* [online]. [cit. 2019-05-16]. Dostupné z: https://riverbankcomputing.com/software/pyqt/intro

[7] Qt Designer Manual. *Qt Documentation* [online]. [cit. 2019-05-16]. Dostupné z: https://doc.qt.io/qt-5/qtdesigner-manual.html

[8] *GNU General Public License* [online]. [cit. 2019-05-16]. Dostupné z: https://www.gnu.org/licenses/gpl-3.0.en.html

[9] PySide2. *Python bindings for the Qt cross-platform application and UI framework* [software]. [cit. 2019-05-16]. Dostupné z: https://pypi.org/project/PySide2/

[10] *GNU Lesser General Public License* [online]. [cit. 2019-05-16]. Dostupné z: https://www.gnu.org/licenses/lgpl-3.0.en.html

[11] Mlejnek, J.: *Analýza a sběr požadavků. Technická zpráva, Českévysoké učení technické v Praze, Fakulta informačních technologií, Katedra softwarového inženýrství, Softwarové inženýrství BI-SI1*, 2019, [online prezentace]. [cit. 2019-05-16]. Dostupné z: https://moodle.fit.cvut.cz/pluginfile.php/88280/mod_resource/content/3/03.prednaska.pdf

[12] QSettings Class. *Qt Documentation* [online]. [cit. 2019-05-16]. Dostupné z: https://doc.qt.io/qt-5/qsettings.html

[13] *PyCharm* [software]. [cit. 2019-05-16]. Dostupné z: https://www.jetbrains.com/pycharm/

[14] *Pro Git: [everything you need to know about the Git distibuted sourcecontrol tool]* [online].2nd ed. New York, NY: Apress, 2014, 1.1 Úvod- Správa verzí [cit. 2019-05-16]. ISBN 978-1484200773. Dostupné z: https://git-scm.com/book/en/v2

[15] *GitLab Enterprise Edition* [online]. [cit. 2019-05-16]. Dostupné z: https://www.gitlab.com/help/

[16] Using Qt Designer. *PyQt v5.12 Reference Guide* [online]. [cit. 2019-05-16]. Dostupné z: https://www.riverbankcomputing.com/static/Docs/PyQt5/designer.html

[17] QGraphicSscene Class. *Qt Documentation* [online]. [cit. 2019-05-16]. Dostupné z: https://doc.qt.io/qt-5/qgraphicsscene.html

[18] QFraphicsView Class. *Qt Documentation* [online]. [cit. 2019-05-16]. Dostupné z: https://doc.qt.io/qt-5/qgraphicsview.html

[19] QFileDialog Class. *Qt Documentation* [online]. [cit. 2019-05-16]. Dostupné z: https://doc.qt.io/qt-5/qfiledialog.html

[20] Signals & Slots. *Qt Documentation* [online]. [cit. 2019-05-16]. Dostupné z: https://doc.qt.io/qt-5/signalsandslots.html

[21] QRegExpValidator Class. *Qt Documentation* [online]. [cit. 2019-05-16]. Dostupné z: https://doc.qt.io/archives/qt-4.8/qregexpvalidator.html

[22] re — Regular expression operations. *Documentation. The Python Standard Library. Text Processing Services* [online]. [cit. 2019-05-16]. Dostupné z: https://docs.python.org/3/library/re.html

[23] QDesktopServices Class. *Qt Documentation* [online]. [cit. 2019-05-16]. Dostupné z: https://doc.qt.io/archives/qt-4.8/qdesktopservices.html

[24] *PyInstaller* [software]. [cit. 2019-05-16]. Dostupné z: https://www.pyinstaller.org/

[25] *PyInstaller* [online]. [cit. 2019-05-16]. Dostupné z: https://github.com/pyinstaller/pyinstaller

[26] Fáze a úrovně provádění testů. *Testování softwaru* [online]. [cit. 2019-05-16]. Dostupné z :http://testovanisoftwaru.cz/metodika-testovani/druhy-typy-a-kategorie-testu/faze-testu/

Příloha A

Seznam použitých zkratek

GUI Graphical User Interface

UI User Interface

JSON JavaScript Object Notation

JVM Java Virtual Machine

JRE Java Runtime Environment

PIC Peripheral Interface Controller

XML eXtensible Markup Language

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF