VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

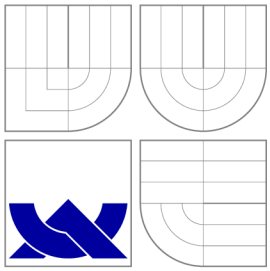
GRAFICKÉ ROZHRANÍ PRO KONFIGURACI PERIFERNÍCH MODULŮ MCU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

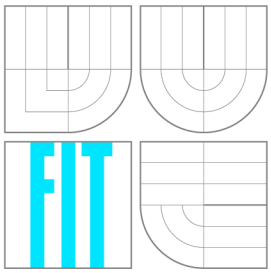
AUTOR PRÁCE
AUTHOR

MARTIN PETR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

GRAFICKÉ ROZHRANÍ PRO KONFIGURACI PERIFERNÍCH MODULŮ MCU

GUI FOR CONFIGURATION OF MCU PERIPHERAL MODULES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN PETR
AUTHOR

VEDOUCÍ PRÁCE — Ing. VÁCLAV ŠIMEK
SUPERVISOR

BRNO 2011

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2011/2012

Zadání bakalářské práce

Řešitel: **Petr Martin**
Obor: Informační technologie
Téma: **Grafické rozhraní pro konfiguraci periferních modulů MCU**
GUI for Configuration of MCU Peripheral Modules
Kategorie: Uživatelská rozhraní

Pokyny:

1. Prostudujte dokumentaci popisující architekturu a periferní moduly konkrétního mikrokontroleru.
2. Podrobně se seznamte s možnostmi konfigurace jednotlivých periferií.
3. Seznamte se s knihovnami pro tvorbu grafických uživatelských rozhraní, např. wxWidgets, GTK+ nebo Swing.
4. Připravte koncepci aplikace s GUI, jejímž úkolem bude generování konfiguračních souborů pro periferní moduly MCU určeného typu. Návrh bude proveden v podobě UML diagramů, případně jinou vhodnou formou.
5. Proveďte implementaci navržené aplikace. Generovaný výstup bude mít podobu *.h nebo *.c souborů v jazyce C.
6. Ověřte funkčnost vaší aplikace na jednoduchém příkladu. Navrhněte možnosti dalšího vylepšení.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 4 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Šimek Václav, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2011
Datum odevzdání: 16. května 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Bakalářská práce popisuje návrh, implementaci a testování aplikace, která zprostředkovává generování konfiguračních souborů mikrokontroléru, pomocí grafického uživatelského rozhraní. Výstupem této aplikace jsou soubory s funkcemi, které lze přidat do projektu s daným mikrokontrolérem.

Abstract

Bachelor thesis describes the design, implementation and testing application that provides generating configuration files for microcontroller, using the graphical user interface. The output of this application are files with functions that can be added to the project for specific microcontroller.

Klíčová slova

aplikace, konfigurace, nastavení, mikrokontrolér, MCU, generování, C++, Qt

Keywords

application, configuration, settings, microcontroller, MCU, generating, C++, Qt

Citace

Martin Petr: Grafické rozhraní pro konfiguraci
periferních modulů MCU, bakalářská práce, Brno, FIT VUT v Brně, 2011

Grafické rozhraní pro konfiguraci periferních modulů MCU

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Václava Šimka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Petr
15. května 2012

Poděkování

Nejprve bych chtěl poděkovat svému vedoucím bakalářské práce Ing. Vaclavu Šimkovi, za vedení ke zdárnému konci této práce. Neméně bych chtěl také poděkovat svým rodičům, za podporu a zázemí, které mi věnovali, nejen při psaní této práce, ale po celou dobu studia.

Obsah

Kapitola 1

Úvod

Cílem této práce je návrh a implementace aplikace, pomocí které bude možno jednoduše a přehledně nastavit periferie mikrokontrolérů. K samotnému nastavení mikrokontroléru bude sloužit zdrojový soubor, který tato aplikace vygeneruje. Tato technická zpráva, která popisuje, jak byla tato aplikace navržena a posléze implementována se dělí na sedm kapitol včetně tohoto úvodu. Podrobnosti o specifikacích jednotlivých částí aplikace a jejich analýze se píše v kapitole 2. V kapitole 3 se popisují technologie, které se Při vývoji využívaly a pomocí kterých aplikace pracuje. Kapitola 4 se zabývá nejen návrhem samotné aplikace, ale také návrhem struktury souborů, které jsou touto aplikací využívány. Zajímavé části implementace jednotlivých částí se rozebírají v kapitole 5. O návrhu formy testování, jeho průběhu a výsledcích se lze dočíst v kapitole 6. Zhodnocení práce, plány na vylepšení a další vývoj budou rozebrány v poslední části, kterou je kapitola 7.

Kapitola 2

Analýza a specifikace

Nejprve bude potřeba se zabývat požadavky jaké jsou kladeny na aplikaci, jak ze strany zadání, tak ze strany analýzy podobných aplikací a jejich funkcí. Dále bude třeba analyzovat také samotný mikrokontrolér, jeho strukturu a požadavky související s nastavováním periferií. na této analýze bude záležet vzhled a chování budoucí aplikace.

2.1 Analýza a specifikace požadavku

Byl zadán požadavek na vytvoření aplikace, s jejíž pomocí by se snadno a přehledně konfigurovaly periferie daného mikrokontroléru (dále jen MCU). Kritéria této aplikace byla tedy kladena na grafické rozhraní, dále na její multiplatformnost a neméně také na samotnou implementaci nastavení MCU. Jako vzor bude sloužit, plugin do CodeWarrior IDE, Processor Expert, který je ovšem přes veškeré své výhody pevně spjat s CodeWarrior a produkty firmy Freescale (více o prostředí CodeWarrior [4]).

Bude tedy nutno vytvořit strukturu jednoho či více souborů uchovávající popis daného MCU se vším co je pro nastavení periferií potřeba a navrhnout jejich hierarchii. Dále vytvořit aplikaci, která tyto soubory načte a informace v souborech obsažené zobrazí v uživateli přívětivé formě. Bude také potřeba zajistit kontrolu uživateli zadaných hodnot a patřičně jej upozornit na chybové hodnoty. Nakonec, jako výsledek, vygenerovat zdrojové soubory, které bude moct uživatel přilinkovat ke svému projektu. Tyto zdrojové soubory budou obsahovat jak funkce pro inicializaci daných periferií, tak i základní funkce pro obsluhu dané periferie. Aplikaci bude možné provozovat pod operačními systémy Linux a Microsoft Windows XP/7.

2.2 Analýza a specifikace grafického rozhraní

Grafické rozhraní by mělo obsahovat v prvé řadě komponentu pomocí které se lze přehledně procházet periferiemi, které dané MCU nabízí. Tato komponenta by také měla opticky oddělit nesouvisící periferie a související sdružit do skupiny. Dále by zmíněná komponenta měla dovolovat editovat hodnoty, kterými se daná periferie nastavuje. do grafického rozhraní programu je třeba zařadit i komponentu, která bude zprostředkovávat náhled na MCU, jeho vývody, tvar pouzdra a další informace, které usnadní náhled na nastavovanou periferii respektive MCU. pro výstup informací o chybách, jak programových, tak chybách nastavení MCU, bude třeba vytvořit komponentu, která tyto informace přehledně zobrazí, podobně bude třeba řešit i nápovědu k nastavovaným periferiím a hodnotám. pro usnadnění

práce uživatele bude do rozhraní zakomponováno i pole pro vyhledávání a lišta s tlačítky pro důležité akce programu.

2.3 Analýza a specifikace MCU

Pro implementaci byl zvolen MCU MC9S08JM60 od firmy Freescale, pro dostupnost jeho materiálů a hlavičkových souborů. pro implementaci daného MCU bude potřeba vytvořit soubory, které budou popisovat paměť, pouzdro, rozložení pinů, periferie a možnost jejich nastavení.

Pro popsání paměti MCU, vzhledem k potřebám aplikace, bude třeba popsat ty registry, které jsou potřeba pro nastavení periferií. Registry tohoto MCU jsou 8-bitové, bude třeba zajistit vkládání 16-bitových hodnot, aby uživatel toto nemusel řešit (například nastavení hodnoty modulo u TPM [5]). V registrech se také vyskytují skupiny bitů, které se budou nastavovat hodnotou nebo výběrem ze seznamu hodnot. je nutné popsat i bity a ukládat jejich výchozí hodnoty.

Při popisu rozdělení pinů je stěžejní tvar pouzdra, počet pinů, jejich názvy a čísla. Jelikož více periferií sdílí piny, bude potřeba uživatele upozornit pokud bude chtít generovat nastavení pro periferie obsazující stejné piny.

U periferií bude třeba popsat jednotlivé hodnoty a to tak, aby bylo pro uživatele snadné je číst a nastavit správně pro požadovanou aplikaci, k tomu mu bude také dopomáhat nápověda ke každé hodnotě a periferii. U některých hodnot periferií bude třeba kontrolovat, jestli má dané nastavení smysl (například nastavení pull-up u pinů portu, které jsou nastaveny jako výstupní), nebo kontrolovat případy kdy se nastavením nějaké hodnoty periferie omezí hodnota jiná (například nastavená 10-bitová adresa u IIC které nemá povolenu rozšířenou adresu [5]).

Kapitola 3

Použité technologie

Jelikož bylo pro vývoj aplikace zvoleno prostředí Qt, díky jeho multiplatformnosti, což byl jeden z požadavků na aplikaci, byly i podle tohoto voleny ostatní technologie, protože jsou multiplatformní a Qt je podporuje.

3.1 Qt framework

Jelikož byl požadavek na multiplatformnost aplikace zvolil jsem pro její vývoj Qt, což je framework pro vytváření aplikací s grafickým rozhraním, které je možno překládat napříč platformami [15]. Qt vyvinuli Haarand Nord a Eirik Chambe-eng [8] a dnes jej spravuje společnost Nokia. pro kódování aplikace byla použita verze 4.7.0, která byla nejnovější verzí v repozitáři. V současné době je nejnovější verze 4.8.1. Současný vývoj tohoto frameworku se upírá ke Qt verzi 5 [9].

3.2 C++

Programovací jazyk jsem zvolil jazyk C++, pro jeho rozšířenost a vhodnost. Jazyk C++ vychází z jazyka C a nabízí prostředky objektově orientované programování [18], byl vyvinut Bjarne Stroustrupem [6] v Bellových laboratořích. Nejnovější verze tohoto jazyka je uvedena pod názvem C++11 [19], v práci používám verzi C++98, která je podporovaná v Qt.

3.3 XML

Pro zápis a čtení dat aplikací jsem použil jazyk XML (Extensible Markup Language), který nabízí přehlednou syntaxi pro čtení i vytváření dokumentů [17] a také pro jeho podporu v Qt. Jazyk XML je značkovací jazyk, který definuje pravidla pro čtení dokumentů, jeho tvůrcem je skupina XML Working Group sformována pod organizací W3C [2], která jej i nadále spravuje. Ve své práci používám verzi 1.0 v revizi 5 vydané roku 2008, protože nejnovější verze 1.1 [3] nepřináší pro potřeby projektu žádné výrazné přednosti.

3.4 XSD

K popisu struktury a omezení XML dokumentů používám XSD (XML Schema Dokument), tato technologie popisuje pomocí XML schémata strukturu jiného XML dokumentu. Takto

lze v XML dokumentu definovat jména značek, parametry značek, definovat unikátní značky a typ obsahu značky [20]. XSD byl vyvinut organizací W3C která jej nadále spravuje. V mé práci je struktura XML dokumentů definována XSD ve verzi 1.0 která byla aktuální v době vzniku projektu. Dnes už je přístupné XSD ve verzi 1.1 [21].

3.5 HTML

Na popis obsahu, který se má zobrazit v komponentách podporující Rich Text, Qt používá podmnožinu značek jazyka HTML (Hyper Text Markup Language), který je značkovacím jazykem pro webové stránky. Jazyk vyvinul Tim Berners-Lee a nadále jej udržují organizace W3C, WHATWG a IETF [11]. Qt podporuje podmnožinu značek HTML verze 4 [16] a proto také tuto verzi používám, nejnovější verzí je verze 4.01 a pracuje se na verzi HTML 5, která je v této době označena jako pracovní verze.

3.6 CSS

K popisu stylu obsahu, který se má zobrazit v komponentách podporující Rich Text, podporuje Qt podmnožinu jazyka CSS (Cascading Style Sheet). CSS je jazykem rozšiřujícím možnosti popisu vzhledu HTML elementů [12]. Jazyk vyvinul Håkon Wium Lie [10] a dnes je spravován organizací W3C. Qt podporuje podmnožinu tohoto jazyka vycházející z verze 2.1 [16] a to je důvodem proč tuto verzi ve své práci používám. Vývoj tohoto jazyka se nyní ubírá k verzi 3, která je nyní stále ve vývoji [1].

Kapitola 4

Návrh řešení

Návrh implementace pro tuto aplikaci bude rozdělen do částí, které se zabývají hlavními bloky aplikace a také soubory které slouží k popisu MCU.

4.1 Návrh implementace grafického rozhraní

Grafické rozhraní se navrhovalo postupně tak, že se nejprve navrhlo rozdělení sekcí a jejich umístění v hlavním okně, následně se provedl výběr a popis komponent a dialogů , které se budou na grafickém rozhraní podílet. Nakonec se uskutečnil návrh v samotném prostředí Qt.

4.1.1 Umístění částí grafického rozhraní

Grafické rozhraní hlavního okna aplikace bude rozděleno na tři části tak, jak je vidět na obrázku 4.1. Část, ve které se bude nacházet panel s tlačítky, která bude umístěna standardně v horní části okna a bude přes jeho celou šířku. na toto jsou uživatelé zvyklí z drtivé většiny aplikací. Část, kde se budou nastavovat periferie, bude umístěna v levé části okna a zaujme zbývající výšku okna a o šířku se bude dělit s částí s informacemi. Toto umístění jsem zvolil proto, že je ve většině zemí zvyklostí číst zleva doprava a proto umist'uji důležitější část doleva, kde se nejprve upře pozornost uživatele. již zmíněná část, kde se budou zobrazovat informace bude umístěna vpravo podle výše zmíněného pravidla.

4.1.2 Komponenty částí grafického rozhraní

Do dříve popsaných částí bude potřeba implementovat následující komponenty s danou funkcionalitou a rozložením jako je ná obrázku 4.2.

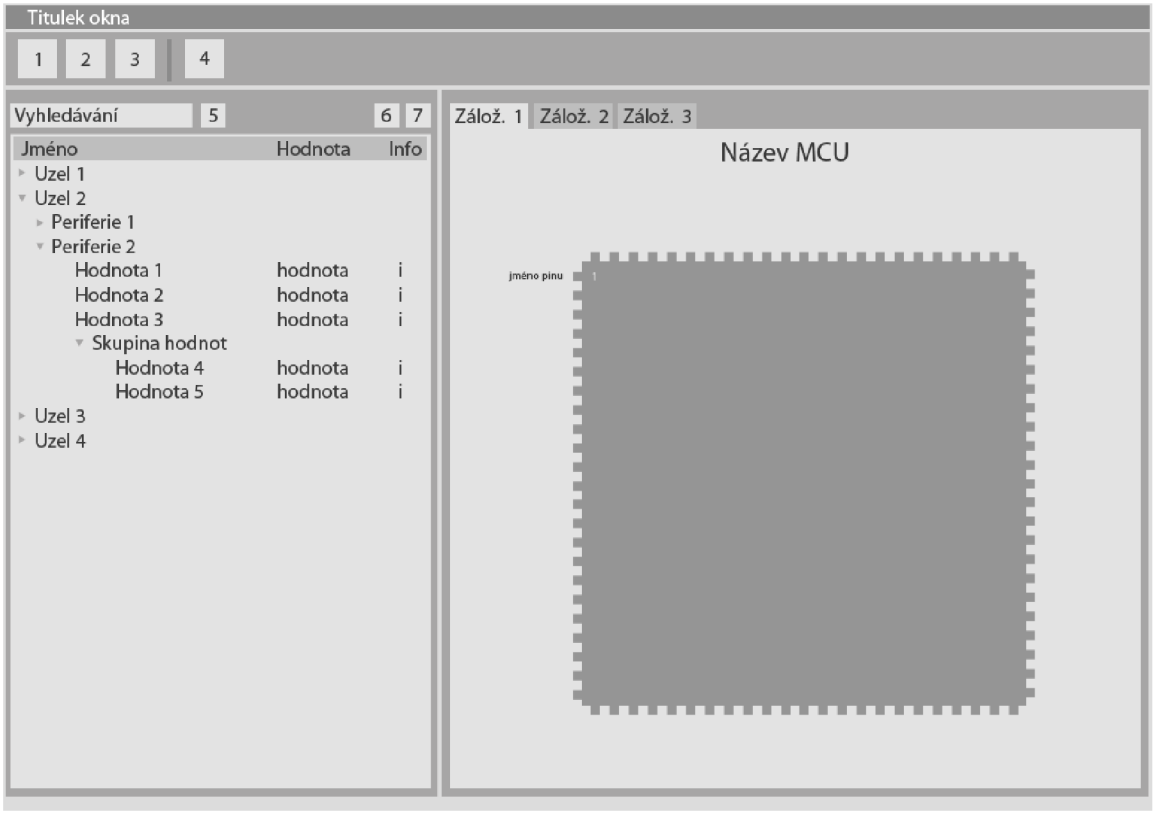
V části, kde se nachází panel s tlačítky, bude potřeba implementovat tlačítka, která budou vykonávat následující funkce. Tlačítko 1 bude otvírat dialog pro vytvoření nového projektu, tlačítko 2 bude spouštět dialog pro otevření staršího projektu, tlačítko 3 bude ukládat vytvořený projekt a to tak, že pokud není uložen, spustí dialog , kde uživatel zadá kam se má projekt uložit, nebo pokud už projekt jednou uložen byl, uloží se opět s novými hodnotami. Po kliknutí na tlačítko 4 se spustí dialog ve kterém uživatel určí složku do které se mají vygenerované zdrojové soubory uložit, poté se dané nastavení zkontroluje a pakliže není chyb tak se zdrojové soubory vygenerují.

V části, která bude sdružovat komponenty pro nastavování MCU bude třeba implementovat komponentu, která bude zobrazovat hodnoty a periferie ve stromové struktuře.

Obrázek 4.1: Návrh rozložení sekcí grafického rozhraní

Stromová struktura nejlépe splňuje požadavky na zobrazení dat těchto objektů. Hlavním požadavkem je přehlednost, díky stromové struktuře lze sdružovat periferie stejného druhu, lze schovat periferie, které se zrovna nenastavují a také je zajištěno zobrazení sounáležitosti hodnot a periferií. Komponenta bude vertikálně rozdělena na tři sloupce, kde první sloupec bude zobrazovat jméno uzlu, periferie či hodnoty, v další sloupec se bude nastavovat hodnota pro periferie a poslední sloupec ponese symbol nápovědy, pokud bude pro danou položku dostupná. V této části grafického rozhraní se také bude nacházet pole, do kterého bude uživatel vkládat řetězec, který chce hledat ve stromu. Pokud bude řetězec nalezen, řádek na kterém se výraz vyskytuje bude označen. Potvrzení vyhledávání bude zajišťovat tlačítko 5 a smazání označení u nalezených prvků bude zprostředkovávat tlačítko 6. Po kliknutí na tlačítka 7 a 8 se rozbalí či sbalí celý strom periferií.

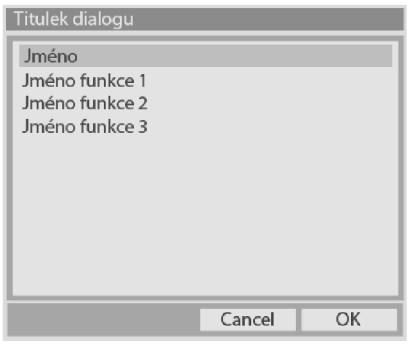
Část s informacemi bude obsahovat komponentu záložek, protože bude aplikace zobrazovat více různorodých informací a také proto, aby bylo na informace dostatek místa. Program bude obsahovat záložku s náhledem na mikrokontrolér a jeho rozložení pinů, také bude informovat o tom jaké piny zabírá daná periferie a také Po kliknutí na pin zobrazí všechny periferie využívající pin a informaci o jejich stavu (zdali jsou povoleny či zakázány). Další záložka bude obsahovat text s nápovědou, tento text se zobrazí kdykoliv uživatel klikne na značku nápovědy v komponentě zobrazující strom periferií. Tato nápověda bude pro větší přehlednost graficky stylizována. Podobně jako záložka nápovědy, bude dále záložka pro výpisy z programu obsahovat komponentu, zobrazující nastylovaný text, tak aby bylo od prvního pohledu zřejmé, že výpis znamená chybu, upozornění, nebo



Obrázek 4.2: Návrh rozložení grafického rozhraní

je pouze informativní. Komponenta záložek také umožní kdykoliv přidat další funkcionalitu aplikace, která se vloží na novou záložku a nenaboura tak vzhled a rozložení komponent aplikace. Záložky budou pohyblivé a bude možnost je schovat a opětovně zobrazit.

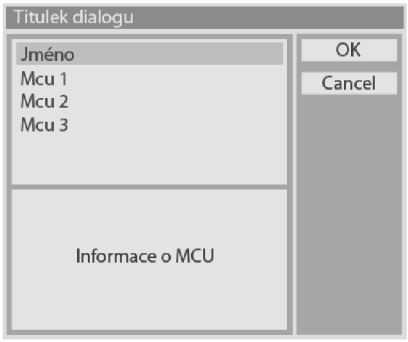
4.1.3 Návrh vlastních dialogů



Obrázek 4.3: Návrh dialogu pro zvolení generovaných funkcí.

Aby se nesnížila přehlednost stromu periferií, bude výběr funkcí, které se mají pro danou periferii generovat, umístěn do samostatného dialogu. Návrh rozmístění komponent je vidět na obrázku 4.3. Tento dialog bude rozdělen na dvě části, první část která se nachází

výše, bude obsahovat komponentu zobrazující seznam funkcí, které se mají generovat, spolu s ikonou která symbolizuje, zda se daná funkce bude či nebude generovat. Druhá část bude obsahovat tlačítka kterými se výběr potvrdí nebo se zahodí.



Obrázek 4.4: Návrh dialogu pro výběr nového projektu (resp. MCU).

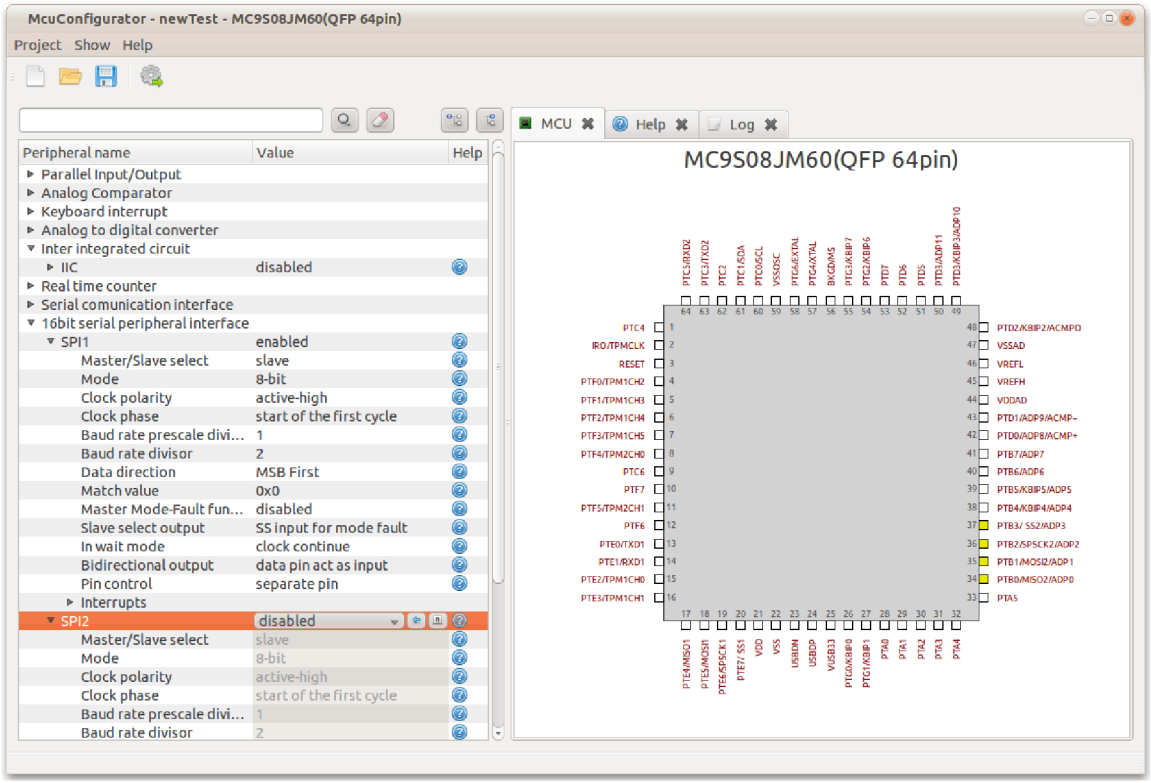
Aby bylo uživateli hned zřejmé jaké MCU může konfigurovat, bude pro otevření nového projektu (respektive načtení nového kontroléru) vytvořen dialog, pomocí kterého se bude tato operace provádět. na obrázku 4.4 je znázorněn návrh tohoto dialogu, tento dialog se také dělí na dvě části. První část obsahuje komponentu, kde se v zobrazí veškeré dostupné MCU ve formě seznamu, tyto informace budou uloženy v souboru a program si je Při každém startu načte. pod tímto seznamem se bude nacházet komponenta, která bude zobrazovat informace o MCU, jako jsou jméno, výrobce, typ pouzdra, počet pinů a další dodatečné informace, které se specifikují v souboru s informacemi o MCU. V druhé části se budou nacházet tlačítka pro potvrzení, či zrušení výběru.

4.1.4 Výsledný návrh v Qt

Na realizaci hlavních komponent v Qt, bude použito následujících objektů. pro zobrazení periferií a jejich komponent bude použit `QTreeView`, jeho výhoda spočívá v tom, že sám o sobě neuchovává data, pouze realizuje pohled na data. Proto nebude potřeba mít v programu duplicitní data (jedny pro vnitřní reprezentaci MCU a jedny v komponentě, která je zobrazuje). na vykreslování rozložení MCU a obsazení pinů, bude použito `QWidget`, do kterého se budou informace kreslit. Jelikož je to třída Qt nebude potřeba řešit integraci do prostředí (automatické nastavování šířky, přijetí zpráv a podobně). na zobrazení nápovědy a textového výstupu programu bude sloužit `QTextEdit`, díky jeho podpoře Rich Textu, který umožňuje stylovat předávaný text pomocí HTML a CSS značek. Nakonec, pro seznamy v obou dialozích, bude použito `QListWidget` komponenty. U seznamů v dialozích se nepředpokládá velký objem dat, proto nebude vadit duplicita, proto bude použito právě této komponenty. Výsledný layout grafického uživatelského rozhraní, na platformě Linux v prostředí Gnome 2.32.0, lze vidět na obrázku 4.5.

4.2 Návrh implementace souborů popisujících MCU

Struktura popisovaného MCU bude rozdělena do čtyř souborů. Tyto soubory budou postupně popisovat, rozložení pinů, paměť, periferie a MCU jako celek. Tyto i ostatní soubory, ze kterých program čte data budou ve formátu XML, z důvodu dobré čitelnosti takovéhoto zápisu (jelikož neexistuje nástroj, který by soubory tvořil automaticky, je potřeba soubory



Obrázek 4.5: Výsledná implementace grafického rozhraní v Qt pod Gnome 2.32.0

popisující MCU tvořit ručně) a také, protože samotné Qt nabízí prostředky pro snadné načítaní souborů XML a zpracování jejich obsahu. aby se ještě více zpřehlednila a zjednodušila práce s načítaným XML souborem, bude struktura XML souboru popsána souborem XSD, odpadne tak kontrola správnosti načítaných XML souborů, typů hodnot v elementech, unikátnosti některých hodnot a také Po vhodném okomentování těchto souborů mohou sloužit jako návod jak vytvářet soubory XML popisující MCU.

4.2.1 Soubor popisující vývody

Pro popsání rozložení pinů, bude potřeba ukládat do soubou jejich jména, která se zobrazí v komponentě, která zobrazuje nákres MCU. Dále bude potřeba uložit unikátní číslo vývodů, pomocí kterého se na ně bude moct odkazovat v souboru s periferiemi. Tento souboru musí také obsaovat informace o tvaru pouzdra a počtu pinů pro dané pouzdro.

4.2.2 Soubor popisující paměť

K popsání paměti MCU je potřeba souboru ve kterém budou popsány registry, svým unikátním jménem a velikostí. Každý registr bude také obsahovat bity (a to tolik, jaká je jeho velikost), které budou popsány unikátním jménem, jejich výchozí hodnotou, informací, zda je lze nastavit a také informaci do jaké skupiny daný bit patří. Pokud se bit nedá nastavit, nebude třeba vyplňovat jeho jméno (vetšinou takové bity ani jméno nemají), příslušnost bitu ke skupině bude také volitelným parametrem.

4.2.3 Soubor popisující periferie

V souboru s popisem periferií MCU bude hierarchie určena pomocí elementu uzel, u kterého bude třeba uchovávat pouze jméno. Uzel bude moci obsahovat další uzly, nebo periferie. Samotná periferie bude moci obsahovat uzel, ale jen v jedné úrovni (bude sloužit k oddělení hodnot). Periferie bude popsána svým jménem, dále jménem které lze vkládat do kódu, nápovědou k dané periferii, sezname pinů, které periferie Při své aktivaci využívá (určené unikátními čísly), seznamem funkcí, které bude možno pro danou periferii generovat, včetně funkce pro inicializaci periferie. Nakonec bude také obsahovat hodnoty, které které ovlivňují nastavení periferie. Hodnota periferie bude popsána jménem, nápovědou vztahující se k dané hodnotě, typem, který bude dále určovat, co se v popisu hodnoty může vyskytovat. Pokud bude periferie bitem nebo seznamem hodnot, bude nutno uvést ty možnosti, kterými bude tento seznam hodnot či bit disponovat. Spolu s těmito možnostmi bude potřeba uvést, zdali se nastavením dané položky obsadí pin, nebo pokud je potřeba pro zadání dané možnosti nastavit i jiné hodnoty. Takováto závislost bude určena unikátním jménem hodnoty na kterou se vztahuje a také požadovanou velikostí této hodnoty, pro bit, nebo minimální a maximální velikostí hodnoty, pro ostatní typy. Pokud bude hodnota skupinou či registrem, bude se v ní vyskytovat informace o minimu a maximu jaké může hodnota nabývat. Důležitou hodnotou pro oba případy bude odkaz na paměťový element, který pojímá dané nastavení (odkazem bude jeho unikátní jméno).

4.2.4 Soubor celkového popisu MCU

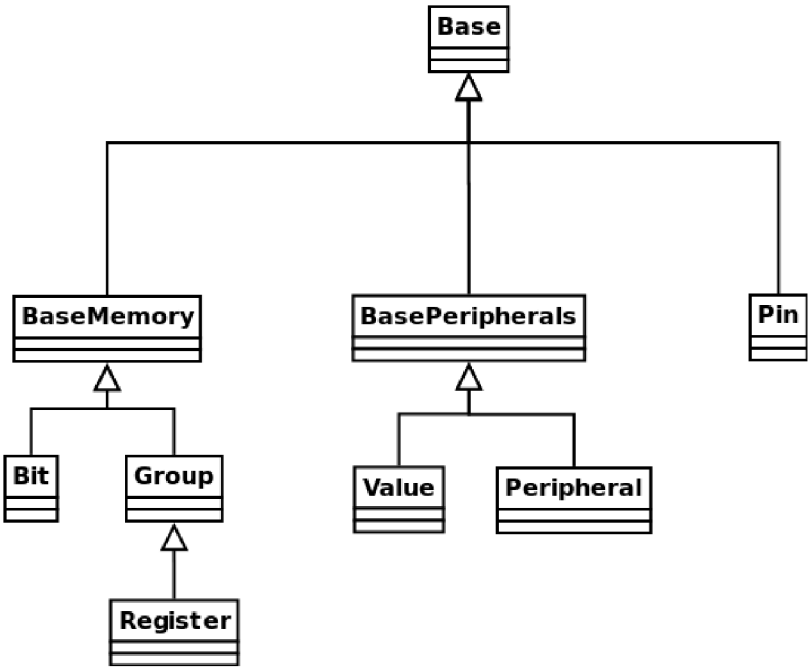
Nakonec je potřeba souboru, který výše popsané soubory bude sdružovat do celku. V tomto souboru bude popsán MCU jako celek, bude v něm uveden název MCU, výrobce, pouzdro, počet pinů, dodatečné informace a také cesty k souborům s popisem rozložení pinů, paměti a periferií.

4.3 Návrh implementace datové části

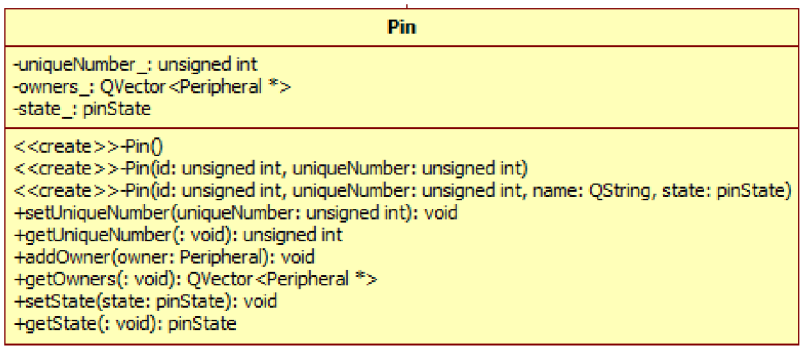
Popis MCU pomocí XML je vhodný pro čtení a pro přehledné vytváření, ale není dobrý pro přímou práci s daty která nese. Proto bude potřeba vše, co je popsáno XML souborem , načíst do paměti a doplnit o další potřebná data pro zpracování v programu. na obrázku 4.6 je zobrazená hierarchie datové části programu. Jelikož je v programu uplatňován objektově orientovaný návrh, tak jsou data patřící dané součásti MCU zapouzdřeny do objektu, který tuto součást reprezentuje. Datová část je opět rozdělena, tentokrát na tři celky, tak jako jsou tři soubory popisující piny, paměť a periferie. Veškeré třídy které popisují součásti MCU budou odvozeny od třídy `Base`, tato třída bude obsahovat parametry, které by jinak byly v každé třídě zvlášť. Těmito parametry jsou jméno, unikátní číslo a informace o tom, jestli byl změněn obsah.

4.3.1 Třídy popisující piny

Objekty reprezentující piny MCU budou dva, jeden objekt bude reprezentovat samotný pin a jeden tyto piny bude sdružovat do modelu pinů. pro první třídu je nutné zajistit metody, pomocí kterých se bude tento pin tvořit a naplnit jej příslušnými hodnotami. Nejdůležitější hodnotou bude unikátní číslo, pomocí kterého se bude na pin odkazovat v jiných souborech. Tato třída bude také obsahovat seznam odkazů (ukazatelů) na periferie, které



Obrázek 4.6: Hierarchie datových tříd.



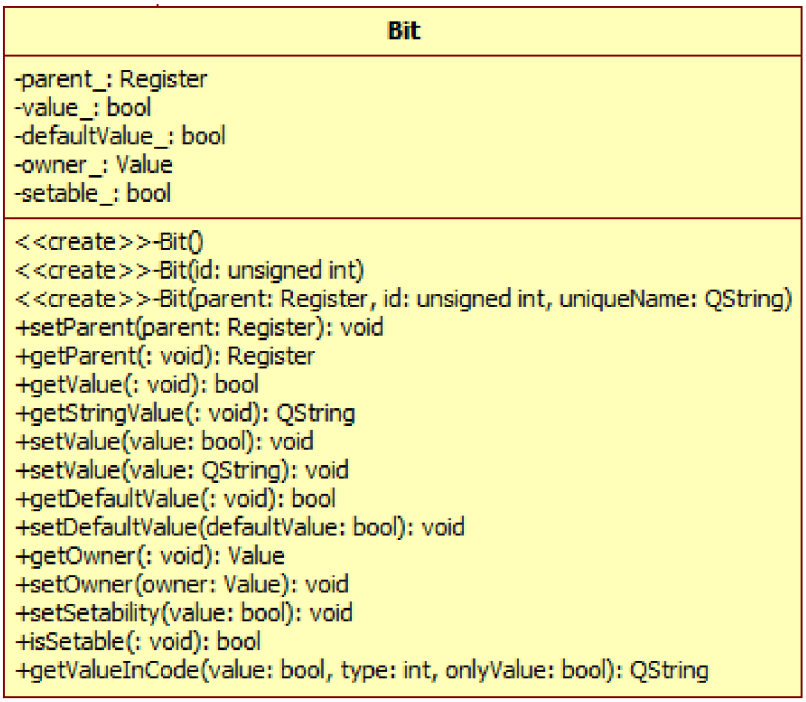
Obrázek 4.7: Náhled na třídu Pin.

daný pin využívají. Což bude důležité pro pozdější kontrolu nastavení. Druhou třídou spadající pod model pinů MCU je třída, která bude sdružovat objekty pinu a to postupně za sebou, tak jak byly vytvořeny. Třída bude obsahovat i informace o počtu pinů a o typu pouzdra. Nad objekty pinů bude zajišťovat metody pro jejich vytvoření, mazání, hledání a také pro postupné procházení pinů. Bude potřeba implementovat metodu, která Po předání cesty k XML souboru s popisem pinů daného MCU, tyto informace načte a vytvoří objekty pro všechny piny a utvoří tak model pinů. Náhled na strukturu této části návrhu je vidět na obrázku 4.7.

4.3.2 Třídy popisující paměť

Další datová část aplikace je část popisující paměť, tato část bude obsahovat pět tříd. Její třída `BaseMemory` je odvozena od třídy `Base` a rozšiřuje její vlastnosti o položku unikát-

ního jména paměťového prvku a informaci o bezejmennosti objektu. Z této třídy se bude odvozovat třída `Bit` a `Group`.

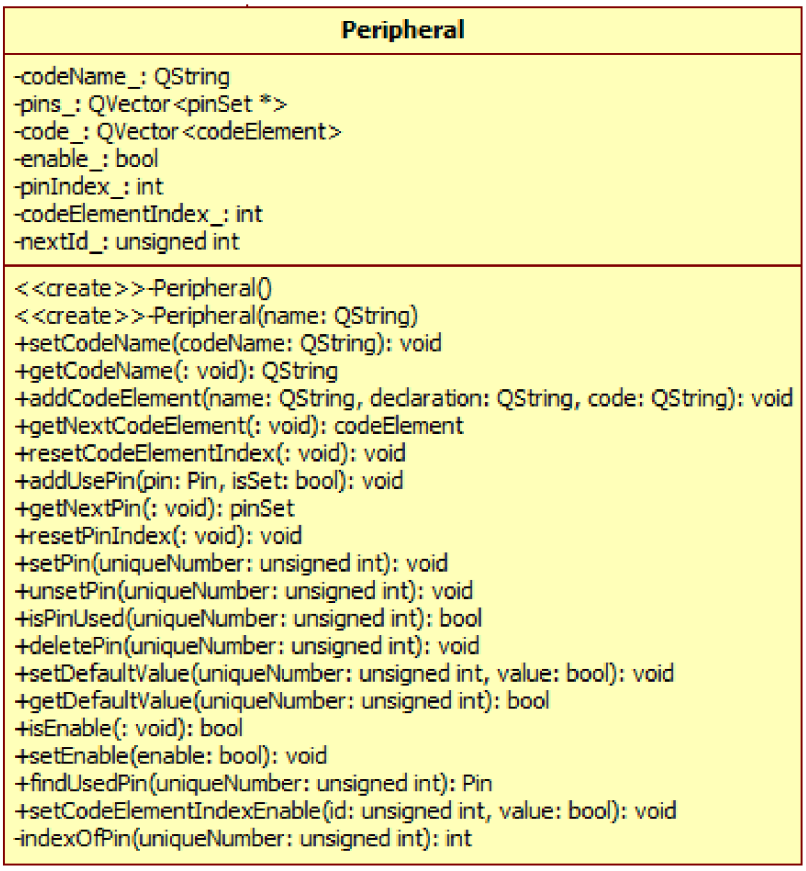


Obrázek 4.8: Náhled na třídu Bit.

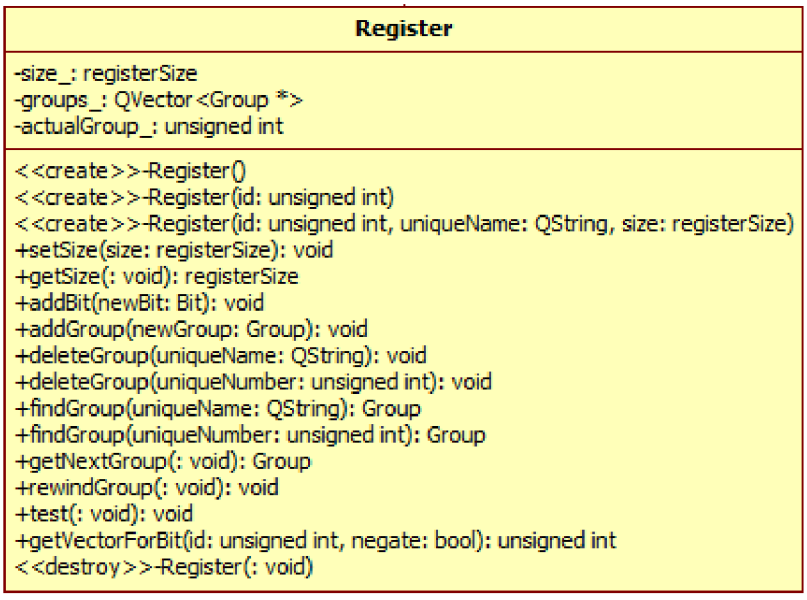
Třída `Bit` reprezentuje základní prvek paměti, třída bude uchovávat v prvé řadě informace obsažené v souboru popisující paměť MCU, ale dále bude obsahovat metody, které poskytují možnost uložit hodnotu číslem nebo hodnotou popsanou řetězcem a také nastavenou hodnotu bitu vrátit jak číselně, tak i jako řetězec. Další důležitou metodou bude ta jenž bude vracet hodnotu bitu i s přiřazením do patřičného místa v paměti, toto místo v paměti je popsáno svým jménem v hlavičkovém souboru pro daný mikrokontrolér, toto jméno by mělo být i unikátní jméno bitu. Mimo této metody bude obsahovat i metodu, která vrátí hodnotu bitu v patřičném formátu v jazyce C++, tak jako metoda předchozí, ale nebude obsahovat přiřazení. U bitu bude nutno uchovávat i odkaz na periferii která s hodnotou pracuje, odkaz na registr ve kterém se bit nalézá a také informaci o nastavitelnosti bitu.

Další třídou bude třída `Group`, která bude reprezentovat skupinu v registru, tato skupina sdružuje bity, které mezi sebou mají souvislost. Třída bude uchovávat pouze odkazy na bity, které do skupiny náleží a nad těmi bude poskytovat uložení hodnoty, která je zadána buď číselně, nebo jako řetězec znaků v určitých formátech. Těmito formáty jsou binární číslo, hexadecimální číslo nebo dekadické číslo. Tyto formáty budou shodné s formáty zápisů v jazyku C. Takto uložené hodnoty bude možno pomocí příslušných metod i číst a to i v příslušných formátech. tak jako třída `Bit`, bude třída `Group` obsahovat metodu, pomocí které bude možno získat řetězec znaků odpovídajících zápisu v jazyce C, který bude nastavovat danou skupinu v MCU. Skupina bude také poskytovat metodu, kterou bude možné přidávat bity a také bity procházet a mazat.

Následující třída, která spadá pod paměť, bude třída `Register`, tato třída bude reprezentovat registr mikrokontroléru, bude odvozena od třídy `Group`, kterou rozšíří o část, kde se



Obrázek 4.9: Náhled na třídu Group.



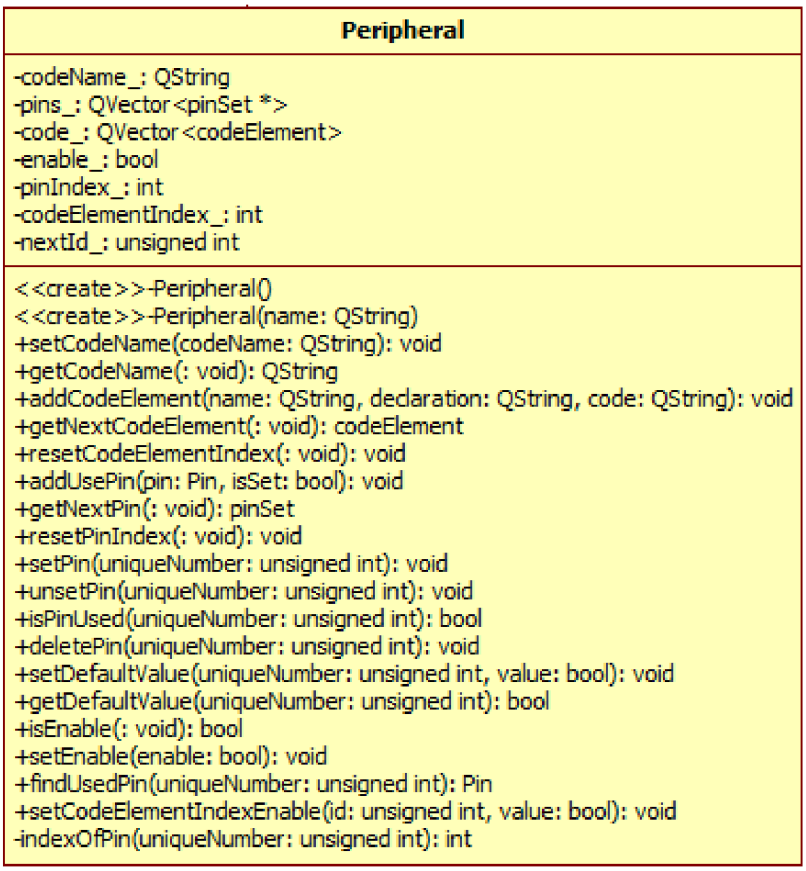
Obrázek 4.10: Náhled na třídu Register.

budou uchovávat právě všechny skupiny, které daný registr MCU obsahuje. Bude také provádět operace vytváření, mazání procházení a vyhledávání skupin. Dále bude obsahovat metodu, která poskytne bitový vektor, jak pro operaci bitového součtu, tak i pro operaci bitového součinu, tento vektor se následně využije pro nastavení hodnot skupiny, která není implementována v hlavičce pro daný mikrokontrolér. Třída `Register` třídu `Group` rozšíří i o omezení, které je dáno velikostí registru, respektive skupina bude moci být libovolně velká, ale registr jen 8,16 či 32 bitů.

Všechny tyto třídy popisující paměť bude sdružovat třída MemoryModel, která, jak již název napovídá, reprezentuje model paměti MCU. Třída poskytne možnost objekty nejen vytvářet a mazat tak, aby byly zachovány unikátní hodnoty id, ale také poskytne vyhledávání skupin, bitů a registrů. Jednou její složkou bude seznam speciálních skupin, které mohou sdružovat bity napříč registry, proto ale nemohou patřit žádnému registru a budou umístěny zvlášť. V neposlední řadě bude třída obsahovat metodu, která načte obsah a strukturu z XML souboru.

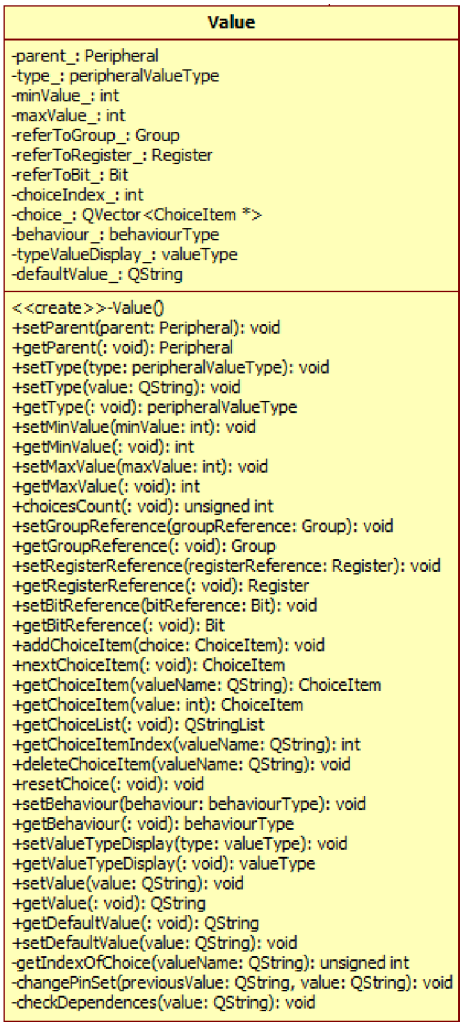
4.3.3 Třídy popisující periferie

Poslední skupinou tříd, jsou třídy uchovávající informace o periferiích MCU a jejich nastavení. Třídy, které se přímo podílejí na reprezentaci, budou odvozeny ze třídy `BasePeripherals`, tato třída rozšiřuje třídu `Base` o parametry nápovědy k danému objektu a odkazu na položku stromu ve které se daný objekt uchovává.



Obrázek 4.11: Náhled na třídu Peripheral.

Třídou reprezentující periferii MCU bude třída `Peripheral`, ta v sobě ponese nejenom informace jako je unikátní jméno, id a další které obsahuje XML soubor, ale také nese seznam funkcí, které se budou pro danou periferii generovat, pokud bude aktivována. S tímto také souvisí položka, která ponese informaci, zdali je periferie aktivována či ne. Třída `Periferie` v sobě také ponese informaci jaké piny daná periferie obsazuje. U obou seznamů (funkcí ke generování a obsazovaných pinů) bude třída zajišťovat vytváření, vyhledávání, procházení a mazání prvků těchto seznamů.



Obrázek 4.12: Náhled na třídu Value.

Vlastnosti, které bude možno u periferie nastavovat, bude reprezentovat třída `Value`, která bude obsahovat typ hodnoty kterou reprezentuje, odkaz na část paměti , která se touto třídou nastaví a také informaci o minimu a maximu jakou může hodnota nabývat, pokud se bude jednat o typ registr nebo skupina. Pokud třída `Value` bude reprezentovat seznam hodnot, tak bude naplněn i seznam těchto hodnot. Dané hodnoty jsou pak reprezentovány třídou `ChoiceItem`, která bude popsána níže. Další informací, kterou bude tato třída chovávat, bude informace o výchozí hodnotě. Mimo data, objekt poskytne i metody, které s těmito daty pracují, metody které pracují se seznamem hodnot a metody, které

budou zajišťovat nejen správné načtení reprezentované hodnoty, ale také správné uložení. Bude potřeba kontrolovat nejenom rozsah daný uživatelem, ale také se postarat o správný formát řetězce, pokud bude takto hodnota předána. Tato třída se bude starat o nastavení informace o změně obsazení pinů.

Další třídou, která se bude podílet na reprezentaci nastavení, je třída `ChoiceItem`, tato třída reprezentuje předem definovanou hodnotu nastavení periferie. Tato hodnota sebou nese také rozšiřující informace o změně obsazení pinu či více pinů periferií, může obsahovat i seznam závislých hodnot a jejich velikostí či nastavení jenž musí být nastaveny, jak je předepsáno, aby byl výběr splněn. Toto se nebude kontrolovat Při nastavování periferie, ale až před generováním výsledných zdrojových souborů. Položka volby reprezentována touto třídou je určená svým jménem a hodnotou, kterou reprezentuje. Také jako ostatní třídy bude `ChoiceItem` obsahovat metody pro práci se seznamem závislostí a seznamem pinů které se mají obsadit.

Poslední třídou této části bude třída `PeripheralsModel`, která mimo metod pro načítání dat a struktury z XML, ukládání projektu do XML, načítání projektu z XML, vytváření a mazání periferií a hodnot bude především orientována na vyhledávání hodnot a periferií. Bude obsahovat metody pro vyhledávání periferií, periferií které jsou povoleny a objekty hodnot periferií, dále bude potřeba hledání funkcí dané periferie a také hledání hodnot a periferií podle unikátních jmen nebo čísel. Tato vyhledávání budou stěžejní pro pozdější operace v GUI a při generování výsledných zdrojových souborů s funkcemi pro nastavení MCU. Třída bude také uchovávat dodatečné informace, které se budou přidávat do hlavičky pro zdrojový soubor s nastavením a které jsou důležité pro správné fungování generovaných funkcí.

4.4 Návrh implementace generátoru zdrojových souborů

Výstupem aplikace budou zdrojové soubory, které budou nastavovat periferie MCU, toto bude zajišťovat třída generátoru. Tato třída bude nejprve muset projít periferie, které uživatel nastavil a zkontrolovat jestli se nedopustil chyby nastavení, nebo nastavení které nemá žádný smysl. Tyto informace bude přebírat z tříd modelu periferií, kde se bude zaměřovat hlavně na informace o závislostech jednotlivých hodnot a také informace o obsazení pinů. Pokud se nějaká chyba vyskytne, dá se toto vědět uživateli, který se pak bude moci rozhodnout jestli dále v generování pokračovat. Výstupem generátoru budou dva soubory, hlavičkový, ve kterém bude jednak popis pro který MCU tyto soubory jsou, ale také deklarace funkcí, které se vygenerovaly na popud uživatele. Druhým souborem bude zdrojový soubor s funkcemi, které si uživatel pro danou periferii zvolí. Každá periferie bude vlastnit minimálně funkci pro její inicializaci, dále pak může obsahovat funkce pro obsluhu přerušení nebo pro obsluhu periferie samotné. Bloky souvisejících funkcí budou odděleny komentáři, které budou obsahovat informace o nastavení periferií.

Kapitola 5

Implementace

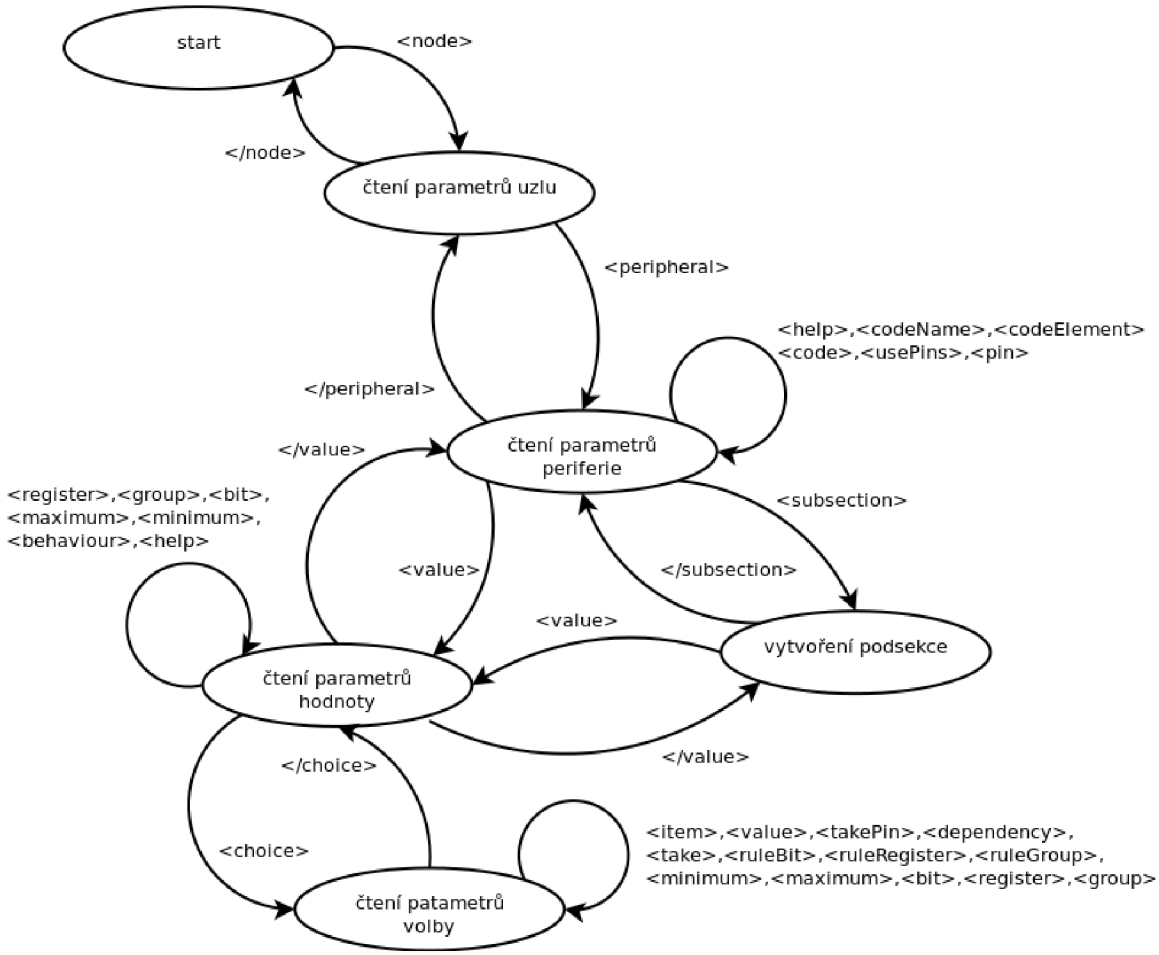
Implementace probíhala také Po sekcích stejných jako u návrhu implementace. Nejprve byla implementována datová část spolu s načítáním souborů XML, posléze se implementovalo uživatelské rozhraní spolu s moduly, které zobrazovaly různé náhledy na data o MCU. Nakonec se pracovalo na generátoru, který pro svou funkci potřebuje datovou část programu i část uživatelského rozhraní. V textu níže budou uvedené některé třídy a metody z těchto částí implementace.

5.1 Implementace zpracování XML souborů

V několika třídách bylo potřeba implementovat načítání i ukládání dat do XML souboru. Jelikož se dané případy liší jen elementy, které daná metoda prochází, respektive na které reaguje, popíši tento mechanizmus v samostatné kapitole.

5.1.1 Čtení XML souborů

V metodě, která čte XML soubor se nejprve otevře XML soubor s daty a hned poté XML soubor s XSD popisem dokumentu s daty. Toto otevření zařizuje třída `QFile`, která také uchovává soubor otevřený pro další práci. Po tomto načtení je třeba vytvořit instanci třídy `QXmlSchemaValidator`, která se stará o validaci souborů XML, tomuto objektu se nejprve předá soubor se schématem, jenž metoda zpracuje pro svou vnitřní potřebu a dále jej i zkontroluje, zda není již samotný soubor chybný. Pokud načtení proběhlo správně, zkontroluje se validnost XML souboru vůči danému XSD schématu pomocí metody `validate()`, kterou taktéž poskytuje třída `QXmlSchemaValidator`. Jakmile je XML soubor zkontrolován, předá se dále objektu třídy `QXmlStreamReader`, který dovoluje číst XML soubor Po elementech, které se zde nazývají tokeny. Pomocí metody `readNext()` se postupně čtou elementy, tak jak jsou hierarchicky uspořádány v XML souboru. Posléze přichází samotné čtení dat, které je ve metodách starajících se o čtení XML dvojího typu a to jednoduché a stavové. Jednoduché čtení probíhá v cyklu, ve kterém se nejprve načte token (resp. element), u jednoduchého načítaní se detekuje pouze token typu `StartElement`, poté se pomocí metody `name()` zjistí jméno načteného elementu. Zjištěné jméno se porovná a podle výsledků porovnání se data obsažená v elementu načtou pomocí metody `readElementText()`. na data obsažená v parametru elementu se používá metoda `attributes()`. U stavového čtení se využívá konečného automatu, který také v cyklu čte a detekuje tokeny typu `StartElement` a `EndElement`. Příklad konečného automatu který čte XML soubor je na obrázku 5.1. na obrázku jsou na přechodech zaznačeny načítané tokeny, pomoci kterých se automat pohybuje



Obrázek 5.1: Diagram automatu, pro čtení XML dokumentu periferií.

mezi stavy, obvykle se otevíracím tokenem vstupuje a zavíracím se vystupuje ze stavu automatu, dále jsou na obrázku zakresleny stavy do kterých automat nabývá. Automat čte informace o periferiích MCU a jejích hierarchii, je tvořen pomocí příkazu `switch`, kde jeho každý blok zpracovává jeden z typů elementů XML souboru. aby se nemuselo složitě zjišťovat do kterého objektu se má zrovna zapisovat, slouží k tomu předem vytvořené ukazatele na třídy objektů, které se budou vytvářet (respektive se budou načítat jejich data). Automat pak v elementu, který symbolizuje vytvoření jednoho z objektů (periferie, hodnota, uzel nebo položku seznamu hodnot) zkontroluje, jestli je přípustné v daném stavu objekt vytvořit, pokud ano, tak jej vytvoří do předem definovaného ukazatele, který se pak dále používá v kódu. Při vytváření nového objektu se také většinou přechází do nového stavu. Zpátky do stavu předchozího se dostává ve chvíli, kdy se detekuje uzavírací značka daného elementu (token typu `EndElement`). Stav ve kterém se automat zrovna nachází se kontroluje u bloků příkazu `switch`, které jsou unikátní pro daný stav, respektive se čtou data, která jsou jen pro daný objekt a v jiném se nevyskytují, u ostatních se musí kontrolovat v jakém stavu se automat nachází a podle toho rozhodnout do kterého objektu se bude daná informace ukládat. Jakmile se přečte celý soubor, to se detekuje pomocí metody `atEnd()`, zavřou se soubory a čtení končí.

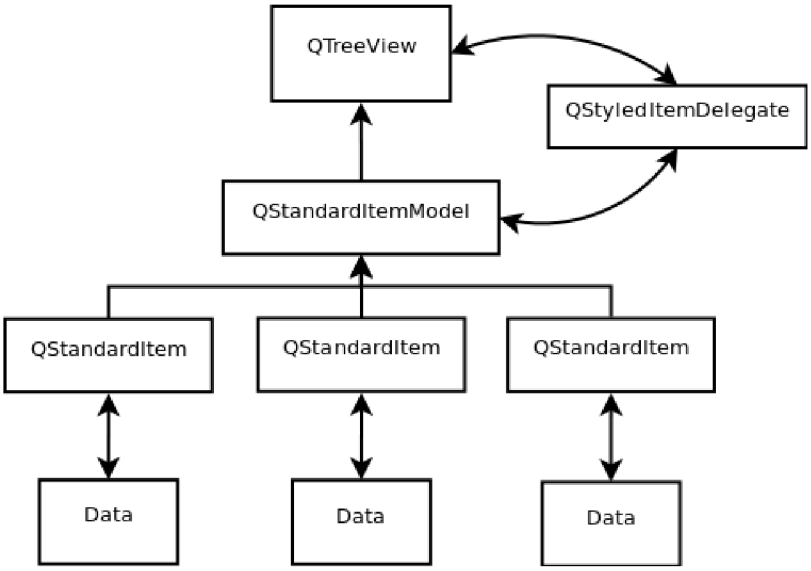
5.1.2 Zápis XML souborů

Ukládání dat do XML dokumentu, probíhá tak, že se nejprve otevře pomocí objektu `QFile` soubor pro čtení. Takto otevřený soubor se vloží do objektu `QXmlStreamWriter()`, pomocí kterého lze jednoduše zapisovat. Zápis začíná metodou `writeStartDocument()`, která do XML vloží element s informacemi o verzi, poté se zapisují další elementy metodou `writeStartElement()` do které se také vloží jméno elementu, dále se vkládají atributy elementu a to pomocí metody `writeAttribute()`, které se předá jméno parametru a jeho hodnota, element se uzavírá metodou `writeEndElement()`. Toto se děje cyklicky dokud nejsou zapsány všechny potřebné elementy. Nakonec se zavolá metoda `writeEndDocument()`, kterou se zápis dokumentu uzavře. Nakonec se také jako Při načítání uzavře soubor do kterého se dokument ukládal.

5.2 Implementace Gui

Nejdůležitějšími komponentami které se podílejí na uživatelském rozhraní jsou třída `SettingsView`, která poskytuje zobrazení a nastavování hodnot jednotlivých periferií a třída `McuView`, která zprostředkovává grafický náhled na MCU, jeho rozložení pinů a další dodatečné informace. Tyto třídy a třídy které se na jejich funkcionalitě podílejí budou popsány v dalších odstavcích.

5.2.1 Komponenty pro zobrazení stromu periferií



Obrázek 5.2: Pozice delegáta Při model-view programování v Qt.

Třída `SettingsView` dědí z Qt třídy `QTreeView`, tato třída zobrazuje pohled na data, která jsou hierarchicky uspořádána v třídách `QStandardItemModel` a `QStandardItem`, v jejich instancích jsou uloženy periferie a jejich hodnoty. `QTreeView` zajišťuje vykreslení těchto dat a práci s nimi. V třídě `SettingsView` byla třída `QTreeView` rozšířena hlavně o slot hledání, který zprostředkovává nalezení a označení řádků se shodným řetězcem jako je metodě předán. Také byl implementován slot, pomocí kterého lze automaticky rozevírat uzly stromu

a to až ke kořenu. Dále byla přidána metoda, která označí nebo od označí řádky, na kterých se nachází nekorektní hodnota a také o metodu, která takto zaznačené řádky nastaví na výchozí hodnotu. jak již bylo řečeno, třída zobrazuje data, která jsou nesena v objektech `QStandardItem`, takto dokáže zobrazit jen omezené množství datových typů, v modelu periferií se ale do `QStandardItem` vkládá celý objekt periferie či hodnoty. aby bylo možno tato data zobrazit a měnit jejich hodnoty, je třeba pro daný sloupec, ve kterém se bude s hodnotami pracovat, připravit delegáta, který bude mít toto na starosti [13]. Princip delegáta je zobrazen na obrázku 5.2, který zobrazuje delegáta jako rozhraní mezi uživatelem a daty, které jsou uloženy v modelu. na jedné straně delegát zobrazí data nesená v prvku modelu, uživatel je nastaví a na straně druhé je delegát uloží do modelu a zobrazí v TreeView widgetu. Qt nabízí třídu `QStyledItemDelegate` ze které pak dědí vlastní třída `TreeDelegate`, většina metod, které třída `QStyledItemDelegate` nabízí, jsou virtuální a byla potřeba je naimplementovat [14]. je třeba podotknout, že pro správný chod delegáta není potřeba implementovat všechny metody. Důležitá metoda je metoda `createEditor()` v této metodě se tvoří widget, který se zobrazí na místě buňky `SettingsView`, tato metoda nejprve zjistí pro jaký typ objektu se bude widget generovat, pokud se jedná o periferii vygeneruje se `QComboBox` pro nastavení aktivity periferie, tlačítko pro nastavení původní hodnoty a tlačítko, které spouští dialog pro výběr generovaných metod dané periferie. Pokud se jedná o hodnotu, tak se vytvoří widget, který obsahuje `QComboBox` pro hodnoty typu bit nebo seznam, nebo `QLineEdit` pro hodnoty typu skupina nebo registr, nakonec se vytvoří tlačítko, které nastavuje původní hodnotu. Takto vytvořené widgety metoda vrací a ty se posléze zobrazí v objektu `SettingsView`. Další důležitou metodou je `setEditorData()` tato metoda se volá pokud se v modelu změnila data, v této metodě se pak změní data i u objektů delegáta. Naopak když se nastaví data v delegátovi zavolá se metoda `setModelData()`. Tato metoda si nalezne v `QStandardItemModel` daný objekt pro který byl delegát vytvořen, poté se podívá jaký je to typ objektu a podle toho do něj uloží data z delegáta. Při tomto procesu se již kontroluje zda jsou hodnoty v příslušných mezích a zdali jsou hodnoty správného formátu a podobně. Pokud se vyskytne nějaká chyba, sdělí se to uživateli pomocí `QMessageBox`, a do delegáta se nastaví předchozí hodnoty.

5.2.2 Grafické znázornění MCU

Třída `McuView` je odvozena od třídy `QWidget`, třída vykresluje mikrokontrolér a jeho rozložení pinů. K tomu aby vykresloval, potřebuje předat naplněný objekt třídy `PinoutsModel`, ze kterého si pomocí privátní metody `makeCopy()` vytvoří vnitřní reprezentaci tvaru pouzdra mikrokontroléru a jeho rozložení pinů a to tak, že postupně prochází modelem pinů a pro každý vytvoří nový záznam do seznamu, do tohoto záznamu doplní ukazatel na pin a informace, jestli je zvýrazněný, nebo se jeho text bude vypisovat svisle, nakonec se zavolá privátní metoda pro počítání pozic prvků `recaluculate()`. V této metodě se nejprve vypočte velikost a poloha pouzdra MCU a velikost a pozice nadpisu. Poté se postupně prochází seznamem pinů, a pro každý se vypočítá jeho velikost, která se odvozuje od velikosti pouzdra, následně se spočítá na jakých souřadnicích tento pin bude ležet. Podle velikosti pinu a jeho pozice se také spočítá pozice textu (jak číslo pinu, tak jeho jméno) a obdélník ve kterém se bude text vykreslovat.

Pro vykreslení samotné vnitřní reprezentace, se používá slot `paintEvent()`, který se volá Při signálech `paint` a `repaint` widgetu. Tato metoda prochází seznamem pinů a postupně je podle předpočítaných souřadnic vykresluje, pokud má některý pin nastaveno zvýraznění, vykreslí se jinou barvou. Pokud je požadováno svislé vykreslení textu, provede se nejprve

uložení stavu plátna, poté se přesunou souřadnice na místo kde se má text vykreslit, plátno se otočí o 90° a použije se standardní metoda pro vykreslení textu. Poté se už jen obnoví stav plátna.

Třída `McuView` také nabízí vlastní dialog, který se zobrazí Po kliknutí na některý z vykreslených pinů. Kliknutí se detekuje pomocí slotu `mouseReleaseEvent()`, v tomto slotu se projde celou vnitřní reprezentací a najde se pin na který se kliklo a vytvoří pro daný pin instanci třídy `ParentDialog`. Tato třída vykreslí jednoduchý dialog ve kterém se postupně pod sebou zobrazí položky, které reprezentují periferie využívající pin, tyto informace si zjistí z ukazatele na objekt `Pin`, který je Při jejím vytváření předán. Dále zobrazí červeným či zeleným obdélníkem u jména periferie, zda je či není tato periferie aktivní. Po kliknutí na položku dialogu se vyšle signál objektu `SettingsView` a tento signál rozevře danou periferii a přesune na ni focus.

5.2.3 Grafické znázornění nastavení registrů

Podobně jako předchozí třída, která vykreslovala tvar a rozložení pinů MCU, tak i třída `RegisterView` je odvozena od třídy `QWidget`. Tato třída vykresluje na plochu, kterou widget zaujímá, grafické znázornění registrů a bitů, které daná hodnota mění. Vykreslování těchto informací začíná Po obdržení ukazatele na objekt `Value`, na slotu `showValueRegister()`, pro kteroý se budou registry kreslit. V metodě obsluhující tento slot se nejprve rozpozná o jakou hodnotu se jedná, pokud hodnota nastavuje bit nebo skupinu, zjistí se, kterému registru náleží, pokud se nastavuje registr použije se přímo registr. Zjištěný registr se postupně prohledá a vytvoří se struktury pro jeho vnitřní reprezentaci a naplní se složky, které informují o jméně bitů, ukazatelů na tyto bity a informaci, zda daný bit nastavuje vybraná periferie. V rámci tohoto průchodu se uchová jméno a ukazatel na procházený registr. Pokud se obdrží hodnota, která nastavuje speciální skupinu, provede se toto pro každý registr ze kterého skupiny sdružuje bity. Po vytvoření vnitřní reprezentace se zavolá funkce `recalculate()`, která vypočítá pozice prvků vnitřní reprezentace.

Funkce `recalculate()` postupně prochází složky vnitřní reprezentace registrů a podle jejich pořadí (registrů i bitů) vypočítává jejich pozici na které se později budou vykreslovat. Pozice se počítají nejen pro prvky samotné, ale i pro místo, kde se bude vykreslovat jejich hodnota, popis či pořadí v registru. Při počítaní těchto hodnot je třeba používat proměnné typu `float`, a ty převádět na celočíselný formát, až když se ukládají do vnitřní reprezentace, jinak dochází k chybám Při převodu z `float` do `int` a při změně velikosti plátna se tyto chyby projevují deformací výsledného obrazu.

Následné vykreslení vnitřní reprezentace se děje pomocí slotu `paintEvent()` stejně jako u předchozí třídy. Taktéž se prochází vnitřním modelem a z informací, které jsou v něm uloženy, se vykreslí výsledný obraz. Při procházení bitů registru si kreslící metoda kontroluje příznaky zvýraznění bitu, tento bit pak vykreslí se žlutým pozadím, Po detekci příznaku nenastavitelného bitu tento bit obarví šedě.

5.3 Implementace datové části

V této části budou uvedeny některé z metod, které se starají o správné uložení a čtení hodnot a které slouží k hledání objektů `Peripheral` a `Value`.

5.3.1 Vyhledávání v datech

Vyhledávání je u většiny tříd řešeno procházením lineárního seznamu, protože je většina dat uložena v objektech typu `QVector`. Ovšem objekty typu `Peripheral` a `Value` nejsou uspořádány postupně za sebou, ale ve stromové struktuře pomocí `QStandardItem`. aby se daly tyto objekty vyhledávat, bylo třeba implementovat algoritmus procházení stromu preorder i postorder [7]. Tyto algoritmy byly implementovány následovně, nejprve se pomocí instancí třídy `QVector` vytvoří seznamy pro uzly, které se mají vyhledávat a pro nalezené objekty, které odpovídají kritériu hledání. Poté se v cyklu `while` nejprve zkontroluje, jestli není seznam prvků k prohledávání prázdný, pokud ano, cyklus se ukončí. Pokud je seznamu uzlů na prohledání neprázdný, získá se pomocí metody `pop_front()` první uzel seznamu a zjistí se, zdali má potomky. Poté se zkontrolují data uzlu na kritéria hledání, pokud se shodují data uzlu, objekt, který uzel uchovává, se umístí do seznamu nalezených objektů. Posléze se projde seznam potomků právě prohledávaného uzlu a umístí se pomocí metody `push_back()` na konec seznamu uzlů k prohledání (u algoritmu postorder) a nebo na začátek pomocí metody `push_front()` (u algoritmu perorder). Nakonec se předá seznam nalezených objektů.

5.3.2 Nastavování hodnot periferií

Jelikož jsou data o nastavení jednotlivých periferií ukládána pouze v objektech typu `Bit`, a to pouze binárně, bylo potřeba implementovat metody, které jsou schopny do objektů `Bit` ukládat hodnoty i jiných velikostí a jiného formátu než binární. pro ukládání číselných hodnot formou řetězce do skupiny bitů slouží metoda `setValueAsString()`, která pracuje následovně. Nejprve si z předaného řetězce pomocí metody `left()` do jiné ho objektu typu `QString` uloží prefix hodnoty, tento prefix je velký 2 znaky. Poté je porovná se známými prefixy a pokud se nějaký shoduje, uloží si typ hodnoty do proměnné typu `valueType`. Následně, pokud je formát řetězce rozeznán, se oddělí prefix (dekadická hodnota nemá prefix a proto se také z předaného řetězce nic neodděluje) a pomocí metody třídy `QString toInt()` převede řetězec na celé číslo. Toto číslo se nakonec předá metodě `setValueAsNumber()`.

Metoda `setValueAsNumber()`, která se stejně jako předchozí metoda, nachází ve třídě `Group`, na předané číslo postupně v cyklu aplikuje operaci posuvu o 1 bit směrem doleva, poté zjistí hodnotu nejvyššího bitu takto posunuté hodnoty pomocí logického součinu s hodnotou, která má na nejvýznamnějším bitu 1 a ostatních bitech 0. Pakliže takto metoda detekuje logickou 1, zkontroluje zdali se předaná hodnota vleze do registru či skupiny do které se ukládá, pokud ne, tak končí s chybou. Pokud je místo kde se hodnota ukládá dostatečně velké, tak se hodnota postupně ukládá bit Po bitu, od pozice kde začíná daná skupina či registr.

5.4 Implementace generátoru

Generátor je stěžejní komponentou tohoto programu, je implementován jako, třída která generuje zdrojové a hlavičkové soubory pro nastavení MCU, ale také provádí základní kontrolu nastavení periferií MCU uživatelem. V následujícím textu budou popsány některé významné metody této třídy.

5.4.1 Kontrola nastavení

Před generováním samotného zdrojového souboru je potřeba zkontrolovat jestli nějaké dvě aktivní periferie nesdílí stejné piny. Toto, mimo jiné, provádí metoda `checkErrors()`. Tato metoda si vyžádá od objektu `PeripheralsModel`, který reprezentuje periferie MCU, seznam ukazatelů na objekty periferií, které se budou generovat (resp. jsou aktivní). Tímto seznamem se pak postupně prochází a každá aktivní periferie se předá metodě `checkPinouts()`. Tato metoda postupně prochází piny které může periferie obsadit a kontroluje, jestli je jí pin obsazen, pokud ano vyžádá si od objektu pinu seznam všech dalších periferií, které vlastní tento pin. Tento seznam se projde a kontroluje se, zda i ostatní periferie nejsou aktivní. Pokud nějaká jiná aktivní je, vytvoří se nový prvek do seznamu chyb. Tento prvek obsahuje ukazatel na periferii, která sdílí pin a také text chyby. Tento seznam vrací nejen metoda `checkPinouts()`, ale také metoda `checkErrors()`. Tento seznam nakonec doputuje až do objektu `MainWindow`, kde se jeho obsah zpracuje.

Další kontrola, která probíhá nad uživatelem nastavenými hodnotami, je kontrola závislostí. Tato kontrola se provádí ve stejném cyklu jako je kontrola obsazení pinu, pro každou periferii je od instance třídy `PeripheralsModel` vyžádán seznam ukazatelů na všechny hodnoty dané periferie. Tento seznam se cyklicky prochází a na kontrolu jeho položek, které jsou typu list (seznam možností) nebo bit, se volá metoda `checkDependences()`. Tato metoda postupně načítá z objektu `Value` položky závislosti, jedna položka reprezentuje jednu závislost, která musí být splněna, aby daná hodnota mohla být nastavena. Tyto závislosti jsou dvojího druhu, jedna závislost se vztahuje k bitu a určuje, jak musí být tento bit nastaven, aby byla závislost splněna. Ukazatel na tento bit je obsažen v položce závislosti, metoda se tedy na bit podívá, zjistí která periferie jej nastavuje a podívá se jestli je aktivní. Pokud ano porovnává se s hodnotou bitu, pokud periferie není aktivní porovnává se s výchozí hodnotou bitu (předpokládá se, že bit nebude nastaven). Pokud se hodnota bitu neshoduje s hodnotou v položce závislosti vytvoří se položka do seznamu chyb. Podobný postup se aplikuje i na druhou možnost položky závislosti a to je taková závislost, která je určená registrem nebo skupinou a minimální a maximální hodnotou jakou daný prvek paměti musí nabývat, aby byla závislost splněna. Liší se pouze porovnávání mezních hodnot.

Pokud je kód zkontrolován a je detekována chyba, znázorní se uživateli. Ten pak může rozhodnout, jestli pokračovat a vygenerovat kód či ne. Pakliže zvolí první variantu zavolá se nejprve metoda generování hlavičkového souboru `generateHeaderFile()`. ta se chová tak, že nejprve pomocí instance objektu `QFile` otevře soubor, který je specifikovaný svou cestou, pro zápis. do tohoto souboru nejprve vloží dodatečný kód hlavičky, který je specifikován v XML souboru a načten do objektu `PeripheralsModel` a dále vygeneruje nápovědu k mikrokontroléru pro který se soubor generuje metodou `generateMcuInfoComment()`. Poté si vyžádá od objektu `PeripheralsModel` pomocí metody `getEnablePeripheralsOnly()` seznam aktivních periferií. Tento seznam cyklicky prochází a u každé periferie vygeneruje komentář se jménem periferie metodou `generatePeripheralComment()`, následně pomocí metody `getNextCode()` objektu `Peripheral` zjistí funkce periferie, a ty které jsou označeny ke generování vloží do hlavičky jejich deklaraci. Takto zpracuje každou periferii v seznamu a po jeho projití uzavře soubor.

5.4.2 Generátor zdrojového kódu pro konfiguraci periferií

Po vygenerování hlavičky se generuje soubor zdrojového kódu metodou `generateCodeFile()`, stejně jako u hlavičky je soubor, do kterého se podle předané cesty má generovat, otevřen pro zápis. Jako první se do souboru vloží řádek, který přilinkuje hlavičku, následně

se vyžádá pomocí metody `getEnablePeripheralsOnly()` od objektu `PeripheralsModel` seznam aktivních periferií, který se v cyklu projde. pro každou periferii se vygeneruje komentář s informací o nastavení hodnot periferie, toto zprostředkuje metoda `generateValuesComment()`. Následně se přečtou funkce které se mají vygenerovat a každá se předá metodě `fillData()` a ta doplní požadované hodnoty. aby se dal nastavit v předem vytvořených funkcích registr, skupina či bit, je potřeba na místo, kde se tomu má dít, vložit data. Tyto místa jsou označena speciálními značkami jako je vidět například zde na ukázkovém kódu.

```
void Port_A_Init(void)
{
    @#@PTADD@@@
    @#@PTAPE@@@
    @#@PTASE@@@
    @#@PTADS@@@
}
```

Metoda `fillData()` tuto funkci projde a uloží vše, co není značka, do řetězce, který se bude navracet, současně vyhledá nejprve počáteční značku ze které zjistí v jakém formátu se mají hodnoty uložit, následně si uloží vše co je mezi startující značkou a ukončující značkou, přečte ukončující značku a zjistí, jak se mají hodnoty do výsledného řetězce vložit (buď s přiřazením nebo jen hodnota). Jakmile je přečtena celá funkce její naplněná verze (viz. ukázka kódu níže) se vrátí a vloží do generovaného souboru.

```
void Port_A_Init(void)
{
    PTADD = 0x7;
    PTAPE = 0x38;
    PTASE = 0x7;
    PTADS = 0x0;
}
```

Zvláštní funkci pro generování hodnoty potřebují speciální skupiny, tyto skupiny nejsou ve hlavičkovém souboru pro daný MCU a proto nelze přiřazovat jejich hodnotu jako u ostatních paměťových prvků. Proto se nastavují registry těch bitů, které jsou ve skupině. Generování jejich kódu zařizuje metoda `generateSpecialGroup()`. Tato metoda pracuje tak, že projde speciální skupinou bit Po bitu a podívá se na rodičovský registr daného bitu. Nejprve zjistí jestli takovýto registr již má ve svém seznamu registrů ke generování, pokud ne, vytvoří novou položku v listu s ukazatelem na zkoumaný registr, položka také obsahuje vektory pro nastavení log. 0 a log. 1 (log. 0 se nastavuje pomocí bitového součinu výchozí hodnota vektoru je 0xFFFFFFFF pro log. 1 je naopak vektor roven 0). Pokud je daný registr již v seznamu nalezen, nová položka se nevytváří, zapisuje se do nalezené položky. Podle hodnoty bitu ve skupině se od rodičovského registru bitu vyžádá vektor pro nastavení dané hodnoty a tento vektor se pak podle příslušné operace přidá k vektoru pro patřičnou hodnotu daného registru v nalezené nebo vytvořené položce seznamu registrů ke generování. Jakmile jsou všechny bity takto zpracovány, už se jen vygenerují příslušné logické součty a součiny pro registry ze seznamu do výstupního řetězce.

Kapitola 6

Testování

Aplikace byla testováná již od počátku implementace, také s každou novou částí a po kompletní implementaci byla testována jako celek. Nejprve byla aplikace testována na fiktivním MCU, kde se podle požadavků měnily jeho komponenty a jejich definice. Nakonec byla aplikace otestována na požadovaném MCU. V následujících odstavcích bude podrobněji toto testování popsáno.

6.1 Testování v průběhu implementace

V průbehu implementace byly jednotlivé částí aplikace testovány samostatně a po zapojení do aplikace testovány i spolu s již funkčními moduly. Nejprve byly implementovány moduly datové části, pro jejich testování byly vytvořeny soubory s fiktivním MCU. U tohoto fiktivního MCU se vytvářely komponenty tak, aby se ověřily různé části implementovaných modulů. U těchto modulů se především testovalo:

- správné načítání souboru
- detekce chyb v souboru
- vyhledávání
- správné vytvoření struktury

Poté co se otestovaly samotné moduly datové části, propojily se tyto moduly mezi sebou tak, aby vytvořily již finální strukturu uchovávající data o nastavovaném MCU, započalo testování celé struktury. Tato struktura byla také testována na fiktivním MCU a převážně se testovala komunikace mezi moduly (resp. modely) a jejích správné hierarchické propojení. pro tyto účely byly vytvořeny funkce, které procházejí vytvořenou strukturu, kontrolují a vypisují informace o jednotlivých elementech této struktury. Například funkce která, prochází modelem paměti, vypisuje postupně všechny vytvořené registry, informace o skupinách v registru a vypsané bity registru. U bitů vypisuje jejich hodnotu a které hodnoty z modelu periferií tyto bity nastavují. Podobné funkce lze nalézt i u modelů paměti a modelu periferií MCU.

Po implementaci ostatních modulů aplikace byly postupně testovány jejich dílčí funkce, především však detekce mezních hodnot (konce polí, prázdné seznamy a podobně), tyto chyby si mezi sebou objekty dávají na vědomí pomocí výjimek.

6.2 Testování výsledné aplikace

Nejprve se u výsledné aplikace testovalo grafické rozhraní, především u komponent, které vykreslují náhled na rozložení pinů, a náhled na nastavované registry. U těchto komponent bylo mimo správného vykreslování otestována i náročnost na výkon počítače (na průměrném PC se překreslovaly komponenty plyule, na slabších strojích již bylo překreslování trhané). Dále byly otestovány funkce ukládání nastavení a chování aplikace pokud není soubor s nastavením dostupný. Posléze se testovalo zadávání nekorektních vstupů do polí ve stromu periferií, zde se testovaly tyto případy:

- Hodnota je neznámého formátu - tato chyba může nastat pokud se Při psaní hodnot uživatel překlikne nebo se splete a píše hodnotu slovně, byly testovány hodnoty zdánlivě správné, například 0x00OA (kde je místo jedné nuly písmeno O), ale také hodnoty naprosto nesmyslné ,například bylo zkoušeno zapsat hodnotu "čahoj".
- Hodnota je mimo rozsah - tato chyba nastává pokud uživatel špatně odhadne velikost hodnoty, ale může také vzniknout pouhou nepozorností a napsání jednoho řádu navíc. Testovanými hodnotami byly například přespřílišné hodnoty typu 0xFFFFFFFF které měly být uloženy do 4 bitů, nebo naopak byly zadávány záporné hodnoty.
- Hodnota nesplňuje závislost - takovéto chyby vznikají opomenutím závislosti mezi hodnotami nastavované periferie. Takovouto chybou je například nastavení adresy delší než 7 bitů u periferie IIC pokud není povolena rozšířená adresa.

Ve všech případech tyto chyby program detekoval, u prvních dvou zobrazil dialog s informací, že je hodnota špatně nastavena a ponechal původní hodnotu. U poslední možnosti chybu odhalil před generováním, kdy se závislosti ověřují, a tuto chybu vypsal do výpisu.

Následně se testování zaměřilo na generátor kódu a to převážně jeho výstup. pro toto testování se také využívalo fiktivního MCU, kde se ve funkcích jednotlivých periferií, zadávaly různé značky a porovnával se vygenerovaný kód s očekávaným výsledkem. Dále se do kódu vkládaly i špatné značky, nebo značky které se odkazují na neexistující paměťové prvky. V obou případech se testovalo zdali toto program správně detekuje a korektně se zachová (zobrazí chybu ve výpisu).

6.3 Nalezené problémy a jejich řešení

Při testování generátoru byl objeven problém s odsazováním generovaného obsahu. Kdy byl Po vygenerováni odsazen dost chaoticky a nepřehledně. Příklad takového odsazení je vidět zde:

```
void KBI_Init(void)
                    {
                        KBISC_KBIE = 0x0;
                        KBIES = 0x0;
                        KBIPE = 0x0;
                        KBISC_KBACK = 0x1;
                        KBISC_KBIE = 0x1;
                    }
```

Po zkoumání tohoto problému, bylo zjištěno, že Qt objekt sprostředkovávající čtení obsahu elemetů, Při značce `<![CDATA[ ]]>`, která ohraničuje sekci, která je ignorovaná parserem, do obsahu elementu přidá i odsazení z původního dokumentu (dokumentu popisující periferie MCU). Proto musela být do generátoru dodatečně implementována metoda, která tento jev eliminuje. Po použití této metody zdrojový kód vypadá tak, jak jej tvůrce dokumentu popisujícího periferie MCU zamýšlel. Výsledek aplikace metody na funkci výše je vidět na této ukázce:

```
void KBI_Init(void)
{
    KBISC_KBIE = 0x0;
    KBIES = 0x0;
    KBIPE = 0x0;
    KBISC_KBACK = 0x1;
    KBISC_KBIE = 0x1;
}
```

Kapitola 7

Závěr

V rámci předmětu Semestrální projekt byly nejprve analyzovány požadavky na aplikaci, její rozhraní a výstup, který by měla poskytovat. Spolu s touto analýzou byly vyhledány podobné aplikace a byly analyzovány jejich rozhraní a funkce které nabízejí. Toto nabídlo ucelený pohled na funkcionalitu a vzhled budoucí aplikace. Poté co bylo jasno co bude aplikace poskytovat, byl analyzován i zvolený mikrokontrolér. Přesto že byl MCU zadán, aplikace byla navrhována, tak aby se neomezovala pouze na jeden typ MCU, ale aby byla univerzální. Nakonec byla navrhnuta struktura souborů popisujících mikrokontroléry, podle potřeb aplikace a zadání.

V letním semestru byla aplikace a její soubory implementovány a to tak, že se nejprve pracovalo na popisu XML pomocí XSD a vytvořily se testovací XML soubory s popisem fiktivního MCU (na tomto fiktivním MCU se posléze testovala datová část aplikace). Poté se implementovala část aplikace, která se starala o načtení XML souborů s popisem částí MCU a jejich interpretaci ostatním modulům aplikace. Poté navazovalo vytvoření a napojení grafického uživatelského rozhraní a součástí, které se starají o zobrazení informací o MCU na již vytvořené části aplikace. Nakonec byl implementován samotný MCU, generátor zdrojových kódů s nastavením periferií a bylo provedeno testování aplikace.

Jelikož je aplikace rozsáhlá, nebyl prostor pro realizaci všech komponent, který by ještě více rozšiřovaly její možnosti. Jako možné rozšíření navrhuji vylepšení kontroly správnosti nastavení tak, aby bylo možno vyjádřit širší škálu omezení kladených na nastavení hodnoty či více hodnot periferie. Dalším rozšířením by mohla být samotná aplikace, nebo rozšíření stávající o prostředek pro generování souborů popisující mikrokontrolér, kde by jedna z částí mohla analyzovat hlavičku popisující rozdělení paměti pro daný MCU a tvořit podle ní soubor s modelem paměti. Jako další možnost rozšíření navrhuji integrovat aplikaci do prostředí Code::Blocks a vytvořit tak nástroj pro programování mikrokontrolérů. Mezi menší rozšíření bych zařadil i vytvoření formuláře nastavující prostředí aplikace, vylepšení zobrazení MCU o informaci o směru dat vývodu, nebo rozšíření aplikace o modul náhledu na generovaný kód.

Literatura

[1] Bos, B.: CSS current work & how to participate. W3C, may 2012, [Online; cit. 2012-5-8].
URL http://www.w3.org/Style/CSS/current-work

[2] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.: Extensible Markup Language (XML) 1.0. W3C, feb 1998, [Online; cit. 2012-5-8].
URL http://www.w3.org/TR/1998/REC-xml-19980210

[3] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; aj.: Extensible Markup Language (XML) 1.1 (Second Edition). W3C, nov 2008, [Online; cit. 2012-5-8].
URL http://www.w3.org/TR/2006/REC-xml11-20060816/

[4] Freescale: CodeWarrior for Microcontrollers (Eclipse IDE) - RS08/HCS08, ColdFire V1, Qorivva 56xx, PX Series, 56800/E DSC, Kinetis.
URL http://www.freescale.com/webapp/sps/site/prod_summary%.jsp?code=CW-MCU10&tid=CWH

[5] Freescale: MC9S08JM60, MC9S08JM32 Data Sheet. Freescale, jan 2009, [Online; cit. 2012-5-8].
URL http:
//freescale.com/files/microcontrollers/doc/data_s%heet/MC9S08JM60.pdf

[6] Gregorie, M.; Solter, N. A.; Klepler, S. J.: *Professional C++, Second Edition*. John Wiley & Sons, Inc., 2011, ISBN 978-0-470-93244-5.

[7] Heineman, G. T.; Pollice, G.; Selkow, S.: *Algorithms in a Nutshell*. O'Reilly Media, 2008, ISBN 059651624X.

[8] Jasmin Blanchette and Mark Summerfield: *C++ GUI Programming with Qt 4 (2nd Edition)*. Prentice Hall, 2008, ISBN 0132354160.

[9] Knoll, L.: Qt 5 Alpha. Nokia Corporation, apr 2012, [Online; cit. 2012-5-8].
URL http://labs.qt.nokia.com/2012/04/03/qt-5-alpha/

[10] Lie, H. W.: Cascading HTML style sheets – a proposal. W3C, oct 1994, [Online; cit. 2012-5-8].
URL http://www.w3.org/People/howcome/p/cascade.html

[11] Lubbers, P.; Albers, B.; Salim, F.: *Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development*. Apress, 2010, ISBN 1430227907.

[12] Meyer, E. A.: *CSS: The Definitive Guide*. O'Reilly Media, 2006, ISBN 0596527330.

[13] Nokia Corporation: Model/View Programming. Nokia Corporation, 2011, [Online; cit. 2012-5-8].
URL http://qt-project.org/doc/qt-4.8/model-view-programmin%g.html

[14] Nokia Corporation: QStyledItemDelegate Class Reference. Nokia Corporation, 2011, [Online; cit. 2012-5-8].
URL http://doc.qt.nokia.com/4.7-snapshot/qstyleditemdelega%te.html

[15] Nokia Corporation: Qt 4.7: Supported Platforms. Nokia Corporation, 2011, [Online; cit. 2012-5-8].
URL http://doc.qt.nokia.com/4.7/supported-platforms.html

[16] Nokia Corporation: Supported HTML Subset. Nokia Corporation, 2011, [Online; cit. 2012-5-8].
URL http://qt-project.org/doc/qt-4.8/richtext-html-subset.%html

[17] Ray, E. T.: *Learning XML, Second Edition*. O'Reilly Media, 2003, ISBN 0596004206.

[18] Stroustrup, B.: *The C++ Programming Language Third Edition*. Addison-Wesley Publishing Company, 1997, ISBN 0201889544.

[19] Stroustrup, B.: C++11 - the recently approved new ISO C++ standard. AT&T Labs, feb 2012, [Online; cit. 2012-5-8]].
URL http://www2.research.att.com/~bs/C++0xFAQ.html

[20] Thompson, H. S.; Beech, D.; Maloney, M.; aj.: XML Schema Part 1: Structures Second Edition. W3C, oct 2004, [Online; cit. 2012-5-8].
URL http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

[21] Thompson, H. S.; Mendelsohn, N.; Maloney, M.; aj.: W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. W3C, apr 2012, [Online; cit. 2012-5-8].
URL http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/

Přílohy

Seznam příloh

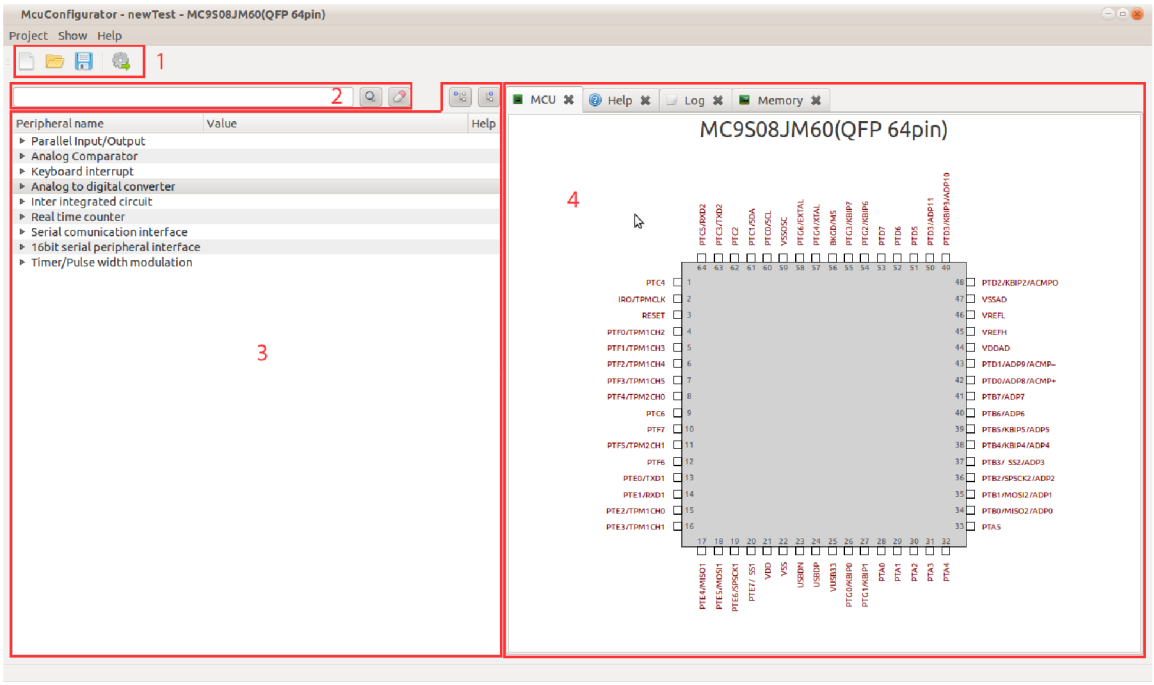
Příloha A

Manual

A.1 Popis okna a tlačítek akcí

A.1.1 Okno aplikace a jeho rozdělení

V tomto odstavci bude popsáno hlavní okno aplikace a její rozdělení. Budou zde uvedeny názvy komponent na které se bude v dalším textu odkazovat. na obrázku A.1 je znázorněno hlavní okno aplikace spolu s označením jednotlivých částí aplikace.



Obrázek A.1: Pohled na hlavní okno aplikace s rozdělením na části.

V části pod číslem 1 se nachází **panel s tlačítky akcí**, tyto tlačítka slouží k ovládání části aplikace, která se stará o načítání, ukládání projektu a generování samotných zdrojových kódu s nastavením pro jednotlivé periferie. Číslo 2 označuje **pole pro vyhledávání** a tlačítka pomocí kterých se vyhledává ve stromu periferií, prvním tlačítkem se spouští vyhledávání (**tlačítko vyhledávání**), druhé maže vyhledáváním označené uzly ve stromu

periferií (**tlačítko mazání vyhledávání**). pod číslem 3 je část, která se stará o zobrazení periferií a vyhledávání v nich, tato komponenta se nazývá **strom periferií**. Část 4 pod sebou pojímá komponenty, které se starají o jiný náhled na nastavovaný mikrokontrolér a které vypisují informace pro uživatele. Tyto komponenty jsou dále v textu popisovány jako **záložky**.

A.1.2 Tlačítka akcí

Na obrázku A.2 je zobrazen panel tlačítek akcí, funkčnost těchto tlačítek bude postupně popsána v následujícím odstavci.

Obrázek A.2: Výřez okna panelu tlačítek akcí s očíslovanými tlačítky.

Tlačítko 1 slouží k vytvoření nového projektu. Pokud je již nějaký projekt otevřen a není uložen, zobrazí se Po stisku tohoto tlačítka dialog, který na toto upozorní a pakliže si to uživatel zvolí, uloží projekt. Samozřejmě jde pokračovat bez uložení, s tím, že jsou neuložená data nenávratně ztracena. Pokud bude tedy uživatel pokračovat, zobrazí se dialog kde lze vybrat pro který MCU se nový projekt vytvoří (viz. další kapitola), Po zvolení požadovaného MCU je program připraven na práci.

Tlačítko 2 zprostředkovává načtení dříve vytvořeného projektu. Pokud je již spuštěný projekt neuložen, zobrazí se stejný dialog jako u předchozího tlačítka. Poté co je dialog uživatelem obsloužen, zobrazí se další dialog, kde se Po uživateli žádá o specifikaci cesty k souboru s uloženým projektem. Tento projekt je posléze načten.

Tlačítko 3 slouží k uložení stávajícího projektu. Pokud projekt dosud nebyl uložen, zobrazí se dialog, který Po uživateli žádá určení místa a jména souboru do kterého se bude projekt ukládat. Po tomto uložení, se podle jména souboru projekt pojmenuje. Pokud se toto tlačítko stiskne kdykoliv Po prvním uložení (platí i po načtení již uloženého projektu), tak se již žádný dialog nezobrazuje a ukládá se do souboru, který byl předtím specifikován.

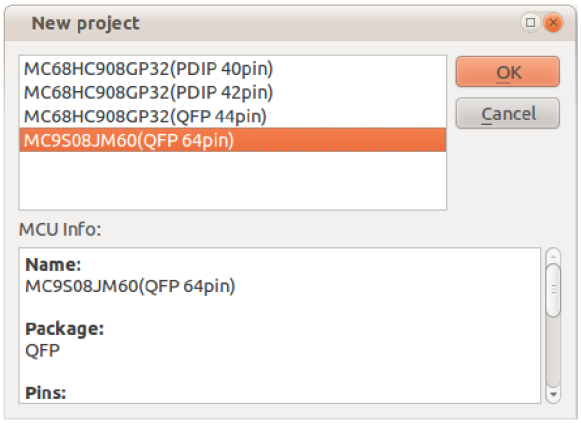
Tlačítko 4 slouží ke zkontrolování projektu na chyby a posléze k vygenerování samotných zdrojových souborů s funkcemi pro dané periferie. Po stisku tohoto tlačítka bude uživatel vyzván k uložení stávajícího projektu (pokud tak již dříve neučinil), následně bude uživatel vyzván k určení složky, kde se budou zdrojové soubory generovat. Posléze se zkontrolují chyby, pokud budou nějaké nalezeny má uživatel možnost buď opravit chyby, nebo je ignorovat a vygenerovat soubory.

A.2 Popis vytvoření projektu

V této kapitole bude ukázáno jak se s programem pracuje a to formou příkladu, jak otevřít nový projekt, nastavit komponentu, uložit projekt a vygenerovat zdrojové kódy pro daný MCU.

A.2.1 Otevření nového projektu

Po spuštění programu, otevřeme nový projekt pomocí prvního tlačítka na panelu tlačítek akcí, Po stisknutí tlačítka se objeví dialog, kde si vybereme MCU (viz. obrázek A.3), výběr potvrdíme bud' dvojím poklepáním levým tlačítkem myši na daný MCU, nebo kliknutím na tlačítko OK. Pokud nebude zvolen MCU a klikne se na tlačítko OK, bude vybrán první MCU ze seznamu. Po výběru se dialog uzavře a program načte potřebné soubory, Po tomto načtení je připraven ke konfiguraci periferií.



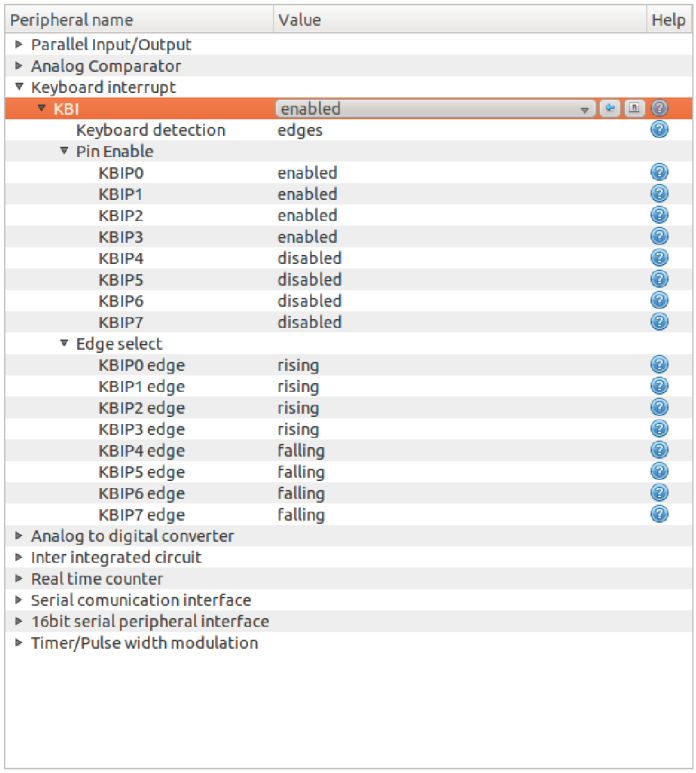
Obrázek A.3: Dialog s výběrem MCU.

A.2.2 Nastavení periferií

Při konfiguraci periferií se převážně pracuje se stromem periferií. V tomto příkladě se bude nastavovat přerušení klávesnice. Nejprve ve stromu periferií najdeme příslušnou periferii, která se nachází pod položkou *Keyboard interrupt* > *KBI*. Jednotlivé uzly stromu se dají rozevřít pomocí šipky u jejich jména, nebo Po dvojkliku na uzel. Pokud chceme periferií najít rychle, lze použít vyhledávání, kde do vyhledávacího pole zapíšeme název periferie, nebo část jejího názvu a potvrdíme klávesou enter, nebo kliknutím na tlačítko vyhledávání. Nalezené výsledky se rozevřou ve stromu periferií. Abychom mohli začít nastavovat periferii, je potřeba ji povolit. Povolení se provede poklikáním na sloupec hodnota dané periferie a nastavením roletového menu na hodnotu enable, poté nastavíme periferii, tak jak je na obrázku A.4. Nastavení jednotlivých hodnot se nastavuje podobně jako se povoluje periferie, dvojím poklikem. Po pokliku se spolu s komponentou pro nastavení zobrazí i tlačítko s šipkou (viz. obrázek A.5), toto tlačítko nastavuje původní hodnoty. aby se funkce pro nastavení vygenerovaly je třeba je nastavit na generování, toto se provádí podobně jako u povolení periferie u sloupce hodnota a řádku periferie. Po kliknutí na tlačítko vypínače (nachází se u uzlu stromu znázorňující periferii), se zobrazí dialog pro výběr funkcí ke generování. Zaznačíme všechny funkce , jak je to mu na obrázku A.6. Poté co je vše nastaveno, se může generovat.

A.2.3 Vygenerování zdrojových kódů

Předtím než se bude generovat kód, je třeba projekt uložit, toto se provádí třetím tlačítkem na panelu tlačítek akcí. Program vyvolá dialog pro uložení, zde označíme složku a soubor



Obrázek A.4: Náhled na strom nastavení, s příkladem nastavení KBI.

Obrázek A.5: Nastavení periferie, první tlačítko nastavuje původní hodnotu, druhé spouští dialog pro výběr generované funkce.

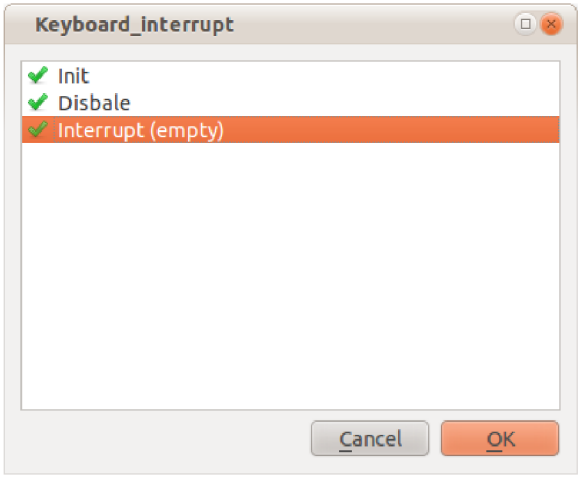
do kterého chceme projekt uložit. Poté klikneme na čtvrté tlačítko panelu tlačítek akcí, tímto vyvoláme dialog, kde se specifikuje kam se mají vygenerované soubory uložit. Poté program zkontroluje veškeré nastavení a pokud je vše v pořádku, tak na specifikované místo vygeneruje soubory s nastavením.

A.3 Popis komponent

Níže popsané komponenty se nachází v části 4 a poskytují jiný pohled na nastavované části MCU než podává komponenta stromu periferií. Dále také zobrazují informace, které přibližují funkci komponent a informace o stavu programu.

A.3.1 Komponenta náhledu na MCU

Komponenta náhledu na MCU se skrývá pod záložkou **MCU**. Tato komponenta, pokud je načten projekt, zobrazuje nákres pouzdra mikrokontroléru spolu s jeho vývody. Každý vývod obsahuje informaci o jeho čísle, které koresponduje s číslem v technické dokumentaci k MCU a dále také jméno daného vývodu, které se skládá ze zkratek periferií, které vývod



Obrázek A.6: Dialog s výběrem funkcí pro generování.

využívají. na obrázku A.7 je vidět tato komponenta vykreslující pouzdro mikrokontroléru MC9S08JM60.

Mimo znázornění MCU a jeho vývodů, komponenta slouží k zobrazení vývodů které využívá daná periferie, která je zvolena ve stromu periferií. Tyto vývody, pokud periferie nějaké používá, jsou pak znázorněny žlutě. Dále komponenta zprostředkovává zobrazení periferií, které daný pin využívají. A to v dialogu, který zobrazuje jednak všechny periferie které vývod používají, ale také zda jsou aktivní či ne a po kliknutí levým tlačítkem myši na danou periferii ji zobrazí ve stromu periferií. Tento dialog zobrazíte kliknutím levým tlačítkem myši na daný vývod, tento dialog se zobrazí pokud vývod obsazuje alespoň jedna periferie.

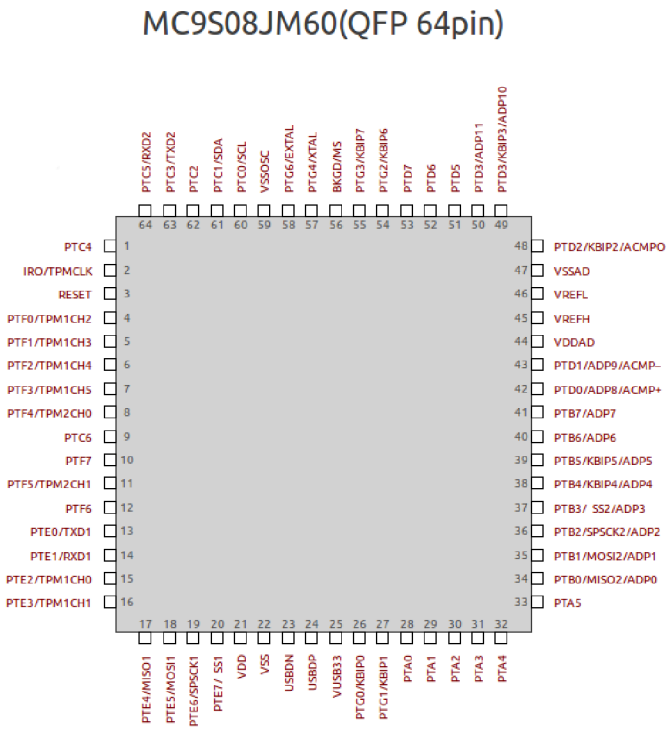
A.3.2 Komponenta náhledu na nastavované registry

Tato komponenta (nachází se pod záložkou **Memory**) zobrazuje registry ve kterých hodnota, zvolená ve stromu nastavení, mění bity. jak je vidět na obrázku A.8 nad každým registrem, který tato hodnota pojímá, je vypsáno jméno registru, dále jsou pod jménem zobrazeny bity. U každého bitu je vypsána informace o pořadí v registru, jméně a hodnotě kterou bit v danou chvíli nabývá. Dále pomocí barvy pozadí jednotlivých bitů dává komponenta najevo, které bity hodnota mění (jsou zobrazeny žlutě), které bity se nenastavují (jsou zobrazeny šedě) a na konec i bity které se nastavují, ale nepatří dané hodnotě (ty jsou bez pozadí).

Další možností této komponenty je zobrazit ve stromu periferií i hodnoty, které nastavují okolní bity registru (resp. ty bity, které nějaká hodnota nastavuje). Toto zobrazení se provádí kliknutím levým tlačítkem myši na plochu zobrazovaného bitu. Pokud lze zobrazit hodnotu která tento bit nastavuje, Po najetí kurzorem na bit se změní tvar kurzoru. Po kliknutí se zobrazí příslušná hodnota ve stromu periferií.

A.3.3 Komponenta log

Komponenta log slouží k zobrazení různých záznamů týkajících se programu, tato komponenta je zobrazena na obrázku A.9 a nachází se pod záložkou **Log**. V prvé řadě jsou to zá-

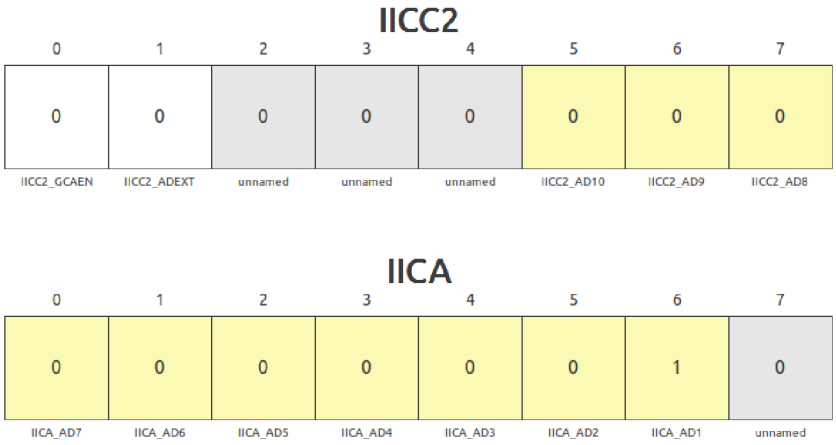


Obrázek A.7: Výsledek kreslení komponenty zobrazení MCU.

znamy informativního charakteru, ty jsou zobrazeny zeleně a informují uživatele co zrovna program provádí. Dále jsou to záznamy varující, tyto jsou červené barvy a upozorňují na chyby, které se Při běhu programu vyskytly, nebo na chyby (či potenciální chyby), které uživatel způsobil Při nastavování periferií. Posledním druhem záznamu je záznam připomínky, který je barvy žluté.

A.3.4 Komponenta nápovědy

Nápověda k periferiím a hodnotám je v tomto programu řešena pomocí komponenty pod záložkou nápověda, která se skrývá pod záložkou **Help**. V této komponentě se zobrazuje nápověda, poté co uživatel klikne na ikonu nápovědy ve stromu periferií (tuto ikonu lze vidět na obrázku A.5). Po kliknutí na tuto ikonu se záložka nápovědy přepne do popředí a zobrazí informace, které se vztahují k dané hodnotě či periferii. Pokud se zobrazuje nápověda k periferii, vypíše se s odsazením i nápověda ke všem hodnotám patřící k této periferii, jak je vidět na obrázku A.10.



Obrázek A.8: Výsledek kreslení komponenty zobrazení nastavovaného registru.

Loading list of mcu from mcu_xml/mcuList.xml.
Loading settings from settings.xml.
Loading pinouts model: pinouts/MC9S08JM60_pinouts_QFP_64pin.xml.
Loading memory model: memory/MC9S08JM60_memory.xml.
Loading peripheral model: peripherals/MC9S08JM60_peripherals_QFP_64pin.xml.
Error: Value [Clock prescaler] in peripheral [Real_time_counter] must be set at value [1] or higher if you will use value [disabled] in peripheral [RTC].
Error: Value [Master/Slave select] in peripheral [Serial_peripheral_interface_1] must be set at value [1] if you will use value [enabled] in peripheral [SPI1].

Obrázek A.9: Náhled na výsledek komponenty vypisující záznamy aplikace.

Analog_to_digital_converter

The ADC can perform an analog-to-digital conversion on any of the software selectable channels. In 12-bit and 10-bit mode, the selected channel voltage is converted by a successive approximation algorithm into a 12-bit digital result. In 8-bit mode, the selected channel voltage is converted by a successive approximation algorithm into a 9-bit digital result. The ADC module has the capability of automatically comparing the result of a conversion with the contents of its compare registers.

Peripheral own this functions:

```
void ADC_Init(unsigned char channel);
void ADC_Disable(void);
void ADC_Get_Value(short int * value);
interrupt VectorNumber_Vadc void ADC_Interrupt(void);
```

Interrupt

This setting enables conversion complete interrupts.

Conversion trigger

Setting selects the type of trigger used for initiating a conversion. Two types of triggers are selectable: software trigger and hardware trigger. When software trigger is selected, a conversion is initiated following a Init() function (or write to ADCSC1). When hardware trigger is selected, a conversion is initiated following the assertion of the ADHWT input.

Conversion

Setting enable continuous conversion (or only one conversion). Start of conversion depends on conversion trigger.

Conversion mode

Setting is used to select between 12-, 10-, or 8-bit operation.

Obrázek A.10: Náhled na výsledek komponenty vypisující nápovědu.