VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# DESIGNER PRO QDEVKIT

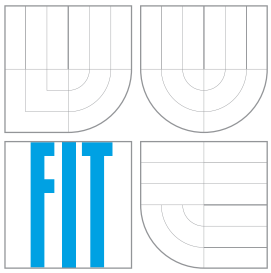
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR ŠIMEK

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# DESIGNER PRO QDEVKIT

QDEVKIT PROJECT DESIGNER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR ŠIMEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZDENĚK VAŠÍČEK

BRNO 2012

ZDE
VLOŽIT
ZADÁNÍ

# Abstrakt

Tato práce si klade za úkol seznámit čtenáře s již existujícími komerčními aplikacemi a ovládáním FITkitu za pomoci aplikace QDevKit. Hlavním cílem práce bylo navrhnout a implementovat modul pro aplikaci QDevKit, který usnadní návrh a vývoj aplikací pro platformu FITkit.



# Abstract

The purpose of this thesis is to introduce a reader with existed commercial applications and control the FITkit using QDevKit. The main goal of the thesis is proposed and implemeted modul for application QDevKit which makes easy proposal and progress applications for platform FITkit.



# Klíčová slova

Designer, grafické uživatelské rozhraní, QDevKit, FITkit, Qt, PythonQt, VHDL, C++.

# Keywords

Designer, graphical user interface, QDevKit, FITkit, Qt, PythonQt, VHDL, C++.


# Citace

Petr Šimek: Designer pro QDevKit, diplomová práce, Brno, FIT VUT v Brně, 2012

# Designer pro QDevKit

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Zdeňka Vašíčka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Šimek
23.5.2012

## Poděkování

Chtěl bych poděkovat Ing. Vašíčkovi za vedení a možnost zpracovat tak zajímavé zadání diplomové práce.

# Obsah

# Kapitola 1

# Úvod

Každým rokem se zlepšuje technologická výroba obvodů na křemíkovém substrátu a tím dochází k stále většímu integrování obvodových součástek do jednoho pouzdra a vzniku nových rozsáhlejších a složitějších FPGA.

Programovatelný hardware se vyučuje na Fakultě informačních technologií VUT v Brně a výsledkem její činnosti je platforma FITkit, na které je možno navrhovat a prakticky realizovat nejen softwarové, ale také hardwarové projekty či dokonce celé aplikace.

Doposud byly aplikace psány v textové formě a jak víme, čím složitější obvod je, tím méně přehledný kód může být a snáze lze udělat chybu. Proto je žádoucí vytvořit SW s dynamickou správou databáze komponent, který usnadní vývoj a návrh aplikací začínajícím studentům.

Cílem diplomové práce je vytvořit modul pro aplikaci QDevKit na základě analýzy komerčních nástrojů usnadňujících vývoj a návrh aplikací s využitím grafického prostředí a způsobu, jakým je dána reprezentace komponent jednotlivých nástrojů. Dále navrhnout modul pro aplikaci QDevKit, pomocí kterého bude možné vygenerovat šablonu aplikace pro FPGA a MCU na základě interakce aplikace s uživatelem. Výsledný návrh implementovat za pomocí knihovny Qt, zakomponovat modul do aplikace QDevKit a ověřit funkčnost modulu na příkladu.

Práce navazuje na semestrální projekt, který se zabýval analýzou již existujících komerčních nástrojů, kde byla obzvláště věnována pozornost způsobu popisu komponent, viz kapitola druhá. V kapitole třetí je proveden rozbor výukové platformy FITkit s jejími komponentami, pro kterou bude především modul navržen. Čtvrtá kapitola popisuje použité nástroje při realizaci aplikace. V páté kapitole prezentuji již konkrétní návrh a specifikaci grafického prostředí a popisu komponent. Následující šestá kapitola podrobně líčí implementaci designeru. V předposlední sedmé kapitole je provedeno ověření funkčnosti hotového řešení modulu. Shrnutí veškerých získáných vědomostí o návrhu, realizaci a ověření funkčnosti bude provedeno v osmé, závěrečné kapitole.

# Kapitola 2

# Existující komerční nástroje

Cílem této části je provést rozbor tak, abychom si mohli udělat představu, co je vše zapotřebí pro návrh modulu. Především se zaměříme na prostředí pro tvorbu schématu a na způsob reprezentace komponent či celé architektury FPGA. Klíčové pro návrh modulu je zjistit, co vše je potřeba o dané komponentě uchovávat, jakou reprezentaci popisu jednotlivé nástroje používají a jakým způsobem řeší kreslení schémat. Pro reprezentaci popisu potřebujeme zjistit, jakým způsobem je definováno rozhraní komponenty, to znamená z jakých generických parametrů se skládá a jak namapovat vstupní a výstupní signály komponenty. Následně zjistit popis architektury komponenty nebo-li její funkci a chování.

## 2.1 Altium Designer

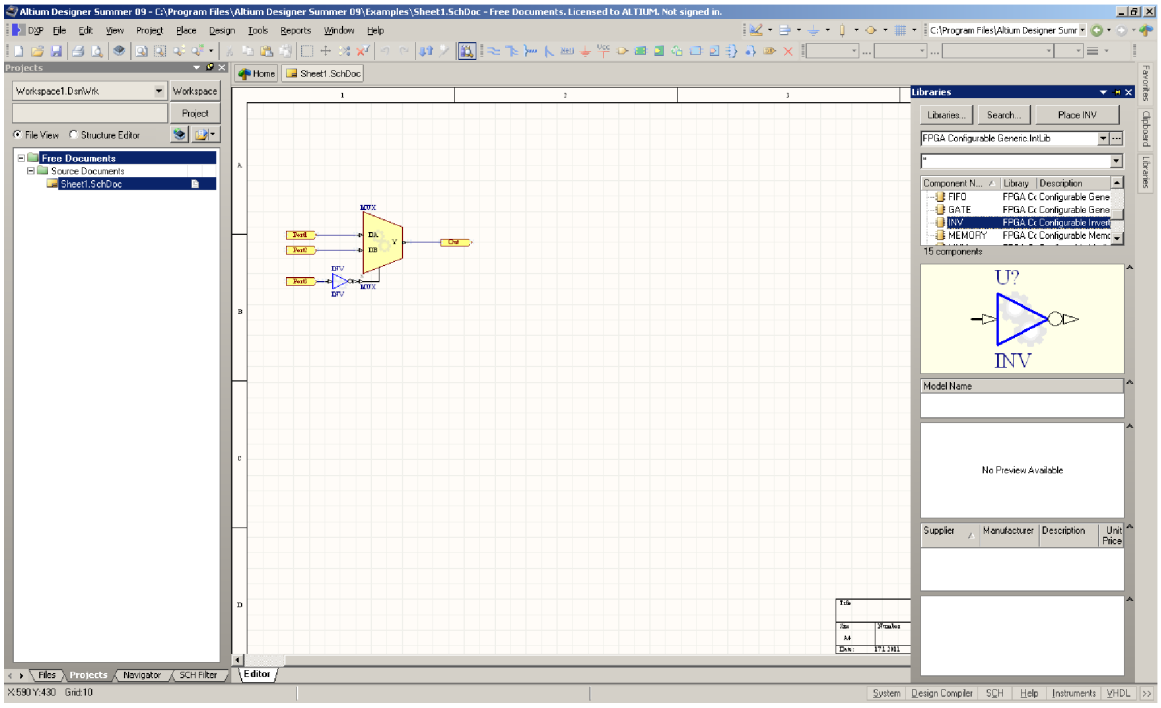
Altium Designer je produkt australské společnosti Altium Ltd., který navazuje na předchozí návrhový systém plošných spojů Protel [4]. Altium Designer si vzal za cíl integrovat většinu činností spojených s návrhem elektroniky do jednoho programu a umožnit tak lepší provázanost dat jednotlivých částí návrhu bez nutnosti používat různá uživatelská rozhraní [11].

### Grafický editor

Ukázka, jak se vytváří projekt v nástroji Altium Designer, je znázorněno na obrázku 2.1. Jak můžeme vidět, program se skládá ze tří hlavních částí. Úplně vlevo je umístěn prohlížeč pracovního prostoru, který se ovládá pomocí několika záložek. Tyto záložky mají na starosti hlavní funkčnost aplikace. Pomocí záložky Files můžeme otevřít naposledy uložené projekty nebo vytvořit projekty nové. Nás především zajímá tvorba projektu FPGA. Po vytvoření prázdného projektu přepneme na záložku Projects, který obsahuje přehlednou stromovou strukturu vytvořených souborů, potřebných pro správnou funkci projektu. Kliknutím pravého tlačítka myši na název projektu přidáme novou položku, a to její schéma.

Uprostřed obrazovky se zobrazí kreslící plátno s pomocnou mřížkou, díky které snáze kreslíme spoje a lehce manipulujeme s komponentami.

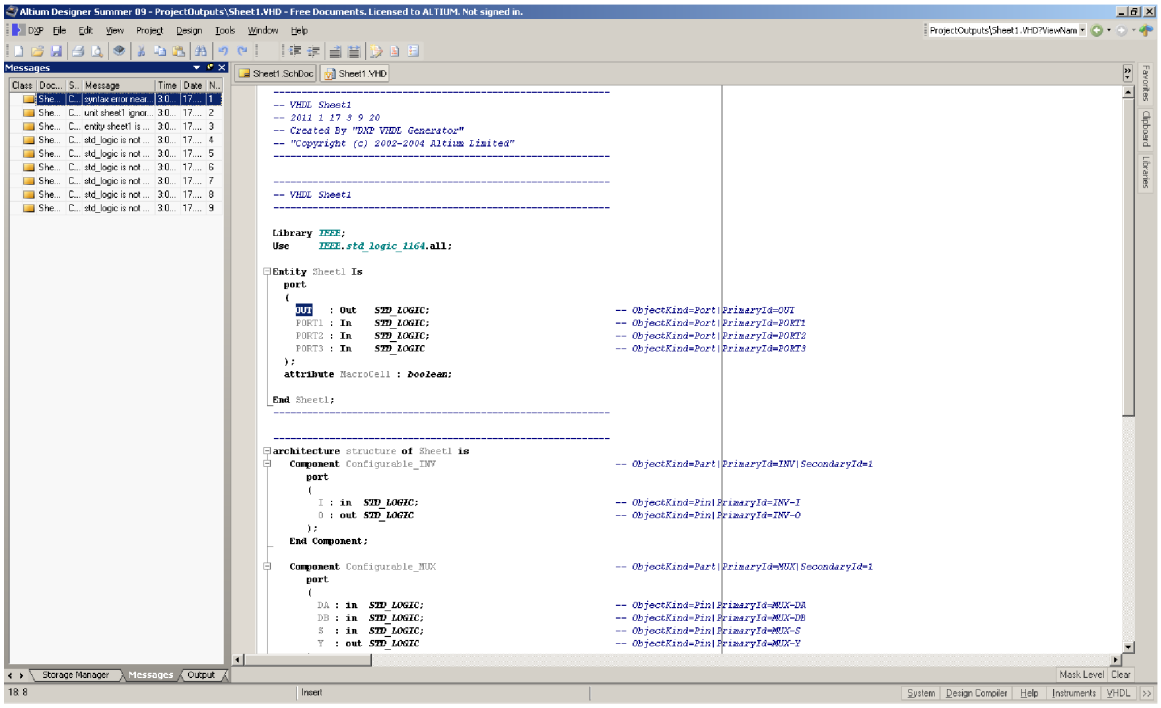
V pravém okraji obrazovky se nachází tlačítko Libraries. Po kliknutí vyjede knihovna komponent, kde jsou jednotlivé komponenty rozděleny do kategorií a jejich soupiska vložena do comboboxu. Volbou kategorie zobrazíme příslušné komponenty do tabulky a kliknutím na jednu z nich dojde k zobrazení schématické značky pod tuto tabulku. Komponentu můžeme také vyhledat za pomoci tlačítka Search. Komponentu umístíme na kreslící plátno uchopením a táhnutím levého tlačítka směrem na kreslící plátno. Kreslení propojení a vložení



Obrázek 2.1: Screenshot aplikace Altium Designer - knihovna komponent.

dalších potřebných součástek provedeme buď za pomoci panelu nad kreslícím plátnem nebo kliknutím pravého tlačítka na kreslící plátno a volbou Place. Tento nástroj má oddělené kreslení vodičů a sběrnic. Vodiče se zalamují do pravého úhlu a obsahují vždy počáteční a koncový bod. Proto, aby bylo možné komponenty propojovat, musí každý vstup komponenty obsahovat seznam vývodů, na které je připojen. Kliknutím pravého tlačítka myši na komponentu a výběrem položky Properties můžeme komponenty konfigurovat. Především se jedná o generické parametry nebo parametry datové šířky. Používáme-li při vytváření obvodů v tomto prostředí nějakou část velmi často, můžeme si usnadnit budoucí návrhy tím, že z obvodu vytvoříme vlastní komponentu. Kliknutím pravého tlačítka na název projektu a přidáním nové položky jako schéma pro knihovnu komponent, přiřadíme nakreslenému obvodu schématickou značku. Poté jen stačí kliknutím pravého tlačítka na projekt vybrat položku compile. Následně dojde k vygenerování VHDL kódu. Při chybě se v oblasti pracovního prostoru a aktivní záložce Messages zobrazí výpis chyb, viz obrázek 2.2. Dvojklikem na chybu dojde přímo ve schématu k zobrazení příslušné chyby.

Až na zbytečně komplikovené ovládání hlavních funkcí programu, pomocí záložek v oblastí pracovního prostoru, lze nástroj považovat za snadno pochopitelný. Pokud uživatel již dříve pracoval s CAD systémy, rychle se zorientuje, protože obsahuje klasické prvky kreslení spojů a práci s komponentami. Databáze komponent obsahuje v základu velké množství komponent, které lze snadno vyhledat, buď už za použití rychlého vyhledávání nebo přímým výběrem ze soupisky. Náhled označené komponenty dělá návrh snadnější a rychlejší, protože v některých případech nevíme, jakou komponentu vložit díky podobnosti jejich názvů.




Obrázek 2.2: Screenshot aplikace Altium Designer - vygenerované VHDL.

### Popis platformy

Společnost Altium nabízí vývojové kity s různým hardwarem, proto je potřeba před syntézou aplikace zvolit příslušný typ platformy. Platforma je popsána v souboru `*.Constraint`. V tomto souboru jde ve zkratce o přiřazení příslušných signálů jednotlivým pinům použitého FPGA . Altium designer obsahuje speciální formát popisu platformy.

Jednotlivé řádky začínající znakem ; značí komentáře, jako na příklad hlavička popisu:

```
;.....................................................................
;Constraints File
; Device : Xilinx Virtex II XC2V1000-4FG456C
; Board : NanoBoard NB1 Revision-6 with Plug-In Daughter Board
; Project : Any
;
; Created 7-May-2004
; Altium Limited
;.....................................................................
```

Zbývající řádky začínají klíčovým slovem `Record`, které nabývají hodnoty `Constraint` nebo `FileHeader`. V záznamu následují další informativní položky, které jsou odděleny znakem |. Následující položkou záznamu je `TargetKind`, který určuje, o jaký cílový druh záznamu se jedná a může nabývat hodnot:

- `Part` - popisovaná část
- `PCB` - druh platformy
- `Port` - signál
- `Net`

Předposlední položka `TargetId` uchovává název záznamu a jedná-li se o signál, tak i jeho datovou šířku. Poslední položka `FPGA PINNUM` udává výčet pinů oddělených čárkou, které odpovídají danému signálu. V popisu jsou uvedeny jako první záznamy:

```
Record=FileHeader | Id=DXP Constraints v1.0
;.....................................................................

;.....................................................................
Record=Constraint | TargetKind=PCB | TargetId=NanoBoard NB1 Revision-6
Record=Constraint | TargetKind=Part | TargetId=XC2V1000-4FG456C
```

Určují, kdo vytvořil popis, o jakou platformu se jedná a jakou část budeme popisovat. Dalším příklad záznamu je přiřazení signálu a pinu:

```
Record=Constraint | TargetKind=Port | TargetId=VGA_R[1..0] | FPGA_PINNUM=L4,K3.
```

Platforma popsána tímto formátem používá velmi malý počet klíčových slov. Díky tomu působí přehledně a pochopitelně. I když je potřeba použít speciální parser, nebude náročný díky malému počtu klíčových slov.

### Popis komponenty

Nástroj nabízí dynamickou knihovnu komponent. Uživatel si může za pomoci designeru přidat komponentu vlastní nebo k tomu postačuje vytvořit popis ve speciálním formátu, uvedeným níže, a následně jej uložit jako soubor `*.OpenBusComp`.

Popis komponenty začíná klíčovým slovem `object`, které blíže specifikuje třídu zařazení komponenty. Popis komponenty končí klíčovým slovem `end`. Hlavičku popisu komponenty tvoří klíčová slova:

- `Kind` - druh komponenty
- `LongName` - dlouhý název komponenty
- `ConfiguratorName` - konfigurační název komponenty
- `Description.Strings` - stručný popis komponenty

Příklad hlavičky:

```
object WB_SPI: TOpenBusLibraryComponent
       Kind = obckPeripheral
       LongName = 'SPI'
       ConfiguratorName = 'WB_SPI'
       Description.Strings = (
           'This SPI Master Controller enables a host processor to communica' +
           'te with multiple slave SPI peripherals. Enable the MODE pin to b' +
           'e able to access SPI peripherals located on the NanoBoard, such ' +
           'as the audio DAC and serial Flash RAM devices.')
```

Za hlavičkou se nachází již fyzický popis, který obsahuje pouze informace o rozhraní komponenty a rozšiřující informace o schématické značce. Rozhraní definují vstupy a výstupy. Ty mohou být tvořeny nezávislými signály začínající klíčovým slovem `Pins = <` a ukončeny `>` nebo nezávislé signály, které tvoří skupinu a začínají klíčovým slovem `PinGroups = <` a

končí znakem `>`. Uvnitř této dvojice klíčových slov se nachází jednotlivé signály jako položky začínající slovem `item` a uzavřeny slovem `end`. U signálu potřebujeme uchovat:

- `Name` - název
- `Direction` - směr (vstup, výstup, obojí)
- `HarnessEntry` - na který signál z platfromy jej namapujeme
- `Order` - pořadí ve schématické značce

Příklad:

```
Pins= <
    item
         Name = 'CLK_I'
         PinType = ptClock
         Direction = eElectricInput
         Order = 0
    end
    item
         Name = 'RST_I'
         PinType = ptReset
         Direction = eElectricInput
         Order = 1
    end>
PinGroups = <
    item
         HarnessType = 'SPI_W_NB'
         Order = 2
         Pins = <
                  item
                       Name = 'SPI_DOUT'
                       Direction = eElectricOutput
                       HarnessEntry = 'SPI_DOUT'
                       Order = 0
                  end
                  item
                       Name = 'SPI_CS'
                       Direction = eElectricOutput
                       HarnessEntry = 'SPI_CS'
                       Order = 3
                  end>
         PinGroups = <>
    end>
```

Poslední možností popisu jsou signály s větší datovou šířkou. Ty začínají klíčovým slovem `Ports = <`, končí znakem `end>` a udávají počet výstupních adresových a datových signálů. Ukázka:

```
Ports= <
     item
          Number = 0
          AddrWidth = 3
          DataWidth = 0
     end>
```

Poslední položkou rozhraní jsou generické parametry začínající klíčovým slovem `Parameters = <` a končící znakem `end>`. Příklad:

```
Parameters= <
          item
               Name = 'ChildModel'
               Value = 'WB_SPI32'
          end
          item
               Name = 'Published'
               Value = '26 Sep 2008'
          end>
```

Celý popis uzavírají možné informace o schématické značce komponenty, které obsahují data o obrázku a použitou paletu barev. Příklad:
`Icon.Data = 04000000`
`Picture.Data = {}`
`PalettePicture.Data = {}`

Popis komponenty tohoto nástroje je přehledný i za použití velkého množství klíčových slov. Strukturování položek za pomoci znaků <> dělá popis přehledným a snáze pochopitelným. Nevýhoda spočívá ve větším počtu klíčových slov a potřeby vlastního parseru, jelikož se jedná o speciální formát popisu komponenty.

## 2.2 Xilinx EDK

Xilinx EDK je produkt americké společnosti Xilinx, která je jedním z největších světových výrobců logických obvodů [24]. Xilinx EDK umožňuje tvorbu software a hardware aplikací, založených na procesoru MicroBlaze a PowerPC. Za pomoci grafického editoru a dalších nástrojů je možno aplikaci navrhnout, přeložit a přímo nahrát na jejich kit.

### Grafický editor

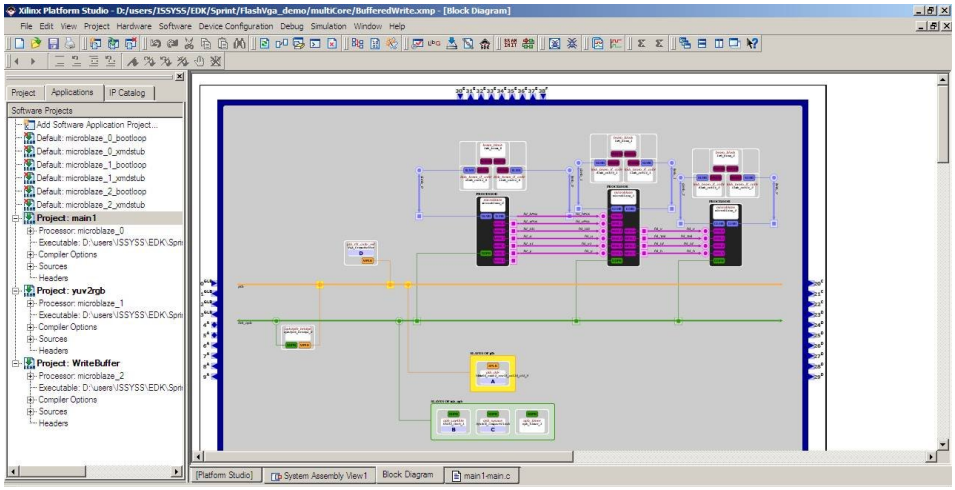
Ukázku prostředí můžeme vidět na obrázku 2.3. Tento nástroj disponuje modulem Base System Builder, který automatizuje hardwarovou a softwarou konfiguraci platformy. Vytvoření nového projektu projde celou řadou kroků, jak budoucí aplikaci vytvořit. Nejdříve zvolíme vývojovou desku, pro kterou aplikaci vytváříme. Dále vybereme procesor, nastavíme jej podle našich požadavků a nakonfigurujeme vstupy a výstupy. Jako poslední krok nastavíme softwarové vlastnosti. Po vytvoření projektu je obrazovka rozdělena do tří hlavních částí.

Vlevo se nachází pracovní prostor se třemi záložkami, kde přepínáme mezi stromovou strukturou rozpracovaných projektů, jejich potřebných částí a katalogem komponent.

Uprostřed obrazovky vidíme blokové schéma vygenerovaného obvodu, aktivní záložka Block Diagram. Přepnutím na další záložku pojmenovanou System Assembly View umístěnou pod schématem vidíme přehledné zapojení sběrnic a komunikaci signálů. Každé schéma minimálně obsahuje PLB a LMB. Pomocí záložek lze přepínat mezi soupiskou přiřazených portů a soupiskou adresovacího prostoru.

Ve spodní části obrazovky se nachází příkazový řádek, který informuje o syntéze a nahrání aplikace na platformu. Projekt přeložíme za pomoci položky Hardware v menu a volbou Generate Bitstream.

Tvorba projektů v tomto prostředí požaduje vyšší znalost cílové platformy a jednotlivých komponent. Program za nás vše vygeneruje a vede nás krok za krokem. Pro snadnější orientaci v programu je potřeba větších zkušeností s tvorbou aplikací v tomto prostředí. Blokové schéma vygenerované aplikace působí nepřehledně díky použití velkého počtu barev.




Obrázek 2.3: Screenshot aplikace Xilinx EDK (převzato z [12]).

### Popis komponenty

Tento nástroj opět disponuje dynamickou databází komponent. Databázi jednoduše rozšíříme o vlastní komponentu vytvořením souboru `*.mpd`. Popis komponenty nástrojem EDK má specialní formát popisu. Řádky začínající znakem # zobrazují komentář, jako například hlavička:

```
## Copyright (c) 1995-2007 Xilinx, Inc. All rights reserved.
##############################################################################
##
## Name : opb_spi
## Desc : Microprocessor Peripheral Description
## : Automatically generated by PsfUtility
##
##############################################################################
```

Tělo popisu komponenty je uvozeno mezi klíčová slova `begin` a `end`, kde za slovem begin následuje název komponenty. Popis komponenty nástroje EDK používá jen pár klíčových slov a to `OPTION`, `PARAMETR` a `PORT`. Za každým z těchto klíčových slov následuje název položky, který přesně definuje požadovanou informaci a za pomoci znaku = přiřadí příslušnou hodnotu. Klíčové slovo `OPTION` udává obecné informace o komponentě. Jako příklad obecných informací uveďme, že se jedná o periferii, použitým HDL jazykem je VHDL, datum poslední modifikace, popis komponenty a na jakou architekturu FPGA je možno komponentu namapovat.

```
## Peripheral Options
OPTION IPTYPE = PERIPHERAL
OPTION HDL = VHDL
OPTION LAST_UPDATED = 9.1.2
OPTION DESC = OPB SPI Interface
OPTION LONG_DESC = OPB to MotorolaŽ Serial Peripheral Interface (SPI) adapter
OPTION ARCH_SUPPORT_MAP = (qrvirtex2=PREFERRED, qvirtex2=PREFERRED,
spartan2=PREFERRED, spartan2e=PREFERRED, spartan3=PREFERRED, virtex=PREFERRED,
virtex2=PREFERRED, virtex2p=PREFERRED, virtex4=PREFERRED, virtexe=PREFERRED,
spartan3e=PREFERRED, virtex5=PREFERRED, spartan3a=EARLY_ACCESS,
spartan3adsp=EARLY_ACCESS)
```

Následující popis komponenty opět obsahuje pouze informace o rozhraní, kde generické parametry označuje klíčové slovo `PARAMETR` a jednotlivé signály klíčové slovo `PORT`. U generických parametrů je především potřeba uvést defaultní hodnotu, datový typ a rozsah hodnot, kterých může nabývat. Příklad:

```
PARAMETER C_DEV_BLK_ID = 4, DT = INTEGER
PARAMETER C_DEV_MIR_ENABLE = 0, DT = INTEGER
PARAMETER C_INTERRUPT_PRESENT = 1, DT = INTEGER, RANGE = (0,1)
PARAMETER C_FAMILY = virtex2, DT = STRING
```

Posledním důležitým klíčovým slovem je `PORT`, který definuje vlastnosti signálů. Zde hlavně potřebujeme uložit směr signálu (vstup, výstup, obojí) a datovou šířku. Příklad:

```
PORT MOSI_I = "", DIR = I
PORT MOSI_O = "", DIR = O
PORT SS_I = "", DIR = I, VEC = [0:(C_NUM_SS_BITS-1)]
PORT SS_O = "", DIR = O, VEC = [0:(C_NUM_SS_BITS-1)]
```

Popis již není tak přehledný, především u portu, ale přesto je velmi intuitivní a snáze lze pochopit, jak danou komponentu popsat. Jednoduchost spočívá v nízkém počtu klíčových slov a snadné orientaci i bez použití uvození souvisejícího kontextu do závorek nebo použití speciálního znaku pro oddělení jednotlivých položek popisu. Nevýhoda spočívá v použití speciálního parseru díky speciálnímu návrhu formátu popisu.

### Popis vlastního dialogového okna pro konfiguraci komponenty

Pokud si nevystačíme se standardním konfigurovacím oknem komponenty, pak tento nástroj disponuje tvorbou vlastního dialogového okna. Vzhled okna popisuje soubor `*.mui`, který je napsán ve formátu XML a o jehož obsluhu se stará skript napsaný v jazyce TCL.

Celý popis okna je uvozen ve dvojici klíčových slov `<doc>` a `</doc>`. Dále následují klíčová slova `<view>`, značící záložku s jednoznačným id jako parametr. Za tímto klíčovým slovem následuje dvojice `<display>` a `</display>`, která učuje název záložky. Poté jednotlivé parametry, které chceme nastavit, vkládáme do skupin uvozených klíčovými slovy `<group>` a `</group>`. Uvnitř je pomocí položky `<display>` uvedeno jméno skupiny konfigurovatelných parametrů. Jednotlivé parametry definuje dvojice `<item></item>`. Příklad:

```
<doc>
    <view id="User" >
        <display>User</display>
        <group id="DCM" >
            <display>DCM</display>
            <item>&C_DFS_FREQUENCY_MODE;</item>
            <item>&C_DLL_FREQUENCY_MODE;</item>
            <item>&C_DUTY_CYCLE_CORRECTION;</item>
        </group>
    </view>
</doc>
```

## 2.3 PSoC Creator

PSoC Creator je produkt americké společnosti Cypress Semiconductor, která zahájila spolupráci s firmou Keil, divizí obvodů ARM, na vytvoření kompilátoru pro PSoC Creator a tím dala vznik kvalitnímu nástroji pro tvorbu vestavěných systémů [7].

### Grafický editor

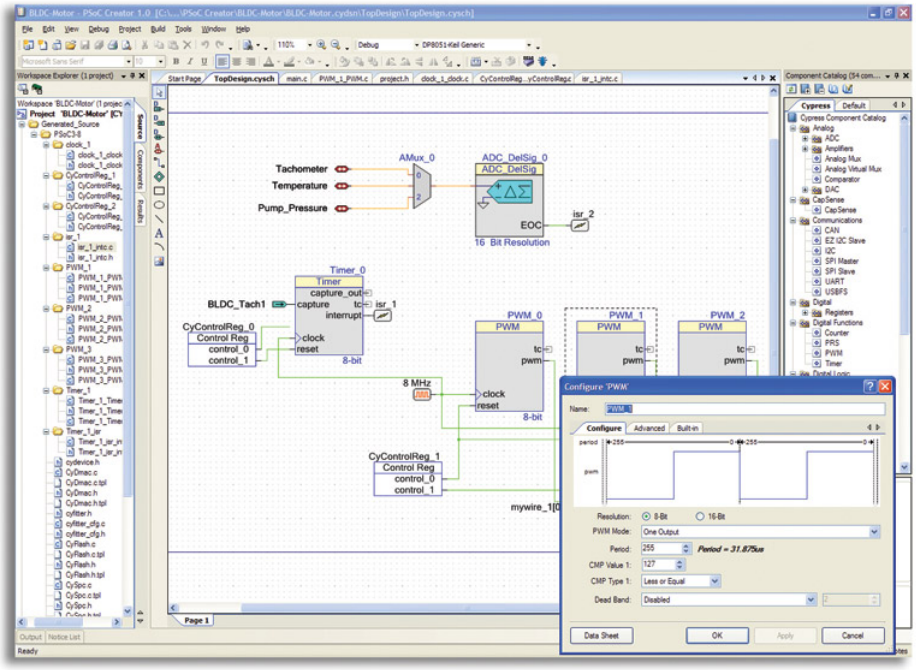
Společnost Cypress Semiconductor nabízí dva vývojové kity, proto je potřeba při tvorbě nového projektu vybrat, pro který kit chceme obvod navrhnout. Po vytvoření projektu se zobrazí designer, který je rozdělen do čtyř hlavních částí, viz 2.4.

V levém okraji obrazovky se nachází prohlížeč pracovního prostoru, který obsahuje ve stromové struktuře veškeré vytvořené součásti, nutné pro tvorbu projektu. Soubor `*.cysch` uchovává doposud nakreslené schéma obvodu. Další položka, a to soubor `*.cydwr`, zobrazí veškeré údaje o platformě, pro kterou obvod navrhujeme. Poté jen následuje popis chování obvodu v jazyce C nebo v jazyce Verilog. Chceme-li se podívat na námi zvolenou platformu, otevře se nová záložka v oblasti grafického editoru, který je umístěn uprostřed obrazovky. Pomocí záložek umístěných v dolní části přepínáme mezi popisem platformy v grafické podobě a různými dalšími doplňujícími informacemi o dané platformě.

Grafický editor tvoří plátno obsahující mřížku pro snadnější kreslení spojů a manipulaci s komponentami a panel, pomocí něhož kreslíme spoje, vkládáme pomocné texty a obrázky.

Tento designer je velmi komplexní nástroj, proto se pod grafickým editorem nachází příkazová řádka. Skládá se ze dvou záložek, kde první zobrazuje pomocná hlášení při tvorbě schématu nebo při překladu obvodu a druhá ukazuje počet chyb, pokud je něco nesprávně zapojeno.

Pravou stranu obrazovky společnost vyhradila prostoru pro zobrazení katalogu komponent, uspořádaného do stromové struktury podle kategorie zařazení. Díky stromové struktuře lze snadno vyhledávat. Pokud ovšem nechceme procházet celý strom než danou komponentu nalezneme, můžeme vyhledávat zadáním klíčového slova. Výběrem komponenty dojde




Obrázek 2.4: Screenshot aplikace PSoC Creator (převzato z [23]).

ve spodní části katalogu k zobrazení schématické značky a odkazu na datasheet.ve formátu pdf. Kliknutím a tahem levého tlačítka vkládáme komponenty do grafického editoru. Poté s ní můžeme pomocí stejného tlačítka manipulovat. Kliknutím pravého tlačítka na komponentu zobrazíme popup menu, kde můžeme komponentu vymazat, kopírovat, přejmenovat, otáčet ji, vytvořit schématickou značku a především ji nastavit. Nastavovací okno lze také zobrazit dvojklikem levého tlačítka na komponentu. Zde jsou uvedeny parametry, které lze u dané komponenty nastavit.

Pokud máme vytvořený obvod a často jej používáme, můžeme ho vložit do katalogu komponent. Pomocí volby generovat symbol otevřeme designer, ve kterém můžeme vytvořit schématickou pro námi napsaný obvod. Poté lze použít obvod jako jednu komponentu. Jestliže chceme rozšířit nastavovací okno o další parametry, poslouží nám opět designer schématické značky. Volbou parametry symbolu vložíme další potřebné parametry.

Jak už bylo zmíněno výše, funkční popis obvodu je proveden v jazyce C či Verilog. Schématická značka a jednotlivé parametry jsou zakódovány v binární podobě v souboru `*.cysch`, tudíž nevíme, co je zde přesně uloženo. Pokud si nevystačíme s designerem schématické značky, můžeme správu komponenty zapouzdřit do několika souborů napsaných v jazyce C#.

Celkově je nástroj velmi snadno pochopitelný. I bez předešlé znalosti lze ihned kreslit schéma. Tento nástroj spatřuji z hlediska grafického editoru jako nejlepší. Přehledný design, databáze komponent uspořádaná do přehledné stromové hirearchie. Náhled označené komponenty opět urychluje a vytváří návrh aplikací snadnější. Spojování komponent je provedeno za pomoci jednoho tlačítka, není potřeba rozlišovat sběrnice apod. Datovou šířku lze nastavit v dialogovém oknu každého propoje. I za cenu nutné znalosti jazyka C# dělá možnost tvorby vlastního schématu a vlastního dialogového okna nástroj velmi flexibilní.

### Popis komponenty

Ani v tomto nástroji nesmí chybět dynamická databáze komponent. Funkční popis komponenty může být napsán v jazyce C nebo Verilog. Schématická značka je kódována do binární podoby a uložena jako soubor `*.cysch`. Pokud ovšem potřebuji vlastní schématickou značku, vytvořím několik souborů psaných v jazyce C#. Konstruktor objektu je opatřen specálními direktivami, které určují tvorbu chématické značky a tvorbu popisu ve Verilogu. Direktiva `#region ICyVerilogCustomize_v1 Members` značí metody pro generování popisu ve Verilogu. Naopak direktiva `#region ICyShapeCustomize_v1 Members` definuje metody pro tvorbu schématické značky. Nástroj PSoC při kreslení schématické značky interpretuje napsaný kód a zobrazí schématickou značku.

Velkou nevýhodu vidím v nutné znalosti programovacího jazyka C#. Bez jeho znalosti se uživatel nevyzná ani v některých existujících komponentách.

# Kapitola 3

# Platforma FITkit

Platforma FITkit vznikla na Fakultě informačních technologií s cílem umožnit studentům realizovat nejen softwarové, ale také hardwarové projekty či dokonce celé aplikace [14].

FITkit obsahuje 16-bitový nízkopříkonový mikrokontrolér rodiny MSP430 firmy Texas Instruments a programovatelné hradlové pole XC3S50-4PQ208C řady Spartan 3 firmy Xilinx. Dále je vybavena řadou periferii jako je audio rozhraní, konektory PS2, rozhraní VGA, konektor RS232, DRAM 8x8Mbit, klávesnice, řádkový LCD displej, rozšiřující konektory, USB převodník. Ten poskytuje dva kanály (A a B), umožňující komunikaci po USB a tím dva nezávisle konfigurovatelné kanály. Převodník je kompatibilní s USB 2.0 Fulistl Speed.

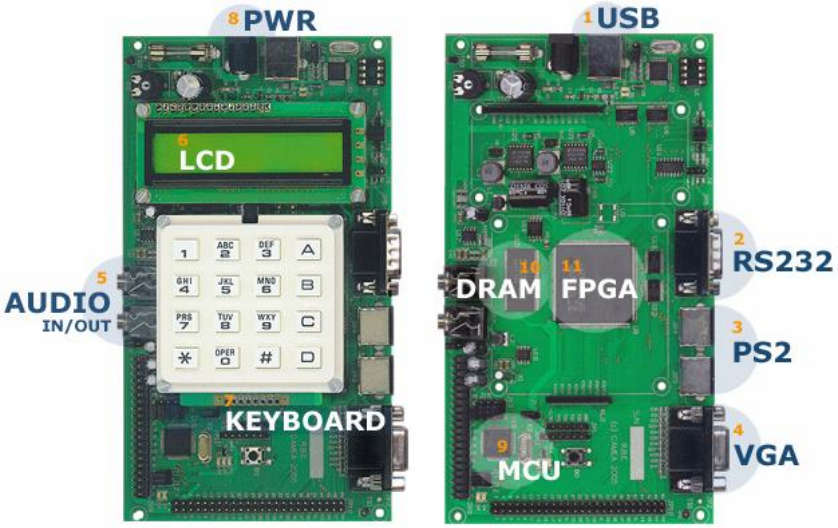
**Kanál A,** který je připojen k FPGA obvodu, umožňuje komunikovat s PC v libovolném z podporovaných režimů. Uvnitř FPGA obvodu lze tedy např. vytvořit zařízení, které bude možné po I2C ovládat z PC. Jinou možností je napojit kanál A na řadič seriového kanálu a z PC zařízení ovládat přes virtuální COM port. Naopak, v FPGA může sloužit pouze jako "drát"mezi zařízením připojeným k FITkitu a počítačem. Ke komunikaci s kitem je možné využít:

- knihovny libkitclient, která umožňuje přímý přístup k FTDI bez nutnosti zjišťovat přiřazený COM port a
- tzv. virtuálního COM portu, který je automaticky vytvořen při připojení FITKitu.

**Kanál B** je připojen k programovacím pinům mikrokontroléru (RESET, TST) a dále k pinům seriového rozhraní (RxD, TxD). Pomocí tohoto kanálu lze tedy mikrokontroler programovat a komunikovat s ním přes terminálový program. Možnostem komunikace s mikrokontrolérem se věnuje dokument Komunikace s FITkitem [18]. FITkit byl vytvořen ve verzích 1.0, 1.2 a 2.0.

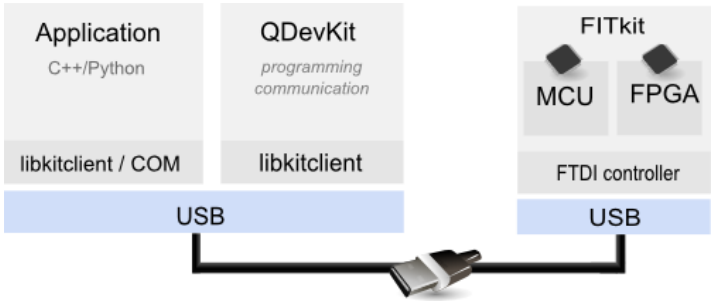
## 3.1 Knihovna libkitclient

Knihovna libkitclient je multiplatformní knihovna pro správu a interakci s FITKity. Je napsána v C++ a existuje pro ni bindings pro jazyk Python, napsaný pomocí systému SWIG, který dovoluje snadnou rozšiřitelnost do dalších skriptovacích jazyků. Knihovna poskytuje jednotné API v prostředí Windows i Linux. Ve Windows je implementována nad knihovnou FTD2XX.dll, v prostředí Linux nad modifikovanou verzí libftdi.




Obrázek 3.1: Pohledy na FITkit (převzato z [13])

Mezi přednosti patří umožnění přímého přístupu k FITkitu přes USB, čímž odpadá nutnost složitě konfigurovat virtuální COM porty, jejichž přiřazení je poněkud nedeterministické. Další výhodou je možnost pracovat s FITkitem z vlastní aplikace [17].



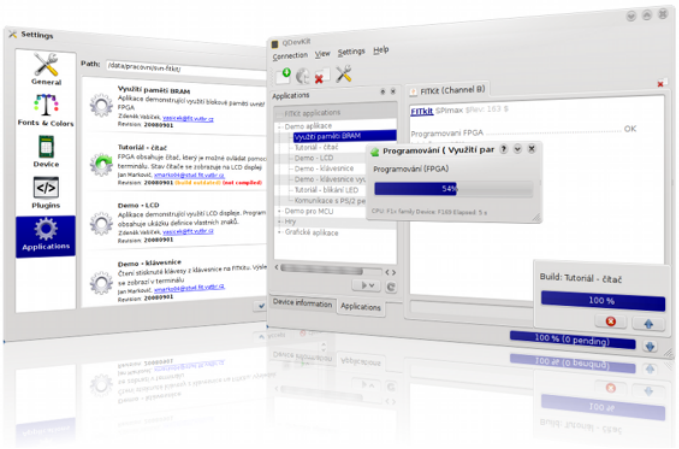

Obrázek 3.2: Možnosti komunikace s FITkitem na platformě Linux/Windows (převzato z [17]).

## 3.2 QDevKit

Multiplatformní terminálový program QDevKit byl navržen s cílem unadnit práci s FITkitem. Nevýhodou klasických terminálů je jejich vzájemná nekompatibilita, nemožnost pracovat s FITkitem přímo, nýbrž přes emulovaný seriový port, nutnost nastavit řadu parametrů a především neexistence jednotného multiplatformního řešení.

Nyní program QDevKit umožňuje:

- automatickou detekci zařízení a komunikaci platformy FITkit s počítačem,
- správu existujících projektů,
- překlad projektů,
- možnost odzkoušet již přeložený projekt přímo na platformě FITkit.

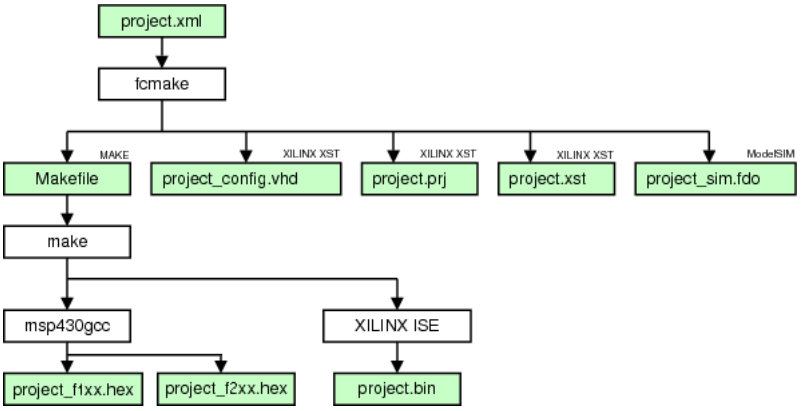


Obrázek 3.3: Screenshot aplikace QDevKit.

Rozšiřující modul Designer umožní snadnou tvorbu šablon pro nové projekty.

## 3.3 Překladový systém

Z důvodu sjednocení překladu pod systémem Windows a Linux byl vytvořen jednoduchý překladový systém založený na XML souborech. XML soubor (typicky projekt.xml) popisuje projekt, závislosti na dalších zdrojových souborech a umožňuje definovat specifická nastavení. Kromě překladu slouží XML pro generování seznamu aplikací a zjišťování informací o konkrétní aplikaci. Schéma překladového systému je zobrazeno na obrázku 3.4.




Obrázek 3.4: Schéma překladového systému (převzato z [15])

XML soubor slouží jako vstup programu fcmake, který na základě svého obsahu vygeneruje skripty pro XILINX ISE, Modelsim a Makefile soubor, se kterým je možné již pracovat pomocí standardních nástrojů. XML soubor obsahuje seznam parametrů, dále pak seznam všech zdrojových souborů a knihoven, které projekt pro překlad potřebuje, apod. Vygenerovaný soubor Makefile obsahuje pouze informace týkající se překladu konkrétního projektu a dále se odkazuje na sadu překladových pravidel umístěnou v adresáři base. XML soubor tvoří tři části:

- popis projektu,

- část obsahující informace pro generování kódu pro mikrokontroler (MCU),
- část s informacemi potřebnými pro vygenerování konfigurace pro FPGA.

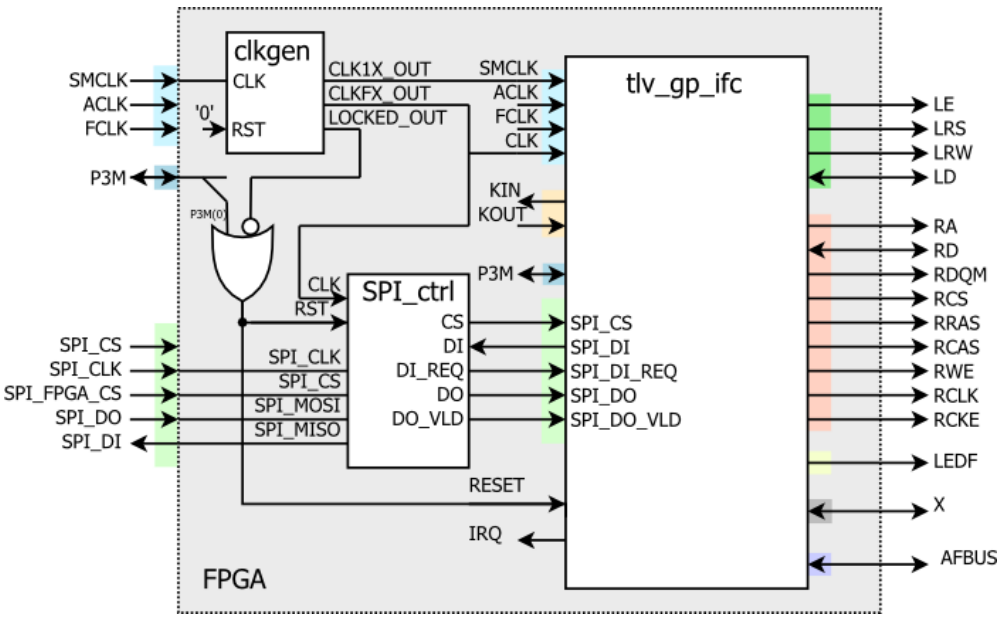
**Popis projektu** obsahuje stručný název projektu, popis projektu, jméno autora, email a aktuální verzi (revizi), která obsahuje datum poslední modifikace.

**MCU sekce** slouží ke specifikaci souborů, které jsou zapotřebí pro překlad kódu pro mikrokontroler a případných dalších parametrů.

**FPGA sekce** používá zcela stejný princip definice zdrojových souborů a závislostí jako MCU sekce, tzn. elementy <file>, <files> a <include>. Na rozdíl od MCU je zapotřebí brát ohled na pořadí vkládání souborů, neboť ModelSIM není schopen řešit automaticky závislosti [15].

## 3.4 Top-level architektura

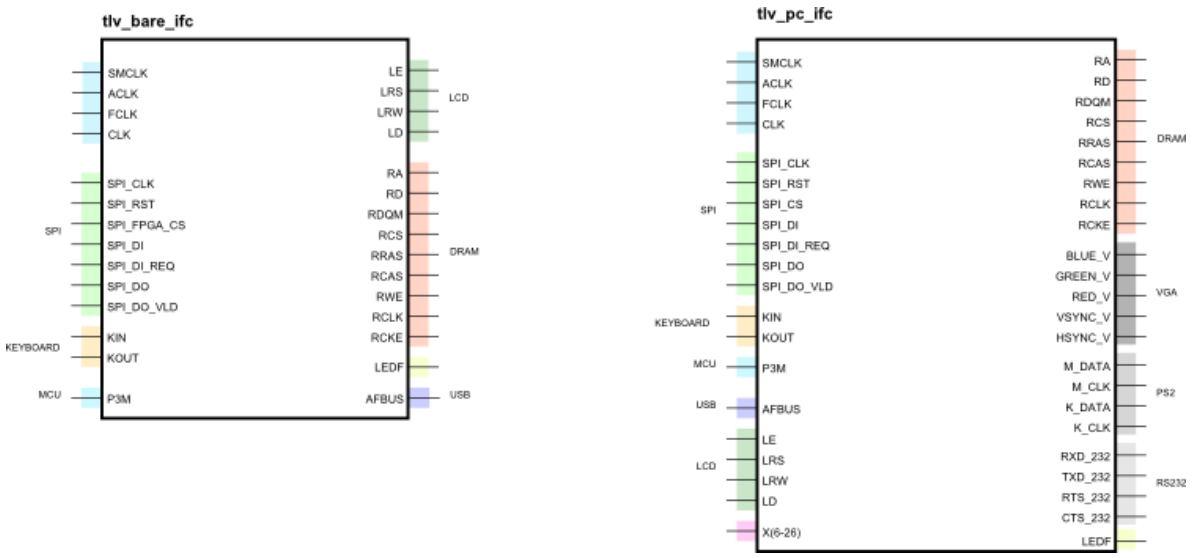
Pro tvorvu FPGA aplikací existují tři předdefinované top-level entity, které integrují nezbytné komponenty (generátor hodinového signálu, řadič rozhraní SPI) a poskytují různý interface dle použití. Volba top-level entity se děje pomocí atributu architecture v souboru project.xml v sekci FPGA. Předdefinované entity:




Obrázek 3.5: Architektura top-level FPGA, umístění entity tlv_gp (převzato z [16]).

- bare - základní entita, vhodná pro aplikace nevyužívající port X ani periferie na FITkitu, viz obrázek 3.5
- gp - entita vhodná pro aplikace, které využívají pouze X port, viz obrázek 3.6

- pc - entita určená pro aplikace využívající periferie FITkitu (VGA port, PS/2, atd). Dodejme, že tato entita reflektuje situaci, kdy propojka J6 je aktivní, periferie kitu jsou povoleny a tudíž část portu X je sdílena s těmito periferiemi. Rozdíl oproti předchozím entitám spočívá v tom, že rozhraní obsahuje signály portu X, pojmenované dle dané periferie, viz obrázek 3.6




Obrázek 3.6: Entity bare a pc (převzato z [16]).

## 3.5 Základní FPGA komponenty

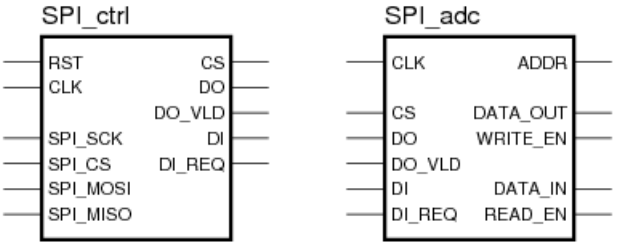
Vzhledem k tomu, že komponenty budou tvořit základ komponentově-orientovaného grafického návrhového nástroje, byla vytvořena sada řadičů, které zjednodušují návrh aplikací a úloh na kitu. Řadiče jsou popsány pomocí jazyka VHDL a budou tvořit základní komponenty modulu.

### SPI řadič

SPI řadič je jednoduchá, ale nesmírně důležitá komponenta, která bude tvořit základ všech vytvářených aplikací v modulu pro QDevKit. Má za úkol realizovat převod SPI protokolu na interní sériovou komunikaci a také detekuje, zda se příslušná transakce týká FPGA.

### SPI dekodér

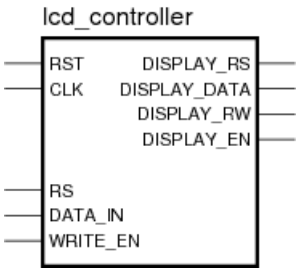
SPI dekodér je synchronní komponenta, která má za úkol převádět vnitřní sériovou komunikaci na jednoduché paralelní rozhraní, které umožní komunikaci s připojenými periferiemi.




Obrázek 3.7: Rozhraní SPI (převzato z [19]).

### Řadič LCD displeje

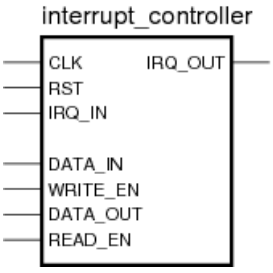
Součástí FITkitu je šestnácti znakový LCD displej, který je připojen přímo na piny FPGA.




Obrázek 3.8: Rozhraní LCD řadiče (převzato z [20]).

### Řadič přerušení

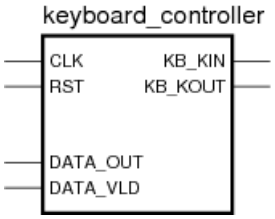
V některých aplikacích potřebujeme získat hodnotu určité komponenty na základě vzniklé události. Aby odpadlo neustálé čtení a tím režie navíc, využívá se systému přerušení. Řadič přerušení je navrhnut genericky, aby bylo možné jednoduchým způsobem nastavit počet vstupních přerušení.




Obrázek 3.9: Rozhraní komponenty přerušení (převzato z [21]).

### Klávesnice 4x4

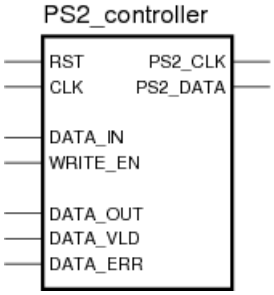
Na kitu je přímo osazena alfanumerická klávesnice, která obsahuje 16 tlačítek zapojených do matice 4x4.




Obrázek 3.10: Rozhraní komponenty (převzato z [10]).

### Řadič PS/2

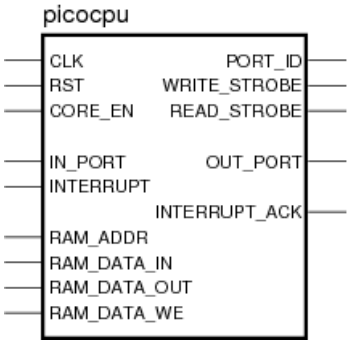
PS/2 řadič je jednoduchá komponenta, která má za úkol realizovat převod sériového protokolu PS/2 na paralelní a naopak. Nejčastěji se používá pro připojení klávesnice nebo myši.




Obrázek 3.11: Rozhraní PS/2 řadiče (převzato z [22]).

### Komponenta PicoCPU

Jedná se o 8 bitový procesor od společnosti Xilinx, který disponuje jednoduchou instrukční sadou.




Obrázek 3.12: Rozhraní komponenty PicoCPU (převzato z [25]).

# Kapitola 4

# Použité nástroje

## 4.1 Qt Framework

Qt je jedna ze dvou nejpopulárnějších multiplatformních knihoven pro vytváření programů s grafickým uživatelským rozhraním.

Qt toolkit byl vytvořen v roce 1999 společností Trolltech, která jej v roce 2008 prodala firmě Nokia. V březnu roku 2011 Nokia ohlásila prodej práv na provoz podpůrných služeb a prodej licencí pro komerční projekty, vytvořených pomocí Qt, společnosti Digia. Zároveň však Nokia ujišťuje, že po transakci zůstane hlavním vývojářem tohoto toolkitu.

Od roku 1999 se Qt toolkit vyvinul v multiplatformní nástroj, ve kterém lze vyvíjet konzolové nebo GUI aplikace v odlišných programovacích jazycích pro různé platformy. Aplikace napsané s pomocí toolkitu je možno distribuovat pod licencí GPL, LGPL, nebo po splnění určitých podmínek i komerčně.

Qt je knihovna programovacího jazyka C++, ale existuje i pro jazyky Python (PyQt, PySide), Ruby (QtRuby), C, Perl, Pascal, C#, Java (Jambi) a Haskell. Podporuje lokalizaci aplikací a také SQL, zpracování XML, správu vláken, přístup k souborům, práci s grafikou a multimédii. Velkou výhodou Qt je velmi přehledně zpracovaná dokumentace a také vývojové programy Qt Creator nebo Qt Designer. Aplikace, vytvořené pro grafické uživatelské prostředí, používají nativní vzhled operačního systému, takže vyvinuté aplikace se vždy přizpůsobí používanému prostředí.

Qt společně s GTK+ nahradila starší Motif. Důkazem kvality a rozšířenosti toolkitu je jeho použití například pro projekty Skype, Google Earth, prostředí KDE, webový prohlížeč Opera, VirtualBox a jiné.

Qt knihovna byla naposledy uvolněna 21. září 2010 ve veri 4.7. Klíčovým rysem verze je uvedení QML, který je popisován jako JavaScript-like deklarativní jazyk pro jednodušší vytváření rozhraní programu. Nová verze třídy QStaticText renderuje dvakrát rychleji než ve verzi Qt4.6. Pomocí enginu QPainter systém efektivněji využívá OpenGL. Nová hardwarová akcelerace QtWebkit renderuje o 31% rychleji.

Podporované platformy:

- Linux/X11 - Qt pro X Window System 32bit a 64bit (Linux, HP-UX, Solaris, AIX, ...)
- Windows - Qt pro Microsoft Windows XP, Vista a 7 (spolu s minGW nebo MSVC 2008)

- Mac - Qt pro Apple Mac OS X 10.6 "Snow Leopard"a Apple Mac OS X 10.5 "Leopard"x86_64 (Cocoa 32 and 64bit).
- Vestavěné linux platformy - Embedded Linux QWS (ARM) (PDA, Smartphone, ...)
- Windows CE 5.0 - Qt pro Windows CE (ARMv4i, x86, MIPS)
- Symbian - QT pro Symbian/S60 5.0 platformu
- Maemo - Maemo 5 (Linux, ARM, X11), plná podpora není zaručena

**Licence**

- Qt Komerční licence pro vývojáře - licence je shodná s licencí používanou při vývoji klasické komerční aplikace. Tato verze je pro vývojáře, kteří nechtějí sdílet zdrojový kód s ostatními v souladu s licencí GPL nebo LGPL.
- Qt GNU LGPL v. 2.1 - tato verze Qt licence je vhodná pro vývoj aplikací open source za předpokladu, že vývojář splní podmínky obsažené ve verzi GNU LGPL 2.1.
- Qt GNU GPL v. 3.0 - tato verze Qt licence je vhodná pro vývoj Qt aplikací, pokud hodláte používat aplikaci v kombinaci se softwarem s podmínkami GNU General Public License verze 3.0 nebo kde jste ochotni dodržet podmínky GNU General Public License verze 3.0

**Signály a sloty**

Důležitou vlastností Qt toolkitu je přítomnost signálů a slotů pro komunikaci mezi objekty.Např. pokud se ve widgetu (element GUI, který zobrazuje nebo předává informace pomocí interakce s uživatelem.) uskutečnila akce, jež změnila jeho stav, pak o tom může být informován widget umístěný v jiném okně aplikace. Signály a sloty tvoří velmi silný programátorský nástroj, viz obrázek 4.1.

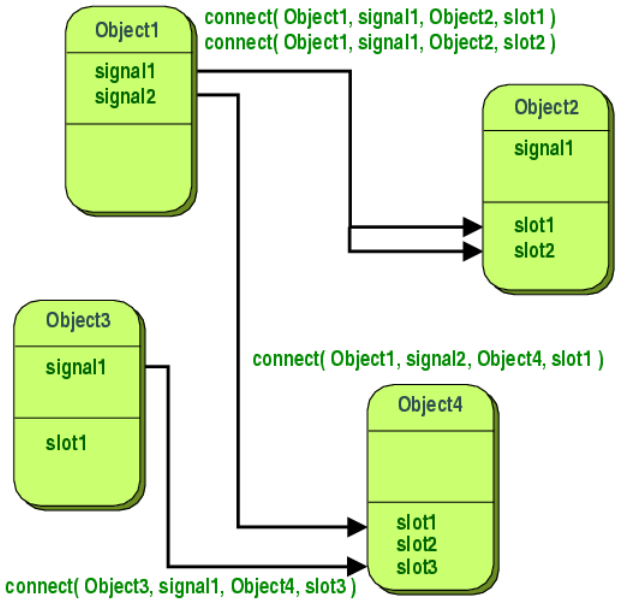
Místo signálů a slotů se dříve pro tento typ komunikace používal tzv. callback, což je ukazatel na metodu objektu, kterou chceme vyvolat po nějaké události jiného objektu. Tento přístup měl dvě nevýhody. Nebyla zde při volání typová kontrola a metody volané pomocí callback jsou silně vázané, tzn. volaná metoda musí znát ukazatel na metodu, z které byla vyvolána. Při používání signálů a slotů se tyto dvě nevýhody nevyskytují. Pro jejich použití se definuje spojení signálu se slotem pomocí metody connect. V případě potřeby se pak spojené signály pouze "vyvolávají".

Sloty a signály mohou být využity ve všech objektech, které jsou přímo nebo nepřímo zděděny ze třídy QObject. Při propojování signálů a slotů může být s jedním slotem spojeno několik různých signálů a stejně tak na jeden signál napojeno několik slotů. Sloty mohou být použity pro přijímání signálů a zároveň mohou být použity jako standardní metoda objektu [3].

**Qt nástroje**

Balík Qt frameworku obsahuje několik nástrojů, které usnadňují vývoj aplikací:

- Qt Creator - integrované vývojové prostředí pro jazyk C++ a QML. Obsahuje designer pro tvorbu návrhu vzhledu aplikace a formulářů, editor C++ kódu se zvýrazněním syntaxe či automatického dokončování. Taktéž obahuje debugger a překladač.




Obrázek 4.1: Znázornění systému signálů a slotů. (převzato z [1]).

- qmake - je nástroj, který generuje projektové soubory, z kterých je následně možné tímto nástrojem automaticky vygenerovat Makefile, používaný pro překlad aplikace.
- Qt Simulator - nepostradatelnou součástí Qt Creatoru, potřebnou pro vývoj aplikací mobilních zařízení, je Qt Simulator. Používá se pro testovaní Qt aplikací, které jsou určené pro mobilní zařízení. Umožňuje testovaní s různými konfiguracemi a simulací různých prostředí.
- Qt Assistant - nástroj pro zobrazení dokumentace.
- Qt Linguist - umožňuje jednoduchý překlad všech textů, s kterými příjde uživatel, vytvářející aplikaci, do styku.

## 4.2 XML

XML je zkratka pro Extensible Markup Language. Je to obecný značkovací jazyk, který byl vyvinutý a standardizovaný konzorciem W3C jako otevřený standard. Je zjednodušenou, textově orientovanou verzí svého předchůdce SGML. Jazyk XML obsahuje podporu Unicode všech světových jazyků, přičemž hlavní důraz klade na jednoduchost, všeobecnost a použitelnost na internetu. XML umožňuje jednoduché vytváření konkrétních značkovacích jazyků, tzv. aplikací pro různé účely a typy dat. Z tohoto důvodu je podporovaný řadou aplikací a má široké spektrum využití, například pro serializaci dat nebo pro uložení databázových dat. Primárním účelem je však výměna dat mezi aplikacemi a publikování dokumentů, kde XML jen popisuje strukturu z pohledu obsahu, bez popisu vzhledu [8]. V současnosti jsou mnohé z používaných formátů založené na XML, z nejznámějších například formáty kancelářských programů - Microsoft Office nebo OpenOffice.org.

### Historie a vývoj

XML je odvozen z jazyka SGML, kde jeho popularita vzrostla s narůstajícím využitím internetu. XML obsahuje nezměněnou část z ISO standardu SGML a některé ostatní časti jsou z ní odvozené. Ze SGML XML přebralo oddělení logické a fyzické struktury - elementy a entity, validaci oproti DTD a také oddělení dat a metadat - elementy a atributy. Dalšími přebranými prvky jsou oddělení zpracovaní od reprezentace, smíšený obsah a syntax používající úhlové závorky. Avšak naproti SGML se změnila deklarace, kterou u XML nahradil pevný oddělovač. Další změnou je možnost využití Unicode pro kódovaní dokumentů. Jednou z výhod XML je možnost vytvářet aplikačně specifické tagy, přičemž současně je možné ověřit platnost vytvořeného dokumentu oproti DTD [6].

### Verze

První verzí XML je XML 1.0, která byla definovaná v roku 1998. Od svého vzniku podstoupila několik revizí, které nebyly příliš významné a proto nebyla povýšena celá verze, ale měnila se jen její edice.

Verze XML 1.1 je druhou dostupnou verzí současnosti. Současná edice této verze je označena jako druhá. Avšak tato verze jazyka XML není zatím příliš používaná, kromě speciálních případů, kde jsou potřebné její specifické vlastnosti [9].

Před vydáním páté edice XML 1.0 byly rozdíly mezi verzemi jazyka XML v přísnějších požadavcích na znaky použitelné v elementech, jménech atributů a také v jedinečných identifikátorech. V prvních čtyřech verzích XML 1.0 byly znaky určené výlučně verzí Unicode standardu. Pátá edice jazyka XML 1.0 přinesla nový přístup, který je také použit ve všech edicích XML 1.1. Princip uváděného přístupu spočívá v povolení použití všech znaků, s výjímkou znaků, které jsou "zakázané"[8]. Takový přístup zlepšuje možnosti pojmenování v uvedených edicích jazyka XML.

Objevily se návrhy na vytvoření nové verze XML 2.0, která by obsahovala jmenné prostory, XML Base [1], XML Information Set [2] a mnohé další [5]. Návrh XML 2.0 však nebyl doposud realizovan.

## 4.3 CMake

CMake je open-source a multiplatformní systém, který je nezávislý na zdrojovém kódu. Jednoduché konfigurační soubory umístěné v každém zdrojovém adresáři (tzv. CMakeLists.txt soubory) se používají pro generování standardních sestavovacích souborů (např. makefiles na Unixu a projekty/pracovní prostory ve Windows Microsoft Visual C++), které jsou použity obvyklým způsobem. CMake může generovat nativní prostředí, které bude kompilovat zdrojové kódy, vytvářet knihovny a budovat programy v libovolných kombinacích. CMake také podporuje sestavení statických a dynamických knihoven. Další příjemnou vlastností je, že CMake generuje cache soubor, který je určen pro editaci editorem. Například, když CMake běží, najde všechny hlavičkové soubory, knihovny a spustitelné programy, a může dojít k volitelnému sestavení direktiv. Tyto informace jsou shromažďovány do cache, která může být změněna uživatelem před generací původních sestavovacích souborů.

Použití CMake je jednoduché. Proces sestavení je řízen tím, že se vytvoří jeden nebo více CMakeLists.txt souborů v každém adresáři (včetně podadresářů), které tvoří projekt.

[1] Atribut používaný pro vkládání URI, ke které se vztahují nasledující odkazy v dokumentu.
[2] Definuje abstraktní datový model pro XML dokumenty.

Každý CMakeLists.txt se skládá z jednoho nebo více příkazů. Každý příkaz má tvar CMD (args...), kde CMD je název příkazu, a args je seznam argumentů oddělený bílými znaky. CMake poskytuje mnoho předdefinovaných příkazů, ale pokud chcete, můžete přidat své vlastní. Kromě toho může pokročilý uživatel přidat další makefile generátory pro konkrétní překladač/OS. Zatímco Unix a MSVC++ je podporován v základu, ostatní vývojáři přidávají další překladač/OS podporu.

CMake byl vytvořen jako reakce na potřebu vybudovat silné, multiplatformní prostředí pro Insight Segmentation and Registration Toolkit (ITK), financovaného z NLM v rámci Visible Human Project. První CMake verze byla uvolněna v polovině roku 2000 a na počátku roku 2001 dochází k výraznému vývoji. Mnoho vylepšení vzniklo díky vlivům jiných vývojářů, kteří zakomponovali CMake do svých systémů [2].

# Kapitola 5

# Návrh a specifikace modulu

Kapitola je věnována návrhu a specifikaci návrhu modulu pro aplikaci QDevKit. V sekci návrhu jsou kladeny požadavky na modul a následně jsou podrobněji rozebrány v sekci specifikace návrhu.

## 5.1 Návrh

Cílem této práce je navrhnout modul pro aplikaci QDevKit, který bude generovat šablonu aplikace pro FPGA a MCU pomocí grafického prostředí. Modul umožní uložit rozpracovaný projekt, načíst již existující a exportovat schéma jako obrázek. Každý projekt by měl obsahovat kreslící plátno pro návrh obvodu a umožní přidávat, propojit a konfigurovat jednotlivé komponenty. Je potřeba vhodně zvolit reprezentaci komponent tak, aby je bylo možné snadno modifikovat, případně rozšířit stávající repertoár. Výstupem bude vygenerovaný projekt pro QDevKit.

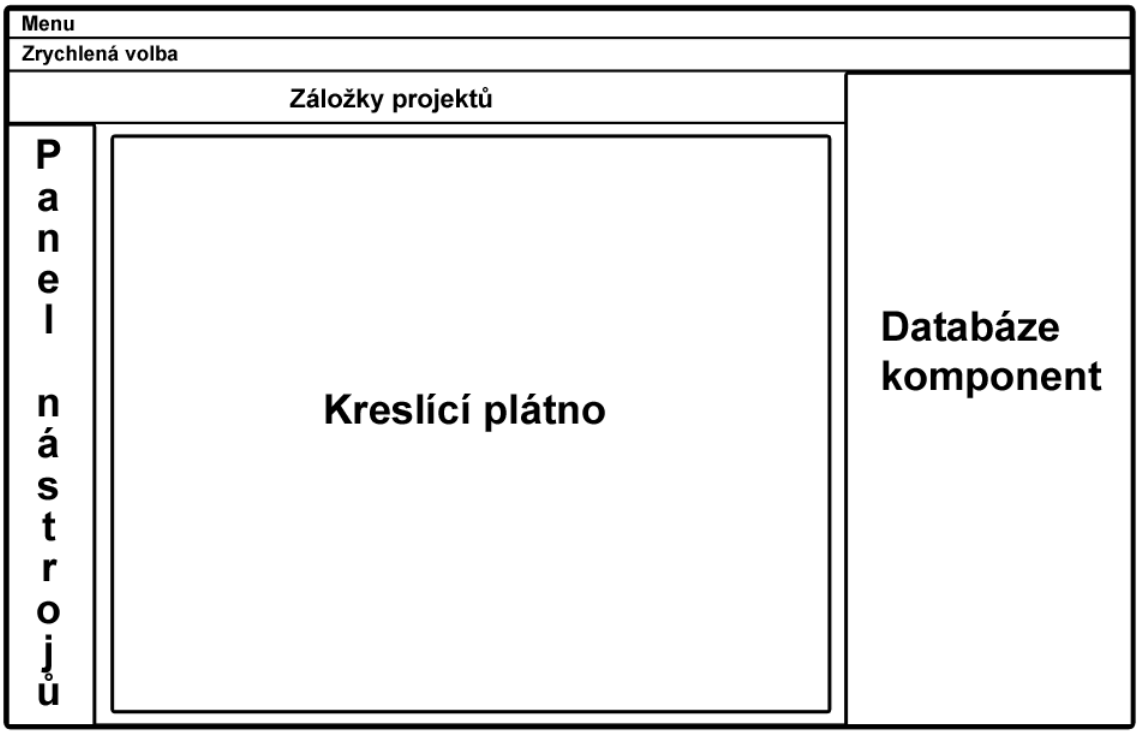
## 5.2 Specifikace

Modul bude psán v jazyce C/C++ za použití Qt knihovny, která tvoří grafický framework. Pro snadnou obsluhu bude modul obsahovat menu a toolbar (zrychlená volba). Menu a zrychlená volba zajistí práci s projekty, možnost nastavení a v případě nouze zavolání nápovědy. Jelikož FPGA na FITkitu doposud obsahuje tři předdefinované top-level entity, bude potřeba vytvořit popis, který definuje jednotlivé typy top-level entit. Při genrování VHDL šablony se bude volit typ top-level entity, pro kterou bude aplikace navrhována. Dynamická databáze komponent taktéž potřebuje vytvořit popis pro editaci či přidání vlastní komponenty. Vytvořené projekty budou reprezentovány záložkami pro intuitivní obsluhu. Rozpracované projekty bude možno ukládat ve formě souborů a do každého z těchto souborů bude zapsána vnitřní struktura obvodu. Načtení rozpracovaného projektu opět vytvoří uloženou strukturu. Každá záložka bude obsahovat kreslící plátno definované bitmapou (obrázkem), do které bude zakresleno navržené schéma obvodu. Ovládání návrhu schématu se bude realizovat pomocí myši, kdy vzniklé údalosti od myši budou zakreslovat návrh do definované bitmapy. Komponenty bude možno propojovat vkládáním spojnic (úseček). Je kladen důraz na vnitřní kontrolu datové šířky spojů. Nastavení konfigurace komponenty bude za pomoci dialogového okna, které bude možno dynamicky měnit vlastním popisem. Výsledné navržené schéma bude možno exportovat jako obrázek a výstupem celého návrhu bude vygenerovaná šablona v jazyce VHDL pro FPGA na FITkitu. Veškeré potřebné po-

pisy budou navrženy ve standardu XML, pro které již existuje parser a nebude nutné tvořit vlastní. Další pozitivum použití XML spočívá v snadném pochopení struktury a možnosti rychlé editace.

### Grafický editor

Rozbor existujících nástrojů přispěl ke konečnému vzhledu modulu, kde jeho schématický návrh můžeme vidět na obrázku 5.1. Základ tvoří klasické menu a pod ním zrychlená volba. Jednotlivé projekty budou řazeny za sebou jako záložky. Každá záložka bude obsahovat kreslící plátno. Databáze komponent a panel nástrojů bude společný pro všechny záložky, protože se nemění.




Obrázek 5.1: Návrh Designeru pro QDevKit.

### Popis platformy

Jak už bylo zmíněno výše, FPGA na platformě FITkit obsahuje tři předdefinované top-level entity, a proto je potřeba před vytvořením projektu v Designeru pro QDevKit zvolit typ entity, pro kterou bude aplikace navržena. Z toho vyplývá, že je nezbytné navrhnout popis pro jednotlivé předdefinované top-level entity. Popis bude psán ve formátu XML a bude vycházet z popisu platformy nástroje Altium Designer.

Každý XML standard začíná položkou, která označuje verzi použitého XML a kódování psaného textu. Komentáře máme blokové a píšeme je jako v jazyce HTML. Začátek komentáře označíme sekvencí `<!--` a konec komentáře značíme `-->`.

Popis fyzické struktury platformy bude uvozen mezi klíčová slova `<platform>` a `</platform>`. Klíčové slovo `<platform>` bude obsahovat parametry:

- `name` - název platformy,
- `architecture` - typ top-level entity, např. tlv_bare_ifc.

Jednotlivé přiřazení signálů a pinů se provede v položce `<record>`, která bude mít následující parametry:

- `name` - název signálu,
- `direction` - směr (vstup nebo výstup),
- `type` - typ signálu s datovou šířkou,
- `width` - datová šířka.

Příklad krátkého popisu platformy:

```
<!--
######################################################################
##
## Platforma bare
##
######################################################################
-->

<platform name="bare" architecture="tlv_bare_ifc" >
     <!-- hodiny -->
     <record name="SMCLK" direction="in" type="std_logic" width="1"/>
     <!-- klavesnice 4x4 -->
     <record name="KIN" direction="in" type="std_logic_vector(3 downto 0)"
      width="4"/>
</platform>
```

### Popis komponenty

Výsledný modul bude disponovat dynamickou databází komponent, a proto je potřeba vytvořit speciální popis pro rozhraní komponenty.Cílem je snadná editace již existující nebo vytvoření úplně nové komponenty. Popis bude psán ve formátu XML a bude vycházet z kombinace popisů nástrojů Altium Designer a Xilinx EDK.

Celá databáze bude uvozena mezi klíčová slova `<database>` a `</database>`. Rozhraní jednotlivých komponent bude popsáno mezi dvojicí slov `<component>` a `</component>`. Výskyt `<component>` bude obsahovat dva povinné atributy a jeden nepovinný. Prvním povinným atributem je `name`, který určuje název komponenty. Druhým je `code`, který udává absolutní nebo relativní cestu k SVN repozitáři nebo adresáři, ve kterém je umístěn VHDL kód nebo package.xml, který přidáme do projektu pomocí include pro správnou funkci navrženého obvodu. Jako volitelný může být uveden atribut `category`, který určuje kategorii, do které bude komponenta přiřazena v databázi komponent modulu.

V rozhraní komponenty mohou být definovány generické parametry. Uveďme si příklad:
`<generic default="4" values="1...24" name="ADDR_OUT_WIDTH" type="integer"/>`

V našem popisu mohou generické parametry obsahovat tyto atributy:

- `default` - přednastavená hodnota,
- `values` - hodnoty, kterých může nabývat, navzájem oddělené čárkou nebo vytvořit zkrácený výčet za pomoci třech po sobě jdoucích teček,
- `name` - název parametru,
- `type` - datový typ parametru.

Každý generický parametr se vždy vztahuje k některému ze signálů, kde je potřeba vložit do parametru signálu `widthtype` jméno daného generického parametru. Více v níže uvedené tabulce 5.1.

Za generickými parametry bude následovat popis signálů. Pokud budeme potřebovat signály sloučit do stejné skupiny, zapíšeme je mezi dvojici klíčových slov `<port>` a `</port>`, jinak signál není v žádném kontextu s jinými signály a je osamocen.

Jednotlivé signály definuje klíčové slovo `<signal/>`. Výčet parametrů, kterých může signál nabývat a jejich podrobnější popis, je uveden v tabulce 5.1.

| **Parametr** | **Význam** | **Hodnoty** | **Význam** |
|---|---|---|---|
| name | název | string | řetězec |
| class | druh | clock<br>reset<br>data<br>control | hodinový signál<br>signál reset<br>data<br>řídící signál |
| direction | směr | in<br>out<br>inout | vstup<br>výstup<br>obousměrný |
| sensitivity | aktivace | high<br>low<br>raisingedge<br>fallingedge<br>bothedges | aktivní v log. 1<br>aktivní v log. 0<br>reakce na náběžnou hranu<br>reakce na sestupnou hranu<br>reakce na náběžnou i sestupnou hranu |
| type | datový typ | std_logic<br>std_logic_vector | signál reprezentuje jeden vodič<br>signál reprezentuje více vodičů |
| width | datová šířka | integer | počet vodičů signálu |
| widthtype | typ datové šířky | fixed<br>generic | pevná šířka<br>šířku udává generický parametr |

Tabulka 5.1: Popis parametrů signálů

Příklad použití signálu:
`<signal direction="in" name="CLK" type="std_logic" width="1" widthtype="fixed"/>`

### Popis vlastního dialogového okna pro konfiguraci komponenty

Pro flexibilnost modulu bude potřeba navrhnout, jak vytvořit vlastní konfigurovací okno pro nastavení komponenty, když si nevystačíme se standadrní nabídkou. Půjdeme stejnou cestou, jako šla společnost Xilinx. Vlastní okno popíšeme pomocí některého ze skriptovacích jazyků a pokud bude potřeba toto okno zavolat, spustí se speciální interpret. V našem případě bude modul implementován za pomoci knihovny Qt, kde se přímo nabízí možnost vytvořit návrh vlastního okna ve skriptovacím jazyce PythonQt. Poté jen Designer pro QDevKit spustí příslušný interpret, který z daného popisu vytvoří okno.

Vlastní dialogové okno reprezentuje skript napsaný v PythonQt. Jednou z možností, jak můžeme vlastní dialogové okno přiřadit ke komponentě, je přímo vložit kód skriptu okna do popisu rozhraní komponenty mezi klíčová slova `<dialog>` a `</dialog>`. Další možností je přidání položky `<dialog />` do popisu rozhraní komponenty s atributem `file`, který obsahuje absolutní nebo relativní cestu ke skriptu.

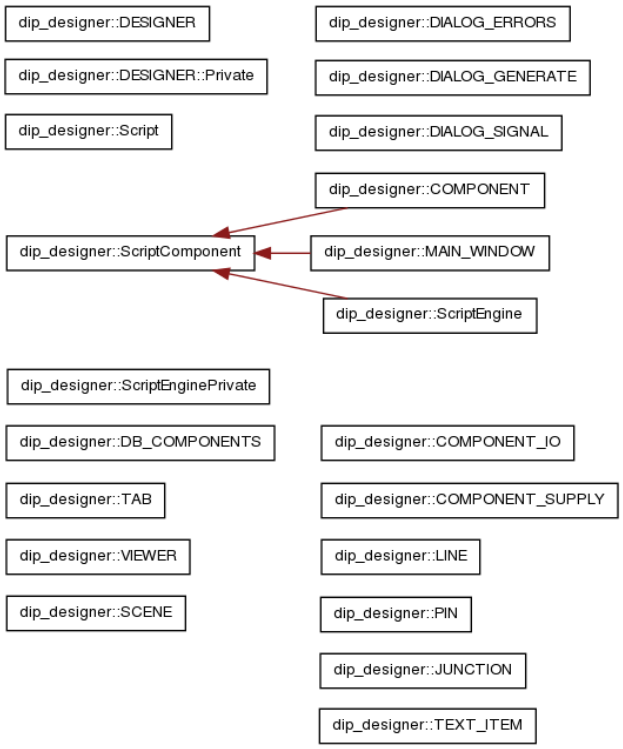
# Kapitola 6

# Implementace a ovládání designeru

Jako implementační jazyk byl zvolen jazyk C++ a Qt framework jako knihovna s grafickými objekty, jak je uvedeno výše. Tato kapitola je věnována popisu implementace a řešení problémů v průběhu práce.

## 6.1 Struktura projektu

Jelikož řešení projektu obsahuje velké množství tříd, je potřeba provést rozbor podrobněji. Hirearchie tříd je uvedena na obrázku 6.1. Podrobná projektová dokumentace je k dispozici ve formě HTML, kterou generuje aplikace doxygen. Zobrazíme ji pomocí webového prohlížeče spuštěním souboru index.html ve složce designer/doc/html na přiloženém DVD.




Obrázek 6.1: Hirearchie tříd

### 6.1.1 DESIGNER

Jedná se o základní třídu, kde její hlavní činností je vytvoření a řízení aplikace. V konstruktoru třídy se vytvoří jak hlavní okno aplikace, tak i nezbytně nutný skriptovací modul pro zpřístupnění objektů z designeru do externě volaných skriptů, v našem případě PythonQt. Tento modul je velmi důležitý pro možnost volby vlastního dialogového okna k nastavení komponenty. Více o této problematice v sekci 6.2. O obsluhu této třídy se stará třída QObject.

### 6.1.2 MAIN_WINDOW

Tato třída tvoří jednu z nejdůležitějších částí modulu, a to objekt hlavního okna. Tato třída se stará o veškerou interakci designeru s uživatelem a o vykreslení základních částí designeru. Obsahuje hlavní menu se základní prací s aplikací a projektem, toolbar nebo-li zrychlenou volbu pro práci s aplikací, toolbox nebo-li panel nástrojů komponent, databázi komponent, PythonQt konzoli, stavový řádek zobrazující zprávy při nějaké události, a to nejdůležitější panel záložek, do kterého se přidávají jednotlivé záložky (nové projekty).

**Hlavní okno** je děděno z třídy QMainWindow pro správu nad všemi grafickými objekty a ScriptComponent jako zpřístupnění zaregistrovaných objektů pro externí skripty.

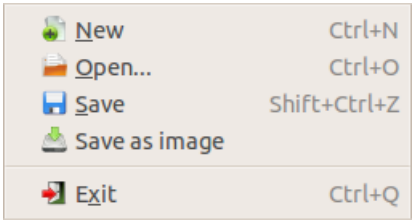
**Hlavní menu** je třídy QMenu a obsahuje položky:

- File - nabízí základní práci s projektem, viz obrázek 6.2:
  - New - vytvoří novou záložku (nový projekt).
  - Open - vytvoří novou záložku (rozpracovaný projekt).
  - Save - uloží rozpracovaný projekt jako XML soubor.
  - Save as image - uloží kreslící plátno jako obrázek.
  - Exit - ukončí aplikaci.
- Edit - nabízí editaci projektu, viz obrázek 6.3:
  - Undo - krok zpět při práci v projektu.
  - Redo - krok vpřed při práci v projektu.
  - Color pen - změna barvy pera.
- Item - nabízí volbu komponenty z panelu nástrojů, viz obrázek 6.4:
  - Line - volba kreslení spojnice.
  - In - volba vložení komponenty vstupu.
  - Out - volba vložení komponenty výstupu.
  - Vcc - volba vložení komponenty simulující připojení na napájení.
  - Gnd - volba vložení komponenty simulující připojení na zem.
  - Junction - volba vložení komponenty uzlu.
  - Text - volba vložení komponenty textu.
  - Select or move - výběr nebo pohyb komponentou ve scéně.

  - Delete - vyjme označenou komponentu ze scény.

- Display - zobrazení databáze komponent nebo PythonQt konzole:
  - Database components - zobrazí nebo schová panel databáze komponent.
  - PythonQt console - zobrazí nebo schová panel PythonQt konzole.

- Help - zobrazí nápovědu nebo informace o aplikaci:
  - Manual - zobrazí předem vytvořenou nápovědu.
  - About - zobrazí základní údaje o aplikaci.
  - About Qt - zobrazí základní údaje o Qt frameworku.




Obrázek 6.2: Ukázka menu a položky File

**Podrobnější rozbor položek menu** - jelikož menu obsahuje větší počet položek, kde některé položky provádí složitější úkony, je proveden jejich popis podrobněji.

**New** - položka menu, která vytvoří novou záložku třídy TAB a přidá ji do panelu záložek. Do záložky je vložen objekt prohlížeč scény, o který se stará třída VIEWER a ta v konstruktoru vytvoří scénu třídy SCENE, která se následně stará o veškeré kreslení.

**Open** - tato volba je poněkud složitější, nejdříve vytvoří novou záložku, dojde k jejímu naplnění komponentami a přidání do panelu záložek. Poté se získají data ze zvoleného XML souboru a naplní se scéna objekty ze souboru. Zvolený XML soubor má podobnou strukturu jako popis komponenty 5.2, který obsahuje navíc komponenty z panelu nástrojů, pozici komponenty ve scéně a její možné nastavení. Proto klíčové slovo `<component>` obsahuje další atribut a to `pos`, který udává pozici ve scéně.

**Save** - uloží rozdělaný projekt, jako XML soubor uvedený v open. Postupně prochází scénu, vytváří XML strukturu, kterou následně uloží na pevný disk.

**Save as image** - uloží kreslící plochu jako obrázek (bitmapu).

**Undo** - krok zpět při práci ve scéně. Jednotlivé úkony při práci ve scéně se ukládají do zásobníku třídy QUndoStack a pomocí speciálních tříd, definovaných pro každý zaznamenávaný úkon, můžeme provést krok zpět či vpřed.

**Redo** - krok vpřed při práci ve scéně.

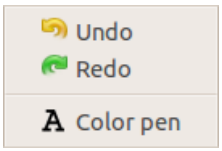
**Items** - jedná se o volby panelu nástrojů uvedeny v odstavci **??**.

**Database components** - akce panelu databáze komponent pro zobrazení či schování panelu.

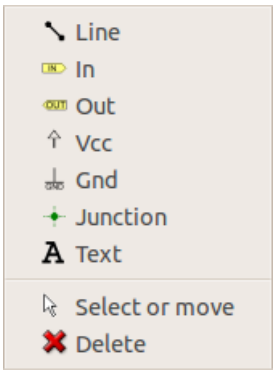
**PythonQt console** - akce panelu PythonQt konzole pro zobrazení či schování panelu.

**Manual** - zobrazí nápovědu ve formátu HTML za pomoci třídy QAssistantClient. Aby bylo možno nápovědu správně zobrazit s možností vyhledávání, je potřeba vyplnit soubory v XML formátu s koncovkou `qhcp` a `qhp` a vytvořit soubor typu SQLite pomocí příkazu `qcollectiongenerator help.qhcp -o help.qhc` v terminálu.

**About** - pomocí QMessageBox zobrazí základní informace o aplikaci a autorovi.
**About Qt** - pomocí QMessageBox::aboutQt zobrazí základní informace o Qt frameworku.



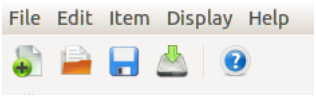

Obrázek 6.3: Ukázka menu a položky Edit




Obrázek 6.4: Ukázka menu a položky Item

**Zrychlená volba (toolbar)** je třídy QToolBar a je pevně umístěna pod menu, viz obrázek 6.5. Jedná se o rychlou volbu, kde kliknutím na danou ikonu můžeme přímo provádět základní práci s projektem. Obsahuje:

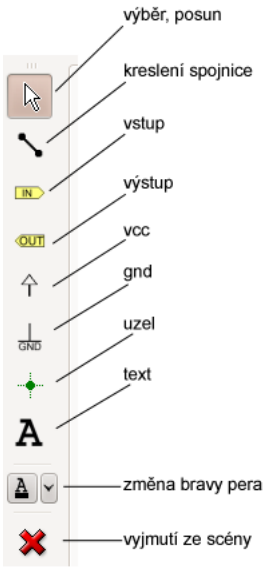
- New - nový projekt.
- Open - otevře rozpracovaný projekt
- Save - uloží projekt.
- Save as image - uloží projekt jako obrázek.
- Manual - zobrazí nápovědu.




Obrázek 6.5: Zrychlená volba (toolbar)

**Panel nástrojů (toolbox)** je také třídy QToolBar, viz obrázek 6.6. Toolbox je přidán do hlavního okna volně, tudíž může měnit polohu. Jedná se o panel nástrojů, kde kliknutím na danou ikonu můžeme měnit typ vkládané komponenty nebo provádět výběr, posun, vyjmutí či změnu barvy písma. Obsahuje:

- Pointer - výběr, posun komponenty.
- Line - kreslení spojnice mezi komponentami.
- In - po kliknutí vloží do scény komponentu vstup.
- Out - po kliknutí vloží do scény komponentu výstup.
- Vcc - po kliknutí vloží do scény komponentu napájecí napětí (log. 1).
- Gnd - po kliknutí vloží do scény komponentu zem (log. 0).
- Junction - po kliknutí vloží do scény komponentu uzlu (spojnice více komponent).
- Text - po kliknutí vloží do scény komponentu text.
- Set color pen - změna barvy pera.
- Delete - vyjme označené komponenty ze scény.




Obrázek 6.6: Panel nástrojů (toolbox)

**Panel záložek** je třídy QTabWidget, do kterého se přidávají jednotlivé záložky třídy TAB (více v podsekci 6.1.4). Pro přidání záložky je potřeba vytvořit nový objekt třídy TAB, přiřadit záložce jméno a obsluhu události pro změnu aktuální záložky, kdy je potřeba přepočítat a zobrazit chyby.

**Panel databáze komponent** je panel třídy DB_COMPONENTS (více v podsekci 6.1.3), který má za úkol zobrazit databázi komponent ve stromové struktuře, umožnit vkládání komponent do scény projektu, dále vytvořit náhled na schématickou značku komponenty při jejím výběru a v neposlední řadě informovat o chybách v rozpracovaném projektu a umožnit generování projektu pro platformu FITkit dle zadané top-level entity.

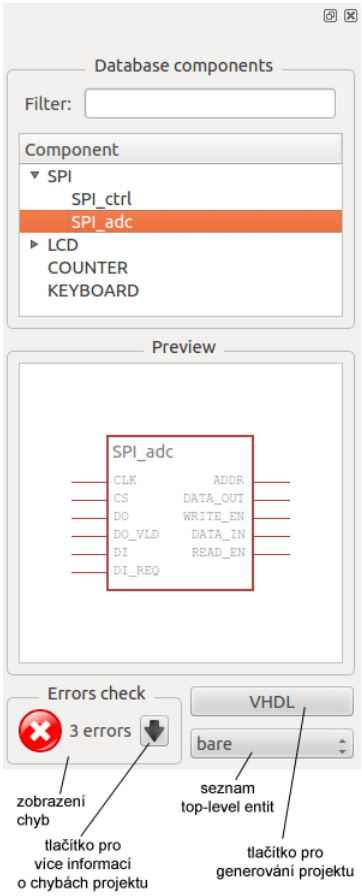
**Panel PythonQt konzole** je panel třídy QDockWidget, viz obrázek 6.7, tudíž s ním můžeme pohybovat po hlavním okně a také ho můžeme zobrazit či skrýt. Panel obsahuje PythonQt konzoli třídy PythonQtScriptingConsole a tlačítko clear, které maže obsah konzole. Je vhodná pro testování PythonQt skriptů.


Obrázek 6.7: Ukázka PythonQt konzole

**Stavový řádek** je třídy QStatusBar a má za úkol zobrazit různá hlášení při interakci s programem.

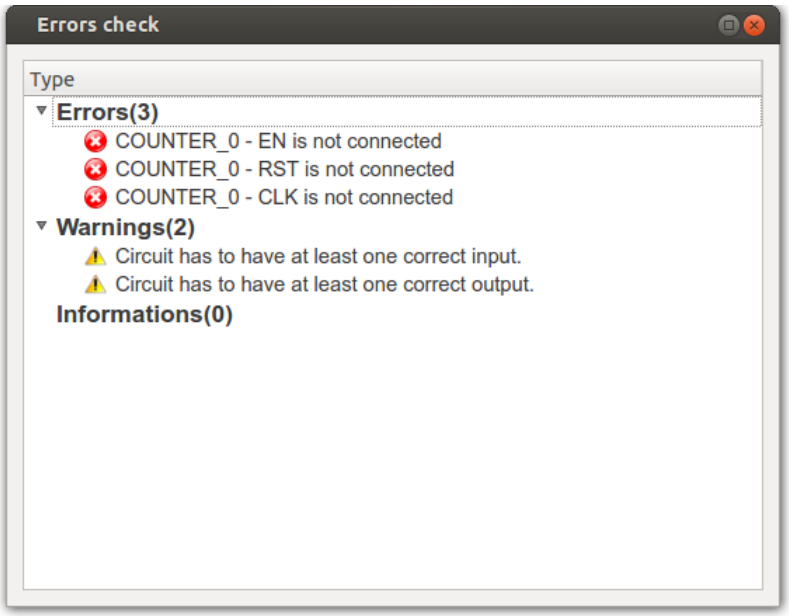
### 6.1.3 DB_COMPONENTS

Je panel, o který se stará třída QDockWidget. QDockWidget umožňuje zobrazit či skrýt panel nebo s ním pohybovat po hlavním okně, viz obrázek 6.8.



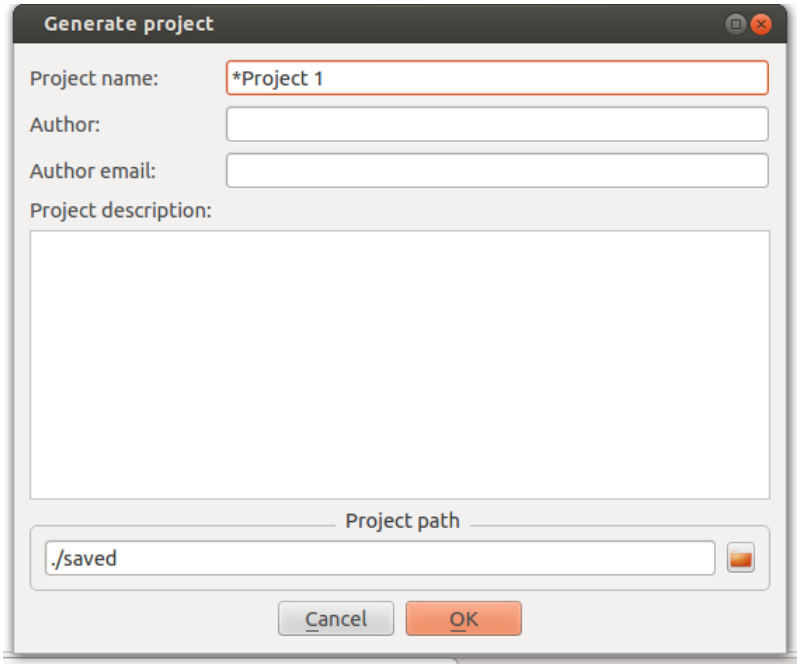

Obrázek 6.8: Panel databáze komponent

V konstruktoru třídy dojde k načtení všech XML souborů z adresáře `db_components`, které reprezentují popis komponent. Každou správně načtenou komponentu uložíme do zásobníku komponent, reprezentovaného datovou strukturou QMap, kde je vždy ke jménu komponenty přiřazen objekt třídy COMPONENT (více v podsekci 6.1.6). Jednotlivé komponenty jsou zobrazeny ve stromové struktuře prvku QTreeView. Kliknutím levého tlačítka myši na název komponenty se zobrazí náhled na schématickou značku komponenty. Dvojím kliknutím levého tlačítka myši na název komponenty ji vložíme do scény, pokud je otevřena nějaká záložka. Následně načteme všechny XML soubory z adresáře `platforms`, které reprezentují popis top-level entit FPGA. Každou správně načtenou top-level entitu uložíme do zásobníku platforem, reprezentovaného datovou strukturou QMap, kde je vždy ke jménu entity přiřazen zásobník signálů datové struktury QMap. Zásobník signálů obsahuje jméno signálu a jemu odpovídající XML popis, reprezentovaný strukturou QDomNode. Jednotlivé typy entit jsou uloženy do seznamu a zobrazeny v prvku QComboBox. Zvolený typ entity zpřístupní dané signály entity komponentě COMPONENT_IO, která reprezentuje vstup nebo výstup obvodu (více v podsekci 6.1.8). Panel dále obsahuje informace o počtu chyb ve scéně aktuální záložky. Pouze se zobrazuje počet chyb nebo varování. Pro bližší informace kliknutím na tlačítko se šipkou se zobrazí dialogové okno třídy DIALOG_ERRORS, děděné z třídy QDialog, viz obrázek 6.9, který má jako parametry tři seznamy QList a ve stromové struktuře zobrazí chyby, varování či důležité informace. Obsah těchto seznamů je aktuali-




Obrázek 6.9: Dialogové okno chyb v projektu

zován při jakékoliv provedené akci ve scéně aktuální záložky. Panel obsahuje poslední velmi důležitý prvek, a to tlačítko generování projektu pro platformu FITKit. Po kliknutí levým tlačítkem myši na tlačítko s popisem VHDL dojde k zobrazení dialogového okna třídy DIALOG_GENERATE, děděného z třídy QDialog, viz obrázek 6.10. Zde je nutné vyplnit název projektu a cestu, kam má být projekt uložen (výchozí cesta je podadresář saved v adresáři src designeru). Po potvrzení dialogu je vytvořen adresář se jménem projektu v uvedené cestě. Do tohoto adresáře jsou vytvořeny dva podadresáře `fpga` a `mcu`, kde podadresář fpga obsahuje vygenerovaný VHDL kód projektu. Podadresář mcu obsahuje soubor `main.c` s potřebným zdrojovým kódem v jazyce C, obsahující firmware, tedy kód pro mikrokontro-




Obrázek 6.10: Dialogové okno generování projektu pro platformu FITkit

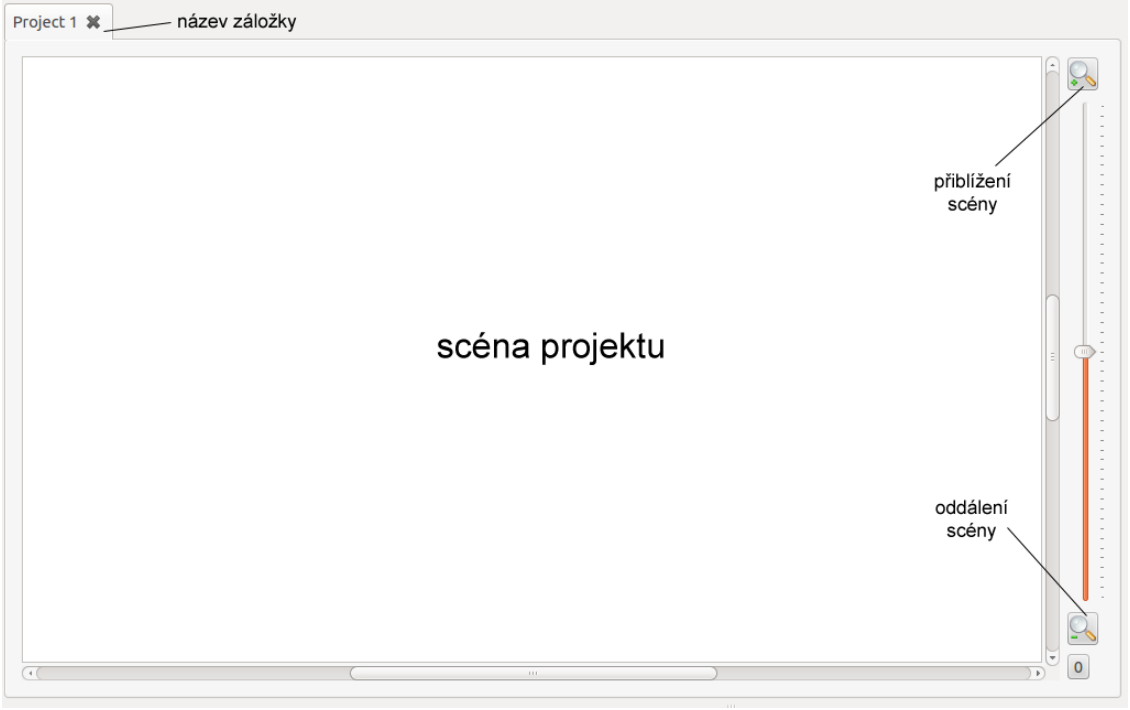
ler. Vedle těchto podadresářů najdeme v adresáři projektu XML soubor, který obsahuje informace o projektu uvedené v dialogovém okně generování projektu, ale také zejména cesty ke zdrojovým kódům použitých komponent v projektu.

Generování VHDL kódu probíhá tak, že z aktuální záložky získáme ukazatel na scénu a ta vrátí seznam všech vložených objektů. V seznamu nás zajímají pouze komponenty typu COMPONENT. Každá z těchto komponent obsahuje vstupní a výstupní signály dle jejich popisu komponenty, které reprezentuje třída PIN (více v podsekci 6.1.7). Výsledný vygenerovaný soubor se jmenuje `top.vhd`.

Vlastní generování probíhá ve dvou průchodech získaným zásobníkem komponent. Nejdříve je ale potřeba získat typ top-level entity, ten vložíme do typu architektury pojmenovanou `main` jako například: `architecture main of tlv_bare_ifc is`. Poté následuje první průchod seznamem komponent, který generuje jejich rozhraní, kde se uvádí pouze jeden typ komponenty, pokud v projektu máme více komponent stejného typu. Rozhraní obsahuje položku `generic`, která se naplní pomocí zásobníku generických parametrů, pokud nějaké parametry komponenta obsahuje. Za ní následuje položka `port`, kde průchodem zásobníku pinů komponenty získáme vstupní a výstupní signály komponenty. Nyní za klíčovým slovem `begin` následuje část propojení jednotlivých komponent. Druhý průchod zásobníku komponent již bere navědomí všechny komponenty typu COMPONENT. Každá z komponent nejdříve generuje své jméno a následně typ komponenty, který se odkazuje na použité typy komponent v předchozí sekci rozhraní. Poté je pro každý pin komponenty rekurzivně prohledávána cesta propojení přes interní ukazatele spojení, dokud nenarazíme na pin, vstup/výstup nebo napájecí napětí či zem. Pokud je v cestě více než jedna spojnice typu LINE, je potřeba vytvořit pomocný signál, který se bude jmenovat stejně jako spojnice. Generování VHDL kódu končí uzavřením entity pomocí klíčových slov `end main;`.

### 6.1.4 TAB

Další velmi důležitou část aplikace představuje záložka reprezentující nový projekt, viz obrázek 6.11. O obsluhu se stará třída QWidget do které se vloží prohlížeč scény třídy VIEWER a vlastní scéna třídy SCENE (více v podsekci 6.1.5). Další důležitou funkcí záložky je počítání chyb v projektu, kdy scéna vrátí seznam vložených objektů. U těchto objektů se kontroluje připojení vstupů a výstupů, zda-li jsou vůbec někam připojeny, na který signál a s jakou datovou šířkou a případně jejich správné nastavení. Chyby, varování a důležité informace se ukládají do třech seznamů zmíněných výše. Dále má za úkol vkládání objektů do scény a načtení či uložení scény.




Obrázek 6.11: Ukázka záložky (nového projektu)

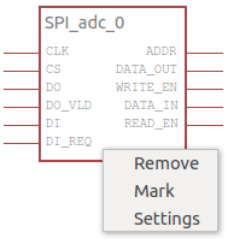
### 6.1.5 VIEWER, SCENE

Tato dvojice tříd tvoří základ záložky TAB. Třída VIEWER vytvoří QWidget s prohlížečem scény třídy QGraphicsView a tlačítka s posuvníkem umožňující přiblížení či oddálení scény. Prohlížeči je přiřazena scéna třídy SCENE, vytořená v konstruktoru třídy TAB. SCENE je děděnou třídou z QGraphicsScene. Má na starosti veškeré vykreslování tvořeného schématu v projektu. Jedná se o velmi inteligentní datovou strukturu, která si dokáže uchovat informace o vložených komponentách. Hlavním úkolem třídy je vkládat nové komponenty dle volby panelu nástrojů, překreslovat scénu dle vzniklých událostí a vracet seznam komponent ve scéně.

### 6.1.6 COMPONENT

Jedná se o třídu, která reprezentuje komponentu z databáze komponent, kde její základ tvoří třída QGraphicsItem. Proto, aby bylo umožněno zpřístupněnit obsah třídy externímu skriptu v PythonQt, dědí vlastnosti tříd QObject a ScriptComponent.

Konstruktor třídy dostává jako parametry seznamy obsahující názvy signálů, které reprezentují pravé a levé piny komponenty. Tyto seznamy jsou vytvořeny při parsování databáze komponent. Stranu signálu (pinů) určuje převažující počet vtupních či výstupních směrů signálů v portu nebo pokud nejsou sdruženy, tak pravá strana představuje vstupní orientaci signálu a levá výstupní. Další parametry tvoří jméno komponenty, pozice ve scéně a jako nejdůležitější celou XML část popisu dané komponenty v datové struktuře QDomElement. Z tohoto popisu v konstruktoru třídy parsujeme generické parametry, které jsou uloženy do zásobníku typu QVariantMap, kde je k názvu generického parametru přiřazen další zásobník typu QVariantMap, který obsahuje datový typ parametru, možné hodnoty nastavení a výchozí hodnotu. Pokud při parsování dojde k nalezení položky dialog, dojde k rozlišení, pokud se jedná o externí skript nebo přímo zdrojový kód. Pokud se jedná o zdrojový kód, je potřeba si dát pozor na python syntaxi a dodržovat odsazení přímo tabulátorem. Je-li v opačném případě zdrojem vlastního dialogového okna externí skript, načte se cesta ke skriptu a stejně jako v předchozím případě dojde k uložení do proměnné typu QString.

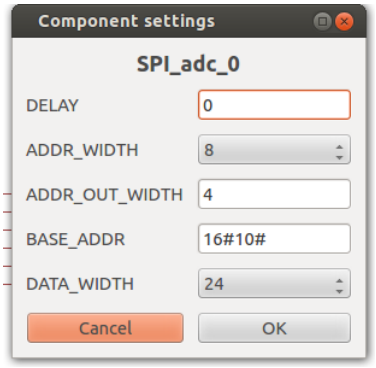



Obrázek 6.12: Příklad komponenty s kontextovým menu

Dvojklikem levým tlačítkem myši na název komponenty ve stromové struktuře databáze komponent dojde k vložení komponenty do scény. Poté se zavolá speciální metoda `drawSignals()`, která projde oba seznamy signálů uvedených výše a pro každý signál (pin) vytvoří objekt typu PIN (více v podsekci 6.1.7) a vykreslí požadovaný pin komponenty. Ten potom slouží k vytváření propojovacích cest ve scéně. Při vkládání do scény se automaticky komponentě přiřadí jméno dle typu komponenty s unikátním číslem a také se vytvoří objekt třídy TEXT_ITEM (více v podsekci 6.1.8), kde je možné s názvem libovolně pohybovat po scéně nebo jej změnit za jiný.

Pokud máme v panelu nástrojů označenou volbu výběru či pohyb, táhnutím levého tlačítka myši označíme objekty ve scéně a můžeme s nimi buď pohybovat nebo je ze scény vymazat klávesou DEL nebo přes tlačítko s křížkem v panelu nástrojů.

Klinutím pravého tlačítka myši na komponentu dojde k vyvolání události typu QGraphicsSceneContextMenuEvent a zobrazí se kontextové menu, viz obrázek 6.12. Zde máme na výběr buď to označit nebo smazat komponentu, ale to nejdůležitější je, že můžeme zavolat nastavovací dialogové okno, pokud má komponenta nějaké generické parametry, viz obrázek 6.13. Pokud žádné parametry nemá, položka v menu není k dispozici.




Obrázek 6.13: Dialogové okno pro nastavení komponenty

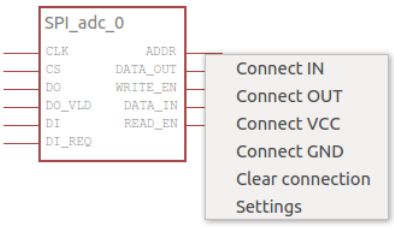
Po potvrzení nastavovacího okna dojde k zavolání speciální metody `pythonQtSetGenParam(name, value)` pro každý z generických parametrů, kde se do zásobníku generických parametrů uloží pro jméno parametru name hodnota value.

### 6.1.7 PIN

Jak již bylo zmíněno výše, jedná se o třídu, která představuje jeden pin (signál) komponenty. Konstruktor dostává jako parametr jméno signálu, směr signálu (určuje, zda-li je signál negovaný), stranu komponenty, na které má být vykreslen, od které se dále získává počáteční a koncový bod pinu, abychom věděli, kam připojit další komponenty.

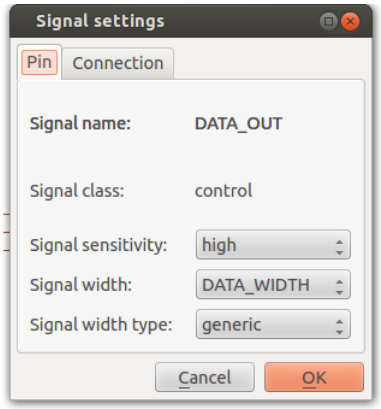
Při pohybu komponentou se aktualizuje pozice pinu a pokud se objeví v blízkosti některé komponenty, tak podle typu komponenty se buď sama přitáhne nebo se propojí spojnicí. Pokud máme na pinu připojenou nějakou komponentu a provedeme s ní pohyb, dojde k automatickému natažení spojnice od pinu k připojené komponentě.

Klinutím pravého tlačítka myši na komponentu dojde k vyvolání události typu QGraphicsSceneContextMenuEvent a zobrazí se kontextové menu, viz obrázek 6.14. Kontextové



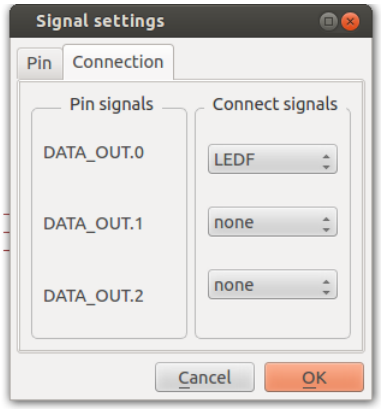

Obrázek 6.14: Příklad komponenty s kontextovým menu pro pin

menu nabízí možnost připojení komponent vstup/výstupu (více v podsekci 6.1.8) nebo napájecího napětí či země (více v podsekci 6.1.8) na pin nebo vyvolání nastavovacího dialogového okna. Nastavovací okno má dvě záložky. Záložka Pin umožňuje nastavit směr pinu nebo jeho datovou šířku, viz obrázek 6.15. Po potvrzení dojde ke změně nastavených hodnot přímo v XML popisu mateřské komponenty.




Obrázek 6.15: Dialogové okno pro nastavení pinu

Druhá záložka Connection nabízí různé možnosti nastavení připojení komponent na pin, viz obrázek 6.16. Po nastavení a potvrzení nastavovacího okna se možnosti připojení uloží




Obrázek 6.16: Dialogové okno pro nastavení připojení signálů na pin

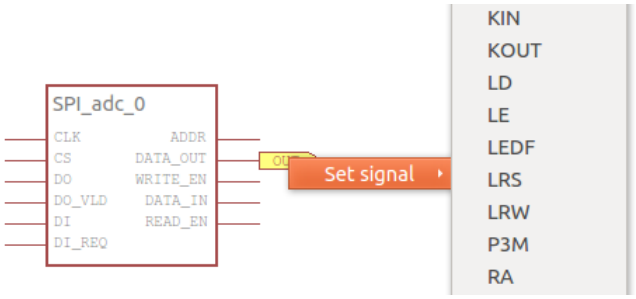
do zásobníku, ze kterého se pak generuje výsledný VHDL kód. Pokud je na pin připojená komponenta stejné datové šířky, automaticky se doplní připojení 1:1. Pokud jsou šířky rozdílné, musíme připojení ručně nastavit, jinak se zobrazí chyba.

### 6.1.8 Další komponenty

Pro možnost komplexního řešení tvorby schémat obvodů je potřeba dalších nezbytně nutných komponent. S každou komponentou lze samozřejmě také pohybovat. Pokud se komponenta připojí na některou z dalších komponent, tak se při pohybu automaticky natáhne spojnice a přenastaví se ukazatele z komponent na spojnici. Spojnice pak ukazuje na obě komponenty. Pokud připojujeme komponentu na místo, kde již nějaké spojení dvou komponent je, pak se vytvoří uzel, který poté bude ukazovat na všechny připojené komponenty a každá z jednotlivých komponent zase bude ukazovat na uzel. Pokud chceme vyjmout komponentu ze scény, musíme nejdříve přenastavit ukazatele na `NULL` a pak ji teprve vyjmout ze scény. Obecný princip napojení komponent je, že kliknutím levým tlačítkem myši do blízkosti jiné komponenty dojde k automatickému posunu nebo vytvoření spojnice.

### COMPONENT_IO

Komponenta představující vstup nebo výstup obvodu je děděná z třídy QGraphicsItem. V konstruktoru uvedeme typ, jestli se jedná o vstup nebo výstup a podle strany pinu, na který je tato komponenta připojena, určíme směr šipky. Abychom věděli, na který signál se má vstup nebo výstup napojit, klikneme na něj pravým tlačítkem myši a zobrazí se kontextové menu vyvolané událostí typu QGraphicsSceneContextMenuEvent, viz obrázek 6.17. Kontextové menu obsahuje signály dle vybrané top-level entity. Po kliknutí na jméno




Obrázek 6.17: Příklad výstupu s kontextovým menu

signálu dojde k načtení signálu z XML top-level entit a celý záznam o signálu se uloží do členské proměnné typu QDomNode a vloží se do scény jeho jméno jako objekt TEXT_ITEM.

### COMPONENT_SUPPLY

Komponenta představující napájecí napětí nebo zem je také děděná z třídy QGraphicsItem. Napájecí napětí představuje připojení logické 1 a zem logické 0.

### LINE

Jedná se o komponentu třídy QGraphicsItem, která představuje spojnici mezi objekty ve scéně. Kliknutím levým tlačítkem myši do scény se jednotlivé body vkládají do zásobníku typu QVector, ale nejdříve před vložením dojde k zaokrouhlení pozice bodů, aby se spojnice zalamovala do pravého úhlu. Při vykreslování se projde celý zásobník, kde se propojí každá dvojice bodů, což ve výsledku vytvoří spojnici. Spojnici je při vložení do scény automaticky vygenerováno jméno, které je možné změnit kliknutím pravého tlačítka myši, kde se zobrazí klasický textový dialog. Změna jména se projeví vložením TEXT_ITEM s upraveným jménem do scény.

### JUNCTION

Uzel je prvkem třídy QGraphicsItem, který vytváří místo spojení, kde se setkávají více jak dvě komponenty.

### TEXT_ITEM

Komponenta, představující textový prvek, je třídy QGraphicsTextItem. Jak bylo uvedeno několikrát výše, může být použit k pojmenování komponent nebo jako libovolný textový prvek ve scéně. U tohoto prvku můžeme měnit barvu změnou barvy pera v panelu nástrojů.

## 6.2 Skriptovací modul(PythonQt)

Celý skriptovací modul se skládá ze tří důležitých tříd převzatých z QDevKitu a to ScriptEngine, Script, ScriptComponent a v neposlední řadě ze skriptu generic_settings.py napsaného v PythonQt, který tvoří dialogové okno a ukládá změny.

### 6.2.1 SkriptEngine

Tvoří základ modulu, který obsahuje zásobník registrovaných objektů, kde jsou jednotlivé objekty následně zpřístupněny pro externí skripty. V konstruktoru třídy dojde k zpřístupnění PythonQt interpretu celé aplikaci. Třída obsahuje několik metod, které si rozebereme podrobněji.

**Script* load(const QString& file, const QString& module)**

Metoda pro zadanou cestu ke skriptu v proměnné file a pro jméno modulu v proměnné module vytvoří PythonQt objekt a uloží ho do zásobníku. Následně vytvořenému objektu předá zdrojový kód skriptu z uvedené cesty a vrátí zpřístupněný skript vytvořením objektu Script.

**void unload(Script* script)**

Zadaný objekt skriptu vyjme ze zásobníku a uvolní paměť.

**void registerObject(const QString& name, QObject* obj)**

Zadaný objekt obj se jménem name uloží do zásobníku.

**void registerObjectToScript(const QString& name, QObject* obj, Script* script)**

První z mnou vytvořených pomocných metod, kdy zadanému skriptu zaregistruje objekt z designeru obj pod jménem name.

**Script* registerObjectToCode(const QString& name, QObject* obj, const QString& scriptCode)**

Druhá z mnou vytvořených pomocných metod, kdy pro zadaný PythonQt zdrojový kód vytvoří skript třídy Script a zaregistruje mu objekt designeru obj pod jménem name.

**QVariant eval(const QString& script)**

Pouze provede zdrojový kód předaný v parametru script.

### 6.2.2 Skript

Třída, která uchovává objekt PythonQt a může mu přiregistrovat objekt, interpretovat kód či soubor o dané cestě a nebo to nejdůležitější, zavolat nejakou metodu uvnitř zdrojového kódu PythonQt.

### 6.2.3 ScriptComponent

Poslední článek modulu, který zděděným třídám zpřístupní skriptovací engine, odkaz na aplikaci a odkaz hlavního okna.

### 6.2.4 generic_settings.py

Jedná se o mnou napsaný skript v PythonQt pro nastavení generických parametrů komponenty, kde pro úspěšné provedení skriptu, je potřeba zavolat metodu init. V ní se nejdříve vytvoří dialogové okno třídy QDialog a následně se získá ze zaregistrovaného objektu z designeru obsah zásobníku generických parametrů za pomoci metody `getGenericParams()`. Poté se vytvoří QGridLayout do kterého se dle zásobníku vytvoří QLabel jméno parametru a vedle něj nastavovací prvek QLineEdit nebo QComboBox. Jednotlivé nastavovací objekty se uloží do zásobníku, který se po potvrzení dialogu prochází a zjišťují se jejich jednotlivé hodnoty a za pomoci metody `int pythonQtSetGenParam(QString name, QString value)` se pro jméno name upraví hodnota parametru na value v zásobníku generických parametrů komponenty. Metoda vrací 1 pro úspěch změny, 0 pro neúspěch a -1 pro neexistenci parametru. Tyto návratové hodnoty jsou obvzláště potřeba při konstrukci vlastního dialogového okna, kde je napsána další speciální metoda `bool pythonQtInsertGenParam(QString name, QString type, QString defaultV, QString values = QString(), QString value = QString())`, která dokáže zaregistrovat úplně nový generický parametr komponentě. Při nové registraci parametru je potřeba zadat jeho jméno, výchozí hodnotu a typ. Nepovinné je zadání možných nabývaných hodnot nebo vložení jiné hodnoty než výchozí. Aby byly zmíněné metody viditelné, je jim potřeba předřadit speciální direktivu `Q_INVOKABLE`.

### 6.2.5 Skriptování uvnitř objektu COMPONENT

Volba nastavení komponenty v kontextovém menu zavolá metodu `settingsComponent()`. Ta nejdříve zjistí, jestli popis komponenty obsahoval volbu vlastního dialogového okna a to tak, že zkontroluje již výše zmíněnou proměnnou typu QString, jestli není prázdná. Pokud ano, má se zobrazit přednastavené dialogové okno a za pomoci metody load se vytvoří objekt Script ze skriptu `generic_settings.py` a přiregistruje se mu zvolená komponenta. Objekt Script následně zavolá metodu init, která se postará o zobrazení okna a přenastavení parametrů. Pokud není proměnná prázdná, zkusí se zjistit, zda-li její obsah odpovídá cestě k exetrnímu skriptu. Pokud ano, metoda load vytvoří objekt Script z dané cesty, opět mu přiregistruje zvolenou komponentu a zavolá metodu init. Pokud obsah proměnné neodpovídá cestě, jedná se o zdrojový kód, kde pomocí metody registerObjectToCode vytvoří objekt Script z daného zdrojového kódu a přiregistruje mu opět zvolenou komponentu. Nakonec následuje volání metody init. Po provedení skriptu se vždy volá metoda unload.

Výše zřejmě docházelo k otázkám, proč je tento zásobník typu QVariantMap. Odpověď je snadná. Je to jediný typ odvozený z QMap, který PythonQt dokáže správně interpretovat tím, že jej interně přetypuje na typ dict.

Při psaní vlastního dialogového okna se uživatel může inspirovat skriptem `generic_settings.py`. Pro správnou funkci je potřeba dodržet existenci metody init, která se volá automaticky pro zobrazení dialogového okna.

## 6.3 CMake

Pro usnadnění překladu napříč různými platformami byl využit nástroj CMake, kde zdrojové kódy designeru obsahují CMakeList.txt, který generuje Makefile pro danou platformu s daným OS. Stačí zadat příkaz `cmake .` nebo `cmake -G "platform"` (platform = daná platforma pro kompilaci) do příkazové řádky a vygeneruje se Makefile. Pod MS Windows můžeme použít jako kompilátor MinGW, pod Unixovými systémy použijeme klasické GCC. Více o spuštění aplikace a instalaci potřebných modulů je uvedeno v souboru INSTALL.txt ve složce designer na přiloženém DVD.

# Kapitola 7

# Ověření funkčnosti

V následující části si ukážeme jednotlivé kroky tvorby nového projektu. Cílem bude vytvořit aplikaci, kterou následně zkompilujeme a nahrajeme do platformy FITkit.

## 7.1 Přidání top-level entity

Abychom mohli pracovat s modulem a navrhovat obvody, je nutné definovat tzn. top-level entitu, která popisuje jednotlivé signály, které máme v rámci vyvíjené aplikace k dispozici. Proto vytvoříme XML popis top-level entity a přidáme ho do adresáře `platforms`. Pokud struktura XML popisu odpovídá správnému formátu, dojde k vložení názvu nové top-level entity do seznamu umístěného v panelu databáze komponent. Pro ulehčení vývoje jsou již top-level entity bare, gp i pc součástí designeru. Jako příklad uvádím kompletní popis top-level entity typu bare, viz příloha A.

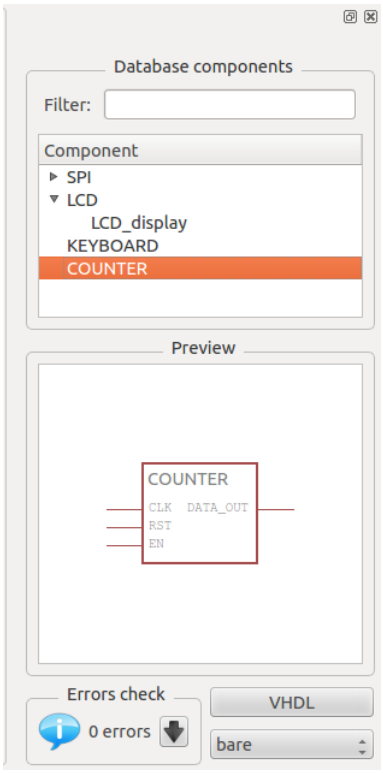
## 7.2 Přidání nové komponenty

Jako další si vytvoříme vlastní komponentu, a to universální čítač nahoru s libovolným počtem bitů jménem COUNTER. Proto vytvoříme podadresář `counter` v adresáři `db_components`. Do tohoto podadresáře vložíme soubor se jménem `counter.vhd`, který obsahuje VHDL kód popisující funkci čítače, viz příloha B a také soubor `package.xml`, který popisuje jednak samotnou komponentu a jednak její závislosti na zdrojových kódech. Tento soubor je důležitý pro překlad, jak už bylo zmíněno výše. Jako další vytvoříme v adresáři `db_components` soubor jménem `counter.xml`, který obsahuje popis komponenty:

```
<database>
   <component name="COUNTER" code="counter/package.xml" >
      <generic default="3" values="1...32" name="DATA_WIDTH" type="integer"/>
      <signal class="control" sensitivity="high" direction="out" name="DATA_OUT"
      type="std_logic_vector" width="DATA_WIDTH"widthtype="generic"/>
      <port>
         <signal class="clock" direction="in" name="CLK" type="std_logic" width="1"
         widthtype="fixed"/>
         <signal class="reset" sensitivity="high" direction="in" name="RST"
         type="std_logic" width="1" widthtype="fixed"/>
         <signal class="data" sensitivity="raisingedge" direction="in" name="EN"
         type="std_logic" width="1" widthtype="fixed"/>
```

```
      </port>
   </component>
</database>
```

Nyní spustíme designer a zkontrolujeme, zda-li se komponenta přidala do databáze komponent, viz obrázek 7.1. Jak můžeme vidět, komponenta byla vpořádku přidána a můžeme se

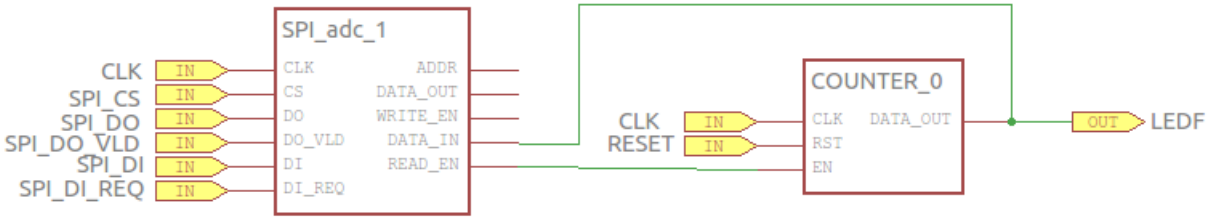



Obrázek 7.1: Ukázka vytvořené komponenty

vrhnout na vytvoření projektu. Pro ulehčení vývoje jsou již základní komponenty uvedeny výše, viz sekce 3.5

## 7.3 Demonstrační úloha 1 - blikání s LED diodou

Jako první příklad uvedu obvod blikání LED s řízením frekvence blikání díky výše uvedenému čítači, viz obrázek 7.2. Obvod se skládá z SPI dekodéru a čítače. Na dekodéru a čítači



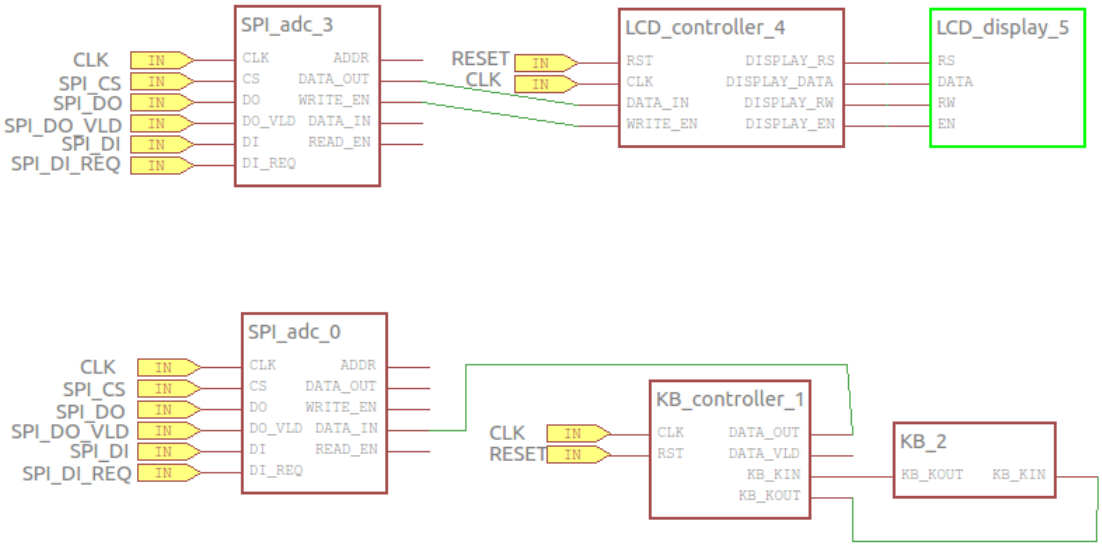

Obrázek 7.2: Ukázka navrženého schématu čítače a LED

necháme výchozí nastavení parametrů. Na pinu dekodéru DATA_IN nastavíme připojení

signálů DATA_IN.0 - DATA_IN.3 na signály DATA_OUT.0 - DATA_OUT.3 komponenty čítače. Dále na pinu DATA_OUT nastavíme připojení signálu LEDF na DATA_OUT.0. Pokud máme obvod bez chyby, dojde k odemknutí tlačítka VHDL a necháme vygenerovat projekt pro tento obvod. Jak přeložit aplikaci se dozvíte v sekci 7.5. Výsledkem aplikace je blikání diody D4.

## 7.4 Demonstrační úloha 2 - LCD display s klávesnicí

Jako druhý příklad obvodu uvedu LCD display, na kterém se budou zobrazovat stiknuté klávesy na FITkitu, viz obrázek 7.3. Základ obvodu tvoří dva SPI dekodéry, kde na první z




Obrázek 7.3: Ukázka navrženého schématu LCD a klávesnice

nich je připojen LCD display a na druhý klávesnice. Jak přeložit aplikaci se dozvíte v sekci 7.5.

## 7.5 Otestování na platformě FITkit

Výsledný vygenerovaný projekt designerem uložíme do SVN repozitáře zdrojových kódů aplikací pro platformu FITkit. Nejlépe projekt umístíme do podadresáře demo v adresáři apps. Tím, že umístíme projekt zde, dojde po spuštění QDevKitu k jeho zobrazení ve stromové struktuře aplikací pro FITkit a to přesně v položce Demo aplikace. Poté jen dvojkliknutím na název projekt přeložíme a pokud jsme dobře zadali cesty ke zdrojovým kódům v databázi komponent, dojde i k jeho nahrání do FITkitu.

# Kapitola 8

# Závěr

Jelikož Fakulta Informačních technologií v Brně umožňuje výuku programovatelného hardwaru na speciální platformě FITkit, je potřeba díky pokroku v oblasti programovatelného hardwaru vytvořit program pro snažší návrh a tvorbu aplikací na základě interakce grafického prostředí s uživatelem. Abychom se co nejvíce přiblížili moderním nástrojům, byl proveden rozvrh existujících komerčních nástrojů, který ucelil představu o tom, jak by měl takový designer vypadat. Dále bylo potřeba se seznámit s platformou FITkit a jejími komponentami.

Byl proveden rozbor tří významných nástrojů. Nástroj od společnosti Altium přinesl první představy o tom, jak by měl takový designer vypadat a co je vše potřeba uchovávat pro dynamickou knihovnu komponent. Tento nástroj pomohl vytvořit návrh popisu jednotlivých platforem, protože se platforma FITkit vyskytuje ve více verzích. Další nástroj od firmy Xilinx přispěl svým řešením tvorby vlasního dialogového okna pro konfiguraci komponenty v případě, kdy si nevystačíme se standardními možnostmi konfigurace komponenty. Dále spolu s nástrojem od společnosti Altium přispěl k návrhu popisu komponenty pro snadnou správu dynamické databáze komponent. Poslední z nástrojů od společnosti Cypress Semiconductor pomohl ucelit představy o vzhledu a funkci grafického editoru.

Navržené popisy jsou vytvořeny ve standardu XML kvůli přehlednosti, snadné editace a existenci hotových parserů. Implementace modulu byla provedena v jazyce C++ za pomoci Qt knihovny jako grafického frameworku. Nástroj CMake umožňuje jednoduchým způsobem překládat aplikaci na rozdílných platformách a operačních systémech. Popis dialogového okna pro konfiguraci komponenty je psán ve skriptovacím jazyce PythonQt a interpretován použitím upravených tříd z QDevKitu. Díky použití PyQt jako grafické knihovny a jazyku Python pro tvorbu dialogového okna se aplikace stává variabilní, kdy můžeme tvořit vlastní konfigurační okna.

Hlavní přínos této práce spatřuji v možnosti snadnějšího návrhu obvodů pro platformu FITkit, což můžeme vidět v kapitole ověření funkčnosti.

I přes to, že je designer kompletní, můžeme najít vylepšení nebo další funkcionalitu, která by mohla usnadnit práci s designerem. V budoucnu by mohla být implementována mřižka pro přesnější pozicování komponent ve scéně. Dále by bylo vhodné vytvořit editační nástroje pro databázi komponent, které by umožnily vytvářet či editovat komponenty přímo v aplikaci. Jako posledním vylepšením by mohl být textový editor zdrojových kódů.

# Literatura


[1] Nokia Corporation: Signals & Slots. [online], 2011.
URL http://doc.qt.nokia.com/latest/signalsandslots.html

[2] CMake: About. [online], 2012.
URL http://www.cmake.org/cmake/project/about.html

[3] Wikipedia: Qt(knihovna). [online], 2012.
URL http://cs.wikipedia.org/wiki/Qt_%28knihovna%29

[4] Altium: Next generation electronics design. Hlavní stránky společnosti Altium. [online], vytvořeno 2010.
URL http://www.altium.com/

[5] J. Boyer, P. D., Glenn Marcy: Canonical XML Version 2.0. [online], 2008.
URL http://www.w3.org/TR/2010/WD-xml-c14n2-20100831

[6] J. Boyer, P. D., Glenn Marcy: Extensible Markup Language (XML) 1.0 (Fifth Edition). [online], 2008.
URL http://www.w3.org/TR/REC-xml

[7] Pandatron: pandatron.cz - PSoC Creator IDE společnosti Cypress Semiconductor. Stránky elektrotechnického magazínu. [online], vytvořeno 12.3.2010.
URL http://pandatron.cz/?1310&psoc_creator_ide_spolecnosti_cypress_semiconductor

[8] Quin, L. R. E.: XML TECHNOLOGY. [online], 2010.
URL http://www.w3.org/standards/xml

[9] Rusty, E.: Effective XML. [online], 2004.
URL http://www.cafeconleche.org/books/effectivexml

[10] Slaný, K.; Markovič, J.: FITkit: Firmware - klávesnice 4x4. Stránky projektu FITkit. [online], poslední modifikace 19.3.2009.
URL http://merlin.fit.vutbr.cz/FITkit/docs/firmware/fpga_keyboard.html

[11] Tošovský, P.: hw.cz - úvod do Altium Designeru I. Stránky o elektronice a programování. [online], vytvořeno 27.3.2010.
URL http://hw.cz/teorieapraxe/software/art3413-uvod-do-altium-designeru-i.html

[12] Vanhoof, B.: fpgacentral.com - New tool enables mapping complex applications onto FPGAs. Portál o CPLD/FPGA. [online], vytvořeno 2.2.2009.

URL http:
//www.fpgacentral.com/article/new-tool-enables-mapping-complex-applica

[13] Vašíček, Z.: FITkit: Hardware. Stránky projektu FITkit. [online], 2009.
URL http://merlin.fit.vutbr.cz/FITkit/hardware.html

[14] Vašíček, Z.: Stránky projektu FITkit. [online], 2009.
URL http://merlin.fit.vutbr.cz/FITkit/

[15] Vašíček, Z.: FITkit: Návody - překladový systém. Stránky projektu FITkit. [online], 2012.
URL http://merlin.fit.vutbr.cz/FITkit/docs/navody/kompilacev2.html

[16] Vašíček, Z.: FITkit: Návody - tvorba vlastní aplikace. Stránky projektu FITkit. [online], 2012.
URL http://merlin.fit.vutbr.cz/FITkit/docs/navody/tutor_blikac.html

[17] Vašíček, Z.: FITkit: Popis hardwaru kitu - libfitkit. Stránky projektu FITkit. [online], 2012.
URL http://merlin.fit.vutbr.cz/FITkit/docs/navody/libkitclient.html

[18] Vašíček, Z.: FITkit: Popis hardwaru kitu - USB. Stránky projektu FITkit. [online], 2012.
URL http://merlin.fit.vutbr.cz/FITkit/docs/hardware/hw_ftdi.html

[19] Vašíček, Z.: FITkit: Firmware - komunikační systém. Stránky projektu FITkit. [online], poslední modifikace 17.9.2009.
URL
http://merlin.fit.vutbr.cz/FITkit/docs/firmware/fpga_interconect.html

[20] Vašíček, Z.: FITkit: Firmware - řadič LCD displeje. Stránky projektu FITkit. [online], poslední modifikace 19.3.2009.
URL http://merlin.fit.vutbr.cz/FITkit/docs/firmware/fpga_lcd.html

[21] Vašíček, Z.: FITkit: Firmware - řadič přerušení. Stránky projektu FITkit. [online], poslední modifikace 19.3.2009.
URL
http://merlin.fit.vutbr.cz/FITkit/docs/firmware/fpga_interrupt.html

[22] Vašíček, Z.: FITkit: Firmware - řadič PS/2. Stránky projektu FITkit. [online], poslední modifikace 20.3.2009.
URL http://merlin.fit.vutbr.cz/FITkit/docs/firmware/fpga_ps2.html

[23] Williams, T.: rtcmagazine.com - CPU Architectures Bring C Programmability into Formerly Specialized Devices. Stránky magazínu o vestavěných systémech. [online], vytvořeno 2.2.2009.
URL http://www.rtcmagazine.com/magazine/articles/view/101392/pg:2

[24] Xilinx: FPGA and CPLD Solutions from Xilinx, Inc.. Hlavní stránky společnosti Xilinx. [online], vytvořeno 2010.
URL http://www.xilinx.com/

[25] Čapka, L.; Vašíček, Z.: FITkit: Firmware - komponenta PicoCPU. Stránky projektu FITkit. [online], poslední modifikace 12.4.2010.
URL http://merlin.fit.vutbr.cz/FITkit/docs/firmware/fpga_picocpu.html

# Příloha A

# Top-level entita bare

```
<platform name="bare" architecture="tlv_bare_ifc" >
   <!-- hodiny -->
   <record name="SMCLK" direction="in" type="std_logic" width="1"/>
   <record name="ACLK" direction="in" type="std_logic" width="1"/>
   <record name="FCLK" direction="in" type="std_logic" width="1"/>
   <record name="CLK" direction="in" type="std_logic" width="1"/>

   <record name="RESET" direction="in" type="std_logic" width="1"/>
   <record name="IRQ" direction="out" type="std_logic := 'Z'" width="1"/>

   <!-- LED -->
   <record name="LEDF" direction="out" type="std_logic" width="1"/>

   <!-- interni SPI -->
   <record name="SPI_CS" direction="in" type="std_logic" width="1"/>
   <record name="SPI_DI" direction="out" type="std_logic" width="1"/>
   <record name="SPI_DI_REQ" direction="in" type="std_logic" width="1"/>
   <record name="SPI_DO" direction="in" type="std_logic" width="1"/>
   <record name="SPI_DO_VLD" direction="in" type="std_logic" width="1"/>

   <!-- klavesnice 4x4 -->
   <record name="KIN" direction="out" type="std_logic_vector(3 downto 0)" width="4"/>
   <record name="KOUT" direction="in" type="std_logic_vector(3 downto 0)" width="4"/>

   <!-- displej -->
   <record name="LE" direction="out" type="std_logic" width="1"/>
   <record name="LRS" direction="out" type="std_logic" width="1"/>
   <record name="LRW" direction="out" type="std_logic" width="1"/>
   <record name="LD" direction="inout"
   type="std_logic_vector(7 downto 0) := (others => 'Z')" width="8"/>

   <!-- SDRAM -->
   <record name="RA" direction="out" type="std_logic_vector(14 downto 0)" width="15"/>
   <record name="RD" direction="inout"
   type="std_logic_vector(7 downto 0) := (others => 'Z')" width="8"/>
```

```
    <record name="RDQM" direction="out" type="std_logic" width="1"/>
    <record name="RCS" direction="out" type="std_logic" width="1"/>
    <record name="RRAS" direction="out" type="std_logic" width="1"/>
    <record name="RCAS" direction="out" type="std_logic" width="1"/>
    <record name="RWE" direction="out" type="std_logic" width="1"/>
    <record name="RCLK" direction="out" type="std_logic" width="1"/>
    <record name="RCKE" direction="out" type="std_logic" width="1"/>

    <!-- P3M -->
    <record name="P3M" direction="inout"
    type="std_logic_vector(7 downto 0) := (others => 'Z')" width="8"/>

    <!-- AFBUS -->
    <record name="AFBUS" direction="inout"
    type="std_logic_vector(11 downto 0) := (others => 'Z')" width="12"/>

    <!-- audio codec -->
    <record name="ALRCIN" direction="inout" type="std_logic := 'Z'" width="1"/>
    <record name="ADIN" direction="out" type="std_logic" width="1"/>
    <record name="ALRCOUT" direction="inout" type="std_logic := 'Z'" width="1"/>
    <record name="ADOUT" direction="in" type="std_logic := 'Z'" width="1"/>
    <record name="ABCLK" direction="inout" type="std_logic := 'Z'" width="1"/>
</platform>
```

# Příloha B

# VHDL kód čítače

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter is
   generic (
      DATA_WIDTH : integer := 3
   );
   port(
      CLK : in std_logic;
      RST : in std_logic;
      EN : in std_logic;
      DATA_OUT : out std_logic_vector (DATA_WIDTH-1 downto 0)
   );
end counter;

architecture arch_counter of counter is
   signal cnt : std_logic_vector(DATA_WIDTH-1 downto 0);
begin
   DATA_OUT <=cnt;

   -- Counter
   process(CLK, RST)
   begin
      if (RST = '1') then
          cnt <= (others => '0');
      elsif (CLK'event) and (CLK='1') then
          if (EN = '1') then
              cnt <= cnt + 1;
          end if;
      end if;
   end process;

end arch_counter;
```

# Příloha C

# Seznam použitých zkratek

**API** - Application Programming Interface (rozhraní pro programování aplikací)
**CMD** - Command (příkaz)
**COM** - sériový port
**C++** - Objektově orientovaný programovací jazyk
**DEL** - DELETE (klávesa delete na klávesnici)
**DTD** - Document Type Definition (definice typu dokumentu)
**FIFO** - First In, First Out (zásobník typu první dovnitř, první ven)
**FPGA** - Field-Programmable Gate Array (programovatelné hradlové pole)
**GCC** - GNU Compiler Collection (sada překladačů vytvořených v rámci projektu GNU)
**GPL** - General Public License (všeobecná veřejná licence)
**GUI** - Graphical user interface (grafické uživatelské prostředí)
**HDL** - Hardware Description Language (programovací jazyk pro popis hardware)
**HTML** - HyperText Markup Language (programovací jazyk pro popis webových stránek)
**ISO** - International Organization for Standardization (mezinárodní organizaci zabývající se tvorbou norem)
**I2C** - I2C-bus, Inter-IC-bus (dvouvodičové datové propojení od firmy Philips)
**LED** - Light-Emitting Diode (svítící dioda)
**LGPL** - Lesser General Public License (všeobecná veřejná licence)
**LMB** - Local Memory Bus (lokální paměťová sběrnice)
**MCU** - Multipoint Control Unit (vícebodová řídicí jednotka)
**OS** - Operating system (operační systém)
**PLB** - Processor Local Bus (procesorová lokální sběrnice)
**PS/2** - konektor pro připojení myši nebo klávesnice
**QML** - Qt Meta-Object-Language (deklarativní JavaScript programovací jazyk)
**SGML** - Standard Generalized Markup Language (univerzální značkovací metajazyk)
**SPI** - Serial Peripheral Interface (sériové periferní rozhraní)
**SWIG** - Simplified Wrapper and Interface Generator (program/knihovna umožňuje rozšiřování a zapouzdřování jazyků)
**USB** - Universal Serial Bus (univerzální sériová sběrnice)
**VGA** - Video Graphics Array (standard pro počítačovou zobrazovací techniku)
**VHDL** - VHSIC Hardware Description Language (programovací jazyk pro popis hardware)
**XML** - Extensible Markup Language (rozšířitelný značkovací jazyk)

# Příloha D

# Obsah CD

- designer - zdrojové kódy aplikace
- tech_report - zdrojové kódy technické zprávy
- downloads - instalační soubory pro OS Windows
- dip.pdf - technická zpráva ve formátu pdf

# Příloha E

# Dokumentace

## E.1 Manuál

Spuštěním souboru index.html ve složce designer/src/manual/html se zobrazí nápověda pro obsluhu programu. Tatéž nápověda lze spustit přímo z aplikace, a to v menu položkou Help → Manual nebo klávesou F1.

## E.2 Projektová dokumentace

Spuštěním souboru index.html ve složce designer/doc/html se zobrazí projektová dokumentace aplikace ve formátu HTML vygenerovaná systémem doxygen.

## E.3 Instalace

Postup instalace potřebných modulů a překlad aplikace je popsán v textovém souboru INSTALL.txt ve složce Repomo.