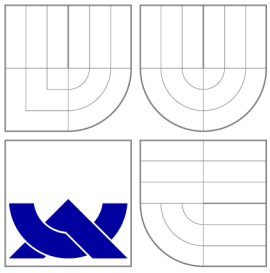
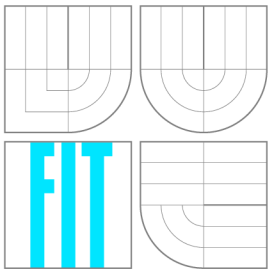


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

FPGA PLATFORMA PODPORUJÍCÍ .NET MICRO FRAMEWORK

FPGA PLATFORMA WITH .NET MICRO FRAMEWORK SUPPORT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN MATYÁŠ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZDENĚK VAŠÍČEK, Ph.D.

BRNO 2013

Abstrakt

Záměrem práce je navrhnout a vytvořit prototyp platformy určené k vývoji vestavěných systémů, která bude sestávat z mikrokontroléru s architekturou ARM Cortex-M a programovatelného hradlového pole FPGA. Cílem je na této platformě umožnit běh open-source prostředí .NET Micro Framework. Práce stanovuje specifikaci této platformy, popisuje proces její konstrukce, způsob portace .NET Micro Frameworku a zprovoznění komunikační sběrnice mezi mikrokontrolérem a obvodem FPGA. V závěru práce je představena sada demonstračních aplikací, které čtenáře seznamují se způsobem tvorby software pro tuto navrženou platformu.

Abstract

The goal of the thesis is to design a development board that may be used for embedded systems prototyping. The board's key parts are an ARM-Cortex-based microcontroller and a FPGA programmable circuit. The platform is designed with .NET Micro Framework support in mind. The thesis contains specifications of the development board, describes the design process as well as the task of .NET Micro Framework porting and the establishment of communication bus between the FPGA and microcontroller circuits. The thesis is concluded by a set of demonstration examples which outline how to develop new applications for the designed platform.

Klíčová slova

Vestavěné systémy, ARM, FPGA, .NET Micro Framework, vývojová a prototypovací platforma

Keywords

Embedded systems, ARM, FPGA, .NET Micro Framework, development and prototyping platform

Citace

Jan Matyáš: FPGA platforma podporující
.NET Micro Framework, diplomová práce, Brno, FIT VUT v Brně, 2013

FPGA platforma podporující .NET Micro Framework

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením pana Ing. Zdeňka Vašíčka, Ph.D. Veškeré zdroje, z nichž jsem čerpal, jsem uvedl v seznamu použité literatury na konci práce.

.....
Jan Matyáš
30. července 2013

Poděkování

Děkuji panu Ing. Zdeňku Vašíčkovi, Ph.D., za odborné vedení práce a pomoc při konstrukci hardwarového prototypu. Rodičům, příbuzným a přátelům děkuji za jejich vytrvalou podporu po celou dobu studií.

Obsah

Kapitola 1

Úvod

Oblast vestavěných systémů má v rámci výpočetních systémů značný význam významu a uplatňuje se snad ve všech oblastech lidské činnosti. Tuto skutečnost dokumentují statistiky, které indikují, že přibližně 98 % ze všech vyrobených procesorů spadá do kategorie vestavěných zařízení a jen pouhé 2 % procesorů jsou určeny pro obecné výpočetní platformy – pro osobní počítače a servery [44].

S rostoucím významem vestavěných systémů jsou spojeny požadavky na zlepšování produktivity vývojářů, zvyšování efektivity vývoje software pro vestavěná zařízení a také poptávka po kvalifikovaných programátorech, kteří budou vestavěné systémy schopni navrhovat a vyvíjet pro ně řídicí software. Jednou z možností, jak na tyto požadavky trhu reagovat, je zvyšovat úroveň abstrakce, s jakou se vestavěné systémy programují. Typicky se jedná o přejímání technik běžně používaných při tvorbě software pro osobní počítače, jako je např. objektově orientované programování, vícevláknový výpočetní model nebo automatická správa paměti. Softwarová platforma .NET Micro Framework, která je ústředním tématem této práce, je jedním z příkladů uvedeného trendu. Jedná se o interpret bytekódu jazyka C# implementovaný speciálně pro vestavěné výpočetní platformy s velmi omezenými prostředky – pro jednoduché mikrokontrolery vybavené pouze desítkami kilobajtů paměti RAM a ROM.

Dalším významným trendem v oboru vestavěných systémů je prosazování výpočetních architektur typu ARM, které vynikají ve výpočetním výkonu vztaženém na jednotku příkonu. Jinou progresivní oblastí jsou rekonfigurovatelné výpočetní platformy FPGA, které se velmi často kombinují právě s mikrokontrolery v rámci návrhové metodiky *hardware-software codesign* [38], jejímž cílem je provádět souběžný návrh vestavěného hardware a software za účelem zvýšení produktivity práce a rychlého uvedení vestavěných produktů na trh.

Cílem této práce je reagovat na aktuální vývoj v oblasti vestavěných systémů, na trendy zmíněné výše a vytvořit novou vývojovou platformu umožňující běh .NET Micro Frameworku. Tato platforma bude sestávat z mikrokontroleru s jádrem ARM a z rekonfigurovatelného čipu typu FPGA, přičemž bude zaměřena jednak na výuku principů moderních vestavěných systémů, jednak také na vývoj a rychlé prototypování vestavěných zařízení.

Text práce je členěn následovně. Kapitola 2 čtenáři představuje .NET Micro Framework, jeho strukturu a nastiňuje způsob tvorby vestavěných aplikací v tomto prostředí. Třetí kapitola stručně analyzuje současnou situaci na trhu s přípravky pro vývoj a prototypování hardware a stanovuje specifikaci nové hardwarové platformy, která bude v rámci práce realizována. Následující kapitola se zabývá návrhem obvodového zapojení hardwarového přípravku a zdůvodňuje volbu komponent a další rozhodnutí, která v této fázi bylo nutné

učinit. Kapitola 5 se věnuje úpravě open-source .NET Micro Frameworku do takové podoby, aby ho bylo možné provozovat na nově představené hardwarové platformě. Šestá kapitola popisuje tvorbu firmware pro obvod FPGA, kdy hlavním řešeným problémem je umožnit výměnu dat mezi mikrokontrolerem a obvodem FPGA po paměťové sběrnici. V kapitole 7 je čtenář seznámen se způsobem tvorby aplikací na nové hardwarové platformě prostřednictvím čtyř ukázkových aplikací. Zároveň jsou také změřeny a vyhodnoceny parametry komunikační linky propojující mikrokontroler a čip FPGA.

Téma řešené v rámci práce je velmi aktuální problematika a softwarový balík .NET Micro Framework podléhá v současnosti rychlému vývoji jak ze strany firmy Microsoft, tak i díky nezávislé komunitě vývojářů. Z tohoto důvodu má většina zdrojů, z nichž bylo při práci čerpáno, elektronický charakter a jedná se zejména o elektronickou dokumentaci k softwarovým produktům nebo datové listy součástek použitých při konstrukci hardwarového prototypu.

Kapitola 2

Základy .NET Micro Frameworku

Text kapitoly představuje .NET Micro Framework (dále jen *.NET MF*) jako jednu ze tří implementací platformy Microsoft .NET, speciálně vyvinutou pro jednoduché 32-bitové mikrokontroléry. Cílem následujícího textu je popsat .NET MF z pohledu aplikačního programátora, nastínit jeho základní strukturní bloky a zmínit operace, které je nezbytné provést při tvorbě vestavěných aplikací prostřednictvím tohoto frameworku.

Při přípravě přehledového textu této kapitoly bylo čerpáno zejména z dokumentace k .NET MF na serveru MSDN [30] a z dokumentace, která je součástí softwarového balíku .NET MF *Porting Kit* [31].

2.1 Postavení .NET MF v rámci platformy Microsoft .NET

Platforma .NET [27] je softwarovým frameworkem firmy Microsoft, původně primárně zaměřená na vývoj aplikací pro osobní počítače s operačním systémem Windows. Klíčovými součástmi .NET Frameworku jsou jednak knihovny pro vývoj software, které jsou programátorovi zpřístupněny prostřednictvím API rozhraní, jednak samotné běhového prostředí (virtuální stroj) pro vytvořené aplikace, tzv. *Common Language Runtime* (CLR).

Aplikace vytvořené v prostředí .NET se překladačem transformují do intermediárního, na platformě nezávislého kódu označovaného jako *Common Intermediate Language* (CIL). Tento kód je možné spustit pomocí virtuálního stroje CLR, který zabezpečuje překlad kódu do strojových instrukcí cílové platformy. Jednotka CLR poskytuje běžící aplikaci další služby, mezi něž patří automatická správa paměti, zpracování výjimek, přístup k souborovému systému a další. Vytváří tedy abstraktní vrstvu nad skutečným operačním systémem a hardwarem a umožňuje přenositelnost aplikace mezi platformami, pro které existuje implementace CLR. Služby, které vrstva CLR běžícím aplikacím nabízí, se souhrnně označují jako *Common Language Infrastructure* (CLI) a spolu s definicí jazyka CIL jsou standardizovány organizacemi ISO [17] a ECMA [10][11].

V současné době je platforma .NET firmy Microsoft dostupná ve třech variantách pro odlišné kategorie výpočetních zařízení zařízení:

- **.NET Framework pro osobní počítače** (tzv. *Desktop Client*),
- **.NET Compact Framework** určený pro přenosná zařízení s operačními systémy Windows CE a Windows Mobile (např. tablety nebo mobilní telefony),
- **.NET Micro Framework** pro vestavěné systémy s velmi omezenými zdroji založené na mikrokontrolerech.

Poslední uvedená implementace, .NET Micro Framework (.NET MF) [26], je ve středu zájmu této diplomové práce a bude podrobněji popsána v následujících odstavcích.

Platforma .NET MF koncepčně vychází z projektu SPOT (*Smart Personal Object Technology*) divize Microsoft Research. Cílem projektu SPOT bylo vytvořit softwarovou podporu pro konstrukci inteligentních domácích spotřebičů a elektroniky. Příkladem zařízení z této kategorie jsou hodinky *Microsoft SPOT Watch* [23], které byly uvedeny na trh v roce 2004, avšak nedočkaly se výrazného komerčního úspěchu. Zásadním milníkem ve vývoji .NET Micro Frameworku je rok 2009, kdy byl kód této platformy uvolněn jako open-source pod Apache licencí (viz např. článek [14]). V úvodu zpracování této diplomové práce (září 2012) byla aktuální stabilní verzí .NET MF verze 4.2, která spolupracuje s Microsoft Visual Studiem 2010. Během práce na projektu byla v prosinci 2012 uvolněna verze .NET MF 4.3, jejíž hlavním přínosem je podpora Visual Studia 2012. Vzhledem k rozpracovanosti projektu bylo pokračováno s prací ve verzi 4.2, nicméně obecná fakta a postupy, které budou v rámci práce zmíněny, je možné aplikovat i na verzi 4.3. Aplikace pro platformu .NET MF je možné vytvářet v jazyce C# a od verze .NET MF 4.2 také v jazyce Visual Basic.

Tato diplomová práce se soustřeďuje na práci s .NET MF prostřednictvím nástrojů od firmy Microsoft. Je však vhodné zmínit, že díky standardizovaným specifikacím CIL existují i alternativní nezávislé softwarové implementace, které o kompatibilitu s produkty firmy Microsoft usilují. Významným zástupcem této kategorie je open-source projekt Mono [3]. Podpora prostředí .NET MF v rámci tohoto projektu však byla v první polovině roku 2013 na experimentální úrovni, a proto autor práce vývojovou platformu Mono nevyzkoušel. V případě rozšíření kompatibility se ale v budoucnu může jednat o alternativu umožňující programovat vestavěné aplikace a .NET MF i v prostředí mimo operační systémy Microsoft Windows, kde pro tyto činnosti v současnosti neexistují dostatečně ověřené a spolehlivé vývojové nástroje.

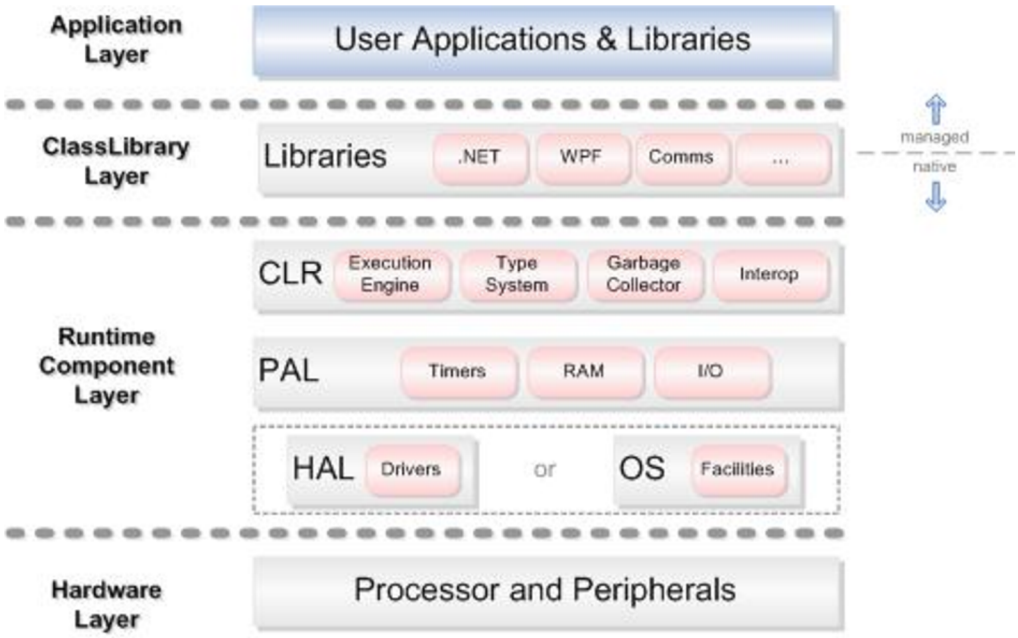
2.2 Struktura .NET MF

.NET MF je souhrnný název pro implementaci CLR pro jednoduché 32-bitové mikrokontrolery a pro sadu knihoven určených k tvorbě vestavěných aplikací. V současné době je možné běhové prostředí .NET MF CLR používat na mikrokontrolerech ARM architektury ARMv7 a na procesorech Blackfin firmy Analog Devices[1]. Aby byl běh .NET MF možný, musí být cílový mikrokontroler vybaven alespoň 256 kB paměti FLASH a nejméně 32 kB paměti RAM [35]. Samotný CLR pro .NET MF a ovladače pro jednotlivé periferie MCU jsou napsány v jazyce C++. V jazyce C# je potom implementována sada vysokoúrovňových knihoven, prostřednictvím nichž se jednotlivé hardwarové periferie zpřístupňují uživatelské aplikaci.

V rámci dokumentace k .NET MF se používá následující terminologie. Implementaci CLR a nízkoúrovňových ovladačů v C++ nazýváme termínem *firmware*. O kódu v jazyce C++ mluvíme jako o *nativním kódu*. Nativní kód se vyznačuje přímým přístupem k hardware cílové platformy.

Uživatelský software, který je vytvořen aplikačním programátorem v jazyce C#, pomocí překladače posléze transformován do CIL a běží na mikrokontroléru v prostřednictvím interpretu CLR, označujeme jako *aplikaci* nebo *interpretovaný kód* (*managed code*). Mluví-li se o *uživateli*, je jím myšlen programátor, který se nezabývá vnitřními principy fungování

[1]Dříve dostupný přehled všech podporovaných platforem a překladačů pro .NET MF již není na stránkách firmy Microsoft k dispozici. Přehled platforem podporujících .NET MF je však možné najít přímo ve zdrojovém kódu frameworku [25].



Obrázek 2.1: Struktura .NET Micro Frameworku [30]

.NET MF a vyvíjí pouze aplikační kód v prostředí jazyka C#, který poběží skrze služby interpretu CLR.

Framework je rozčleněn do několika hierarchicky uspořádaných vrstev, jak je znázorněno na obr. 2.1. Cílem tohoto návrhu je dosáhnout oddělení kódu závislého na hardware do samostatných vrstev, což umožní přenositelnost (portabilitu) frameworku na různé hardwarové platformy. Význam jednotlivých vrstev je popsán v navazujících oddílech.

2.2.1 Vrstva „Hardware Abstraction Layer“

Vrstva „Hardware Abstraction Layer“ (HAL) se skládá ze sady nízkoúrovňových ovladačů napsaných v jazyce C++, které umožňují obsluhu periferií cílového mikrokontroleru. Jedná se tedy o vrstvu hardwarově zcela závislou, která je spjata s konkrétní cílovou platformou a interaguje přímo s hardware. Při portaci frameworku na novou platformu je zcela nezbytné tuto vrstvu přizpůsobit dle potřeby konkrétního hardware. Některé ovladače jsou pro běh .NET MF povinně vyžadované a musí být implementovány (jde např. o sériové komunikační rozhraní USART, řízení hodin, časovačů, přerušení nebo nízkopříkonových režimů MCU). Ostatní ovladače jsou volitelné a v případě jejich nevyužití v konkrétní cílové platformě je možné je ignorovat a stačí ponechat výchozí prázdnou implementaci (tzv. „pahýl“ – *stub*). O procesu portace .NET MF, organizaci zdrojového kódu nativních ovladačů a rozšiřování funkcionality frameworku bude podrobněji pojednáno v kapitole 5.

2.2.2 Vrstva „Platform Abstraction Layer“

Součást „Platform Abstraction Layer“ (PAL) se nachází ihned nad vrstvou HAL a je navržena tak, aby byla zcela nezávislá na cílové platformě. Vytváří spojovací složku mezi jádrem CLR a vrstvou HAL pro obsluhu hardware. Vrstva PAL specifikuje množinu všech ovladačů periferií, které .NET MF podporuje a které je tedy možné v rámci vrstvy HAL implementovat.

2.2.3 Common Language Runtime

Common Language Runtime (CLR) je vlastním jádrem .NET MF, neboť zajišťuje samotné vykonávání intermediárního kódu aplikačního programu. Jedná se o na platformě nezávislou vrstvu a tedy ji není třeba při procesu portace frameworku žádným způsobem měnit. CLR je implementován v nativním kódu pomocí jazyka C++ jako jediné vlákno, které má k dispozici veškeré zdroje cílové platformy, tj. paměť i výpočetní výkon mikrokontroleru. CLR kromě samotné interpretace a vykonávání mezikódu poskytuje aplikačnímu programu služby správy paměti, plánování vláken, synchronizace, výjimky, přístup k údajům z časovače a obdobnou funkcionalitu.

2.2.4 Podpůrné třídy – „Class Library“

Vrstva *Class Library* poskytuje funkcionalitu, které lze použít v rámci z uživatelské aplikace. Oproti „plnému“ .NET MF z prostředí PC se jedná pouze o redukovanou podmnožinu tříd. Jednotlivé třídy jsou rozděleny do hierarchických jmenných prostorů, zapisovaných pomocí standardní tečkové notace. Významnou skupinou z hlediska vývoje vestavěných systémů je balík tříd `Microsoft.SPOT`, který uživatelské aplikaci dovoluje přístup k typickým periferiím vestavěného zařízení, jako je sériová komunikační linka USART, rozhraní SPI, obecné vstupně výstupní piny GPIO a podobné. Dostupné třídy v rámci *Class Library* jsou dobře zdokumentované v API referenci na webu firmy Microsoft [33]. Vývoji aplikací v .NET MF prostřednictvím *Class Library* se věnují i tištěné publikace, např. [20].

2.2.5 Uživatelská aplikace

Nejvyšší vrstvou na schématu z obr. 2.1. je vrstva obsahující uživatelskou aplikaci a knihovny. Jedná se o aplikaci napsanou v jazyce C# (popř. Visual Basic) v prostředí aplikace Visual Studio a přeloženou do intermediárního jazyka CIL. V rámci .NET MF je do vestavěného zařízení možné nahrát vždy právě jednu aplikaci, může však být složena z více vláken. Plánování a vykonávání aplikačních vláken provádí výše uvedený interpret CLR. Uživatelská aplikace je tedy odstíněna od specifik cílové platformy a komunikuje s hardware výhradně pomocí aplikačních knihoven a ty následně využijí služeb nižších vrstev PAL a HAL.

2.3 Vývoj aplikací v .NET MF

Před zahájením vývoje vestavěných aplikací v .NET MF je nutné provést tyto kroky:

1. **Instalovat Microsoft Visual Studio** (plná verze nebo verze „Express“). Je nezbytné zvolit takovou verzi MS Visual Studia, která odpovídá použité verzi .NET MF. Pro verzi .NET MF 4.2 se jedná o Visual Studio 2010. Verze 4.3 vyžaduje Visual Studio 2012.

2. **Instalovat softwarový balík .NET MF SDK** [32]. Jedná se o doplněk pro MS Visual Studio, který obsahuje potřebné knihovny pro vývoj aplikací pro .NET MF a zároveň rozšíří schopnosti Visual Studia o možnost komunikace s externím vestavěným vývojovým přípravkem. Verze .NET MF SDK se musí shodovat s verzí .NET MF obsaženého v cílovém zařízení.

3. **Nahrát firmware .NET MF do cílového zařízení** – volitelný krok. V případě komerčně dostupných vývojových přípravků je zpravidla .NET MF již výrobcem předinstalován. U vlastních hardwarových zařízení nebo při aktualizaci firmware na novější verzi je třeba opatřit si binární obraz frameworku (buď ručním překladem ze zdrojového kódu, nebo v binární podobě od výrobce cílového hardware) a nahrát ho do nevolatilní paměti Flash vývojového přípravku. Konkrétní způsob provedení této operace se liší dle schopností konkrétní hardwarové platformy a vhodným rozhraním pro tuto operaci může být například rozhraní JTAG.

Po dokončení výše uvedených kroků postačí pro komunikaci mezi PC aplikací MS Visual Studio a hardwarovým přípravkem pouze běžný USB kabel. Prostředí Visual Studio komunikuje pomocí USB rozhraní mikrokontroléru přímo s interpretem CLR a kromě samotného nahrání aplikace poskytne standardní operace nutné pro ladění, například zastavení vykonávání programu pomocí *breakpoints*, krokování nebo prohlížení okamžitého obsahu proměnných. Výhodou tohoto přístupu je snížení ceny výsledné vývojové platformy, protože není vyžadován externí ladicí adaptér, jako by tomu bylo při nativním programování zařízení pomocí JTAG. Mimo USB rozhraní je pro účely ladění podporována také sériová linka nebo síťové připojení Ethernet, nicméně u komerčních výrobků podporujících .NET MF se využitím těchto alternativních rozhraní nesetkáváme.

Přehledně zpracované instrukce pro zprovoznění vývojového prostředí .NET MF čtenář nalezne např. v dokumentu [19]. O hardwarových přípravcích s podporou .NET MF, které je v současné době (první polovina roku 2013) možné na trhu sehnat, bude podrobněji pojednáno v oddíle 3.3.

Kapitola 3

Specifikace vývojové platformy

Motivací pro návrh nové vývojové platformy jsou aktuální trendy v oblasti vestavěných systémů. Zaměříme-li se na segment mikrokontrolérů, typickým rysem pro tuto doménu je značná fragmentace trhu a dlouhý životní cyklus výpočetních architektur, což je patrné zejména při pohledu na nabídku 8-bitových čipů.

Požadavky na zvyšování výkonu a schopností vestavěných systémů vedou v oblasti mikrokontrolerů k prosazování 32-bitových architektur, což je trend, který se v budoucnu bude nepochybně dále prohlubovat. Trh s 32-bitovými mikrokontroléry je jednotnější než v případě 8-bitových architektur a významnou roli zde zaujímá architektura ARM, která dle statistik [8] připadá na 30 % tohoto segmentu. Z pohledu budoucí perspektivy je proto žádoucí, aby nově navrhovaná platforma obsahovala jako výpočetní element právě 32-bitový mikrokontrolér typu ARM. Významný rys architektury ARM vzhledem k tématu práce je také fakt, že pro ni existuje podpora v rámci .NET Micro Frameworku.

Text této kapitoly se zaměřuje na aktuální situaci na trhu s vestavěnými systémy. V prvních dvou oddílech jsou stručně představeny mikrokontrolery s architekturou ARM a programovatelné obvody FPGA, výrobci těchto komponent a produktové řady, které připadají v úvahu pro zamýšlenou práci. Další sekce poskytuje přehled o současných komerčně dostupných hardwarovým platformách, které jsou blízké zadání práce, a zvažuje, je-li návrh nové platformy s ohledem na stav trhu opodstatněný. V závěru kapitoly jsou shrnuty požadavky na novou platformu ve formě specifikace klíčových parametrů výrobku.

3.1 Mikrokontrolery ARM

Historie procesorů s architekturou ARM sahá do 80. let 20. století a její přehledné shrnutí čtenář nalezne např. v článku [43]. Jedná se o 32-bitovou procesorovou architekturu typu RISC, která je vyvíjena společností ARM Ltd. Výpočetní jádra typu ARM se uplatňují jednak v oblasti výkonných aplikačních procesorů, které jsou mimo rozsah této práce (řada ARM Cortex-A), jednak v oblasti mikrokontrolerů (jádra ARM Cortex-M), které jsou svým charakterem vhodné pro zamýšlenou vývojovou desku.

Řada ARM Cortex-M [7] obsahuje podskupiny procesorů odstupňované dle výkonnosti a šíře své instrukční sady a označené čísly jako Cortex-M0, M0+, M1, M3 a M4 (seřazeno vzestupně podle výkonnosti a složitosti jádra). Společnost ARM Ltd. přímo konkrétní fyzické komponenty nenabízí, ale prodává licence (*intellectual property*) výrobcům polovidičových prvků, kteří produkují výsledné obvody nebo je integrují do složitějších systémů na čipu. K hlavním výrobcům mikrokontrolerů s jádry ARM Cortex-M patří společnosti

Atmel, Freescale, NXP, STMicroelectronics a Texas Instruments. Portfolia výrobců jsou dle názoru autora práce z pohledu ceny a parametrů do značné míry srovnatelné.

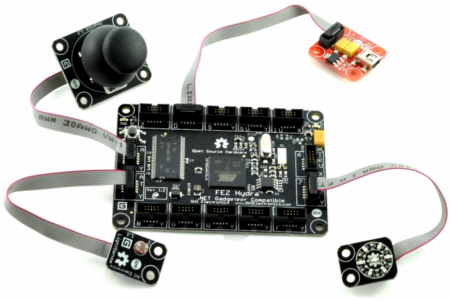
3.2 Programovatelná hradlová pole FPGA

Hradlová pole FPGA jsou digitální integrované obvody, které jsou výrobcem navrženy tak, že jejich funkci může zákazník–vývojář sám definovat a měnit. Jedná se o technologické nástupce programovatelných polí typu CPLD (*Complex Programmable Logic Devices*). Vyznačují se složitou nehomogenní vnitřní strukturou, která je tvořena zejména maticí konfigurovatelných logických bloků a propojovací logickou sítí. Funkci obvodu FPGA lze popsat v jazyce určeném pro popis hardware (např. VHDL nebo Verilog). Tento popis se pomocí softwarových nástrojů výrobce čipů FPGA transformuje do podoby binárních dat, tzv. *konfiguračního řetězce*. Nahráním konfiguračního řetězce do obvodu FPGA se definuje nastavení vnitřních konfigurovatelných elementů čipu a obvod FPGA jako celek začne plnit požadovanou funkci. O většinový podíl na trhu s obvody FPGA se dělí společnosti Altera a Xilinx.

3.3 Současný stav na trhu s vývojovými přípravky

Zařízení, které by splnilo požadavky vymezené zadáním, tedy integraci mikrokontroleru ARM Cortex-M a programovatelného hardwarového pole typu FPGA a zároveň bylo vybaveno podporou .NET Micro Frameworku, v současnosti na trhu neexistuje. Při analýze komerčně dostupných možností proto zmiňme platformy, které se tomuto cíli alespoň z části blíží. Nejdříve uvedeme mikroprocesorové platformy podporující .NET Micro Framework bez požadavku na integraci čipu FPGA. V druhé části zanalyzujeme vývojové platformy, které jsou tvořeny kombinací mikrokontroleru a FPGA, přičemž upustíme od požadavku na podporu .NET Micro Frameworku.

(a) Hardwarový přípravek Netduino [37]



(b) Modulární platforma .NET Gadgeteer (na obr. je zařízení FEZ Hydra výrobce GHI Electornics [1])

Obrázek 3.1: Příklady hardwarových platforem pro .NET Micro Framework

3.3.1 Mikroprocesorové platformy s podporou .NET MF

Nabídka vývojových a prototypovacích platforem primárně určených pro běh .NET MF je široká. Mezi významné výrobce patří společnosti *GHI Electronics*[1], *Secret Labs*[2] a aliance *Mountaineer Group*[3]. Pro lepší orientaci na trhu s nabídkou platforem pro .NET MF je možné existující výrobky rozdělit do těchto kategorií:

- **Zařízení formátu Arduino**. Jedná se o výrobky, které navazují na popularitu vývojových přípravků Arduino[4] a jejich konektory jsou rozmístěny ve formátu, který je s platformou Arduino kompatibilní. Tato zařízení je možné doplňovat a rozšiřovat jejich funkčnost pomocí modulů nazývaných „štíty“ (*shields*). Příkladem vývojového kitu této kategorie je přípravek Netduino 2 společnosti Secret Labs (na obr. 3.1a) nebo deska FEZ Panda II od firmy GHI Electronics.

- **Zařízení typu .NET Gadgeteer**, což je platforma navržená společností Microsoft a vybudovaná nad .NET MF, která nachází uplatnění především při výuce vestavěných systémů a různých „hobby“ a zájmových činnostech. Umožňuje konstruovat zařízení modulárním způsobem a připomíná elektronickou „stavebnici“. Zařízení patřící do platformy .NET Gadgeteer se dělí na *základní desky* osazené mikrokontrolerem umožňující běh .NET MF a *periferní moduly*, které je možné se základními deskami propojit pomocí speciálních zásuvek a plochých kabelů. Informace o platformě .NET Gadgeteer a odkazy na související informace lze nalézt v [28]. Jeden z příkladů základních desek – zařízení FEZ Hydra – je zachycen na obr. 3.1b.

- **Moduly pro integraci do vestavěných systémů**. Zatímco výrobky spadající do předchozích dvou kategorií se uplatní především ve fázi vývoje a prototypování vestavěných zařízení, zařízení typu „moduly“ slouží pro produkční nasazení jako součást rozsáhlejších systémů a výrobků. Jejich typickým rysem je kompaktní provedení. Reprezentativními zástupci této kategorie jsou výrobky Netduino Mini (jde o redukovanou verzi kitu Netduino do formátu DIL16, který lze vložit do standardní patice pro integrované obvody) nebo modul G120 firmy GHI Electronics (výrobek o rozměrech přibližně 4 x 2,5 cm určený pro povrchovou montáž na desku plošných spojů cílového zařízení).

- **Ostatní vývojové přípravky**. Do této skupiny lze zařadit vývojové desky, které nebyly původně navrženy pro běh .NET MF, ale tento framework pro ně byl dodatečně upraven a portován. Příkladem takového zařízení je vývojová deska STM32F4 Discovery[5] od společnosti ST Microelectronics.

Společným rysem prakticky všech vývojových přípravků určených pro běh .NET MF je uplatnění 32-bitových mikrokontrolerů řady ARM. Převažují čipy typu ARM Cortex-M3 a M4 od společností Atmel, ST Microelectronics a NXP – jedná se o platformy, na které je již .NET Micro Framework portován a je zde patrná snaha výrobců v co nejvyšší míře znovuvyužít existující a ověřený kód.

Ceny produktů se pohybují v přepočtu v rozmezí 500 – 2500 Kč. Levnější výrobky jsou vybaveny zpravidla pouze USB rozhraním pro připojení k hostitelskému PC, tlačítky,

[1] http://www.ghielectronics.com/
[2] http://netduino.com/
[3] http://www.mountaineer.org/
[4] http://arduino.cc/en/
[5] http://www.st.com/web/en/catalog/tools/FM116/SC959/SS1532/PF252419

indikačními LED diodami a vývody pro připojení rozšiřujících modulů. Desky vyšší cenové hladiny jsou osazeny dalšími periferiemi; často se jedná o čtečku paměťových karet MicroSD, rozhraní sítě Ethernet nebo sběrnici CAN.

3.3.2 Komerčně dostupné platformy integrující ARM Cortex-M a FPGA

Nabídka vývojových kitů integrujících mikrokontroler a rekonfigurovatelný obvod FPGA v jediném zařízení je velmi omezená. Jako zástupce lze zmínit přípravek FITkit [12] používaný při výuce vestavěných systémů na FIT VUT v Brně, který je však dostupný pouze studentům fakulty a který obsahuje jen 16-bitový mikrokontroler s architekturou MSP430. Jinou platformou kombinující reprogramovatelnou logiku a mikroprocesor jsou součástky Zynq [51] firmy Xilinx. Zde je ale integrováno výkonné aplikační jádro ARM Cortex-A9, které je určeno pro běh plnohodnotných operačních systémů (např. Linux nebo Android). Tato komponenta spadá do kategorie vestavěných mikroprocesorů (nikoli mikrokontrolerů) a je mimo rozsah této práce.

Pravděpodobně jediným v současnosti komerčně dostupným vývojovým přípravkem, který velmi dobře odpovídá požadavkům a očekáváním vysloveným výše, je deska Xynergy-M4 Board od firmy SILICA. Toto zařízení integruje mikrokontroler řady STM32F4 od ST Microelectronics (jedná se o ARM Cortex-M4) a FPGA obvod Spartan-6 firmy Xilinx. Specifikaci tohoto produktu čtenář nalezne na webových stránkách výrobce [39], cena činí v přepočtu přibližně 5500,- Kč.

3.4 Specifikace požadavků na vyvíjenou hardwarovou platformu

Nepostradatelnými výpočetními prvky vyvíjené platformy, jejichž uplatnění přímo plyne ze zadání, je již zmiňovaná dvojice komponent – hradlové pole FPGA a 32-bitový mikrokontroler (MCU) ARM Cortex-M. Dalším klíčovým požadavkem je umožnit na cílové desce běh .NET MF.

Aby bylo možné účelně využít zařízení, které je tvořeno koexistencí MCU a FPGA, je nutné ho vybavit komunikačním kanálem pro obousměrnou výměnu dat mezi těmito dvěma komponentami. Od této komunikační linky se očekává jednak dostatečná propustnost, jednak možnost při komunikaci logicky oddělit různé podsystémy, které v rámci čipu FPGA mohou být obsaženy, tedy být schopen „adresovat“ data konkrétnímu podsystému.

Nově navržená platforma má sloužit jako podpůrný prostředek při vývoji vestavěných zařízení, popřípadě jako pomůcka při výuce principů vestavěných systémů, což se promítá do specifikace jako tyto požadavky:

- **Požadavek na vybavenost běžnými komunikačními rozhraními**. Jedná se zejména o rozhraní sítě Ethernet, rozhraní USB, CAN 2.0, sériové sběrnice SPI, USART, I2C a sada obecně použitelných vstupně/výstupních vývodů (GPIO).
- **Požadavek na optimalizaci výrobku vzhledem k výrobním nákladům**. Důvodem je konkurenceschopnost vůči stávajícím produktům na trhu a možnost oslovit výrobkem i neziskový a vzdělávací segment trhu.

Redukce ceny výrobku bude ve vztahu k prototypu platformy zajištěna následujícími opatřeními. Maloobchodní cena komponent pro realizaci prototypu při započítání DPH

Požadavek	**Způsob realizace**
Výpočetní prostředky	– mikrokontroler (MCU) typu ARM Cortex-M – rekonfigurovatelný čip FPGA
Komunikace FPGA a MCU	– linka s dostatečnou propustností – komunikační protokol pro adresaci podsystémů v FPGA
Vyžadovaná rozhraní	– Ethernet 10/100 Base-T/TX – USB 2.0 On-The-Go – sběrnice SPI, I2C a USART vyvedené na – rozšiřující konektory pro vývody GPIO čipů FPGA i MCU
Preferovaná rozhraní	– Rozhraní CAN (Controller Area Network) – slot pro paměťovou kartu Micro-SD – kodek umožňující zpracovat audiosignály – analogový výstup pro monitor (VGA)
Programování přípravku	– primárně uvažovat využití platformy .NET MF – umožnit programování i bez .NET MF (JTAG) – umožnit konfiguraci FPGA adaptérem Xilinx Platform Cable
Optimalizace návrhus ohledem na cenu zařízení	– limit pro maloobchodní cenu součástek je 2000 Kč vč. DPH – použít pouze dvouvrstvou desku plošného spoje
Rozměry výrobku	– plocha desky (šířka x délka) nepřesahující 1 dm^2
Napájení zařízení	– umožnit napájení prostřednictvím rozhraní USB
Upevnění přípravku	– montážní otvory pro distanční sloupky (šroubky M3)

Tabulka 3.1: Specifikace požadavků na cílovou platformu

nepřesáhne 2000 Kč. Plošný spoj výrobku bude navržen na dvouvrstvé desce s prokovy a uplatnění desek s více než dvěma vrstvami mědi bude kvůli vyšším výrobním nákladům vyloučeno. Plocha výsledného plošného spoje nepřesáhne 1 dm^2.

Napájení systému bude řešeno prostřednictvím rozhraní USB z důvodu pohodlí práce, jednoduchosti a obecné dostupnosti tohoto rozhraní. Další nutnou podmínkou je vybavenost desky konektory s rozhraními JTAG, a to pro oba obvody – MCU i FPGA. Cílem je umožnit programování systému prostřednictvím JTAG adaptérů třetích stran. Tato varianta se využije například v případě, kdy uživatel bude vyvíjet aplikaci přeloženou přímo do nativního kódu architektury ARM běžící bez přítomnosti .NET MF a nevyužije se tedy možnost programovat aplikaci prostřednictvím USB rozhraní a služeb vestavěného běhového prostředí .NET MF CLR.

Úplná specifikace platformy je shrnuta v tabulce 3.1.

Kapitola 4

Návrh prototypu vývojové desky

První část kapitoly 4 pojednává o volbě klíčových komponent pro realizovaný systém a o procesu návrhu obvodového zapojení vývojového přípravku. Stručně jsou zmíněny varianty, které byly v případě jednotlivých podsystémů zvažovány a důvody, které vedly ke konkrétním zvoleným řešením. Nedílnou součástí textu kapitoly jsou odkazy na datové listy a aplikační poznámky použitých součástek, ze kterých bylo ve fázi návrhu čerpáno.

Druhá část textu se zabývá návrhem desky plošných spojů (DPS). V závěru kapitoly jsou pak nastíněny problémy, které bylo ve fázi návrhu obvodu, návrhu DPS a oživování prototypu nutné řešit.

4.1 Volba hlavních komponent

Na základě úvah a specifikace shrnuté v oddíle 3.4 je nutné zvolit konkrétní hradlové pole FPGA a konkrétní mikrokontroler ARM Cortex-M, který bude vhodný pro použití v uvažovaném systému.

4.1.1 Výběr mikrokontroléru

Při volbě konkrétní součástky MCU se autor práce rozhodoval podle těchto kriterií:

- existující podpora cílového mikrokontroleru v rámci .NET MF,
- dostupnost vývojových přípravků s daným obvodem,
- vybavenost mikrokontroleru periferiemi uvedenými ve specifikaci,
- dostupnost a cena součástky,
- škálovatelnost do budoucna (např. existence výkonnějších alternativ, které jsou dostupné v pinově kompatibilních pouzdrech),
- předchozí zkušenost autora s prací na dané platformě.

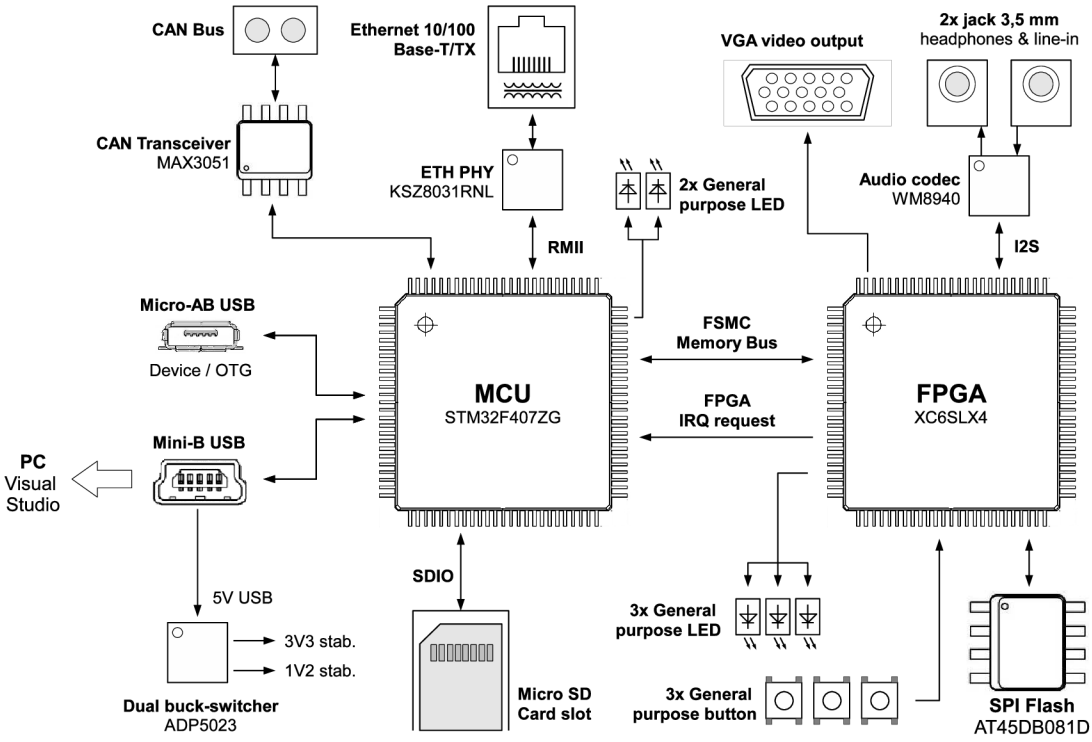
Autor práce se zaměřil na nabídku vhodných mikrokontrolerů v portfoliu hlavních výrobců, tj. společností Atmel, Freescale, STMicroelectornics a Texas Instruments. Po zvážení výhod a nevýhod dle uvedených kriterií redukoval výběr na čipy společnosti ST Microelectronics, konkrétně na produktovou řadu STM32F4. Jedná se o výkonné mikrokontrolery řady ARM Cortex-M4 s vybavené hardwarovou jednotkou pro operace s čísly v plovoucí řádové čárce a rozšířeným instrukčním souborem s instrukcemi typu DSP. Konkrétní zvolený

obvod má označení `STM32F407ZG` [40] je k dostání ve vývodovém pouzdře typu LQFP se 144 piny. Pracuje na frekvenci až 168 MHz a disponuje 1024 kB paměti Flash a 192 kB paměti SRAM. Škálovatelnost platformy do budoucna je zajištěna díky dostupnosti výkonnějších variant mikrokontrolerů z řady STM32F4, které jsou se zvolenou součástkou kompatibilní po stránce instrukční i co do rozmístění vývodů. Existuje tedy možnost mikrokontroler v budoucnu prostým způsobem zaměnit za jeho výkonnější variantu bez nutnosti zásahů do zbylých částí obvodového zapojení.

4.1.2 Výběr hradlového pole FPGA

Při výběru konkrétního hradlového pole byla uplatněna stejná kriteria jako při volbě MCU. Netriviální složitost čipů FPGA a jejich obvodového zapojení však vyžaduje zaměřit se na ty produkty, s nimiž měl autor práce již předchozí zkušenost. Výběr se z tohoto důvodu omezil na portfolio firmy Xilinx.

S ohledem na účel výsledné platformy a vymezenou na cenu výsledného zařízení připadají v úvahu produktové řady Xilinx Spartan-3 a Spartan-6. Řada produktů Spartan-3 byla z výběru vyloučna, jelikož firma Xilinx již tyto komponenty nedoporučuje pro nasazení v nově navrhovaných produktech. Při volbě čipu FPGA ze třídy Spartan-6 se jako nejvíce omezující kriterium jeví požadavek na typ pouzdra součástky. Nebudeme-li uvažovat komponenty v pouzdrech typu BGA, jejichž zapojení na pouze dvouvrstvé desce plošných spojů je nerealizovatelné, zbývají pouze dvě možné součástky: `XC6SLX4` a `XC6SLX9`. Pro konstrukci prototypu byla zvolena první z nich – v rámci řady Spartan-6 se jedná se o kapacitně nejmenší variantu obvodu FPGA. V případě potřeby je možné v budoucnu platformu bez dalších zásahů škálovat záměnou této komponenty za její kapacitně větší variantu `XC6SLX9`.



Obrázek 4.1: Blokové schéma navržené vývojové platformy

4.2 Návrh obvodového zapojení

Navržené obvodové zapojení lze rozdělit do těchto podsystémů: napájecí obvody, podsystém mikrokontroleru, podsystém čipu FPGA a souvisejících obvodů, periferie mikrokontroleru a periferie čipu FPGA. Blokové schéma navržené platformy zachycuje obr. 4.1 a úplné schéma elektrického zapojení je obsahem přílohy A. Následující odstavce se zaměřují zejména na klíčové komponenty obvodu a diskutují také případná méně standardní řešení, která jsou hodna zvláštního komentáře.

4.2.1 Zapojení mikrokontroléru

Při integraci zvoleného MCU (komponenta `U4` ve schématu) do vestavěného systému je nutné zvolit způsob zavádění firmware pomocí pinů BOOT0 a BOOT1, které se nastaví do příslušných logických úrovní předepsaných dokumentací podle požadovaného režimu startu MCU. Existují tři možnosti zaváděcí konfigurace:

1. start firmware z paměti flash, což je typická výchozí varianta,
2. zavádění firmware z paměti RAM a
3. spuštění speciálního firmware typu „bootloader“ definovaného výrobcem čipu a uloženém v nesmazatelné paměti ROM mikrokontroleru.

Třetí možnost, tj. režim bootloaderu označovaný také jako režim DFU (*Device Firmware Update*) může být užitečný v případě nutnosti aktualizovat firmware bez použití rozhraní JTAG. Obvodové zapojení s touto variantou počítá a mód DFU lze vyvolat pomocí zkratovací propojky na konektoru `CN11`.

Hlavním hodinovým zdrojem MCU je krystal `X3` s frekvencí 8 MHz. Pro možnost použít jednotku v rámci MCU hodin reálného času (Real-time clock) a funkci kalendáře je doplněn druhý krystal `X2` oscilující na frekvenci 32,768 kHz.

Při zapojení MCU bylo čerpáno především z datového listu konkrétní součástky [40] a z referenčního manuálu produktové řady STM32F4 [41].

4.2.2 Zapojení obvodu FPGA

Zdroj hodinového signálu Zdroj přesného hodinového kmitočtu pro obvody FPGA obvykle bývá realizován pomocí krystalového oscilátoru v podobě integrovaného obvodu. V navrženém systému bylo z důvodu redukce ceny a počtu komponent zvoleno jiné řešení. Využívá se mechanismus MCO (*Main Clock Out*) mikrokontroleru, který dovoluje přivést interní hodinový signál MCU na výstupní pin obvodu. V popisovaném zapojení jde konkrétně o pin s označením `MCO1` produkující hodinový signál o frekvenci 8 MHz.

Uchování konfigurační informace V každém produktu obsahujícím čip FPGA je nutné vyřešit, jakým způsobem bude uchován příslušný konfigurační řetězec, jelikož konfigurační paměť obvodu FPGA je volatilní a její obsah je ztracen při přerušení napájecího napětí. Čipy Spartan-6 jsou z pohledu konfiguračních metod flexibilní a nabízí řadu variant, které se liší rychlostí, cenou, složitostí zapojení a vhodností pro produkční nasazení. Pro přehled dostupných metod konfigurace je čtenář odkázán do dokumentu [47].

Jednou z variant je doplnit obvod FPGA pamětí z řady Xilinx Platform Flash. Výhodou je ověřenost tohoto řešení a výborná kompatibilita s nástroji firmy Xilinx. Nevýhodou je vysoká cena paměťové součástky součástky.

Alternativní metodou, která byla zvolena pro navrženou platformu, je propojit obvod FPGA s běžně komerčně dostupnou pamětí typu flash vybavenou standardním rozhraním SPI. Tuto paměť je posléze možné programovat prostřednictvím softwarových nástrojů firmy Xilinx a rozhraní JTAG speciálním postupem nazvaným nepřímé programování – *indirect programming* [52]. Procedura probíhá následovně. Po připojení JTAG adaptéru k JTAG konektoru na vývojové desce proběhne nahrání speciální konfigurace, která obvodu FPGA umožní pracovat jako „most“ mezi rozhraním JTAG a pamětí s rozhraním SPI. Pomocí takto ustanoveného spojení, v němž obvod FPGA funguje jako prostředník, dojde k nahrání konfigurační informace z hostitelského počítače do SPI flash paměti. Výhodou řešení je velmi nízká cena – oproti variantě s pamětí typu Xilinx Platform Flash došlo k redukci ceny konfiguračního řešení na přibližně osminu. Konkrétní typy pamětí SPI flash, které jsou v rámci nepřímého programování podporovány, jsou uvedeny v dokumentaci k softwarovému nástroji Xilinx iMPACT [46]. Pro zamýšlený obvod byla zvolena flash paměť `AT45DB081D` od firmy Adesto Technologies; ve schématu zapojení jde o komponentu `U3`.

Nahrávání konfiguračního řetězce U v současnosti navrženého obvodového zapojení je pro nahrávání konfigurace do obvodu FPGA a pro nepřímé programování zmíněné v předchozím odstavci nutné použít vhodný adaptér pro rozhraní JTAG, např. Xilinx Platform Cable. Nutnost zakoupení takového adaptéru ovšem zvyšuje celkové náklady, které je nutné vynaložit při používání navržené prototypové platformy.

Jako řešení, které je námětem pro budoucí vylepšení platformy, by mohlo být použití technologie *Xilinx Virtual Cable*. Ta dovoluje „tunelovat“ protokol JTAG prostřednictvím síťového protokolu TCP, což po doplnění vhodného firmware v rámci MCU mohlo vést k úplnému nahrazení externího JTAG adaptéru. Detaily konceptu jsou shrnuty např. v dokumentu [15]. Navržené obvodové zapojení je pro toto budoucí rozšíření připravené, neboť rozhraní JTAG čipu FPGA je propojeno s obecnými piny GPIO čipu MCU a ten tedy může vůči čipu FPGA vystupovat v roli programovacího adaptéru.

4.2.3 Komunikace mezi obvody FPGA a MCU

Při návrhu obvodového zapojení komunikační linky mezi obvody FPGA a MCU byly zohledněny požadavky specifikace stanovené v oddíle 3.4, tj. obousměrnost komunikace, dostatečná propustnost linky a schopnost rozlišení (adresace) dat příslušejících různým podsystémům, které mohou být obsaženy v rámci čipu FPGA.

Pro realizaci tohoto komunikačního rozhraní byly zvažovány následující přístupy:

- **Využít sériových komunikačních linek mikrokontroléru**, např. rozhraní USART nebo SPI. Tato varianta vyniká jednoduchým obvodovým zapojením, a minimálním počtem vodičů na desce plošných spojů. Z důvodu sériového charakteru komunikace ale nenabízí příliš vysokou propustnost. Dále pro splnění podmínky adresné komunikace by bylo nutné nad prostou sériovou linkou vybudovat vlastní komunikační protokol.

- **Komunikovat prostřednictvím paralelní fronty FIFO a signálů typu handshake**. Při této variantě komunikace by se přenos dat odehrával prostřednictvím paralelní sběrnice tvořené sadou vstupně výstupních pinů (GPIO), která by byla řízena firmwarem mikrokontroleru. Taková sběrnice by byla doplněna signály typu handshake – „data ready“ a „data accepted“. Uvedené komunikační linky by musely pro obousměrnou komunikaci být buď duplikované, nebo schopné pracovat v třístavovém

režimu. I když tento přístup umožní vyšší propustnost, opět není zohledněn požadavek na adresaci dat, který by opět musel být vyřešen navržením vlastního protokolu.

- **Periferie FSMC mikrokontroleru.** Jedná se o řadič statické paměti (*Flexible Static Memory Controller*), který je integrován na čipu MCU jako periferie. Použití tohoto řadiče umožňuje vybudovat paralelní obousměrnou komunikační linku a zároveň již z pricnipu fungování pamětí řeší adresaci komunikačních transakcí. Pro realizaci kompletního komunikačního rozhraní je nutné rozšířit .NET MF o možnost práce s periferií FSMC a také vytvořit podsystém řadiče linky FSMC v FPGA, který tomuto obvodu umožní chovat se navenek vůči MCU, jako by se jednalo o statickou paměť.

Po zvážení uvedených možností byla pro vytvoření komunikační linky mezi čipy MCU a FPGA zvoleno řešení využívající periferie FSMC obsaženou v mikrokontroleru. Tuto periferii lze nakonfigurovat pro práci v řadě režimů a se škálou komunikačních protokolů, které jsou popsány v referenčním manuálu MCU rodiny STM32F4 [41]. Z hlediska práce je vhodným režimem mód *Multiplexed Asynchronous NOR Flash Memory*. Signály paměťové sběrnice a komunikační protokol tohoto rozhraní bude podrobněji rozebrán v sekci 6.2, která se věnuje návrhu řadiče linky FSMC v obvodu FPGA.

4.2.4 Periferie mikrokontroleru

Při návrhu zapojení periferií mikrokontroleru byla brána na zřetel výše stanovená specifikace a dále požadavek na co nejvyšší užitné hodnoty výsledného výrobku při respektování omezení daných maximální stanovenou cenou prototypu. Na vývojovou desku byly integrovány a s MCU propojeny následující podsystémy.

Rozhraní USB

Použitý MCU disponuje dvěma nezávislými řadiči sběrnice USB, které jsou označené jako `USB_OTG_HS` a `USB_OTG_FS`. Obě tyto periferie je možné nakonfigurovat do režimů „zařízení“, „hostitel“ nebo „USB On-the-Go“.

V navrženém obvodovém zapojení jsou periferie USB využity takto. Řadič `USB_OTG_FS` je vyhrazen pro připojení vývojové desky k hostitelskému počítači a pro komunikaci vestavěného běhového prostředí .NET MF CLR s aplikací Visual Studio. Funguje tedy trvale v režimu „zařízení“. Z této sběrnice je také odebíráno napájecí napětí pro celý vestavěný systém. Rozhraní je zakončeno konektorem `CN8` formátu MiniUSB–B.

Druhá periferie `USB_OTG_HS` je zpřístupněna uživatelské aplikaci a může být použita dle potřeb programátora v libovolném ze tří dostupných módů činnosti. Je osazena konektorem MicroUSB–AB; ve schématu jde o komponentu `CN10`. Pro možnost pracovat v hostitelském režimu musí být toto USB rozhraní vybaveno spínatelným zdrojem napájecího napětí. V navrženém zapojení byl pro tento účel uplatněn spínač `STMP2151STR` [42], který zároveň plní funkci ochrany před nadměrným proudovým zatížením a nedovolí připojenému zařízení proudový odběr vyšší než než 500 mA.

Obě rozhraní USB jsou z důvodu ochrany před elektrostatickým výbojem (ESD, *Elektrostatic discharge*) vybavena ochrannými diodovými poli – jde o integrované obvody `U8` a `U10`.

Podsystém Ethernetu 10/100 Base-T/TX

Zvolený MCU obsahuje integrovanou implementaci vrstvy MAC (*Media Access Control*) pro síťové rozhraní Ethernet 10Base–T a 100Base-TX, nikoli však vrstvy fyzické (PHY). Proto je potřebné doplnit mikrokontroler ještě externím čipem fyzické vrstvy Ethernetu. S přihlédnutím k ceně výsledného řešení a počtu potřebných komponent byl jako čip fyzické vrstvy zvolen produkt `KSZ8031` [24] společnosti Micrel. Významnou předností součástky je fakt, že nepotřebuje jako zdroj hodinového taktu externí krystalový oscilátor, ale postačí běžný a nenákladný krystal s rezonančním kmitočtem 25 MHz. Kvůli redukci složitosti zapojení desky plošných spojů komunikuje čip fyzické vrstvy s MCU pomocí rozhraní RMII (*Reduced Media-independent Interface*) a nikoli „plného“ rozhraní MMI (*Media Independent Interface*). Rozhraní Ethernetu je zakončeno konektorem RJ–45 (`CN1`), který obsahuje integrované oddělovací transformátory.

Oživení podsystému Ethernetu a jeho úplná integrace do .NET MF přesahuje rozsah této diplomové práce a je námětem pro budoucí rozšíření projektu.

Sběrnice CAN 2.0B

Použitý MCU je vybaven rozhraním MAC sběrnice CAN 2.0B (*Controller Area Network*). Aby bylo navrženému zařízení umožněno po této sběrnici komunikovat, musí být MCU vybaven ještě externím čipem fyzické vrstvy CAN. V popisovaném zapojení tuto úlohu zastává součástka `MAX3051` do společnosti Maxim Integrated.

Rozhraní paměťové karty MicroSD

Pro komunikaci s paměťovou kartou typu MicroSD je zvolený mikrokontroler vybaven speciální integrovanou periferií SDIO [41]. Rozhraní SDIO podporuje paměťové karty ve formátech MicroSD i MicroSDHC. Alternativní variantou komunikace s paměťovou kartou, která však nebyla využita, by mohlo být i rozhraní SPI. To by však vyloučilo kompatibilitu s kartami MicroSDHC, které již rozhraním SPI vybaveny nejsou. Pro vložení paměťové karty je určen konektor `CN7`.

Konektory pro rozšiřující moduly

Je velmi žádoucí, aby mikrokontroler byl doplněn rozšiřujícími konektory umožňujícími připojení k externím zařízením a vnějším systémům. Navržený systém obsahuje dva konektory tohoto typu. Prvním z nich je rozhraní formátu UEXT (`CN14`), které obsahuje sběrnice UART, SPI a I2C a bylo navržen firmou Olimex a zveřejněn v podobě otevřené specifikace [36]. Výhodnou vlastností je dostupnost řady rozšiřujících modulů na trhu, které jsou s rozhraním UEXT kompatibilní.

Druhým rozšiřujícím prvkem je konektor `CN10` připojený k 14 obecně použitelných vstupně výstupních pinům MCU.

4.2.5 Periferie obvodu FPGA

Obvod FPGA je vybaven konektorem pro videovýstup ve formátu VGA, audio kodekem, tlačítky, indikačními LED diodami a rozšiřujícími konektory pro připojení vývojové platformy k externím systémům.

Videovýstup formátu VGA

Navržený hardwarový přípravek je schopen produkovat výstup v analogovém formátu VGA s barevným rozsahem 8 barev (1 bit, tj. jeden digitální výstup FPGA pro každý z barevných kanálů R, G a B). Toto jednoduché řešení je dáno omezeným počtem volných vývodů čipu FPGA. Tři linky nesoucí informaci o barvě právě vysílaného pixelu jsou doplněny dvěma signály pro horizontální a vertikální synchronizaci obrazových dat. Vzhledem k analogové povaze signálů s barevnou informací nelze použít výstupní piny FPGA přímo, ale je nutné omezit protékající proud sériově zapojenými rezistory (součástky `R22` až `R26`). Při návrhu zapojení bylo čerpáno z referenčního obvodu v na desce *Xilinx Spartan-3 FPGA StarterKit Board* [48].

Audio kodek

Zapojení navržené platformy bylo rozšířeno o audiokodek WM8940 [45] společnosti Wolfson Microelectronics, který je pomocí rozhraní I2S a konfigurační sériové linky SDIN připojen k čipu FPGA. Zprovoznění kodeku vyžaduje implementovat řadič těchto komunikačních rozhraní v čipu FPGA. Tato činnost je mimo rozsah práce a představuje námět pro rozšíření schopností platformy v budoucnu.

Tlačítka a indikační LED diody

Jednoduchými komponentami typickými pro vývojové platformy vestavěných systémů jsou indikační LED diody a tlačítka. I když se jedná o velmi jednoduché periferie, poskytují programátorovi nezbytné základní ovládací prostředky a možnost kontroly okamžitého stavu vestavěného zařízení.

K FPGA obvodu jsou přímo připojeny tři LED diody označené jako `LED2` až `LED4` a trojice tlačítek `B1` až `B3`, jejichž využití je plně pod kontrolou aplikačního programátora. Pro správné vyhodnocování stisku tlačítek je nezbytné vhodným způsobem zpracovat a potlačit oscilace způsobené zákmitem mechanického kontaktu v okamžicích sepnutí nebo uvolnění tlačítka (tzv. *debouncing*). O návrhu podsystému FPGA pro tuto operaci je pojednáno v oddíle 7.3, který popisuje demonstrační aplikace pro navrženou platformu.

Rozšiřující konektory čipu FPGA

Podobně jako v případě mikrokontroleru je čip FPGA vybaven rozšiřujícími konektory umožňujícími interagovat s externími systémy a zařízeními. V navrženém zapojení se jedná o konektory `CN12` a `CN13`. Každý z je propojen s 24 vstupně výstupními vývody čipu FPGA.

4.2.6 Podsystém napájení

Cílem podsystému napájení je transformovat napětí 5 V (± 0,5 V) dostupné v rámci sběrnice USB na stabilizované napájecí napětí pro veškeré komponenty na vývojové desce. Navržené obvodové zapojení vyžaduje dvě napájecí větve o napětích 3,3 V a 1,2 V – v přiloženém schématu jsou označeny jako `V3.3` a `V1.2`. Větev s napětím 1,2 V slouží výhradně pro napájení interní logiky obvodu FPGA (tj. vstupy FPGA označené jako `Vccint`). Větev s 3,3 V zajišťuje napájení zbylé části obvodu FPGA (vstupy `Vccaux` a `Vcco`n), MCU a všech ostatních komponent na desce.

Požadovaný minimální výstupní proud stabilizátoru není možné v případě řešeného zapojení možné přesně vyčíslit kvůli konfigurovatelnému charakteru obvodu FPGA, jelikož

Komponenta	Nejvyšší příkon v činném režimu
MCU STM32F4	93 mA
FPGA XC6SLX4	(závisí na aktuální konfiguraci)
Ethernet PHY KSZ8031	50 mA
Zvukový kodek WM8940	60 mA
MicroSDHC karta	40 mA
CAN 2.0B PHY MAX3051	35 mA
Celkový příkon (bez FPGA)	278 mA

Tabulka 4.1: Maximální možné příkony komponent na napájecí větvi `V3.3`

příkon je velmi závislý na momentální konfiguraci součástky, konkrétně především na množství využitých zdrojů a momentální pracovní frekvenci čipu FPGA. V zájmu prototypovací platformy je dosáhnout co největší univerzálnosti a je tedy nezbytné napájecí obvody dostatečným způsobem naddimenzovat. Maximální příkon komponent připojených k napájecí větvi `V3.3` bez započtení obvodu FPGA je shrnut v tabulce 4.1 a slouží jako výchozí bod pro výběr vhodného stabilizátoru.

Při návrhu podsystému napájení byly zvažovány dva přístupy:

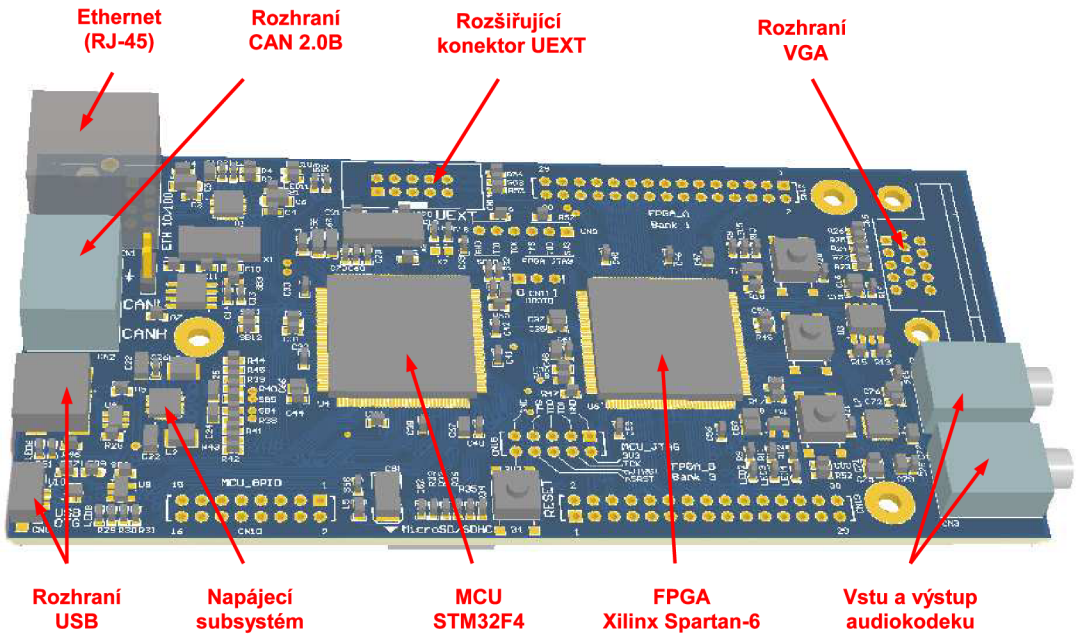
- **Využití sériového (kaskádního) zapojení dvojice lineárních stabilizátorů**. Výhodou tohoto přístupu je jednoduché a levné zapojení; nevýhodou nižší účinnost, vyšší rozměry komponent, větší tepelné vyzařování a s ním spojená nutnost chlazení a větší zabraná plocha na desce plošných spojů.
- **Využití spínaných stabilizátorů**. Toto řešení přináší vyšší účinnost při nižším zahřívání komponent a zpravidla nabízí vyšší výstupní proudy při stejných vstupních parametrech stabilizátoru ve srovnání s lineárními regulátory. Je však vykoupeno složitějším obvodovým zapojením a vyšší cenou komponent.

Po analýze dostupných možností byla zvolena metoda spínaných stabilizátorů a jako konkrétní obvod vybrána součástka `ADP5023` [6] od společnosti Analog Devices. Její výhodou je fakt, že na jednom čipu jsou integrovány dva nezávislé spínané snižující regulátory (*step-down/buck regulators*) a jeden lineární regulátor typu LDO. Dvojice integrovaných spínaných stabilizátorů se v rámci zapojení využije pro generování požadovaných stabilizovaných napětí 1,2 V a 3,3 V, přičemž maximální výstupní proud každé z větví nezávisle činí 0,8 A. Integrovaný lineární regulátor zůstává nevyužit.

Nezbytnou součástí napájecího subsystému je také velký počet tzv. oddělovacích kondenzátorů (*decoupling capacitors*), jejichž úkolem je sloužit jako nízkoimpedanční zdroj pokrývající krátkodobé výkyvy v proudovém odběru napájených součástek. Pro správnou funkci je nutné oddělovací kondenzátory umístit do fyzické blízkosti napájených komponent. Počet kondenzátorů, jejich kapacity a požadavky na fyzické umístění respektují doporučením výrobců v příslušných datových listech použitých součástek.

4.2.7 Problémy řešené při návrhu obvodového zapojení

Při návrhu obvodového zapojení bylo obtížným krokem navrhnout zapojení periferií MCU. Zvolený čip řady STM32F4 umožňuje pomocí vestavěného systému multiplexorů přiřadit konkrétnímu jednomu pinu až 16 alternativních funkcí, přičemž může docházet ke kolizím



Obrázek 4.2: Model navržené desky plošných spojů

Základní materiál DPS	FR4, tloušťka 1,5 mm
Počet vodivých vrstev	dvě vrstvy o tloušťce 18 µm
Průměr vrtaných otvorů	$\geq$ 0,3 mm
Prokovení otvorů	ano
Tloušťka spoje/mezery	$\geq$ 0,2 mm
Servisní potisk	na svrchní straně, bílá barva

Tabulka 4.2: Parametry použité výrobní technologie DPS

mezi více periferiemi „soupeřícími“ o jediný pin MCU. Naopak v jiných případech existuje více možných pinů, kam je možné konkrétní vývod periferie pomocí zabudovaných multiplexorů na čipu připojit. Celkové obvodové zapojení muselo tato fakta respektovat a pomocí vhodné volby a konfigurace vstupně výstupních pinů vyloučit zmíněné kolize. Dále bylo nutné obvodové zapojení navrhovat s ohledem na návazný proces tvorby DPS, aby bylo možné výsledný obvod realizovat na dvouvrstvé DPS.

4.3 Návrh desky plošných spojů

Návrh desky plošných spojů (DPS) byl proveden na oboustranné desce o rozměrech 13,7 cm x 6,5 cm, která je opatřena nepájivou maskou, prokovy a servisním potiskem (*silkscreen*) na svrchní straně. Deska je osazena převážně součástkami pro povrchovou montáž (SMT). Ve většině případů jde o součástky v pouzdrech o velikostech 0603, 0805 a vývodových pouzdrech s rozestupem pinů nejméně 0,5 mm. Cílem volby těchto komponent je kompromis mezi požadavky umožnit ruční osazení prototypu a zároveň dostáhnout kompaktních rozměrů zařízení. Omezení výrobní technologie, která bylo nutné dodržet, jsou shrnuta v tabulce 4.2.

3D model navrženého zařízení spolu s rozmístěním klíčových komponent zachycuje obr. 4.2.

Návrh DPS byl proveden za pomoci softwarového nástroje Altium Designer. Výrobní podklady DPS jsou obsahem příloh této technické zprávy a v elektronické podobě také na přiloženém disku CD.

4.3.1 Problémy řešené během práce s prototypem zařízení

Jelikož z hlediska zadání práce je prioritním úkolem zprovoznění komunikace mezi obvody FPGA a MCU a provedení portace .NET MF, nebylo vzhledem k časové náročnosti práce provedeno oživení některých z komponent zapojení uvedených výše. Součástky, které nebyly oživeny a zaintegrovány do .NET MF, jsou prvky, které výrobek obsahuje nad rámec zadání. Jedná se o fyzickou vrstvu Ethernetu, rozhraní paměťových karet MicroSD/SDHC a zvukový kodek. Tyto komponenty však zůstávají fyzickou součástí osazeného výrobku a jejich zprovoznění představuje námět pro budoucí rozšíření práce.

Při práci s prototypem desky se jako významný nedostatek ukázala nízká mechanická spolehlivost konektoru periferie USB_OTG_HS (jde o konektor typu MicroUSB–AB pro povrchovou montáž, ve schématu je to komponenta označená `CN10`). Při běžné manipulaci s deskou a připojování USB kabelu opakovaně došlo k mechanickém uvolnění této zapájené součástky a v souvislosti s tím i k poškození cest vedoucích ke konektoru na plošném spoji zařízení. Pro budoucí verzi prototypu je vhodné zvážit uplatnění jiného provedení konektoru, např. varianty ukotvené pomocí otvorů v DPS (tzv. THT, *Through-Hole Technology*) s vyšší mechanickou odolností.

Obrázek 4.3: Osazený prototyp navrženého zařízení

Kapitola 5

Portace .NET Micro Frameworku

První část kapitoly 5 zdůvodňuje, proč je nezbytné se v rámci této práce zabývat vnitřní strukturou .NET MF a mechanismy jeho portace a nestačí pouze uplatnit již existující nemodifikovaný open-source kód frameworku. Jelikož portace frameworku je velmi málo zdokumentovaná oblast, bude v druhé části kapitoly představena struktura zdrojového kódu .NET MF a překladový systém s cílem usnadnit čtenáři orientaci v procesu portace. Třetí část kapitoly zmiňuje konkrétní modifikace, které byly v kódu frameworku provedeny, aby byl umožněn jeho běh na nově navržené platformě. Na závěr bude pojednáno o tom, jaké kroky bylo nutné podniknout pro rozšíření frameworku o komunikaci po sběrnici FSMC, jak bylo navrženo v oddíle 4.2.3.

5.1 Motivace pro portaci .NET MF

Pro platformu STM32F4, která byla zvolena pro realizaci práce, neexistuje podpora v oficiálních zdrojových kódech .NET MF spravovaných firmou Microsoft [25], avšak tato podpora byla vytvořena třetí stranou – firmou Oberon Microsystems [34] – a uvolněna pod open-source Apache licencí. Slabinou této implementace je skutečnost, že při její tvorbě bylo počítáno s podporou pouze nákladných komerčních překladačů *ARM RVDS* a *Keil MDK*. Možnost rekompilace ze zdrojových kódů je přitom základní nutností, chceme-li framework upravit pro běh na nové platformě nebo ho rozšířit o nové funkce. Jedním z dílčích cílů práce je proto upravit stávající zdrojový kód frameworku do podoby, kterou bude možné pro architekturu STM32F4 přeložit volně dostupným překladačem GCC (*GNU Compiler Collection*).

5.2 Úvod do procesu portace .NET MF

Úprava .NET Micro Frameworku pro novou platformu je úkol, který se skládá z několika fází a složitost procesu závisí na tom, jaká je aktuální podpora v .NET MF pro jednotlivé části cílové platformy. Proces portace může probíhat například podle některého z následujících scénářů:

- **Prostý překlad .NET MF ze zdrojového kódu**. Tento nejjednodušší případ je vhodný v situaci, kdy již podpora cílové platformy existuje a je úplná. Vývojáři pak pouze stačí upravit např. konfigurační informace v některých hlavičkových souborech. Nový překlad frameworku je také nezbytný tehdy, pokud chceme uživatelské aplikaci

umožnit volat speciální rutiny implementované v nativním kódu, tzv. *interops*. Tento případ bude podrobněji rozebrán později.

- **Přidání nových ovladačů periferií**. Programátor může při portaci, pokud to cílová platforma dovoluje, využít existující kódovou základnu pro podobné podporované zařízení a port doplnit o ovladače pro periferie specifické pro konkrétní hardwarový přípravek.
- **Vytvoření zcela nového portu .NET MF nad samotným hardware**. Tento případ je nejsložitější z uvedených a vyžaduje vytvoření nové sady ovladačů vrstvy HAL (*Hardware Abstraction Layer*) a postupné zprovoznění portu.
- **Vytvoření portu nad existujícím operačním systémem**. Jedná se o speciální variantu, kdy požadujeme, aby .NET MF neinteragoval přímo s cílovým hardware, ale využil služeb již existujícího operačního systému. Může jít např. o operační systém reálného času běžící v mikrokontroleru nebo emulátor zařízení spuštěný na hostitelském počítači. Tento typ portu však není záměrem práce a nebude mu dále věnována pozornost.

5.2.1 Potřebné nástroje

Pro překlad frameworku je třeba nainstalovat vývojové prostředí Visual Studio firmy Microsoft; postačí i volně dostupné verze s označením Express, v tomto případě je nutné nainstalovat prostředí Visual Studio C++ Express i Visual Studio C# Express. Důvodem přítomnosti těchto softwarových nástrojů je fakt, že při překladu frameworku dochází i ke kompilaci podpůrných softwarových nástrojů pro operační systém Windows, které jsou napsány právě v uvedených jazycích.

Ke kompilaci samotného běhového prostředí .NET MF pro cílovou hardwarovou platformu je potřebné mít k dipozici odpovídající překladač (*crosscompiler*). V rámci této práce byl zvolen překladač GCC verze 4.7.3 pro architektury ARM Cortex z binární distribuce *GNU Tools for ARM Embedded Processors* [9], která je udržována přímo zaměstnanci firmy ARM.

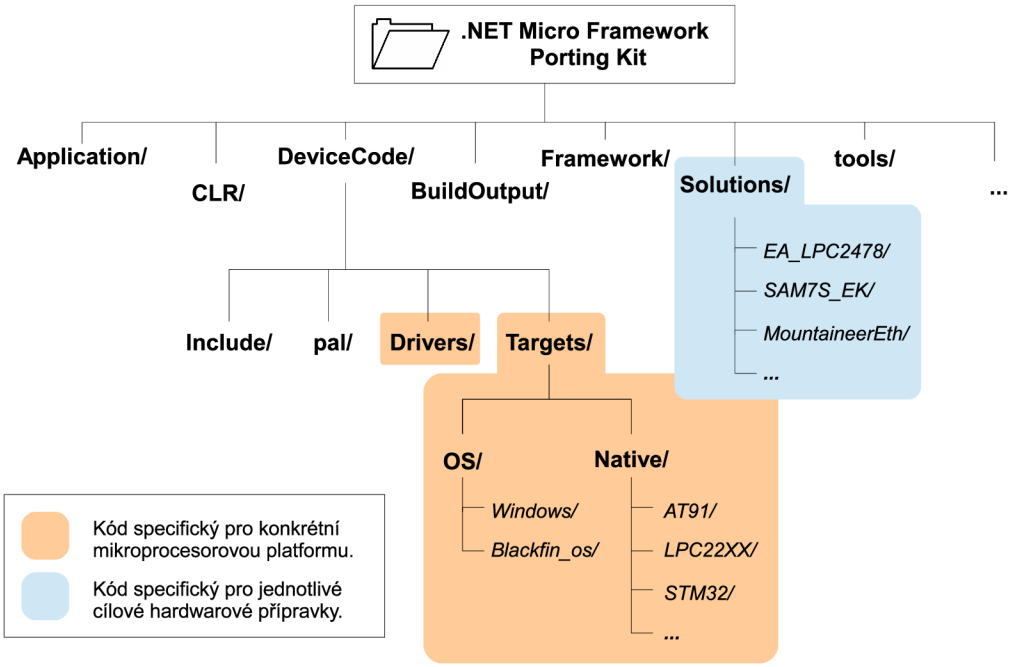
V případě, že požadujeme možnost krokovat a ladit nativní kód frameworku, lze použít kombinaci nástrojů GDB, OpenOCD a grafického vývojového prostředí Eclipse. Integrace těchto nástrojů je netriviální operací a autor práce doporučuje nahlédnout např. do návodu v článku [4].

5.2.2 Organizace zdrojového kódu frameworku

Aktuální zdrojový kód .NET MF lze získat přímo z repozitáře firmy Microsoft na serveru Codeplex [25] nebo lépe prostřednictvím instalačního balíčku *.NET Micro Framework Porting Kit* [31] (dále jen „.NET MF PK“). V případě požadavku na použití rozšíření třetích stran je nutné jejich zdrojový kód zaintegrovat nakopírováním do adresářové struktury .NET MF PK, jako je tomu v případě výše uvedeného zdrojového kódu od firmy Oberon.

Adresářová struktura zdrojového kódu .NET MF je znázorněna na obr. 5.1. Pokusíme-li se zdrojové soubory frameworku rozdělit z pohledu závislosti na cílových platformách, je možné vymezit tyto tři kategorie:

- **Kód, který implementuje funkcionalitu určenou pro konkrétní cílové hardwarové zařízení.** Tento celek se označuje termínem *solution*. Kód specifický pro kaž-



Obrázek 5.1: Adresářová struktura zdrojového kódu .NET MF

dé podporované cílové zařízení je umístěn v odděleném podadresáři adresáře `Solutions`. Typickými funkcemi, které je nutné implementovat pro každé *solution* zvlášť, jsou inicializační rutiny konkrétního zařízení nebo specifikace paměťového modelu konkrétního mikrokontroleru.

- **Kód specifický pro určitou výpočetní platformu.** Jedná se o rutiny, které jsou specifické pro jednu mikroprocesorovou platformu, tj. např. pro určité výpočetní jádro nebo pro konkrétní řadu mikrokontrolerů některého z výrobců. Do této skupiny patří veškeré ovladače vrstvy HAL a jejich implementaci lze najít v adresáři `DeviceCode\Targets`. Je zřejmé, že rutiny tohoto charakteru mohou být uplatněny v rámci více *solutions*, neboť .NET MF může být portován na odlišná zařízení, která přitom jsou vystavěna nad stejnou hardwarovou platformou. Aby se v těchto případech předešlo duplikaci kódu, v rámci překladového systému se používá systém sdílení kódu pomocí *inkluzí*, který bude vysvětlen v odstavci o překladu .NET MF. Referenční manuál k API rozhraní vrstvy HAL je součástí .NET MF PK[1].

- **Kód nezávislý na cílové platformě.** Všechny ostatní části zdrojového kódu .NET MF, které nespadají do předchozích dvou skupin, jsou implementovány v přenositelném kódu C++ a na cílové platformě nezávislé. Je tedy možné na ně nahlížet jako na „černou skříňku“ (*black box*) a při procesu portace se jejich vnitřní strukturou nezabývat. Významnými prvky této kategorie jsou běhové prostředí CLR se zdrojovými kódy ve stejnojmenné složce a součást TinyBooter, která bude vysvětlena níže a jejíž zdrojový text je umístěn ve složce `Application`.

[1] API dokumentace je obsažena v souboru `Framework\Documentation\RCLPort.chm`

5.2.3 Překladový systém .NET MF

Součásti zdrojového kódu .NET MF jsou propojeny pomocí sestavovacího systému MSBuild [29], který plní obdobnou funkci jako např. nástroj GNU Make a určuje tedy pořadí a způsob kompilace a sestavování jednotlivých částí frameworku. Systém MSBuild získáme při instalaci vývojového nástroje MS Visual Studio.

Každá logicky související skupina zdrojových souborů .NET MF je vybavena kontrolním souborem s názvem `*.proj` ve formátu XML, který pro nástroj MSBuild slouží jako zdroj metadat nutných pro stanovení způsobu překladu (tj. jako obdoba souborů *Makefile* systému GNU Make). Pomocí direktiv `<Include>` v souborech `*.proj` je možné odkazovat se na již existující kompilační jednotky a umožnit provázání stávajícího kódu s kódem již napsaným. Tohoto mechanismu se s výhodou používá např. pro připojení již existujících ovladačů vrstvy HAL do příslušného *solution* bez potřeby kopírovat a duplikovat zdrojové soubory.

Ústřední metadata pro kompilaci se nachází v souborech `*.proj` v adresáři s každého `solution`. Tyto soubory obsahují reference (`<Include>`) na veškeré součásti .NET MF, které mají být v daném *solution* obsažené. V případech, kdy konkrétní cílová platforma určitou funkcionalitu vrstvy HAL nepodporuje, je nutné ji v tomto souboru nahradit referencí na prázdnou implementaci, tzv. „pahýl“ (*stub*).

Informace pro systém MSBuild o tom, jakým způsobem má použít dostupné překladové nástroje, jsou v rámci .NET MF PK definované v souboru `Microsoft.Spot.system.-<compilerName>.targets` ve složce `tools\Targets`. Zde je možné určit, s jakými parametry budou kompilátor, sestavovací program a související nástroje systémem MSBuild volány.

Překlad zdrojového kódu frameworku probíhá v prostředí příkazové řádky systému Windows a skládá se ze dvou fází. Nejdříve je nutné nastavit proměnné prostředí, které obsahují název použitého překladače, cestu do adresáře s jeho instalací a verzi překladových nástrojů. Druhou operací je iniciace samotného překladového systému MSBuild. Tento sled úkonů je přehledně vysvětlen v dokumentu [18]. Výsledný binární obraz frameworku je po dokončení překladu k dispozici ve složce `BuildOutput`.

5.2.4 Nahrání .NET MF do cílového zařízení

Výsledkem překladu .NET MF je několik souborů ve formátech vhodných pro nahrání do mikrokontroleru. Tyto soubory společně tvoří binární obraz .NET MF.

První z nich, program Tinybooter, plní funkci bootloaderu – zajišťuje inicializaci hardware zařízení, umožňuje provádět upgrade .NET MF a poskytuje kryptografické schopnosti[2]. Program Tinybooter je obsažen v souborech `Tinybooter.(bin|hex)` a pro jeho nahrání do cílového přípravku je nutné využít programovací prostředky poskytnuté výrobcem hardwarového zařízení, například rozhraní JTAG.

Obraz běhového prostředí CLR je po překladu k dispozici v binárních souborech `ER_CONFIG` a `ER_FLASH`, které se do zařízení nahrají pomocí aplikace *MFDeploy*, která je součástí distribuce .NET MF PK. Aplikace *MFDeploy* komunikuje s firmware Tinybooter v cílovém zařízení pomocí rozhraní USB a umožní nahrání nebo aktualizaci binárního obrazu běhového prostředí CLR. Dále dovolí provádět diagnostické a kontrolní operace.

[2]Kryptografické nástroje aplikace Tinybooter se uplatní v případě komerčních zařízení, kde může výrobce pomocí metod asymetrické kryptografie zajistit autenticitu a integritu .NET MF v případě upgradu na novější verzi.

Po dokončení uvedené procedury je hardwarové zařízení připraveno ke komunikaci s aplikací Visual Studio a k nahrání aplikací přeložených do intermediárního kódu.

5.3 Úprava portu .NET MF pro podporu překladače GCC

Při tvorbě firmware pro zamýšlenou desku byl výchozím bodem otevřený zdrojový kód .NET MF verze 4.2 [25] od firmy Microsoft rozšířený firmou Oberon o podporu platformy STM32F4 [34]. Jako zdrojové *solution*, které bylo pro cílovou desku a překladač GCC upraveno, autor zvolil port .NET MF na desku STM32 F4Discovery, neboť ta je osazena kompatibilní variantou mikrokontroleru vzhledem k zapojení navrženému v rámci této práce.

Pro zajištění překladu frameworku pomocí kompilátoru GCC bylo nutné zaměřit se především na ty části kódu, které obsahují konstrukce, jež jsou napříč různými překladači neportabilní. Jedná se o:

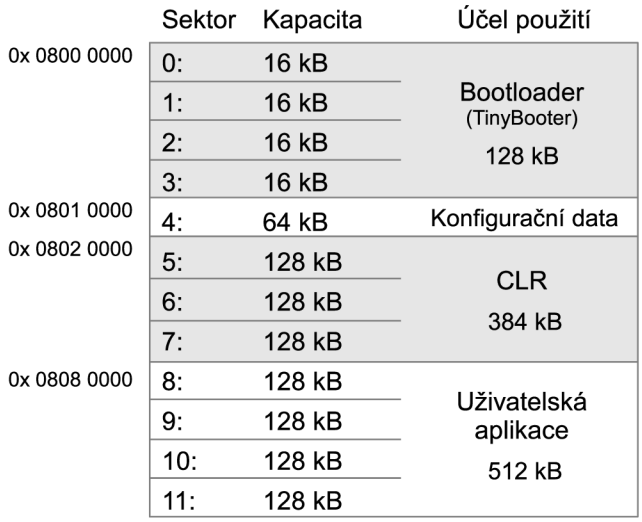
- **Kompilační jednotky napsané v jazyce symbolických instrukcí**; ty bylo nutné přepsat kód do dialektu, který je slučitelný s nástrojem *GNU Assembler*. Konkrétně bylo nutné vytvořit tyto nové soubory, které jsou součástí vrstvy HAL:
  - `CortexM3\TinyHal\GNU_S\FirsEntry.s`
  - `CortexM3\TinyHal\GNU_S\IDelayLoop.s`
  - `CortexM3\TinyHal\GNU_S\VectorHandlers.s`

  Modul `FirstEntry` představuje vstupní bod binárního obrazu aplikace TinyBooter a obsahuje základní inicializační rutiny. V modulu `DelayLoop` najdeme implementaci jednoduché zpožďovací smyčky. V kódu modulu `VectorHandlers` se vykonávání programu octne v případě, že nastala závažná chyba, z níž není možné se zotavit.

- **Moduly v jazyce C++ s vloženými instrukcemi assembleru**, neboli zdrojové texty obsahující vložené (*inline*) instrukce assembleru v neportabilním formátu. Zde bylo nutné použít makra preprocesoru a podmíněný překlad k zajištění kompatibility vůči překladači GCC. Tímto způsobem musel být upraven soubor `CortexM3\GlobalLock\SmartPtr_cortex.cpp`, který obsahuje implementaci datové struktury pro uchování ukazatele, která je bezpečná (thread-safe) vzhledem k případné asynchronně vyvolané obsluze přerušení.

- **Definice paměťového modelu mikrokontroleru**. Pro vymezení paměťového modelu cílové platformy je nutné vytvořit soubory *scatterfile* v adresáři s příslušným `solution`, které řídí proces sestavování binární podoby frameworku, tj. *linking*. Jedná se o dvojici souborů `scatterfile_bootloader_gcc.xml` a `scatterfile_tinyclr_gcc.xml`. Dále je nezbytné určit rozdělení nevolatilní paměti Flash MCU na sektory, které budou vyhrazeny pro bootloader, úložiště konfiguračních informací, běhové prostředí CLR a pro uživatelskou aplikaci. Toto nastavení se provede v souboru `BlockStorage\addDevices\Bl_addDevices.cpp` v adresáři *solution* určeného pro navrženou platformu. Konkrétní zvolené rozdělení paměti flash respektuje velikosti binárního obrazu frameworku a je znázorněno na obr. 5.2.

  Aby bylo možné používat v rámci .NET MF periferii FSMC mikrokontroleru a hodinový výstup MCO1, jak to bylo navrženo v oddílech 4.2.2 a 4.2.3, bylo třeba provést inicializaci periferií a příslušných pinů MCU. Jelikož se jedná o funkcionalitu určenou



Obrázek 5.2: Rozdělení paměti flash MCU STM32F4 z pohledu .NET MF

pro jediný specifický vývojový přípravek, bylo žádoucí příslušný kód umístit přímo do složky daného *solution*, konkrétně do souboru `IO_Init.cpp`.

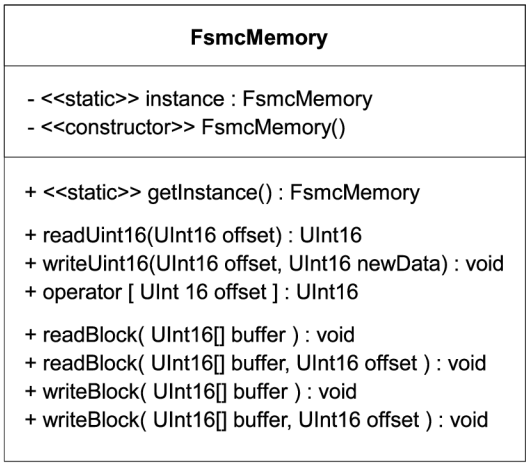
5.4 Rozšíření .NET MF o komunikaci s periferií FSMC

Potřebujeme-li rozšířit .NET MF o podporu nové periferie, může nastat několik případů. Je-li, tato periferie již zohledněna ve vrstvě PAL .NET MF, pak je její oživení jednoduché. Pro softwarovou obsluhu takové periferie stačí pouze implementovat příslušný ovladač ve vrstvě HAL v nativním kódu C++. Příkladem zařízení z této kategorie může být komunikační linka SPI, analogově-digitální převodník nebo znakový LCD displej.

Je-li konkrétní periferie vrstvě PAL neznámá, pak je možné implementaci její obsluhy zajistit jedním z těchto způsobů:

- **Implementovat obsluhu periferie v interpretovaném (*managed*) kódu.** Tato varianta je možná pouze tehdy, pokud je daná periferie ovladatelná prostřednictvím některého nízkoúrovňového rozhraní, které je již ve vrstvě PAL podporované (USART, GPIO, SPI a podobné). Příkladem takové periferie může být akcelerometr komunikující skrze rozhraní SPI. V tom případě stačí vytvořit třídu v jazyce C#, která uvedené rozhraní zapouzdří a tím nad ním vybuduje abstraktní vrstvu pro aplikačního programátora.

- **Implementovat ovladač periferie v nativním kódu.** Jedná se o alternativní způsob, kdy naprogramujeme rutiny v nativním kódu C++, a umožníme jejich přímé volání z prostředí interpretovaného (*managed*) kódu. Tento přístup se označuje termínem *interop*. Výhodou tohoto přístupu je rychlost výsledného kódu a vysoká flexibilita, tj. ničím nelimitovaný přístup k hardware cílové platformy. Uvedený postup není zdaleka omezen jen na ovladače periferií – stejným způsobem je možné akcelerovat kritické části interpretovaného kódu tím, že je přepíšeme do kódu nativního. Nevýhodou je nutnost opětovného překladu celého .NET MF.



(a) Diagram třídy `FsmcMemory`

```
FsmcMemory fpgaMem
      = FsmcMemory.getInstance();

const UInt16 BTN_REGISTER = 0x0001;
const UInt16 VIDEO_MEM_ADDR = 0x0020;
UInt16[] video_buf = new UInt16[1024];

// ...

// Stav tlačítka připojeného k FPGA?
if (fpgaMem[BTN_REGISTER] != 0) {

  // Tlačítko stisknuto.
  // Překopírovat blok videodat
  // do FPGA, kde budou zobrazena
  // např. pomocí videovýstupu VGA.
  fpgaMem.writeBlock( video_buf,
                      VIDEO_MEM_ADDR);
}

// ...
```

(b) Příklad použití třídy `FsmcMemory` v kódu interpretované aplikace

Obrázek 5.3: Rozhraní pro přístup k periferii FSMC z uživatelské aplikace

Komunikace po sběrnici FSMC, která poslouží pro výměnu dat mezi čipy MCU a FPGA, probíhá takto. Adresový prostor externí paměti připojené k periferii FSMC je „mapován“ do fyzického paměťového prostoru MCU [41, sekce 32.4] a čtecí nebo zápisová transakce se z pohledu MCU realizuje jako pouhé čtení nebo zápis do vymezené paměťové oblasti. Tyto operace je možné provést výhradně v nativním kódu C++, jelikož interpretovaný kód je od paměťového prostoru MCU a fyzické adresace na konkrétní platformě zcela odstíněn. Z toho důvodu je pro rozšíření .NET MF o práci s periferií FSMC nutné zvolit metodu *interop*.

5.4.1 Aplikační rozhraní pro přístup k FSMC

Obvod FPGA je ke řadiči paměťové sběrnice FSMC mikrokontroleru připojen pomocí 16 třístavových signálů sloužících v časovém multiplexu pro přenos adresy i dat o šířce 16 bitů (viz schéma v příloze A). Z pohledu uživatelské aplikace se tedy čip FPGA chová jako statická paměť tvořena celkem 2^{16} datovými položkami, z nichž každá má velikost 16 bitů. Operace čtení z paměti znamená přenos dat z FPGA do MCU, operace zápisu slouží k přenosu opačným směrem. Aby byl umožněn aplikačnímu programátorovi přístup k FSMC, byla v rámci práce navržena jednoduchá třída `FsmcMemory` v jazyce C#, jejíž diagram a příklad použití je na obr. 5.3. Pomocí ní je možné provádět tyto operace:

- **Přístup k jednotlivým datovým položkám externí paměti.** Operace čtení nebo zápisu jediného 16-bitového slova na zadané paměťové adrese (offsetu) zabezpečují metody
  - `UInt16 readUint16(UInt16 offset)` a
  - `void writeUint16(UInt16 offset, UInt16 newData)`.

Z důvodu lepší přehlednosti a výstižnosti aplikačního kódu používajícího třídu `FsmcMemory` byl také přetížen operátor indexace pole „`[]`“, který je s uvedenými metodami ekvivalentní.

- **Přístup na úrovni datových bloků.** Při kopírování souvislých datových bloků mezi MCU a čipem FPGA je nevýhodné tuto operaci provádět opakovaným voláním podprogramů předchozí kategorie, neboť je zde značná režie při přechodu mezi interpretovaným a nativním kódem, která omezí propustnost přenosu. Tuto situaci řeší následující metody, které implementují přenos celého bloku datových položek mezi čipem FPGA a MCU v nativním kódu a tedy s minimální režií:
  - `void readBlock(UInt16[] buffer)`,
  - `void readBlock(UInt16[] buffer, UInt16 offset)`,
  - `void writeBlock(UInt16[] buffer)`,
  - `void writeBlock(UInt16[] buffer, UInt16 offset)`.

5.4.2 Propojení nativního a interpretovaného kódu

Třídu `FsmcMemory` navrženou v předchozím odstavci je nutné propojit s nativní implementací metod pro přímý přístup k paměťovému prostoru periferie FSMC pomocí metody *interop*. Tato procedura byla provedena následovně. V prvním kroku byl vytvořen nový projekt v softwarovém nástroji MS Visual Studio (popř. ve Visual Studio C# Express) a doplněn aplikační kód v jazyce C#, který odpovídá struktuře navržené v předchozím odstavci. Podprogramy, jejichž implementace měla být provedena v nativním kódu, byly označeny klíčovým slovem `extern` a speciálním atributem. Při překladu projektu vygenerovalo Visual Studio pro takto označené metody „kostru“ kódu v jazyce C++, do níž byla ručně doplněna jejich implementaci v nativním kódu.

Soubory s intermediárním i nativním kódem byly zaintegrovány do adresářové struktury zdrojového kódu .NET MF, reference na ně vloženy do souborů `*.proj` překladového systému MSBuild a proveden nový překlad .NET MF z takto rozšířených zdrojových kódů. Posledním krokem bylo nahrání nového binárního obrazu frameworku do cílového hardwarového přípravku.

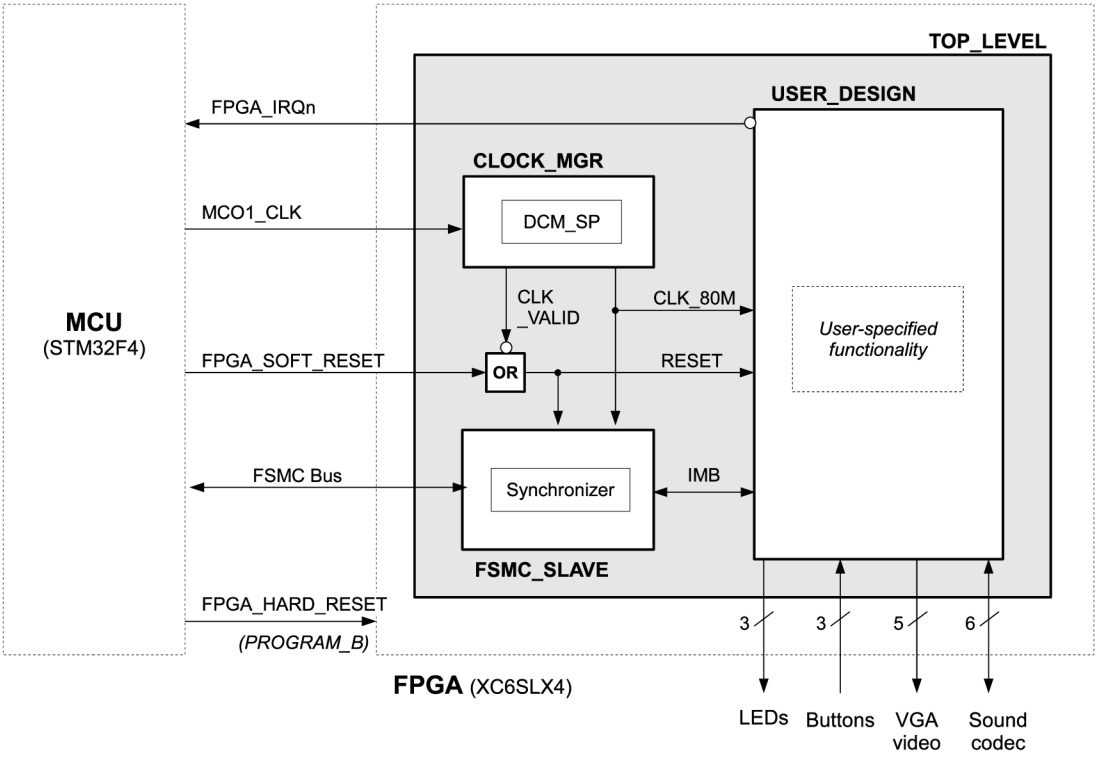
Pro detailní popis sekvence operací nastíněných v předchozím odstavci je čtenář odkázán do článku [21], z něhož bylo v rámci práce čerpáno. Výsledný kód třídy `FsmcMemory`, který se skládá z nativních i interpretovaných rutin, je možné nalézt ve zdrojovém kódu frameworku v elektronických přílohách práce ve složce: `Solutions\FPGAkit\DeviceCode\Interops`.

Kapitola 6

Firmware pro obvod FPGA

Jelikož výrobek představený v rámci práce slouží k vývoji a prototypování, je nutné uživateli této platformy vhodným způsobem usnadnit tvorbu nových aplikací. Kapitola 6 se zabývá návrhem firmware obvodu FPGA pro dokumentovanou hardwarovou platformu. Ten uživateli poslouží jako startovní bod pro vývoj vlastních systémů, tedy jako podpůrná „kostra“ uživatelských projektů, kterou bude možné snadno rozšiřovat a doplňovat o novou funkcionalitu.

Významným podproblémem, který je nutné v rámci firmwaru obvodu FPGA řešit, je komunikace po paměťové sběrnice FSMC mikrokontroleru. Je třeba zajistit, aby obvod FPGA vystupoval vůči MCU jak statická paměť a vhodným způsobem reagoval na čtecí a zápisové transakce iniciované MCU. Poddrobnosti obsahuje oddíl 6.2.



Obrázek 6.1: Hierarchická struktura firmware pro obvod FPGA

6.1 Struktura firmware

Firmware obvodu FPGA se skládá z hierarchicky uspořádané sady komponent, jak je znázorněno na obr. 6.1. Význam a úloha jednotlivých komponent bude představena v následujících odstavcích.

6.1.1 Komponenta `top_level`

Komponenta `top_level` je nejvyšším prvkem hierarchické struktury systému, který „zastřešuje“ všechny ostatní komponenty. Jeho cílem je poskytnout rozhraní mezi fyzickými vstupně výstupními bloky obvodu FPGA a signály uvnitř firmwaru FPGA. Jednotka `top_level` je doplněna sadou omezení (*constraints*) definovaných v souboru typu UCF (*User Constraints File* [50]), která určují zejména propojení vnitřních signálů systému s vývody čipu FPGA a frekvenci použitých hodinových signálů.

6.1.2 Komponenta `clock_mgr`

Jedná se o komponentu, která zajišťuje generování hlavního interního hodinového signálu čipu FPGA s kmitočtem 80 MHz. Jako vstupní hodinový zdroj slouží signál `MCO1_CLK`, jehož zdrojem je pin `MCO1` mikrokontroleru, jak to bylo navrženo v rámci obvodového zapojení v oddíle 4.2.2. Transformaci hodinového signálu provádí jednotka DCM_SP [49], což je jednotka integrovaná na čipu FPGA výrobcem ve formě jádra (*hard IP core*) a jejím úkolem je zejména frekvenční syntéza hodinového signálu a eliminace fázového zpoždění hodin (*clock skew*). Instanci jednotky DCM_SP je možné v rámci navrženého systému najít v souboru `dcm_elem.vhd`, který byl vygenerován nástrojem Xilinx Core Generator.

6.1.3 Komponenta `fsmc_slave`

Tato jednotka je spojovacím prvkem mezi externí paměťovou sběrnicí FSMC propojující čipy MCU a FPGA a jednoduchým synchronním rozhraním IMB (*Internal Memory Bus*), které se používá pro paměťové transakce v rámci uživatelské aplikace uvnitř čipu FPGA. Komponenta `fsmc_slave` bude podrobně popsána níže.

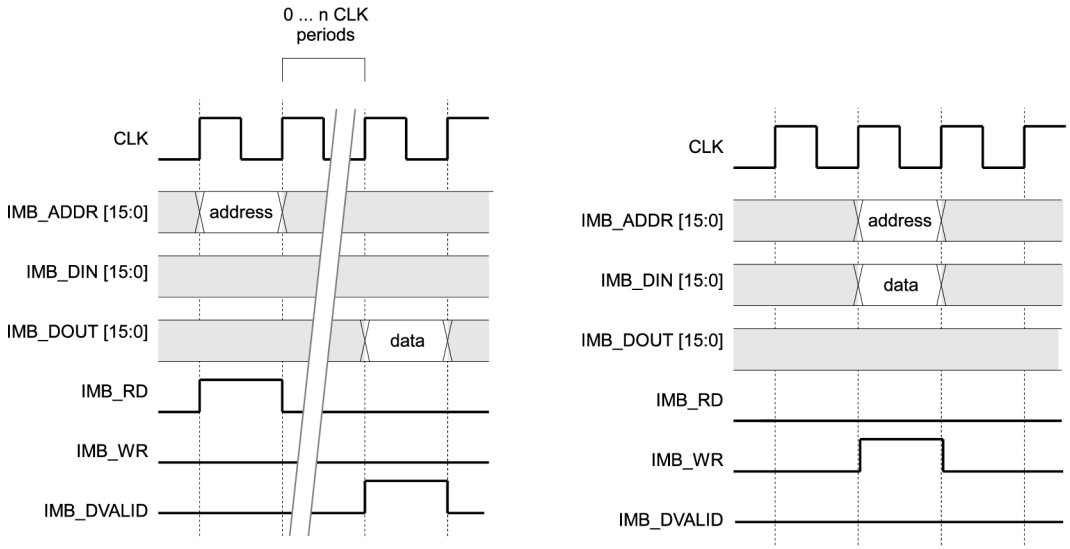
6.1.4 Komponenta `user_design`

Jednotka `user_design` je prostorem, do něhož může uživatel prototypovací platformy integrovat svůj vlastní vyvíjený systém. Oddělením systému dovnitř této komponenty je dosaženo abstrakce od specifických a výše popsaných vlastností konkrétní vývojové platformy, jako je syntéza hodinového kmitočtu nebo detailní fungování použité komunikační sběrnice.

Rozhraní komponenty `user_design` je tvořeno signály pro komunikaci po vnitřní paměťové sběrnici, hodinovým vstupem CLK a nulovacím vstupem RESET, signály pro obsluhu zvukového kodeku, videovýstupu VGA, pro řízení LED diod a čtení stavu tlačítek umístěných na vývojové desce. Nedílnou součástí jsou také signály připojené na rozšiřující konektory, pomocí nichž může být vývojová platforma propojena s externími systémy. Podrobnou definici rozhraní je možné nalézt spolu s komentáři přímo v záhlaví souboru `user_design.vhd`.

6.1.5 Generování signálu RESET

Reset obvodu FPGA je v rámci navrženého systému možné provést dvěma způsoby:



(a) Čtecí transakce

(b) Zápisová transakce

Obrázek 6.2: Komunikační protokol rozhraní IMB

- **„Soft“ reset**, který je iniciován uživatelskou aplikací v mikrokontroler pomocí signálu `FPGA_SOFT_RESET` nebo jednotkou správy hodin `clock_mgr` v případě, došlo-li k přerušení vnějšího hodinového zdroje. Při „soft“ resetu dojde k inicializaci komponent `user_design` a `fsmc_slave`. Konfigurace obvodu FPGA zůstává nezměněna a po uvolnění resetovacího signálu obvod FPGA pokračuje ve své původní funkci.
- **„Hard“ reset**, který může být vyvolán mikrokontrolerem nebo stiskem tlačítka „Reset“ na vývojové desce. Projeví se nízkou logickou úrovní na vývodu `PROGRAM_B` obvodu FPGA, tím se vynutí zastavení činnosti čipu a následně proběhne rekonfigurace obvodu FPGA opětovným načtením konfiguračního řetězce z paměti flash.

6.2 Komunikační jednotka pro rozhraní FSMC

Motivací pro návrh komunikační komponenty `fsmc_slave` je převod externího komunikačního protokolu FSMC na jednoduché interní paměťové rozhraní použitelné uvnitř čipu FPGA v rámci uživatelské aplikace.

6.2.1 Protokol vnitřního rozhraní IMB

Pro připojení komponenty `user_design` ke komunikační jednotce `fsmc_slave` uvnitř čipu FPGA autor práce zvolil jednoduché vlastní rozhraní, pracovně označené jako IMB (*Internal Memory Bus*[1]). Tato linka pracuje na principu protokolu synchronizovaného náběžnou

[1] Označení „sběrnice“ (*bus*) není zcela technicky přesné, neboť se nejedná o komunikační linku třístavového charakteru, kde je vysílání po společných vodičích dovoleno více komunikujícím subsystémům.

Signál	Směr	Význam signálu
DA[15:0]	obousměrné, třístavové	Přenos adresy a dat v časovém multiplexu
A[24:16]	nepropojeno	Nejvyšších 8 bitů adresy (implicitně nulové)
NE	MCU → FPGA	Aktivace sběrnice FSMC
NADV	MCU → FPGA	Indikace platnosti adresy
NOE	MCU → FPGA	Indikace čtecí transakce
NWE	MCU → FPGA	Indikace zápisové transakce
NBL0, NBL1	MCU → FPGA	Indikace částečného zápisu dat (na úrovni 8-bitových jednotek)
NWAIT	FPGA → MCU	Požadavek na prodloužení čtecí transakce

Tabulka 6.1: Signály externí sběrnice FSMC mezi čipy MCU a FPGA

hranou hodinového signálu CLK, který byl inspirován analogickým komunikačním mechanismem uplatněným u vývojových karet Combo-PTM [22]. Časové diagramy čtecí i zápisové transakce vystihuje obr. 6.2.

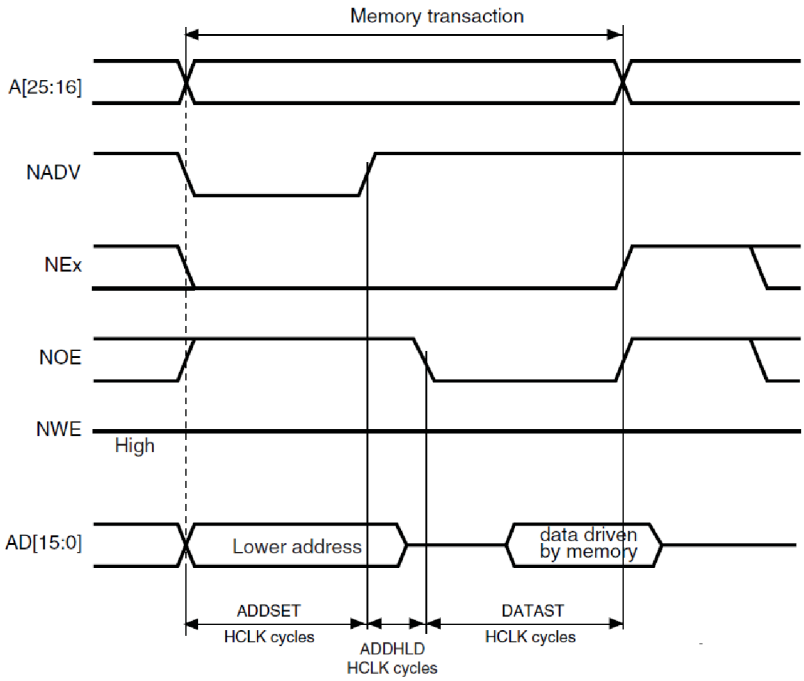
Je-li komponenta `fsmc_slave` požádána mikrokontrolerem o dodání dat nacházejících se na určité adrese (čtecí transakce), pak `fsmc_slave` vystaví příslušnou 16-bitovou adresu na vodiče `IMB_ADDR` a čeká na dodání dat ze strany uživatelského systému. Jakmile jsou požadovaná data připravena, vloží je uživatelská komponenta `user_design` na rozhraní `IMB_DOUT` a zároveň indikuje platnost dat signálem `IMB_DVALID`. Čtecí transakce na rozhraní IMB trvá nejméně 2 hodinové takty a zmíněný signál `IMB_DVALID` představuje mechanismus, který v případě potřeby dovolí prodloužit čtecí transakci podle požadavků konkrétního uživatelem implementovaného systému. Takto navržený protokol čtecích transakcí vyžaduje, aby na každý požadavek o čtení dat bylo vždy odpovězeno signálem `IMB_DVALID` a tím čtecí transakce ukončena. Počet hodinových cyklů, které se čekáním na data stráví, je tedy v režii uživatelské aplikace (komponenty `user_design`).

Mechanismus zápisové transakce na rozhraní IMB je jednodušší – komponenta `fsmc_slave` po přijetí dat provede jejich vystavení na port `IMB_DIN` a indikuje zápisovou transakci signálem `IMB_WR`. Zápisová transakce na sběrnici IMB je tedy vždy provedena v rámci jediné periody hodinového signálu `CLK`. Způsob zpracování takto přijatých dat je plně na zodpovědnosti uživatelské aplikace `user_design` (ta může data podle potřeby uložit do registru, do paměti BRAM nebo zápisovou transakci i zcela ignorovat, je-li např. adresa mimo platný rozsah).

6.2.2 Elektrické zapojení a komunikační protokol sběrnice FSMC

Řadič externí statické paměti FSMC integrovaný ve zvoleném MCU STM32F4 je do značné míry konfigurovatelný a umožňuje propojení s různými druhy paměti. Pro podrobnou specifikaci řadiče lze odkázat do kapitoly 30 referenčního manuálu rodiny MCU STM32F4 [41].

Z možných režimů práce periferie FSMC byl pro realizaci komunikačního rozhraní jako vhodný zvolen mód *Asynchronous Multiplexed NOR Flash Memory*. Rozhraní je tvořeno 16 vodiči pro přenos adresy i dat v časovém multiplexu a dále sedmi řídicími signály aktivními v log. 0. Z důvodu redukce počtu spojů na DPS bylo nutné ignorovat vodiče `FSMC_A[24:16]` odpovídající 8 nejvyšším bitům adresy a ponechat je nezapojené. Při všech paměťových transakcích jsou proto tyto bity považovány za nulové a čip FPGA vystupuje jako paměť



Obrázek 6.3: Časový diagram čtecí transakce na sběrnici FSMC [41]

Parametr	Význam parametru	Hodnota (počet cyklů HCLK)
ADDSET	Délka fáze *address set*	8
ADDHLD	Délka fáze *address hold*	3
DATAST	Délka fáze *data strobe*	10
BUSTRN	Minimální interval mezi dvěma po sobě následujícími transakcemi (*bus turnaround*)	15

Tabulka 6.2: Konfigurace délky pro jednotlivé fáze paměťových transakcí

s 16-bitovým adresovým rozsahem. Všechny signály, které tvoří rozhraní mezi obvodem FPGA a MCU, jsou obsaženy v tabulce 6.1.

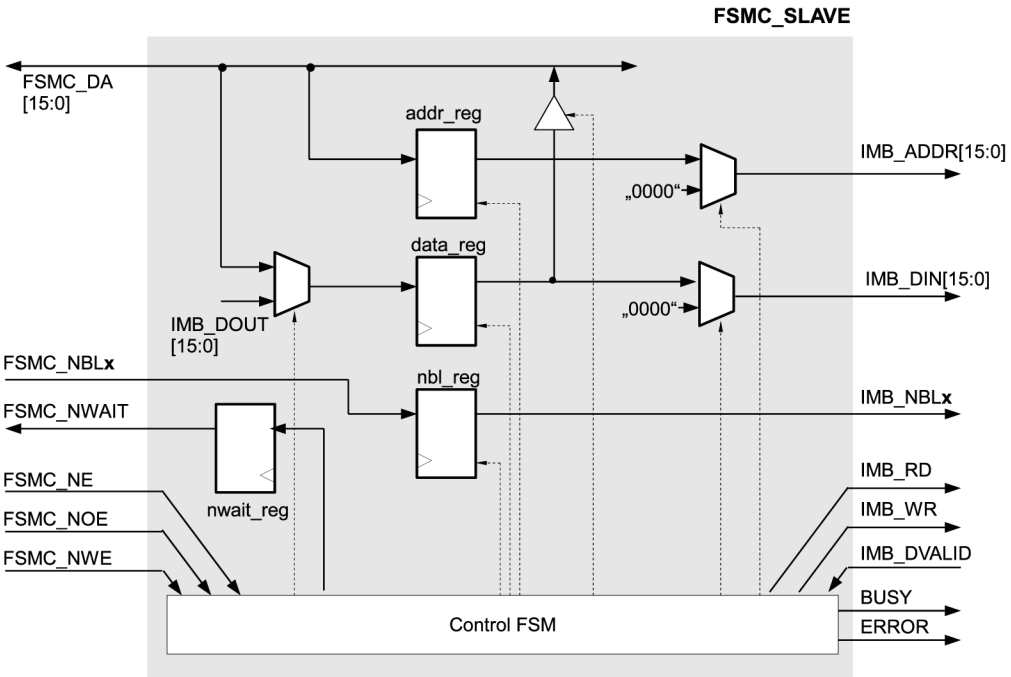
Každá transakce na sběrnici FSMC při zvoleném protokolu *NOR Flash* sestává ze tří fází – nastavení adresy (*address set*), „pozdržení“ adresy (*address hold*) a vzorkování dat (*data strobe*). Indikace jednotlivých fází se děje prostřednictvím kontrolních signálů sběrnice FSMC a jejich primárním úkolem je vymezit okamžiky, kdy je možné vzorkovat platnou adresu nebo platná data vystavená na sběrnici. Délky jednotlivých fází jsou konfigurovatelné v násobcích period hodinového signálu HCLK, což je taktovací frekvence jádra mikrokontroleru (v našem případě 168 MHz). Konkrétní nastavení použité v rámci práce je shrnuto v tabulce 6.2. Je zřejmé, že při tomto nastavení je základní délka transakcí 36 taktů hodin HCLK a teoretická propustnost sběrnice je 4,67 milionu transakcí za sekundu.

Jak se tyto fáze včetně sledu kontrolních signálů projeví v rámci čtecí transakce, je znázorněno na obr. 6.3. Začátek každé paměťové transakce je indikován přechodem signálu `NE` do úrovně log. 0. Hodnota na vodiči `NADV` vymezuje okamžik, kdy adresa vystavená na

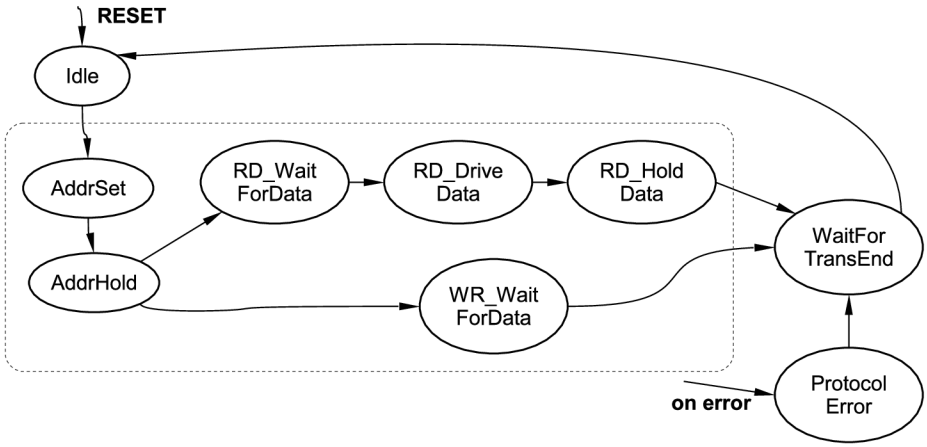
sběrnici `DA[15:0]` je platná. Očekává se, že připojená paměť provede vzorkování adresy v okamžiku přechodu signálu `NADV` z log. úrovně 0 do úrovně 1, tedy na konci fáze *address set*. Signál *NOE* přechodem do aktivního stavu v log. 0 indikuje, že aktuální transakce je čtecí; proběhne tedy přenos dat směrem z čipu FPGA do mikrokontroleru. Fázi čekání na dodání dat z paměti (*data strobe*) je možné prodloužit, pokud během ní čip FPGA aktivuje signál `NWAIT`. Čtecí transakce končí po deaktivaci signálu `NWAIT`, nejdříve však po uplynutí nastaveného intervalu DATAST. Periferie FSMC mikrokontroleru provede vzorkování dat z paměti v okamžiku ukončení transakce, když řídicí signály `NE` a `NOE` přechází do úrovně log. 1.

Zápisová transakce, probíhá obdobně jako transakce čtecí. Řídícím signálem pro indikaci zápisu je vodič `NWE`, který přechází do aktivní úrovně log. 0 na začátku fáze `address hold`. Od připojené paměti (tj. v našem případě od čipu FPGA) se očekává, že vzorkování adresy proběhne při přechodu `NADV` log. 1 a vzorkování dat při náběžné hraně signálu `NWE`. Celá zápisová transakce končí, stejně jako transakce čtecí, návratem signálu `NE` do neaktivní úrovně (log. 1).

Řídicí signály sběrnice FSMC označené jako `NBL0` a `NBL1` (*byte lanes*) slouží v případě zápisové transakce k omezení šířky zapisovaných dat na 8 bitů. Jeli např. aktivován pouze signál `NBL1`, mikrokontroler tím indikuje, že během transakce má být zapsáno 8 vyšších bitů dat (tj. data vystavená na sběrnici `DA[15:8]` a nižších 8 bitů příslušné datové položky má zůstat nezměněno. Třída `FsmcMemory`, o niž byl .NET MF rozšířen v oddíle 5.4, však vždy komunikuje na úrovni celých 16-bitových datových položek, proto v případě jejího použití je možné signály NBL0 a NBL1 v uživatelské aplikaci v FPGA plně ignorovat. Tímto opatřením se dosáhne plné propustnosti sběrnice a se zjednoduší a zpřehlední aplikační kód. Vodiče `NBL0` a `NBL1` však v systému zůstaly zachovány a jednotka `fsmc_slave` je pro jejich použití připravena, pokud by to bylo žádoucí vzhledem k potřebám specifické uživatelské aplikace.



Obrázek 6.4: Struktura jednotky `fsmc_slave`



Obrázek 6.5: Zjednodušený stavový diagram řídicího automatu jednotky `fsmc_slave`

6.2.3 Konstrukce komunikační jednotky `fsmc_slave`

Komponenta `fsmc_slave` slouží v rámci firmware FPGA jako spojovací prvek mezi externí asynchronní paměťovou sběrnicí a synchronním rozhraním IMB uvnitř čipu FPGA. Musí tedy být schopna komunikovat oběma protokoly popsanými v předchozích odstavcích a vhodným způsobem provádět konverze paměťových transakcí mezi nimi.

Strukturu této jednotky vystihuje obr. 6.4. Jádrem jednotky je Mealyho řídící stavový automat, který koordinuje činnost všech prvků komponenty. Dalšími základními stavebními bloky jsou adresový a datový registr, které jsou nezbytné pro uchování adresy a dat v rámci paměťové transakce, jelikož v případě rozhraní FSMC jde o multiplexovaný protokol a informace o adrese a datech je tedy na sběrnici vystavena v různé časové okamžiky a nikoli souběžně.

Jelikož protokol linky FSMC je asynchronní, probíhá jeho zpracování jednotkou `fsmc_slave` na principu převzorkování (*supersampling*) vstupních signálů. Vzorkovací kmitočet je dán hlavním hodinovým signálem CLK obvodu FPGA o frekvenci 80 MHz.

Vzhledem k faktu, že změna úrovní vstupních signálů rozhraní FSMC probíhá asynchronně vůči vzorkovacímu kmitočtu, je na ně potřeba nahlížet jako na signály vedené napříč různými hodinovými doménami obvodu. Aby v takovém případě byla potlačena možnost vzniku metastabilních stavů během vzorkování, je každý ze vstupních signálů veden přes kaskádu dvou klopných obvodů typu D, které fungují jako *synchronizér* a poskytují čas na vyřešení případných metastabilních jevů. Tato funkcionalita je zapouzdřena do jednotky `fsmc_synchronizer`. Problematika metastability je hlouběji diskutována např. v dokumentu [5].

Řídicí automat komponenty `fsmc_slave` je ve zjednodušené podobě zachycen na obr. 6.5 a v plné podobě potom v příloze C. Stavy automatu respektují fáze komunikačního protokolu po sběrnici FSMC, které byly rozebrány výše.

Výchozím stavem je stav nečinnosti `Idle`. V případě zahájení paměťové transakce ze strany mikrokontroleru aktivací signálů `NE` a `NADV` automat přechází do stavu `AddrSet`, v němž proběhne vzorkování adresy do registru `addr_reg`. Po zneplatnění signálu `NADV` pak následuje přechod do stavu `AddrHold`. V tomto okamžiku čeká jednotka `fsmc_slave` na informaci o typu konkrétní paměťové transakce.

Je-li mikrokontrolerem aktivován signál `NOE`, pak jde o čtecí transakci. Automat přejde do stavu `RD_WaitForData` a požádá uživatelskou aplikaci o dodání čtených dat prostřed-

nictvím vnitřního paměťového rozhraní IMB. Po dobu čekání na uživatelksá data je signál `NWAIT` držen v aktivní úrovni (log. 0), čímž se transakce prodlouží do doby, než jsou data připravena. Výstupní signál NWAIT je opatřen klopným obvodem typu D, aby se předešlo zákmitům (*glitches*), které by mohly nastat v okamžicích stavových přechodů automatu.

Okamžik kdy jsou data k dispozici, je indikován signálem `IMB_DVALID` na vnitřním rozhraní IMB propojujícím jednotku `fsmc_slave` a uživatelský design. Automat přejde do stavu `RD_DriveData`, vloží získaná data na třístavovou sběrnici `FSMC_DA` a čeká na jejich přijetí mikrokontrolerem ve stavu `RD_HoldData`. Jakmile mikrokontroler provede vzorkování dat a zneaktivní signál `NOE`, automat přejde do stavu `WaitForTransEnd`, kdy již čeká pouze na uzavření paměťové transakce indikované náběžnou hranou signálu `NE`.

Pokud ve stavu `AddrHold` dojde dojde k aktivaci signálu `NWE` namísto signálu `NOE`, jedná se o transakci zápisovou, jejíž mechanismus je jednodušší. Automat reaguje stavem `WR_WaitForData` a v něm provede vzorkování dat dodaných mikrokontrolerem. Při zneplatnění signálu `NWE` automat přechází do stavu `WaitForTransEnd`. V momentě tohoto stavového přechodu vloží získanou adresu a data pro zápis na vnitřní paměťové rozhraní IMB a aktivuje řídicí signál `IMB_WR` pro indikaci požadavku na zápis.

Komponenta `fsmc_slave` obsahuje také mechanismus pro detekci a zotavení se z chyb komunikačního protokolu po externí sběrnici FSMC. Je jím stav `ProtocolError`, který je aktivován v případě neplatného sledu signálů (např. současná aktivace `NOE` a `NWE` nebo zneplatnění signálu `NE` uprostřed rozpracované transakce). Došlo-li během transakce k chybě, která musela být ošetřena stavem `ProtocolError`, je toto indikováno uživatelské aplikaci pulzem na výstupním portu `PROTOCOL_ERROR` jednotky `fsmc_slave`.

6.3 Problémy řešené ve fázi návrhu firmware

Významným problémem, který byl při vývoji firmware obvodu FPGA řešen, byla chybná konfigurace jednotky `DCM_SP` pro syntézu hodinového kmitočtu, která způsobovala nepravidelné náhodné pulzy (*glitches*) na výstupu `CLK_VALID` indikujícího platnost hodinového signálu. Toto vedlo k restartům ostatních komponent firmware, zejména jednotky `fsmc_slave` v nepředvídatelné okamžiky a narušovalo to integritu přibližně 1 % všech paměťových transakcí. Uvedený problém bylo vzhledem k jeho podstatě obtížné odhalit a pro jeho lokalizaci byl uplatněn softwarový logický analyzér *Xilinx ChipScope Pro*.

Zdroj čipu FPGA	Celkem dostupno	Využito	Spotřebovaný podíl
Occupied Slices	600	39	6 %
Slice LUTs	2400	61	2 %
Slice Registers	4800	83	1 %
MUXCYs	1200	24	2 %
BUFG/BUFGMUXs	16	3	18 %
DCM units	4	1	25 %
Bonded IOBs	102	96	94 %

Tabulka 6.3: Zdroje čipu FPGA spotřebované navrženým firmware

6.4 Simulace a syntéza systému

Firmware obvodu FPGA dokumentovaný v předchozích oddílech byl vytvořen v jazyce VHDL, simulován pomocí nástroje Xilinx ISE ISim a syntetizován nástrojem Xilinx ISE Webpack ve verzi 14.3. Množství zabraných zdrojů čipu Xilinx Spartan-6 XC6SLX4 obsahuje tabulka 6.3. Z pohledu množství prostředků alokovaných navrženým firmware je podstatné množství tzv. *slices*, které jsou základními konfigurovatelnými elementy čipu FPGA. Uživateli vývojové platformy proto pro jeho vlastní aplikace zbývá přibližně 94 % kapacity obvodu FPGA.

Kapitola 7

Testovací a demonstrační aplikace

Kapitola 7 představuje čtyři jednoduché demonstrační aplikace, které dokumentují způsob práce s navrženou platformou. Každá aplikace se skládá ze dvou částí – z kódu pro mikrokontroler a hardwarového designu pro čip FPGA popsaného v jazyce VHDL. Programy pro mikrokontroler jsou s výjimkou první aplikace napsány v interpretovaném kódu pro běh v rámci .NET MF a využívají třídu `FsmcMemory`, která byla představena v oddíle 5.4. Hardwarové designy pro obvod FPGA jsou vybudovány nad firmware, jehož struktuře se věnovala kapitola 6.

První dvě demonstrační aplikace slouží k verifikaci činnosti sběrnice FSMC a k vyhodnocení jejich parametrů. Druhé dvě aplikace implementují každá jednoduchý systém, který demonstruje některé praktické rysy vývoje aplikací pro .NET MF i FPGA a může uživateli vývojové posloužit jako startovní bod pro vývoj vlastních systémů.

7.1 Aplikace 1: Propustnost sběrnice FSMC v nativní aplikaci

První aplikace, kterou lze na přiloženém CD nalézt v složce `fsmc_read_test`, slouží ke dvěma cílům – k jednoduchému ověření činnosti sběrnice FSMC při čtecích transakcích a k měření skutečné propustnosti paměťové sběrnice. Jelikož aplikace vznikla primárně za účelem lokalizace problému popsaného v oddíle 6.3, je požadována její maximální jednoduchost. Kód pro mikrokontroler byl proto vytvořen v jazyce C nad jednoduchým operačním systémem reálného času ChibiOS/RT [2].

Design v obvodu FPGA je tvořen systémem, který pouze interaguje s čtecími transakcemi na sběrnici FSMC a na požadavek o data odpoví datovou položkou, jejíž obsah je bitovou inverzí příslušné adresy, z níž je čteno. Mikrokontroler posléze provede porovnání načtené datové položky s očekávanou hodnotou a v případě neshody aktivuje signál `FPGA_SOFT_RESET`. Ten je možné zachytit funkcí *trigger* nástroje *Xilinx ChipScope Pro* a ze získaných časových diagramů analyzovat, proč konkrétní paměťová transakce neproběhla podle očekávání.

Odkomentujeme-li v souboru `main.c` volání funkce `measureSpeed()`, proběhne měření propustnosti sběrnice v nativním kódu. Program provádí čtecí transakce po dobu jedné sekundy a na závěr je vyhodnocen jejich počet. Stejný postup je aplikován na transakce zápisové. Celé měření čtecích i zápisových transakcí se opakuje celkem desetkrát. Z počtu provedených transakcí získáme propustnost sběrnice v jednotkách *Mbit/s* vynásobením konstantou 16, neboť datová šířka sběrnice je právě 16 bitů. Hodnoty získané průměrem deseti

Typ transakcí	Počet transakcí/s	Propustnost sběrnice
Čtecí transakce	3325298,2	50,74 (Mbit/s)
Zápisová transakce	3893117,1	59,40 (Mbit/s)

Tabulka 7.1: Propustnost sběrnice FSMC při přístupu z aplikace v C (průměr z 10 měření)

měření obsahuje tabulka 7.1.

Naměřené hodnoty odpovídají očekávání. Jedna paměťová transakce trvá, nedojde-li k jejímu prodloužení signálem `NWAIT`, 36 hodinových taktů signálu `HCLK` MCU, který je nastaven na frekvenci 168 MHz. To odpovídá teoretické hodnotě 4,67 milionů provedených transakcí za sekundu. Nižší naměřené hodnoty jsou způsobeny režií běhu programu v mikrokontroleru (řídící instrukce pro realizaci smyček). Dále nižší propustnost při čtecích transakcích oproti zápisovým je dána faktem, že data jsou dostupná na sběrnici IMB při čtecí transakci až následující hodinový cyklus po vystavení požadavku na čtení, jak to bylo vysvětleno v oddíle 6.2.1.

7.2 Aplikace 2: Práce se sběrnicí FSMC v interpretované aplikaci

Cílem aplikace 2 je zaměřit se na práci se sběrnicí FSMC z interpretovaného kódu v rámci .NET MF, ověřit činnost implementované třídy `FsmcMemory` a změřit, jaké reálné propustnosti je možné dosáhnout při komunikaci z kódu interpretované aplikace. Implementaci demonstrační aplikace 2 najdeme ve složce `fsmc_bram_test`.

Syntetizovaný obvod v čipu FPGA se skládá ze sady jader BRAM, které dohromady tvoří jedinou logickou paměť s 8192 datovými položkami o šířce 16 bitů. Tato paměť je připojena přímo k rozhraní IMB. Obvod FPGA jako celek tedy v tomto případě vystupuje jako skutečná statická paměť a data do něho zapsaná je možné ve stejné podobě později přečíst.

Aplikace pro mikrokontoroler je napsána v interpretovaném kódu a plní dvě funkce – verifikace správnosti komunikace a měření propustnosti. Odkomentujeme-li volání `performIntegrityTest()`, proběhne testování korektnosti paměťové komunikace takto. Aplikace nejdříve vygeneruje pole o velikosti 8192 položek s náhodnými hodnotami, které poté nakopíruje pomocí blokového přístupu (`FsmcMemory.writeBlock()`) do paměti BRAM na čipu FPGA. Po dokončení analogickým způsobem data zpět přečte a porovná je se zaslanými daty. V případě, že by došlo k chybě komunikace a přečtená data by se neshodovala se zapsanými, bude na to programátor upozorněn voláním `Debug.Assert(false)`, které zajistí pozastavení vykonávání kódu a umožní inspekci momentálního stavu pozastaveného programu v prostředí aplikace Visual Stuido.

Voláním metody `performSpeedTest()` lze změřit propustnost sběrnice při přístupu z řízeného kódu. Je provedeno měření čtyř typů transakcí – čtení a zápisu, u obojího jak režim práce po jednotlivých položkách, tak přístup po blocích. U blokového přístupu, který byl představen v oddíle 5.4.1, lze očekávat významné urychlení aplikace, neboť celá komunikace proběhne v rámci nativního kódu a neuplatní se ani režie smyček v interpretovaném kódu, ani režie přechodu mezi kódem řízeným a nativním.

Měření proběhlo takto. U každého ze zkoumaných typů transakcí byl proveden přesun 81920 datových položek mezi MCU a FPGA, přičemž u blokového přístupu se data rozdělila

na 10 bloků. Byl měřen čas nutný pro dokončení celého datového transferu a tento údaj přepočten na počet transakcí, které byly provedeny za jednu sekundu. Toto měření bylo opakováno desetkrát a hodnoty zprůměrovány. Výsledky v tabulce 7.2 ukazují, že režie interpretovaného kódu při přístupu k paměti „po položkách“ je značná (je nezbytné provádět mnoho iterací smyčky v interpretovaném kódu) a uplatnění blokového režimu zvýší datovou propustnost o tři řády.

Typ transakcí	Počet transakcí/s	Propustnost
Čtecí transakce („po položkách“)	2842,0	0,0455 Mbit/s
Čtecí transakce (blokový režim)	3114355,2	49,38 Mbit/s
Zápisové transakce („po položkách“)	3041,3	0,0487 Mbit/s
Zápisové transakce (blokový režim)	3588261,1	57,41 Mbit/s

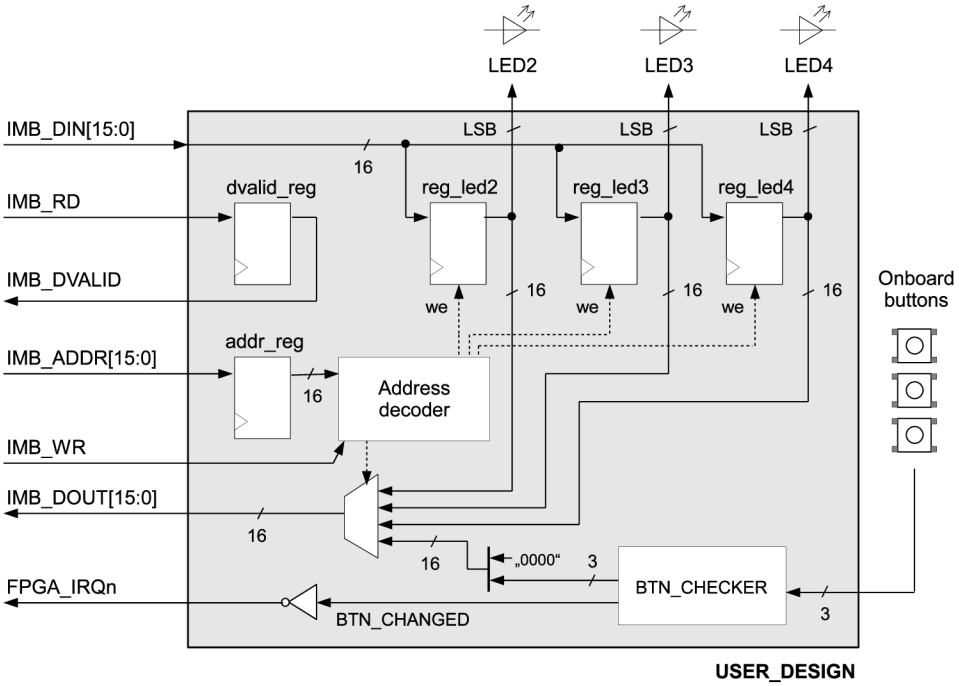
Tabulka 7.2: Propustnost sběrnice FSMC při přístupu z interpretované aplikace

7.3 Aplikace 3: Řízení periferií FPGA z aplikace v MCU

Třetí aplikace (ve složce `btns_leds`) na praktickém příkladě ukazuje, jakým způsobem je možné ovládat periferie čipu FPGA z interpretovaného aplikačního kódu v mikrokontroleru. Pro jednoduchost jsou jako vstupní periferie použity tlačítka B1 až B3 na desce a jako výstupní periferie trojice LED diod připojených k obvodu FPGA.

V této konkrétní aplikaci vystupuje čip FPGA vůči MCU jako entita obsahující čtyři 16-bitové registry sloužící pro řízení periferií a kontrolu jejich stavu. Trojice registrů přístupných na adresách `0x0000`, `0x0001` a `0x0002`, umožňuje pomocí svého nejnižšího významového bitu (LSB) ovládat stav připojených LED diod. Registr na adrese `0x0010` obsahuje ve třech svých nejnižších bitech aktuální stav tlačítek B1, B2 a B3 na desce. Systém je řízen adresovým dekodérem, což je sada komparátorů generujících signály pro výstupní multiplexor signálu `IMB_DOUT` a popřípadě signály „write enable“ pro zápis do registrů. Použití tří registrů pro ovládání tří diod je v praxi nadbytečné; zde je však zvoleno z demonstračních účelů, aby bylo možné naznačit způsob tvorby paměťového dekodéru po sadu více registrů. Schéma popsaného systému na obr. 7.1 ukazuje, jak je možné systém připojit na komunikační rozrhaní IMB. Odpovídající kód v jazyce VHDL je možné najít v souboru `btns_leds/ise_project/user_design.vhd`. Ten může posloužit uživateli hardwarového přípravku jako základ pro tvorbu vlastních aplikací.

Jelikož je v systému řešena práce s mechanickými spínacími kontakty (tlačítky), je nutné potlačit nežádoucí oscilace vstupních signálů, které vzniknou v okamžicích sepnutí nebo rozepnutí kontaktu. Za tímto účelem byla navržena komponenta `btn_checker`, která problém řeší. Pracuje na principu vzorkování stavu tlačítka v periodických intervalech a nový stav tlačítka propaguje na svůj výstup až v okamžiku, kdy všech n posledních vzorků má shodnou hodnotu a mechanický kontakt se tedy ustálil. Počet provedených vzorků a časovou prodlevu mezi nimi lze nastavit pomocí generických parametrů (`generic map`). V případě změny stavu některého ze sledovaných tlačítek je toto indikováno pulzem na výstupu `BTN_CHANGED` komponenty. Ten se v popisované aplikaci přenese vně čipu FPGA na výstupní vodič `FPGA_IRQn`, kde může být mikrokontrolerem zaznamenán a obsloužen



Obrázek 7.1: Struktura obvodu v čipu FPGA pro řízení periferií z MCU (aplikace `btns_leds`)

pomocí přerušení. Komponenta `btn_checker` byla navržena jako univerzální a znovuvyužitelná jednotka, kterou může uživatel vývojové platformy zařadit i do svých systémů. Problematiku zákmitů mechanických kontaktů a jejich potlačení (tzv. *debouncing*) včetně vhodných algoritmů popisuje např. dokument [13], z něhož autor při řešení tohoto podproblému čerpal.

Interpretovaný kód v rámci této demonstrační aplikace funguje následovně. Reaguje na žádosti o přerušení obdržené od obvodu FPGA a při přerušení zkontroluje aktuální stav tlačítek přečtením registru v FPGA po paměťové sběrnici FSMC. Bylo-li některé tlačítko stisknuto, pak provede změnu stavu příslušné LED diody připadající k danému tlačítku zápisem do konkrétního registru v FPGA. V rámci této jednoduché aplikace jsou tedy demonstrovány dva komunikační prvky – jednak obousměrná komunikace datová komunikace iniciovaná čipem MCU, jednak asynchronní žádost o přerušení vyvolaná obvodem FPGA. Část kódu, která ukazuje princip obsluhy přerušení v interpretované aplikaci, je na obr. 7.2.

7.4 Aplikace 4: Generování grafického výstupu VGA

Čtvrtá ukázková aplikace se zabývá generováním grafického výstupu pomocí rozhraní VGA řízeného čipem FPGA, kdy vlastní data k zobrazení jsou vytvořena v mikrokontroleru a dodána do čipu FPGA po paměťové sběrnici FSMC.

Design zapojení v obvodu FPGA je ve čtvrté demonstrační aplikaci tvořen strukturou zachycenou na obr. 7.3. Jako úložiště pro videodata určená k zobrazení se použije soustava jader BRAM, která jsou nakonfigurována a propojena tak, že dohromady vystupují jako jediná statická paměť se dvěma nezávislými porty A a B[1].

[1]Vygenerování a konfigurace soustavy jader BRAM byla provedena nástrojem *Xilinx Core Generator*.

```
private static void handleBtnChange( /* ... */ )
{
    FsmcMemory fpgaMem = FsmcMemory.getInstance();

    // read current button changes
    UInt16 newBtns = fpgaMem[16];

    // decide which button has changed its state
    UInt16 changedBtns = (UInt16)(newBtns ^ oldBtns);
    UInt16 pressedBtns = (UInt16)((~newBtns) & changedBtns);

    oldBtns = newBtns;

    // toggle FPGA LEDs that belong to the pressed buttons
    if ((pressedBtns & 0x0001) != 0) fpgaMem[0] = (UInt16)(1 - fpgaMem[0]);
    if ((pressedBtns & 0x0002) != 0) fpgaMem[1] = (UInt16)(1 - fpgaMem[1]);
    if ((pressedBtns & 0x0004) != 0) fpgaMem[2] = (UInt16)(1 - fpgaMem[2]);

}

// Application entry point.
public static void Main()
{
    InterruptPort fpgaInt = new InterruptPort(
            PIN_FPGA_IRQn,
            false,          // false = no glitch filtering
            Port.ResistorMode.Disabled,
            Port.InterruptMode.InterruptEdgeLow);
    // ...
}
```

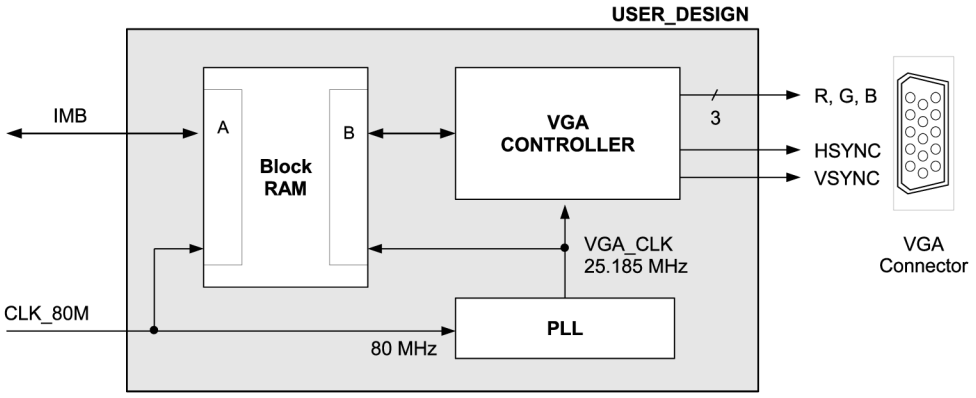
Obrázek 7.2: Příklad obsluhy přerušení v kódu interpretované aplikace

Port A slouží pro připojení paměti k rozhraní IMB a touto cestou jsou nahrávána nová data ke zobrazení ze strany mikrokontroleru. Port B propojuje paměť s řadičem rozhraní VGA, který takto získává konkrétní hodnoty pixelů, jenž mají být momentálně odvysílány po tomto analogovém videorozhraní. S výhodou se zde využívá faktu, že porty A a B sdílí uložená data, ale pracují zcela nezávisle, tj. mohou být řízeny rozdílnými hodinovými kmitočty a být nastaveny pro práci s různou datovou šířkou.

Port A pracuje s šířkou datové položky 16 bitů, což odpovídá šířce komunikační linky mezi mikrokontrolerem a čipem FPGA a umožňuje dosáhnout co nejvyšší propustnosti. Port B je nastaven pro datovou šířku pouhého 1 bitu, tj. jediného monochromaticky definovaného pixelu, který má být v konkrétní okamžik odvysílán k zobrazení. Implementovaný a níže popsaný řadič rozhraní VGA pracuje s rozlišením 640x480 pixelů a s barevnou hloubkou 8 barev. Z důvodu omezené kapacity pamětí BRAM uvnitř obvodu FPGA je však v rámci této demonstrační aplikace omezeno rozlišení na 320x240 pixelů o barevné hloubce 1 bit, je tedy generován monochromatický binární černobílý obraz.

7.4.1 Princip fungování řadiče rozhraní VGA

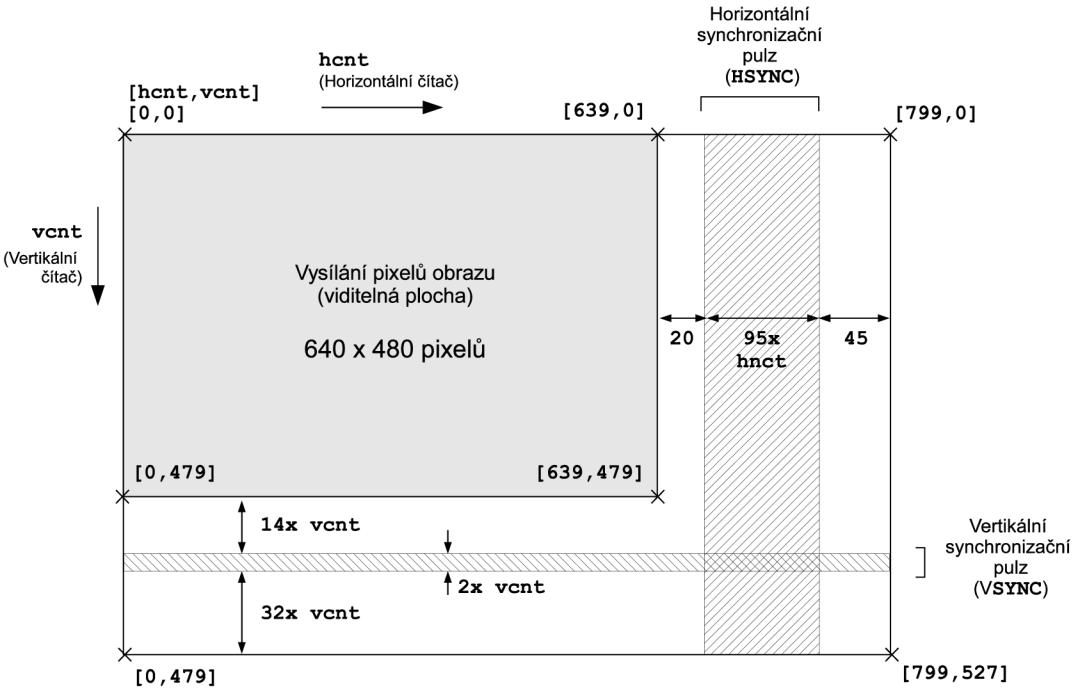
Grafický port VGA je rozhraní tvořené pěticí signálů – signály R, G a B, jejichž analogové napětí udává intenzitu barevných složek právě vysílaného pixelu, a dvěma synchronizačními signály HSYNC a VSYNC pro řádkovou a snímkovou synchronizaci obrazu. Uvedené signály jsou vzájemně časovány takto: Pixely obrazu jsou na rozhraní VGA vysílány po řádcích počínaje levým horním pixelem obrazu. Po odvysílání každého řádku je aktivován signál HSYNC (horizontální synchronizace), který indikuje konec aktuálního řádku a



Obrázek 7.3: Generování videosignálu VGA v obvodu FPGA

přechod na řádek následující. Stejným způsobem jsou odvysílány ostatní řádky obrazu. Jakmile dojde zaslání posledního řádku obrazovky, aktivuje se signál VSYNC indikující dokončení vykreslení jednoho snímku obrazovky a celá procedura se periodicky opakuje. Tento protokol rozhraní VGA vychází z principů řízení vykreslování obrazu na analogových monitorech typu CRT (*Cathode Ray Tube*). Při časování uvedených signálů je nutné přesně dodržet specifikované intervaly pro dobu vysílání pixelů i synchronizačních signálů, jak je to popsáno např. v [16].

Úkolem komponenty `vga_controller` je zajistit generování signálů rozhraní VGA a dodržení protokolu uvedeného výše. Navržená jednotka pracuje s fyzickým rozlišením 640x480 pixelů a generuje 60 snímků za sekundu. Základní hodinovou frekvencí rozhraní je v případě implementované jednotky kmitočet 25,185 MHz (signál `VGA_CLK`), který definuje interval vy-



Obrázek 7.4: Generování videosignálu VGA v jednotce `vga_controller`

sílání dvou po sobě následujících pixelů. Hlavními prvky definujícími stav komponenty VGA řadiče jsou horizontální a vertikální čítač (signály `hcnt` a `vcnt`). Horizontální čítač je buzen přímo signálem `VGA_CLK` a jeho perioda je 800 hodinových pulzů. Perioda vertikálního čítače 528 jednotek a k inkrementaci dojde vždy při přetečení čítače vertikálního. Prostor čítačů znázorňuje obr. 7.4. V něm je naznačeno, jak probíhá vykreslování obrazu řízené těmito čítači a kdy jsou vysílány příslušné synchronizační pulzy.

Jednotka `vga_controller` během vysílání obrazového signálu komunikuje s externími subsystémy a získává data k zobrazení pomocí výstupů `MEM_ROW_ADDR` a `MEM_COL_ADDR` a vstupu `MEM_PIXEL`. Očekává se, že uživatel připojí tyto signály na rozhraní synchronní videopaměti, jako je tomu v ukázkové aplikaci. Toto však není podmínkou a v případě požadavku na vykreslování jednoduchých vzorů je paměť možné nahradit prostou kombinační logikou s registry na výstupech.

Komponenta `vga_controller` je obecným a znovuvyužitelným stavebním prvkem, který může uživatel hardwarového přípravku zaintegrovat i do svých aplikací.

7.4.2 Tvorba grafického výstupu v aplikaci pro MCU

Aplikace pro mikrokontroler je napsána v interpretovaném kódu a běží pomocí služeb .NET MF. Využívá služeb třídy `FsmcMemory` pro přístup k videopaměti v čipu FPGA a pro blokový přenos videodat.

Pole videodat typu `UInt16[]` je v aplikačním kódu zapouzdřeno uvnitř třídy `VideoBuffer_320x240`, který uživateli nabízí přístup k videopaměti na úrovni pixelů. Jelikož jedno šestnáctibitové slovo videopaměti obsahuje informaci o 16 po sobě jdoucích pixelech, je nutné operace čtení a zápisu pixelů uvnitř třídy `VideoBuffer_320x240` převést na adekvátní bitové operace.

Schopnosti třídy pro přístup k videopaměti jsou rozšířeny třídou `DrawingPlane` o možnost vykreslovat grafická primitiva – čáry (využil se návrhový vzor *Decorator*). Hlavní funkcionalitu aplikace zajišťuje třída `Clock`, která pomocí všech výše popsaných mechanismů provede vykreslení analogových hodin s animovanými ručičkami do videopaměti v mikrokontroleru. Data z videopaměti jsou posléze přenesena do čipu FPGA pomocí blokového zápisu, což je úkolem třídy `FsmcMemory`.

Kapitola 8

Závěr

Náplní diplomové práce bylo v souladu s aktuálními trendy v oblasti vestavěných systémů navrhnout novou platformu, která poslouží k vývoji a prototypování vestavěných zařízení. Klíčovými vlastnostmi představeného zařízení je přítomnost mikrokontroleru typu ARM Cortex-M, programovatelného hradlového pole FPGA a podpora open-source .NET Micro Frameworku.

Autor v rámci práce prozkoumal situaci na trhu s obdobnými vývojovými platformami, na základě ní stanovil specifikaci pro zamýšlený hardwarový přípravek a zvolil komponenty vhodné pro jeho realizaci. Práce dále pokračovala navržením vlastního obvodového zapojení, které bylo fyzicky realizované formou desky plošných spojů.

Pro takto vzniklé zařízení byla zajištěna podpora v rámci .NET Micro Frameworku a sestaven doplňující firmware, jehož úkolem je usnadnit uživateli vývojové platformy tvorbu vlastních systémů. Činnost zařízení byla ověřena pomocí sady demonstračních aplikací, které zároveň posloužily jako praktický návod pro tvorbu aplikací na této nově představené platformě. V rámci ověření činnosti výrobku autor také změřil skutečnou propustnost komunikační linky, která propojuje čip FPGA a mikrokontroler obsažené v navržené platformě a tedy tvoří hlavní komunikační kanál mezi oběma integrovanými výpočetními systémy.

Práce poskytuje řadu námětů pro budoucí rozšiřování schopností vývojové platformy. Jedním z nich je oživení některých integrovaných periferií a doplnění jejich úplné podpory do .NET Micro Frameworku. Těmito komponentami jsou rozhraní sítě Ethernet, čtečka paměťových karet MicroSD a audio kodek. Jinou myšlenkou je zprovoznit mechanismus Xilinx Virtual Cable, který poskytne alternativní způsob nahrávání konfigurace do obvodu FPGA, čímž odpadne nutnost pořizovat pro tento účel externí hardwarové adaptéry. Dalším námětem je vytvoření nových demonstračních aplikací a znovuvyužitelných komponent, které je může uživatel platformy zapojit do svých vlastních projektů.

Hlavní přínos práce spatřuje autor ve vytvoření prototypu inovativní a konkurenceschopné platformy určené pro vývoj vestavěných systémů, která nemá obdoby na současném trhu. Výrazným rysem výrobku je jeho vysoká užitná hodnota při zachování nízkých výrobních nákladů. Dílčím produktem práce, který má charakter samostatně využitelného celku, je úprava stávajících zdrojových kódu .NET Micro Frameworku do podoby, která umožní překlad frameworku pro platformu mikrokontrolerů řady STM32F4 pomocí volně dostupného překladače GCC. Tento krok zjednodušuje budoucí rozšiřování frameworku o novou funkcionalitu a usnadní portace na jiné platformy, které jsou založené na této řadě výkonných mikrokontrolerů.

Literatura

[1] Webová prezentace firmy GHI Electronics. [online], [2003]. [cit. 2012-12-10].
URL <http://www.ghielectronics.com/>

[2] ChibiOS/RT Homepage. [online], [cit. 2013-07-12].
URL <http://www.chibios.org/dokuwiki/doku.php>

[3] Mono - cross-platform, open source .NET development framework. [online], [cit. 2013-07-12].
URL <http://www.mono-project.com/Main_Page>

[4] Al-Hertani, H.: Setting up Debugging with the Eclipse IDE (OpenOCD 0.6.x). [online], September 25, 2012. [cit. 2013-07-13].
URL
<http://hertaville.com/2012/09/16/part-3-debugging-openocd-0-6-0/>

[5] Altera Corporation: White Paper – Understanding Metastability in FPGAs. [online], 2009. [cit. 2013-07-12].
URL <http:
//www.altera.com/literature/wp/wp-01082-quartus-ii-metastability.pdf>

[6] Analog Devices, Inc.: ADP5023 Data Sheet. Datový list součástky. [online], 2011. [cit. 2012-12-11].
URL
<http://www.analog.com/static/imported-files/data_sheets/ADP5023.pdf>

[7] ARM Ltd.: ARM Processors – Cortex-M Series. [online], [2013?]. [cit. 2013-07-05].
URL <http://www.arm.com/products/processors/cortex-m/index.php>

[8] ARM Ltd.: Trends, Risks and Opportunities. [online], [cit. 2013-07-12].
URL <http://ir.arm.com/phoenix.zhtml?c=197211&p=irol-trends>

[9] Arm Ltd.: GNU Tools for ARM Embedded Processors: Binární distribuce překladače GCC udržovaná firmou ARM. [online], [cit. 2013-07-13].
URL <https://launchpad.net/gcc-arm-embedded>

[10] ECMA International: C# Language Specification. Norma ECMA-334. 2006.
URL <http:
//www.ecma-international.org/publications/standards/Ecma-334.htm>

[11] ECMA International: Common Language Infrastructure (CLI) Partitions I to VI. Norma ECMA-335. 2006.
URL <http:
//www.ecma-international.org/publications/files/ECMA-ST/ECMA-335.pdf>

[12] Fakulta informačních technologií VUT v Brně: FITkit. Webové stránky projektu. [online], [cit. 2013-07-12].
URL <http://merlin.fit.vutbr.cz/FITkit/>

[13] Ganssle, J.: A Guide to Debouncing. [online], Rev3. June, 2008. [cit. 2013-07-12].
URL <http://www.eng.utah.edu/~cs5780/debouncing.pdf>

[14] Germain, J.: Microsoft's .NET Micro Framework is now free and open source. [online], [November 2009], [cit. 2012-12-10].
URL <http://betanews.com/2009/11/23/microsoft-s-net-micro-framework-is-now-free-and-open-source/>

[15] Horkel, M.: Xilinx Virtual Cable s USB obvodem FTDI FT220X. [online], 2013-05-01, [cit. 2013-07-12].
URL <http://www.mlab.cz/Modules/CPLD_FPGA/XILINX_XVC/XVC_FT220X02A/DOC/XVC_FT220X02A.cs.pdf>

[16] Hwang, E.: Build a VGA Monitor Controller. [online], November 2004. [cit. 2013-07-12].
URL <http://faculty.lasierra.edu/~ehwang/public/mypublications/VGA%20Monitor%20Controller.pdf>

[17] International Standards Organization: Information technology – Common Language Infrastructure (CLI) Partitions I to VI. Standard ISO/IEC 23271:2006. 2006.
URL <http://standards.iso.org/ittf/PubliclyAvailableStandards/c042927_ISO_IEC_23271_2006(E).zip>

[18] Isa, G.: *The Beginners Guide to Porting NETMF*. GHI Electronics, Rev. 2.0, January 2012.
URL <http://www.ghielectronics.com/downloads/FEZ/Beginners%20Guide%20to%20Porting%20NETMF.pdf>

[19] Isa, G.: *The Beginners Guide to C# and the .NET Micro Framework*. GHI Electronics, Rev. 2.1, September 2012.
URL <http://www.ghielectronics.com/downloads/FEZ/Beginners%20guide%20to%20NETMF.pdf>

[20] Kühner, J.: *Expert .NET Micro Framework*. New York: Apress Media LLC, druhé vydání, 2009.

[21] Maillet, S.: Using Interop in the .NET Micro Framework V3.0. [online], April 27, 2009. [cit. 2013-07-12].
URL <http://blogs.msdn.com/b/smaillet/archive/2009/04/27/using-interop-in-the-net-micro-framework-v3-0.aspx>

[22] Martínek, T.: Platforma COMBO-PTM – Handbook. Dokument je součástí studijních materiálů předmětu NAV v LS2013, [cit. 2013-07-12].

[23] Mentor, J.: REVIEW: 2004 Microsoft SPOT Watch Smartwatch. [online], [cit. 2013-07-12].
URL
<http://www.smartwatchnews.org/2004-microsoft-spot-watch-smartwatch/>

[24] Micrel Inc.: KSZ8021RNL / KSZ8031RNL. Datový list součástky. [online], 2010. [cit. 2013-07-12].
URL
<http://www.micrel.com/_PDF/Ethernet/datasheets/ksz8021rnl_8031rnl.pdf>

[25] Microsoft Corporation: Repozitář .NET Micro Frameworku na serveru CodePlex. [online], [2009?], [cit. 2012-12-10].
URL <http://netmf.codeplex.com/>

[26] Microsoft Corporation: .NET Micro Framework Homepage. [online], [2011?]. [cit. 2012-12-10].
URL <http://www.netmf.com/>

[27] Microsoft Corporation: .NET Platform. [online], [2011?]. [cit. 2012-12-10].
URL <http://www.microsoft.com/net>

[28] Microsoft Corporation: Microsoft Research – .NET Gadgeteer. [online], [2012?]. [cit. 2012-12-10].
URL <http://research.microsoft.com/en-us/projects/gadgeteer/>

[29] Microsoft Corporation: MSBuild Reference. [online], [2012?]. [cit. 2012-12-10].
URL <http://msdn.microsoft.com/en-us/library/0k6kkbsd.aspx>

[30] Microsoft Corporation: Understanding .NET Micro Framework Architecture. [online], [2012?]. [cit. 2012-12-10].
URL <http://msdn.microsoft.com/en-us/library/cc533001.aspx>

[31] Microsoft Corporation: .NET Micro Framework Porting Kit verze 4.2 RTM. Instalátor softwarového produktu pro operační systém Windows. [online], August 2012. [cit. 2012-12-10].
URL <http://netmf.codeplex.com/downloads/get/218285>

[32] Microsoft Corporation: .NET Micro Framework SDK verze 4.2. Instalátor softwarového produktu pro operační systém Windows. [online], August 2012. [cit. 2012-12-10].
URL <http://netmf.codeplex.com/downloads/get/266459>

[33] Microsoft Corporation: API Reference for .NET Micro Framework. [online], [cit. 2013-07-12].
URL <http://msdn.microsoft.com/en-us/library/hh401281.aspx>

[34] Oberon Microsystems: NETMF_for_STM32: Repozitář zdrojových kódu pro port .NET Micro Frameworku na platformu STM32F4. [online], [cit. 2012-12-11].
URL <http://netmf4stm32.codeplex.com/>

[35] Oberon microsystems AG: NETMF for STM32 - Tour d'Horizon. [online], Zürich, 2012. [cit. 2012-12-10].
URL <http://www.mountaineer.org/app/download/6083569375/NETMF+for+STM32+-+Tour+d%27Horizon.pdf?t=1339779332>

[36] Olimex Ltd.: Universal EXTension connector (UEXT). [online], Revision B, October 2012. [cit. 2013-07-12].

URL
<https://www.olimex.com/Products/Modules/UEXT/resources/UEXT_rev_B.pdf>

[37] Secret Labs LCC: Netduino, an open-source electronics platform using the .NET Micro Framework. [online], 2010. [cit. 2012-12-10].
URL <http://netduino.com/>

[38] Serra, M.: What is Hardware/Software Codesign. [online], [cit. 2013-07-12].
URL <http://webhome.cs.uvic.ca/~mserra/HScodesign.html>

[39] SILICA: Specifikace vývojové desky Xynergy-M4 Board. [online], [cit. 2013-07-12].
URL <http://www.silica.com/product/silica-xynergy-m4-board.html>

[40] ST Microelectronics: STM32F407ZG – High-performance and DSP with FPU, ARM Cortex-M4 MCU with 1 Mbyte Flash, 168 MHz CPU, Art Accelerator, Ethernet. Datový list součástky. [online], [cit. 2012-12-11].
URL <http://www.st.com/internet/mcu/product/252136.jsp>

[41] ST Microelectronics: RM0090 Reference manual for STM32F40xxx, STM32F41xxx, STM32F42xxx, STM32F43xxx advanced ARM-based 32-bit MCUs. Referenční manuál. [online], [cit. 2013-07-12].
URL <http://www.st.com/web/en/resource/technical/document/reference_manual/DM00031020.pdf>

[42] ST Microelectronics: TMPS2141, STMPS2151, STMPS2161, STMPS2171 – Enhanced single channel power switches. Datový list součástky. [online], [cit. 2013-07-12].
URL <http://www.st.com/web/en/resource/technical/document/datasheet/CD00167470.pdf>

[43] Tišnovský, P.: Co se děje v počítači – Mikroprocesory s architekturou ARM. [online], 2012-03-06. [cit. 2013-07-12].
URL <http://www.root.cz/clanky/mikroprocesory-s-architekturou-arm/>

[44] Turley, J.: The Two Percent Solution. [online], [cit. 2012-12-10].
URL <http://www.embedded.com/electronics-blogs/significant-bits/4024488/The-Two-Percent-Solution>

[45] Wolfson Microelectronics plc: WM8940 – Mono CODEC with Speaker Driver. [online], November 2011. [cit. 2013-07-13].
URL <http://www.wolfsonmicro.com/documents/uploads/data_sheets/en/WM8940.pdf>

[46] Xilinx Inc.: iMPACT SPI and BPI PROM Support. Dokumentace k softwarovému produktu. [online], [2012?]. [cit. 2013-07-12].
URL <http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_4/pim_r_supported_spi_bpi_proms.htm>

[47] Xilinx Inc.: Spartan-6 FPGA Configuration User Guide. [online], January 23, 2013. [cit. 2013-07-12].
URL
<http://www.xilinx.com/support/documentation/user_guides/ug380.pdf>

[48] Xilinx Inc.: Spartan-3 FPGA Starter Kit Board User Guide. [online], June 20, 2008. [cit. 2013-07-13].
URL
<http://www.xilinx.com/support/documentation/boards_and_kits/ug130.pdf>

[49] Xilinx Inc.: Spartan-6 FPGA Clocking Resources User Guide. [online], June 20, 2013. [cit. 2013-07-12].
URL
<http://www.xilinx.com/support/documentation/user_guides/ug382.pdf>

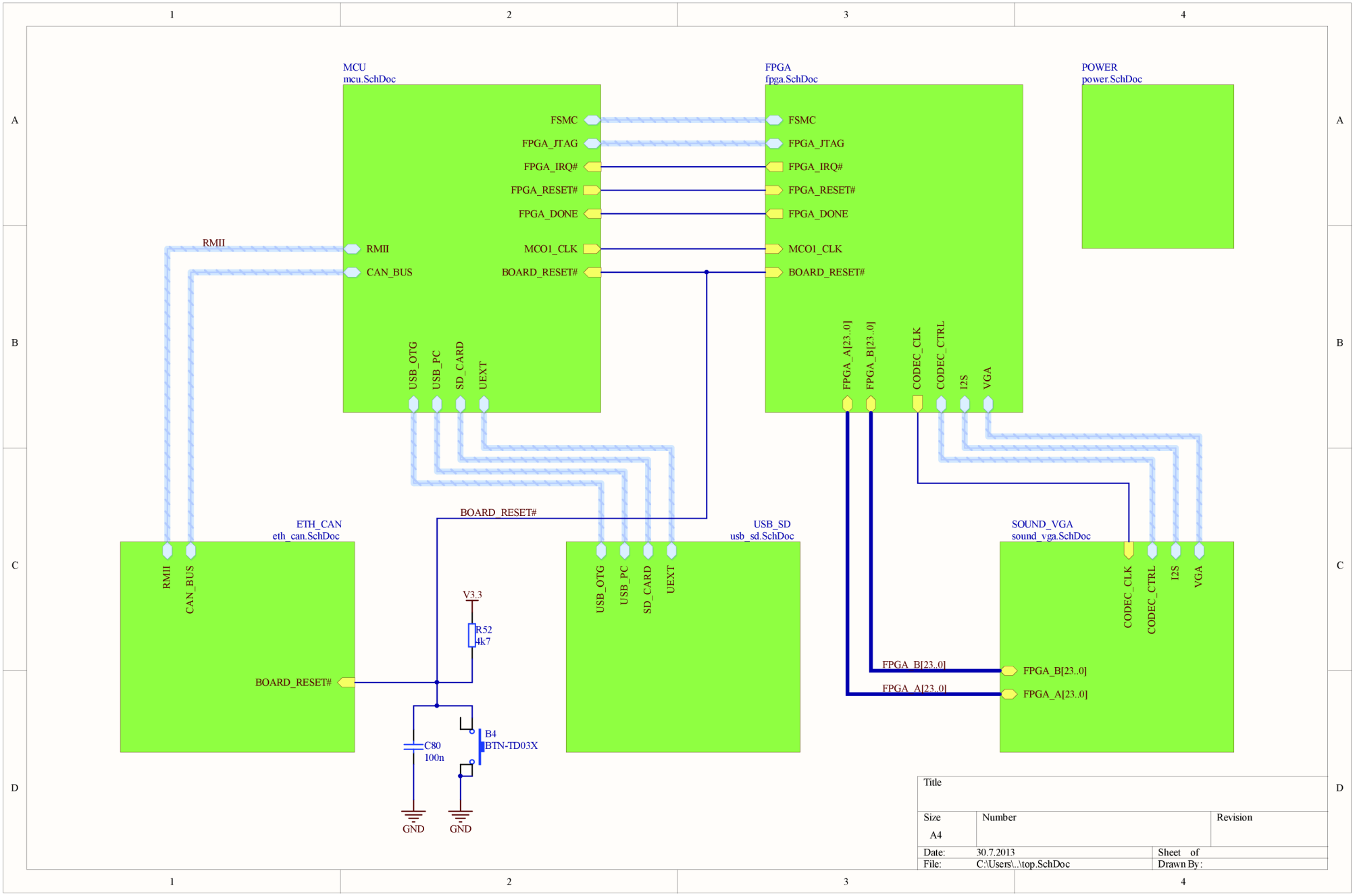
[50] Xilinx Inc.: Constraints Guide. Uživatelská příručka. [online], March 1, 2011. [cit. 2013-07-12].
URL <http:
//www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/cgd.pdf>

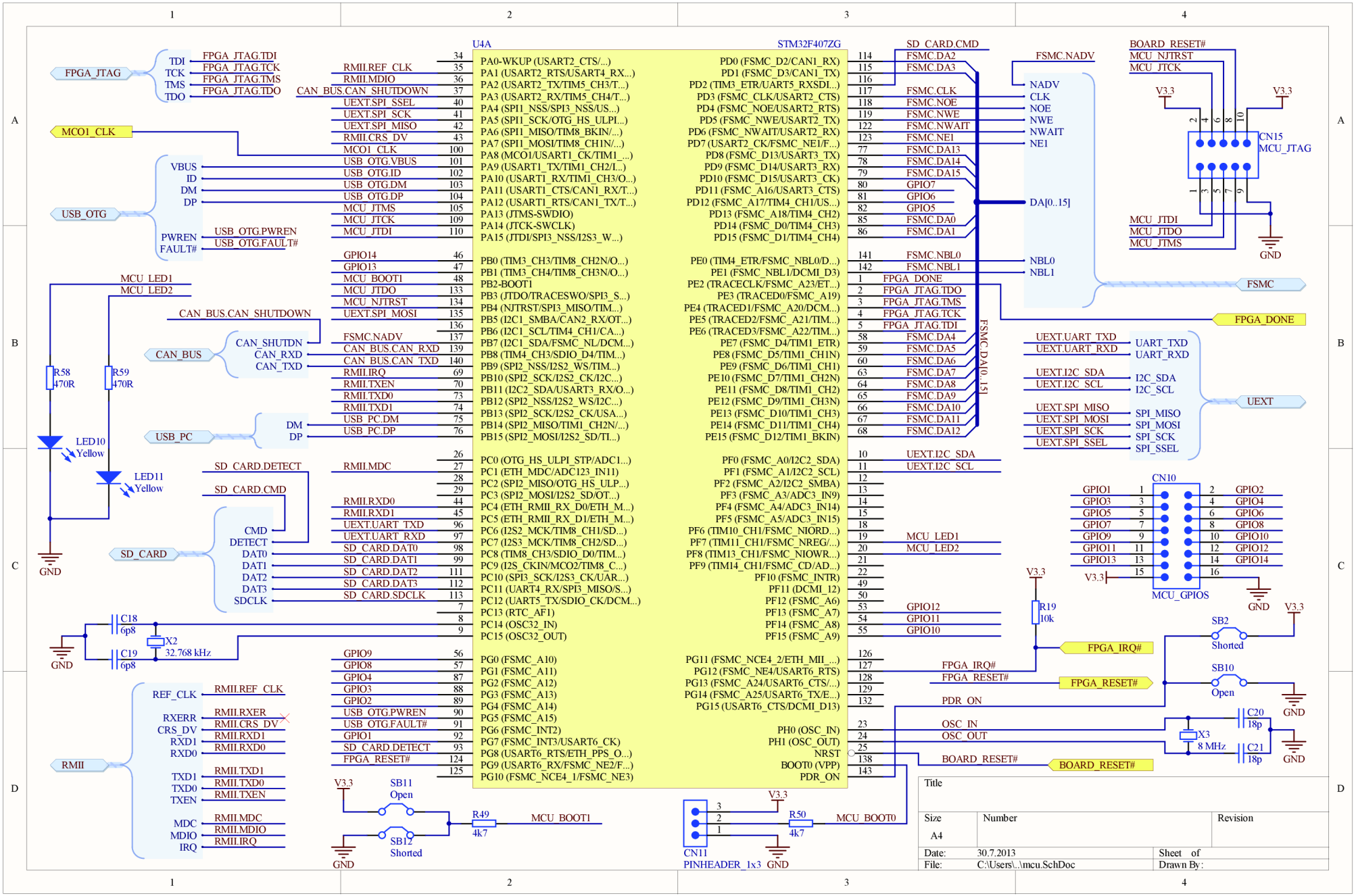
[51] Xilinx Inc.: Zynq-7000 All Programmable SoC First Generation Architecture. Specifikace produktu. [online], March 15, 2013. [cit. 2013-07-12].
URL <http://www.xilinx.com/support/documentation/data_sheets/
ds190-Zynq-7000-Overview.pdf>

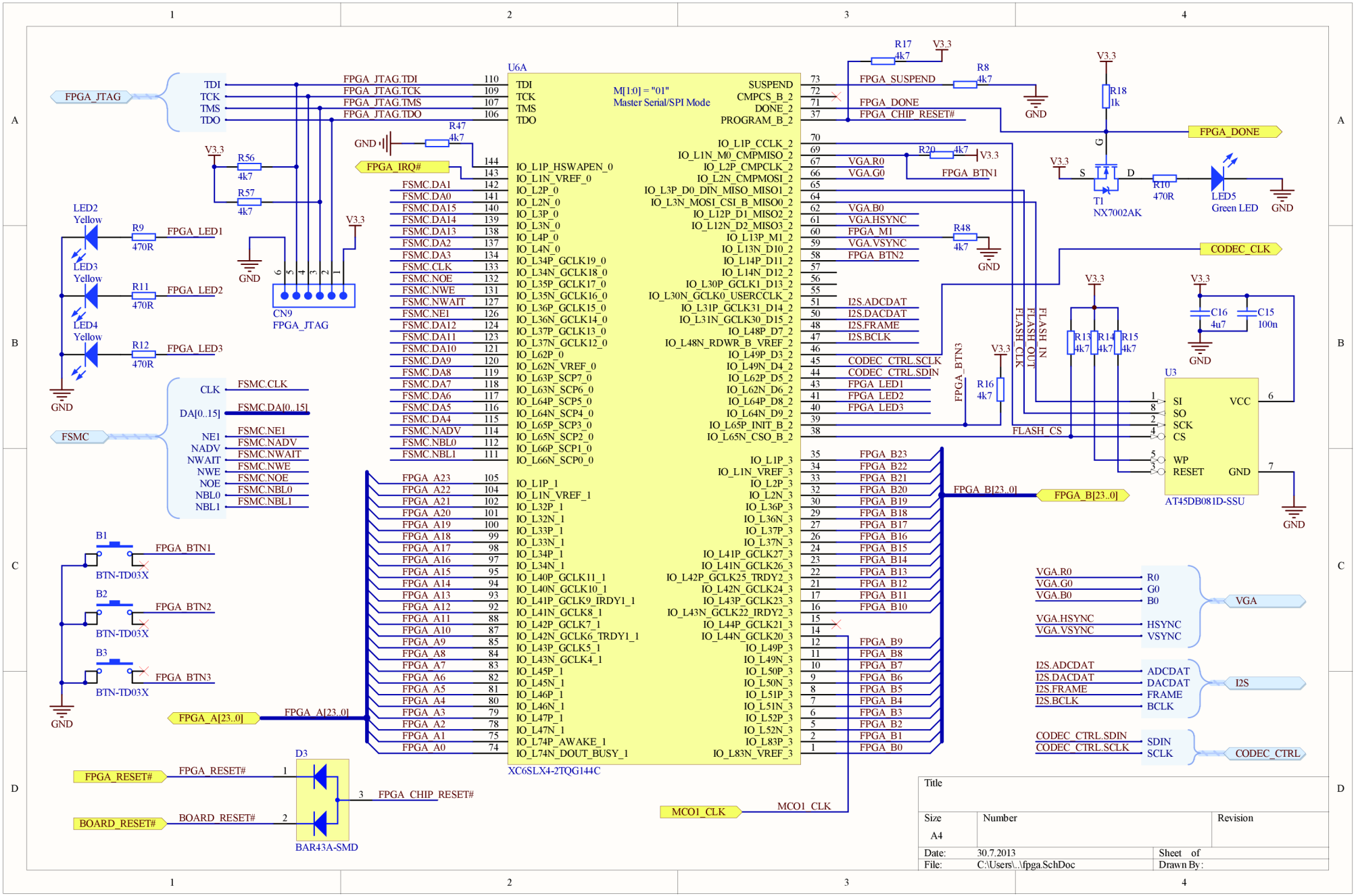
[52] Xilinx Inc.: Indirect Programming of SPI Serial Flash PROMs with Spartan-3A FPGAs. Aplikační poznámka. [online], March 24, 2009. [cit. 2013-07-12].
URL <http:
//www.xilinx.com/support/documentation/application_notes/xapp974.pdf>

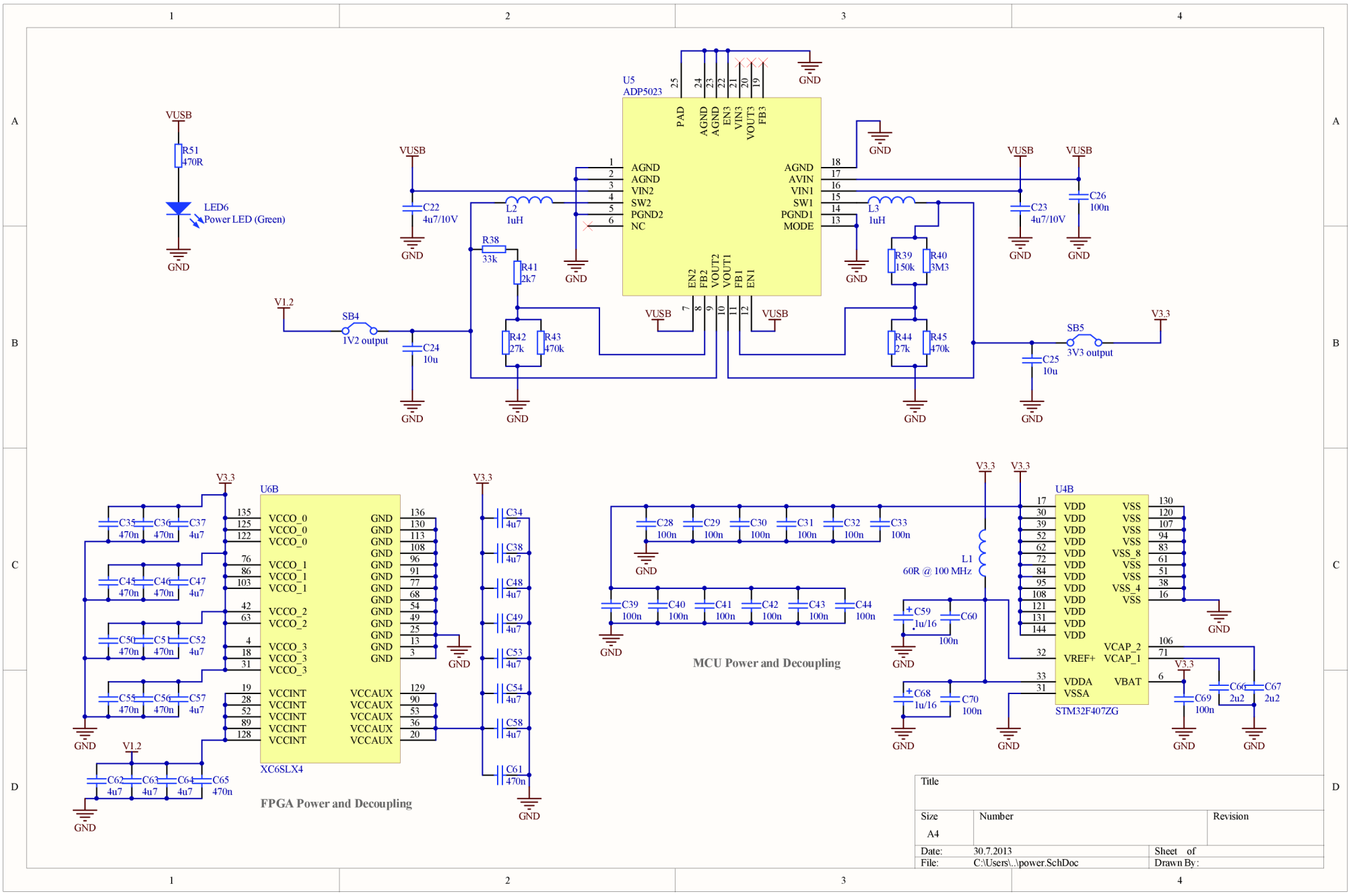
Příloha A

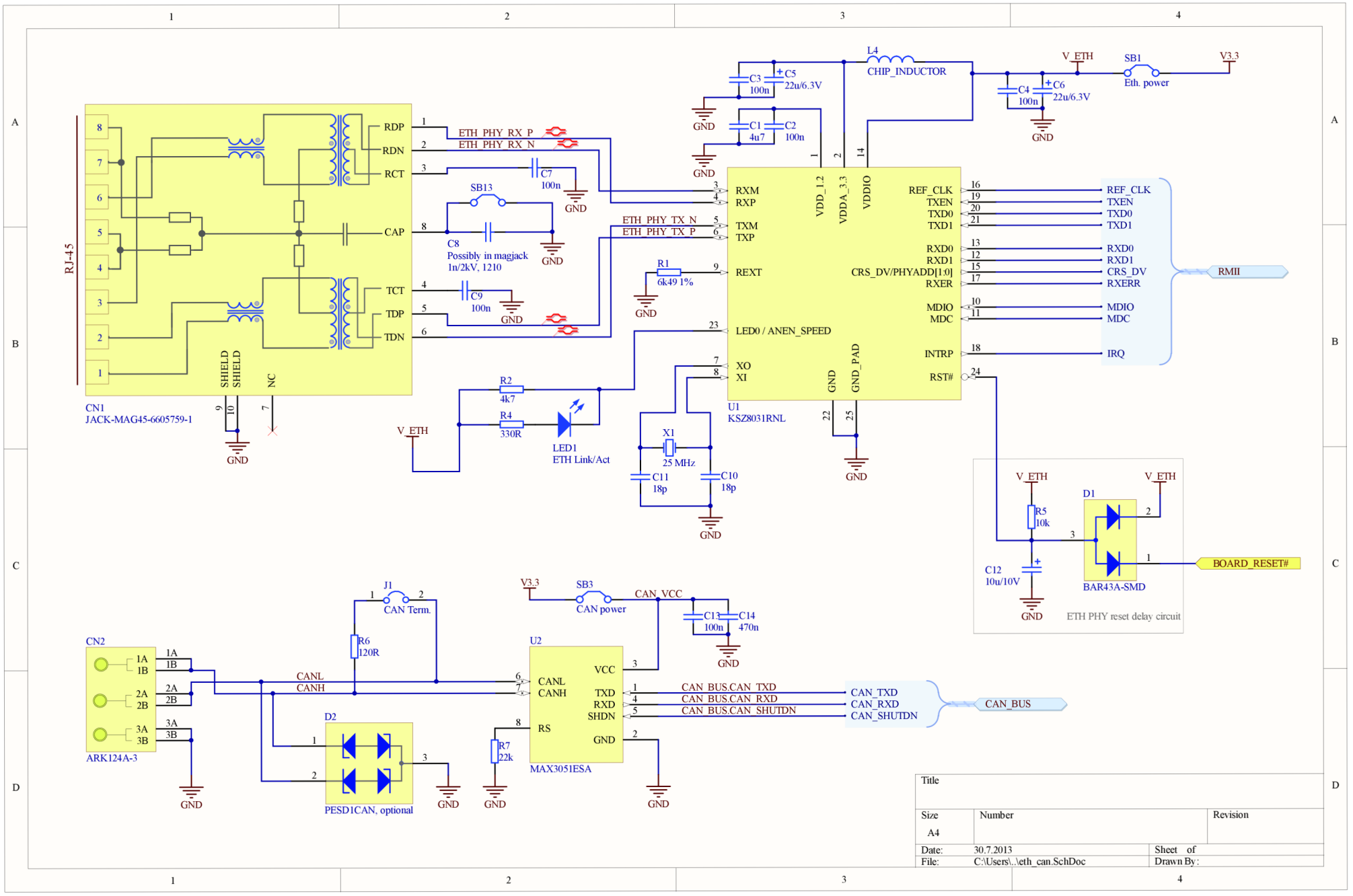
Schéma navržené vývojové platformy

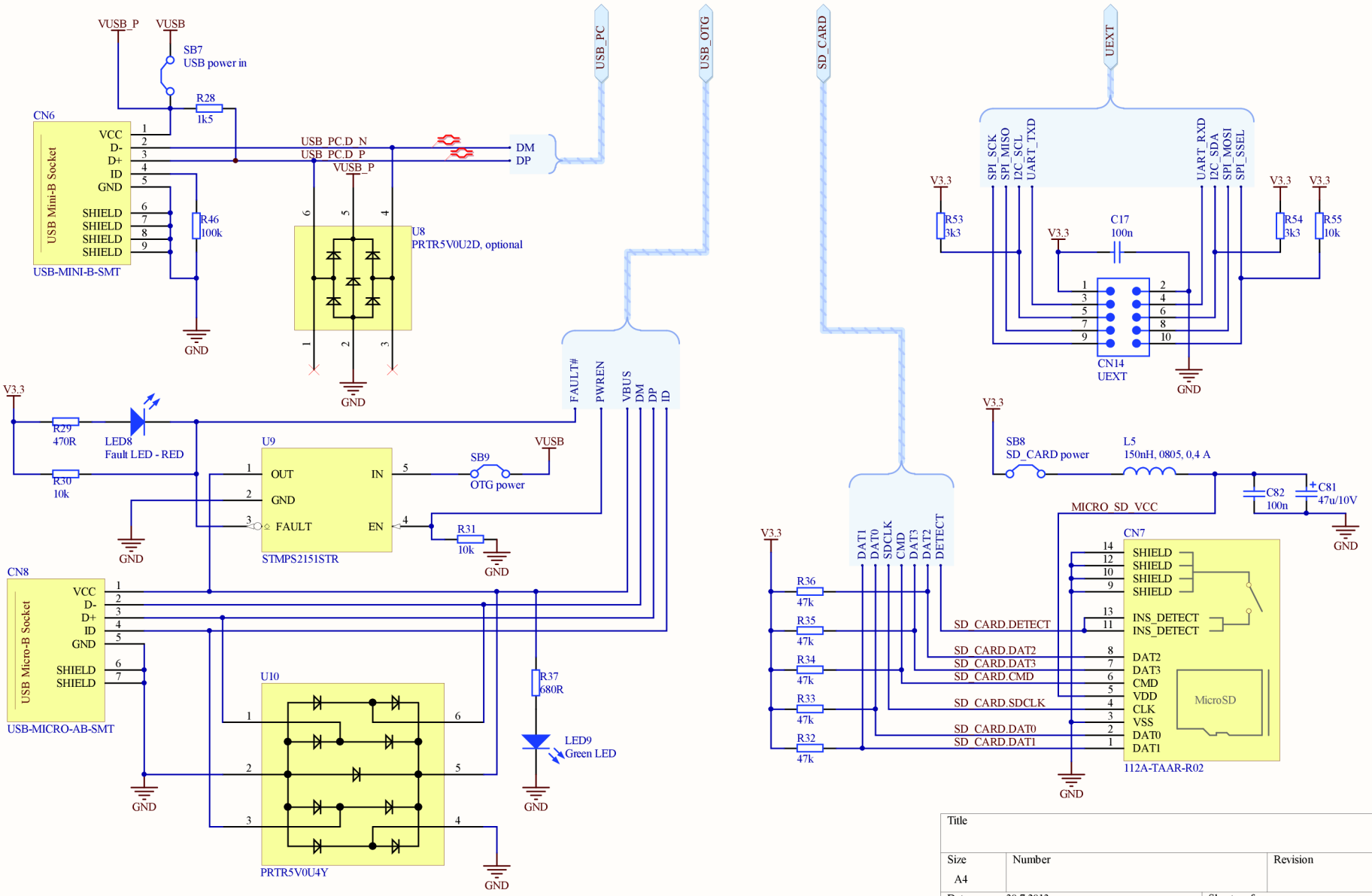

MCU
mcu.SchDoc
FPGA
fpga.SchDoc
POWER
power.SchDoc
FSMC
FPGA_JTAG
FPGA_IRQ#
FPGA_RESET#
FPGA_DONE
MCO1_CLK
BOARD_RESET#
RMII
CAN_BUS
USB_OTG
USB_PC
SD_CARD
UEXT
FPGA_A[23..0]
FPGA_B[23..0]
CODEC_CLK
CODEC_CTRL
I2S
VGA
ETH_CAN
eth_can.SchDoc
USB_SD
usb_sd.SchDoc
SOUND_VGA
sound_vga.SchDoc
V3.3
R52
4k7
C80
100n
B4
BTN-TD03X
GND
Title
Size
A4
Number
Revision
Date:
30.7.2013
Sheet of
File:
C:\Users\..\top.SchDoc
Drawn By:

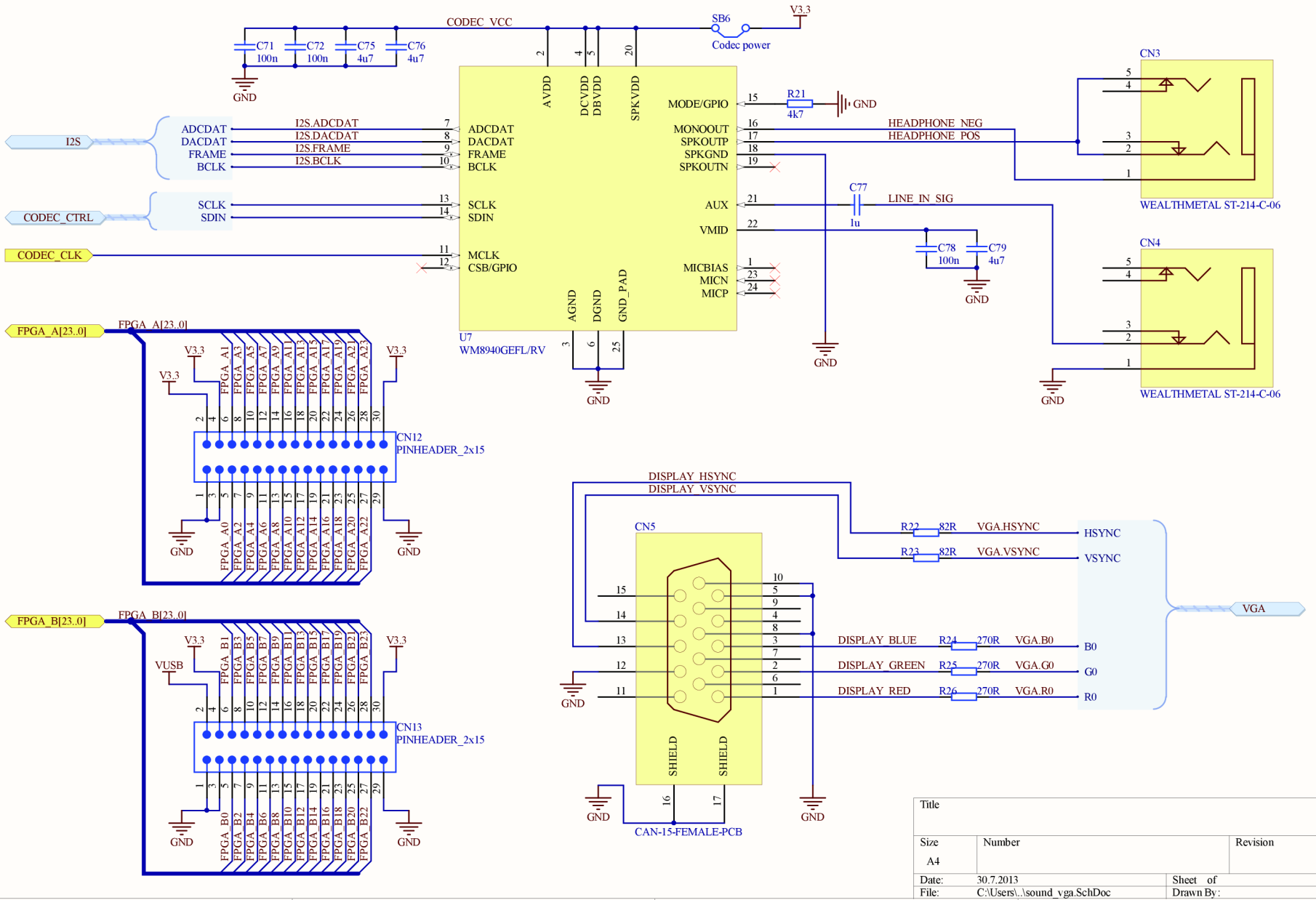

U4A
STM32F407ZG
FPGA_JTAG
TDI
TCK
TMS
TDO
FPGA_JTAG.TDI
FPGA_JTAG.TCK
FPGA_JTAG.TMS
FPGA_JTAG.TDO
MCO1_CLK
USB_OTG
VBUS
ID
DM
DP
PWREN
FAULT#
USB_OTG.PWREN
USB_OTG.FAULT#
MCU_LED1
MCU_LED2
R58
470R
R59
470R
LED10
Yellow
LED11
Yellow
GND
CAN_BUS.CAN_SHUTDOWN
CAN_BUS
CAN_SHUTDN
CAN_RXD
CAN_TXD
CAN_BUS.CAN_RXD
CAN_BUS.CAN_TXD
USB_PC
DM
DP
USB_PC.DM
USB_PC.DP
SD_CARD.DETECT
SD_CARD.CMD
SD_CARD
CMD
DETECT
DAT0
DAT1
DAT2
DAT3
SDCLK
SD_CARD.DAT0
SD_CARD.DAT1
SD_CARD.DAT2
SD_CARD.DAT3
SD_CARD.SDCLK
C18
6p8
C19
6p8
X2
32.768 kHz
RMII
REF_CLK
RXERR
CRS_DV
RXD1
RXD0
TXD1
TXD0
TXEN
MDC
MDIO
IRQ
RMII.REF_CLK
RMII.RXER
RMII.CRS_DV
RMII.RXD1
RMII.RXD0
RMII.TXD1
RMII.TXD0
RMII.TXEN
RMII.MDC
RMII.MDIO
RMII.IRQ
V3.3
SB11
Open
SB12
Shorted
R49
4k7
MCU_BOOT1
CN11
PINHEADER_1x3
R50
4k7
MCU_BOOT0
PA0-WKUP (USART2_CTS/...)
PA1 (USART2_RTS/USART4_RX...)
PA2 (USART2_TX/TIM5_CH3/T...)
PA3 (USART2_RX/TIM5_CH4/T...)
PA4 (SPI1_NSS/SPI3_NSS/US...)
PA5 (SPI1_SCK/OTG_HS_ULPI...)
PA6 (SPI1_MISO/TIM8_BKIN/...)
PA7 (SPI1_MOSI/TIM8_CH1N/...)
PA8 (MCO1/USART1_CK/TIM1_...)
PA9 (USART1_TX/TIM1_CH2/I...)
PA10 (USART1_RX/TIM1_CH3/O...)
PA11 (USART1_CTS/CAN1_RX/T...)
PA12 (USART1_RTS/CAN1_TX/T...)
PA13 (JTMS-SWDIO)
PA14 (JTCK-SWCLK)
PA15 (JTDI/SPI3_NSS/I2S3_W...)
PB0 (TIM3_CH3/TIM8_CH2N/O...)
PB1 (TIM3_CH4/TIM8_CH3N/O...)
PB2-BOOT1
PB3 (JTDO/TRACESWO/SPI3_S...)
PB4 (NJTRST/SPI3_MISO/TIM...)
PB5 (I2C1_SMBA/CAN2_RX/OT...)
PB6 (I2C1_SCL/TIM4_CH1/CA...)
PB7 (I2C1_SDA/FSMC_NL/DCM...)
PB8 (TIM4_CH3/SDIO_D4/TIM...)
PB9 (SPI2_NSS/I2S2_WS/TIM...)
PB10 (SPI2_SCK/I2S2_CK/I2C...)
PB11 (I2C2_SDA/USART3_RX/O...)
PB12 (SPI2_NSS/I2S2_WS/I2C...)
PB13 (SPI2_SCK/I2S2_CK/USA...)
PB14 (SPI2_MISO/TIM1_CH2N/...)
PB15 (SPI2_MOSI/I2S2_SD/TI...)
PC0 (OTG_HS_ULPI_STP/ADC1...)
PC1 (ETH_MDC/ADC123_IN11)
PC2 (SPI2_MISO/OTG_HS_ULP...)
PC3 (SPI2_MOSI/I2S2_SD/OT...)
PC4 (ETH_RMII_RX_D0/ETH_M...)
PC5 (ETH_RMII_RX_D1/ETH_M...)
PC6 (I2S2_MCK/TIM8_CH1/SD...)
PC7 (I2S3_MCK/TIM8_CH2/SD...)
PC8 (TIM8_CH3/SDIO_D0/TIM...)
PC9 (I2S_CKIN/MCO2/TIM8_C...)
PC10 (SPI3_SCK/I2S3_CK/UAR...)
PC11 (UART4_RX/SPI3_MISO/S...)
PC12 (UART5_TX/SDIO_CK/DCM...)
PC13 (RTC_AF1)
PC14 (OSC32_IN)
PC15 (OSC32_OUT)
PG0 (FSMC_A10)
PG1 (FSMC_A11)
PG2 (FSMC_A12)
PG3 (FSMC_A13)
PG4 (FSMC_A14)
PG5 (FSMC_A15)
PG6 (FSMC_INT2)
PG7 (FSMC_INT3/USART6_CK)
PG8 (USART6_RTS/ETH_PPS_O...)
PG9 (USART6_RX/FSMC_NE2/F...)
PG10 (FSMC_NCE4_1/FSMC_NE3)
PD0 (FSMC_D2/CAN1_RX)
PD1 (FSMC_D3/CAN1_TX)
PD2 (TIM3_ETR/UART5_RXSDI...)
PD3 (FSMC_CLK/USART2_CTS)
PD4 (FSMC_NOE/USART2_RTS)
PD5 (FSMC_NWE/USART2_TX)
PD6 (FSMC_NWAIT/USART2_RX)
PD7 (USART2_CK/FSMC_NE1/F...)
PD8 (FSMC_D13/USART3_TX)
PD9 (FSMC_D14/USART3_RX)
PD10 (FSMC_D15/USART3_CK)
PD11 (FSMC_A16/USART3_CTS)
PD12 (FSMC_A17/TIM4_CH1/US...)
PD13 (FSMC_A18/TIM4_CH2)
PD14 (FSMC_D0/TIM4_CH3)
PD15 (FSMC_D1/TIM4_CH4)
PE0 (TIM4_ETR/FSMC_NBL0/D...)
PE1 (FSMC_NBL1/DCMI_D3)
PE2 (TRACECLK/FSMC_A23/ET...)
PE3 (TRACED0/FSMC_A19)
PE4 (TRACED1/FSMC_A20/DCM...)
PE5 (TRACED2/FSMC_A21/TIM...)
PE6 (TRACED3/FSMC_A22/TIM...)
PE7 (FSMC_D4/TIM1_ETR)
PE8 (FSMC_D5/TIM1_CH1N)
PE9 (FSMC_D6/TIM1_CH1)
PE10 (FSMC_D7/TIM1_CH2N)
PE11 (FSMC_D8/TIM1_CH2)
PE12 (FSMC_D9/TIM1_CH3N)
PE13 (FSMC_D10/TIM1_CH3)
PE14 (FSMC_D11/TIM1_CH4)
PE15 (FSMC_D12/TIM1_BKIN)
PF0 (FSMC_A0/I2C2_SDA)
PF1 (FSMC_A1/I2C2_SCL)
PF2 (FSMC_A2/I2C2_SMBA)
PF3 (FSMC_A3/ADC3_IN9)
PF4 (FSMC_A4/ADC3_IN14)
PF5 (FSMC_A5/ADC3_IN15)
PF6 (TIM10_CH1/FSMC_NIORD...)
PF7 (TIM11_CH1/FSMC_NREG/...)
PF8 (TIM13_CH1/FSMC_NIOWR...)
PF9 (TIM14_CH1/FSMC_CD/AD...)
PF10 (FSMC_INTR)
PF11 (DCMI_12)
PF12 (FSMC_A6)
PF13 (FSMC_A7)
PF14 (FSMC_A8)
PF15 (FSMC_A9)
PG11 (FSMC_NCE4_2/ETH_MII_...)
PG12 (FSMC_NE4/USART6_RTS)
PG13 (FSMC_A24/USART6_CTS/...)
PG14 (FSMC_A25/USART6_TX/E...)
PG15 (USART6_CTS/DCMI_D13)
PH0 (OSC_IN)
PH1 (OSC_OUT)
NRST
BOOT0 (VPP)
PDR_ON
SD_CARD.CMD
FSMC.DA2
FSMC.DA3
FSMC.CLK
FSMC.NOE
FSMC.NWE
FSMC.NWAIT
FSMC.NE1
FSMC.DA13
FSMC.DA14
FSMC.DA15
GPIO7
GPIO6
GPIO5
FSMC.DA0
FSMC.DA1
FSMC.NBL0
FSMC.NBL1
FPGA_DONE
FPGA_JTAG.TDO
FPGA_JTAG.TMS
FPGA_JTAG.TCK
FPGA_JTAG.TDI
FSMC.DA4
FSMC.DA5
FSMC.DA6
FSMC.DA7
FSMC.DA8
FSMC.DA9
FSMC.DA10
FSMC.DA11
FSMC.DA12
FSMC.DA[0..15]
UEXT.I2C_SDA
UEXT.I2C_SCL
MCU_LED1
MCU_LED2
GPIO12
GPIO11
GPIO10
FPGA_IRQ#
FPGA_RESET#
PDR_ON
OSC_IN
OSC_OUT
BOARD_RESET#
FSMC.NADV
NADV
CLK
NOE
NWE
NWAIT
NE1
DA[0..15]
NBL0
NBL1
FSMC
FPGA_DONE
UEXT.UART_TXD
UEXT.UART_RXD
UEXT.I2C_SDA
UEXT.I2C_SCL
UEXT.SPI_MISO
UEXT.SPI_MOSI
UEXT.SPI_SCK
UEXT.SPI_SSEL
UART_TXD
UART_RXD
I2C_SDA
I2C_SCL
SPI_MISO
SPI_MOSI
SPI_SCK
SPI_SSEL
UEXT
BOARD_RESET#
MCU_NJTRST
MCU_JTCK
MCU_JTDI
MCU_JTDO
MCU_JTMS
CN15
MCU_JTAG
V3.3
GND
CN10
GPIO1
GPIO2
GPIO3
GPIO4
GPIO5
GPIO6
GPIO7
GPIO8
GPIO9
GPIO10
GPIO11
GPIO12
GPIO13
GPIO14
MCU_GPIOS
R19
10k
SB2
Shorted
SB10
Open
X3
8 MHz
C20
18p
C21
18p
Title
Size
A4
Number
Revision
Date:
30.7.2013
Sheet of
File:
C:\Users\..\mcu.SchDoc
Drawn By:


U6A
XC6SLX4-2TQG144C
M[1:0] = "01"
Master Serial/SPI Mode
FPGA_JTAG
TDI
TCK
TMS
TDO
FPGA_JTAG.TDI
FPGA_JTAG.TCK
FPGA_JTAG.TMS
FPGA_JTAG.TDO
CN9
FPGA_JTAG
FPGA_IRQ#
LED2
Yellow
LED3
Yellow
LED4
Yellow
FPGA_LED1
FPGA_LED2
FPGA_LED3
FSMC
FSMC.CLK
FSMC.DA[0..15]
FSMC.NE1
FSMC.NADV
FSMC.NWAIT
FSMC.NWE
FSMC.NOE
FSMC.NBL0
FSMC.NBL1
FPGA_A[23..0]
FPGA_B[23..0]
B1
B2
B3
FPGA_BTN1
FPGA_BTN2
FPGA_BTN3
BTN-TD03X
D3
BAR43A-SMD
FPGA_RESET#
BOARD_RESET#
FPGA_CHIP_RESET#
MCO1_CLK
FPGA_SUSPEND
FPGA_DONE
FPGA_CHIP_RESET#
VGA.R0
VGA.G0
VGA.B0
VGA.HSYNC
VGA.VSYNC
FPGA_M1
FPGA_BTN2
I2S.ADCDAT
I2S.DACDAT
I2S.FRAME
I2S.BCLK
CODEC_CTRL.SDIN
CODEC_CTRL.SCLK
FLASH_CS
FLASH_CLK
FLASH_OUT
FLASH_IN
CODEC_CLK
T1
NX7002AK
LED5
Green LED
U3
AT45DB081D-SSU
SI
SO
SCK
CS
WP
RESET
VCC
GND
VGA
I2S
CODEC_CTRL
R0
G0
B0
HSYNC
VSYNC
ADCDAT
DACDAT
FRAME
BCLK
SDIN
SCLK
Title
Size
A4
Number
Revision
Date:
30.7.2013
Sheet of
File:
C:\Users\..\fpga.SchDoc
Drawn By:


U5
ADP5023
PAD
AGND
AGND
EN3
VIN3
VOUT3
FB3
AGND
AGND
VIN2
SW2
PGND2
NC
AGND
AVIN
VIN1
SW1
PGND1
MODE
EN2
FB2
VOUT2
VOUT1
FB1
EN1
VUSB
GND
V3.3
V1.2
R51
470R
LED6
Power LED (Green)
C22
4u7/10V
L2
1uH
R38
33k
R41
2k7
R42
27k
R43
470k
C24
10u
SB4
1V2 output
L3
1uH
R39
150k
R40
3M3
R44
27k
R45
470k
C23
4u7/10V
C26
100n
C25
10u
SB5
3V3 output
U6B
VCCO_0
VCCO_1
VCCO_2
VCCO_3
VCCINT
VCCAUX
GND
XC6SLX4
C35
C36
C37
C45
C46
C47
C50
C51
C52
C55
C56
C57
470n
4u7
C62
C63
C64
C65
C34
C38
C48
C49
C53
C54
C58
C61
FPGA Power and Decoupling
C28
C29
C30
C31
C32
C33
C39
C40
C41
C42
C43
C44
100n
L1
60R @ 100 MHz
C59
1u/16
C60
C68
C70
U4B
VDD
VSS
VSS_8
VSS_4
VREF+
VDDA
VSSA
VCAP_2
VCAP_1
VBAT
STM32F407ZG
C66
C67
2u2
C69
MCU Power and Decoupling
Title
Size
A4
Number
Revision
Date:
30.7.2013
Sheet of
File:
C:\Users\..\power.SchDoc
Drawn By:


RJ-45
CN1
JACK-MAG45-6605759-1
RDP
RDN
RCT
CAP
TCT
TDP
TDN
SHIELD
SHIELD
NC
ETH_PHY_RX_P
ETH_PHY_RX_N
ETH_PHY_TX_N
ETH_PHY_TX_P
C7 100n
SB13
C8
Possibly in magjack
1n/2kV, 1210
C9 100n
R1 6k49 1%
R2 4k7
R4 330R
LED1
ETH Link/Act
V_ETH
GND
L4
CHIP_INDUCTOR
C3 100n
C5 22u/6.3V
C1 4u7
C2 100n
C4 100n
C6 22u/6.3V
V_ETH
SB1
Eth. power
V3.3
U1
KSZ8031RNL
RXM
RXP
TXM
TXP
REXT
LED0 / ANEN_SPEED
XO
XI
VDD_1.2
VDDA_3.3
VDDIO
GND
GND_PAD
REF_CLK
TXEN
TXD0
TXD1
RXD0
RXD1
CRS_DV/PHYADD[1:0]
RXER
MDIO
MDC
INTRP
RST#
RMII
X1
25 MHz
C11 18p
C10 18p
V_ETH
D1
R5 10k
C12 10u/10V
BAR43A-SMD
BOARD_RESET#
ETH PHY reset delay circuit
CN2
ARK124A-3
1A
1B
2A
2B
3A
3B
J1
CAN Term.
R6 120R
CANL
CANH
D2
PESD1CAN, optional
V3.3
SB3
CAN power
CAN_VCC
C13 100n
C14 470n
U2
MAX3051ESA
VCC
CANL
CANH
TXD
RXD
SHDN
RS
GND
R7 22k
CAN_BUS.CAN_TXD
CAN_BUS.CAN_RXD
CAN_BUS.CAN_SHUTDN
CAN_TXD
CAN_RXD
CAN_SHUTDN
CAN_BUS
Title
Size
A4
Number
Revision
Date: 30.7.2013
Sheet of
File: C:\Users\..\eth_can.SchDoc
Drawn By:


VUSB_P
VUSB
SB7
USB power in
R28
1k5
CN6
USB Mini-B Socket
VCC
D-
D+
ID
GND
SHIELD
SHIELD
SHIELD
SHIELD
USB-MINI-B-SMT
R46
100k
GND
USB_PC.D_N
USB_PC.D_P
VUSB_P
DM
DP
U8
PRTR5V0U2D, optional
GND
USB_PC
USB_OTG
SD_CARD
UEXT
SPI_SCK
SPI_MISO
I2C_SCL
UART_TXD
UART_RXD
I2C_SDA
SPI_MOSI
SPI_SSEL
V3.3
R53
3k3
C17
100n
R54
3k3
R55
10k
CN14
UEXT
GND
FAULT#
PWREN
VBUS
DM
DP
ID
V3.3
R29
470R
LED8
Fault LED - RED
R30
10k
U9
OUT
GND
FAULT
IN
EN
STMPS2151STR
SB9
OTG power
VUSB
R31
10k
GND
SB8
SD_CARD power
L5
150nH, 0805, 0,4 A
C82
100n
C81
47u/10V
MICRO_SD_VCC
GND
CN8
USB Micro-B Socket
VCC
D-
D+
ID
GND
SHIELD
SHIELD
USB-MICRO-AB-SMT
U10
PRTR5V0U4Y
R37
680R
LED9
Green LED
GND
DAT1
DAT0
SDCLK
CMD
DAT3
DAT2
DETECT
V3.3
R36
47k
R35
47k
R34
47k
R33
47k
R32
47k
SD_CARD.DETECT
SD_CARD.DAT2
SD_CARD.DAT3
SD_CARD.CMD
SD_CARD.SDCLK
SD_CARD.DAT0
SD_CARD.DAT1
CN7
SHIELD
SHIELD
SHIELD
SHIELD
INS_DETECT
INS_DETECT
DAT2
DAT3
CMD
VDD
CLK
VSS
DAT0
DAT1
MicroSD
112A-TAAR-R02
GND
Title
Size
A4
Number
Revision
Date:
30.7.2013
Sheet of
File:
C:\Users\..\usb_sd.SchDoc
Drawn By:


CODEC_VCC
V3.3
SB6
Codec power
C71 100n
C72 100n
C75 4u7
C76 4u7
GND
AVDD
DCVDD
DBVDD
SPKVDD
MODE/GPIO
R21 4k7
I2S
ADCDAT
DACDAT
FRAME
BCLK
I2S.ADCDAT
I2S.DACDAT
I2S.FRAME
I2S.BCLK
MONOOUT
SPKOUTP
SPKGND
SPKOUTN
HEADPHONE_NEG
HEADPHONE_POS
CN3
WEALTHMETAL ST-214-C-06
CODEC_CTRL
SCLK
SDIN
AUX
VMID
C77 1u
LINE_IN_SIG
C78 100n
C79 4u7
CODEC_CLK
MCLK
CSB/GPIO
MICBIAS
MICN
MICP
CN4
WEALTHMETAL ST-214-C-06
AGND
DGND
GND_PAD
U7
WM8940GEFL/RV
FPGA_A[23..0]
CN12
PINHEADER_2x15
FPGA_B[23..0]
VUSB
CN13
PINHEADER_2x15
DISPLAY_HSYNC
DISPLAY_VSYNC
CN5
R22 82R VGA.HSYNC
R23 82R VGA.VSYNC
DISPLAY_BLUE R24 270R VGA.B0
DISPLAY_GREEN R25 270R VGA.G0
DISPLAY_RED R26 270R VGA.R0
HSYNC
VSYNC
VGA
B0
G0
R0
SHIELD
SHIELD
CAN-15-FEMALE-PCB
Title
Size A4
Number
Revision
Date: 30.7.2013
Sheet of
File: C:\Users\..\sound_vga.SchDoc
Drawn By:

Příloha B

Osazovací plán hardwarového prototypu

Tabulka B.1: Rozpiska součástek pro osazení DPS

Komponenta	Typ	Hodnota	Pouzdro
B1, B2, B3, B4	SMT push button TD-03XBT-A00-TR	BTN-TD03X	BTN-TD-03X
C1, C16, C34, C37, C38, C47, C48, C49, C52, C53, C54, C57, C58, C62, C63, C64, C75, C76, C79	Capacitor	4u7	C0805
C2, C3, C4, C7, C9, C13, C15, C17, C26, C28, C29, C30, C31, C32, C33, C39, C40, C41, C42, C43, C44, C60, C69, C70, C71, C72, C78, C80, C82	Capacitor	100n	C0603
C5, C6	Tantal Capacitor (Surface Mount)	22u/6.3V	TCA
C8	Capacitor	Possibly in magjack	C1210
C10, C11, C20, C21	Capacitor	18p	C0603
C12	Tantal Capacitor (Surface Mount)	10u/10V	TCA
C14, C35, C36, C45, C46, C50, C51, C55, C56, C61, C65	Capacitor	470n	C0603
C18, C19	Capacitor	6p8	C0603
C22, C23	Capacitor	4u7/10V	C1206
C24, C25	Capacitor	10u	C0805
C59, C68	Tantal Capacitor (Surface Mount)	1u/16	TCA

(pokračuje na další straně)

Tabulka B.1 – pokračování

Komponenta	Typ	Hodnota	Pouzdro
C66, C67	Capacitor	2u2	C0805
C77	Capacitor	1u	C0805
C81	Tantal Capacitor (Surface Mount)	47u/10V	TCC
CN1	TE CONNECTIVITY MAGJACK MAG45, RJ-45, integ. magnetics, tab up	JACK-MAG45-6605759-1	MAG45-6605759-1
CN2	XINYA Wire Terminal Bar with Spring, 3 contacts	ARK124A-3	ARK124A/3
CN3, CN4	Wealthmetal Jack; 3,5 mm stereo; PCB mounting	WEALTHMETAL ST-214-C-06	JACK_ST-214-C
CN5	CANON 15 connector; PCB; female; 90deg	CAN-15-FEMALE-PCB	CAN-15-PCB-FEMALE-90deg
CN6	USB Mini-B socket, SMD	USB-MINI-B-SMT	USB-MINI-B-SMT
CN7	ATTEND MicroSD Slot, Push-Push, SMD	112A-TAAR-R02	MICRO-SD-SLOT-112-TAAR-R02
CN8	USB Micro-AB socket, SMD	USB-MICRO-AB-SMT	USB-MICRO-AB-SMT
CN9	Pin Header 1x6 pins, 2.54mm pitch	FPGA_JTAG	PIN_HEADER_1x6
CN10		MCU_GPIOS	PIN_HEADER_2x8
CN11	Pin Header 1x3 pins, 2,54mm pitch	PINHEADER_1x3	PIN_HEADER_1x3
CN12, CN13	Pin Header 2x15 pins, 2,54mm pitch	PINHEADER_2x15	PIN_HEADER_2x15
CN14	Pin Header 2x5 pins with key, 2.54mm pitch	UEXT	IDC10_2x5_WITH_KEY
CN15	Pin Header 2x5 pins, 2.54mm pitch	MCU_JTAG	PIN_HEADER_2x5
D1, D3	Dual Schottky Diode; 2x 30V/0,2A; SMD; common cathode	BAR43A-SMD	SOT23-3N
D2	TVS Diode Set; CAN Bus Protection; SOT-23	PESD1CAN, optional	SOT23-3N
J1		CAN Term.	JUMPER
L1		60R @ 100 MHz	C0805
L2, L3	SMD Chip Inductor	1uH	INDC3225N
L4	SMD Chip Inductor	CHIP_INDUCTOR	C0805
L5	SMD Chip Inductor	150nH, 0805, 0,4 A	C0805
LED1	Typical RED, GREEN, YELLOW, AMBER GaAs LED	ETH Link/Act	D0603
LED2, LED3, LED4, LED10, LED11	Typical RED, GREEN, YELLOW, AMBER GaAs LED	Yellow	D0603

(pokračuje na další straně)

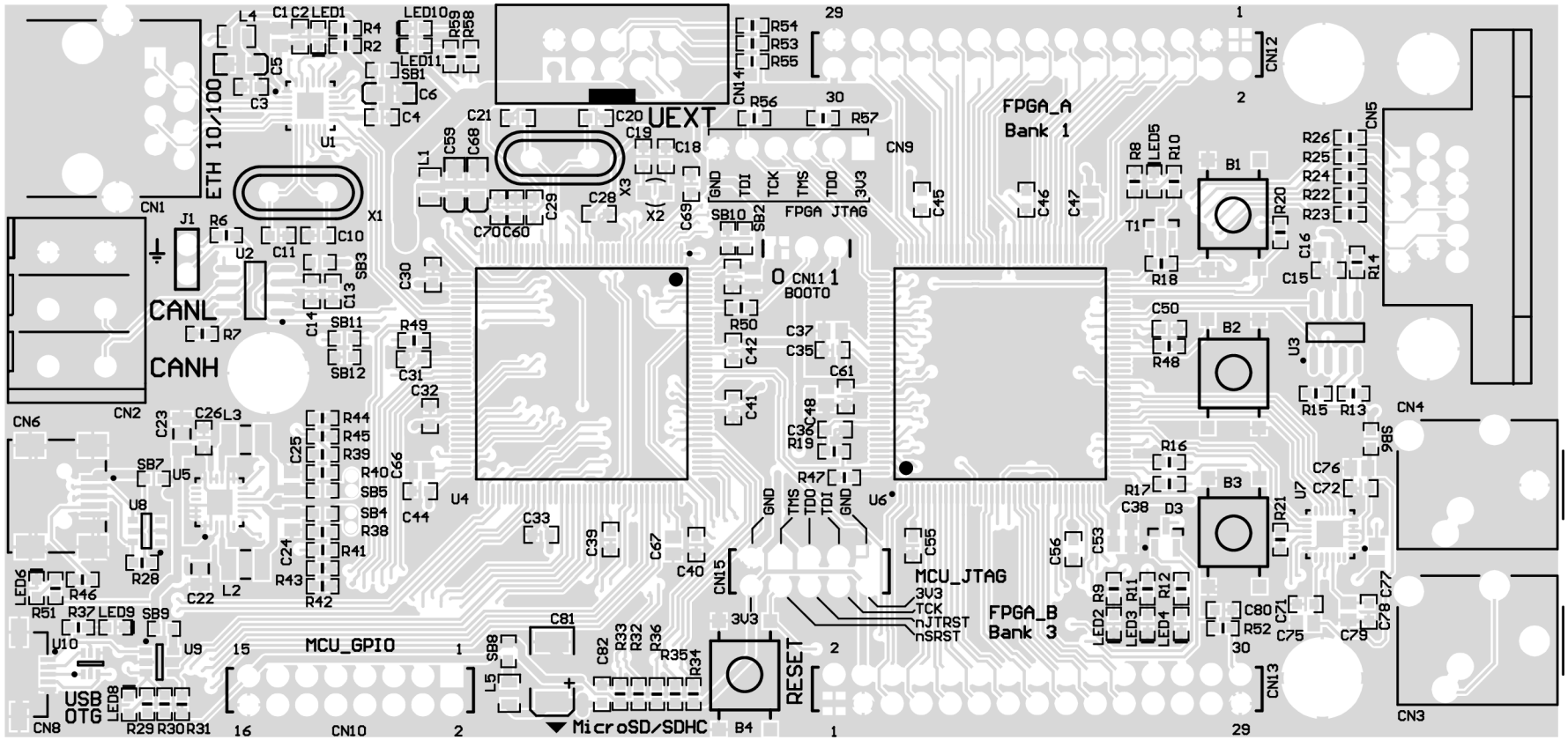
Tabulka B.1 – pokračování

Komponenta	Typ	Hodnota	Pouzdro
LED5, LED9	Typical RED, GREEN, YELLOW, AMBER GaAs LED	Green LED	D0603
LED6	Typical RED, GREEN, YELLOW, AMBER GaAs LED	Power LED (Green)	D0603
LED8	Typical RED, GREEN, YELLOW, AMBER GaAs LED	Fault LED - RED	D0603
R1	Resistor	6k49 1%	R0805
R2, R8, R13, R14, R15, R16, R17, R20, R21, R47, R48, R49, R50, R52, R56, R57	Resistor	4k7	R0603
R4	Resistor	330R	R0603
R5, R19, R30, R31, R55	Resistor	10k	R0603
R6	Resistor	120R	R0603
R7	Resistor	22k	R0603
R9, R10, R11, R12, R29, R51, R58, R59	Resistor	470R	R0603
R18	Resistor	1k	R0603
R22, R23	Resistor	82R	R0603
R24, R25, R26	Resistor	270R	R0603
R28	Resistor	1k5	R0603
R32, R33, R34, R35, R36	Resistor	47k	R0603
R37	Resistor	680R	R0603
R38	Resistor	33k	R0603
R39	Resistor	150k	R0603
R40	Resistor	3M3	R0603
R41	Resistor	2k7	R0603
R42, R44	Resistor	27k	R0603
R43, R45	Resistor	470k	R0603
R46	Resistor	100k	R0603
R53, R54	Resistor	3k3	R0603
SB1	Solder bridge	Eth. power	SB0603
SB2, SB12	Solder bridge	Shorted	SB0603
SB3	Solder bridge	CAN power	SB0603
SB4	Solder bridge	1V2 output	SB0603
SB5	Solder bridge	3V3 output	SB0603
SB6	Solder bridge	Codec power	SB0603
SB7	Solder bridge	USB power in	SB0603
SB8	Solder bridge	SD_CARD power	SB0603
SB9	Solder bridge	OTG power	SB0603
SB10, SB11	Solder bridge	Open	SB0603
SB13	Solder bridge		SB0603
T1	MOSFET 60 V, 0.19 A, SOT23	NX7002AK	SOT23-3N

(pokračuje na další straně)

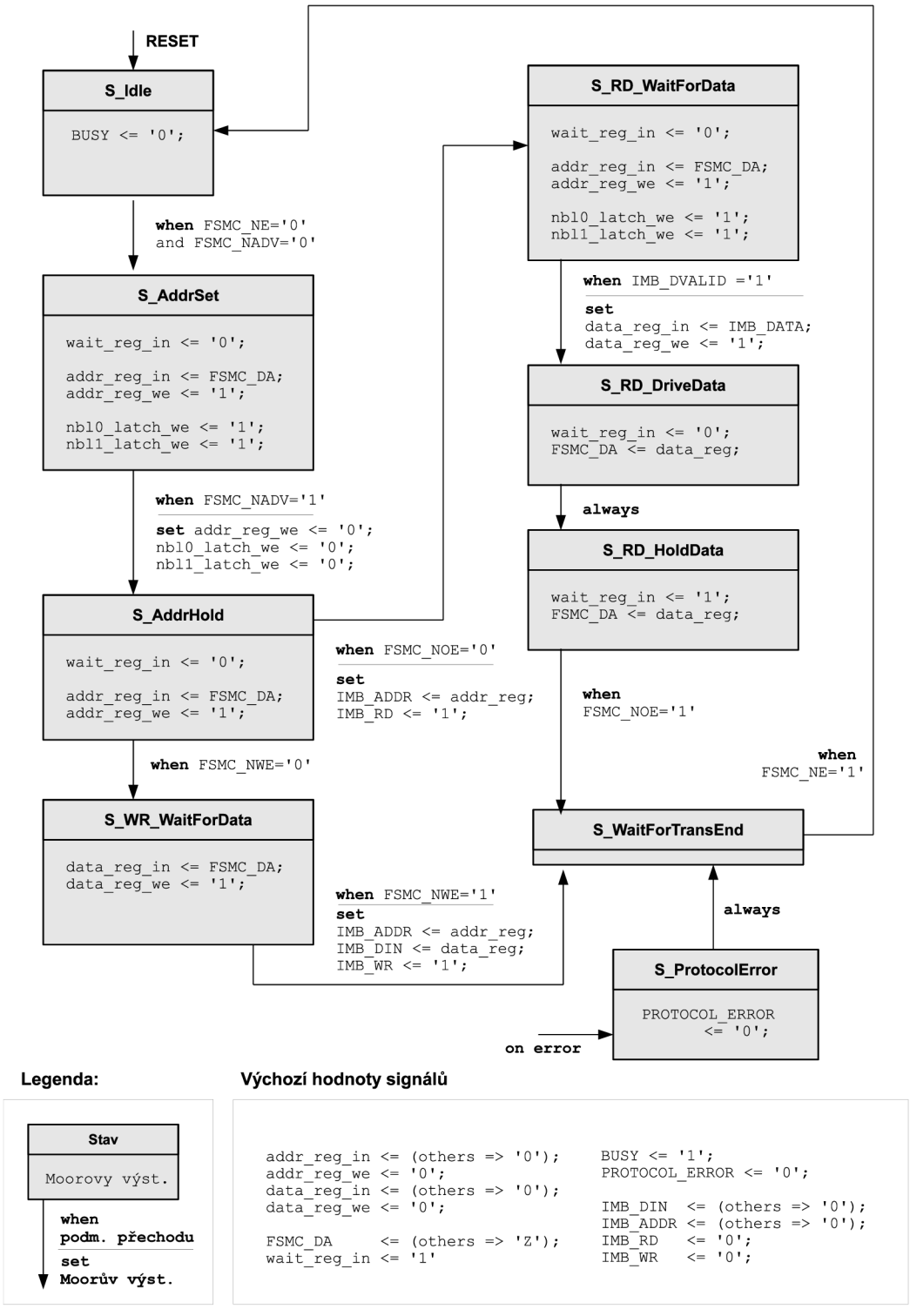
Tabulka B.1 – pokračování

Komponenta	Typ	Hodnota	Pouzdro
U1	Micrel 10Base-T/100Base-TX Ethernet PHY, RMII	KSZ8031RNL	QFN50P400X400-24VBM
U2	MAXIM CAN Transceiver, 1Mbps, 3V3	MAX3051ESA	SOIC127P600-8N
U3	ATMEL AT45DB081D serial SPI data flash	AT45DB081D	SOIC127P600-8M
U4	STM32F407 MCU, 144pin package	STM32F407ZG	TSQFP50P2200X2200-144N
U5	Dual 3 MHz 800 mA buck switcher + single 300 mA LDO	ADP5023	QFN50P400X400-24WAM-ADP5023
U6	Xilinx Spartan-6 LX 4 FPGA	XC6SLX4-2TQG144C	TSQFP50P2200X2200-144N
U7	Wolfson Mono CODEC with Speaker Driver	WM8940GEFL/RV	QFN50P400X400-24V2M
U8	Integrated double ESD protection	PRTR5V0U2D, optional	SOT457
U9	STM Single Channel Power Switch w. current limit	STMPS2151STR	SOT23-5AN
U10	Integrated quad ESD protection	PRTR5V0U4Y	SOT363-6N
X1	Quartz crystal	25 MHz	XTAL_HC49
X2	Quartz crystal	32.768 kHz	XTAL_CYLIND_6.2x2.1-_VERT
X3	Quartz crystal	8 MHz	XTAL_HC49


ETH 10/100
UEXT
FPGA_A
Bank 1
FPGA_B
Bank 3
FPGA JTAG
MCU_JTAG
MCU_GPIO
RESET
MicroSD/SDHC
USB
OTG
CANL
CANH
BOOT0
CN1
CN2
CN3
CN4
CN5
CN6
CN8
CN9
CN10
CN11
CN12
CN13
CN14
CN15
B1
B2
B3
B4
J1
X1
X2
X3
U1
U2
U3
U4
U5
U6
U7
U8
U9
U10
T1
D3
L1
L2
L3
L4
L5
3V3
GND
TDI
TCK
TMS
TDO
nJTRST
nSRST

Příloha C

Schéma řídicího FSM jednotky `fsmc_slave`



Obrázek C.1: Řídicí automat jednotky `fsmc_slave`

Příloha D

Obsah přiloženého disku CD

Přiložené CD obsahuje text této diplomové práce ve zdrojové podobně i ve formátu PDF, dále veškeré zdrojové texty systému v jazyce VHDL a také úplné podklady pro výrobu desky plošných spojů hardwarového prototypu.

Složka	Obsah složky
`/thesis_text/pdf`	Text práce ve formátu PDF
`/thesis_text/latex`	Text práce ve formátu LaTeX[1]
`/demo_apps`	Zdrojové texty demonstračních aplikací, jak byly popsány v kapitole 7. Každá aplikace se skládá ze software pro mikrokontroler a projektu s obvodem pro čip FPGA vytvořeném v nástroji Xilinx ISE Project Navigator 14.3.
`/impact_scripts`	Dávkové soubory pro řízení aplikace Xilinx ISE iMPACT pro nahrání konfiguračního řetězce do vývojového přípravku.
`/pcb_design/project`	Projekt s návrhem desky plošného spoje pro software Altium Designer verze 10.391.22084.
`/pcb_design/lib`	Knihovny komponent, které byly autorem vytvořeny během návrhu DPS a nejsou tedy součástí distribuce aplikace Altium Designer.
`/pcb_design/factory`	Podklady pro výrobu desky plošných spojů ve formátech Gerber RS-274X (`*.gpi`, `*.gbr`) a Excellon (`*.exc`, `*.dri`).

Tabulka D.1: Obsah přiloženého CD

[1] Zpracování zdrojových textů práce ve formátu LaTeX bylo provedeno pomocí softwarového balíku `MiKTeX` ve verzi `2.9`.