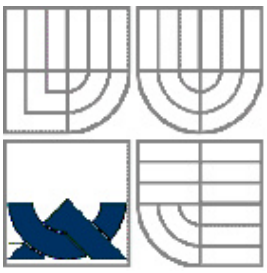


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

TESTOVÁNÍ VÝUKOVÉ PLATFORMY FITKIT

FITKIT PLATFORM TESTING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ FILIP

VEDOUCÍ PRÁCE
SUPERVISOR

JAROSLAV ŠKARVADA, Ing.

BRNO 2007

Zadání diplomové práce

Řešitel: **Filip Tomáš, Bc.**
Obor: Počítačové systémy a sítě
Téma: **Testování výukové platformy FITkit**
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s architekturou výukové platformy FITkit, způsobem práce s ní a s postupy jejího programování.
2. Seznamte se s jazykem VHDL a s možnostmi testování jednotlivých hardwarových komponent obsažených ve výukové platformě FITkit.
3. Navrhněte robustní a samočinný způsob testování platformy FITkit, který umožní prověřit funkčnost všech jejích komponent. Uvažujte i možnost testování jednotlivých obvodových spojů. Při návrhu testů se zaměřte i na možnost prostorové lokalizace případných chyb.
4. Návrh prakticky realizujte.
5. Ověřte funkčnost realizovaného způsobu testování na existujícím HW.

Literatura:

- dle pokynu vedoucího
- dostupné na URL: http://merlin.fit.vutbr.cz/FITkit/

Při obhajobě semestrální části diplomového projektu je požadováno:

- Splnění bodů zadání 1-3

Vedoucí: **Škarvada Jaroslav, Ing.,** UPSY FIT VUT

Datum zadání: 28.února 2006
Datum odevzdání: 22. května 2007

Tato strana je v tištěné verzi nahrazena originálním listem zadání DIP

Abstrakt

Součástí platformy FITkit je vestavěný systém (anglicky Embedded system), který slouží studentům k návrhu a realizaci nejen softwarových, ale i hardwarových projektů či celých aplikací. Pro ověření správné funkce všech komponent systému vzniká potřeba vyvíjet prostředky, které umožňují FITkit testovat v celém období jeho životního cyklu.

Tato práce se zabývá testováním FITkitu. V první části se práce věnuje seznámení s FITkitem a celou jeho architekturou. V další části práce lze najít popis terminologie a analýzu problematiky testování následovanou popisem metod a postupů pro otestování platformy FITkit. Součástí práce je i návrh testovací aplikace a popis její praktické realizace.

Klíčová slova

FITkit, FPGA, MSP430, testování, VHDL

Abstract

The FITkit platform incorporates an embedded system which enables students to create and implement complex designs not only of software projects but also of hardware projects or complete applications. So it is very important to invent and implement methods that can be used for testing the system during its whole life cycle.

This thesis is engaged in testing the FITkit. The first part of the thesis is dedicated to familiarizing with the FITkit and its whole architecture. Terminology description and analysis of the dilemmas in testing can be found in the next part of the thesis. Further follows description of methods and procedures for testing the FITkit platform. A part of the thesis is a design of a testing application and description of its practical realization.

Key words

FITkit, FPGA, MSP430, platform testing, testing, VHDL

Tomáš Filip: **Testování výukové platformy FITkit**, diplomová práce, Brno, FIT VUT v Brně, 2007

Testování výukové platformy FITkit

Odevzdáno na Fakultě informačních technologií Vysokého učení technického v Brně, dne 3. ledna 2007.

Prohlášení

Prohlašuji, že jsem diplomovou práci „Testování výukové platformy FITkit“ vypracoval samostatně pod vedením Ing. Jaroslava Škarvady

A uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Jaroslavu Škarvadovi za poskytnutou literaturu, cenné rady a připomínky. Dále bych chtěl poděkovat svému otci Jaroslavu Filipovi za pomoc s návrhem destiček pro testování sběrnic.

V Brně, 22.5. 2007

….....

Tomáš Filip

Obsah

Seznam obrázků

Úvod

Nefunkčnost obvodu může být způsobena chybou při návrhu nebo závadou vzniklou při výrobě. Problematikou chyb vzniklých při návrhu se zabývá samostatná disciplína zvaná verifikace návrhu. Na chyby vzniklé při výrobě se pak zaměřuje disciplína všeobecně známá jako diagnostika. I když bude při verifikaci obvodu zjištěno, že navržený obvod plní správně požadované funkce, není tím zajištěno, že obvod popřípadě výrobek vyrobený podle verifikovaného návrhu bude pracovat bezchybně. Během procesu výroby mohou být do obvodu zaneseny poruchy, jejichž projevem bude chybná (jiná než žádaná) funkce obvodu. Navíc se taková chybná funkce obvodu v praxi nemusí projevit ihned, pokud je zasažena taková funkce obvodu, která není aktivně využívána. Je tedy třeba vyvíjet prostředky, které umožní navržený obvod po jeho vyrobení otestovat.

Tato práce se zabývá testováním karty pojmenovanou FITkit. FITkit je univerzální výrobek. který může být použit v různých aplikacích. Hlavním zaměření FITkitu je nasazení do výuky na Fakultě informačních technologií a umožnit tak studentům, aby mohli navrhovat a prakticky realizovat nejen softwarové, ale také hardwarové projekty či dokonce celé aplikace.

Tato práce se zabývá vytvořením metodických pokynů z oblasti software a hardware umožňující testování FITkitu. Vedlejším produktem bude aplikace, která má za úkol, ověřit nejen jednotlivé komponenty FITkitu, ale i propojení sběrnic a výstupní periferie. Uživatel pomocí interaktivního ovládání bude mít možnost testovat jednotlivé části ať autonomně nebo za pomocí doplňkového hardwarového vybavení.

Diplomová práce je rozdělen na několik části. V první kapitole se čtenář seznámí s platformou FITkitu a jejími komponentami. V druhé kapitole se zavádějí základní pojmy související s testováním a diagnostikou a schopností detekce chyb. Třetí část představuje vybranou kolekci metod, algoritmů a postupů určené k otestování jednotlivých externích komponent. Ve čtvrté je navržena architektura testovací aplikace následovaná popisem její praktické realizace.

1 Platforma FITkit

Platforma FITkit obsahuje výkonný mikrokontrolér (μC) s nízkým příkonem a řadou periferií. Důležitým aspektem je také využití pokročilého reprogramovatelného hardwaru na bázi hradlových polí FPGA (anglicky Filed Programmable Gate Array) jenž lze, podobně jako software na počítači, neomezeně modifikovat pro různé účely dle potřeby - uživatel tedy nemusí vytvářet nový hardware pro každou aplikaci znovu.

K popisu vlastností hardware se na platformě FITkit používá jazyk VHDL, díky jemuž se návrh software a hardware provádí do značné míry obdobně. Veškerý návrhový software k generování programovacích dat pro FPGA, včetně rozsáhlé dokumentace je k dispozici zdarma na internetových stránkách firmy Xilinx Inc. [1]. Kompilace zdrojových kódů se provádí zcela automaticky, za pomocí systému Makefile, jenž se běžně používá při tvorbě software.

Software pro mikrokontrolér se tvoří v programovacím jazyce C a do spustitelné formy se překládá pomocí GNU překladače GCC, volně dostupného z internetových stránek komunity SourceForge. [2].

Uvnitř mikrokontrolér je umístěn kód, který umožňuje naprogramovat čip přes sériový kanál bez použití dalších externích prostředků a za pomocí volně dostupných programů anebo lze použít externí programátor, který se napojuje na JTAG (Joint Test Action Group) rozhraní mikrokontroléru. FPGA lze obdobně naprogramovat přes JTAG anebo za pomocí mikrokontroléru přes vysokorychlostní komunikaci protokolem SPI. Podrobnější popis vzájemné komunikaci a programování těchto komponent je uveden v kapitole 4.

FITkit komunikuje s vnějším okolím přes USB rozhraní, ze kterého je přímo napájen, v případě velkého proudového odběru je nutné připojit externí napájení +5V do konektoru JP8. Externí napájení je chráněno pojistkou F0 o hodnotě 1A. Z napětí +5V se dále stabilizátory získává pro většinu komponent napětí +3.3V a pomocná napájení +2.5V a +1.2V pro napájení FPGA.

1.1 FPGA

Na FITkitu lze najít FPGA, programovatelné hradlové pole XC3S50-4PG208C, firmy XilinX Inc., jedná se o produkt řady Spartan 3. Tento konkrétní zástupce obsahuje:

- 50000 logických hradel,
- 192 konfigurovatelných logických bloků (CLB) uspořádaných do matice 16x12,
- 1728 logických buněk umožňující provádět všechny možné logické operace,
- 4x násobičky 18x18 bitů,
- paměť RAM,

    - 72KBitů v BlockRAM
    - 12KBitů jako Distribuované RAM
- 2x DCM (Digital Clock Manager) - jednotka pro správu vstupního hodinového signálu (násobení, dělení frekvence, posuny, eliminace zkreslení, atd...),
- 124 I/O pinů, které mohou pracovat až ve 26 napěťových standardech.

Z blokového schématu FITkitu [3] lze vyčíst, že k čipu FPGA je připojen LCD Display, maticová klávesnice, SDRAM 8M, Video výstup, 1x UART, PS/2 Myš, PS/2 Klávesnice a I/O piny pro všeobecné použití. Tyto vstupně výstupní piny jsou sdruženy na sběrnici X, která je vyvedena na konektoru JP10. Další piny slouží ke komunikaci s již zmiňovaným převodníkem FT2232. FPGA může také komunikovat s μC pomocí SPI(Serial Peripheral Interface) anebo také pomocí společné 8bitové sběrnice, označenou jako P3M.

K dispozici je i 46-bitová sběrnice X, která je vyvedena přímo na konektor JP10. Na konektoru je navíc kromě této sběrnice i vyvedeno napájecí napětí +3.3V a +5V. Konektor JP10 je 50-pinový počítačový konektor s roztečí mezi jednotlivými piny 2.54mm.

Vstupně/výstupní zařízení jako VIDEO(VGA), UART a obě PS/2 zařízení komunikují také po sběrnici X a tuto sběrnici sdílejí. Jsou tak zapojené mezi dva I/O napěťové budiče/převodníky, které jednak budí sběrnici (zařízení uvedené výše ji dále nezatěžují) a převádějí napěťovou úroveň z +3.3V na +5V. K povolení funkčnosti všech těchto zařízení je nutné zkratovat propojku (jumper) JP6. Tato propojka přivádí nízkou logickou úroveň na povolovací pin těchto dvou I/O datových budičů/převodníků, za kterými jsou tato zařízení připojena.

Vstupní hodinový signál o frekvenci 7.3728 MHz generuje μC, signál má označení SMCLK. Dalším hodinovým signálem je signál ACLK, generuje jej opět μC. Na desce plošného spoje je vyhrazeno místo pro externí krystal U9, který je připojen k FPGA, ten však v době psání této práce nebyl osazován.

JTAG rozhraní, splňující normu IEEE 1149.1 je vyvedeno na propojce JP1. Lze využít jak k testování, kontrole chodu tak i k programování FPGA. Zapotřebí je programátor, např. Parallel Cable III/IV nebo Platform Cable USB a programovací software Impact, dostupný z balíčku [1].

1.2 LCD Display

Na FITkitu je osazován maticový LCD display od firmy DATA IMAGE Corporation s označením CM1610NR-J2. Jedná se o jednořádkový 16-znakový maticový display, obsahující řadič, plně kompatibilní s řadičem firmy Hitachi [4], který je používán ve většině dostupných LCD maticových displejů.

Display s použitím tohoto řadiče má tyto vlastnosti [5]:

- 8-bitová nebo 4-bitová obousměrná komunikace (FITkit používá 8-bitovou komunikaci),
- paměť ROM s 248 předefinovanými znaky (použitý typ používá znaky latinské a japonské abecedy),
- umístění znaků v ROM odpovídá umístění znaků v ASCII tabulce
- paměť RAM pro 8 znaků definovaných uživatelem (např. znaky s diakritikou),
- paměť RAM pro aktuální zobrazované znaky,
- ovládání displeje pomocí příkazů,
- řízení kontrastu (na FITkitu řídí kontrast potenciometrem P0) a další.

K FPGA je display zapojen přímo, kde signály LD0-LD7 jsou datové signály a signály LRW, LE a LRS jsou řídící signály. Display je napájen napětím +3.3V.

Firmware pro ovládání LCD přes FPGA pomocí SPI rozhraní je dostupné na stránkách FITkitu [6]. Firmware obsahuje jednak kódy pro FPGA tak i ukázkovou aplikaci pro μC

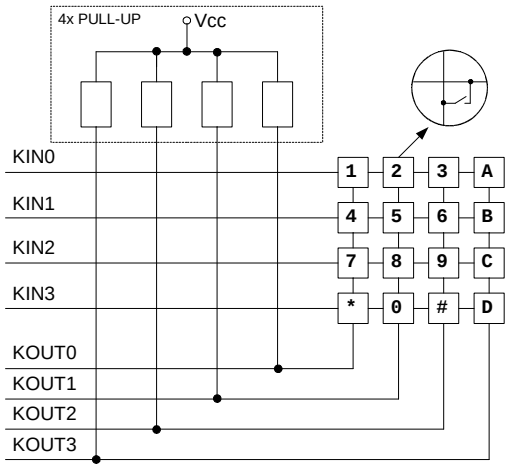
1.3 Maticová klávesnice 4x4

Použitá klávesnice na FITkitu nese označení AK-1604-A-WWB jedná se o maticové zapojení 16 kláves do čtyř řádků a čtyř sloupců.

Princip identifikace stisknutých kláves spočívá v udržování sloupcových vodičů ve vysoké logické úrovni pomocí PULL-UP rezistorů. Na jednotlivé řádkové vodiče KIN0 - KIN3 se postupně přivádí nízká logická úroveň.

Při stisknutém tlačítku na aktivním řádku se nízká logická hodnota propaguje na vodiče KOUT0 - KOUT3. Vysoká logická hodnota vytvořena PULL-UP rezistorem je snížena na nízkou přes tlačítko do budiče na pinu FPGA.

Tyto informace lze například uložit na patřičné pozice pomocného 16bitového pole, ve které je uložena informaci o stisknutých klávesách. Při stisku tlačítek vznikají zákmity (řádově jednotky ms), se kterými si musí řadič poradit.



Obrázek 1.: Schéma zapojení klávesnice

Klávesnice je připojena přímo k FPGA pomocí konektoru KL10. Zmíněné PULL-UP rezistory lze zapnout přímo v čipu FPGA, pomocí vyhrazeného slova PULLUP u požadovaných pinů v souboru ucf.

Firmware, včetně dokumentace pro FPGA k ovládání klávesnice přes SPI rozhraní je dostupné na stránkách FITkitu [7].

1.4 SDRAM

Osazená SDRAM je od firmy Hynix Semiconductor, nese označení GM72V66841ET-7K [8]. Použitím firemní literatury lze zjistit, že paměť má tyto vlastnosti:

- pracovní frekvence 100 MHz,
- 8bitová organizace paměti,
- paměť rozdělena do 4 banků (vybírají je signály BA0, BA1),
- plně synchronní přístup,
- 2 097 152 slov v jednom banku,
- 4096 obnovovacích cyklů k obnovení celé paměti,
- celková velikost 8 Mbytů.

Paměť je přímo připojena k čipu FPGA, signály jsou pojmenovány jako RA0-RA14 (adresa), RD0-RD7 (data) a řídícími signály (CS, RAS, CAS, WE, CLK, CKE, DQM). Napájena je z napětí +3.3V.

Připraveny jsou dva řadiče, které komunikují s pamětí, jeden na nízké úrovní, ovladatelný pomocí příkazů a druhý na vyšší úrovní ovladatelný přes SPI. Tyto řadiče včetně dokumentace jsou dostupné na stránkách FITkitu [9].

1.5 VGA Výstup

VGA (Video Graphics Array) je standard firmy IBM pro zobrazování digitálních informací na analogových zařízeních. Zobrazována data jsou reprezentována třemi základními barevnými složkami, ze kterých je možné vytvořit širokou škálu barev. Aby bylo možné identifikovat pozici právě zobrazeného bodu, je potřeba kromě informace o barvě přidat i informaci o pozici. Ta se získává ze dvou synchronizačních pulsů, pojmenované jako vertikální a horizontální pulsy.

Způsob zobrazování, tj. rozlišení a obnovovací frekvence jsou připojenou periferií automaticky detekovány podle délky a trvání synchronizačních pulzů. Analogový signál se na FITkitu získává pomocí jednoduchého odporového D/A převodníku. Na každou barvu jsou vyhrazeny 3bity, které tak mohou vytvořit na výstupním zařízení až 2^9= 512 různých barev.

D/A převodník je zapojen za budičem/převodníkem a tudíž VGA rozhraní je nutné aktivovat zkratováním propojky JP6. Pro VGA rozhraní jsou vyhrazeny signály sběrnice X31-X37.

Na stránkách FITkitu lze také najít dostupnou dokumentaci včetně řadiče VGA [10].

1.6 FPGA UART - sériová komunikace (RS232)

Na FITkitu je také osazen převodník úrovní RS232. Používá se zapojení běžně používaného obvodu SP232, ten v sobě obsahuje nábojovou pumpu, která generuje napětí +10V, -10V. Toto napětí se také používá jako napájecí napájení pro A/D převodníky.

Na konektoru JP5 (DSUB9 – zástrčka) lze najít výstupní signály **TXD** (PIN 3, Transmitted Data), **RTS** (PIN 7, Request To Send) a vstupní signály **RXD** (PIN 2, Received Data), **CTS** (PIN8, Clear To Send). Všechny signály jsou vyvedeny ze sběrnice X42-X45, procházejí přes budič/převodník sběrnice. I zde se musí aktivovat budič/převodník zkratováním propojky J6.

1.7 PS/2 Mouse a PS/2 Keyboard

FITkit je vybaven dvěma 6pinovými mini-DIN konektory. Dají se využít pro synchronní sériový protokol, běžně používaný u PS/2 myší a klávesnic. Na konektoru je vyvedeny hodinový a datový signál a napájení +5V pro napájení výstupního zařízení.

PS/2 protokol se vyznačuje nízko rychlostním sériovým rozhraním master-slave. Umožňuje point-to-point komunikaci mezi koncovým zařízeními(myš, klávesnice). Koncové zařízení se označuje jako master. PC, μC je obvykle považován za slave zařízení. Hodinový kmitočet řídí koncové zařízení. Samostatný sériový přenos jednoho byte je zahájen start bitem, poté následuje synchronní přenos osmi bitů následovaných paritním bitem a přenos byte je ukončen stop bitem.

Datové a hodinové signály obou konektorů jsou připojeny na sběrnici X38-X40 přes budič/převodník napěťových úrovní. K povolení chodu je zapotřebí aktivovat budič/převodník zkratováním propojky J6.

1.8 AFBUS sběrnice

AFBUS je 12bitová sběrnice propojující FPGA s převodníkem USB/UART – FT2232C, který může pracovat dle konfigurace v různých módech jako:

- Plný UART,
- FIFO registr,
- režim Host Bus μC Emulation (8051/8048),
- MPSSE (Multi Protocol Synchronous Serial Engine),
- Synchronní/Asynchronní paralelní port, atd.

Konfigurace obvodu (VID, PID, sériové číslo, atd.) a nastavení aktuálního módu obou kanálu je uložena v malé externí EEPROM, která na FITkitu není osazována. Je však na ní vyhrazeno místo v patici U1. Osazuje se pamětí firmy Microchip s označením 93LC66 nebo 93C66. V případě, že paměť osazena není, obvod funguje autonomně. Oba kanály jsou nastaveny do módu UART.

Programování EEPROM, tj. nastavení módu a konfigurace lze provést přes USB komunikaci programem dodávaným výrobcem obvodu.

1.9 µC - MSP430F168

Osazovaný mikrokontrolér MSP430F168 je vyroben firmou Texas Instruments, Inc. Jedná se nízko příkonový mikrokontrolér, který se vyznačuje těmito vlastnostmi [11]:

- 16-bitová RISC architektura,
- 48kB Programová FLASH paměť,
- 2 kB paměť RAM,
- 2x 16 Bitové časovače,
- 48 I/O pinů pro všeobecné použití,
- 2x Asynchronní/ Synchronní UART,
- 2x 12-Bit D/A převodník,
- 8-kanálový 12-bitový A/D ,
- SPI a I^2C™ hardwarová podpora na UART0,
- JTAG rozhraní, Bootloader, a další.

Vstupně výstupní porty jsou označeny jako P1-P6, jim přísluší označení sběrnic P1M-P6M. Většina signálů z těchto sběrnic je vyvedena na konektor JP9, což je 40pinový konektor s roztečí 2.54mm. Na tomto konektoru je také vyvedeno napájecí napětí +3.3V a +5V.

Port P5 se používá pro SPI komunikaci mezi FPGA a FLASH pamětí dále na programování FPGA a vysílání hodinových signálů SMCLK a ACLK.

Programování µC lze provést dvěma způsoby, první přes JTAG rozhraní, připojeného přes konektor JP11 nebo přes USB/UART za použitím rutin označené jako Bootloader. Bootloader je krátký programový kód, který umožňuje přijímat data přes sériovou linku a zapisovat je do vnitřní FLASH paměti. Tento kód je výrobcem napevno uložen v obvodu µC.

Propojky J8 a J9 slouží k povolení programování µC. Nezkratované umožňují spustit program přímo z paměti mikrokontroléru. Zkratovaná propojka J17 přivádí referenční napětí pro převodníky A/D na mikrokontrolér. Propojky J11, J12 připojují signály RXD, TXD z USB na port P3.

Oba D/A převodníky z µC jsou připojeny přes propojovací konektory (J15-J16) na vstup stereofonního operačního zesilovače. Výstupy jsou vyvedeny na stereofonní konektor JP7, který je typu 3.5" Jack.

Na FITkitu lze najít konektor JP6, který je také typu stereofonní Jack 3.5", výstupy tohoto konektoru jsou přivedeny na vstupy dvou operačních zesilovačů. Jejich výstupy lze připojit na sběrnici

P6M, zkratováním konektory J13-J14. Sběrnice P6M obsahuje osm kanálů A/D převodníku a dva D/A převodníky. Nejvyšší dva bity portu P6, P6.6 a P6.7 se dají využít buď jako A/D nebo D/A převodník. Celá sběrnice je vyvedena také na konektor JP9.

1.10 Serial Flash Memory

Tato 2-megabitová DataFlash™ paměť od firmy ATMEL nese označení AT45DB021B. Komunikačním protokolem je SPI a paměť je zapojena na sběrnici SPI mezi μC a FPGA. Obě zařízení mohou s pamětí komunikovat. Mezi další vlastnosti paměti patří:

- 1024 stránek pamětí,
- 264 bytů na jednu stránku,
- Dvě 264-bytové SRAM jako vyrovnávací paměti pro uchování dat při programování bloku,
- blokové nebo stránkové mazání,
- nízký příkon,
- pouze jedno napájení pro všechny činnosti a další.

Paměť FLASH jsou svojí základní strukturou a principem podobné pamětem EEPROM, rozdíl mezi nimi je především v tom, že u paměti FLASH se maže celý blok najednou u EEPROM se maže každá buňka zvlášť. Paměťová buňka je jeden speciální MOSFET tranzistor, který má svůj gate rozdělen na dvě části. Paměťová buňka udržuje přes svoji speciální izolační vrstvu náboj, který ovlivňuje chování tranzistoru. Tato izolační vrstva však časem (zápisy) degraduje a buňka se nedá použít.

Samotná paměť je rozdělena na sektory(4), bloky(127) a stránky(1023). Toto rozdělení umožňuje lepší flexibilitu při operaci s pamětí. Všechny operace tak vypadají jako kdyby pracovaly pouze se stránkou. Stránku lze například načítat do vyrovnávací paměti, modifikovat ji a zpětně zapsat do FLASH paměti. Kompletní popis operací lze najít v dokumentaci k obvodu [12].

2 Úvod do testování

Kapitola vysvětluje základní terminologii zabývající se testováním a diagnostikou. V závěru kapitoly jsou uvedeny problémy, se kterými se lze setkat při testování FPGA a μC.

2.1 Terminologie

2.1.1 Technika DFT (Design for Testability)

Technika DFT jsou metody návrhů pro lepší testovatelnost. Tato disciplína prolíná všemi etapami návrhů od systémového návrhu až po testování vyrobených produktů.

Jeden z DFT návrhů je založen na vložení kontrolních a řídicích bodů vyhrazeným prvkům uvnitř čipu. Přináší výbornou schopnost detekce chyb na úkor komplikovanějšího hardware, možným snížením výkonu a zvýšeným odběrem tepla. Nepoužitím této DFT techniky sice odpadnou výše zmiňované problémy, avšak pokrytí detekčních schopnosti chyb je menší.

Další techniky popisují publikace [13],[14], zabývají se metodami testování, generováním testovacích vektorů, testováním propojení (Boundary Scan), atd.

2.1.2 Externí test x interní test (BIST, Built-In-Self-Test)

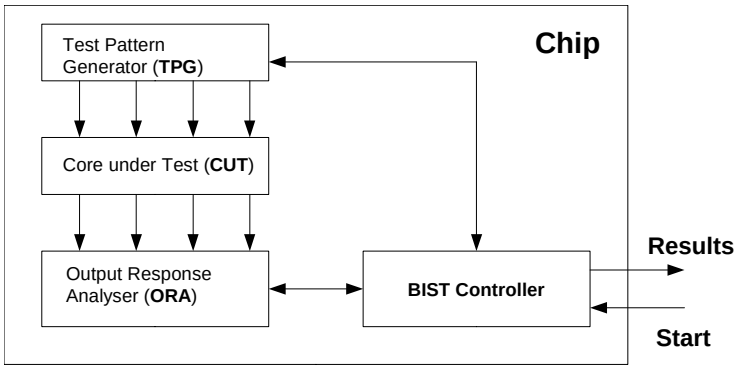
Externí test znamená vkládání a vyhodnocení testovacích vektorů mimo testovanou jednotku. BIST je jedna z technik DFT, jak vysvětluje [14], která test, tj. generování testovacích vektorů, testování a vyhodnocení výsledků, provádí přímo uvnitř testovaného hardware.

U externího testu se využívá ATE (Automatic Test Equipment) jedná se automatické testovací zařízení [13]. Moderní ATE jsou výkonné počítače u kterých se testovací program píše ve vyšším programovacím jazyku. Dále je zapotřebí vytvořit testovací vektory a korektní odezvy na ně. Zařízení ATE je obvykle velmi drahé a v určitých případech se využívá technika BIST.

Další důvod zavedení BIST je zvětšující se kapacita dat, která musí být přenesena mezi zařízením (DUT, Device Under Test) a samotným testerem. Externí testování využívající pomalé komunikační sběrnice vždy prodlužuje samotné testování.

BIST , jak ukazuje obrázek 2., vyžaduje pouze vstupní signály pro spuštění testu a vyhodnocení zda test proběhl v pořádku či nikoliv. BIST model předpokládá, že řídící část obvodu, tj. ta část která není testována, je bez chyb.

- TPG – je jednotka na generování testovacích vektorů,
- CUT/DUT – část čipu, jednotky , která je testována,
- ORA – je jednotka pro vyhodnocení testovacích vektorů.



Obrázek 2.: Základní schéma BIST architektury

2.1.3 Funkční x strukturální testování

Funkční testování je založeno na ověření funkce dané jednotky. To nevyžaduje podrobné informace o nízko-úrovňovém zapojení jednotlivých prvků jednotky (jako je např. RTL schéma, Register Transfer Level). Což sice sebou nese nevýhodu v nemožnosti identifikovat strukturální chyby. Pokud však potřebujeme ověřit funkčnost jednotky, je návrh testu jednodušší. Koncept přináší replikaci testovacích funkcí i na jiné podobné jednotky a tím i nižší cenu vývoje testu. Strukturální test je opakem funkčního testu, zde je již zapotřebí podrobné schéma a metoda přináší lepší pokrytí možných výskytu chyb.

2.1.4 Statické x dynamické testování

Metody založené na statické testování se zabývají detekcí chyb v kombinační logice. Statické testy kontrolují výstupní hodnoty až v době, kdy již skončily všechny přechodné jevy, případně byla přečtena informace z paměti. Pokud se výstupní hodnoty sledují i v době, kdy probíhají přechodné jevy, jde o testy dynamické. Statické testování odhaluje chyby známé jako [14]:

- stuck-at faults (spojení, které trvale generuje určitý logický stav),
- bridging faults (zkraty nebo propojení mezi dvěma a více skupinami signálů),
- opens faults (chybějící propojení),

Dynamické testování odhaluje chyby jako jsou:

- delay faults (nesplnění časových specifikací),
- parametric faults (neplnění předepsaného chování výrobcem vlivem prostředí, napájení, atd.).

Obě metody vyžadují informace o vnitřním zapojení testovaného obvodu. Dynamické testování obvykle trvá mnohem delší dobu a je více náročnější než testování statické.

2.1.5 Pseudonáhodné x deterministické testování

Nalezení chyb závisí na testovací technice, množství a typu testovacích vektorů na testovanou jednotku. Pseudonáhodné testování je založeno na vygenerování pseudonáhodných vektorů. Ty se spolu s operandy, které se explicitně negenerují, používají k testování jednotky. Technika vyžaduje menší znalosti o testované jednotce. Bohužel k dosazení optimálního pokrytí k nalezení chyb je metoda časově velmi náročná.

Deterministické testování je založeno na konkrétním testování určených částí testované jednotky (jako jsou ALU, násobičky, děličky, posuvné registry, paměti...). Obvykle je s deterministickým testováním spojena sada různých testů, které tyto jednotlivé části testují.

2.1.6 Testování x Diagnostika

Diagnostika je vědní disciplína, zabývající se problematikou zjišťování technického stavu výrobků, metod rychlého zjištění existence poruch, určení místa jejich vzniku a způsob jejich eliminace. Naproti tomu testování je aplikace těchto diagnostických metod.

2.2 Problematika testování FPGA

Dnešní FPGA svojí architekturou už vypadají spíše jak množina systémů umístěných na jednom čipu (tzv. SOC - System on a Chip) než samotné hradlové pole sestaveno z logických hradel. Jsou to vetšinou kombinace různých signálových IO, vysílačů na GHz frekvencích a blokových pamětí. FPGA obsahují tisíce konfigurovatelné logiky. FPGA je tak nasazována i do míst, která byla spíše doménou ASIC (Application-Specific Integrated Circuit) čipů.

Se zvyšující hustotou integrace se zvyšují i nároky na testování. Komplikací testování takového čipu způsobuje fakt, že oproti jiným podobným obvodům jako je ASIC, je FPGA určeno pro všeobecné použití. Pro jedno zařízení může existovat stovky různých designů a stejné množství testovacích produktů. To není ani praktické a ani ekonomicky výhodné.

Pokud bychom se dívali na FPGA jako na VLSI(Very Large-Scale Integration) čip, je zapotřebí dosáhnout řízení, kontrolu a dohled nad každým uzlem obvodu [14]. U FPGA jenž je konfigurovatelné, narážíme na potřebu měnit vstupy všech dostupných prvků uvnitř FPGA a získávat odezvy z jejich všech výstupů. Celkový čas testování odvíjející se od počtu testovacích vektorů je úzce spojen s velikosti čipu. Přirozeně pro velké programovatelné pole s větším počtem zdrojů bude tento čas delší.

Chyby, které se však mohou objevit u těchto FPGA jsou stejné jak u VLSI čipů (chyby jsou uvedeny v kapitole 2.1.4 - x dynamické testování).

Spartan 3 – FPGA, jenž je použito na FITkitu je velmi uniformní a homogenní zařízení. Lze jej rozdělit do těchto částí:

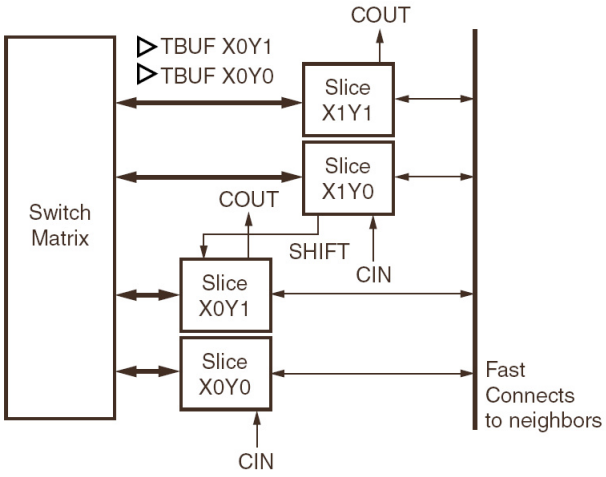
- Konfigurovatelné logické bloky (Configurable Logic Blocks - CLBs) obsahují RAM-based Look-Up Tables pro vytvoření logických a paměťových prvků, které mohou být použity například jako klopné obvody.
- Vstupně/Výstupních Bloky (Input/Output Blocks - IOBs), které řídí tok dat mezi vstupně výstupními piny a interní logikou na čipu. Každý IOB prvek může být uveden do třetího stavu (de facto je odpojen od výstupního pinu) a umožňuje obousměrnou komunikaci. Ten může fungovat i v režimu DDR (Double Data-Rate). DCI (Digitally Controlled Impedance) je vlastnost, která zjednodušuje výsledný návrh desky.
- Blokové paměti (Block RAM) nabízejí datový prostor v 18-Kbit blocích, mohou mít až dva nezávislé přístupové porty.
- Násobičky pracují na vstupu s dvěma 18-bitovými čísly.
- DCM (Digital Clock Manager) je blok, který provádí automatickou kalibraci, distribuci a úpravy (zpoždění, násobení, dělení, fázové posuvy) hodinových signálů.

2.2.1 CLB - Konfigurovatelný logický blok

CLB bloky jsou sestaveny do matice a používají se pro vytváření kombinační a sekvenční logiky. Každý CLB je svázán do přepojovací matice, která má přístup do hlavní přepojovací sítě. Jak ukazuje obrázek 3., každý CLB obsahuje 4 stejné části (Slice) s rychlým připojením mezi sebou.

Mezi důležité vlastnosti Slice patří:

- Každý Slice obsahuje dva 4-vstupní funkční generátory.
- Obsahuje Carry logiku, aritmeticko-logické hradla.
- Kombinacemi jej lze využít jako čtyř až osmi vstupní multiplexor.
- Funkční generátor je programovatelná LUT (Look-Up-Table), 16bitová distribuovaná RAM nebo 16-bitový posuvný registr



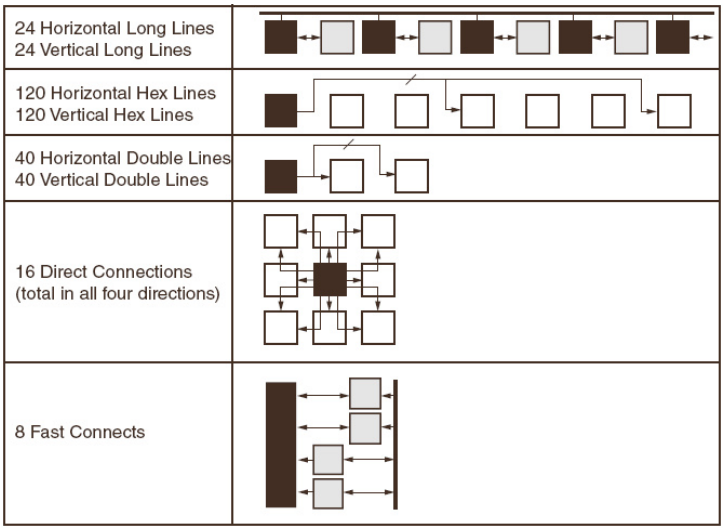
Obrázek 3.: Schéma CLB buňky

Čtyři nezávislé vstupy dvou Slice umožňují vytvořit libovolnou Booleovu funkci čtyř vstupů. Časové zpoždění průchodem Slice (propagation delay) je nezávislá na použité funkci. Lze vytvářet celou škálu klopných obvodů bez/s reset signálu, reagující na hodinový signál, s před nastavením, atd. Distribuované paměti lze sestavit jako jedno i dvou portové.

2.2.2 Hierarchical Global Routing Resources

Většina signálu jsou propojena skrz globální propojovací zdroje, které se nacházejí v horizontálních a vertikálních kanálech mezi každou přepínací matici. Jedná se o různé techniky propojení CLB uvnitř čipu tak, aby nedocházelo k velkému zpoždění při průchodu signálů.

Některé FPGA mají ještě navíc speciální rychlejší dedikované sběrnice na rozvod hodinových signálů.



Obrázek 4.: Různé propojení CLB

2.3 Problematika testování µC

Integrované obvody dnešních µC jsou sestaveny z milionů logických hradel a nejrůznějších paměťových buněk. Mohou obsahovat také již před vytvořené entity nazývané IP jádra (Intellectual Property Core) jenž ve výsledku vytvářejí z µC obvod známý pod označením SOC (System-on-Chip). Základem takového µC je však stále základní procesorová architektura (ať CISC nebo RISC) sestavena z procesorového jádra (ALU, instrukčního dekodéru, zásobníku, registrového pole, čítačů adres, atd.), periferních zařízení (vstupně/výstupní porty) a pomocných jednotek (přerušovací jednotka, časovače, atd.).

Samotné testování µC spočívá v rozdělení a návrhu testovacích úloh tak, aby pokryly jednotlivé části µC, tj. celou množinu IP, pamětí, ALU, časovačů, atd.. Problémem, se kterými se při testování setkáme jsou části procesoru, které jsou uživateli skryty a bez použití DFT techniky je lze špatně testovat. Jedná se převážně o použití jednotek zřetězeného zpracování instrukcí, jednotek pro zřetězení registrů, používáním predikci skoků, atd..

Dalším problémem je použití překladačů vyšších jazyků pro překlad testovacích programů do jazyka symbolických instrukcí (JSI), různé prvky optimalizátoru mohou optimalizovat kód do podoby, která již neodpovídá představě toho co chceme testovat.

3 Návrh metod k testování FITkitu

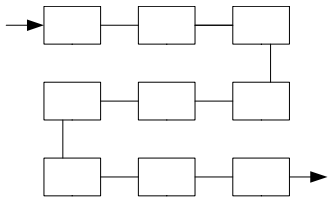
Kapitola Metody k testování FITkitu představuje kolekci metod a postupů, které budou aplikovány na platformu FITkit. Jednotlivé podkapitoly se zabývají funkčními částmi platformy, jenž byly představeny v kapitole 1.

3.1 BIST pro FPGA

U FPGA není vždy možné plně postihnout testem všechny logické a propojovací zdroje v jedné testovací konfiguraci. Testování propojovacích spojů může být fyzicky a koncepčně odděleno od testování logiky. Testovací metody uvedeny v [15], [16] jsou založené na technice BIST (Built-In-Self-Test).

Článek [15] se zabývá celkovým testováním FPGA. Samotné testování rozděluje na dvě části.

V první části navrhuje článek vytvořit ze základních prvků (např. registrů) řetěz, kdy výstup předchozího prvku je spojen se vstupem dalšího prvku. Test začíná tak, že všechny buňky se inicializují na logickou hodnotu 0. Na první prvek se vloží logická jednička, která se každou hranou hodinového signálu posouvá po tomto řetězu.



Obrázek 5.: Registrový řetěz

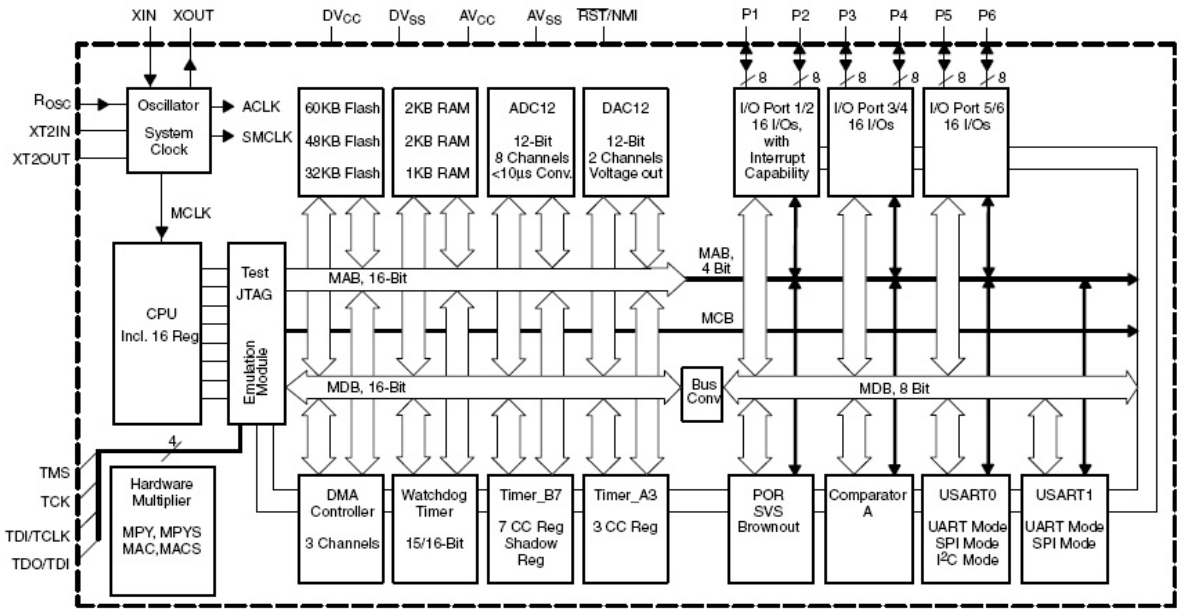
Účelem testování je ověření, zda nejsou některé CLB buňky propojeny, případně propojení mezi těmito buňkami chybí. Zkratovaný, resp. chybějící spoj uvnitř FPGA se projeví dřívějším objevením informace na výstupu, resp. na výstup se nemusí vůbec objevit. Případné kombinace zkratů se projeví výskytem signálů v době, který neodpovídá době průchodu přes všechny registry.

Druhá část testování se zabývá deterministickým testováním jednotek jako jsou BlockRAM, násobičky, DCM, atd., kde se podobně jak u klopných obvodů vytvoří síť těchto prvků, na které se aplikují algoritmy pro jejich otestování. V případě RAM lze aplikovat na testování specializované algoritmy jako je například IFA-13, March-G algoritmus a další, popsané v [13].

Článek [16] podává pouze návod na vyhledání a testování LUT. Hlavní zaměřením článku je konfigurace propojení mezi jednotlivými LUT. Popisuje tzv. vertikální a horizontální propojení LUT, princip testování je dále podobný jak v předchozím článku.

3.2 BIST pro µC MSP430F168

Stránky výrobce tohoto µC, firma Texas Instruments nabízí dokument [17], který poskytuje celou škálu testovacích programů (tzv. Benchmark testů), které slouží pro ověření různých části tohoto mikroprocesoru.



Obrázek 6.: Blokové schéma MSP430F168

Původní záměr uvedeného dokumentu slouží k porovnání výkonů procesorů, ale po menších úpravách, jako je sjednocení do jednoho celku a přidání BIST kontroléru mohou být principy použity k otestování µC. Integrací by měl vzniknout BIST produkt, v souladu s publikací [18], která se zabývá vývojem testů, spuštěných uvnitř procesoru.

Test je aplikován na jádro samotného procesoru, ALU (i operace s plovoucí řadovou čárkou), registrový přenos. Operace spadají do těchto oblastí: 8-bitová aritmetika, 16-bitová aritmetika, 32-bitová aritmetika, operace s plovoucí čárkou, 8-bitový příkaz case, 16-bitový příkaz case, násobení matic a aplikace FIR filtru).

DMA kontrolér lze otestovat při přenosu dat z AD převodníku do paměti RAM. Na ostatní prvky mikrokontroléru je potřeba vyhotovit specializované testy.

Do poslední skupiny k otestování patří vstupně/výstupní porty a UART, k otestování této skupiny je zapotřebí externí tester nebo speciální vybavení.

3.3 (D)RAM

V publikaci [13] a [14] je poměrně podrobně popsána technika testování RAM pamětí. Mezi známé chyby, které se objevují u pamětí patří [14]:

- Stuck-at faults (**SAF**) – trvalé připojení na hodnotu log.1 (SAF1) nebo log.0 (SAF0).
- Stuck-opens faults (**SOF**) – znamená chybu kde mezi buňkami je přerušený vodič.
- Transition faults (**TF**) – speciální případ SAF. Paměťová buňka selže pokud je proveden zápis ↑ (zápis 1 do buňky, která obsahovala 0) nebo ↓ (zápis 0 do buňky, která obsahovala 1).
- Address decoder faults (**AF**) – jsou známy čtyři možné případy selhání adresového dekodéru.
- Coupling faults (**CF**) – jedná se o různé spojení dvou buněk, kde zápisem do buněk se navzájem ovlivňují uložené data. Cfid, Cfin jsou speciální případy CF chyb.
- Neighbourhood pattern sensitive faults (**NPSF**) – zvláštní případ ovlivňování okolními buňkami. V určitých případech může dojít kombinací uložených informací v okolních buňkách k ovlivňování informace v prostřední buňce.

Mezi známé algoritmy pro testování pamětí patří skupina Marching algoritmů. Vyznačují se lineární složitostí, která je významná pro velké kapacity pamětí a dobrou lokalizací chyb. Neumějí však lokalizovat NPSF problémy. Na detekci těchto problémů se používají vektory vygenerované algoritmy tvořící Eulerovy sekvence.

March test je konečná sekvence zápisů logické 0/1 a čtení logické 0/1 (mající zápis W0,W1,R0,R1) aplikované po sobě na paměťovou buňku před přechodem na další buňku. Inkrementace adresy další buňky je určena znakem ↑, dekrementace adresy další buňky je určena znakem ↓, pokud nezáleží na výběru směru změny adresy používá se znak ↕. Další informace, která se používá při definování operací je časová pauza mezi jednotlivými operacemi. Ta se používá pro kontrolu zda daná buňka uchovává svůj obsah.

V tabulce 1. jsou pro srovnání uvedeny tři algoritmy. V tabulce lze najít jejich časovou náročnost, a schopnost lokalizace chyb. Uvedené algoritmy je výhodné nasadit na paměti RAM vyskytující se na FITkitu.

Pro paměti FLASH, které jsou v porovnání s pamětmi RAM omezeny maximálním počtem zápisů je výhodnější nasadit algoritmus March-X. Algoritmus se v porovnání s ostatními vyznačuje menší časovou náročnost a s tím související i menší potřebu zápisu a čtení do buněk než ostatní algoritmy. Dosahuje však stále dobrých lokalizačních vlastnosti. Další důvod k výběru tohoto algoritmu je související menší náročnost na komunikaci mezi testovaným prvkem a testerem.

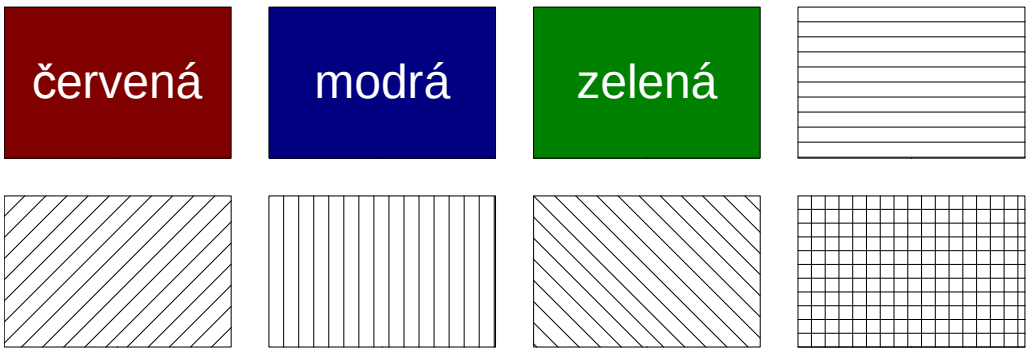
	IFA-9 (12N+Pause)	**IFA-13** (16N+Pause)	**March X** (6N)
1	↕ W0	↕ W0	↕ W0
2	↑R0, W1	↑R0, W1,R1	↑R0, W1
3	↑R1,W0	↑R1,W0, R0	↓R1,W0
4	↓R0,W1	↓R0,W1,R1	↕ R0
5	↓R0,W1	↓R1,W0,R0	
6	pause	pause	
7	↑ R0, W1	↑ R0, W1	
8	Pause	Pause	
9	↑ R1	↑ R1	
Pokrytí:	SAF, TF, AF, CFid, test na uchování dat	SAF, TF, AF, SOF ,CF pro bity v jednom slově, test na uchování dat	SAF, AF, TF ,CFin

Tabulka 1.: algoritmy na testování pamětí

3.4 Testování VGA portu a přesnost hodinového kmitočtu

Jeden z jednoduchých aplikovatelných testů VGA portu, je zobrazení vygenerované množiny obrazců v rozlišení 640x480 nebo 800x600 pixelů, u kterých uživatel bude opticky na výstupním VGA monitoru kontrolovat jejich kvalitu.. Pro identifikaci základních barev je výhodné doplnit obrazovky identifikační textem: „červená“, „modrá“, „zelená“. Ohodnocení základních barev převodníku je čistě subjektivní záležitost, ovlivněná nejen zrakem pozorovatele, ale i kvalitou monitoru, ke kterému se FITkit bude připojovat, je výhodné rozšířit test VGA portu zobrazením jednotlivých barevných obrazců aspoň ve třech odstínech.

Dalším rozšířením testu VGA portu může být zobrazení obrazců sestavených z různě orientovaných čar, zobrazených na obrázku 7.

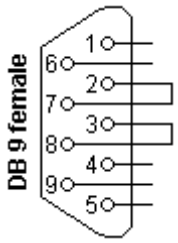


Obrázek 7.: Obrazce pro testování VGA portu

Tyto obrazce zároveň ověří i přesnost hodinového kmitočtu. Pokud vstupní hodinový signál nebude nabývat požadované přesnosti, budou zobrazované obrazce různě deformované, případně nebudou vůbec rozeznatelné.

3.5 Testování FPGA - UART (JP5)

Pro automatizované testování sériové rozhraní dostupného z FPGA je vhodné použít zapojení známé pod označením LoopBack. K otestování je zapotřebí jeden konektoru, u kterého jsou navzájem propojeny výstupní a vstupní piny. Toto zapojení nevyžaduje kromě konektoru žádné další externí prostředky. Jedná se o konektor DSUB-9.



Obrázek 8.: LoopBack

Konektor by měl mít propojeny piny 2 a 7, 3 a 8 (viz zapojení na obrázku 8). Tento konektor se bude zasunovat do konektoru JP5.

K otestování rozhraní je zapotřebí vytvořit jednoduchý kontrolér, dostupný například přes rozhraní SPI, který bude umožňovat ovládat výstupní piny portu a vstupní piny vyčítat.

Testování se pak zredukuje na ověření, že informace vyslaná po výstupních vodičích je správně identifikována na patřičných vstupních datových linkách konektoru.

3.6 Testování PS/2 vstupů

Zapojení konektoru PS2 odpovídá zapojení standardních PS/2 klávesnic (varianta 104-kláves) a PS/2 myši. K otestování bude zapotřebí externí hardware, pro snazší dostupnost je možné použít právě standardní klávesnici či myš. Na straně software je zapotřebí vytvořit podporu pro identifikaci a dekódování přijatých znaků z obou zařízení a jejich rozumnou formu zobrazení uživateli.

Protokol PS/2 probíhá obousměrně. Připojenými zařízeními (klávesnice, myš) se jednak otestuje ovládání výstupních vodičů, tak i čtení hodnot signálů. Poslední část k ověření je ověření napájecího napětí na konektoru JP3/JP4, připojena zařízení však požadují externí napájení z konektoru, bez kterého nemohou pracovat.

3.7 Testování maticové klávesnice 4x4

Maticová klávesnice 4x4 připojená pomocí šroubků k FITkitu může být využit k jednomu z uživatelských prvků k ovládání aplikace vykonávající testy na FITkitu.

V případě poruchy klávesnice, by však měla existovat možnost deaktivovat funkci pro ovládání aplikace přes tuto klávesnici (například přes spuštěný terminál na lokálním počítači).

K samotnému testu klávesnice, je zapotřebí interakce uživatele, který ověří stisk všech kláves a jejich odpovídající odezvu.

3.8 LCD Display

LCD display může podobně jak klávesnice sloužit jako jeden z prvků k intuitivnějšímu ovládání spuštěné aplikace.

Součástí testů by mělo spočívat v otestování všech segmentů zobrazovacího zařízení, včetně uložení uživatelských fontů do vnitřní RAM paměti. Uživatelské fonty by mohly obsahovat znaky národní abecedy nebo sadu rozlišitelných symbolů, pro rozlišení správné funkčnosti LCD.

3.9 A/D, D/A z µC (JP6/JP7)

Pro ověření funkčnosti A/D, D/A převodníku a celé signálové cesty z mikroprocesoru MSP430 je možné použít propojovací kabeláž s dvěma koncovkami typu stereofonní JACK 3.5“, u kterého budou všechny tři piny konektoru navzájem propojeny.

Toto vzájemné propojení převodníků vede k automatizovanému testu, jehož hlavní myšlenka spočívá ve vygenerování napětí v rozsahu 12-bitů pomocí D/A převodníku a snímání tohoto napětí zpětně A/D převodníkem do digitální podoby. Výsledkem mohou být grafy odchylek, požadovaných a naměřených hodnot mezi těmito převodníky.

3.10 Testování P3M sběrnice

Na testování společné obousměrné sběrnice P3M mezi µC a FPGA je nejvhodnější použít algoritmus universal test set. Algoritmus se vyznačuje dobrou rychlosti a umí identifikovat všechny rozpoznatelné chyby.

Podstatou tohoto algoritmu je spojení algoritmů rotující nuly a rotující jedničky. Více o lokalizaci chyb na vodičích lze najít v bakalářské práci [19].

$$S = \left[\begin{array}{c|cccc|c|cccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}\right]$$

Obrázek 9.: Testovací matice algoritmu 'Universal Test Set'

Principem testování je vkládání testovacích vektorů na vysílací straně testovacích vodičů, které mají tvar uvedený na obrázku 9. Přijaté data na druhé straně zařízení se ukládají do pomocné paměti, popřípadě ihned porovnají se správnými údaji. Výsledkem testu je vektor špatných vodičů. Role vysílač/přijímač se u obousměrné sběrnice po provedení celé sady testu vymění.

Běžně se v aplikacích používá jeden signál sběrnice P3M pro přivedení signálu RESET generovaného z mikrokontroléru. Test této sběrnice by měl být zaměřen na testování všech vodičů. Nezbývá, než použít dvě různé konfigurace, které budou využívat různé umístění resetovácího signálu pro design uvnitř FPGA.

3.11 Testování AFBUS sběrnice

Pro testování AFBUS sběrnice mezi FPGA a FT2232C bude zapotřebí osadit FITkit sériovou pamětí 93C66/93LC66 a nakonfigurovat kanál A převodníku FT2232C jako FIFO registr.

Získá se tak 12bitová obousměrná paralelní sběrnice, která bude z jedné strany připojena přes USB kanál k PC a z druhé strany bude připojena k FPGA.

Testování může být založeno na zapojení, kdy přijatá data budou přijímaný třetí stranou nebo se budou vracet (zpětně vysílat) zdroji dat, kde budou vyhodnocena.

3.12 Testování sběrnic na I/O konektorech JP9 a JP10

Sběrnice X a většina portu z µC jsou vyvedeny na konektory JP9, JP10. Tyto konektory mají sice jiné mechanické rozměry, ale testování obou konektoru bude probíhat podle stejného schématu. Pro oba konektory budou navrženy desky plošného spoje. Destičky budou osazeny konektorem a pomocnými součástkami pro ověření funkčnosti.

U obou sběrnic bude testování založeno na optické kontrole signalizačních prvků (LED), které budou signalizovat stav logické jedničky a stav logické nuly na pinech výstupního konektoru. Signalizace logických hodnot nechť je pro přehlednost rozlišena barvou. Nastavení vodiče do hodnoty logické jedničky nechť je zelenou barvou a výskyt logické nuly barvou červenou.

Destičky budou dále vybaveny signalizaci výskytu napájecího napětí +5V. Napájení 3.3V bude použito k napájení signalizačních prvků.

V příloze C. a D. je uvedeno schéma zapojení a obrázek desky plošného spoje. Originální zdrojové soubory lze najít přiložené na CD v adresáři `výroba`. Destičky byly vytvořeny v programu Eagle 5.11 (http://www.cadsoft.de).

3.13 Testování externího oscilátorů U9 a hodinového signálu ACLK

Externí hodinové signály jsou přivedeny buď z oscilátoru U9 nebo z mikrokontroléru z výstupu hodinového signálu ACLK. Oscilátor U9 není standardně na FITkitu osazován, ale test spočívá na stejném principu jak pro hodinový signál.

Test je založen na čítači, reagující na (čítající) hodinový signál mezi dvěma mezi známými časy. Lze tak snadno ověřit, že krystal nebo vstupní hodinový signál není přítomen, zároveň lze i hrubě ověřit frekvenci. V případě, že oscilátor neběží nebo je signálová cesta vstupních hodin přerušena je vypočtená hodnota frekvence nulová.

4 Architektura testovací aplikace

Platforma FITkit umožňuje umístit řídící jádro testů celkem na tři místa. Řídícím jádrem je myšlena aplikace pro řízení testování a vizualizaci výsledků testů, které budou prezentovány uživateli.

Testovací aplikace může být rozdělena na části, které budou umístěny na programovatelné prvky jako je FPGA a mikrokontrolér. Tyto autonomní části budou nezávisle testovat své připojené periferie BIST modelem (2.1.2), výsledky testování tak mohou být uživateli zaslány na terminál přes sériový kanál.

Řešením se může i uplatnit systém master-slave, kde stejně jako v předchozím případě je testovací aplikace rozdělena na menší části (slaves), umístěné na programovatelných prvcích (FPGA, mikrokontrolér), zde však jednotlivé části spolu komunikují, slaves zpřístupňují periferie a testování je prováděno hlavním řídícím orgán (master), který řídí a vyhodnocuje testování.

Na začátku kapitoly je provedena analýza všeobecných uživatelských požadavků na testovací aplikaci, dále následuje analýza komunikačních prostředků vyskytující se na platformě FITkit, které mohou sloužit k propojení a komunikaci jednotlivých komponent systému. Shrnuty jsou i možnosti programování obou čipů.

Stěžejní částí kapitoly je představení a návrh architektury testovací aplikace a jejích jednotlivých částí. V závěru kapitoly je provedena analýza umístění aplikace, dopad a schopnost identifikace chyb.

4.1 Všeobecné požadavky na testovací aplikaci

Mezi hlavní uživatelské požadavky a charakteristiky testovací aplikace patří:

- test všech částí systému,
- přístup ke všem komponentám testovaného systému,
- malá komunikační režie mezi testovacími objekty a testovacím jádrem,
- rychlost testování,
- a srozumitelné výsledky testování.

Požadavek testu všech částí systému naráží na skutečnost, kdy je tester a vyhodnocovací objekt umístěn v testovacím objektu, ztíží se tak nebo úplně znemožní schopnost lokace a detekce všech chyb. Oblast té části testovaného bloku, kde je umístěna testovací aplikace, je komplikovanější testem ověřit. Řešení vede spíše k vícenásobným testujícím designům.

Přístup ke všem komponentám testovaného systému usnadňuje celkový návrh testovací aplikace, může být integrována jako celek, což snižuje nároky na komunikaci mezi aplikací, zjednodušuje celkový návrh a zkracuje dobu testování, způsobenou komunikační režií při opakovaném

programování různých části aplikace.

Požadavek malé komunikační režie mezi testovacími objekty a testovacím jádrem úzce souvisí s dalším požadavkem, kterým je rychlost testování. Malou komunikační režii mezi testovanými objekty a vysokou rychlost testování řeší technika BIST (uvedená v 2.1.2), kde se komunikace s testovaným jádrem a testovanou komponentou zredukuje na spuštění, ukončení testu a vyčtení výsledků. Na druhou stranu, mezi nevýhody se může řadit potenciálně složitější návrh BIST a umístění části testovací aplikace v testovaném objektu.

Srozumitelné výsledky o průběhu testování se odrážejí v potřebě a množství informací, které se poskytují uživateli. Informace může být jednoduchá, dvoustavová. Například může být signalizována nebo zobrazena ve tvaru „V pořádku“ nebo “nalezena chyba“. Pro lokaci chyb je však vhodné sestavit výstupní zprávu z podrobného protokolu, ve kterém bude uvedeno co bylo testováno s popisem a lokalizací chyby.

4.2 Komunikační prostředky – převodník USB/UART

Hlavním komunikačním prostředkem s vnějším prostředím, např. PC je univerzální vysoko-rychlostní asynchronní přijímač a vysílač, anglicky UART (Universal Asynchronous Receiver Transmitter) FT2232C [20], firmy FTDI Chip. Tento čip převádí rozhraní USB ze strany PC na dva nezávislé konfigurovatelné UART/FIFO kanály. K připojení FITkitu se používá USB kabel typu A-B. Na straně PC se po nainstalování ovladačů výrobce jeví oba komunikační kanály jako dva nové sériové porty COM. Komunikace s prvky na FITkitu se tak zredukuje na komunikaci po sériové lince, kde lze s výhodou použít již existující terminálový program nebo využít vlastní komunikaci pomocí svého protokolu.

Jeden asynchronní kanál je připojen k FPGA druhý k mikrokontroléru. Sériový kanál, jenž je připojen k FPGA může být konfigurován do různých módu komunikace, jak bylo popsáno v kapitole 1.8.K mikrokontroléru je asynchronní kanál připojen pomocí signálu RXD a TXD. Oba komunikační kanály mohou pracovat až na rychlosti 921600 baudů/sek.

Projekt FITkit používá ke komunikaci s µC na straně PC terminálový program, který slouží nejen k ovládání nahraných aplikací v mikrokontroléru, ale i posílání binárních souborů (firmware pro FPGA) přes protokol X-modem. Komunikace probíhá rychlostí 460800 baudů/sek.

Je-li zkratovaná propojka J8, je mikrokontrolér ve stavu reset. Aktivací terminálového programu k sériové lince, je tento signál deaktivován, ovládá jej signál ze sériové linky DTR. Po uvolnění resetu se procesor ohlásí na terminálu posíláním textových výpisů. Uživatel ovládá mikrokontrolér posíláním povelů a příkazů v podobě textových řetězců. Protokol X-modem se používá při programování FPGA, protokol je uvnitř mikrokontroléru zpracován a surová binární data jsou použitá k nahraní nové konfigurace do FPGA.

4.3 Komunikační prostředky – sběrnice SPI

FPGA a mikrokontrolér mohou spolu komunikovat pomocí sběrnice SPI (Serial Peripheral Interface). Jedná se o synchronní sériovou komunikaci navrženou firmou Motorola, pracující ve full duplex režimu s architekturou master/slave.

SPI komunikace je založena na jednom Master zařízení a jedním nebo více slave zařízení. Typově stejné zařízení mohou být zapojeny do kaskády (řetězu) anebo mohou sdílet stejnou sběrnici s ostatními SPI zařízeními. Výběr jednotlivých čipů je proveden signálem CS (Chip Select). Na FITkitu je obvykle v pozici Master mikrokontrolér, FPGA a FLASH paměť pak je v pozici slave zařízení.

Maximální komunikační rychlost mezi jednotlivými zařízeními na sběrnici SPI je 460800 bitů za sekundu. SPI protokol je v mikroprocesoru podporován přímo na čipu jako jeden z módu druhého sériového kanálu, FPGA obsahuje také nativní podporu tohoto komunikační protokolu.

4.4 Programování mikrokontroléru a FPGA

Uvnitř mikrokontroléru je výrobcem umístěn operační kód (tzv. boot loader), který umožňuje naprogramování mikrokontroléru přes sériový kanál. K tomu je zapotřebí volně dostupné programové vybavení z balíku msgcc [2], které plně kontroluje celý proces programování. Na FITkitu je zapotřebí mít zkratované propojky J8 a J9, které dovolují ovládat resetovací signál pro mikroprocesor pomocí sériové linky. Maximální rychlost programování je 38600 baudů/sek.

Programování mikrokontroléru lze také realizovat přes JTAG rozhraní, speciálním programátorem MSP-FET430 fy. Texas Instruments. Programátor lze navíc využít k ladění kódu uvnitř mikroprocesoru, což komunikace přes sériovou linku neumožňuje. Ovládání programátoru k programování a ladění lze realizovat buď v uživatelsky přívětivém vývojovém prostředí IAR [21], jehož omezená distribuce je volně k stáhnutí na stránkách společnosti.

Nebo využít konzolové aplikace msp430gdb-debug, která je v plné verzi bez jakýkoliv omezení dostupná v rámci již zmiňovaného vývojového prostředí mspgcc [2].

Flash paměť, ve které je uložen programový kód je nevolatilní (semipernametní) paměť typu RAM. Data uložená v této paměti se tedy uchovávají i po ztrátě napájení.

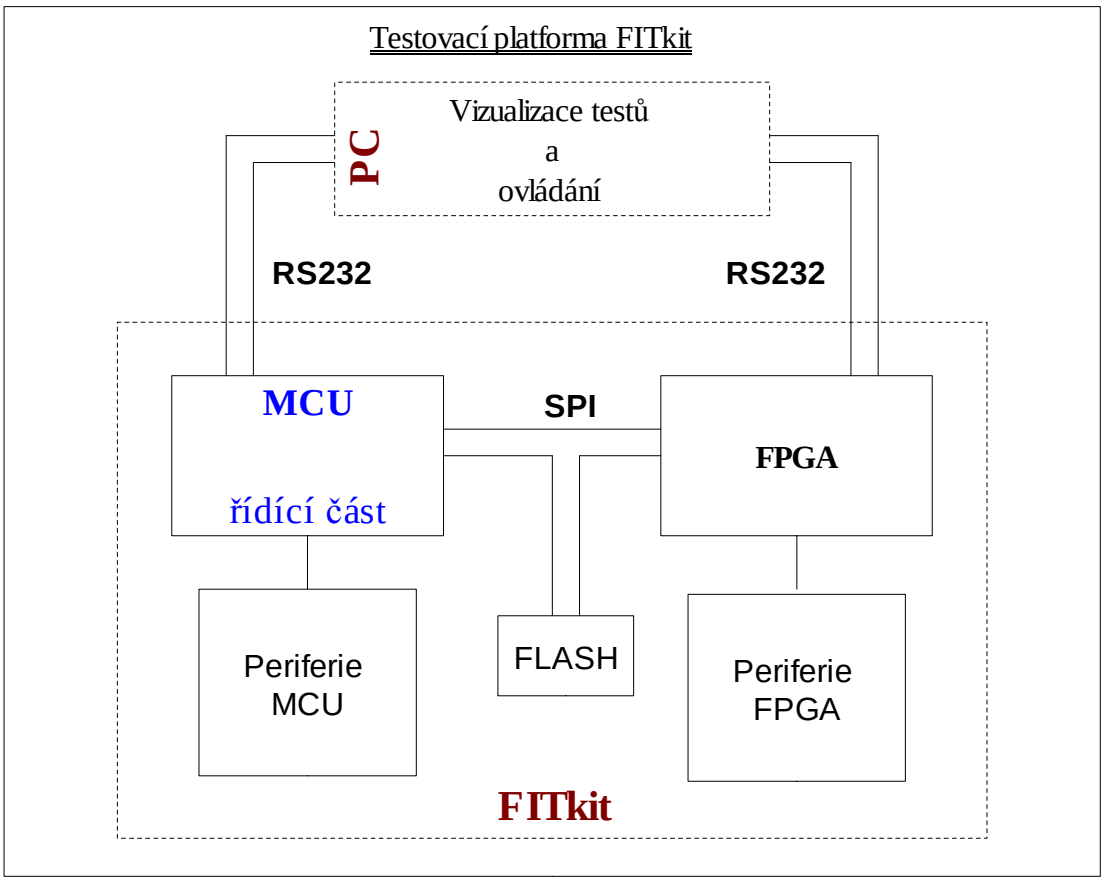
FPGA oproti mikrokontroléru po ztrátě napájení svoji konfiguraci ztrácí a je nutné jej pokaždé naprogramovat. Programování FPGA lze realizovat přes SPI rozhraní, programování ovládá mikrokontrolér, jenž ovládá potřebné signály na signálech FINIT a FDONE. Konfiguraci pro FPGA lze naprogramovat buď z externí FLASH paměti umístěné na FITkitu nebo načíst konfiguraci z počítače přes mikrokontrolér anebo použít programátor připojený na rozhraní JTAG.

4.5 Architektura testovací aplikace – komplexní pohled

Návrh testovací aplikace pro platformu FITkit je rozdělena na tři části a odpovídá schématu master/slave. Jako hlavní řídící člen (master) byl zvolen mikrokontrolér. Jeho programový kód je uchován v nevolatilní paměti, může komunikovat nejen s FPGA ale i s vnějším počítačem. Za určitých omezených podmínek může otestovat FITkit bez komunikace po sériové lince s PC.

Další část testovací aplikace je uložena v FPGA, charakterizována jako slave architektura, která zpřístupňuje periferie připojené k FPGA, podrobnějšímu popisu této architektury je věnována samostatná podkapitola 4.6.

Poslední část testovací aplikace je umístěna v PC, slouží k vizualizaci a ovládání testů. Hlavní schéma návrhu architektury lze najít na na obrázku 10.



Obrázek 10: Testovací platforma FITkit

Jednotlivé části architektury mají za úkoly tyto činnosti:

PC

- je-li to potřebné, programuje mikrokontrolér,
- dodává konfiguraci (FW) pro naprogramování FPGA,
- v kooperaci s uživatelem posílá příkazy pro testování,
- sbírá a vyhodnocuje výsledky z testování jednotlivých komponent,
- testuje komunikační cestu PC-> FPGA.

FPGA – Architektura testující periferie

- zpřístupňuje připojené periferie přes rozhraní SPI mikrokontroléru,
- odpovídá na požadavky přijaté po sériové lince z PC.

Mikrokontrolér

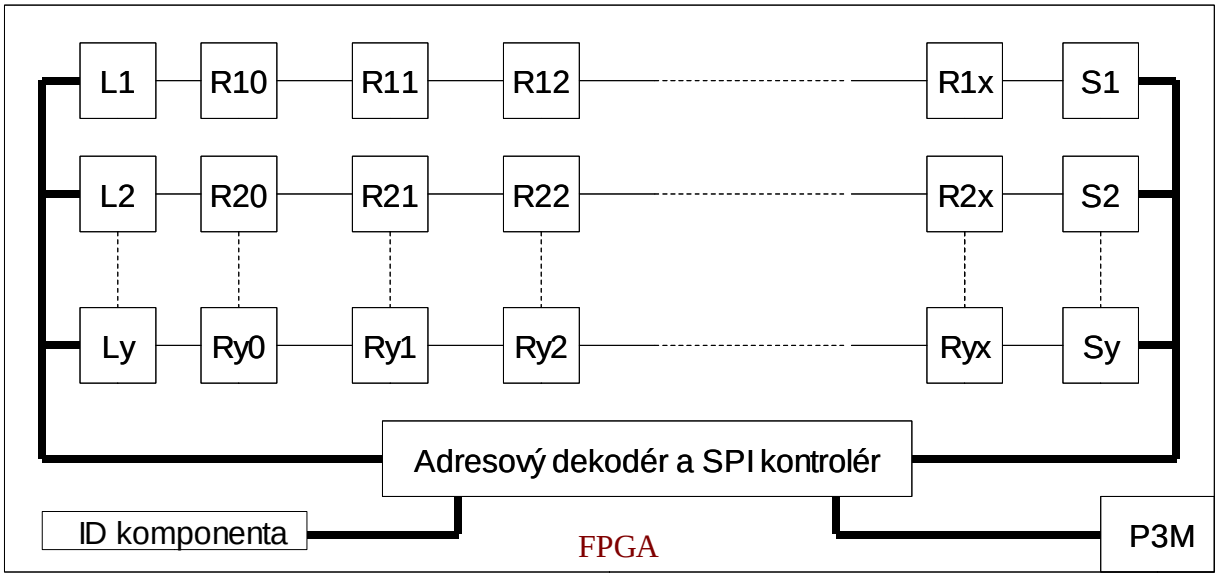
- vyhodnocuje podměty uživatele z PC nebo z připojené klávesnice,
- nahrává konfiguraci pro FPGA,
- testuje svoje vlastní periferie,
- testuje periferie umístěné na FPGA a samotné FPGA,
- testuje FLASH paměť, je-li to možné,
- v omezené míře informuje o výsledcích testování uživatele.

4.6 Architektura testovací aplikace v FPGA

Vnitřní architektura testování FPGA musí být rozdělena na dvě části. První část se zaměřuje na testování samotného čipu FPGA a bloků CLB umístěných v něm. Druhá část zpřístupňuje externí periferie přes SPI sběrnici procesoru.

4.6.1 Vnitřní architektura testovací aplikace v FPGA – Testování čipu FPGA

První konfigurace, jejíž úkolem je otestovat čip FPGA je založena na vytvoření sítě navzájem propojených registrů, popisovaných v kapitole 3.1 - BIST pro FPGA. Zjednodušené blokové schéma komponent na testování čipu FPGA lze najít na obrázku 11.



Obrázek 11: Testování čipu FPGA

Propojením jednobitových registrů do řetězu a napojením na adresový dekodér vznikne matice řádků a sloupců, kde v jednotlivých sloupcích jsou navzájem propojeny výstupy předchozích registrů se vstupem následujícího registru. Jen první a poslední registr každého řádku je připojen na adresový dekodér. Adresový dekodér tak bude adresovat jednotlivé řádky propojených řetězů, dále bude řešit zápis dat do prvního registru a čtení dat z každého řádku z posledního registru. Každý řádek tak odpovídá jednomu bitu adresy v adresovém prostoru.

Adresový dekodér bude připojen k SPI kontroléru, jenž bude napojen na SPI rozhraní k procesoru. Procesor tak bude řídit a vyhodnocovat celé testování FPGA. Pro propagaci informace mezi jednotlivými registry by mohl být využit hodinový signál získaným ze vstupu ACLK, který bude mikrokontrolér měnit po každém vyčtení všech výstupních registrů spojených do řetězů.

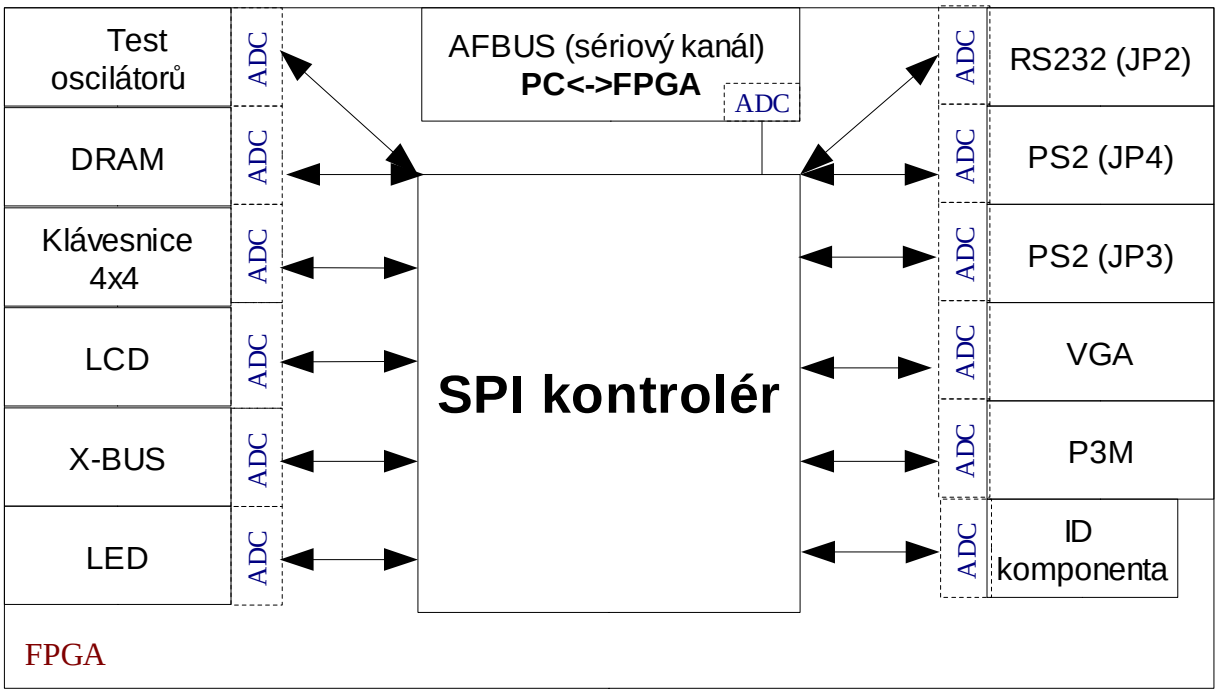
Další využití této konfiguraci spočívá v testování spojů na sběrnici P3M, pro jejíž otestování jsou zapotřebí dvě rozdílné konfigurace lišící se v přiřazení resetovacího signálu pro FPGA.

Speciální postavení v této konfiguraci má blok označený jako ID komponenta. Jedná se o blok jednoduchých osmibitových registrů, vytvořených ze vnitřních struktur FPGA, ve kterých jsou trvale uloženy informace o verzi testovacího designu včetně číselného identifikátoru, který může rozlišovat jednotlivé konfigurace.

Hlavní využití ID komponenty spočívá v ověření komunikace, tedy kontrole SPI rozhraní mezi FPGA a mikrokontrolérem. První operaci, kterou mikrokontrolér vykoná před použitím a ovládáním testovacího designu, je ověření výskytu ID komponenty. Po porovnání získaných hodnot z ID komponenty s předpokládanými může mikroprocesor pokračovat v komunikaci s komponentami uloženými v FPGA.

4.6.2 Architektura testovací aplikace v FPGA – Testování externích rozhraní

Druhá konfigurace FPGA čipu se zaměřuje na testováním externích zařízení připojených k FPGA. Blokové schéma architektury pro testování lze najít na obrázku 12. Architektura je rozdělena na bloky podle funkčnosti nebo periferie, kterou daný blok obsluhuje.



Obrázek 12: Testování externích periferie na FPGA

Srdcem architektury je SPI kontrolér, který je napojen na sběrnici SPI. U všech uvedených bloků, se předpokládá vybavení adresovým dekodérem s definovanou jedinečnou adresou pro identifikaci přes rozhraní SPI.

Lze předpokládat, že každý blok bude mít kromě tohoto adresového dekodéru i určité množství další logiky, aby zpracoval příchozí data a odchozí data. Také mapovaní na konkretní rozhraní bude spojeno s vybudováním pomocné logiky jakou jsou čítače, registry, multiplexory, komparátory, RAM paměti a DCM.

Jednotlivé jména bloků odpovídají funkčnosti a periferiím připojených a připojitelných k FITkitu, které zpřístupňují přes SPI. Jedná se o: LCD, DRAM, PS2 rozhraní, VGA port, RS232, maticová klávesnice a P3M sběrnice.

Další chování bloků lze shrnout takto:

- **LED** - blok je vyhrazen pro ovládání LED diody, kterou lze najít na desce plošného spoje pod označením D4.
- Blok **X-bus**, resp. P3M je určen pro zpřístupnění ovládání stavu sběrnice, která je

vyvedena na konektor JP10, resp. na sběrnici P3M přes rozhraní SPI.

- Blok '**Test Oscilátorů**' by měl poskytovat funkčnost popsanou v kapitole 3.13.
- ID komponenta má stejné funkční postavení jako v předchozí konfiguraci.

Při návrhu architektury by měl být z důvodu případných dalších rozšíření a přehlednosti dodržen modulární návrh představený v této architektuře. Na projektu FITkit se používá programovací jazyk VHDL, u kterého lze s výhodou zachovat modulární systém nasazením VHDL komponent [1].

Tyto VHDL komponenty obsahují entitu, což je popis popis vstupních a výstupních signálů, které daná komponenta ovlivňuje. Uvnitř komponenty tedy může být behaviorální nebo strukturní popis funkčnosti každého bloku.

4.7 Architektura testovací aplikace v MCU

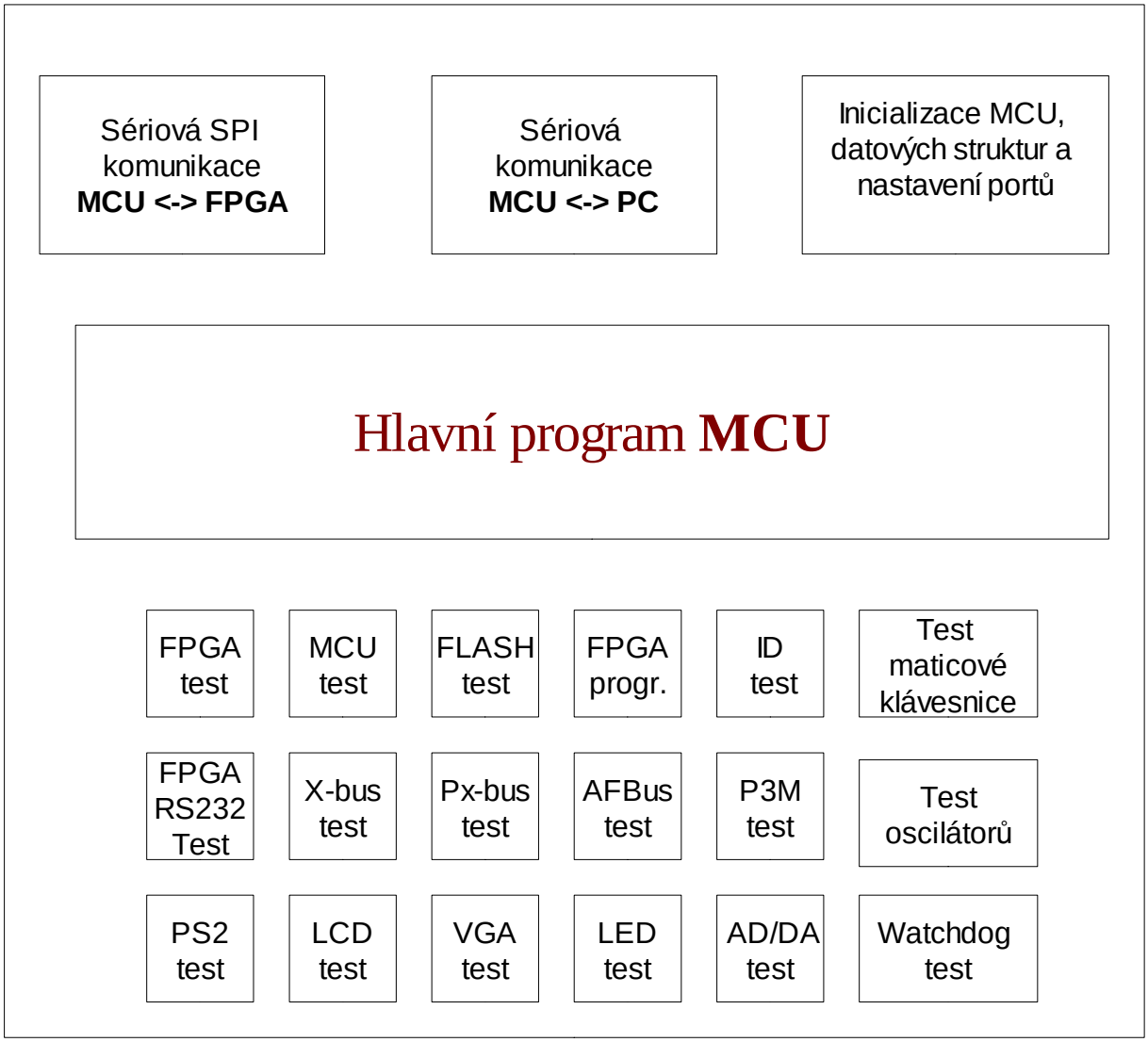
Hlavní řídící jednotkou vykonávající samotné testování FITkitu je mikrokontrolér. Architekturu testovací aplikace mikrokontroléru lze najít na obrázku 13.

Architektura obsahuje tyto části:

- inicializaci mikrokontroléru a datových struktur potřebných pro chod testu,
- hlavní program,
- obsluhy sériové komunikace s vnějším rozhraním (s PC a FPGA),
- testovací rutiny pro testování FITkitu a proces programování FPGA.

Úkolem hlavního programu je přijímat požadavky z PC nebo z maticové klávesnice a na základě těchto požadavků testovat zařízení a podávat informace o jejích výsledcích. Součásti těchto úkolů je i inicializace portů a programování FPGA.

1 *Je to určitá náhrada za přímé vkládání zdrojových souborů jak lze provádět například u programovacího jazyku C příkazem preprocesoru #include.*



Obrázek 13: Návrh architektury testovací aplikace v MCU

Jednotlivé bloky mají tuto funkčnost:

- **FPGA test** – testuje FPGA čip, podle metodiky uvedené v kapitole 3.1. Na začátku testu nastaví vstupní registry registrových řetězů. Během testu generuje mikrokontrolér hodinový signál ACLK. Po každé změně náběžné hrany se vyčte obsah registrů a vyhodnotí se jejich stav. Výsledkem testu je dvoustavová informace, říkající zda je FPGA čip v pořádku či nikoliv.
- **MCU test** – testuje jádro a ALU mikrokontrolér, podle metodiky uvedené v kapitole 3.2. Výsledky jednotlivých části testů jsou porovnány s uloženými konstanty. Celkovým výsledkem testu může být osmibitová hodnota, sestavena z dvoustavové informace (0/1)

vyjadřující úspěšnost či neúspěšnost vykonaného testu, zakódována pomocí této tabulky:

Bit	7	6	5	4	3	2	1	0
Operace	8-bitová aritmetika	16-bitová aritmetika	32-bitová aritmetika	Float operace	8-bitový case	16-bitový case	Násobení matic	Aplikace FIR filtru

- **FLASH test** - test FLASH paměti bude prováděn jen v případě, že FITkit je propojen s PC. Podmínka plyne z destruktivního testování paměti FLASH, která vymaže případnou uloženou konfiguraci FPGA uloženou v této paměti. Činnost bloku je implementace algoritmu March-X, uvedeného v kapitole 3.3. Pokud při testu byly nalezeny chyby, je výsledkem testu adresa a pár dat s předpokládanými správnými a skutečně načtenými daty.
- **Programování FPGA** – je blok rutin umožňující naprogramování FPGA ať ze sériového portu nebo přímo z paměti FLASH. Jedna konfigurace pro čip SPARTAN-3, XC3s50 zabírá 54908 bytů. Do 2MBitové flash paměti lze tak uložit obě konfigurace.
- **ID Test** – úkolem tohoto bloku je jednak vyčtení informací z hlavní identifikační komponenty v FPGA. Zároveň komponenta poskytuje verzi a identifikaci nahraného firmware v µC pro potřeby identifikace ve vizualizační vrstvě na straně PC. ID komponenta pro mikrokontrolér by měla také obsahovat hodnotu verze SW a hodnotu identifikující testovací design.
- **Test maticové klávesnice** – možný rozsah činnosti tohoto bloku lze rozdělit do dvou částí:
  - Ovládání testovací aplikace pomocí této klávesnice

    Maticová klávesnice připojena k FITkitu může posloužit k intimnímu ovládání testovacího procesoru. Procesor musí však vyloučit případnou vadu klávesnice, která by se projevila nekorektním chováním aplikace. Lze ověřit například tak, že po nahrání odpovídající konfigurace do FPGA procesor ověří, zda není stisknuta některá z kláves, pokud ano, dá se předpokládat, že rozhraní klávesnice vykazuje chyby. Deaktivace musí být také možná pomocí ovládacího programu spuštěného na PC.
  - Samotné testování klávesnice

    Metodický postup testování je uveden v kapitole 3.7. Uživatel by mělo být zobrazeno při stisku tlačítka odpovídající textová reprezentace symbolu uvedeného na klávesnici buď na LCD nebo v PC.

- **FPGA RS232 Test** – blok testuje sériové rozhraní připojené k FPGA postupem uvedeným v kapitole 3.5. V případě neúspěšného testu by měl být uživatel informován o dvojici špatných vodičů, které neprošly testováním.
- **X-bus Test a Px-Bus Test –** Oba moduly postupně mění logické hodnoty na výstupních pinech FPGA, resp. mikrokontroléru podle schématu uvedeného v kapitole 3.12. Rychlost změny přiložených testovacích vektorů ovládá uživatel z klávesnice nebo z PC.

Průběh testu a výsledek bude uživatel ověřovat vizuálně. Modul by měl uživateli zároveň poskytovat informaci o názvu pinu, který je testován, pro lokalizaci případné chyby na sběrnici.

- **AFBus test –** k úspěšnému provádění této sběrnic je zapotřebí osazena paměť, jak je uvedeno v kapitole 1.8. Modul by měl být sestaven ze dvou části. První část testu by měla být zaměřena na dvouvodičovou asynchronní komunikaci RX/TX, druhá na kontrolu FIFO režimu AFBus 12-bitové sběrnice. Test v druhé části může být založen na algoritmu „universal test set“ popisovaný v kapitole 3.12
- **P3M test** – Funkce sestavené v tomto modulu se zabývají testováním P3M sběrnice mezi FPGA a mikrokontrolérem. Samotné testování musí být prováděno samostatně ve dvou krocích, podle konfigurace uložené v čipu FPGA. Samotné konfigurace se totiž liší lokalizaci resetovacího signálu pro FPGA, které je vyvedeno na sběrnici P3M. Algoritmus, který je aplikován na test této sběrnice je popsán v kapitole 3.10
- **Test oscilátorů** – je blokem, který vyčítá a vyhodnocuje hodnoty čítačů uložených v FPGA, výsledkem testu je vypočtená frekvence, která byla načtena mezi dvěma časovými intervaly, jak je podrobněji popsáno v kapitole 3.13.
- **PS2 test** - je zaměřen na testování připojeného externí zařízení (klávesnice nebo myši) do vstupních konektorů PS2. Blok a rutiny uložené v něm by měly řešit jednak samotnou komunikaci s řadičem uloženým v FPGA a jednak i vyhodnocení a interpretaci výsledků uživateli, například informaci o stisknutých klávesách.
- **LCD test** - podobně jak u maticové klávesnice, i blok rutin pro obsluhu LCD může být sestaven ze dvou části
  - vizualizace výsledků testů
  - rutiny pro otestování LCD, uvedené v metodice v kapitole 3.8. Výsledek a průběh testu bude uživatel ověřovat vizuálně.
- **VGA test –** mikrokontrolér zde plní úlohu komunikace s FPGA a generuje obrayce pro testování VGA portu, sepsaných v kapitole 3.4. Výsledek opět hodnotí uživatel vizuálně.
- **LED test –** rutiny pro ovládání signalizačních diod umístěných na FITkitu. Činnost bloku je změna aktivací a deaktivací jejich svitu. Uživatelsky definované LED lze využít i pro signalizaci stavu chodu hlavního programu.
- **AD/DA test** – jsou funkce pro řízení operací s D/A a A/D převodníky, které se vyskytují na FITkitu. Součást funkcí by měl být test těchto převodníků, popisovaný v kapitole 3.9. Výsledkem testování jsou naměřené hodnoty získané po převodu z A/D převodníku.
- **Watchdog test** – je založen na kontrole Watchdog. Funkce Watchdog je kontrolní funkce uvnitř procesoru, která systematicky počítá impulsy vykonáváním programu. Program musí obsahovat příkazy, které tento čítač nulují. V případě, že se tak nestane a čítač přeteče, je procesor restartován.

Rutinu, která bude funkci Watchdog testovat, lze založit na dlouhotrvající smyčce prázdných příkazů, v které nebude tento čítač nulován. Pokud funkce Watchdog funguje, mikrokontrolér bude po jistém čase restartován. Vizualizační vrstvě (PC) lze po restartu poslat kód, identifikující, že došlo k restartu systému nebo po ukončení této smyčky poslat jiný kód identifikující stav, že naopak k restartu nedošlo.

Sériová komunikace mezi mikrokontrolérem a SPI lze řešit na hardwarové úrovni vhodným nastavením UART1 kanálu. Rutiny pro komunikaci lze najít v aplikační části programových návodů na stránkách FITkitu.

Prvním krokem po restartu mikrokontroléru je inicializace portů a datových struktur. Mezi tuto činnost patří i čekání na stabilizaci napájecího napětí a generátorů hodin.

4.8 Architektura testovací aplikace na straně PC

Vizualizace průběhu testování a výsledků lze řešit dvěma způsoby. První způsob je použití terminálového programu na straně PC, kde mikrokontrolér bude vyhodnocovat a zpracovat výsledky do srozumitelné, ale mnohdy méně přehledné formy uživateli.

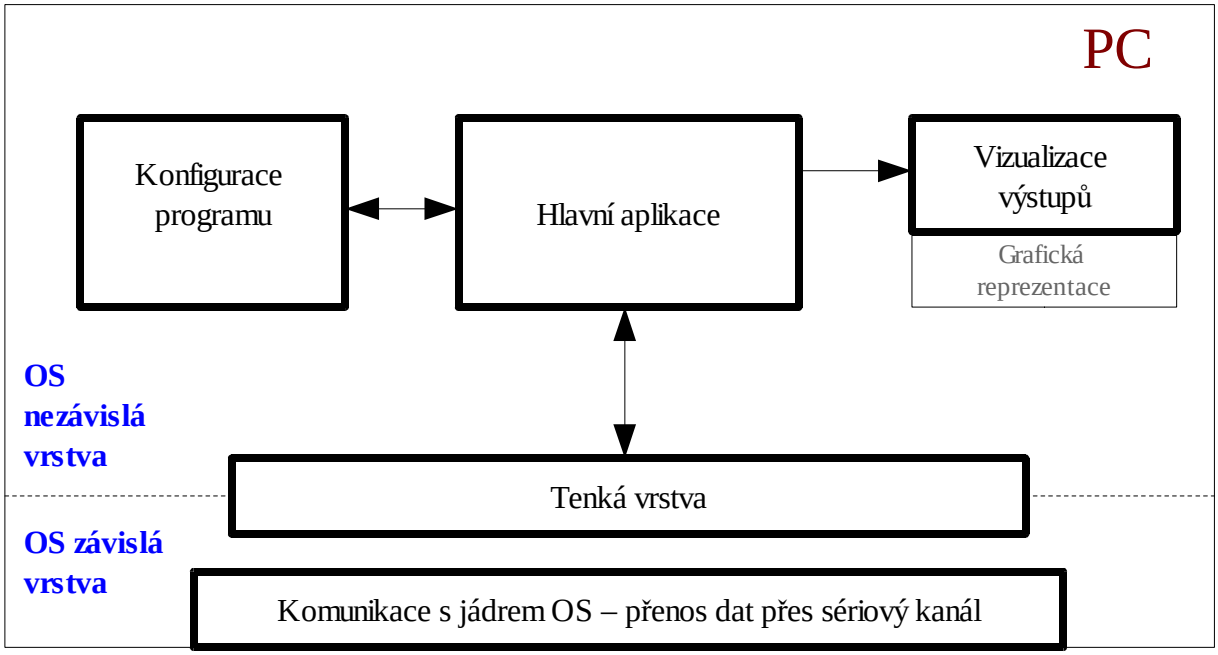
Druhý robustnější způsob je naprogramování vlastního protokolu a naprogramování vlastní aplikace, která komunikuje s mikrokontrolérem. V porovnání s předchozím přístupem vede vytvoření testovací aplikace k:

- automatizovanějšímu procesu testování,
- jednoduššímu ovládání,
- lepších vizualizací výsledků,
- nezávislosti na softwarovém vybavení operačního systému a použitém terminálovém programu.

Nevýhodou použití vlastního komunikačního protokolu je v nutnosti vytváření specifických knihoven komunikujících s konkrétním operačním systémem na kterém aplikace běží. Tato nevýhoda může být značně potlačena v případě, že je celá aplikace rozdělena na dvě vrstvy.

První vrstva, často označovaná jako spodní, obsahuje rutiny a funkce komunikující s jádrem operačního systému. Zatímco druhá vrstva, označovaná jako horní vrstva, je obvykle nezávislá na konkretním operačním systému a poskytuje zbývající funkcionalitu. Pro spojení těchto dvou částí se často používá tenká vrstva, která převádí, pro zachování kompatibility a přenositelnosti, volání funkcí horní vrstvy na volání funkcí spodní vrstvy a naopak.

Komplikovanější návrh dvouvrstvové architektury lze zjednodušit použitím vhodné open-source knihovny, která řeší problémy spojené s návrhem spodní vrstvy. Důsledkem převládajících kladných vlastností nad negativními vznikl návrh architektury testovací aplikace pro platformu FITkit, který lze najít na obrázku 14.



Obrázek 14: Návrh architektury testovací aplikace na PC

Srdcem horní vrstvy je blok rutin, označených jako „hlavní aplikace“. Ta komunikuje s tenkou vrstvou, posílá a přijímá data ze sériového kanálu.

Další součástí je i vizualizace výstupů a výsledků z testování, výsledky mohou být reprezentovány graficky. Poslední blokem je konfigurace programu, která poskytuje hlavní aplikaci nastavení, které mohou být načteny z externího souboru jako jsou:

- jména komunikačních portů, na kterých bude probíhat komunikace,
- seznam testovaných periferií, které si přeje uživatel testovat,
- formu vizualizace a zpracování výsledků (do souboru, v konzoli, graficky, atd.),
- a další.

Celkově tedy aplikace zpracovává požadavky uživatele, zobrazuje výsledky testování, připravuje data pro programování FPGA, testuje sériovou komunikaci s FPGA.

Součástí této aplikace může být i systém pro přípravu binárních souborů, které jsou nedílnou součástí automatizovaného postupu pro testování. Tato příprava spočívá ve spouštění externích programů, které provádějí kompilaci zdrojových souborů a programů, které řeší programování mikrokontroléru.

4.9 Analýza architektury

V této kapitole byla představena architektura testovací aplikace pro otestování platformy FITkit. Cílem bylo navrhnout robustní architekturu s jednoduchým ovládáním, samočinným testováním, která

umožní prověřit funkčnost všech jejich komponent.

Návrh architektury byl rozdělen na tři části, jedna část se věnovala FPGA, druhá část mikrokontroléru a třetí obsluze a vizualizaci na PC. Tyto části dále byly rozděleny na komponenty, popisující jednotlivé periferie a součásti bloků k otestování. Inkrementálním způsobem programování lze postupně vytvořit celou testovací aplikaci pro platformu FITkit.

Výběr umístění hlavního řídícího orgánu testu do mikrokontroléru umožní otestovat platformu FITkit i bez použití PC v případě, že FPGA je nebo může být naprogramováno z uložené konfigurace v paměti FLASH. Jednotlivé konfigurace pro čip FPGA jsou vybaveny identifikátorem, které konfiguraci čipu identifikují a pro procesor není problém ověřit, že má správnou konfiguraci FPGA k dispozici.

Samotná ID komponenta sice neověří, zda zbývající část konfigurace odpovídá požadované funkcionalitě, ale zabrání používání designu, který byl nahrán do FPGA omylem a mohl by potenciálně způsobit poškození komponent systému. Problém ověření přítomnosti jednotlivých komponent v konfiguraci FPGA lze řešit vyhrazením lokální identifikace, například vyhrazením jedné adresy v adresovém prostoru každé komponenty pro získaní identifikátoru.

Výsledky testování mohou být v omezené formě zobrazeny na LCD a pro řízení a ovládání testu lze použít maticovou klávesnici. Použití spuštěné testovací aplikace v PC tedy není vždy nutné, ostatně konfiguraci FPGA lze nahrát i přes JTAG rozhraní, což platí i pro samotný mikrokontrolér, kde programování v případě nefunkčnosti sériového propojení s PC selže.

Architektura je navržena pro otestování všech komponent systému. K otestování je zapotřebí externí vybavení, jako je VGA monitor, PS/2 klávesnici(PS/2 myš), konektor zapojený do LoopBack zapojení pro test sériového kanálu a testovací destičky pro test sběrnic X-bus a Px-Bus.

V menší míře se architektura zabývá testem mikrokontroléru, kde není navržen postup pro otestování vnitřních komponent procesoru, jako je FLASH a RAM paměť mikrokontroléru.

Testování čipu FPGA bylo zredukováno na vytvoření sítě navzájem propojených registrů. Ostatní části jako jsou BlockRAM paměti a DCM mohou být otestovány přidáním další konfigurace zaměřenou na jejich otestování.

Lokalizace případných chyb je rozdělena na dva případy. Při použití automatizovaných prostředků je možnost lokalizace chyby ovlivněna použitým algoritmem (test P3M sběrnice, test RS232, atd.). Při vizuálním, neautomatizovaném hodnocení průběhu testu musí uživatel sám vyhodnotit stav periferie. Pro lokalizaci chyb mu však mohou posloužit vizualizační prostředky, které testera-uživatele informují o procesu testování (VGA výstup, test sběrnic X-bus a Px-Bux).

Architektura není schopna odhalit chyby, nacházející se v aplikační části programu mikrokontroléru, při návrhu aplikačního programu by se však mělo pamatovat na použití funkce Watchdog, která může odhalit nestandardní chování aplikace jejím restartováním. Na test této funkce

se v architektuře pamatuje.

Dále architektura není schopna odhalit chyby v čipu FPGA, umístěné v části obvodu řešící komunikaci přes rozhraní SPI a uložení ID komponenty. Platí to i pro konfiguraci, která testuje samotný čip FPGA.

Řešením tohoto problému by bylo vytvoření dvou funkčně stejných konfiguraci, kde se komponenty pro SPI komunikaci a ID komponenta budou nacházet na dvou různých místech čipu. Tento požadavek může zaručit jen technika označována jako 'partition', ta umožňuje rozmístit na určené místo čipu část konfigurovatelných bloků. V opačném případě nelze totiž umístění konfigurovatelných bloků garantovat. Popisovaná technika je však mimo rámec této práce a momentálního systému překladu zdrojových souborů na projektu FITkitu.

Architektura testů nemůže lokalizovat chybu vyskytující se na vodičích určených pro sběrnici SPI a vodiči vyhrazený pro rozvod hodinového signálu SMCLK. Obě chyby se projeví v nemožnosti komunikace s FPGA nebo FLASH pamětí.

5 Implementace aplikační části v PC

Aplikační obsluha na straně PC komunikující s FITkitem je na projektu řešena pomocí terminálového programu, který zjednodušuje návrh aplikace na úkor ovládacích schopnosti. Pro tuto práci však bylo rozhodnuto naprogramování vlastní aplikace a komunikačního protokolu, který do značné míry zjednodušuje ovládání a automatizuje proces testování platformy FITkit.

Kapitola implementace aplikační části v PC je rozdělena na několik částí. V první podkapitole je zdůvodněn výběr programovacího jazyka a programovacího prostředí pro vývoj aplikace. Kapitola dále pokračuje v seznámení s knihovnou RxTx, kterou lze použít pro komunikaci po sériové lince.

Hlavní část kapitoly je představení komunikačního protokolu a rozboru testovací aplikace a jejích části.

Součásti implementace je i programová dokumentace generovaná systémem Doxygen [22] ta však není součásti této práce, lze ji najít na přiloženém CD, jehož struktura je uvedena v kapitole 8 - Aplikace. Při implementaci zdrojových kódu jsem čerpal z knihy [23].

5.1 Výběr programovacího jazyka

Po pečlivé úvaze a zhodnocení výhod a nevýhod byl programovacím jazykem pro vytvoření aplikace zvolen programovací jazyk Java Programovací jazyk Java je objektově orientovaný programovací jazyk, který vyvinula firma Sun Microsystems. Zdrojový kód Javy je interpretovaný a kompilovaný do tzv. mezikódu, (byte-code), který je nezávislý na operačním systému či architektuře počítače.

Mezi argumenty, které vedly k výběru tohoto programovacího jazyky patří:

- multiplatformnost a přenositelnost zkompilovaného kódu (byte-code),
- volně dostupné kompilátory kódu,
- volně dostupné IDE (Integrated Development Environment) pro vývoj aplikací,
- velmi dobrá podpora dokumentace a nápovědy k jazyku,
- zabudovaná podpora komunikace po sériové lince.

Další úvaha směřovala k výběru IDE, ve kterém aplikace bude vznikat. Byly stanoveny podmínky, které toto prostředí musí obsahovat a lze je shrnout do těchto bodů:

- volně dostupný produkt bez licenčních omezení,
- multiplatformní aplikace běžící minimálně na operačních systémech Windows a Linux,
- integrovaný editor a prostředky k ladění kódu,
- podpůrné prostředky pro vytváření grafického vzhledu aplikace,

- podpora správy projektu (integrace souborů do balíčků).

Uvedeným vlastnostem vyhovují například vývojové nástroje Netbeans IDE a Eclipse. Vybrán byl nakonec produkt NetBeans [24]. Implementace byla prováděna ve verzi 5.5 (stav k 1.3.2007).

5.2 Knihovna RxTx – podpora sériové komunikace

Knihovna RxTx [25] jak ji nazvali autoři, je multiplatformní knihovnou pro jednoduší řízení komunikace po sériové a paralelní lince. Tato knihovna rozšiřuje vlastnosti balíku io.*, které řídí přístupy k periferiím počítače, mezi které patří i sériový kanál. Knihovna podporuje operační systémy Windows, Linux, Solaris a Mac OS.

Knihovna je šířena pod LGPL licencí a redukuje obsluhu komunikace aplikace se sériovým kanálem na sadu jednoduchých funkcí. Samotná knihovna ctí model tenkého klienta a knihovny přistupující k jádru operačního systému, uvedeného v kapitole 4.8 - Architektura testovací aplikace na straně PC.

Tenký klient je balíček pojmenovaný jménem RXTXcomm.jar. Knihovny pro přístup k jádru operačního systému jsou rozlišeny podle operačního systému takto:

- rxtxserial.dll, (rxtxparallel.dll) [Windows]
- librxtxserial.so , (librxtxparallel.so) [Linux, Solaris]
- librxtxSerial.jnilib [Mac OS]

Tyto soubory musí být umístěny v systémové cestě nebo v adresáři při spouštění aplikace. Načtení odpovídající knihovny je postaráno automaticky kompilátorem Javy podle typu operačního systému. Pro činnost aplikace jsou však zapotřebí jen knihovny zabývající se sériovým kanálem (rxtxserial.dll,librxtxserial.so...)

První nevýhoda je nutný výskyt externích souborů při spuštění aplikace, druhou nevýhodou je poměrně chudá dokumentace k popisu jednotlivých parametrů metod. Na internetu lze však najít dostatečné množství návodů a příkladů pro použití této knihovny. V příloze B. lze najít jednoduchý zdrojový program, který reprezentuje použití knihovny pro otevření sériového portu, vyslání a načtení jednoho znaku, v závěru je uvedeno i uzavření portu.

Po otevření sériové portu a jeho přiřazení vstupnímu a výstupnímu streamu (InputStream/ OutputStream) se pracuje se sériovým portem v bajtově orientovaném proudu jako z běžnými soubory v Javě. K dispozici jsou tedy metody read, write.

5.3 Komunikační protokol mezi PC a µC

Oboustranný protokol mezi PC a mikrokontrolérem je založen na paketové výměně. Paket obsahuje identifikační příkaz, počet odesílaných dat, data a hodnotu kontrolního součtu každého paketu.

Data, která lze najednou poslat přes sériový kanál jsou omezena velikostí 32767 bytů, při odesílaní

jsou však data rozdělena na pakety o velikosti 127 bytů.

Řídící paket I.	Byte 0	Byte 1	Byte 2	Byte 3..n-2	Byte n-1	Byte n
	CMD	**LEN_HI**	**LEN_LO**	**DATA0..DATA121**	**CRC_HI**	**CRC_LO**

Tabulka 1: Řídící paket – struktura prvního paketu

Řídící paket II.	Byte 0..n-2	Byte n-1	Byte n
	DATAx..DATA(x+124)	**CRC_HI**	**CRC_LO**

Tabulka 2: Řídící paket – struktura druhého a další paketu

První odesílaný paket má strukturu uvedenou v tabulce 1., druhý a další paket má strukturu mírně odlišnou, schéma je uvedeno v tabulce 2. Jednotlivé položky mají tento význam:

- **CMD** – řídící příkaz, který identifikuje operaci. [0..255]
- **LEN_HI+LEN_LO** – vyšší a nižší číslo hodnoty udávající počet bytů, které se budou odesílat přes sériový kanál. [0..32767][2]
- **DATA0..DATA122** – data k odeslání, maximální počet v prvním paketu je 122bytů.
- **DATAx-DATA(x+124)**– jsou data posílána v druhém a dalším paketu. Maximální počet, které se v jednom paketu vysílají v jednom paketu je 125bytů.
- **CRC_HI+CRC_LO** – vyšší a nižší byte čísla kontrolního součtu (CRC-16) aktuálního paketu [-32768..32767]

Nejmenší množství dat, které je možné odeslat je 1 byte. Počet bytů, které se vysílají po sériové lince je roven velikosti 6 bytů.

Jednotlivé pakety jsou chráněny CRC 16-bitovým součtem, který je vypočten před každým odesláním paketu. Pokud je přijatý paket přenosem poškozen, tj. CRC součet neodpovídá, je na přijímací straně paket zahozen. Po příjmu paketu se na straně mikrokontroléru posílá tzv. stavový paket, který má následující strukturu:

Stavový paket	Byte 0	Byte 1	Byte 2	Byte 3	Byte n-1	Byte n
	R_CMD	**LEN_HI**	**LEN_LO**	**STATUS**	**CRC_HI**	**CRC_LO**

Tabulka 3: Stavový paket – struktura

Význam jednotlivých polí je stejný jak u normálního paketu, až na:

2 *Programovací jazyk JAVA neobsahuje celočíselné proměnné, proto je velikost dat (LEN) omezena na velikost 32767 bytů, záporná hodnota velikosti dat zde ztrácí smysl. CRC součet však může nabývat i záporných hodnot, zde to tak nevadí, jedná se totiž jen o číselnou reprezentaci 16bitové hodnoty, která je na straně mikrokontroléru porovnávána na rovnost.*

R_CMD – odesílací strana posílá zpětně přijatý kód [0..255]

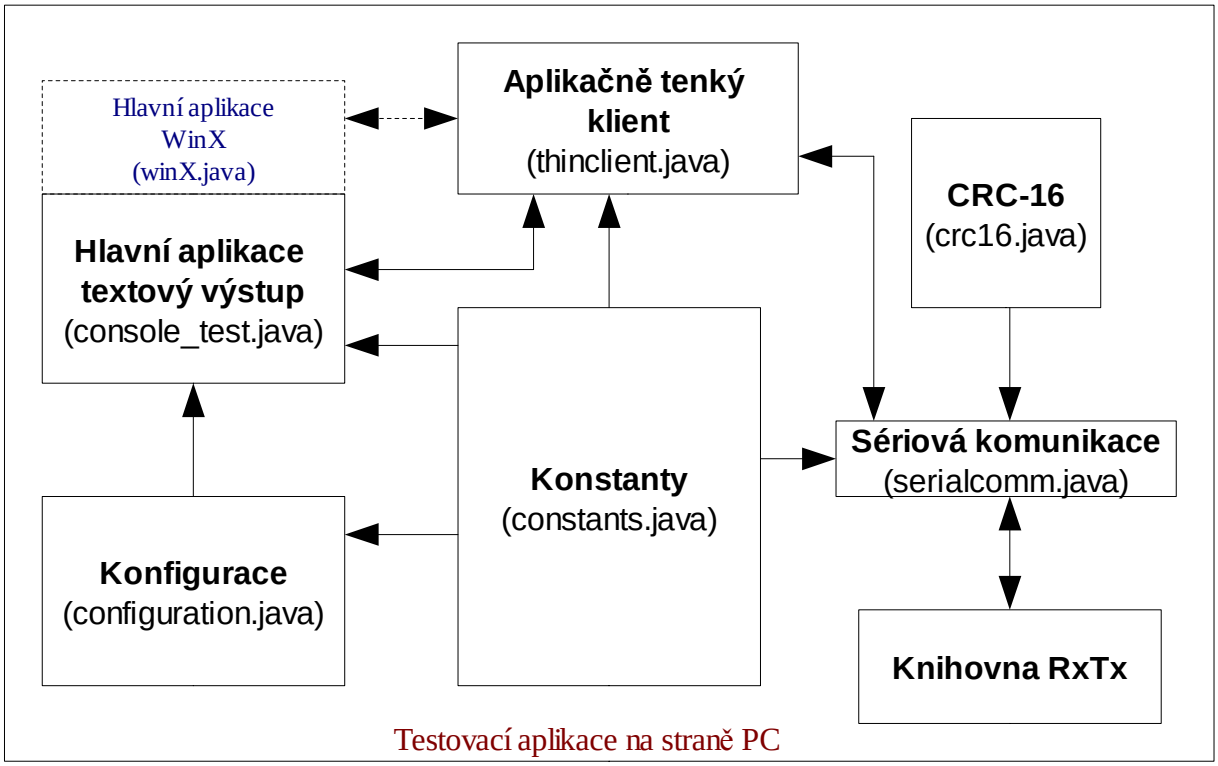
STATUS – stav přijetí nebo odmítnutí paketu, pole může nabývat těchto hodnot:

Hodnota	Konstanta	Význam
0x08	#STATUS_OK[3]	Paket byl přijat v pořádku
0x20	#STATUS_BAD_COMMAND	Příkaz nelze rozpoznat
0x40	#STATUS_CRC_BAD	CRC na příjmu neodpovídá

Příjem dat na straně PC je blokující, je však vybaven funkcí Time-out, ta je nastavena konstantou #COMMUNICATION_TIME_OUT.

5.4 Schéma aplikační části na straně PC

Aplikace je sestavena z několika zdrojových souborů, všechny patří do balíčku application. Jejich propojení a závislosti mezi třídami je zobrazeno na obrázku 15.



Obrázek 15: závislosti bloků v testovací aplikace na straně PC

3 *Takto uvedené jméno konstanty, která reprezentuje uvedenou hodnotu, lze najít v souboru constants.h nebo constans.java.*

Celá aplikace je sestavena ze souborů console_test.java, configuration.java, thinclient.java, crc16.java a knihovny RxTx což jsou soubory RXTXcomm.jar a knihovny pro operační systémy.

Tyto soubory jsou uloženy včetně adresářové struktury pro NetBeans IDE v adresáří sw_pc/src. Výsledkem kompilace v prostředí NetBeans je jeden soubor pojmenovaný jako application.jar, nacházející se v adresáři `sw_pc/dist`.

Dále v textu následuje krátký popis funkčnosti jednotlivých části aplikace.

5.4.1 Konstanty – constants.java

Třída konstanty má vůdčí postavení v celé testovací aplikaci. Sdružuje všechny konstantní hodnoty, které se využívají v aplikaci na jednom místě. Strukturu třídy lze rozdělit na tyto části, za pomlčkou je uveden znakový prefix, který přiřazuje konstanty do skupin, xxx je pak název konstanty.

- konstanty pro chod aplikace,
- návratové hodnoty metod – ERR_xxx,
- hodnoty příkazů, které slouží pro řízení a ovládání aplikace na mikrokontroléru – CMD_xxx,
- hodnoty vracející se ve status paketu - STATUS_xxx,
- definice tříd, které se používají podobně struktury v jazyku C – c_xxx,
- definice jednoduchých tříd, které fungují podobně jako makra.

Například návratová hodnota metody, která operuje se soubory, může vrátit `ERR_FILE_NOT_FOUND` – soubor nebyl nalezen nebo `ERR_OK` – operace proběhla v pořádku. Význam jednotlivých konstant je uveden v souboru, případně v programové dokumentaci.

Konstanty CMD_xxx je seznam příkazů pro řízení ovládací aplikace umístěné v mikrokontroléru.

Vzhledem k faktu, že třída constants se používá ve všech ostatních třídách této aplikace, je do této třídy i přidaná definice metod, které se používají pro výměnu parametrů mezi voláním funkcemi. Pro jednoduchost, kompaktnost a možnost měnit hodnotu proměnných předávaných metodám, se jako typ vstupních proměnných používají jednoduché třídy[4]. Tyto metody mají před jménem prefix c_.

Konstanty uvedené dále v této práci mají tvar: `#konstanta`, kde `konstanta` je jméno konstanty, které ze je najít právě v tomto souboru, kopie hodnot je uvedena v souboru constants.h pro mikrokontrolér (liší se však zápisem konstant)

5.4.2 Aplikačně tenký klient – thinclient.java

Při vytváření vizualizační části aplikace byla vytvořena tenká vrstva, tenký klient, která převádí

4 *Jednoduché datové typy se v jazyku Java zásadně předávají hodnotou (call-by-value) což neumožňuje jejich modifikaci uvnitř volající metody. Zatímco třídy a pole se předávání referencí. Třída v porovnání s polem umožňuje lepší správu a přehlednost. Vnitřní proměnné mohou být například po vytvoření inicializovány.*

strukturované data lišící se od jednotlivých příkazů a činností, které lze vykonávat na platformě FITkit na data, která jsou přes sériový kanál mikrokontroléru posílány.

Tento tenký klient zjednoduší vytvoření různých nadstaveb pro reprezentaci výsledků, do budoucna také umožní zaměnit případně celý sériový protokol za jiný bez dopadů na samotnou testovací aplikaci.

Tenkého klienta využívá třída console_test, která může být v budoucnu nahrazena třídou `winX.java`, která bude reprezentovat grafickou vizualizaci výsledků.

Každá veřejná metoda komunikující s FITkitem vytváří paket, vkládá do něj příkaz, podle typu příkazu nastavuje velikost odesílaných dat a je-li potřeba, vkládá do paketu vstupní data. Tento paket pak metoda odešle pomocí třídy serialcomm patřičným sériovým kanálem.

Hierarchie použití tříd ilustruje jednoduchý příklad na vyčtení čísla verze z mikrokontroléru.

```
//třída c_Firmware je umístěna v souboru constants.java
static class c_Firmware        {
        int Major,Minor;               //třída obsahuje prvky Major a Minor
        c_Firmware(){Major=0;Minor=0;}; //inicializace prvků
}

//metoda getFW_version_MCU je umístěna v souboru thinclient.java
public int getFW_version_MCU(constants.c_Firmware fw){
       ......              //komunikace s uC
      fw.Major=data[0];   //změna vnitrnich hodnot
      fw.Minor=data[1];
      return(OK);
}

//metoda console_getversion() je umístěna v souboru console_test.java
public void console_getversion() {
       constants.c_Firmware fw=new constants.c_Firmware(); //vytvoření instance
       client.getFW_version_MCU(fw);                       //volaní metody
}
```

Třída `constants.java` již byla vzpomínána výše. Metoda **getFW_version_MCU()** v tenkém klientu tedy získá voláním nadřazené funkce (`console_getversion()`) referenci na třídu c_firmware, kterou po získání dat z mikrokontroléru naplní hodnotami. Na podobném principu pracuje většina metod umístěných v tenkém klientovi.

Každá metoda v tenkém klientovi vrací návratovou hodnotu, ta informuje volajícího o výsledku operace. Podrobný popis a seznam návratových kódů jednotlivých funkcí v tenkém klientovi jsou uvedeny v programové dokumentaci.

5.4.3 Sériová komunikace – serialcomm.java

Třída serialcomm, s kterou komunikuje jen tenký klient, poskytuje metody pro komunikaci se

sériovými porty. Pro komunikaci s FPGA a mikrokontrolérem jsou vytvořeny dva nezávislé streamy pro příjem a vysílaní dat.

Identifikace streamu je provedena přes výčtovou hodnotu `#COM_ID {FPGA/MCU}.`

Mezi operace, které třída poskytuje patří:

- otevření sériového portu,
- vyslání dat,
- příjem dat – blokující s time-out,
- zavření portu,
- metoda, která vrací počet nenačtených bytů v bufferu sériového kanálu.

Metoda pro odesílání dat, vytváří ze vstupních dat pakety a stará se o jejich odeslání, včetně vyhodnocení příchozího status paketu, je-li vyžadován. Počet opakování odesílání paketu je definováno konstantou `#MAX_REPEAT_SEND.` Po vypotřebování všech pokusů, pro odeslání dat, vrací funkce send status `#ERR_COM_NOTRESPOND.`

Time-out při příjmu je nastaven v jednotkách milisekund, pokud do této doby nedorazí data, je nastaven příznak Time-out a metoda pro příjem dat vrací statut `#ERR_COM_NOTRESPOND`.

Obě metody pro příjem i pro vysílání dat běží v jiném vlákně než hlavní aplikace. Aplikační program tedy během procesu příjmu a odesílaní není blokován.

Třída serialcomm využívá třídu crc16 pro výpočet CRC nad vysílanými daty.

5.4.4 Konfigurace programu – Configuration.java

Třída má úkolem načíst a dekódovat konfigurační soubor „tfk_main.ini“ `#FILENAME_INI`, ve kterém je uložena konfigurace programu. Lze tak konfigurovat jména sériových portů, na které se bude připojovat aplikace.

Aplikace také umí načíst nastavení o konfiguraci sériových portů ze souboru Settings.inc, který je dostupný v projektové cestě FITkitu v adresáři base.

Součástí konfiguračního souboru je také možnost zapínat a vypínat jednotlivé testy včetně nastavení jména souboru a aktivace zápisu do výstupního souboru.

Konfigurační soubor musí být součástí adresáře odkud se spouští aplikace, v případě, že soubor není nalezen, je vytvořen nový.

5.4.5 Vizualizace a ovládání testů – console_test.java

Hlavní program, obsahuje statickou třídu `main`. Třída komunikuje s mikrokontrolérem, spouští

jednotlivé testy a zobrazuje výsledky. Jednotlivé činnosti lze charakterizovat do těchto bodů:

- inicializace a vytvoření instancí tříd,
- načtení konfigurace programu,
- otevření komunikačních portů,
- ověření komunikace s mikrokontrolérem,
- dle potřeby poslat mikrokontroléru konfigurace pro FPGA,
- spuštění manuálního nebo automatického testu dle výběru uživatele,
- vizualizaci výsledků testování na obrazovku anebo do souboru (je-li povolen).

Vizualizace výsledků je prováděna na standardní textový výstup v konzoli. Uživatel ovládá program pomocí klávesnice, ovládání a spouštění aplikace je popsáno v kapitole 8 - Aplikace

6 Implementace aplikace v FPGA

Při implementaci aplikace v FPGA odpadl krok výběru programovacího jazyka a kompilačního prostředí. Jako doporučený a používaný programovací jazyk pro popis chování na projektu FITkit je jazyk VHDL. Pro překlad zdrojových souborů se používá produkt Xilinx ISE [1]. Pro studium programovacího jazyka VHDL jsem využíval informace uvedené v publikaci [26]

Na projektu FITkit v rámci spolupráce studentů, doktorandů a ostatních pracovníků vznikla řada předpřipravených kódů, funkčních aplikací a řadičů, které významným způsobem pomohly při implementaci testovací aplikace v FPGA.

V úvodní části kapitoly je popsáno rozšíření systému kompilace zdrojových souborů pro vytvoření binárních souborů pro testovací aplikaci.

Popis implementace je rozdělen na dvě části, v první části je vysvětlena implementace aplikace pro testování FPGA. V další části se popis zaměřuje na aplikaci pro testování periferii.

6.1 Řízení kompilace zdrojových souborů

Kompilace zdrojových souborů na projektu FITkit se provádí systémem Makefile. Pro komponenty navržené na obrázku 12. na straně 33 byl vytvořen adresář `comp`, kde každý blok je reprezentován svým adresářem.

V adresáři `comp` lze najít soubor `components.inc`, který vkládá do Makefile zdrojové soubory uvedené v jednotlivých adresářích komponent. Pro každou komponentu je třeba definovat cestu ve tvaru

```
xxx_BASE = adresar_komponenty_xxx

# a příkaz pro vložení zdrojových souborů
include $(xxx_BASE)/xxx_pkg.inc
```

V každém adresáři komponenty jsou uloženy dvě architektury, jedna prázdná, tzv. empty a druhá určená pro plně funkční chování. Pokud chceme kompilovat plně funkční architekturu, musíme definovat proměnnou prostředí ve tvaru

```
xxx_ARCH          = FULL

# příklad pro komponentu LED(FULL) a RS232(EMPTY)
LED_ARCH          = FULL
RS232_ARCH        = EMPTY
```

Takto navržený komponentní systém umožňuje kombinovat komponenty mezi různými designy. Pro implementaci testů jsou zapotřebí dvě rozdílné konfigurace, u kterých je architektura vložených komponent rozlišena dvěma soubory `components_fpga.inc` a `components_ifc.inc`. Ty obsahují definice proměnných prostředí a tedy ovlivňují překládanou architekturu komponent.

Tyto konfigurační soubory jsou využívány Makefile souborem umístěným v adresářích `top_fpga` a `top_ifc`, kde je umístěn i zdrojový kód `top_level.vhd` pro každou architekturu. O úroveň výše, v adresáři top je také soubor Makefile, který řídí kompilaci jednotlivých testovacích designů.

6.2 Aplikace pro testování FPGA

Je založena na architektuře uvedené v kapitole 4.6.1. Pro vytváření sítě registrů byl použit příkaz for ve spolupráci s příkazem generate. Tento příkaz generuje (duplikuje) přiřazení, bloky nebo i celé komponenty.

Pro duplikaci byl navržena komponenta test_block, která se generuje po čipu. Jejím rozhraním je signál reset, hodinový vstup, vstup a výstup dat. V případě dalšího rozšíření může být komponenta nahrazena libovolně komplexnějším objektem.

Pro propojení byl použit dlouhý vektor dat, kde nultý prvek je spojen s prvním registrem v řetězu, jeho vstup je přiřazen výstupnímu signálu z adresového dekodéru a poslední prvek pole je spojen se vstupním signálem dekodéru.

Architektura obsahuje i instanci ID komponenty a instanci komponenty (řadič a adresový dekodér) pro test P3M sběrnice.

Rozlišení přiřazení signálu reset na sběrnici P3M je vyřešeno dvěma jednoduchými komponentami, jejíž vstupem je celá sběrnice P3M a výstupem je sběrnice P3M_I a signál RESET. Šířka sběrnice P3M_I je o jeden bit menší než sběrnice P3M, signál RESET je vybrán z patřičného bitu sběrnice. Přiřazení správné komponenty správné aplikaci je rozlišeno na úrovni kompilace zdrojových souborů pomocí řízení zdrojových souborů uvedených v předchozí kapitole.

SPI kontrolér a SPI adresový dekodér je součástí řadičů dostupné v knihovně FITkit.

6.3 Aplikace pro testování externích rozhraní

Je založena na architektuře uvedené v kapitole 4.6.2. Na implementaci testovací aplikace v FPGA byly z knihovny FITkit použity tyto řadiče:

Řadič	Cesta v rámci $BASE adresáře	**Řadič**	Cesta v rámci $BASE adresáře
SPI	units/spi_ics	RS232	ctrls/rs232
Keyboard	ctrls/keyboard	PS2	ctrls/ps2
SDRAM	ctrls/sdram	VGA	ctrls/vga
LCD	ctrls/lcd		

Zbývající řadiče pro sběrnice X-bus a P3M musely být naprogramovány. Jednalo se však o pouhé registrování pinů a jejich zpřístupnění nadřazené entitě (adresový dekodér). Řadič pro sběrnici P3M je stejný jako u předchozí architektury, liší se jen přiřazením vnitřních signálů.

Ke všem řadičům je připojen adresový dekodér. Adresy umístění jednotlivých komponent v adresovém prostoru SPI jsou uloženy `package` v souboru `addr_space.vhd` v adresáři `comp`. Linkuje se k oběma testovacím aplikacím.

Rozsah adres použitý pro každou komponentu se liší podle šířky dat, které bylo třeba předávat řadiči. Nultá adresa každé komponenty je vyhrazena pro lokální identifikaci. Komponenty umožní tak procesoru ověřit, zda architektura komponenty je prázdná či nikoliv.

Pro implementaci VGA byly navíc použity komponenty clk_gen a komponenty pro komunikaci po sběrnici SPI.

Na sběrnici AFBUS byl připojen řadič RS232, sběrnice se tak netestuje ve 12bitovém FIFO režimu, ale jen jako sériový asynchronní kanál. Test tak nevyžaduje připojení externí paměti. V budoucnu může být při doplnění paměti nahrazen sofistikovanějším testem celé sběrnice.

7 Implementace aplikace v µC

Implementace pro mikrokontrolér byla vytvořena v programovacím jazyku C a a překládána do spustitelné formy pomocí volně dostupného prostředí mspgcc. Podobně jak u FPGA byla použita řada funkcí a příkladů, které výrazným způsobem pomohly při vytváření kódu aplikace.

Programová dokumentace byla vytvořena za pomocí nástroje doxygen a překlad zdrojových souborů do spustitelné formy se provádí systémem Makefile. Zdrojové soubory, včetně souboru Makefile se nachází v adresáři src/sw.

Kapitola je rozdělena na části popisující jednotlivé části implementace. První část je zaměřena na popis knihovny libfitkit, která výrazným způsobem zjednodušila implementaci testovací aplikace. V první části lze také najít soupis použitých řadičů z této knihovny. Kapitola pokračuje představením schématu testovací aplikace a popisem činností hlavního programu a hlavních bloků testovací aplikace.

Při implantaci zdrojových kódů jsem čerpal informace z publikace [27].

7.1 Knihovna libfitkit

Knihovna libfitkit, poskytuje na projektu FITkit základní funkce pro snazší používání mikrokontroléru a komunikaci MCU s komponentami umístěných na FITKitu nebo PC. Poskytuje tak základní funkce nezbytné pro interakci µC s periferiemi, s pamětí FLASH, FPGA a apod.

Knihovna obsahuje také funkce pro komunikaci s terminálem na PC, který však v implementaci testovací aplikace nebyly využity.

Knihovna libfitkit je rozdělena na dvě části a to na podpůrné funkce pro činnost mikrokontroléru a sadu knihovních funkcí, pojmenovaných jako ovladače pro řízení připojených periferií. Ovladače, umístěné v adresáři `base/sw/libs` jsou napsané pro:

- dekodování znaku z maticové klavesnice – keyboard_4x4.c,
- podproru sériového kanálu na FPGA – fpga_rs232.c,
- podporu ovládání a řízení LCD – display.c,
- podporu přerušovacího kanálu – fpga_interrupt.c.

Programové části pro podporu programování FPGA, komunikaci s FLASH, řízení SPI, nastavení časovačů jsou umístěny v adresáři `sw/libfikt.`

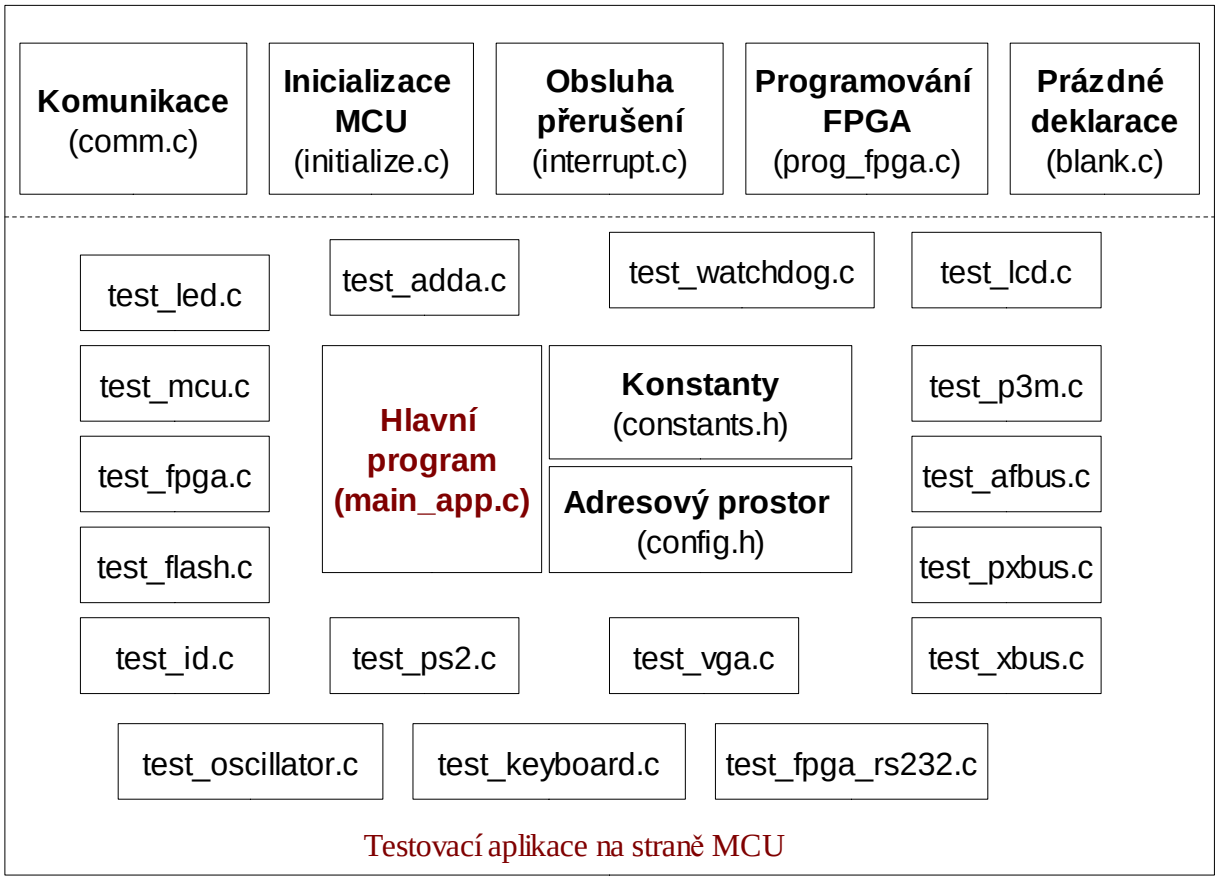
Jednotlivé řadiče poskytují funkce pro přístup a ovládání hardware přes sběrnici SPI. Což

vyhovuje navrženému schématu architektury testovací aplikace. U všech řadičů, lze také měnit bázové adresy umístění jednotlivých adresových dekodéru.

Implementace zbývajících funkcionality pro paměť DRAM, sběrnice X-BUS a P3M, rozhraní PS2 a VGA musely být naprogramovány odděleně. Z větší části jsem však vycházel z aplikačních kódu uvedených v adresáři /apps.

7.2 Schéma aplikační části na straně mikrokontroléru

Schéma aplikační části lze rozdělit na dvě části. První část se zabývá funkcionalitou nezabývající se testováním. Jsou to všechny bloky v horní části schématu testovací aplikace MCU na obrázku 16.



Obrázek 16: bloky v testovací aplikaci na straně MCU

Do této skupiny patří řešení inicializace portů a samotného mikrokontroléru (`initialize.c`), funkce pro komunikaci s PC (`comm.c`), programování FPGA (`prog_fpga.c`) , obsluha přerušení (`interrupt.c`) a blok označený jako (`blank.c`).

Druhá skupina se zabývá implementací testovacích metod a zpracování výsledků z testování. Jsou to všechny bloky, jejíž soubory mají prefix test_.

Posledními bloky, umístěné uprostřed schématu, jsou bloky Hlavní program (main_app), soubor s konstantami (`constants.h`) a adresový prostor (`config.h`). Posledně dva zmíněné hlavičkové soubory jsou obsaženy (připojeny) ke všem testovacím blokům. Soubor `constants.h` obsahuje konstanty, makra a definice proměnných. Soubor `config.h` obsahuje adresy komponent umístěných na FPGA v adresovém prostoru SPI. Tento soubor je automaticky připojen překladovým systémem i k ovladačům v knihovně libfit,

Soubor prázdných deklarací (`blank.c`) obsahuje prázdné deklarace funkcí, které musejí být definovány pro úspěšný překlad aplikace.Na tyto funkce existují totiž externí odkazy z knihovny FITkitu. Jedna se o funkce, které se používají pro obsluhu terminálu.

7.2.1 Hlavní program – main_app.c

Činnost hlavního programu lze resetu procesoru shrnout do těchto bodů:

- Inicializace datových struktur a portů.
- Nakonfigurování obou sériových rozhraní pro komunikaci SPI a PC.
- Provedení kontroly, zda FPGA obsahuje testovací design (podle ID komponenty)
  - pokud ne, pokusí mikrokontrolér konfiguraci FPGA nahrát z FLASH paměti nebo čeká na připojení PC (bliká D6).
- Reset FPGA a test nahrané konfigurace, pokud odpovídá konfiguraci pro testování periferií
  - tak µC otestuje zda není stisknuta žádná klávesa na maticové klávesnici, pokud je, vyřadí ovládání aplikace z klávesnice z provozu (předpokládá se, že uživatel během resetu nestiskl žádnou klávesu a tudíž tak klávesnice obsahuje vady, které by mohly negativně ovlivňovat chování aplikace)
  - ohlásí se na LCD zobrazením zprávy obsahující název aplikace a verze firmware v FPGA a v mikrokontroléru
- V hlavní smyčce čeká mikrokontrolér na příkazy z PC nebo je-li povoleno ovládání maticové klávesnice nabízí na LCD možnosti ovládání a menu pro testování.
- Součásti ovládání z PC je i možnost uložení konfigurací pro FPGA do FLASH paměti, které poslouží pro testování FITKitu i bez připojeného PC.

7.2.2 Programovaní FPGA – prog_fpga.c

Hlavní části jsou dvě rutiny, které programují FPGA. Jedna přijímající data přes sériový kanál z PC a druhá, která programuje FPGA přímo z FLASH paměti. Funkcionalita obou metod je převzata z knihovního souboru `flash_fpga.c`, byly však odstraněny volání na funkce pro posílaní řetězců na

terminál.

Součásti tohoto souboru je funkce pro identifikaci, zda ve FLASH paměti je uložena konfi-gurace pro testování FPGA a dále i rutiny pro obsluhu uložení konfigurací do FLASH paměti.

7.2.3 Obsluha přerušení – interrupt.c

Zpracovává přerušení přijaté z FPGA. Zdrojem přerušení jsou sériové kanály a řadiče PS2, které nastavují v případě aktivity přerušení. Řadič přerušení pro identifikaci stisknuté klávesy na maticové klávesnici nebyl využit. Důvodem je chybějící propojení pro signál informující o přerušení na starších verzích FITkitu mezi FPGA a MCU, která by omezila možnost ovládat aplikaci touto klávesnici.

Pro správnou funkčnost identifikace je nutné propojit pin 5 konektoru JP10 a pin 26 konektoru JP9. Testovací destičky pro kontrolu konektoru JP10 a JP9 obsahují drátovou propojku, která toto propojení vytváří.

7.2.4 Inicializace MCU – intitialize.c

Modul řeší inicializaci datových struktur a správné nastavení portů, které je klíčové pro základní chování mikrokontroléru po resetu.

Obsahuje veřejnou funkci `initialize_mcu()`, spouštěnou po restartu mikrokontroléru, která postupně spouští inicializační funkce jednotlivých modulů. Každý modul může obsahovat dvě funkce. Jména funkcí jsou vytvořena podle jména modulu ke kterému je přidán suffix `init_data` a `init_ports` (například `ps2_init_data` nebo `adda_init_ports`).

Funkce `*_init_data` inicializuje svoje vlastní datové struktury, `*_init_ports` inicializuje porty a uvádí periferii do základního režimu (platí i pro periferie připojené k FPGA).

V modulu initialize se také nachází inicializační kód provedený po naprogramování FPGA.

7.2.5 Další moduly systému

- Comm.c – Komunikace - Obsahuje funkce pro vytváření a zpracování příchozího paketu z PC
- test_fgpa.c – testuje FPGA konfigurací pro testování FPGA, výsledek testování je zaslán aplikaci po sériové lince. V této konfiguraci není dostupný LCD display, tudíž výsledek testu je signalizován LED na umístěnými na FITkitu. V případě nalezené chyby blikají LED D5 a D6 střídavě s frekvencí 1Hz.

  Činnost testování v případě nalezení chyby je pozastavena do příchodu řídící informace ze sériového kanálu nebo resetu mikrokontroléru.

  Funkcionalita dalších bloků pro testování periferií na FITkitu odpovídá popisu uvedeného v kapitole 4.7. Výsledky jsou zobrazovány na LCD a jsou také zasílány pro vizualizaci PC.

8 Aplikace

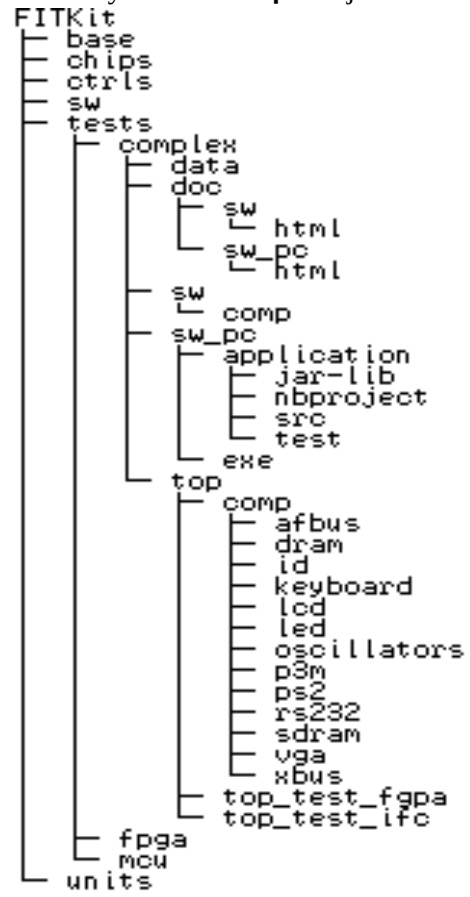
Cílem této kapitoly je shrnutí adresářové struktury testovací aplikace, proces kompilace a jejího spuštění . Kapitola obsahuje popis konfiguračního souboru aplikace a příklady spuštění.

8.1 Adresářová struktura

Při vytváření testovací aplikace byl kladen důraz na integraci celé aplikace do adresářové struktury projektu FITkit. Jako adresář pro umístění aplikace byl vytipován adresář `tests`. V tomto adresáři se již vyskytovaly adresáře pro samostatné testování FPGA a mikrokontroléru. Pro nekonfliktnost s již existujícími soubory byl vytvořen podadresář `complex`, který je kořenovým adresářem celé testovací aplikace.

Stromová struktura SVN (odpovídá revizi 199) včetně zařazení testovací aplikace je zobrazena na obrázku 17.

Kořenový adresář **`complex`** je sestaven z adresářů jejíž význam je následující:



Obrázek 17: SVN struktura

`data` – obsahuje datové soubory s popisem VGA obrazců a uživatelských fontů pro LCD.

`doc` - obsahuje automaticky vygenerovanou programovou dokumentaci zdrojových souborů pro mikrokontrolér a PC v systému doxygen. Dále adresář obsahuje ovládací příručku programu spuštěného pod PC.

`sw` – adresář se zdrojovými soubory pro mikrokontrolér. V podadresáři **`comp`** jsou umístěny zdrojové soubory s testovacími moduly.

`sw_pc` – projektový adresář pro NetBeans IDE 5.5. Podadresář **`application/src`** obsahuje zdrojové kódy a podadresář **`exe`** `obsahuje` spustitelnou aplikaci pro PC, včetně potřebných knihoven, nezávislých na operačních systému.

`top` – je adresář pro umístění zdrojových souboru pro FPGA, Podadresář comp obsahuje řadiče jednotlivých bloků uvedených v kapitole 4.6.2.

8.2 Kompilace testovací aplikace

Kompilace zdrojových souborů byla integrována do jednoho Makefile, umístěného v kořenovém adresáři (`tests/complex`), který nabízí tyto činnosti (cíle):

- **`make clean`** – vymaže překladové soubory (kromě aplikace pro PC)
- **`make all`** – zkompiluje všechny zdrojové soubory,
- **`make fpga`** – zkompiluje zdrojové soubory pro FPGA,
- **`make mcu`** – zkompiluje zdrojové soubory pro mikrokontrolér,
- **`make load`** – nahraje do mikrokontroléru zkompilovaný binární soubor,
- **`make run_app`** - spustí aplikaci na PC,
- **`make test nebo make run`** – spustí cíle clean, all, load a run_app,
- **`make nebo make help`** – zobrazí stručnou nápovědu a seznam činností tohoto Makefile.

Činnost tohoto Makefile je založena na spouštění jednotlivých Makefile souborů umístěných v podadresářích aplikace. Jedná se tedy spíše o spouštěcí skript, než o využívání schopností překladového systému Makefile.

Jednotlivé Makefile soubory umístěné v podadresářích již odpovídají struktuře, která se na projektu FITkit používá. Kompilaci lze provádět lokálně za pomocí příkazů (make clean, make all, make load).

Překlad zdrojových souborů aplikace pro PC, lze překlad zdrojových souborů vytvořit v Netbeans IDE (po načtení projektu v menu Built-> Clean and Build Main Project) nebo kompilací za pomocí Makefile souboru umístěného v adresáři sw_pc (vyžaduje nainstalován překladač Javy JDK).

8.3 Spuštění aplikace

8.3.1 Příprava na testování

K samotné přípravě testování FITkitu je zapotřebí:

1. J13, J14, J15, J16, J17. Při komunikaci s PC i J8, J9.
2. Klávesnici PS2 (popřípadě dvě), VGA monitor, D-SUB9 konektor v Loopback zapojení, propojovací kablík pro ad/da převodníky a testovací destičky pro sběrnice Xbus a PxBus konektory JP10 a JP9.
3. V případě problému s napájením z USB portu i externí zdroj +5V/1A
4. Aktualizovaný soubor tfk_main.ini,

5. Kompletní adresářovou strukturu FITkitu pro překlad souborů anebo zkompilované kódy všech částí aplikace, dále pro překlad souborů prostředí msgcc a prostředí Xilinx ISE
6. Nainstalované ovladače USB/UART převodníku [20],
7. nainstalované JRE (Java Runtime Enviroment), http://java.com/

Body 1 až 3 jsou minimálním předpokladem pro testování FITkitu, když není vyžadováno ovládání z PC a ve FLASH paměti jsou uloženy konfigurace pro FPGA.

8.3.2 Konfigurační soubor tfk_main.ini

Pro navázání komunikace mezi FITkitem a PC je zapotřebí uvést jména a čísla sériových portů podle operačního systému a aktuálního zavedení virtuálních sériových portů. Soubor `tfk_main.ini` se nachází v adresáři s přeloženou aplikací. Pokud se v adresáři nevyskytuje, je při prvním spuštění konfigurační soubor vytvořen.

Obsah tohoto konfiguračního souboru je následující:

```
#adresa base adresare v SVN repozitari
base_directory=../../../../

#pouzit base_directory pro nacteni mcu_dev a fpga_dev? true/false
use_base_dir=true

#COMPORT pro MCU
mcu_dev=COM3

# COMPORT pro FPGA
fpga_dev=COM4

log_file= output.log
log_file_enabled=false
```

`base_directory` – definuje kořenový adresář celé SVN struktury projektu FITkit.

`use_base_dir` – tento parametr říká, zda se má aplikace pokusit vyhledat a načíst nastavení portů v adresáři `$(base_directory)/base/settings.inc`

`mcu_dev` a **`fpga_dev`** jsou parametry, pokud není použit soubor settings.inc nebo nebyl nalezen.

Parametr **log_file** parametrizuje jméno souboru pro uložení výsledků testování a parametr **log_file_enabled** povoluje zápis do tohoto logovacího souboru. Zbývající parametry se zabývají zapínám a vypínání jednotlivých testů, zde nejsou uvedeny, v souboru je však uveden konkretní popis.

8.3.3 Spuštění aplikace – komunikace je v pořádku

Testovací aplikace se spouští příkazem **`make run`** spuštěného v kořenovém adresáři **`complex`**. Podle stavu překladu zdrojových souborů se provede jejich kompilace, poté naprogramování mikrokontroléru a spuštění aplikace.

Pokud je spojení s FITkitem v pořádku, zobrazí se v terminálu konsolové aplikace tento text:

```
========================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7

FITKit - Testing program
------------------------

SW (FW) Version: 01.00
MCU (FW) Version: 01.01
FPGA (FW) Version: N/A

....
```

dále aplikace pokračuje v testování periferií. Ovládání programu je popsáno v příručce ovladani_testovaci_aplikace.pdf v adresáři **`doc`** a není součástí této práce.

8.3.4 Spuštění aplikace – komunikační port nebyl nalezen

Hláška informuje uživatele, že komunikační port pro komunikaci s mikrokontrolérem nebyl v systému nalezen. Je nutné napravit informace v tfk_main.ini, popř. v settings.inc

```
========================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
Port:COM6 not in found in system
```

8.3.5 Spuštění aplikace – komunikační port je obsazen

Další obrazovka informuje uživatele, že požadovaný port je obsazen jiným procesem. Je zapotřebí zkontrolovat, zda není v systému spuštěna jiná aplikace, která komunikuje po tomto sériovém portu.

```
========================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
Port:COM1 is used in another process
```

8.3.6 Spuštění aplikace – problém s komunikací s mikrokontrolérem

Poslední obrazovka informuje uživatele, že vypršel čas určený pro reakci mikrokontroléru na počáteční komunikaci ze strany PC. Ověřte, že sériový port je správně přiřazen k mikrokontroléru nebo resetujte mikrokontrolér.

```
========================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
Port:COM1 Communication time-out...
```

9 Závěr

Tato práce se zabývala testováním výukové platformy FITkit. V práci byl popsán princip testování jednotlivých hardwarových komponent. Pozornost byla věnována zejména návrhu metod a architektury robustní aplikace testující platformu FITkit.

Přínosem této práce je vytvoření modulárního návrhu testovací aplikace, který umožňuje přidávat a modifikovat jednotlivé moduly testující hardwarové prostředky nejen podle aktuálních trendů a požadavků na testovací nástroje, ale i další vývoj této výukové platformy.

Testování bylo zaměřeno na běžné, předpokládané používání FITkitu jako platformy určenou pro výuku na fakultě informačních technologií. Z tohoto důvodu byl vývoj aplikace věnován na využití prostředků se kterými se FITkit bude používat. Mezi ně patří bezesporu externí PS/2 klávesnice a VGA monitor. Potřeba dalších externích zařízení pro testování byla zredukována na jednoduchý hardware, jako je loopback zapojení sériového konektoru a propojovací kablík pro AD/DA převodníky. Posledním externím hardwarovým prostředkem sloužící k testování platformy FITkit je sada destiček, zapojující se do konektorů JP9 a JP10.

Výsledkem této práce je aplikace, která je schopna nejen otestovat hardwarové prostředky umístěné na FITkitu,ale i informovat uživatele o výsledku testování a připadnou lokalizaci chyb.

Testování FITkitu probíhá ve větší míře automatizovaně, zvýšená interakce uživatele při testování platformy se vyskytuje při testování externě zapojené klávesnice zapojené jednak do vstupních konektorů PS2 ale i na maticovou klávesnici umístěné na platformě FITkitu.

Vyhodnocení výsledku testování u periferií jako je klávesnice, LCD, VGA musí uživatel provést vizuálně. U plně automatizovaných testů je uživatel o výsledku informován přímo pomocí FITkitu nebo na PC ve spuštěné aplikaci, která je také součástí této práce.

Návrh testovací aplikace byl koncipován tak, aby bylo možné FITkit otestovat i bez připojení k PC, využívá se faktu, že paměť FLASH může uchovávat konfiguraci pro FPGA i po ztrátě napájení, kde u FPGA se konfigurace čipu ztrácí. Využitím této myšlenky je rozšíření aplikace, která v omezené formě testuje platformu a řídí se pomocí maticové klávesnice se zobrazováním výsledků testování na LCD a nemusí komunikovat s PC.

Zdrojové kódy jsou přeložitelné v open-source nástrojích, nezávislých na operačním systému, včetně prostředků pro programování konfigurovatelných prvků. Zůstala tedy zachována myšlenka konceptu open-source a open-core aplikace.

Splnění bodů zadání

Seznámení se s architekturou výukové platformy je nutné pro pochopení problému analýzy jejího testování

Návrhem testování jednotlivých hardwarových komponent se zabývala kapitola 3, předcházela ji kapitola druhá, která vysvětlovala terminologie testování, o kterou se opírá zbývající část práce. Čtvrtá kapitola představila architekturu testovací aplikace, která byla prakticky realizována a jednotlivě popsaná v kapitolách 5,6,7 a 8.

Možná rozšíření a další vývoj

Možná rozšíření a další vývoj se může ubíhat dvěma směry. Testovací aplikaci lze rozšířit o grafickou reprezentaci výsledků testování. Druhý směr vychází z faktu modulárního návrhu ,kde jednotlivé testovací metody mohou být rozšířeny o novou funkcionalitu a to i v případě, že část hardwarového vybavení se zamění za jiný výkonnější, novější nebo dostupnější prvek.

10 Literatura

[1] Xilinx, Inc., WebPACK ISE 8.2i, produkt dostupný na URL: http://www.xilinx.com/ise/logic_design_prod/webpack.htm (prosinec 2006)

[2] The GCC toolchain for the Texas Instruments MSP430 MCUs, produkt dostupný na URL: http://mspgcc.sourceforge.net/index.html (prosinec 2006)

[3] CAMEA s.r.o. & R.B.E., Brno, FITkit - Block Diagram, 2005, Dokument dostupný na URL: http://merlin.fit.vutbr.cz/FITkit/docs/pdfs/hw_block.pdf (prosinec 2006), dokument lze také najít jako přílohu A. tohoto dokumentu.

[4] Dot-matrix liquid crystal controller, Hitachi/Renesas, 2003, dokument dostupný na URL:http://eu.renesas.com/media/products/lcd/discontinued_products/d_e780u.pdf (prosinec 2006)

[5] Lažanský, R.: Maticové LCD Moduly, Radioplus s.r.o. , 1999, dokument dostupný na URL: http://www.radioplus.cz/clanky/pdf/lcd.pdf (prosinec 2006)

[6] Vašíček, Z.: Firmware pro ovládání LCD Displeje přes SPI, 2006, produkt dostupný na URL: http://merlin.fit.vutbr.cz/FITkit/docs/firmware/20060227lcd.html (prosinec 2006)

[7] Markovič, J.: Firmware pro ovládání klávesnice přes SPI, 2006, produkt dostupný na URL: http://merlin.fit.vutbr.cz/FITkit/docs/firmware/20060226key.html (prosinec 2006)

[8] Hynix, SDRAM 4x2Mx8, CMOS, TSOP, dokument dostupný na URL: http://www.ic-on-line.cn/IOL/datasheet/GM72V66841ET_441604.pdf (prosinec 2006)

[9] Čapka, L.: Firmware/řadič SDRAM, 2006, produkt dostupný na URL: http://merlin.fit.vutbr.cz/FITkit/docs/firmware/sdramctrl.html (prosinec 2006)

[10] Čapka, L.: Firmware k VGA rozhraní, 2006, produkt dostupný na URL: http://merlin.fit.vutbr.cz/FITkit/docs/firmware/20060931vga.html (prosinec 2006)

[11] Texas Instruments, Inc., MSP430F168 Mixed Signal Microcontroller, dokument dostupný na URL: http://focus.ti.com/lit/ds/symlink/msp430f168.pdf (prosinec 2006)

[12] Atmel Corporation, AT45DB021B 2-megabit 2.7 Volt Only DataFlash, dokument dostupný na URL: http://www.atmel.com/dyn/resources/prod_documents/doc1937.pdf (prosinec 2006)

[15] Toutounchi, S. , Lai, A. : FPGA Test and Coverage in ITC 2002, Baltimore, USA, IEEE Computer Comunity, strana 599-607. Dokument dostupný na URL:

http://csdl.computer.org/dl/proceedings/itc/2002/7543/00/75430599.pdf (prosinec 2006)

[16] Lu, S., Chen, Ch. : Fault Detection and Fault Diagnosis Techniques for Lookup Table FPGA's, 11th Asian Test Symposium, 2002, IEEE Computer Comunity, strana 236-242. Dokument dostupný na URL: http://csdl.computer.org/dl/proceedings/ats/2002/1825/00/18250236.pdf (prosinec 2006)

[14] Novák, O. aj. : Handbook Of Testing Electronic Systems, ČVUT 2005, počet stran: 395

[13] Bushnell, Michael L., Agrawal Vischwani D. : Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits, Springer – Verlag 2000, počet stran: 734

[17] Morton,G.,Venkat K. : MSP430 Competitive Benchmarking, Texas Instruments, 2005. Dokument dostupný na URL: http://focus.ti.com/general/docs/techdocsabstract.tsp?abstractName=slaa205b (prosinec 2006)

[18] Gizopoulos, D. aj. : Embedded Processor-Bassed Self-Test, Kluwer Academic Publisher, Boston, 2004, počet stran: 217

[19] Filip, T. : Testování spojů na desce Combo6 [bakalářská práce], 2005, Vysoké Učení Technické, Brno

[20] Future Technology Devices International Ltd., Dual USB UART/FIFO IC , 2006 dokument dostupný na URL: http://www.ftdichip.com/Products/FT2232C.htm (prosinec 2006)

[21] IAR, Embedded Development Tools, produkt dostupný na URL: www.iar.com (květen 2007)

[22] Doxygen, Source code documentation generator tool, produkt dostupný na URL: www.doxygen.org (květen 2007)

[23] Herout P. : Učebnice jazyka JAVA. České Budějovice, KOPP, 2003

[24] NetBeans IDE, open-source Integrated Development Environment, produkt dostupný na URL: www.netbeans.org (květen 2007)

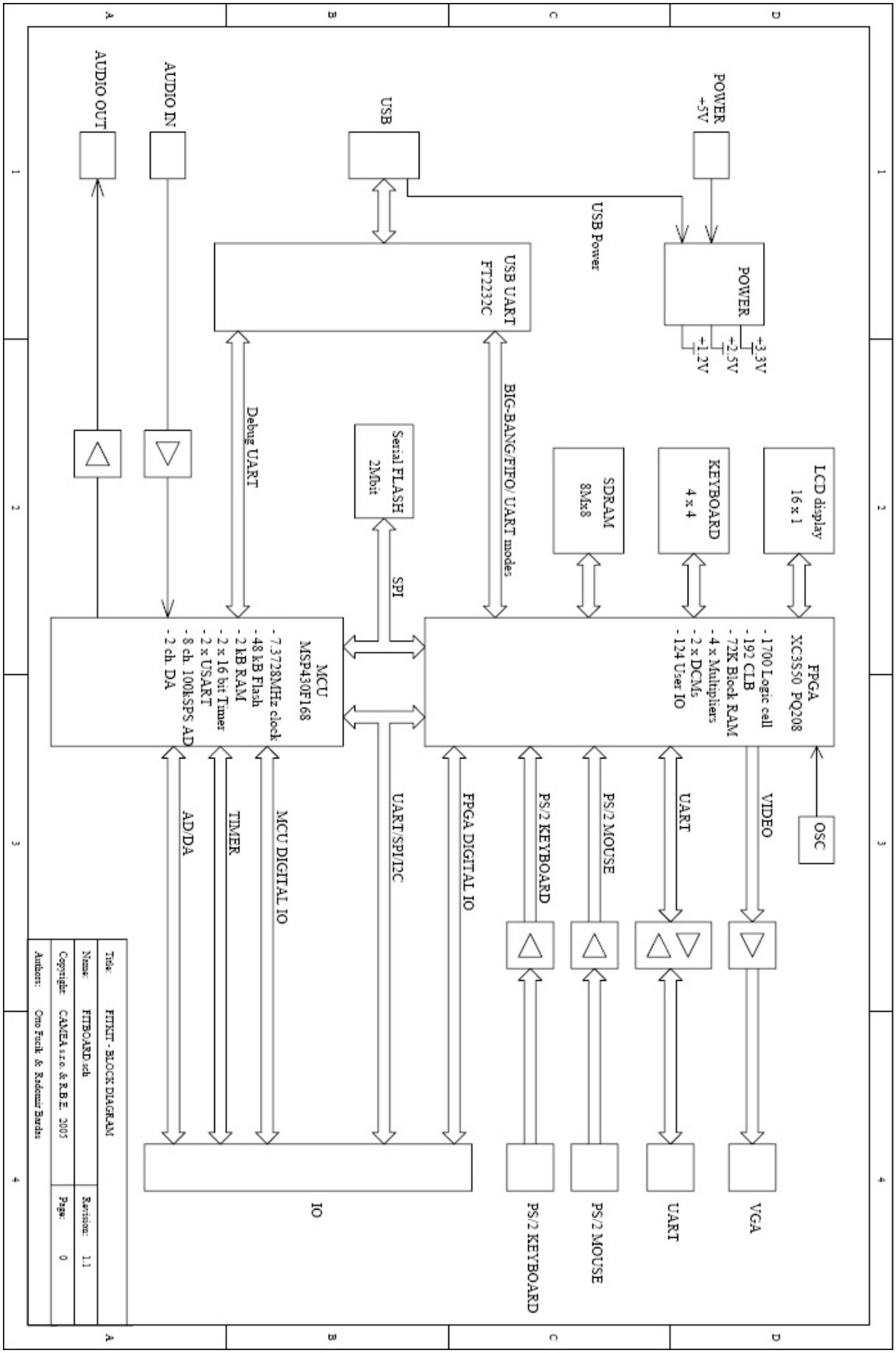
[25] RXTX, Serial and Parallel I/O Libraries supporting Sun's CommAPI , produkt dostupný na URL: www.rxtx.org, (květen 2007)

[26] Douša, J.: Jazyk VHDL, Vydavatelství ČVUT, Praha, 2003

[27] Herout P. : Učebnice jazyka C. České Budějovice, KOPP, 1998

11 Přílohy

A. Blokové schéma FITkitu:



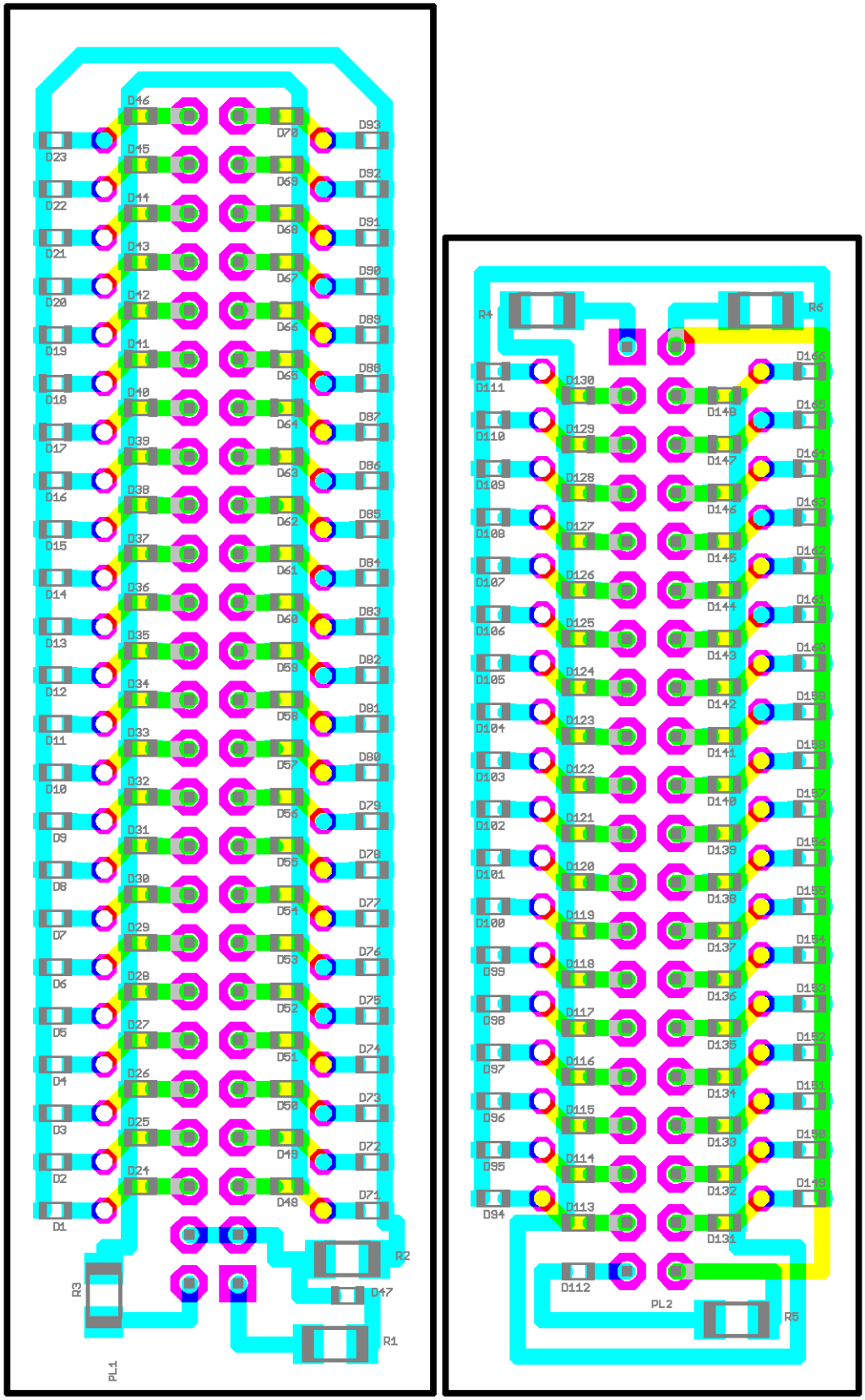
B. Použití knihovny RxTx pro komunikaci po sériové lince

Příklad pro operační systém windows. Hlavní program otevře port COM1, pošle znak 'A' a pokusi se blokujicím čtením načíst byte ze seriové linky, na konci uzavře seriový port

```
import gnu.io.CommPort;
import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;

import java.io.FileDescriptor;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class SerialExample
{
    public SerialExample()   {
        super();
    }
     private InputStream in;
     private OutputStream out;
     private SerialPort serialPort;

    void open( String portName ) throws Exception
    {
       CommPortIdentifier portIdentifier =
                    CommPortIdentifier.getPortIdentifier(portName);

        if ( portIdentifier.isCurrentlyOwned() )
        { System.out.println("Error: Port is currently in use"); }

        else
        {

        CommPort commPort = portIdentifier.open(this.getClass().getName(),2000);
        if ( commPort instanceof SerialPort )
           {
              // otevri port s temito parametry 57600, 8bitu, 1stopbit, zadna parita
              serialPort = (SerialPort) commPort;
              serialPort.setSerialPortParams(
                                    57600,
                                    SerialPort.DATABITS_8,
                                    SerialPort.STOPBITS_1,
                                    SerialPort.PARITY_NONE);


              in = serialPort.getInputStream();
              out = serialPort.getOutputStream();
           }
           else
           {System.out.println("Error: Only serial ports are handled"+
                               "by this example.");
           }
        }
    }
    void send(byte data)
    {
       out.write(data);                       //zapis znak na seriovy port
    }
```

```
    byte receive(void)
    {
      byte tmp;
       while (in.available()==0) ;            //cekej dokud nemas data
       tmp=in.read();                         //nacti znak
       return (tmp);                          //vrat znak
    }

    void close(void)
    {
       SerialPort.close();
    }

    public static void main ( String[] args )
    {
        SerialExample serial = new SerialExample();

        try
        {
            System.loadLibrary("rxtxSerial");

        }
        catch (java.lang.UnsatisfiedLinkError e)
        {
          System.out.println("Can't load rxtxSerial.dll");
          return;
        }

        try
        {
           serial.open("COM1");
        }
        catch ( Exception e )
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        serial.send(0x65);              //posli znak A
        byte data_in=serial.receive();  //prijmi znak ze serioveho portu

        serial.close();                 //zavri port
    }
}
```

C. Testovací destičky - deska plošného spoje



Obrázek 18: měřítko 3 :1

D. Testovací destičky - schéma