VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EMULÁTOR MALÉHO DOMÁCÍHO POČÍTAČE ZX SPECTRUM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

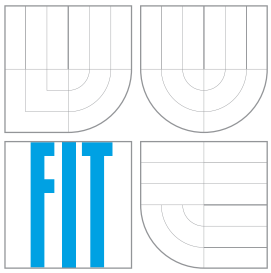
AUTOR PRÁCE
AUTHOR

Bc. PETR ŠIMON

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EMULÁTOR MALÉHO DOMÁCÍHO POČÍTAČE ZX SPECTRUM

ZX SPECTRUM SMALL HOME COMPUTER EMULATOR

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR ŠIMON

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RICHARD RŮŽIČKA, Ph.D.

BRNO 2012

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2011/2012

Zadání diplomové práce

Řešitel: **Šimon Petr, Bc.**
Obor: Počítačové a vestavěné systémy
Téma: **Emulátor malého domácího počítače ZX Spectrum**
ZX Spectrum Small Home Computer Emulator
Kategorie: Návrh číslicových systémů

Pokyny:

1. Seznamte se s architekturou jednoduchého domácího počítače Sinclair ZX Spectrum.
2. Seznamte se s na FIT dostupnými deskami s FPGA (vývojové kity firmy XILINX).
3. Navrhněte emulátor počítače s využitím FPGA a volně šiřitelného jádra T80 (implementace procesoru Z80 ve VHDL). Jako vstupní zařízení emulátoru uvažujte běžnou PC klávesnici a rozhraní SD karet se soubory TAP, ZXS atd., výstupem bude signál VGA.
4. Vyberte vhodnou platformu pro implementaci emulátoru (velikost FPGA, vhodná dostupná vývojová deska) a navrhněte připojení periferií.
5. Implementujte návrh dle bodu 3 a 4 ve VHDL s maximálním využitím volně dostupných zdrojových kódů.
6. Ověřte funkčnost implementovaného zařízení s volně dostupným dobovým softwarem pro platformu ZX Spectrum.

Literatura:

- Dle pokynů vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

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

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

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

Vedoucí: **Růžička Richard, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 19. září 2011
Datum odevzdání: 23. května 2012

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

L.S.



doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Osmibitový počítač ZX Spectrum vznikl před 30 lety. Ve své době zažil obrovský úspěch a dodnes má početnou základnu příznivců, kteří stále vytvářejí nové aplikace a hry. Vznikají však i nová hardwarová rozšíření pro připojení moderních periferií, jako jsou IDE HDD, paměťové karty apod. Tato práce si dává za cíl navrhnout a postavit emulátor počítače ZX Spectrum, který bude založen na moderní technologii FPGA a bude používat moderní periferie jako VGA monitor, SD/MMC paměťové karty apod.

Abstract

Eight-bit computer ZX Spectrum has been created 30 years ago. It was extremely popular in its time and it has many fans till now, which still developing new application and games. There are also many new hardware extensions like IDE HDD driver, SD/MMC memory driver etc. The aim of this thesis is the design and develop of ZX Spectrum emulator, which will be based on modern PFGA technology and it will use modern periphery like VGA monitor, SD/MMC memory cards etc.

Klíčová slova

ZX Spectrum, Clive Sinclair, emulátor, FPGA, FITkit, Z80

Keywords

ZX Spectrum, Clive Sinclair, emulator, FPGA, FITkit, Z80

Citace

Petr Šimon: Emulátor malého domácího počítače ZX Spectrum, diplomová práce, Brno, FIT VUT v Brně, 2012

Emulátor malého domácího počítače ZX Spectrum

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením pana Ing. Richarda Růžičky, Ph.D.

.....
Petr Šimon
21. května 2012

Poděkování

Děkuji vedoucímu mé práce, panu Ing. Richardu Růžičkovi, Ph.D. za jeho velikou ochotu a vstřícnost. Dále děkuji panu Ing. Václavu Šimkovi za pomoc při návrhu a realizaci plošného spoje emulátoru.

© Petr Šimon, 2012.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **4**

2 Historie počítačů ZX Spectrum **6**
2.1 Sir Clive Marles Sinclair 6
2.2 Verze počítačů ZX 7
2.2.1 ZX80 7
2.2.2 ZX81 7
2.2.3 ZX Spectrum 16K / 48K 8
2.2.4 ZX Spectrum + ... 8
2.2.5 ZX Spectrum 128K 8
2.2.6 Pozdější modely ZX Spectrum 9
2.2.7 ZX Spectrum kompatibilní klony 9

3 Architektura počítače ZX Spectrum **10**
3.1 Procesor Z80A 10
3.2 Uspořádání paměťového prostoru 11
3.2.1 ZX Spectrum 16K / 48K 13
3.2.2 ZX Spectrum 128K 13
3.3 Grafický výstup 15
3.3.1 CRT obrazovka 15
3.3.2 Rozklad obrazu na řádky a prokládané řádkování 15
3.3.3 Televizor jako grafický výstup počítače ZX Spectrum 16
3.3.4 Generování obrazu 17
3.3.5 Struktura Video RAM 19
3.4 Klávesnice 20
3.5 Další I/O rozhraní 22

4 Návrh emulátoru počítače ZX Spectrum **23**
4.1 FPGA technologie 23
4.2 Platforma FITkit 24
4.3 Návrh architektury emulátoru ZX Spectrum 25
4.3.1 Procesor 26
4.3.2 Paměť 26
4.3.3 SD/MMC karta 26
4.3.4 Další vstupní a výstupní periferie 27

5 Hardwarové řešení **28**
5.1 PS/2 řadič . . . 28
5.2 Klávesnice . . . 29
5.3 Myš . . . 30
5.3.1 Komunikační protokol PS/2 myši . . . 31
5.3.2 Kempston Mouse . . . 32
5.4 VGA řadič . . . 34
5.5 Grafická jednotka . . . 35
5.6 SPI rozhraní . . . 37
5.7 Řadič SD/MMC karet . . . 38
5.7.1 Komunikační protokol . . . 38
5.7.2 Rozhraní řadiče SD/MMC karet . . . 40
5.8 Procesor T80 . . . 42
5.9 SDRAM řadič . . . 43
5.10 SIMI DMA . . . 44
5.11 Top level entita . . . 46
5.11.1 Generování maskovatelného přerušení INT . . . 46
5.11.2 Fyzický adresový prostor . . . 46
5.11.3 Přepínání běžících procesů . . . 48
5.11.4 Řízení přístupu do paměti . . . 48

6 Softwarové řešení **50**
6.1 Úprava standardní ROM ZX Spectrum 128K . . . 50
6.2 Operační systém ZX Simi OS . . . 51
6.2.1 Knihovna s abstraktní vrstvou nad souborovým systémem . . . 51
6.2.2 Knihovna pro práci se souborovým systém FAT32 . . . 51
6.2.3 Knihovna pro práci s SD/MMC kartou . . . 52
6.2.4 Knihovna pro práci se SIMI DMA řadičem . . . 53
6.2.5 Načítání a spouštění aplikací z SD/MMC karty . . . 53

7 Výsledná fyzická realizace **55**
7.1 Externí paměť FLASH . . . 56
7.2 Spuštění a ovládání emulátoru . . . 57

8 Ověření funkčnosti emulátoru **58**

9 Závěr **59**

A Překlad a zprovoznění emulátoru **63**
A.1 Překlad hardwarové architektury pro FPGA . . . 63
A.2 Překlad systému ZX Simi OS . . . 63
A.3 Tvorba obsahu FLASH paměti . . . 63

B Ovládání operačního systému ZX Simi OS **65**

C Výpis obslužné rutiny LOAD z ROM ZX Spectrum 128K **66**

D Schéma zapojení emulátoru **69**

E Plošný spoj **76**

Kapitola 1

Úvod

ZX Spectrum je legendární 8-bitový počítač od firmy Sinclair, pojmenované po jejím zakladateli Clive Sinclairovi. První počítač ZX Spectrum přišel na trh v roce 1982 a ihned zaznamenal obrovský úspěch, kvůli své „lidové“ ceně a jednoduchosti. Ani po 30 letech tento počítač neupadl v zapomnění a stále pro něj vznikají nové fanouškovské aplikace a hry.

Poprvé jsem se s tímto počítačem seznámil ve svých 14 letech v roce 2002. Během odpolední procházky městem jsem si všiml, že vedle kontejneru leží jakási „zvláštní klávesnice“. V tu chvíli jsem ještě nevěděl, jaký poklad jsem právě objevil. Přesto jsem tuto „klávesnici“ vzal a přinesl si jí domů.

Ukázalo se, že se mi do ruky dostal počítač Didaktik Gama, což je slovenský klon počítače ZX Spectrum. Po zapojení bohužel nefungoval - nezobrazoval se žádný obraz, ale po stisku klávesy přitom vydával zvuky. Po detailnějším prozkoumání se ukázalo, že v televizním modulátoru byl ulomený jeden keramický kondenzátor. Po jeho výměně konečně obraz naskočil a já tak získal svůj první ZX Spectrum kompatibilní počítač.

Psal se rok 2004, když si náš učitel v kroužku elektroniky na Střední průmyslové škole v Jihlavě všiml mé záliby v počítačích ZX Spectrum a daroval mi jeho vlastní počítač Didaktik M. Stal jsem se tedy majitelem již druhého klonu počítače ZX Spectrum.

Protože moje záliba v počítačích ZX Spectrum ani po tolika letech neopadla, rozhodl jsem se, že v rámci své diplomové práce vytvořím svůj vlastní klon tohoto počítače, který bude sestaven z moderních součástek (především FPGA) a používat moderní periferie (VGA monitor, SD/MMC karty…).

Kapitola 2 stručně pojednává o Siru Clive Sinclairovi - muži, který za vším stojí. Také čtenáře informuje o historii počítače ZX Spectrum a jeho různých variantách, které se objevily na trhu. Protože je mým cílem vytvořit emulátor, který bude kompatibilní s počítači ZX Spectrum, popsal jsem v kapitole 3 jejich vnitřní architekturu.

Kapitola 4 stručně popisuje můj návrh emulátoru a také technologie, které jsem se rozhodl k implementaci využít. Z návrhu jsem vycházel při implementaci hardwarové architektury emulátoru v FPGA, popsané v kapitole 5.

Aby bylo možné pracovat s SD/MMC kartami, ze kterých se načítají a spouštějí aplikace, musel jsem také naprogramovat operační systém ZX Simi OS popsaný v kapitole 6, který tuto funkčnost poskytuje.

Po otestování emulátoru na platformě FITkit jsem se rozhodl, že navrhnu vlastní plošný spoj, který bude šitý přímo na míru emulátoru - bude obsahovat všechny potřebné konektory a bude pokud možno co nejmenší a nejjednodušší. Tomuto tématu se věnuje kapitola 7.

Nakonec jsem celý emulátor otestoval pomocí dobových aplikací a her, které jsou nyní

volně dostupné na internetu. Výsledky testování jsou shrnuty v kapitole 8.

Kapitola 2

Historie počítačů ZX Spectrum

2.1 Sir Clive Marles Sinclair

Informace v této kapitole jsou čerpány z [21] a [3].

Počítače ZX Spectrum jsou neodmyslitelně spjaty se jménem Clive Sinclair. Mužem s typickou pleší, brýlemi a plnovousem (obr. 2.1) , známý také jako „strýček Clive“. Řekneme si tedy o tomto muži pár slov.

Narodil se 30. června 1940 nedaleko Richmondu v hrabství Surrey. Jeho otec i strýček pracovali jako techničtí inženýři, což mělo na Clivea značný vliv. Clive neměl ve škole s učením problémy. Dokonce studoval i obory, které ani neměl v rozvrhu. V deseti letech jeho učitel matematiky na základní škole poznal, že už nemá Clivea co naučit a tak ho doporučil na střední školu. Těch prošel hned několik a závěrečné zkoušky složil nejprve na Highgate School a pak speciální zkoušky z fyziky a matematiky na St. Georges.

Matematika ho zajímala zřejmě nejvíce. Jako chlapec si navrhl kalkulátor programovatelný pomocí kartiček z dětské hry. Pro jednoduchost převáděl veškeré výpočty na kombinace nul a jedniček. Teprve později zjistil, že nevymyslel nic nového, ale že objevil již dávno existující dvojkovou soustavu. Kromě matematiky se však zajímal také o elektroniku.

Rozhodl se, že na vysokou školu nenastoupí a místo toho se vše naučí sám. Chtěl vyrábět elektronické výrobky, ale chyběly mu peníze. Pracoval tedy jako redaktor. Navrhoval a sestavoval elektronické obvody, které pak publikoval. Nakonec se mu podařilo sehnat potřebné finance a zakládá firmu Sinclair Radionics Ltd. Pod touto značkou pak mimo jiné prodává svoje minirádio Sinclair Slimline, o které byl obrovský zájem.

Ke konci 70. let se začínají na trhu objevovat první osobní počítače (Atari, Commodore),



Obrázek 2.1: Sir Clive Sinclair s jeho ZX Spectrum 48K [11]

které však byly poměrně drahé (cca. 700 liber). To se Clive rozhodl změnit a tak v lednu 1980 prezentoval na výstavě ve Wembley první model svého superminiaturního počítače ZX-80, který se jako stavebnice prodával jen za 79 liber (to odpovídá 227 librám v roce 2010). Rok poté se na trhu objevil ZX-81. V roce 1982 pak vznikl legendární ZX Spectrum, který se stal možná nejúspěšnějším domácím počítačem té doby.

Titul „Sir“ získal při příležitosti svých 43. narozenin, při nichž byl povýšen do šlechtického stavu. Toto povýšení bylo zdůvodněno tím, že se Sinclairova firma zasloužila o významný rozvoj britské elektrotechniky.

Počátek konce slavné značky Sinclair byl počítač Sinclair QL. Měl být nástupcem ZX Spectra, ale kvůli špatné reklamní kampani a nekompatibilitě s původním Spectrem o něj nebyl příliš zájem. Firmu nezachránil ani ZX Spectrum Plus, který byl rozšířením původního ZX Spectrum. Zájem postupně klesal. Firma přišla o množství financí kvůli dalšímu neúspěšnému projektu - elektrickému vozítku C5. Postupně dochází k zadlužování firmy, kterému nezabrání ani vydání dalšího výborného počítače ZX Spectrum 128K, které však již pochází ze Španělska.

V roce 1986 firma Amstrad zakoupila firmu Sinclair i s právy na použití této obchodní značky a vyrobila modely Spectrum Plus 2 a Plus 3. Sir Clive Sinclair tímto nezmizel zcela ze scény. V současné době se opět zabývá vývojem elektrických vozítek. Jeho nejnovějším produktem je vozítko X-1.

2.2 Verze počítačů ZX

I když se tato práce zabývá především návrhem emulátoru počítače ZX Spectrum 48K (případně ZX Spectrum 128K), pro lepší pochopení problematiky bude vhodné zmínit se i o přímých předchůdcích těchto počítačů.

Informace v této kapitole jsou čerpány z [26, 27, 25, 25, 11]

2.2.1 ZX80

Jak již bylo zmíněno v kapitole 2.1, jednalo se o Sinclairův první počítač. Využíval 8-bitový procesor Z80A firmy Zilog, který pracoval na frekvenci 3,25 MHz. Počítač byl vybaven 1 kB paměti DRAM a 4 kB paměti ROM, která obsahovala interpretr jazyka BASIC a veškeré obslužné rutiny (klávesnice, magnetofon, vykreslování obrazu…).

Grafický výstup byl pouze černo-bílý textový s rozlišením 32 * 24 znaků. Vzhledem k softwarovému řešení zobrazování (pomocí CPU) nebylo možné při výpočtu překreslovat obraz, čímž docházelo k blikání obrazu.

Celý počítač byl zkonstruován pomocí 17 klasických TTL integrovaných obvodů, které byly normálně k dostání na trhu. Jako výstupní zařízení se používala obyčejná televize (stejně jako u všech následujících modelů) a jako vnější paměť se používal běžný magnetofon.

2.2.2 ZX81

Tato verze s sebou přinesla vylepšený BASIC, který byl uložen v 8 kB paměti ROM. BASIC nově podporoval aritmetiku v plovoucí řádové čárce. Počítač však zůstal stále vybaven pouze 1 kB paměti RAM. Ta se však dala (stejně jako u ZX-80) rozšířit pomocí externího modulu.

Zřejmě nejdůležitější změnou však bylo nahrazení všech integrovaných obvodů pouze jediným integrovaným obvodem, tzv. ULA. To přineslo značné snížení spotřeby a snížení



Obrázek 2.2: ZX Spectrum 48K [25]

ceny (49,99 liber, což odpovídá cca 134 librám v roce 2010). Navíc byla ULA schopna provádět zobrazování i během výpočtu CPU.

2.2.3 ZX Spectrum 16K / 48K

Po obrovském úspěchu ZX81 začala Sinclairova firma pracovat na dalším počítači s kódovým označením ZX82. Do světa byl vypuštěn v roce 1982 pod jménem ZX Spectrum (obr. 2.2). Později se pro rozlišení od dalších modelů vžilo označení „gumák“, odkazující na provedení jeho klávesnice. Vyráběl se ve dvou verzích: s 16 kB RAM, nebo s 48 kB RAM, což je oproti 1 kB u ZX81 obrovský skok. To však nebyla jediná významná změna.

Obvod ULA byla výrazným způsobem vylepšen. Nyní již byl schopen sám generovat obraz zcela nezávisle na procesoru, zobrazoval 8 barev se 2 úrovněmi jasu (tedy celkem 16 barev), podporoval blikání a uměl zobrazovat grafiku s rozlišením 256*192 pixelů. Z důvodu zjednodušení hardware a snížení paměťových nároků však měla grafika zásadní omezení. Obrazovka byla rozdělena do 32*24 čtverců o straně 8 pixelů, přičemž v rámci jednoho čtverce bylo možné zobrazovat pouze 2 barvy.

Dále byla zvětšena paměť ROM na 16 kB, která obsahovala vylepšený BASIC a zvuk byl zajištěn pomocí jednoduchého bzučáku, který produkoval jednokanálový zvuk o rozsahu 10 oktáv.

2.2.4 ZX Spectrum +

Objevil se na trhu v roce 1984. Hardware je shodný s předchozí verzí ZX Spectrum 48K. Liší se pouze vzhledem, kvalitnější (plastovou) klávesnicí a přidaným tlačítkem RESET.

2.2.5 ZX Spectrum 128K

Počítač se na trh dostal v roce 1986. Od předchozího modelu ZX Spectrum+ se vzhledem příliš nelišil (obr. 2.3). Z pohledu hardwarového vybavení však šlo o druhou revoluci.

Paměť ROM byla zvětšena na 32 kB. V dolních 16 kB byla uložena lehce upravená verze původního 48K BASICu, horních 16 kB obsahovalo vylepšený editor.

Hlavní změnou ale bylo rozšíření paměti na 128 kB. Vzhledem k tomu, že použitý Z80 CPU má pouze 16-bitovou adresovou sběrnici, dokáže adresovat pouze 64 kB. Pro



Obrázek 2.3: ZX Spectrum 128K [25]

adresování takto velké paměti tedy bylo potřeba požít podporu HW a speciální postup adresování, který bude vysvětlen v kapitole 3.2.2.

Počítač také nově obsahuje tříkanálový generátor zvuku AY-3-8912.

2.2.6 Pozdější modely ZX Spectrum

ZX Spectrum +2

Hardwarově je zpětně kompatibilní se svým předchůdcem, obsahuje však vestavěný magnetofon a porty pro připojení joysticku.

ZX Spectrum +3

Kazetový magnetofon byl nahrazen 3”disketovou mechanikou. Pro její použití bylo potřeba upravit ROM, takže byla její velikost navýšena na 64 kB. Také bylo potřeba upravit stránkování paměti, čímž tento počítač přinesl řadu nekompatibilit s předchozími modely.

ZX Spectrum +2A / +2B

Poslední varianta tohoto počítače. V podstatě se jedná o ZX Spectrum +3, kde je místo disketové mechaniky zpět kazetový magnetofon. Označení A a B rozlišovalo jazykovou verzi ROM (A = anglická, B = španělská)

2.2.7 ZX Spectrum kompatibilní klony

Díky obrovské popularitě počítačů ZX Spectrum se na trhu začaly brzy objevovat různé programově kompatibilní klony. V Československu například firma Didaktik Skalica začala v roce 1987 vyrábět počítače Didaktik Gama, dále pak Didaktik M a nakonec Didaktik Kompakt, který již měl vestavěnou 3,5”disketovou mechaniku.

Obrovské množství klonů pochází také z Ruska. Zmiňme například počítače Pentagon 48/128/512, Moskva 48/128 a jiné.

Kapitola 3

Architektura počítače ZX Spectrum

Jak již bylo zmíněno v kapitole 2.2, všechny počítače řady ZX Spectrum používají procesor Z80A od firmy Zilog. O vše ostatní se pak stará speciální zákaznický obvod ULA. Ten zajišťuje generování video-signálu pro televizor, generování hodin pro procesor, přístup do paměti, obnovování paměti, řadič klávesnice, magnetofonu a interního zvukového generátoru a pochopitelně umožňuje přístup CPU k těmto periferiím tak, aby televizní signál nebyl nikdy přerušen a aby nedocházelo k žádným kolizím na sběrnici.

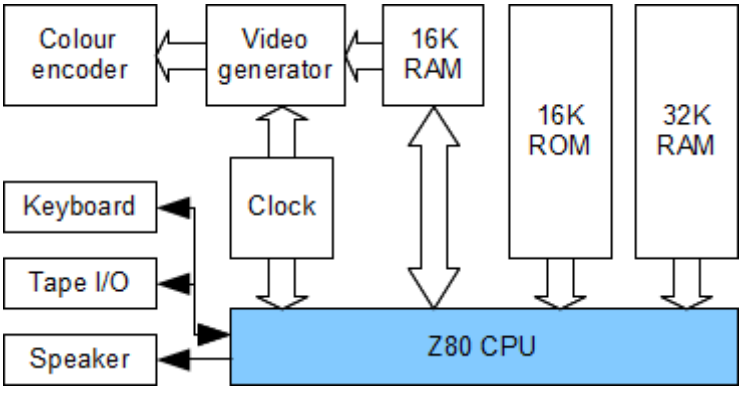
V této kapitole jsem vycházel z informací uvedených v této literatuře: [15] [16]

Blokové schéma počítače je znázorněno na obrázku 3.1. Jádrem počítače je procesor Z80A. Ten je připojen přímo k ULA, která realizuje již zmíněné funkce. V této kapitole se detailně podíváme na architekturu počítače ZX Spectrum a vysvětlíme si, jakým způsobem vlastně ULA funguje.

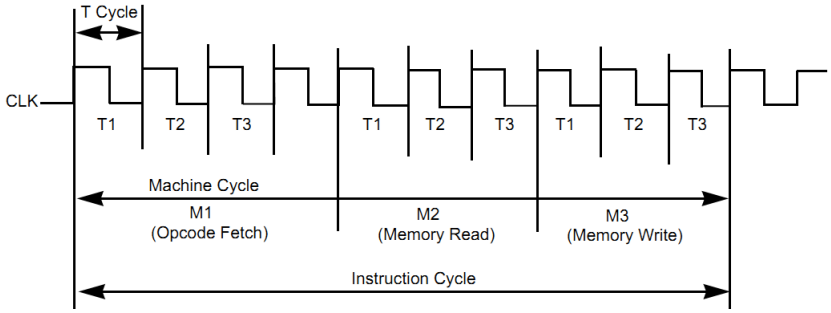
3.1 Procesor Z80A

Procesor Z80 byl vyroben firmou Zilog Inc. v roce 1976 a vychází z procesoru Intel 8080 z roku 1974. Ve firmě Intel ho navrhl Federico Faggin, zakladatel firmy Zilog. Procesor Z80 se pak stal jedním z nejdominantnějších procesorů koncem 70. a začátkem 80. let.

Procesor Z80 je binárně kompatibilní se svým předchůdcem 8080, ale přináší velké množ-



Obrázek 3.1: Blokový diagram počítače ZX Spectrum



Obrázek 3.2: Příklad základního časování CPU [18]

ství rozšíření. Obsahuje rozšířenou instrukční sadu, nové registry, vylepšenou obsluhu přerušení a podporu 16-bitového I/O adresování. Navíc dokáže sám obstarávat refresh dynamických pamětí, čehož bylo využito např. u počítače ZX80 a ZX81. U řady ZX Spectrum se již o obnovování paměti stará ULA.

Pro pochopení mého řešení je důležité čtenáře seznámit s časovými průběhy procesoru. Z80 je procesor typu CISC (Complex Instruction Set Computer) - instrukce jsou tedy komplexnější a jejich provádění trvá různě dlouhou dobu. Na obrázku 3.2 je ukázán obecný časový průběh zpracování jedné konkrétní instrukce. Jeden *instrukční cyklus* (instruction cycle) je rozdělen do několika *strojových cyklů* (machine cycle). První strojový cyklus každé instrukce je vždy *načtení operačního kódu* (opcode fetch) označovaný jako M1. Následující sekvence strojových cyklů, které přesouvají data mezi pamětí nebo I/O rozhraním, se pak již u každé instrukce liší.

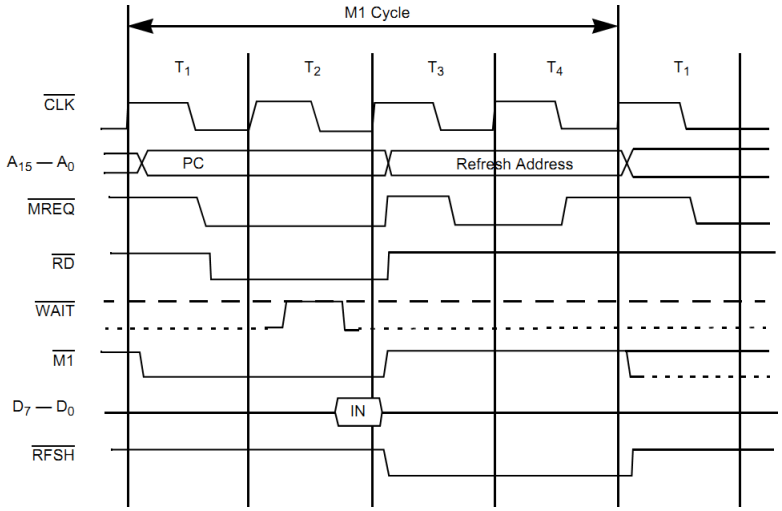
Obrázek 3.3 ukazuje časování během fáze M1 (čtení operačního kódu). Nejdříve je na adresovou sběrnici vystaven obsah registru PC (Program Counter) a pin /M1 se nastaví do nízké (aktivní) úrovně, který indikuje provádění fáze M1. S určitým zpožděním se do nízké (aktivní úrovně) nastaví piny /MREQ a /RD, které indikují požadavek čtení z paměti. V taktu T3 procesor načte data na vstupu a současně začne obnovovat dynamickou paměť. Důležitou vlastností je, že můžeme pomocí signálu /WAIT procesoru indikovat, že data na jeho vstupu ještě nejsou připravena. V takovém případě procesor čeká, dokud signál /WAIT opět nevrátíme do stavu vysoké úrovně a teprve pak procesor vstupní data načte a pokračuje ve výpočtu. Tato vlastnost je velice důležitá, protože přístup do DRAM není okamžitý (obecně trvá více taktů) a kromě procesoru do ní potřebuje přistupovat i ULA (například pro vykreslování obrazu nebo obnovu paměti).

Časování procesoru při běžném čtení a zápisu do paměti je ukázán na obrázku 3.4. Princip je velice podobný jako u fáze M1. To, že procesor chce přistupovat k paměti, je indikováno signálem /MREQ a signály /RD nebo /WR. Opět zůstává důležitá vlastnost, že je možné zápis i čtení pozdržet pomocí signálu /WAIT. Úplně stejně funguje i přístup k I/O rozhraní, pouze s tím rozdílem, že místo signálu /MREQ se použije signál /IORQ. Z úsporných důvodů zde již není časový průběh zobrazen.

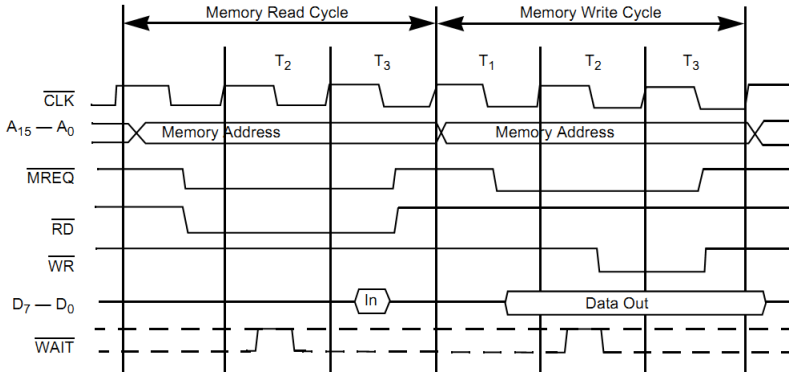
3.2 Uspořádání paměťového prostoru

Vzhledem k tomu, že procesor Z80A má 16-bitovou adresovací sběrnici, je schopen adresovat $2^{16} = 65536$ bajtů. Do tohoto adresového prostoru je v počítači ZX Spectrum namapována paměť ROM i paměť RAM. Funkci adresového dekodéru plní obvod ULA.

Procesor indikuje požadavek na komunikaci s pamětí pomocí signálu /MREQ. Signál



Obrázek 3.3: Načtení operačního kódu (opcode fetch) [18]



Obrázek 3.4: Čtení z paměti / zápis do paměti [18]

Adresa	A15	A14	Paměť
0000h..3FFFh	0	0	16 kB ROM
4000h..7FFFh	0	1	16 kB RAM
8000h..BFFFh	1	0	32 kB RAM - prvních 16 kB
C000h..FFFFh	1	1	32 kB RAM - posledních 16 kB

Tabulka 3.1: Mapování paměti v počítači ZX Spectrum 48K

/RD pak indikuje požadavek čtení z paměti, signál /WR zápis do paměti. Pokud je právě paměť obsazena, obvod ULA pozastaví procesor pomocí signálu /WAIT. Jakmile se paměť uvolní a ULA může tento požadavek splnit, vystaví na adresovou, datovou a řídící sběrnici potřebné signály pro paměť a po dokončení opět spustí procesor.

Z kapitoly 2.2 víme, že paměťová hierarchie je u verze 48K a 128K odlišná. Proto si nyní obě tyto architektury popíšeme odděleně.

3.2.1 ZX Spectrum 16K / 48K

Mapování paměti v počítači ZX Spectrum 48K je zobrazeno v tabulce 3.1. Celý adresní prostor o velikosti 64 kB je rozdělen na 4 banky o velikosti 16 kB. Adresace v rámci jedné banky se provádí pomocí nejnižších 14 bitů adresy (A13..A0). Horní dva bity (A15..A14) rozhodují o tom, která banka se použije.

Toto adresování je stejné u počítače ZX Spectrum 16K i ZX Spectrum 48K. U ZX Spectrum 16K je však osazeno pouze spodních 16 kB RAM. Od adresy 8000h..FFFFh žádná paměť není.

Paměť ROM

V počítači ZX Spectrum 16K/48K se nachází paměť ROM o velikosti 16 kB. Ta je umístěna do adresového prostoru od adresy 0000h až do 3FFFh. V této paměti je uložen uložen kompletní operační systém včetně interpretu jazyka BASIC. Tento systém na zakázku vytvořili programátoři John Grant a Steven Vickers ze společnosti Nine Tiles Information Handling Ltd. [15]

Paměť RAM

Paměť RAM se dělí na dvě části. První část je velká 16 kB a nachází se na adrese 4000h..7FFFh. Prvních 6912 bajtů je použito pro uložení obrazu (video display memory, nebo-li Video RAM). ULA z této paměti postupně vybírá informace zapsané procesorem a zobrazuje je na TV. Zbytek této paměti je použit pro uložení systémových proměnných a uživatelského programu.

Druhá část RAM má velikost 32 kB a nachází se na adrese 8000h až FFFFh. Jak již bylo řečeno, u verze ZX Spectrum 16K tato paměť chyběla. Bylo ale možné zakoupit rozšiřující kartu s touto pamětí, která se dala zapojit do systémové sběrnice.

3.2.2 ZX Spectrum 128K

Mapování paměti v počítači ZX Spectrum 128K je zobrazeno v tabulce 3.2.

Adresa	Paměťová banka							
C000h..FFFFh	RAM0	RAM1	RAM2	RAM3	RAM4	RAM5	RAM6	RAM7
8000h..BFFFh	RAM2							
4000h..7FFFh	RAM5							
0000h..3FFFh	ROM0	ROM1						

Tabulka 3.2: Mapování paměti v počítači ZX Spectrum 128K

Paměť ROM

Jak bylo zmíněno v kapitole 2.2.5, v počítači ZX Spectrum 128K je paměť ROM o velikosti 32 kB. Procesor je však stále schopen adresovat pouze 65536 bajtů, takže pokud by byla paměť ROM do adresového prostoru mapována celá, mohla by mít paměť RAM velikost pouze 32 kB. Navíc by takový počítač nebyl kompatibilní s původním ZX Spectrum 48K. Proto bylo nutné přistoupit k jinému řešení.

Paměť ROM byla rozdělena na dvě části (ROM0 a ROM1), každá o velikosti 16 kB. Obě tyto ROM jsou mapovány od adresy 0000h do 3FFFh (viz. tab. 3.2). O výběr správné ROM se pomocí speciálního registru stará obvod ULA. Toto řešení bude popsáno dále. Aby byla zajištěna kompatibilita s počítačem ZX Spectrum 48K i po softwarové stránce, obsahuje paměť ROM0 původní (lehce modifikovaný) obsah ROM z počítače ZX Spectrum 48K, ROM1 obsahuje vylepšený editor BASICu a úvodní startovací menu.

Paměť RAM

Z názvu je zřejmé, že v počítači je dostupných 128 kB RAM. Vzhledem k tomu, že procesor je schopen adresovat pouze 65536 bajtů, bylo nutné provést speciální úpravy tak, aby byl počítač schopen adresovat celých 128 kB.

V ULA přibyl speciální 8-bitový registr na adrese 7FFDh, ke kterému procesor přistupuje pomocí I/O instrukcí zápisu a čtení. Kvůli zjednodušení HW se registr ve skutečnosti adresuje pouze pomocí dvou bitů: A15=0 a A1=0. Díky tomu nebylo nutné v ULA implementovat celý 16-bitový komparátor adresy. Nevýhodou je, že je kvůli tomuto řešení obsazena čtvrtina celého adresního prostoru, který již není možné kvůli kolizi adres použít. V aplikaci se vždy doporučuje adresovat tento registr pomocí jeho celé adresy.

Jednotlivé bity v registru mají následující význam:

- **bity 2..0** - Vybírají paměťovou banku (RAM0 až RAM7), která je aktuálně mapována na adrese C000h až FFFFh (viz. tab. 3.2).
- **bit 3** - Vybírá paměťovou banku, ze které má ULA vykreslovat obraz (0=RAM5, 1=RAM7).
- **bit 4** - Vybírá aktuální paměť ROM, která se namapuje na adresu 0000h až 3FFFh (viz. tab. 3.2).
- **bit 5** - Zápisem log. 1 do tohoto bitu se zakáže zápis do celého registru. Opětné povolení je možné pouze resetováním celého počítače.
- **bity 7..6** - Nejsou využity.

Čtení z tohoto registru je stejné, jako zápis hodnoty FFh.

RAM2 je vždy mapována na adresu 8000h - BFFFh. Nyní však víme, že je ji navíc možné mapovat na adresu C000h - FFFFh. Obecně to však není doporučované. To stejné platí i pro RAM5.

Další důležitá informace je, že ZX Spectrum 128K umožňuje přepínat paměťovou banku, ze které se vykresluje obraz (3. bit v registru 7FFDh). To umožňuje vytvářet plynulé animace. Z jedné banky se vykresluje aktuální snímek a do druhé banky se mezi tím může generovat následující snímek. Tento bit ale nemá žádný vliv na mapování paměti. Na adrese 4000h - 7FFFh je vždy mapována banka RAM5. Pokud chceme zapisovat do RAM7, musíme si jí namapovat na adresu C000h - FFFFh pomocí bitů 0 až 2 v registru 7FFDh

3.3 Grafický výstup

Dříve než se pustíme do vysvětlování konkrétního řešení v počítači ZX Spectrum, nebude na škodu vysvětlit základní princip CRT televizoru, který se používal pro zobrazení obrazu v originálním ZX Spectrum (zpracováno podle [12], [20]).

3.3.1 CRT obrazovka

CRT (Cathode Ray Tube) obrazovka je složena ze systému elektrod, které jsou uloženy ve vyčerpané baňce, jejíž přední stěna je z vnitřní strany pokryta luminiscenční vrstvou - luminoforem, který tvoří stínítko. Na luminofor je zaostřován elektronový svazek, jehož zdrojem je elektronová tryska umístěná v hrdle obrazovky.

Elektrony svazku jsou urychlovány vysokým kladným anodovým napětím, dopadají na luminofor a odevzdávají jeho atomům svojí kinetickou energii. Atomy přijatou energii opět vyzařují ve formě viditelného světla, jehož barva závisí na složení luminoforu. V místě dopadu svazku na luminofor vzniká zářící bod. Působením síly magnetického pole dochází k vychylování svazku a je tak možné zářící bod po ploše stínítka posouvat a vytvářet pomocí něj světelnou stopu.

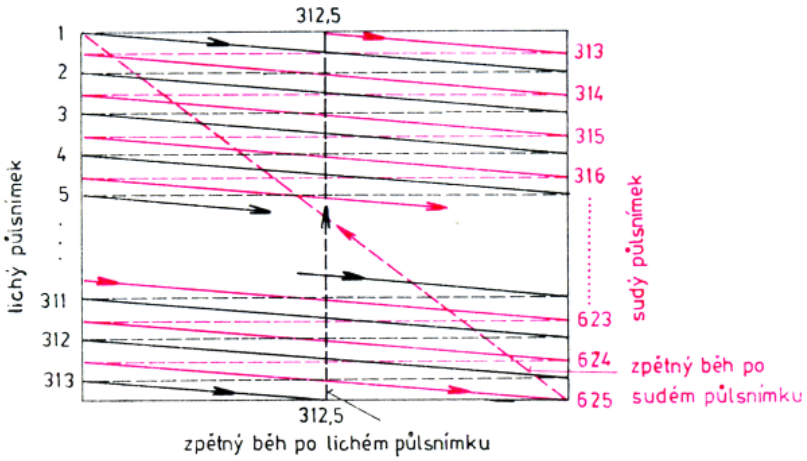
Elektronový svazek je postupně po řádcích vychylován po celé ploše stínítka. K vytvoření dojmu svícení celé obrazovky přispívá jednak setrvačnost lidského oka, a také světelná setrvačnost (dosvit) luminoforu.

Barevné obrazovky vytvářejí pro každou základní barvu (červená, zelená, modrá) samostatný elektronový svazek, který dopadá na jinak obarvený luminofor. Pro správné zaměření paprsku na luminofor je stínítko složeno např. z úzkých svislých proužků (obrazovka typu trinitron), kterými musí elektronové svazky projít. Při pozorování z větší vzdálenosti pak dochází ke splynutí dílčích barevných bodů v jeden obraz.

3.3.2 Rozklad obrazu na řádky a prokládané řádkování

Obraz je rozložen určitým počtem řádků do jednoho snímku. Elektronový paprsek postupuje po stínítku plynulým rovnoměrným pohybem zleva doprava a vytváří tak jednotlivé řádky. Tento děj nazýváme *horizontální činný běh*. Jakmile paprsek dorazí na pravý okraj obrazu, vrací se rychle zpět (*horizontální zpětný běh*).

Pro jeden snímek je zvoleno tolik řádků, aby struktura obrazu nebyla příliš hrubá. U televizního přenosu se snímek u nejpoužívanější evropské soustavy PAL skládá z 625 řádků. Jakmile paprsek vykreslí poslední řádek, vrací se ve *snímkovém zpětném běhu* zdola zpět vlevo nahoru.



Obrázek 3.5: Prokládané řádkování [20]

Počet snímků za jednu sekundu musí být zvolen tak, aby obraz neblikal. V kinematografii je rychlost posuvu filmu 24 obrázků za sekundu, ale každý obrázek se promítá dvakrát, aby oko nerozeznalo dílčí vjem (blikání). V televizi se zobrazuje 25 snímků za sekundu a blikání se zamezuje složením každého snímku ze dvou půlsnímků zabírajících celou plochu stínítka obrazovky.

Toto prokládání bylo zavedeno z úsporných důvodů. Kdyby se obraz uskutečňoval pomocí 50 snímků za sekundu a jeden snímek byl složen z 625 řádků, bylo by pro přenos obrazu nutné využít šířku pásma 13 MHz. Rozdělením obrazu na dva půlsnímky lze šířku přenášeného pásma snížit na polovinu (6 MHz).

Princip prokládaného řádkování je znázorněn na obrázku 3.5. 625 řádků je rozděleno na dva půlsnímky, z nichž každý se rozprostírá přes celou výšku stínítka a každý má 312,5 řádků. Obraz nebliká, protože počet půlsnímků je 50 za sekundu. Rozlišovací schopnost se téměř nezmenší, neboť informace o jasu se mezi dvěma půlsnímky za 1/50 sekundy příliš nezmění.

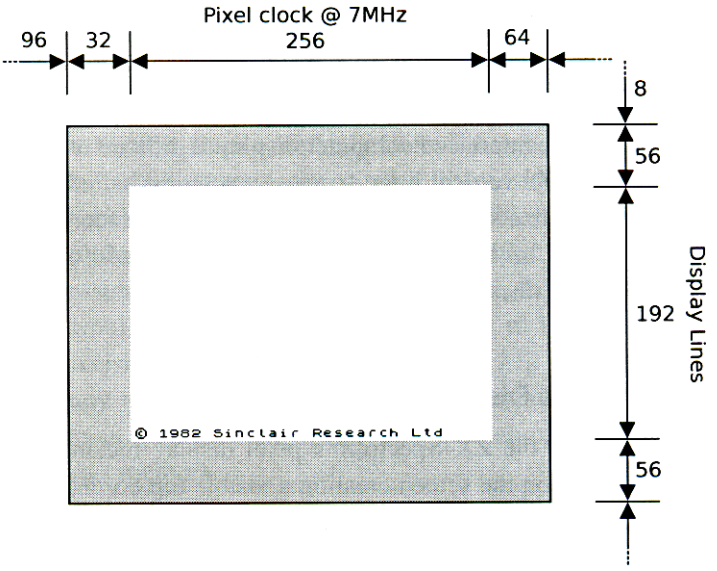
3.3.3 Televizor jako grafický výstup počítače ZX Spectrum

V předchozích kapitolách jsme se seznámili s principem klasické televize. Nyní se podíváme, jak byla televize využita počítačem ZX Spectrum k zobrazování [15].

Pro použití televize jako počítačového monitoru je nutné rozdělit obrazovku na matici pixelů. U normy PAL nemůže být počet řádků vyšší než 625. Počet sloupců (šířka) není přesně dána. Závisí na rozlišovací přesnosti elektronového svazku a stínítka. Aby byly pixely ve tvaru čtverce, měla by šířka odpovídat poměru 4:3, což je poměr obrazu původních televizí.

Protože televize nemá žádnou paměť, vyžaduje nepřetržitý proud dat, aby byl obraz stále viditelný. To musí zajistit buď CPU, které by tím ale spotřebovalo většinu svého strojového času, nebo speciální obvod, který autonomně zajišťuje vykreslování. Jak již víme z kapitoly 2.2, u počítače ZX80 se o vykreslování staralo CPU, u dalších již vykreslování obrazu zajišťoval speciální obvod v ULA. V obou případech je však nutné mít obraz uložen v paměti, kam má přístup CPU i ULA.

Pokud by byl řádek rozdělen na 1000 pixelů, musel by do televize jít signál o frekvenci 15,625 MHz. Při vykreslování pomocí CPU by musel procesor pracovat minimálně na této



Obrázek 3.6: Rozměry obrazu generovaného počítačem pro normu PAL [15]

frekvenci. Navíc množství paměti, které by bylo nutné použít pro uložení, by bylo neúnosné. Pokud by např. 1 pixel odpovídal 1 bitu, pak 1000 pixelů na řádku by vyžadovalo 125 bajtů paměti. Při plném vertikálním rozlišení televize s použitím 312 řádků by uložení celé videopaměti vyžadovalo cca 38K RAM. To je víc jak polovina adresovatelného prostoru 8-bitového CPU.

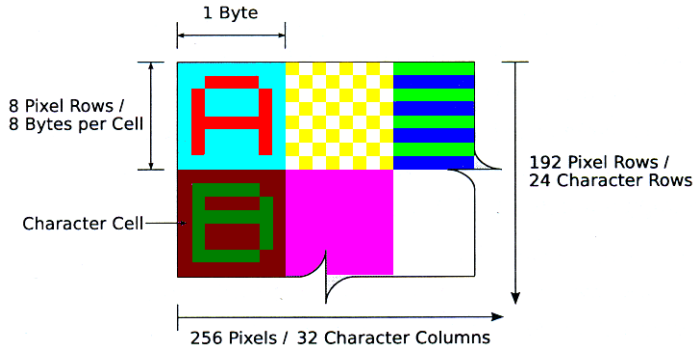
Proto bylo nutné vymyslet jiné řešení - snížila se frekvence a zmenšila se velikost zobrazované části, okolo které je zobrazen okraj (viz obr. 3.6). ZX81 má rozlišení 256×192 pixelů, ale kvůli limitované velikosti RAM (1 KB) je možné zobrazovat pouze znaky, jejichž matice je uložena v ROM. Obraz je tedy rozdělený na 32×24 znaků a nic jiného než tyto znaky není možné zobrazit. Pro uložení ASCII hodnot celé obrazovky je vyžadováno pouze 768 bajtů.

Počítače ZX Spectrum zdědily toto rozlišení, ale každému pixelu přiřazují jeden bit. Pro uložení jednoho řádku je tedy vyžadováno $256 \div 8 = 32$ bajtů. Při vertikálním rozlišení 192 řádků je potřeba $32 \times 192 = 6144$ bajtů pro uložení celého obrazu.

Protože PAL televize je složena z 625 prokládaných řádků, které jsou rozděleny sudé a liché snímky o velikosti 312,5 řádků, a ZX Spectrum má vertikální rozlišení 192 řádků, zobrazují sudé i liché snímky stejný obraz.

3.3.4 Generování obrazu

Již víme, že obraz má rozlišení 256×192 pixelů, organizovaných jako virtuálních 32×24 znaků. kde znak je 8 pixelů široký a 8 pixelů vysoký (1 bajt široký a 8 bajtů vysoký). Navíc je možné pro každý znak nastavit barvu popředí a barvu pozadí. Toto nastavení barev je aplikováno na celý znak. Pokud pracujeme pouze se znaky, je toto řešení velice jednoduché a flexibilní. Pokud ale pracujeme na úrovni pixelů, dvě barvy na jednu buňku 8×8 pixelů může být málo. Má to však i značné výhody. Generování grafiky procesorem je velice rychlé a uložený obraz zabírá jen cca 7 KB RAM. Díky tomu má programátor k dispozici mnohem více uživatelské paměti. A jak již dnes víme, i na ZX Spectrum lze programovat hry s pěknou grafikou.



Obrázek 3.7: Vztah bajtu, znaku a pixelů [15]

7	6	5	4	3	2	1	0
FLASH	BRIGHT	PAP2	PAP1	PAP0	INK2	INK1	INK0

Tabulka 3.3: Přidělení významu jednotlivým bitům atributů

Obrázek 3.7 zobrazuje výřez části obrazu, ve kterém je vidět rozdělení obrazu na jednotlivé znaky o velikosti 8×8 pixelů a různé nastavení barev popředí a pozadí u každého znaku.

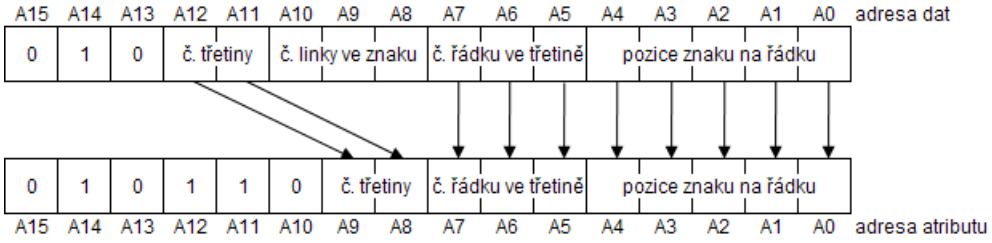
Při zobrazování obrazu na obrazovku musí ULA neustále číst z paměti dvojice bajtů. Jeden z nich obsahuje bitovou informaci „pixel svítí/nesvítí“ (*data*), druhý bajt (*atribut*) obsahuje barvu rozsvícených a zhasnutých pixelů v celém znaku. Kromě barvy je zde také uložen příznak blikání znaku (*flash*) a jas znaku (*bright*). V průběhu vykreslování těchto 8 pixelů musí ULA začít načítat následující dvojici, aby bylo možné ihned začít vykreslovat následující znak.

Význam jednotlivých bitů atributu je vysvětlen v tabulce 3.3. Pokud má být daný pixel rozsvícen, jeho barva se určí podle bitů 0 - 2 atributu. Pokud má být naopak pixel zhasnutý, jeho barva se určí podle bitů 3 - 5 atributu. Tabulka 3.4 ukazuje odpovídající barvu k zadané binární hodnotě. Vidíme, že pixel ve skutečnosti není rozsvícený/zhasnutý, ale pouze určujeme, zda se má vykreslit barvou popředí či pozadí.

V tabulce 3.4 vidíme pouze 8 barev, ZX Spectrum však zvládá až 16 barev. U každého znaku je totiž možné nastavit, pomocí bitu BRIGHT v jeho atributu, zda má znak normální

		PAPER			INK		
barva	č.	5	4	3	2	1	0
černá	0	0	0	0	0	0	0
modrá	1	0	0	1	0	0	1
červená	2	0	1	0	0	1	0
fialová	3	0	1	1	0	1	1
zelená	4	1	0	0	1	0	0
sv. modrá	5	1	0	1	1	0	1
žlutá	6	1	1	0	1	1	0
bílá	7	1	1	1	1	1	1

Tabulka 3.4: Význam bitů atributů PAPER a INK



Obrázek 3.8: Vztah adres atributů k adresám dat

nebo zvýšený jas. To v podstatě přidává dalších 8 barev. Bohužel se však tento jas nastavuje společně jak pro barvu popředí, tak i pro barvu pozadí. Jeden znak tedy může mít pouze 8 barev, buď se zvýšeným jasem, nebo se sníženým jasem.

Poslední bit FLASH v atributu povoluje/zakazuje blikání znaku. Pokud je FLASH nastaven do vysoké úrovně, ULA invertuje sériový tok dat na výstupu (signál INK/PAPER) s kmitočtem asi 3 Hz. V podstatě se jedná pouze o „přehazování“ atributů INK a PAPER s daným kmitočtem.

Jak jsme již viděli na obrázku 3.6, kolem celého obrazu je zobrazen okraj (border). Ten může nabývat všech 8 barev (tab. 3.4) s normálním jasem. U okraje není možné nastavit vyšší jas. Barva okraje je uložena ve speciálním registru v bitech 0 až 2. K registru muže CPU přistupovat pomocí instrukce OUT 0xFE,A.

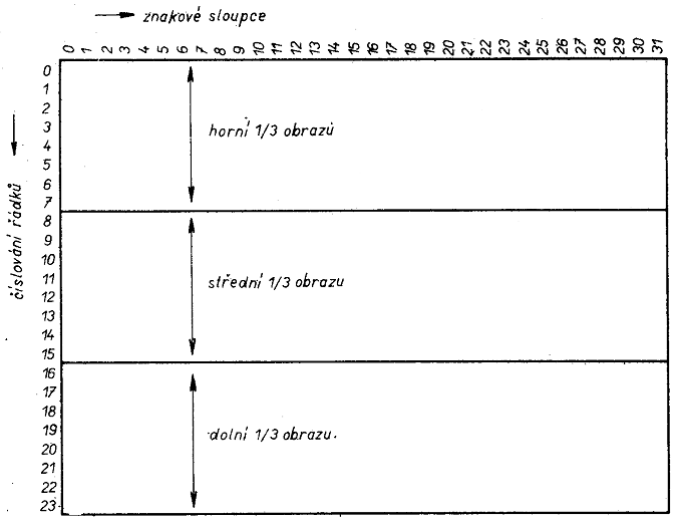
3.3.5 Struktura Video RAM

Nyní se podíváme, jakým způsobem je obraz ve video paměti uložen [16]. Z kapitoly 3.2 již víme, že Video RAM se nachází v adresním prostoru 4000h - 7FFFh. Prvních 6912 bajtů obsahuje tzv. DISPLAY FILE (část zobrazovaná TV přijímačem), zbytek obsahuje např. systémové proměnné, které však pro nás v tuto chvíli nejsou zajímavé.

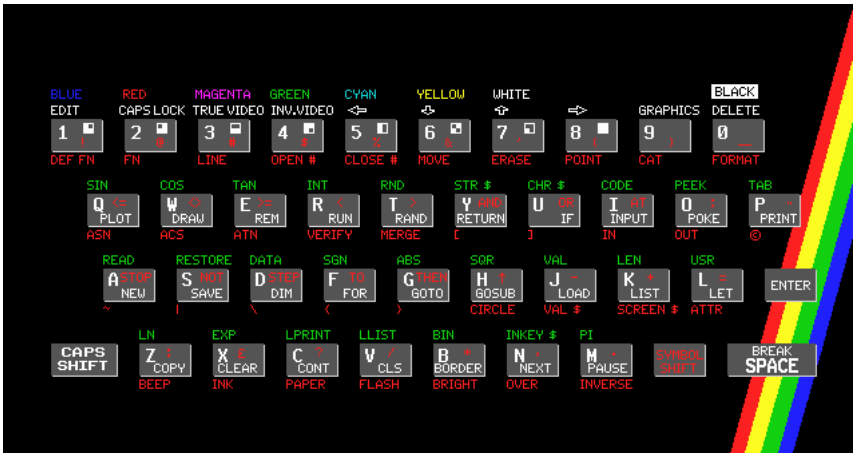
Dále již víme, že obraz má rozměr 256×192 pixelů, který je rozdělen na matici 32×24 znaků o velikosti 8×8 pixelů. Každá osmice bodů na obrazovce má svoji pevnou fyzickou adresu ve Video RAM. Každému znaku (poli 8×8 bodů) odpovídá jedna fyzická adresa atributu. Jejich vztah je znázorněn na obrázku 3.8. Na obrázku 3.9 je znakové pole, zobrazované na TV přijímači, s rozdělením do třetin. Každý znak v tomto poli je složen z osmi TV linek (přeběhů paprsku).

Na první pohled se může zdát tato adresace nesmyslně složitá. Ve vztahu mezi adresami zobrazovaných dat a adresami atributů je ale určitý systém. Znakové řádky jsou adresovány tak, jako kdyby vždy 8 řádků „leželo vedle sebe“ (tj. 1 řádek o 256 znacích). Pak by adresace jednotlivých bytů dat vzestupně narůstala „po linkách“ přes pomyslné 3 řádky až do konce DISPLAY FILE, včetně odpovídajících atributů.

Toto rozložení adres dat a atributů bylo zvoleno zcela záměrně z důvodu snadné adresace atributů. Při zobrazování libovolného bajtu dat na obrazovce je nutné současně znát i okamžitý stav odpovídajícího atributu. Z obrázku 3.8 vidíme, že ke každému znaku, zobrazovanému na obrazovce, existuje atribut, který má spodní část adresy (8 bitů) stejnou, jako 8 bajtů zobrazovaného znaku. Použité paměti DRAM totiž mají řádkovou část adresy (zapisované řídícím signálem RAS) širokou 7 bitů. Při adresování se pak využívalo stránkového módu, který vychází z principu, že adresy dat, které mají být čteny z paměti, mají společnou část řádkové adresy. Pak stačí pouze jeden zápis řádkové adresy do paměti a postupným zápisem sloupcových adres (řídícím signálem CAS) je možno z paměti postupně



Obrázek 3.9: Rozdělení obrazovky na znakové řádky a sloupce. 1/3 obsazuje 2048 bajtů dat a 256 bajtů atributů. [16]



Obrázek 3.10: Klávesnice ZX Spectrum [6]

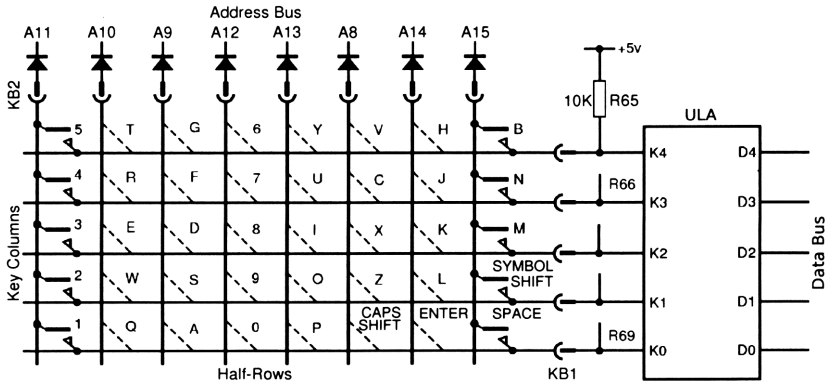
vybírat adresovaná data.

3.4 Klávesnice

V počítači ZX Spectrum je klávesnice typu QWERTY (obr. 3.10), která je řešena jako matice 40 spínačů. Pokud bychom chtěli adresovat každé tlačítko individuálně, u 8-bitové datové sběrnice by bylo nutné použít 5 vstupně výstupních 8-bitových portů. Pro snížení počtu vstupů je klávesnice zapojena do dvourozměrné matice o rozměrech 5×8. Obrázek 3.11 ukazuje její konkrétní zapojení.

Sloupce matice jsou připojeny k horním 8 bitům adresové sběrnice A8 až A15. Řádky matice jsou připojeny k obvodu ULA, který zprostředkovává styk s CPU. Řádky jsou navíc pomocí odporů 10K připojeny k napájecímu napětí 5V a je na nich tedy log. 1.

Protože jsou tlačítka zapojena do matice, je nutné jejich stav číst softwarově pomocí tzv. multiplexování. Procesor nejprve na adresovou sběrnici vystaví adresu sloupce, jehož stav



Obrázek 3.11: Zapojení klávesnice do matice a její připojení k ULA [15]

Klávesy	Horních 8 bitů adresy	Nastavená adresová linka	A15 - A8
B - SPACE	0x7F	A15	01111111
H - ENTER	0xBF	A14	10111111
Y - P	0xDF	A13	11011111
6 - 0	0xEF	A12	11101111
1 - 5	0xF7	A11	11110111
Q - T	0xFB	A10	11111011
A - G	0xFD	A9	11111101
CAPS - V	0xFE	A8	11111110

Tabulka 3.5: Horní část I/O adresy procesoru pro čtení stavu kláves

chce zjistit. Sloupec, jehož hodnota nás zajímá, musí být nastaven do log. 0, ostatní sloupce jsou nastaveny do log. 1. Procesor poté přečte stav na datové sběrnici, která je pomocí ULA připojena k řádkům matice. Pokud byla stisknuta nějaká klávesa v adresovaném sloupci (který má hodnotu log. 0, tedy GND), dojde k uzemnění odpovídajícího řádku (na kterém původně byla log. 1, tedy 5V) a řádek se pro procesor jeví jako log. 0 v odpovídajícím bitu datové sběrnice. Ostatní bity datové sběrnice mají hodnotu log. 1.

Vidíme, že v jednom čtení procesoru lze zjistit stav pouze jednoho sloupce. Pro přečtení celé klávesnice je nutné postupně zkontrolovat všech 8 sloupců matice klávesnice. Tabulka 3.5 ukazuje horní část adresové sběrnice, kterou je potřeba nastavit pro čtení odpovídajícího sloupce.

Již víme, že řádky matice klávesnice (data) jsou k procesoru připojeny přes ULA. ULA totiž slouží jako adresový dekodér. Pokud procesor při čtení pomocí I/O instrukce vystaví na spodní část adresové sběrnice A0 - A7 hodnotu 0xFE, ULA připojí řádky matice klávesnice (data) na datovou sběrnici. V opačném případě je klávesnice od sběrnice odpojená. Klávesnice ve skutečnosti není adresovaná celou osmibitovou hodnotu 0xFE, ale pouze nultým bitem adresy (A0) ve stavu log. 0. Díky tomuto řešení nebylo v ULA nutné implementovat celý 8-bitový komparátor adresy, ale pouze jednobitový. Na druhou stranu však ztrácíme polovinu adresového prostoru, který již nemůže být k ničemu jinému použit.

Softwarová obsluha klávesnice využívá vlastnosti procesoru Z80, která spočívá v tom, že při vykonávání vstupní instrukce `IN A,adr` se na horní polovině adresové sběrnice A8 až A15 objeví hodnota v registru `A`, na spodní polovině adresy A0 až A7 se objeví hodnota `adr`. Správnou hodnotu v registru `A` (viz tab. 3.5) tedy procesor vybere požadovaný slou-

pec, hodnotou adr=0xFE je klávesnice adresována a následně připojena k datové sběrnici, procesor přečte stav v adresovaném sloupci klávesnice a uloží ho zpět do registru A. Tuto akci je nutné provést postupně pro všech 8 sloupců klávesnice.

3.5 Další I/O rozhraní

Nyní se ve stručnosti podíváme na další I/O rozhraní, která jsou interně zabudovaná v ZX Spectrum. Veškerá tato I/O rozhraní jsou mapována na port 0xFE, o kterém jsme již mluvili v souvislosti s klávesnicí. Tento port má však odlišné chování pro zápis a pro čtení.

Při čtení z portu 0xFE pomocí instrukce IN je v bitech D0 - D4 hodnota načtená z klávesnice (viz kapitola 3.4). Bit D6 slouží k nahrávání dat z magnetofonu do počítače. Signál, který byl zaznamenaný na magnetofonovou pásku, se snímá, zesiluje a přivádí na bit D6. Načítání dat z magnetofonu se provádí softwarově. CPU vzorkuje tento vstupní bit, sestavuje z něj bajty a ty následně zpracovává a ukládá do paměti.

Zápis na port 0xFE pomocí instrukce OUT má odlišné chování. V kapitole jsme již řekli, že v bitech D0 až D2 je uložena barva okraje obrazovky. Bit D3 slouží jako výstup z počítače do magnetofonu. Procesor softwarově generuje zvukový signál pro magnetofon a postupně bit po bitu ukládá zapisovaná data na pásku. Bit D4 slouží pro ovládání reproduktoru. Zvuk se generuje softwarově tak, že procesor postupně střídá logickou hodnotu 0 a 1. Doba mezi přepnutím logické hodnoty mění frekvenci tónu.

Jak již bylo řečeno v předchozích kapitolách, port 0xFE je ve skutečnosti dekódován pouze pomocí hodnoty log. 0 v nultém bitu adresy. Tyto periferie proto lze adresovat celkem 127 různými adresami, např 0x00, 0x10 apod. Důrazně je však doporučováno používat pouze adresu 0xFE, aby nedocházelo k případným kolizím na sběrnici při použití externích periferií.

Kapitola 4

Návrh emulátoru počítače ZX Spectrum

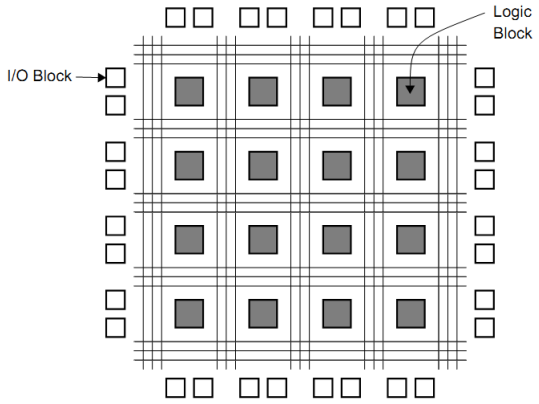
V této kapitole probereme můj návrh emulátoru počítače ZX Spectrum. Vzhledem k tomu, že chci celý emulátor implementovat pomocí technologie FPGA, stručně si tuto technologii popíšeme v kapitole 4.1. Kapitola 4.2 představí platformu FITkit, na které chci emulátor vyvíjet a otestovat. V kapitole 4.3 se již podíváme na konkrétní návrh celého emulátoru.

4.1 FPGA technologie

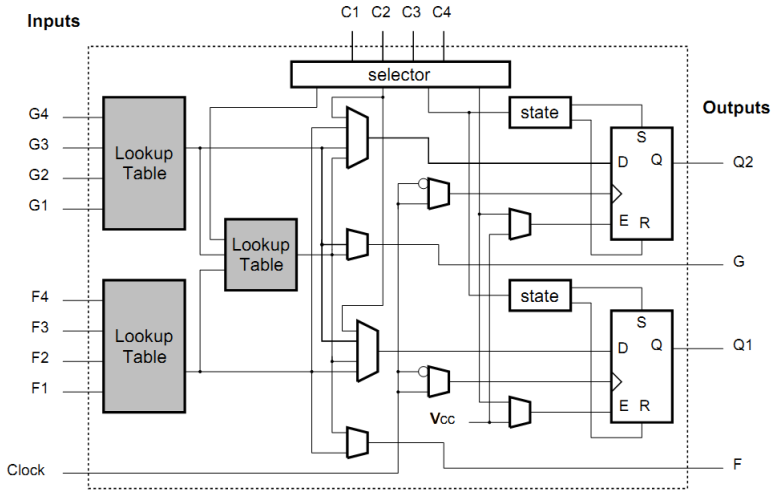
[1] Field-Programmable Gate Array (FPGA) je speciální číslicový integrovaný obvod, který je složen z mnoha malých konfigurovatelných logických bloků (CLB), které jsou vzájemně propojeny pomocí propojovací sítě. Pomocí speciálního konfiguračního řetězce lze libovolně měnit chování konfigurovatelných logických bloků a taktéž je možné libovolně měnit jejich propojení.

U klasických ASIC (Application-Specific Integrated Circuit) obvodů je jejich funkce (vnitřní zapojení) dána již při výrobě a nelze ji již nijak změnit. FPGA je oproti tomu univerzální architektura, jejíž vnitřní propojení a chování lze libovolně měnit i po výrobě. Daní za tuto flexibilitu je mnohem větší složitost obvodu, vyšší spotřeba a nižší rychlost, než u ASIC obvodu.

Obrázek 4.1 ukazuje základní strukturu FPGA obvodu. Vidíme, že FPGA je složeno



Obrázek 4.1: Základní struktura FPGA obvodu [1]



Obrázek 4.2: Struktura konfiguračního logického bloku (CLB) v FPGA Xilinx XC4000 [1]

z matice programovatelných logických bloků (CLB), které jsou vzájemně propojeny pomocí konfigurovatelné propojovací sítě. Dále se v FPGA nacházejí IOB (Input/Output Block), které jsou připojeny k I/O pinům na pouzdru a slouží pro připojení FPGA k okolnímu prostředí (převádí logické úrovně apod.). Součástí FPGA mohou být i další vestavěné speciální bloky, jako např. HW násobičky, BRAM paměti, bloky pro správu hodin (DCM) apod. Protože jsou všechny tyto bloky včetně propojovací sítě konfigurovatelné, je také nutné někde uchovávat jejich konfiguraci. K tomu slouží konfigurační paměť.

Největším problémem FPGA je zřejmě její prostorová složitost. Až 80% plochy čipu totiž zabírá propojovací síť. Další velkou část plochy zabírá konfigurační paměť. Samotná logika zabírá pouze jednotky procent z celého čipu. Velice zjednodušeně lze vyjádřit vztah obsazené plochy touto rovnicí [1]: $P_{propojovaciSit} \approx 10 \cdot P_{konfiguracniPamet} \approx 100 \cdot P_{logika}$

Struktura konfigurovatelného logického bloku (CLB) je na obrázku 4.2. Je složen z několika (např. 2) SLICE bloků, což jsou menší logické buňky, mezi kterými je rychlé propojení a „carry řetězce“ pro konstrukci rychlých sčítaček, násobiček apod.

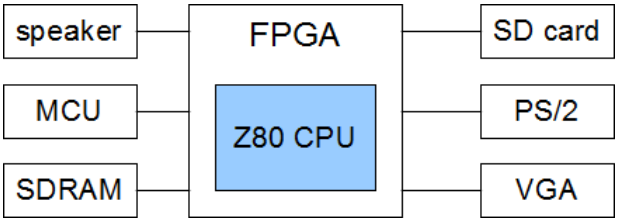
SLICE je založen na tzv. look-up tabulce (LUT), což je malá paměť o datové šířce jeden bit. Její adresová sběrnice má obvykle šířku 4 až 6 bitů. Označíme-li šířku adresové sběrnice LUT tabulky neznámou k, lze velikost paměti vyjádřit vztahem $2^k \times 1$ bitů. Pomocí LUT tabulky lze realizovat libovolnou k bitovou logickou funkci. Kromě LUT tabulky navíc SLICE obsahuje také registr pro uložení libovolné hodnoty, multiplexory a „carry řetězec“.

4.2 Platforma FITkit

FITkit [7] je vývojový kit, který vznikl na Fakultě informačních technologií Vysokého učení technického v Brně. Cílem tohoto kitu je umožnit studentům, aby mohli navrhovat a prakticky realizovat nejen softwarové, ale také hardwarové projekty či dokonce celé aplikace.

Platforma FITkit obsahuje výkonný mikrokontrolér s nízkým příkonem, hradlové pole FPGA a řadu periferií. Důležitým prvkem FITkitu je právě hradlové pole, které studentům umožňuje pokročilý návrh a implementaci hardwarových zařízení.

V současné době existují tři verze FITkitu: 1.0, 1.2 a 2.0. K implementaci emulátoru využiji FITkit 2.0 ve vylepšené verzi, který oproti klasickým verzím 2.0 obsahuje mnohem větší FPGA. Konkrétní hardwarové parametry použitého FITkitu jsou:



Obrázek 4.3: Blokové schéma emulátoru

- MCU Texas Instruments MSP430F2617
- FPGA Xilinx Spartan 3 XC3S400-4PQ208C (400 000 hradel, 896 CLB)
- SDRAM 8×8 Mbit
- USB převodník FT2232C
- audio rozhraní a kodek TLV320AIC23B
- 2 rozhraní PS/2
- rozhraní VGA
- rozhraní RS232
- maticová klávesnice 4×4
- řádkový LCD displej
- rozšiřující konektory

Centrem celého FITkitu je FPGA. K němu je připojen MCU, SDRAM i veškeré rozhraní. Pokud chce např. MCU komunikovat s PS/2 rozhraním, je nutné v FPGA implementovat řadič a propojit MCU s tímto řadičem. Pro komunikaci mezi FPGA a MCU je v knihovně implementován SPI komunikační systém, který umožňuje adresování libovolných komponent. Součástí knihovny je např. také VHDL implementace řadiče PS/2 a VGA rozhraní a SDRAM paměti.

V tuto chvíli nebudeme zabíhat do podrobnějších detailů. Veškeré detailní informace o FITkitu čtenář nalezne v [7].

4.3 Návrh architektury emulátoru ZX Spectrum

Již jsme se seznámili s technologií FPGA i platformou FITkit, kterou jsem se rozhodl použít pro prvotní návrh a implementaci mého emulátoru počítače ZX Spectrum. Tato kapitola pojednává o návrhu celého emulátoru.

Zadání této práce požaduje konstrukci emulátoru počítače ZX Spectrum 48K. Rozhodl jsem se, že si zadání rozšířím a implementuji emulátor počítače ZX Spectrum 128K. Ten vychází z původního modelu, ale přináší řadu vylepšení. Z počátku tedy budu vyvíjet verzi 48K, po jejím odladění jí rozšířím na verzi 128K.

Obrázek 4.3 ukazuje blokové schéma emulátoru. Vidíme, že centrem celého zařízení je FPGA, ke kterému jsou připojeny všechny ostatní periferie. Takto tomu je i v použitém FITkitu (viz kap. 4.2).

4.3.1 Procesor

„Mozkem“ počítače je procesor. Víme, že v ZX Spectrum je použit 8-bitový procesor Z80 od firmy Zilog. Protože chci vytvořit emulátor co nejmenší a nejjednodušší, rozhodl jsem se, že nepoužiji skutečný procesor Z80 jakožto samostatný integrovaný obvod připojený k FPGA, ale implementuji jeho jádro přímo v FPGA.

Na internetu [14] se nachází otevřená implementace procesoru Z80 v jazyce VHDL pod názvem T80. Kromě jádra procesoru Z80 tento projekt podporuje také jádro 8080 (protože Z80 je jeho vylepšená modifikace, viz kap. 3.1) a navíc také modifikovanou instrukční sadu, která je použita v konzoli GameBoy.

4.3.2 Paměť

V kapitole 3.2 byla detailně popsána paměťová hierarchie v počítači ZX Spectrum 48K a 128K. Víme, že v paměti ROM je uložen interpret jazyka BASIC a zbytek paměťového prostoru zabírá paměť RAM, která má velikost 48KB respektive 128KB.

V počítači je navíc ještě oddělená klasická RAM a Video RAM, ze které ULA vykresluje obraz na obrazovku. Oddělená je proto, aby procesor nebyl zdržován obvodem ULA, pokud pracuje s jinou částí RAM, než s Video RAM.

Protože na FITkitu je osazena paměť SDRAM o velikosti 8 MB, která má mnohem vyšší kapacitu a také je mnohem rychlejší, než původní paměti, rozhodl jsem se, že tuto paměť použiji jak pro normální RAM, tak i pro Video RAM. Abych celý emulátor ještě více zjednodušil, nahradím i původní paměť ROM touto SDRAM. Díky tomuto řešení bude celý adresní prostor počítače pracovat pouze s pamětí SDRAM, která je přítomna na FITkitu.

Protože ale obsah ROM (BASIC) musí být někde uložen, využiji k jeho uložení MCU, který k tomu má dostatečně velkou Flash paměť. Po zapnutí FITkitu MCU překopíruje obsah paměti ROM do paměti SDRAM a spustí vykonávání programu procesorem Z80 v FPGA.

4.3.3 SD/MMC karta

Další velice důležitou periferií počítače ZX Spectrum bylo magnetofonové rozhraní, pomocí kterého se nahrávaly a ukládaly programy. Toto řešení se bohužel neukázalo býti příliš spolehlivé. Často se stávalo, že uložený záznam na pásce se podařilo načíst až na několikátý pokus, v nejhorším případě vůbec. Dalším nevýhodou byla příliš dlouhá doba nahrávání/ukládání programu z kazety. Jednalo se řádově o minuty, u větších aplikací až desítky minut. Pokud navíc během nahrávání došlo k chybě, bylo o zábavu postaráno.

Z toho důvodu jsem se rozhodl, že magnetofon nebudu emulovat vůbec a místo něj vytvořím svoje speciální rozhraní pro práci s SD/MMC kartami. Vzhledem k tomu, že tyto paměťové karty pracují s napájecím napětím 3.3V (stejně jako FPGA a MCU) a používají ke komunikaci klasické SPI rozhraní, lze paměťovou kartu přímo připojit k FPGA přes externí konektor na FITkitu.

Náhrada magnetofonu za SD kartu má však i nevýhodu. Rutina pro práci s magnetofonem byla přímo součástí BASICu. Stačilo v něm napsat příkaz „LOAD“, zapnout magnetofon a aplikace se začala načítat. Při použití SD karty ale nic takového nebude možné použít, protože paměťová karta používá souborový systém (většinou FAT32) a rutina pro její obsluhu je příliš složitá a velká, než aby se vešla do standardní ROM.

Z tohoto důvodu je nutné naprogramovat speciální obslužnou aplikaci (buď ve formě kompletní ROM, nebo jako obyčejná spustitelná aplikace), která bude umět komunikovat

s SD/MMC kartou, procházet souborový systém na kartě a zavádět z ní aplikace do paměti.

4.3.4 Další vstupní a výstupní periferie

Originální ZX Spectrum používá jako zobrazovací výstup televizor. Já jsem se rozhodl použít klasický počítačový monitor, který se k mému emulátoru připojí pomocí standardního VGA rozhraní. V dnešní době je sice toto rozhraní nahrazováno digitálním rozhraním jako je DVI nebo HDMI, ale pro svou jednoduchost a dostupnost na FITkitu jsem se přesto rozhodl použít VGA. Součástí knihovny FITkit je navíc již implementovaný řadič, který tím pádem mohu s výhodou použít.

Standardním vstupem počítače ZX Spectrum je speciální zabudovaná klávesnice. Tu nahradím klasickou počítačovou klávesnicí s rozhraním PS/2, které je opět přítomné na FITkitu včetně jeho řadiče implementovaného v knihovně FITkitu.

Postupem času vzniklo pro počítač ZX Spectrum také několik druhů myší, které jsou některými aplikacemi podporovány. Jsou to především Kempston Mouse a Amiga Mouse. Princip připojení k počítači a jejich ovládání se liší. Vzhledem k tomu, že na FITkitu jsou celkem dvě rozhraní PS/2, využiji druhý konektor právě pro připojení klasické počítačové PS/2 myši, která bude emulovat jeden (případně více) druhů původních myší.

V kapitole 3.5 byl popsán integrovaný reproduktor, který slouží pro generování jednoduchých tónů. Na FITkitu se nachází jednoduchý piezo-reproduktor, kterým nahradím původní reproduktor. V případě potřeby je také možné k FITkitu, pomocí externího rozhraní, připojit externí reproduktor.

Kapitola 5

Hardwarové řešení

Tato kapitola popisuje moje řešení emulátoru z pohledu hardware. Celý emulátor je realizován pomocí FPGA s použitím jazyka VHDL. Pro lepší pochopení řešeného problému proto budu používat ukázky VHDL kódu.

Obrázek 5.1 zjednodušeně ukazuje princip propojení jednotlivých komponent emulátoru. Již víme, že procesor Z80 rozlišuje paměťový adresový prostor a I/O adresový prostor. Oba prostory proto používají oddělenou sběrnici. K paměťové sběrnici je připojena SDRAM paměť, procesor T80 a grafická jednotka (GPU), která čte data z paměti a vykresluje je na obrazovku. Poslední komponentou, která přistupuje k paměti, je můj vlastní DMA řadič. Plní velice významnou funkci, která bude vysvětlena v kapitole 5.10. K I/O sběrnici je připojen procesor, klávesnice, myš, řadič SD/MMC karet a opět můj DMA řadič.

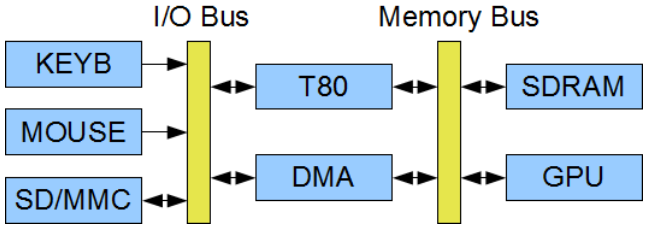
V následujících kapitolách budou detailně vysvětleny všechny zmíněné komponenty.

5.1 PS/2 řadič

Rozhraní PS/2 vniklo ve firmě IBM v pozdějších 80 letech. Svůj název získalo podle série počítačů IBM Personal System/2. Od té doby se rozhraní široce rozšířilo a používá se dodnes. Slouží pro připojení klávesnice, myši, tabletu, touchpadu, trackpointu a podobných periferií. V současné době je pomalu nahrazováno univerzálnější sběrnicí USB, ale přesto je stále velice populární a rozšířené.

Implementace řadiče PS/2 je součástí knihovny FITkit [7] a proto zde nebudu jeho konkrétní implementaci popisovat. Přesto považuji za nutné seznámit čtenáře alespoň s jeho rozhraním.

```
entity PS2_controller is
    port (
```



Obrázek 5.1: Architektura emulátoru ZX Simi

```
        RST     : in   std_logic;
        CLK     : in   std_logic;
        -- Rozhrani PS/2
        PS2_CLK  : inout std_logic;
        PS2_DATA : inout std_logic;
        -- Vstup (zapis do zarizeni)
        DATA_IN  : in  std_logic_vector(7 downto 0);
        WRITE_EN : in  std_logic;
        -- Vystup (cteni ze zarizeni)
        DATA_OUT : out  std_logic_vector(7 downto 0);
        DATA_VLD : out  std_logic;
        DATA_ERR : out  std_logic
    );
end entity;
```

Signály `PS2_CLK` a `PS2_DATA` slouží pro komunikaci pomocí PS/2 protokolu s externím zařízením (klávesnice, myš). Jedná se o synchronní obousměrný protokol master-slave. Hodinový signál vždy generuje slave (např. klávesnice). Master (např. PC) je však nadřazen a má možnost komunikaci zakázat nebo pozastavit.

Při komunikaci slave-master se přijatá data objeví na osmibitové sběrnici `DATA_OUT` a jejich platnost je signalizována bitem `DATA_VLD`.

Při komunikaci master-slave je nutné na osmibitovou sběrnici `DATA_IN` přivést odesílaná data a zápis potvrdit signálem `WRITE_EN`.

5.2 Klávesnice

V kapitole 3.4 byl popsán princip a zapojení klávesnice v počítači ZX Spectrum. Nyní se podívejme na implementaci rozhraní klávesnice v mém emulátoru. Entita klávesnice je následující:

```
entity KEYBOARD is
    port (
        CLK             : in  std_logic;
        RESET           : in  std_logic;
        -- rozhrani pro komunikaci s~procesorem
        ADDR            : in  std_logic_vector(15 downto 0);
        DATA_OUT        : out std_logic_vector(7 downto 0);
        -- rozhrani pro komunikaci s~PS2 klavesnici pripojenou k~FPGA
        K_CLK           : inout std_logic;
        K_DATA          : inout std_logic
    );
end entity;
```

Součástí architektury KEYBOARD je i PS/2 řadič (viz kap. 5.1). Signály `K_CLK` a `K_DATA` vystupují z PS/2 řadiče a jsou přivedeny na I/O piny FPGA. Slouží ke komunikaci s PS/2 klávesnicí.

Procesor periodicky čte stav klávesnice pomocí I/O instrukce. Sběrnice `ADDR` (konkrétně bity 15 až 8) je přímo připojena k adresové sběrnici procesoru a procesor pomocí ní adresuje

konkrétní sloupec klávesnice. Obrázek 3.11 ukazuje reálné zapojení v počítači ZX Spectrum. Na sběrnici `DATA_OUT` je nutné vystavit stav stisknutých kláves ve sloupci, který procesor adresuje.

Adresace sloupce se provádí neustále, a to i v případě, kdy procesor ve skutečnosti nechce klávesnici adresovat. Na sběrnici `DATA_OUT` se proto neustále objevují nějaké hodnoty. Připojení a odpojení sběrnice `DATA_OUT` k procesoru se řeší až vně této architektury.

Při každém přijetí stisknuté klávesy od klávesnice je nastavena log. 0 do odpovídající pozice v registrovém poli `signal keys : reg5x8;`. Klávesy v poli jsou seřazeny stejně jako na obr. 3.11. Pokud je od klávesnice naopak přijat příznak uvolněné klávesy, je v poli `keys` nastavena na odpovídající pozici hodnota log. 1. Každá klávesa má tedy v poli svůj vlastní bit, který udává, zda je klávesa stisknuta (log. 0) či nikoliv (log. 1).

Klávesnice ZX Spectrum obsahuje pouze klávesy A-Z, 0-9, ENTER, SPACE, CAPS SHIFT a SYMBOL SHIFT. Klasická klávesnice má však kláves mnohem více a byla by škoda je nevyužít. Konkrétně se jedná o šipky, celý numerický blok, tečka, čárka a backspace. Každá z těchto kláves má svůj jednobitový registr, který opět indikuje stav klávesy. Z těchto registrů je pomocí kombinační logiky vytvořena maska, která má také rozměr pole 5×8 bitů (stejně jako pole `keys`). Jednotlivé bity kláves jsou v masce rozmístěny tak, aby odpovídaly skutečnému zapojení v počítači ZX Spectrum. Na obrázku 3.10 například vidíme, že šipka nahoru sdílí stejnou klávesu, jako klávesa 7. Jako šipka nahoru je to chápáno pouze v případě, že uživatel drží současně stisknutou klávesu CAPS SHIFT.

Maska je tedy zapojena tak, že pokud uživatel stiskne kurzorovou šipku nahoru, v masce se nastaví log. 0 na pozici klávesy 7 a na pozici klávesy CAPS SHIFT. Podobným způsobem jsou vyřešeny i ostatní speciální klávesy.

S maskou a polem `keys` je proveden logický součin. Podle horních osmi bitů adresy `ADDR` je na výstup `DATA_OUT` připojen jeden z osmi řádků výsledku logického součinu masky a pole `keys`, přičemž zbylé tři datové bity jsou doplněny jedničkami. Konkrétní řádek je vybrán logickou nulou v adrese, zbylé bity adresy musí mít log. 1. Principiálně je tedy `ADDR` přiveden na dekodér 1 z 8 a výstup z dekodéru je přiveden na multiplexor, který na výstup přepíná konkrétní řádek z výsledného pole stisknutých kláves.

5.3 Myš

Myš není standardní součástí počítače ZX Spectrum. Začaly se vyrábět až v pozdějších letech a vzniklo hned několik různých standardů, které bohužel nejsou navzájem kompatibilní. Jejich další nevýhodou je, že obslužné rutiny nejsou uloženy v paměti ROM, takže pokud chce programátor ve své aplikaci myš používat, musí si sám naprogramovat i obslužné rutiny. I přesto však vzniklo mnoho aplikací, ve kterých jsou myši využívány a proto jsem se i já rozhodl, že budu myš ve svém emulátoru podporovat.

Nyní si stručně představíme jednotlivé standardy myší pro počítač ZX Spectrum [4, 19]:

- **Amiga Mouse** - Zásadní nevýhodou je obrovské zatížení procesoru, protože myš je potřeba obsluhovat zcela kompletně programově. Pro další činnost (kromě obsluhy myši) zbývá cca 50% výkonu procesoru, pokud má být pohyb kurzoru opravdu plynulý. Přesto se myš velice rozšířila. K počítači ZX Spectrum se připojuje pomocí paralelního portu UR-4 [5] a ve své podstatě je tedy paralelní, protože používá 4 vodiče pro přenos pulzů ze čtyř enkodérů a další tři vodiče pro přenos stavu tlačítek. Náročnost obsluhy souvisí právě s čtením stavu enkodérů, které se musí provádět velice často, aby myš byla plynulá a nevynechávala.

	7	6	5	4	3	2	1	0
1. bajt	přeteč. Y	přeteč. X	znam. Y	znam. X	1	prostř. tl.	pravé tl.	levé tl.
2. bajt	posun v ose X							
3. bajt	posun v ose Y							

Tabulka 5.1: Standardní datový paket polohy a stavu tlačítek myši

- **Atari Mouse** - Myš je svým chováním identická s Amiga Mouse. Liší se pouze zapojením konektoru.
- **AMX Mouse** - Tato myš se připojuje do systémové sběrnice, na které generuje IM2 přerušení [18] a na datovou sběrnici vystavuje směr pohybu myši. Bohužel má velice slabou softwarovou podporu.
- **Kempston Mouse** - popis v kapitole 5.3.2
- **Ostatní** - Existuje ještě mnoho dalších standardů, většinou se jedná o různé myši z Ruska, jejichž podpora je naprosto mizivá a nebudu je zde proto popisovat. Zmiňme např.: GEOS Mouse a AY Mouse.

5.3.1 Komunikační protokol PS/2 myši

Tato kapitola je zpracována podle [2]. Samotné rozhraní PS/2 bylo popsáno v kapitole 5.1. Nyní se podíváme na komunikační protokol, který PS/2 myši používají.

Standardní myš má dva 9-bitové čítače v doplňkovém kódu, které počítají posun myši v ose X a ose Y. Každý čítač má navíc ještě příznak přetečení. Obsah těchto čítačů společně se stavem tří tlačítek je zasílán hostu ve formě tří-bajtového paketu. Hodnota čítačů reprezentuje relativní změnu polohy zařízení od předchozího odeslání paketu. Při každém odeslání je obsah těchto registrů vynulován. Tabulka 5.1 ukazuje strukturu standardního paketu, který je odesílán hostiteli.

Myš umí pracovat v několika možných režimech:

- **Reset mode** - Do tohoto režimu se myš dostane po zapnutí, nebo při přijetí příkazu „Reset“ (0xFF). Myš při vstupu do tohoto režimu provede tzv. BAT diagnostiku (Basic Assurance Test) a nastavení výchozí parametry: vzorkovací frekvence (100/sekundu), rozlišení (4/mm), škálování (1:1) a zasílání dat (vypnuto).

  Následně myš odešle hostiteli kód 0xAA (BAT successful) nebo 0xFC (Error). Poté myš odešle svoje ID s hodnotou 0x00. Tím se liší od klávesnice a nestandardních myší, které mají své ID různé od 0. Po tomto kroku myš přejde do režimu „Stream mode“.

- **Stream mode** - V tomto režimu myš pravidelně zasílá hostiteli informaci o změně polohy a stavu tlačítek. Frekvence odesílání těchto informací je tzv. vzorkovací frekvence, která je defaultně nastavena na hodnotu 100/sec. Hodnotu lze změnit příkazem „Set Sample Rate“ (0xF3).

  Pozor! Po resetu je bohužel zasílání dat (poloha a stav tlačítek) zakázáno a je nutné jej explicitně povolit zasláním příkazu „Enable Data Reporting“ (0xF4).

  Stream mode je výchozí režim. Po resetu myš přejde právě do tohoto režimu. Lze do něj také přejít příkazem „Set Stream Mode“ (0xEA).

Funkce	port	adresace portu binárně (A15 až A0)
tlačítka	64223 (0xFADF)	xxxxx0x0 11011111
osa X	64479 (0xFBDF)	xxxxx0x1 11011111
osa Y	65503 (0xFFDF)	xxxxx1x1 11011111

Tabulka 5.2: Adresace portů Kempston Mouse

- **Remote mode** - V režimu Remote mode myš průběžně aktualizuje svůj vnitřní stav, ale neodesílá ho hostovi. Místo toho se musí host sám dotazovat pomocí příkazu „Read Data“ (0xEB). Po přijetí příkazu myš odešle jeden datový paket a vynuluje svůj stav. Tento režim není příliš využíván.
- **Wrap mode** - Tento režim se nevyužívá.

Populárním rozšířením standardní PS/2 myši je tzv. „Microsoft Intellimouse Extensions“. Přidává podporu až pro 5 tlačítek a scrollovací kolečko. Data jsou pak odesílána v rozšířeném 4-bajtovém paketu. Toto rozšíření ve své práci nepoužívám.

Protože chci používat standardní režim myši v stream mode režimu, následuje ukázka typické komunikace mezi hostem a myší při inicializaci:

```
Host:  FF  Reset command
Mouse: FA  Acknowledge
Mouse: AA  Self-test passed
Mouse: 00  Mouse ID
Host:  F4  Enable Data Reporting
Mouse: FA  Acknowledge
... Nyní již bude myš automaticky zasílat datové pakety se svým stavem
... Např.: stisknuté levé tlačítko a posunutí myši po ose X o~16 pixelů
Mouse: 09  bit0 = levé tlačítko, bit3 = vždy 1
Mouse: 10  posun v~ose X
Mouse: 00  posun v~ose Y
...
```

5.3.2 Kempston Mouse

Rozhodl jsem se, že ve své práci budu podporovat právě tuto myš. Důvodem je extrémně snadná a rychlá softwarová obsluha a zároveň i snadná hardwarová implementace. Kempston mouse se dočkala řady různých modifikací a vylepšení. Má implementace vychází z verze Kempston Mouse Turbo [19].

Softwarová obsluha myši je velice snadná. Uživatel totiž pouze čte hodnoty z patřičných I/O portů, které jsou ukázány v tabulce 5.2. Kempston Mouse Turbo oproti originální verzi Kempston Mouse využívá plného adresování v dolní části adresy, aby se zamezilo nechtěným konfliktům na sběrnici.

Každá osa má rozsah od 0 do 255 (jeden bajt). Při pohybu myši vpravo je hodnota osy X postupně inkrementována, při pohybu vlevo se hodnota naopak snižuje. To stejné platí i pro osu Y, která je inkrementována při pohybu dolů. Mezní hodnoty ale nejsou ošetřovány, takže při inkrementaci hodnoty 255 dojde k přetečení na hodnotu 0 a naopak při dekrementaci hodnoty 0 dojde k přetečení na hodnotu 255.

Stav tlačítek je dostupný na portu 0xFADF. Bit 0 odpovídá pravému tlačítku, bit 1 levému tlačítku a bit 2 prostřednímu tlačítku. Bity D3 až D7 jsou vždy nastaveny na hodnotu log. 1. Kempston Mouse Turbo ještě navíc přidává podporu 4. tlačítka, já jej však nepodporuji. Log. 0 značí stisknuté tlačítko, log. 1 značí uvolněné tlačítko.

Rozhraní architektury je následující:

```
entity MOUSE is
    port (
        CLK      : in  std_logic;
        RESET    : in  std_logic;
        -- komunikace s~procesorem
        ADDR     : in  std_logic_vector(15 downto 0);
        DATA_OUT : out std_logic_vector(7 downto 0);
        -- signaly pro komunikaci s~PS2 vystupem z~FPGA
        M_CLK    : inout std_logic;
        M_DATA   : inout std_logic
    );
end entity;
```

Součástí architektury je řadič PS/2 popsaný v kapitole 5.1. Signály `M_CLK` a `M_DATA` jsou výstupy z řadiče a slouží pro komunikaci s myší pomocí PS/2 protokolu.

O komunikaci s PS/2 řadičem se stará stavový automat FSM_MOUSE. Jak bylo uvedeno v kapitole 5.3.1, myš po resetu přejde do stream módu, ale zasílání dat je defaultně vypnuto. Proto automat odešle do myši příkaz „Enable Data Reporting“ (0xF4) a očekává odpověď 0xFA. Poté přejde do stavu S_WAIT, ve kterém čeká na přijetí datových paketů se stavem myši.

Automat ignoruje počáteční inicializaci myši, kdy mu myš zasílá informace o stavu BAT testu (0xAA při úspěchu nebo 0xFC při chybě) a dále svoje ID (0x00). Důvodem je zjednodušení hardware. Pokud myš nebude fungovat, uživatel to zajisté sám pozná a proto není potřeba žádné speciální hardwarové ošetřování. Taktéž se využívá vlastnosti, že myš po zapnutí sama provede RESET a není nutné jí explicitně posílat příkaz RESET.

Automat ve stavu S_WAIT čeká na příchozí datové pakety, jejichž struktura je vysvětlena v tabulce 5.1. Pokud je přijat bajt, který má ve 3. bitu hodnotu log. 1, jedná se o první bajt paketu a je uložen do registru `BYTE1`. Druhý bajt paketu je následně uložen do registru `BYTE2` a třetí do registru `BYTE3`.

Po přijetí celého paketu dojde ve stavu S_CHANGE ke zpracování přijatých dat. Do registru `K_BUTTON` se zapíše stav stisknutých kláves z prvního bajtu. Vzhledem k tomu, že Kempston mouse chápe log. 0 jako stisknuté tlačítko a log. 1 jako uvolněné, je nutné provést negaci přijatých hodnot a také změnit pořadí přijatých bitů, aby tlačítka odpovídala rozmístění bitů v Kempston Mouse rozhraní.

Dále se provede aritmetický součet aktuální pozice myši, uložené v registrech `K_AXIS_X` a `K_AXIS_Y`, s přijatou změnou polohy myši, uloženou v registrech `BYTE2` (osa X) a `BYTE3` (osa Y). Již víme, že myš zasílá relativní změnu polohy od předchozího paketu a hodnota je reprezentována v doplňkovém kódu. Pokud je relativní změna polohy myši záporná, aritmetický součet ve skutečnosti provede rozdíl, což je přesně náš cíl.

Procesor pomocí sběrnice `ADDR` adresuje registry z tabulky 5.2. Obsah registrů se okamžitě objevuje na výstupní sběrnici `DATA_OUT`.

5.4 VGA řadič

Implementace řadiče VGA je součástí knihovny FITkit [7] a proto zde opět nebudu popisovat jeho konkrétní implementaci, ale pouze čtenáře seznámím s jeho rozhraním.

```
entity vga_controller is
    generic (
        REQ_DELAY : natural := 1
    );
    port (
        -- Hodiny + reset
        CLK     : in std_logic;
        RST     : in std_logic;
        ENABLE  : in std_logic;
        -- Nastaveni rozliseni (popis v~VGA_CONFIG.VHD)
        -- Pozor: Hodnota musi zustat stale platna!
        MODE    : in std_logic_vector(60 downto 0);
        -- Vstup / pozadavek
        DATA_RED    : in  std_logic_vector(2 downto 0);
        DATA_GREEN  : in  std_logic_vector(2 downto 0);
        DATA_BLUE   : in  std_logic_vector(2 downto 0);
        ADDR_COLUMN : out std_logic_vector(11 downto 0);
        ADDR_ROW    : out std_logic_vector(11 downto 0);
        -- Vystup na VGA
        VGA_RED   : out std_logic_vector(2 downto 0);
        VGA_GREEN : out std_logic_vector(2 downto 0);
        VGA_BLUE  : out std_logic_vector(2 downto 0);
        VGA_HSYNC : out std_logic;
        VGA_VSYNC : out std_logic;
        -- H/V Status
        STATUS_H  : out vga_hfsm_state;
        STATUS_V  : out vga_vfsm_state
    );
end entity;
```

Signály `VGA_RED`, `VGA_GREEN`, `VGA_BLUE`, `VGA_HSYNC` a `VGA_VSYNC` jsou výstupy z řadiče, které se připojují na VGA port vně FPGA.

Výstupy `ADDR_COLUMN` a `ADDR_ROW` společně určují pozici bodu, který se bude zobrazovat. Vstupy `DATA_RED`, `DATA_GREEN` a `DATA_BLUE` jsou hodnoty barev RGB pro právě adresovaný bod obrazovky. Tyto vstupy nejsou nijak potvrzované, řadič si je sám automaticky vzorkuje.

Protože počet taktů mezi požadavkem na data a vlastním vystavením dat se může u každé periferie lišit, existuje v řadiči generický parametr `REQ_DELAY`, pomocí kterého lze nastavit, kolik taktů potřebuje periferie mezi požadavkem (nastavení `ADDR_COLUMN` a `ADDR_ROW`) a vystavením barevných hodnot pro tento požadavek (signály `DATA_{RED,GREEN,BLUE}`). Výchozí hodnota 1 znamená, že ihned v následujícím taktu za požadavkem musí být vystavena platná data.

Pomocí vstupu `MODE` se nastavuje rozlišení obrazu. Popis správného nastavení je v dokumentaci.

Signály `STATUS_H` a `STATUS_V` indikují stav řadiče. Pro mé řešení nejsou důležité.

5.5 Grafická jednotka

Grafická jednotka se stará o autonomní vykreslování obrazu na monitor. Implementace vychází z informací uvedených v kapitole 3.5. Rozhraní entity je následující:

```
entity GPU is
    port (
        CLK         : in  std_logic;
        RESET       : in  std_logic;
        -- okraj (BORDER)
        BORDER_IN   : in  std_logic_vector(7 downto 0);
        BORDER_WR   : in  std_logic;
        -- signaly VGA radice
        VGA_VSYNC   : out  std_logic;
        VGA_HSYNC   : out  std_logic;
        VGA_RED     : out std_logic_vector(2 downto 0);
        VGA_GREEN   : out std_logic_vector(2 downto 0);
        VGA_BLUE    : out std_logic_vector(2 downto 0);
        -- signaly pro komunikaci s~pameti
        ADDR        : out std_logic_vector(15 downto 0);
        READ_EN     : out std_logic;
        DATA_IN     : in  std_logic_vector(7 downto 0);
        DATA_IN_VLD : in  std_logic;
        REFRESH     : out std_logic -- povoleni refresh pameti
    );
end entity;
```

Součástí architektury GPU je i VGA řadič popsaný v kapitole 5.4. Výstupy z řadiče jsou přímo připojeny na výstupy entity GPU. Jedná se o signály s předponou `VGA_`.

Řadič je pomocí svého signálu `MODE` nakonfigurován na rozlišení 640×480 pixelů s obnovovací frekvencí 60Hz. Z předchozích kapitol ale víme, že rozlišení počítače ZX Spectrum je pouze 256×192 pixelů. Rozhodl jsem se, že každý pixel obrazu nahradím čtvercem o rozměrech 2×2 pixely, čímž se obraz zvětší na rozměry 512×384 pixelů. Obraz je posunut do středu obrazovky a kolem něj je zobrazován okraj (BORDER). Horní i spodní okraj je vysoký 48 bodů, levý a pravý okraj je široký 64 bodů, čímž je dosaženo celkového rozlišení 640×480 pixelů.

Grafická jednotka je složena ze dvou stavových automatů. Jeden z nich (FSM_SDRAM) zajišťuje komunikaci s SDRAM pamětí, druhý (FSM_DRAW) obsluhuje řadič VGA.

Z předchozích kapitol víme, že obraz je rozdělen do mřížky 32×24 znaků o rozměrech 8×8 pixelů a každý znak má navíc i svůj vlastní atribut. Protože VGA řadič pracuje postupně po řádcích od souřadnice (0,0) po souřadnici (639,479), nevykresluje se celý znak naráz, ale kreslí se postupně první řádek všech znaků, pak druhý řádek všech znaků atd.

Vždy, když VGA řadič začíná kreslit nový znak (první pixel libovolného řádku ve znaku), automat FSM_SDRAM začne z SDRAM paměti přednačítat novou dvojici bajtů - data a odpovídající atribut znaku. Adresa v paměti se vypočítá podle obrázku 3.8. Z toho důvodu je nutné mít dvě sady registrů. V jedné sadě je uložen atribut a data právě kresleného znaku, do druhé sady registrů se přednačítá následující znak. Tyto dvě sady se pravidelně střídají podle toho, zda se kreslí sudý či lichý znak. Konkrétně jsou multiplexorem přepínány pomocí

signálu `ADDR_COLUMN(4)`, který vystupuje z VGA řadiče a je součástí adresy udávající pozici kresleného bodu v ose X.

Automat FSM_SDRAM pomocí signálu `ADDR` a `READ_EN` vysílá požadavek na čtení z SDRAM paměti. Načtená data se s určitým zpožděním objeví na vstupu `DATA_IN` a jejich připravenost je signalizována signálem `DATA_IN_VLD`. Automat pak přijatá data zapisuje do odpovídajících registrů.

Generický parametr `REQ_DELAY` je ve VGA řadiči nastaven na hodnotu 0, což znamená, že doba taktu mezi požadavkem na data a vlastním vystavením dat je 0 taktů. To je v pořádku, protože včasné přednačtení dat má na starosti automat FSM_SDRAM a data pro VGA řadič jsou tak vždy dostupná v aktuálním taktu. Automat k tomuto účelu používá svoje vlastní počítadlo řádku a sloupce, pomocí kterých se adresuje SDRAM paměť.

Vstupní signály `BORDER_IN` a `BORDER_WR` slouží k zápisu barvy okraje do registru `border`. Adresace registru je prováděna vně architektury. Pokud procesor pomocí vstupně/výstupní instrukce zapisuje na adresu 0xFE, je nastaven signál `BORDER_WR` a barva okraje se uloží do registru.

Automat FSM_DRAW se stará o obsluhu VGA řadiče. Řadič VGA generuje adresu zobrazovaného pixelu a automat FSM_DRAW mu na vstupu přepíná zdroj barvonosného signálu. Pokud je automat ve stavu kreslení horizontálního nebo vertikálního okraje, je na vstupu řadiče připojena barva uložená v registru `border`. Pokud je automat ve stavu kreslení obrazu, jsou zdrojem barev registry s atributem a daty.

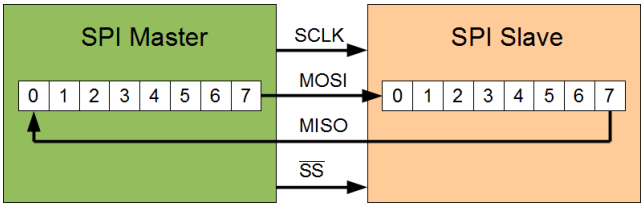
Pokud se právě začíná kreslit poslední znak levého vertikálního okraje, dojde k povolení činnosti automatu FSM_SDRAM, který z SDRAM paměti začne načítat data a atributy následujícího znaku. Díky tomu jsou data dopředu připravena a je možné začít včas vykreslovat obraz. Jakmile je dosaženo pravého okraje, je automat opět pozastaven, aby se zbytečně nezatěžovala paměť.

Tabulka 3.3 ukazuje význam bitů v registru atributu. Konkrétní barva pozadí či popředí je vysvětlena v tabulce 3.4. Pokud je v datovém registru pro aktuální pixel nastavena log. 1, znamená to, že se má bod vykreslit barvou popředí. V opačném případě se bod zobrazí barvou pozadí.

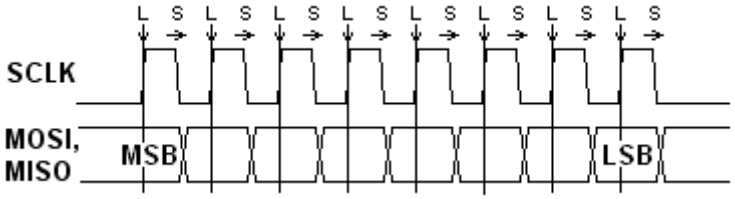
Vzhledem k tomu, že VGA řadič vyžaduje RGB hodnoty barev, je nutné barvy ZX Spectra do tohoto formátu transformovat. To provádí multiplexor mx_color_output. K tříbitové informaci o barvě pixelu (buď popředí nebo pozadí) se přidá čtvrtý bit, který udává jas pixelu (6. bit v registru atributu). Tím získáme čtyřbitovou adresu pro multiplexor, který na svůj výstup přepíná odpovídající RGB hodnotu barvy. Tato hodnota je následně vedena na vstup VGA řadiče.

Další důležitou částí architektury je blikání znaku, které se povoluje 7. bitem v registru atributu. Pokud je blikání povoleno, dochází k přepínání barvy popředí a pozadí s kmitočtem cca 3 Hz. V architektuře je 5-bitový volně běžící čítač `flash_counter`, který se inkrementuje vždy, když dojde k vertikální synchronizaci. Při obnovovací frekvenci 60Hz dochází k blikání s frekvencí $f_{flash} = 60 \div 2^4 = 3,75Hz$. Blikání se odvozuje z obnovovací frekvence obrazovky kvůli úspoře hardware. Pokud by bylo blikání odvozeno od hodinového signálu, byl by potřeba 25-bitový čítač.

Grafická jednotka pomocí signálu `REFRESH` signalizuje, že začala vykreslovat spodní vertikální okraj a nebude tedy delší dobu potřebovat přistupovat k paměti (po celou dobu kreslení spodního okraje a následně i horního okraje). V tuto chvíli je možné bezpečně provést obnovu paměti. Při kreslení obrazu totiž grafická jednotka potřebuje neustále přistupovat k paměti a pokud by se v tu chvíli začal provádět refresh, nebylo by možné včas načíst potřebná data a v obrazu by docházelo k chybám.



Obrázek 5.2: Propojení mastera s jedním slavem



Obrázek 5.3: Časování SPI rozhraní pro SD karty [9]

5.6 SPI rozhraní

SPI (Serial Peripheral Interface) je jednoduché sériové rozhraní, které nalezneme téměř v každém mikrokontroléru a lze pomocí něj snadno připojit různé periferie, které protokol SPI podporují. V našem případě se bude jednat o SD/MMC paměťové karty a paměť FLASH pro uložení obsahu ROM ZX Spectrum.

Komunikace je založena na principu Master-Slave. Všechna zařízení mají společnou datovou sběrnici (MOSI a MISO) a hodinový signál (SCLK). Každý slave má svůj vlastní adresový signál /SS (Slave Select), pomocí kterého master adresuje konkrétního slavea. Princip propojení mastera se slavem je na obrázku 5.2.

Hodinový signál musí generovat host. Signálem /SS se provede aktivace slavea. Samotná komunikace probíhá sériově. V každém taktu si master a slave navzájem vymění jeden bit. Přes vodič MOSI (Master Output Slave Input) master posílá data slaveovi, vodičem MISO (Master Input Slave Output) naopak master přijímá data od slavea. Po osmi taktech má master i slave přijatý celý bajt.

Obrázek 5.3 ukazuje časový průběh SPI rozhraní. Při vzestupné hraně hodin se provede vzorkování vstupního signálu a uložení do posuvného registru, při sestupné hraně se provede posun bitů v registru.

Nyní se podíváme na konkrétní implementaci SPI rozhraní v emulátoru:

```
entity spi is
    port (
        RESET    : in std_logic;
        CLK      : in std_logic;
        -- vstup/vystup
        DATA_IN  : in std_logic_vector(7 downto 0);
        DATA_OUT : out std_logic_vector(7 downto 0);
        WRITE_EN : in std_logic;
        READY    : out std_logic;
        -- SPI rozhrani
        MOSI     : out std_logic;
```

```
        MISO     : in  std_logic;
        SCLK     : out std_logic
    );
end spi;
```

Pomocí signálu `READY` řadič informuje mastera, že je připraven ke komunikaci. Pokud se na vstupním signálu `WRITE_EN` objeví hodnota log. 1, znamená to, že host (master) chce odeslat data. V následujícím stavu se na výstupu `READY` nastaví log. 0 a do datového registru `DATA_IN_REG` se zapíše vstupní hodnota z portu `DATA_IN`.

Nyní se spustí volně běžící čítač `FLASH_CLK`. Jeho nejnižší bit je použit jako výstupní hodinový signál `SCLK`. Komunikace po SPI rozhraní tedy probíhá s poloviční frekvencí, než jaká je do řadiče přivedena signálem `CLK`.

Pomocí bitů `FLASH_CLK(3 downto 1)` se na výstupní datový vodič `MOSI` přepíná jeden z osmi bitů odesílaných dat uložených v registru `DATA_IN_REG`. Zároveň se s každou náběžnou hranou signálu `SCLK` vzorkuje vstupní vodič `MISO` a tento bit se postupně přidává zprava k posuvnému registru `DATA_OUT_REG`. Po osmi taktech máme v tomto registru kompletní načtený bajt od slavea.

Jakmile volně běžící čítač `FLASH_CLK` nabyde hodnoty „1111“, znamená to, že všech 8 bitů bylo odesláno a zároveň i přijato, komunikace se ukončí (volně běžící čítač se pozastaví, aby negeneroval hodinový signál) a dokončení komunikace je indikováno signálem `READY`.

5.7 Řadič SD/MMC karet

Tato kapitola je zpracována podle [9, 8].

SD (Secure Digita) karta je paměťové zařízení určené pro mobilní zařízení, jako jsou fotoaparáty, mobilní telefony apod. Standard SD vznikl jako rozšíření staršího formátu MMC (Multi Media Card). Přináší například digitální správu práv DRM, nové komunikační protokoly (jednobitová a čtyřbitová sběrnice) a nebo schopnost I/O rozšíření.

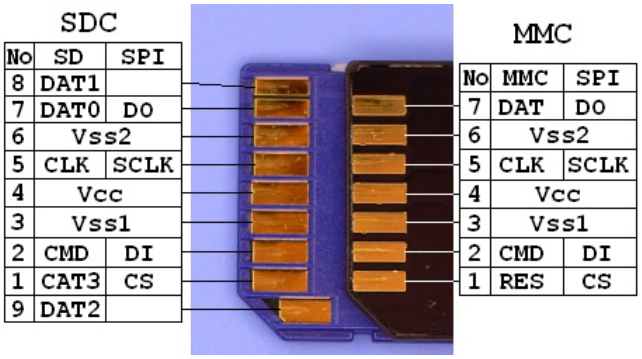
SD karty jsou při dodržení určitých podmínek zpětně kompatibilní s MMC. Řadič, který budu dále popisovat, je primárně určen pro MMC karty a proto by na něm měly fungovat také SD karty.

Data jsou mezi kartou a hostem přenášeny po blocích o velikosti 512 bajtů, takže na vyšší vrstvě můžeme na kartu nahlížet jako na klasický disk a používat na ní např. klasické souborové systémy FAT16/FAT32.

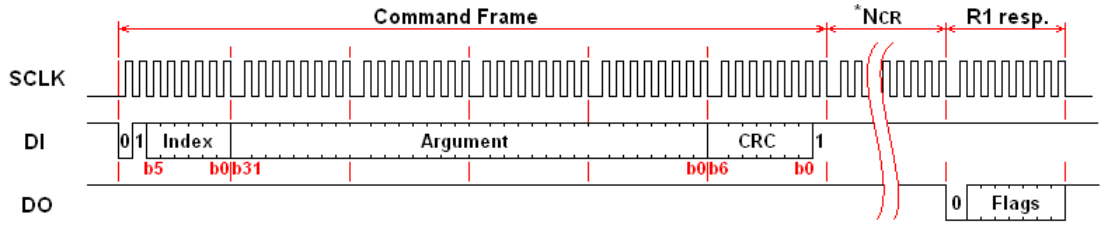
Obrázek 5.4 ukazuje rozmístění vývodů na paměťové kartě MMC v porovnání s paměťovou kartou SD. Vidíme, že MMC má 7 vývodů, kdežto SD jich má rovných 9. Tři z nich jsou napájecí, zbytek jsou aktivní vodiče. Důležité je, že rozmístění vodičů je shodné u obou typů karet.

5.7.1 Komunikační protokol

Oba typy karet podporují sériový protokol SPI (Serial Peripheral Interface), který byl vysvětlen v kapitole 5.6. Ten má obrovskou výhodu ve své jednoduchosti. Většina mikrokontrolérů je totiž vybavena SPI řadičem, takže lze s paměťovou kartou velice snadno komunikovat bez nutnosti používat nějaké speciální rozhraní. Karta sice pomocí SPI nemůže komunikovat s hostem nejvyšší možnou rychlostí, ale to nemusí být u mnoha aplikací na škodu. Ani v našem případě nepožadujeme od karty vysokou komunikační rychlost, protože budeme



Obrázek 5.4: Rozmístění pinů u SD a MMC karty [9]



Obrázek 5.5: Příkazový rámec při komunikaci host → karta [9]

načítat soubory o velikosti několik desítek kilobajtů. Proto se v této práci budeme věnovat pouze SPI módu.

Podle definice MMC nepoužívá standardní SPI časování, ale vzorkování i posun se provádějí při nástupné hraně hodinového signálu. Ukazuje se však, že MMC karta je také schopna pracovat v tomto režimu a proto jsem se ho rozhodl použít pro oba typy paměťových karet.

V SPI režimu se komunikuje po bajtech. Jeden příkazový rámec od hosta do karty má fixní délku 6 bajtů. Jeho struktura včetně časování je na obrázku 5.5. Karta odpovídá příkazem R1, R2 nebo R3. Vzhledem k tomu, že komunikace je řízena hodinami, které generuje host, musí po odeslání svého příkazu stále odesílat bajt 0xFF a přijímat data tak dlouho, dokud neobdrží validní odpověď. Doba mezi příkazem a odpovědí je 0 až 8 bajtů u SD a 1 až 8 u MMC karty. Kontrolní součet CRC je při SPI komunikaci volitelný, ale nelze tento bajt zcela vynechat (a poslední bit musí mít hodnotu log. 1).

První bajt příkazu je složen ze dvou bitů, které indikují začátek příkazu a za nimi následuje 6 bitů indexu (může nabývat hodnot 0 až 63), který určuje číslo příkazu. V tabulce 5.3 jsou vybrány pouze ty nejdůležitější příkazy, které budeme v této práci potřebovat. Kompletní dokumentaci nalezne čtenář v [17].

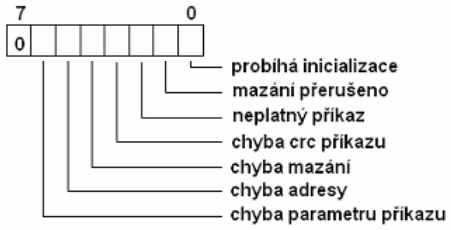
Příkaz ACMD<n> ve skutečnosti znamená sekvenci příkazů CMD55-CMD<n>. Tyto příkazy se však používají pouze u SD karet. MMC je nevyžaduje. Příkaz CMD55 totiž určuje, že příští příkaz bude aplikačně specifický.

Existuje celkem 3 formáty odpovědí. Nás však bude zajímat pouze odpověď R1, jejíž struktura je ukázána na obrázku 5.6. Bit 0 signalizuje, že probíhá inicializace karty. V tuto dobu karta přijímá pouze příkazy CMD0, CMD1, CMD58, CMD59 a ACMD41. Ostatní bity signalizují různé typy chyb.

Po připojení napájecího napětí ke kartě je nutné vyčkat alespoň jednu milisekundu, poté

Index	Argument	Odpověď	Data	Popis
CMD0	Ne (0)	R1	Ne	Reset, výběr protokolu
CMD1	Ne (0)	R1	Ne	Inicializace
CMD17	Adresa[31:0]	R1	Ano	Přečte blok dat
CMD55	Ne (0)	R1	Ne	Musí předcházet každý ACMD<n> příkazy.
ACMD41		R1	Ne	Pouze pro SD. Inicializace karty.

Tabulka 5.3: Přehled nejdůležitějších příkazů pro SD/MMC karty



Obrázek 5.6: Struktura odpovědi R1 [8]

nastavit signály DI a CS do vysoké úrovně a provést alespoň 74 hodinových taktů signálem SCLK. Tím se karta uvede do nativního operačního módu a teprve poté je schopna přijímat další příkazy. Abychom kartu přepnuli do SPI módu, je nutné provést následující akce:

První příkaz, který je nutné kartě zaslal, je CMD0, čímž se provede reset karty a přepnutí do SPI módu. Vzhledem k tomu, že CMD0 je ještě odesláno v nativním módu, je nutné odeslat i správný CRC. Jakmile karta přejde do SPI, není již CRC vyžadováno, pokud jej řadič explicitně nepovolí. Odpovědí karty na příkaz CMD0 je odpověď R0 s nastaveným nultým bitem (hodnota 0x01), kterou signalizuje, že karta čeká na inicializaci.

Nyní je nutné odeslat příkaz CMD1 v případě MMC karty, nebo příkaz ACMD41 (sekvence CMD55 a CMD41) v případě SD karty, čímž se provede inicializace. Odpovědí na tyto příkazy je zpráva R1, která již bude mít v nultém bitu hodnotu log. 0, tedy zpráva 0x00. V tuto chvíli je karta plně inicializována a je možné číst a zapisovat data.

Inicializace SD a MMC karet se tedy liší. Dnešní karty však většinou podporují oba typy inicializace. Proto jsem se rozhodl využít variantu, která se používá u MMC karet, tedy příkaz CMD1.

Nejmenší jednotka dat, kterou lze číst/zapisovat, je jeden blok o velikosti 512 bajtů. Příkaz CMD17 přečte ze zadané adresy jeden blok. Příkazem CMD18 lze přečíst více bloků po sobě. Tuto funkci však nebudeme využívat.

Po odeslání příkazu CMD17 karta odešle odpověď R1. Host poté musí číst vstup tak dlouho, dokud nepřijme úvodní hlavičku datového paketu. Ta má v případě příkazu CMD17 hodnotě 0xFE. Za ní již následuje 512 bajtů dat.

5.7.2 Rozhraní řadiče SD/MMC karet

Nyní se podíváme na horní vrstvu řadiče SD/MMC karet. Již víme, že řadič ke komunikaci používá SPI rozhraní popsané v kapitole 5.6. Vývody pro připojení řadiče k SD/MMC kartě začínají předponou `SD_`.

```
entity sdcard is
    port (
```

CPU_ADDR(15..8)	Registr	Popis
0x00	REG0	CMD příkaz
0x01	REG1	Nejvyšší bajt argumentu
0x02	REG2	Prostřední bajt argumentu
0x03	REG3	Nejnižší bajt argumentu
0x04	REG4	Kontrolní součet CRC
0x05	CONTROL	Řídící registr

Tabulka 5.4: Adresy registrů řadiče SD/MMC

```
        -- Reset a synchronizace
        RESET     : in std_logic;
        CLK     : in std_logic;
        -- Datova a adresova sbernice
        DATA_IN  : in std_logic_vector(7 downto 0);
        DATA_OUT : out std_logic_vector(7 downto 0);
        ADDR     : in std_logic_vector(2 downto 0);
        WRITE_EN : in std_logic;
        READ_EN  : in std_logic;
        WAIT_n   : out std_logic;
        ACK      : out std_logic;
        -- Rozhrani SD karty
        SD_CLK   : out std_logic;
        SD_CS    : out std_logic;
        SD_MOSI  : out std_logic;
        SD_MISO  : in std_logic
    );
end sdcard;
```

Součástí řadiče je i 6 registrů přístupných pro procesor. Jejich horní části I/O adresy jsou uvedeny v tabulce 5.4. Spodní část I/O adresy řadiče SD/MMC je 0xF7. Registry `REGn` odpovídají jednotlivým bajtům v SPI příkazu (viz obr. 5.5). Procesor naplní registry `REGn` a odešle příkaz do SD karty zápisem log. 1 do nultého bitu registru `CONTROL`.

Komunikace s SPI řadičem je řízena pomocí automatu `SD_SPI_FSM`. Po jeho aktivaci postupně do karty odesílá přes SPI rozhraní obsah registrů `REGn` a poté čeká na odpověď R1.

Celý řadič je však řízen pomocí automatu `SD_FSM`, který s SPI řadičem komunikuje prostřednictvím automatu `SD_SPI_FSM`. Po resetu automat nejprve počká 74 taktů na reset SD karty, následně odešle příkaz CMD0 (softwarový reset) a poté CMD1 (inicializace karty). Nyní je řadič připraven přijímat a vyřizovat požadavky na čtení od procesoru.

Při odeslání příkazu čtení (CMD17) je tento příkaz odeslán do paměťové karty a počká se na odpověď R1, kterou karta oznamuje, že příkaz byl dokončen. Nyní bude na jejím výstupu 512 bajtů dat. Při každém nastavení signálu `READ_EN` se z karty přečte jeden bajt. Platnost tohoto bajtu je signalizována bitem `ACK`. Správný počet čtených bajtů si musí hlídat host (buď softwarová aplikaci, nebo jiný externí obvod). Signálem `ACK` řadič informuje, že je přečtený bajt platný.

Během činnosti řadiče je signál `WAIT_n` aktivní (log. 0), čímž řadič pozastaví procesor. Po dokončení operace je procesor opět aktivován nastavením signálu `WAIT_n` do log. 1.

5.8 Procesor T80

Jak již bylo řečeno v kapitole 4.3.1, využiji ve své práci volně dostupnou implementaci procesoru Z80 pod jménem T80 [14]. Nebudu zde pochopitelně popisovat vnitřní implementaci, pouze se seznámíme s rozhraním entity:

```
entity T80s is
    generic(
        Mode    : integer := 0; -- 0 => Z80, 1 => Fast Z80, 2 => 8080, 3 => GB
        T2Write : integer := 0; -- 0 => WR_n active in T3, /=0 => WR_n active in T2
        IOWait  : integer := 1 -- 0 => Single cycle I/O, 1 => Std I/O cycle
    );
    port(
        RESET_n : in std_logic;
        CLK_n   : in std_logic;
        WAIT_n  : in std_logic;
        INT_n   : in std_logic;
        NMI_n   : in std_logic;
        BUSRQ_n : in std_logic;
        M1_n    : out std_logic;
        MREQ_n  : out std_logic;
        IORQ_n  : out std_logic;
        RD_n    : out std_logic;
        WR_n    : out std_logic;
        RFSH_n  : out std_logic;
        HALT_n  : out std_logic;
        BUSAK_n : out std_logic;
        A~: out std_logic_vector(15 downto 0);
        DI      : in std_logic_vector(7 downto 0);
        DO      : out std_logic_vector(7 downto 0)
);
end T80s;
```

Důležité je, že rozhraní entity je shodné se skutečným rozhraním procesoru Z80. Všechny řídící signály jsou aktivní v log. 0 a neaktivní v log. 1. Rozdíl je pouze u datové sběrnice. Zatímco u skutečného procesoru je datová sběrnice současně vstupní i výstupní, implementace T80 rozlišuje samostatné vstupy a samostatné výstupy. Klasické pouzdro procesoru má totiž omezený počet vývodů a sdílením vstupů a výstupů lze ušetřit značné množství vývodů. Uvnitř FPGA však toto nehraje roli a je praktičtější mít vstupy a výstupy oddělené.

Časové průběhy procesoru již byly vysvětleny v kapitole 3.1. Pomocí generických parametrů `T2Write` a `IOWait` je však možné je lehce modifikovat.

Všimněme si generického parametru `Mode`. Lze jím nastavit různé módy procesoru. Ať už standardní Z80, či jeho zrychlenou variantu Fast Z80. Je ho však možné používat i jako klasický procesor 8080, nebo upravenou variantu Z80, která se používala v kapesních handheldech GameBoy.

5.9 SDRAM řadič

Řadič SDRAM je součástí knihovny FITkit. Tato kapitola čtenáře seznámí pouze s rozhraním řadiče, popis implementace je dostupný v [7]. Jsou k dispozici dvě varianty: nízkoúrovňový nebo vysokoúrovňový řadič. Rozhodl jsem se využít nízkoúrovňový řadič, protože umožňuje manuálně obsluhovat obnovu (refresh) paměti. Proto zde popíšu pouze tento.

```
entity sdram_raw_controller is
   generic (
      -- Generovani prikazu refresh radicem automaticky
      GEN_AUTO_REFRESH : boolean := true;
      OPTIMIZE_REFRESH : sdram_optimize := oAlone
   );
   port (
      CLK     : in std_logic;
      RST     : in std_logic;
      ENABLE  : in std_logic;
      BUSY    : out std_logic;
      -- Address/data
      ADDR_ROW    : in std_logic_vector(11 downto 0);
      ADDR_COLUMN : in std_logic_vector(8 downto 0);
      BANK        : in std_logic_vector(1 downto 0);
      DATA_IN     : in std_logic_vector(7 downto 0);
      DATA_OUT    : out std_logic_vector(7 downto 0);
      DATA_VLD    : out std_logic; -- Output data valid
      -- Command signal/set
      CMD     : in sdram_func;
      CMD_WE  : in std_logic;
      -- Signals to SDRAM
      RAM_A      : out std_logic_vector(13 downto 0);
      RAM_D      : inout std_logic_vector(7 downto 0);
      RAM_{DQM,CS,RAS,CAS,WE,CLK,CKE} : out std_logic;
   );
end sdram_raw_controller;
```

Pomocí generického parametru `GEN_AUTO_REFRESH` se povoluje automatický refresh paměti. Při hodnotě True je generován refresh s intervalem 15 us a refresh celé SDRAM (4096 příkazů) proběhne za 61,44 ms lineárně rozložených v čase. Aby nedocházelo k obnovování paměti během kreslení obrazu, je automatické obnovování zakázané a provádí se manuálně odesláním příkazu fRefresh během kreslení horizontálního okraje obrazu.

Pokud je automatické obnovování paměti zakázané, lze parametrem `OPTIMIZE_REFRESH` nastavit, zda bude uživatel zasílat příkazy fRefresh ve skupinách nebo samostatně. Při hodnotě oAlone řadič po provedení příkazu refresh automaticky aktivuje předchozí bank, což urychlí následující příkaz čtení/zápis. Při hodnotě oMultiple řadič očekává více příkazů fRefresh po sobě a proto neaktivuje původní banku.

Adresování u nízkoúrovňového řadiče se realizuje pomocí adresy sloupce (`ADDR_COLUMN`), řádku (`ADDR_ROW`) a banky (`ADDR_BANK`). Adresa musí být platná v době načítání příkazu, tj. na nejbližší nástupné hraně hodinového signálu od nastavení signálů `CMD` a `CMD_WE`.

Existují 3 příkazy: fRefresh (obnova paměti), fWrite (zápis dat do paměti) a fRead (čtení z paměti). Příkaz může být proveden pouze v případě, že signál `BUSY` má hodnotu log. 0. Pokud je ve stavu log. 1, znamená to, že paměť buď ještě nedokončila předchozí příkaz, nebo provádí refresh paměti. V obou případech je nutné počkat na dokončení aktuální operace a teprve pak příkaz odeslat.

Po dokončení příkazu čtení jsou čtená data vystavena na sběrnici `DATA_OUT` a tento stav je po dobu jednoho taktu signalizován signálem `DATA_VLD`. Sběrnice `DATA_OUT` však obsahuje registr, takže data jsou platná až do dalšího čtení.

Příkaz zápisu vyžaduje platná data na vstupu `DATA_IN` v době načítání příkazu. Dokončení zápisu není nijak signalizováno. Doba zápisu je 1 takt, pokud nedojde ke kolizi s požadavkem o automatický refresh nebo aktivaci jiného banku.

Signály s předponou `RAM_` v názvu vystupují z řadiče a pomocí nich se řadič připojuje k externí SDRAM paměti.

5.10 SIMI DMA

Vzhledem k tomu, že při načítání aplikací z SD karty se mezi kartou a pamětí přenáší velké množství dat a softwarové kopírování se ukázalo býti velice pomalé, vznikla potřeba vytvořit speciální hardwarový obvod, který by byl schopen autonomně přenášet větší bloky dat. Tomuto řešení se říká DMA (Direct Memory Access).

V některých periferiích pro počítač ZX Spectrum (např. diskový řadič MB02) se využíval obvod Z80-DMA od firmy Zilog. Nebyl určen pouze pro počítače ZX Spectrum, byl navržen univerzálně a používal se v široké škále aplikací. Podporoval obousměrný přenos mezi pamětí a I/O zařízeními.

Pro můj emulátor však tento obvod není příliš vhodný. V první řadě je zbytečně komplikovaný. Dále má použitá SDRAM paměť mnohem větší kapacitu (8 MB), než kolik je počítač ZX Spectrum 128K, ale i Z80-DMA schopen adresovat. Pokud bych se rozhodl využít pouze oněch 128K paměti, speciální obvod by nebyl třeba. Já však chci mít možnost zápisu/čtení v rámci celé paměti, abych mohl např. přepínat běžící aplikace. Detailní řešení bude popsáno v kapitole 6.2. V tuto chvíli nám stačí pouze informace, že můj DMA obvod zvládne adresovat celých 8 MB paměti a podporuje přenosy mezi pamětí i mezi I/O rozhraními.

Abych můj DMA obvod odlišil od originálního Z80-DMA, pojmenoval jsem ho SIMI DMA, zkráceně SMDMA. Podívejme se nyní na rozhraní obvodu:

```
entity SMDMA is
    port(
        RESET       : in std_logic;
        CLK         : in std_logic;
        --signaly pro komunikaci s~procesorem
        CPU_IORQ_n  : in std_logic;
        CPU_RD      : in std_logic;
        CPU_WR      : in std_logic;
        CPU_WAIT_n  : out std_logic;
        CPU_ADDR    : in std_logic_vector(15 downto 0);
        CPU_DATA_IN : in std_logic_vector(7 downto 0);
        --signaly pro komunikaci s~pameti a IO zarizenimi
        IORQ        : out std_logic;
```

CPU_ADDR(15..8)	Registr	Popis
0x00	ADDRA(7..0)	Nejnižších 8 bitů registru ADDRA
0x01	ADDRA(15..8)	Prostředních 8 bitů registru ADDRA
0x02	ADDRA(22..16)	Nejvyšších 8 bitů registru ADDRA
0x03	DATA_IN	Vstupní data určená k zápisu
0x04	ADDRB(7..0)	Nejnižších 8 bitů registru ADDRB
0x05	ADDRB(15..8)	Prostředních 8 bitů registru ADDRB
0x06	ADDRB(22..16)	Nejvyšších 8 bitů registru ADDRB
0x07	LENGTH(7..0)	Spodních 8 bitů registru LENGTH
0x08	LENGTH(15..8)	Horních 8 bitů registru LENGTH
0x09	CONFIG	Konfigurační registr

Tabulka 5.5: Adresy registrů řadiče SMDMA

```
        MREQ        : out std_logic;
        RD          : out std_logic;
        WR          : out std_logic;
        ACK         : in std_logic;
        ADDR        : out std_logic_vector(22 downto 0);
        DATA_IN     : in std_logic_vector(7 downto 0);
        DATA_OUT    : out std_logic_vector(7 downto 0)
    );
end SMDMA;
```

DMA řadič obsahuje 5 význačných registrů, do kterých může CPU zapisovat a pomocí kterých se řídí datové přenosy. Tyto registry jsou adresovány pomocí horních 8 bitů adresy `CPU_ADDR`. Konkrétní hodnoty jsou uvedeny v tabulce 5.5.

Registr `ADDRA` je široký 23 bitů a udává cílovou adresu pro zápis. Registr `ADDRB` udává zdrojovou adresu pro čtení. Pomocí 16-bitového registru `LENGTH` se nastavuje, kolik bajtů se má v rámci transakce přenést.

Posledním registrem je `CONFIG`, který slouží ke konfiguraci DMA řadiče. Bit 0 povoluje inkrementaci registru cílové adresy (ADDRA) po každém zápisu. Bit 1 rozlišuje zda se má zapisovat do paměti (log. 0) nebo do I/O zařízení (log. 1). Podobně bit 4 povoluje inkrementaci registru zdrojové adresy (ADDRB) a bit 5 rozlišuje, zda se má číst z paměti (log. 0) nebo z I/O rozhraní (log. 1).

DMA řadič podporuje dva režimy činnosti: přenosy bloků dat nebo přenosy bajtů. V obou případech je nejdříve nutné naplnit registry `ADDRA`, `ADDRB` a `CONFIG`. Přenos bloku dat se odstartuje zápisem délky přenášeného bloku (počtu bajtů) do registru `LENGTH`. DMA řadič v tu chvíli převezme kontrolu a autonomně provede transakci. Podle konfigurace v registru `CONFIG` může během činnosti docházet k inkrementaci adresových registrů.

Druhou možností je zápis jednotlivých bajtů. K tomu slouží registr `DATA_IN`. Po zápisu hodnoty do tohoto registru se daná hodnota automaticky zapíše na cílovou adresu.

Pokud procesor čte z libovolné I/O adresy DMA řadiče, řadič automaticky provede čtení ze zdrojové adresy a přečtenou hodnotu vloží na svůj výstup. Díky tomu může procesor číst data z libovolného paměťového místa SDRAM paměti.

5.11 Top level entita

Tato kapitola vysvětluje princip propojení všech komponent (entit), které byly vysvětleny v předchozích kapitolách, do jediného celku. Především se pak věnuje práci s pamětí, protože ta se ukázala být středem celého počítače a zásadním způsobem ovlivňuje jeho výkon.

5.11.1 Generování maskovatelného přerušení INT

V počítači ZX Spectrum je nutné pravidelně generovat maskovatelné přerušení procesoru prostřednictvím signálu INT. Při každém přerušení provede obslužná rutina v BASICu obsluhu klávesnice a magnetofonového rozhraní.

Procesor vzorkuje signál INT při náběžné hraně hodinového signálu během provádění posledního strojového taktu aktuální instrukce (tedy při jejím dokončení). Vzhledem k tomu, že provedení nejdelší instrukce trvá 23 strojových taktů, je nutné podržet přerušovací signál po dostatečně dlouhou dobu. V počítači ZX Spectrum je signál aktivní přesně 32 taktů a je generován vždy při vertikální synchronizaci obrazu, ke které dochází s frekvencí 50 Hz.

Podívejme se nyní na mé řešení. Každý hodinový takt vzorkuji signál `VGA_VSYNC`, který indikuje, že se začala provádět vertikální synchronizace. Při náběžné hraně tohoto signálu se nastaví přerušovací signál INT a také se spustí volně běžící čítač `VGA_VSYNC_COUNT`, který je řízen hodinovým signálem procesoru. Po 32 taktech hodinového signálu procesoru se signál INT zruší a čítač je opět pozastaven.

5.11.2 Fyzický adresový prostor

Jak již bylo řečeno v předchozích kapitolách, rozhodl jsem se implementovat verzi počítače ZX Spectrum 128K. Jeho uspořádání paměťového prostoru bylo popsáno v kapitole 3.2.2. V této kapitole se podíváme na mojí implementaci.

Víme, že počítač ZX Spectrum 128K má dvě paměťové banky ROM o celkové velikosti 32KB a 8 paměťových bank RAM o celkové velikosti 128KB. Použitá SDRAM paměť má celkovou velikost 8MB, takže problém s nedostatkem paměti zde nehrozí. Proto jsem paměť rozdělil především z pohledu snazší implementace.

Paměť RAM mého emulátoru leží od adresy 0x000000 do 0x01FFFF, což odpovídá osmi bankám po 16384 bajtech (dohromady 128KB). Paměti ROM je přidělen adresový prostor od adresy 0x200000 do 0x03FFFF, což opět odpovídá osmi bankám po 16384 bajtech (celkem tedy 128KB). Víme, že ROM je umístěna pouze ve dvou paměťových bankách (dohromady 32KB), zbylých 6 bank je tedy nevyužitých.

Od adresy 0x040000 začíná nový adresový prostor RAM o velikosti 128K. Do této části se načítají aplikace z paměťové karty. V počítači jsou tedy přítomné dva adresové prostory RAM. První z nich leží od adresy 0 a je zde uložen operační systém ZX Simi OS, druhý adresový prostor leží od adresy 0x040000 a používá se pro běh spuštěných aplikací. Podrobné hardwarové řešení je popsáno v kapitole 5.11.3, softwarové řešení je vysvětleno v kapitole 6.2.5.

Tabulka 5.6 ukazuje uspořádání fyzického paměťového prostoru.

Fyzická adresa procesoru

V kapitole 3.2.2 byl vysvětlen princip přepínání aktivní paměťové banky pomocí registru na I/O adrese 0x7FFE. V mé implementaci se registr jmenuje `RAM_CNTRL_REG`. Zápis do registru

Od	Do	Velikost	Popis
0x000000	0x01FFFF	128KB	Paměť RAM pro operační systém ZX Simi OS
0x020000	0x027FFF	32KB	Paměť ROM
0x028000	0x03FFFF	96KB	Volné místo
0x040000	0x05FFFF	128KB	Paměť RAM pro spouštěné aplikace

Tabulka 5.6: Uspořádání fyzického paměťového prostoru v emulátoru

je možný, pouze pokud má 5. bit v registru hodnotu log. 0. Pokud obsahuje hodnotu log. 1, je registr uzamčený proti zápisu a opětovný zápis je možný pouze po resetu.

Nejvyšší dva bity adresy procesoru (A15..A14) jsou použity jako adresový vstup multiplexoru `MX_RAM_CNTRL_ADDR`, který na svůj 19-bitový výstup `RAM_CNTRL_ADDR` vybírá odpovídající adresu pro SDRAM řadič. Pro snadnější pochopení vysvětlím jeho činnost přímo na části kódu.

```
case T80_ADDR(15 downto 14) is
when "00" => -- 0x0000 adresace ROM
  RAM_CNTRL_ADDR <= '0' & '1' & "00" & RAM_CNTRL_REG(4)
                    & T80_ADDR(13 downto 0);
when "01" => -- 0x4000 adresace Video RAM
  RAM_CNTRL_ADDR <= RAM_CNTRL_TASK & '0' & "101" & T80_ADDR(13 downto 0);
when "10" => -- 0x8000
  RAM_CNTRL_ADDR <= RAM_CNTRL_TASK & '0' & "010" & T80_ADDR(13 downto 0);
when "11" => -- 0xC000 adresa mapovatelne pametove oblasti
  RAM_CNTRL_ADDR <= RAM_CNTRL_TASK & '0' & RAM_CNTRL_REG(2 downto 0)
                    & T80_ADDR(13 downto 0);
end case;
```

Výstupní adresa je v podstatě složena ze čtyř částí: výběr oblasti RAM (ZX Simi OS nebo uživatelská aplikace), výběr ROM/RAM, číslo paměťové banky a offset v rámci banky. Nejvyšší bit (18) slouží k výběru aktivní oblasti paměti RAM (registr `RAM_CNTRL_TASK`). Operační systém ZX Simi OS je uložen v oblasti RAM od adresy 0 (log. 0), uživatelská aplikace leží v oblasti RAM od adresy 0x040000 (log. 1). Toto rozdělení je využito při spouštění aplikací z paměťové karty a bude detailně vysvětleno v kapitolách 5.11.3 a 6.2.5.

Následující bit (17.) je nastaven na hodnotu log. 0, pokud chce procesor přistupovat k paměti RAM (leží od adresy 0x000000h), nebo log. 1, pokud chce procesor přistupovat k paměti ROM (leží od adresy 0x020000h).

Následující 3 bity určují číslo banky (0-7) paměti ROM nebo RAM. Zde se v některých případech využívá již zmíněný registr `RAM_CNTRL_REG`. Jeho nejnižší 3 bity udávají číslo aktivní banky (RAM0-RAM7), která má být mapována v adresovém prostoru procesoru od adresy 0xC000h. Čtvrtý bit tohoto registru udává číslo aktivní ROM (ROM0-ROM1)

Poslední část výstupní adresy je vždy nejnižších 14 bitů adresy procesoru. Tato část totiž udává offset v rámci jedné banky o velikosti 16384 bajtů.

Fyzická adresa grafické jednotky

Protože ZX Spectrum 128K umí vykreslovat obraz buď z banky RAM5 (výchozí) nebo RAM7, je nutné fyzickou adresu v paměti vypočítat. O tom, která banka se použije rozhoduje 3. bit registru `RAM_CNTRL_REG`. Protože adresa paměťové banky RAM5 (binárně

CPU_ADDR(15..8)	Registr	Popis
0x00	TASK_ADDR(7..0)	Nejnižších 8 bitů registru TASK_ADDR
0x01	TASK_ADDR(15..8)	Nejvyšších 8 bitů registru TASK_ADDR
0x02	TASK_7FFD	Dočasný registr 7FFD počítače ZX Spectrum 128K
0x03	TASK_CMD	Řídící registr

Tabulka 5.7: Adresy registrů pro přepínání běžících procesů.

101) a RAM7 (binárně 111) se liší pouze v jednom bitu, použije se k adresaci pouze bit RAM_CNTRL_REG(3). Registr RAM_CNTRL_TASK určuje, zda se vykresluje z oblasti RAM operačního systému ZX Simi OS (adresa 0), nebo z oblasti uživatelské aplikace (adresa 0x040000). Celková fyzická adresa se získá takto:

```
RAM_CNTRL_TASK & '0' & '1' & RAM_CNTRL_REG(3) & '1' & gpu_addr(13 downto 0);
```

Všimněme si, že z adresy GPU se použije pouze spodních 14 bitů, které slouží k adresaci uvnitř banky. Jak již víme, její velikost je $2^{14} = 16384$ bajtů.

5.11.3 Přepínání běžících procesů

V kapitole 5.11.2 bylo uvedeno, že v adresovém prostoru se nacházejí celkem dvě paměti RAM o velikostech 128K. První z nich slouží pro běh operačního systému ZX Simi OS (fyzická adresa 0h) a druhý se používá pro spouštěné aplikace z karty (fyzická adresa 0x040000). O výběru aktivní oblasti rozhoduje registr REG_CNTRL_REG. Pro přepínání mezi těmito prostory (respektive běžícími aplikacemi) bylo nezbytné vytvořit speciální hardwarový obvod, který bude registr REG_CNTRL_REG ovládat.

Operační systém uloží do speciálního registru TASK_ADDR adresu. Tato adresa je neustále porovnávána s adresou, ze které procesor čte z paměti. Jakmile se procesor pokusí číst z adresy shodné s adresou TASK_ADDR, dojde k přepnutí adresových prostorů (nastavení registru REG_CNTRL_REG).

V kapitole 3.2.2 byl vysvětlen princip I/O registru na adrese 0x7FFD (v mé implementaci nazvaný RAM_CNTRL_REG). Aplikace, které jsou uloženy jako obraz paměti (formát SNA a Z80) v sobě také uchovávají původní obsah tohoto registru. Operační systém během načítání obrazu uloží přečtenou hodnotu registru 7FFD do pomocného registru TASK_7FFD. Během přepnutí procesu se překopíruje obsah pomocného registru TASK_7FFD do skutečného registru na adrese 0x7FFD, čímž dojde ke korektnímu nastavení prostředí, které bylo uloženo v obrazu SNA nebo Z80.

Přepnutí procesu je nutné explicitně povolit nastavením nultého bitu v kontrolním registru TASK_CMD.

Celý obvod je mapován na I/O adresu 0x1F (spodních 8 bitů I/O adresy). Horních 8 bitů slouží k adresaci jednotlivých registrů obvodu. Detailní informace jsou uvedeny v tabulce 5.7.

5.11.4 Řízení přístupu do paměti

Protože paměť dokáže obsluhovat vždy pouze jeden požadavek o čtení/zápis, je nutné určit prioritu požadavků a zabránit kolizi při více požadavcích současně. Následující seznam ukazuje prioritu požadavků od nejvyšší po nejnižší:

1. Refresh paměti
2. Grafická jednotka - čtení
3. SIMI DMA - zápis
4. SIMI DMA - čtení
5. T80 - zápis
6. T80 - čtení

Vidíme, že nejvyšší prioritu má obnova paměti. To je logické, protože pokud by se neustále objevoval požadavek s vyšší prioritou a obnova paměti by musela čekat na jeho obsloužení, mohlo by dojít porušení obsahu paměti. Požadavek na obnovu paměti je generován manuálně v přesně definovaný moment (během kreslení horizontálního okraje), takže zbytečně neomezuje ostatní požadavky (především čtení dat z grafické jednotky).

Druhou nejvyšší prioritu má grafická jednotka. Ta totiž musí pravidelně číst grafická data a pokud by její požadavky nemohly být včas obslouženy, docházelo by v obraze k artefaktům (poruchám).

Obvod SIMI DMA má vyšší prioritu než procesor. Pokud totiž procesor vydá příkaz přenést určitý blok paměti, musí být akce okamžitě provedena, aby následující instrukce procesoru již pracovala s aktualizovanou pamětí.

Obsluha požadavků a generování signálů pro řadič paměti jsou řešeny pomocí automatu `FSM_SDRAM`. Po resetu automat přejde do stavu S_WAIT, ve kterém čeká na jakýkoliv požadavek. Pokud SDRAM pracuje (signál `BUSY`=1), pak v tomto stavu automat setrvává tak dlouho, dokud se paměť neuvolní. Teprve pak může začít obsluha dalšího požadavku.

Refresh paměti

V kapitole 5.5 bylo vysvětleno, že během vykreslování horizontálního okraje má signál `REFRESH` hodnotu log. 1. Tento signál je vzorkován a při detekci jeho nástupné hrany je do registru `GPU_REFRESH_EN` uložena log. 1. Ta indikuje požadavek na obnovu paměti.

Po vyřízení požadavku na obnovu paměti automatem `FSM_SDRAM` je registr resetován.

Grafická jednotka

Grafická jednotka drží během požadavku na čtení na svém výstupu `READ_EN` hodnotu log. 1 tak dlouho, dokud není její požadavek obsloužen a potvrzen signálem `DATA_IN_VLD`.

Procesor T80

Pokud procesor potřebuje přistoupit k paměti (`MREQ_n`=0), zapíše se do registru `SDRAM_WAIT_n` hodnota log. 0. Tento registr je přiveden na vstup `WAIT_n` procesoru, který pozastaví provádění zápisu nebo čtení, dokud se `WAIT_n` nevrátí zpět do log. 1. Přesný průběh časování je na obrázku 3.4. Paměť tak má dostatek času na vyřízení jeho žádosti.

Samotný požadavek čtení/zápisu procesoru (signály `RD_n`/`WR_n`) je vzorkován a při sestupné hraně signálu je uložen do registru `T80_READ_EN_REG` nebo `T80_WRITE_EN_REG`. Automat `FSM_SDRAM` po vyřízení žádosti signál vyresetuje a může začít obsluhovat další žádosti.

Protože procesor pracuje na mnohem nižší frekvenci než paměť, ukládá se výsledek čtení z paměti do pomocného registru `T80_DATA_IN_MREQ`. Paměť díky tomu může začít okamžitě obsluhovat další požadavek a data pro procesor přesto zůstanou platná.

Kapitola 6

Softwarové řešení

Tato kapitola se zabývá softwarovým řešením emulátoru ZX Spectrum. V první fázi bylo nutné upravit standardní ROM počítače ZX Spectrum 128K. Tato úprava je popsána v kapitole 6.1. Aby bylo možné pracovat s SD/MMC paměťovou kartou, naprogramoval jsem vlastní operační systém ZX Simi OS, který je popsán v kapitole 6.2

6.1 Úprava standardní ROM ZX Spectrum 128K

Standardní ROM počítače ZX Spectrum umožňuje načítat data z magnetofonu. Po zadání příkazu LOAD se spustí obslužná rutina, která vzorkuje magnetofonový vstup, převádí posloupnost bitů na bajty a ty následně dále zpracovává. Protože jsem magnetofon nahradil paměťovou kartou, bylo nutné tuto obslužnou rutinu upravit.

LOAD rutina se nachází v paměti ROM1, která obsahuje původní BASIC 48K (viz kap. 3.2). Při úpravě jsem vycházel z disassemblované verze ROM [24].

Rutina LOAD začíná od adresy 0x556. Její výpis je v příloze C. Červenou barvou jsou označeny instrukce, které je nutné vymazat a nahradit je instrukcí NOP (No Operation). Většina funkcí v rutině se totiž věnuje převodu zvukového signálu z magnetofonu na posloupnost bitů.

Funkce LD-8BITS skládá posloupnost osmi bitů do bajtu, který se následně ukládá do paměti. Proto je nutné přeprogramovat právě tuto funkci. Ihned na její začátek (adresa 0x05CA) se vloží následující kód:

```
LD A,0          ; 0x3E 0x00
IN A,(0xF3)     ; 0xDB 0xF3
LD L,A          ; 0x6F
```

První instrukce nastaví horní část I/O adresy na hodnotu 0. Druhá instrukce přečte ze SIMI DMA řadiče (I/O adresa 0x00F3) bajt dat, který představuje čtená magnetofonová data. Poslední instrukce uloží načtený bajt z registru A do registru L a ten je pak dále zpracován zbytkem obslužné rutiny.

Tato úprava tedy k načítání dat využívá SIMI DMA řadič, který musí být před zavoláním rutiny LOAD správně nakonfigurován. O to se stará operační systém ZX Simi OS. Ten nejprve načte data z SD karty do paměti a následně nakonfiguruje SIMI DMA řadič tak, aby na tato data v paměti ukazoval. Pokud pak uživatel v BASICu spustí příkaz LOAD, magnetofonová data se správně čtou ze zadané pozice v paměti a ne z magnetofonu.

6.2 Operační systém ZX Simi OS

Ukázka a návod k ovládání je v příloze B.

6.2.1 Knihovna s abstraktní vrstvou nad souborovým systémem

Protože základní funkcí ZX Simi OS je práce se soubory, vytvořil jsem abstraktní knihovnu filesystem.c, která dovoluje čtení adresářů a souborů nezávisle na použitém operačním systému. V současnosti je podporován pouze souborový systém FAT32 popsaný v kapitole 6.2.2, knihovnu však lze snadno rozšířit o ostatní souborové systémy.

V hlavičkovém souboru je definována struktura `tFileSystemDir`, která udržuje všechny důležité informace o aktuálním adresáři, jako je např. číslo clusteru a velikost. Dále obsahuje seznam souborů a složek v adresáři. Každý záznam je reprezentován datovou strukturou `tFileSystemDirEntry`, která obsahuje veškeré podstatné informace o souboru/adresáři, např. číslo clusteru, jméno (případně dlouhé jméno) přípona, velikost apod.

Další důležitou strukturou je `tPartitionTableEntry`, ve které je uložena informace o diskovém oddílu (partition). Tato struktura se naplní pomocí funkce:

- `errc fsStart(tFileSystemDir *dir)`.

  Nastaví proměnnou `dir` typu `tFileSystemDir`, do které uloží informaci o kořenovém adresáři (číslo clusteru, velikost a prvních FS_DIR_SIZE souborů a složek v adresáři).

Protože je paměť počítače velice malá, není možné do paměti ukládat kompletní výpis adresáře. Maximální velikost načteného seznamu souborů a adresářů je omezena konstantou FS_DIR_SIZE, která má hodnotu 22. Funkce pro získání seznamu souborů a adresářů má proto možnost zadat offset (posunutí), od kterého chce výpis získat.

- `errc fsDir(unsigned int pos, tFileSystemDir *dir);`

  Tato funkce získá část seznamu souborů a adresářů v daném adresáři. Proměnná `pos` udává offset (posunutí), neboli kolik záznamů má být přeskočeno. V proměnné `dir` je vrácen načtený obsah adresáře.

- `errc fsReadFile(unsigned long fileCluster, unsigned long sectorNumber, unsigned char *data);`

  Přečte jeden sektor ze souboru a vrátí ho v poli `data`. Proměnná `fileCluster` udává číslo clusteru, na kterém soubor začíná. Proměnnou `sectorNumber` se udává číslo sektoru, který chceme přečíst.

6.2.2 Knihovna pro práci se souborovým systém FAT32

Operační systém podporuje paměťové karty se souborovým systémem FAT32, který je široce rozšířen a podporován téměř všemi platformami. Jeho implementace je snadná a proto i vhodná pro malé vestavěné systémy, jakým je i tento emulátor. Vznikl jako rozšíření souborového systému FAT16. Podporuje disky o kapacitě až 16 TB, dlouhé názvy souborů, neomezený počet souborů v adresářích atd.

Z úsporných důvodů zde nebudu souborový systém popisovat. Implementace vychází ze specifikace [13] a nachází se v knihovně fat32.c.

Struktura `tFAT32BootSector` definovaná v hlavičkovém souboru fat32.h slouží pro uchování veškerých důležitých informací o souborovém systému, jako jsou počet sektorů na cluster, počet sectorů v partition, číslo prvního clusteru apod. Tuto strukturu je nutné před prvním použitím naplnit pomocí funkce `FAT32readBootSector`. Všechny ostatní funkce tuto strukturu využívají ke své činnosti.

- `errc FAT32getNextAddrOfCluster(tFAT32BootSector *bootSector, unsigned long addr, unsigned long *next)`

  Funkce pro zadané číslo clusteru zjistí jeho následující cluster. Parametr `addr` udává číslo clusteru, pro který chceme zjistit jeho následníka, parametr `*next` je reference na proměnnou, do které bude číslo následníka uloženo.

- `errc FAT32readDir(tFAT32BootSector *bootSector, unsigned int pos, tFileSystemDir *dir);`

  Přečte obsah adresáře. Protože počet položek v adresáři není nijak omezen, může být vrácený výsledek příliš velký a nevešel by se do paměti. Proto je počet vrácených položek omezen konstantou `FS_DIR_SIZE` definovanou v souboru filesystem.h. Parametr `pos` udává číslo souboru/adresáře ve FAT tabulce, od kterého chci získat výpis. Výsledek se uloží do pole `dir`. Soubory ani adresáře nejsou ve FAT tabulce nikterak seřazeny. Pořadové číslo tedy odpovídá skutečnému umístění záznamu ve FAT tabulce.

- `errc FAT32readFile(tFAT32BootSector *bootSector, unsigned long firstCluster, unsigned int sectorNumber, unsigned char *data);`

  Přečte jeden sektor souboru. Proměnná `firstCluster` udává číslo prvního clusteru, na kterém soubor začíná, parametrem `sectorNumber` lze zvolit, kolikátý sektor souboru chceme načíst. Pomocí FAT tabulky se zjistí fyzické umístění sektoru na disku a jeho obsah je vrácen v poli `data`.

Knihovna FAT32 ke své činnosti využívá knihovnu sdcard.c, pomocí které přistupuje k SD paměťové kartě. Knihovna je vysvětlena v kapitole 6.2.3.

6.2.3 Knihovna pro práci s SD/MMC kartou

Tato knihovna komunikuje s SD/MMC řadičem popsaným v kapitole 5.7.2. Bázová adresa řadiče SD_ADDR=0xF7 je definována v hlavičkovém souboru sdcard.h.

Knihovna poskytuje pouze jedinou funkci, která z SD/MMC karty přečte jeden sektor zadaný jeho adresou uloženou v proměnné `sector`. Přečtený sektor je navrácen v poli `data`. Prototyp funkce je následující:

- `errc sdReadSector(unsigned long sector,unsigned char *data);`

Funkce odešle SD kartě příkaz CMD17 (čtení sektoru) s adresou sektoru, poté softwarovou smyčkou čeká na přijetí startovacího tokenu 0xFE, který označuje začátek dat. V tuto chvíli je SD/MMC řadič připraven odesílat data. Pro zrychlení přenosu se celý sektor přenese do paměti pomocí SIMI DMA řadiče. Funkce sdReadSector nastaví SIMI DMA řadič tak, aby data četl z I/O rozhraní SD/MMC řadiče a data zapisoval do paměti (viz kap. 6.2.4). Poté nastaví délku dat na 512 bajtů (celý sektor), čímž se odstartuje DMA přenos.

6.2.4 Knihovna pro práci se SIMI DMA řadičem

Architektura SIMI DMA řadiče byla popsána v kapitole 5.10. Knihovna dma.c slouží pro ovládání SIMI DMA řadiče procesorem. V hlavičkovém souboru dma.h je definována bázová I/O adresa řadiče DMA_BASE_ADDR=0xF3 a funkce pro ovládání řadiče:

- `void dmaSetDestAddr(unsigned long addr);`

  Nastaví cílovou fyzickou adresu.

- `void dmaSetSrcAddr(unsigned long addr);`

  Nastaví zdrojovou fyzickou adresu.

- `void dmaSetConfig(unsigned char conf);` Nastavení parametrů přenosu. Ve výchozím stavu (`conf`=0) je řadič nastaven tak, že zdrojová i cílová adresa využívá paměťový prostor a registry s cílovou a zdrojovou adresou nejsou během přenosu inkrementovány. Pomocí následujících příznaků lze toto chování změnit:

  - `DMA_INC_DST` - Inkrementace cílové adresy po každém zapsaném bajtu.
  - `DMA_IORQ_DST` - Zapisovat data do I/O adresového prostoru.
  - `DMA_INC_SRC` - Inkrementace zdrojové adresy po každém přečteném bajtu.
  - `DMA_IORQ_SRC` - Číst data z I/O adresového prostoru (např. SD/MMC řadič).

- `void dmaTransfer(unsigned int length);`

  Odstartuje přenos bloku dat o velikosti `length` ze zdrojové adresy do cílové adresy.

- `unsigned char dmaReadByte();`

  Přečte jeden bajt ze zdrojové adresy.

- `void dmaSendByte(unsigned char byte);`

  Uloží jeden bajt `byte` do cílové adresy.

6.2.5 Načítání a spouštění aplikací z SD/MMC karty

Tato kapitola popisuje princip načítání a spouštění aplikací ze souborů uložených na paměťové kartě. Existuje velké množství různých formátů pro uložení aplikací a dat pro ZX Spectrum, nejpoužívanější jsou zřejmě formáty SNA, Z80, TAP a TZX, které podporuje i operační systém ZX Simi OS. Strukturu formátů zde z prostorových důvodů nebudu uvádět, podrobnou specifikaci nalezne čtenář např. v [10].

Protože počítače ZX Spectrum standardně načítají data z magnetofonové pásky, nejpřirozenějším formátem pro uložení dat na klasickém PC je obraz magnetofonové pásky ve formátu TAP nebo TZX. Naprostá většina aplikací na internetu je uložena právě v tomto formátu. Druhý způsob, jak aplikace a data uložit, je pomocí obrazu paměti. Příkladem jsou formáty SNA a Z80.

Aby bylo možné aplikace spouštět, musel jsem vytvořit pomocnou knihovnu task.c, která umožňuje za běhu vygenerovat assemblerový (respektive binární) kód, uložit jej do paměti a následně jej procesorem provést. Tato funkce se používá ke spouštění obrazů paměti (SNA, Z80). Každý soubor v sobě totiž kromě obrazu paměti obsahuje také obraz všech registrů procesoru. Operační systém během načítání souboru vygeneruje patřičný binární kód, který má za cíl naplnit registry procesoru správnými hodnotami.

V kapitole 5.11.3 byl popsán speciální obvod, který zajišťuje korektní přepnutí paměťových prostorů z paměti operačního systému ZX Simi OS na paměť spouštěné aplikace. Knihovna task.c podporuje zápis do registrů tohoto obvodu.

Formát SNA

Prvních 27 bajtů v obrazu obsahuje obsah všech registrů procesoru. Během načítání se do libovolného volného místa v paměti pomocí knihovny task.c vygeneruje binární kód, po jehož provedení se inicializují všechny registry. Před jeho provedením se však nejprve nakopíruje zbytek souboru obrazu paměti do paměti RAM aplikace, která začíná od fyzické adresy 0x040000, poté se nastaví registr `TASK_ADDR` z kapitoly 5.11.3 tak, aby ukazoval za poslední instrukci vygenerovanou knihovnou task.c, inicializuje se pomocný registr `TASK_7FFD` hodnotou z obrazu a povolí se činnost přepínacího obvodu pomocí registru `TASK_CMD`. V tuto chvíli je počítač připraven ke spuštění aplikace. Provedením vygenerovaného kódu se inicializují všechny registry, HW obvod rozpozná dokončení inicializace a přemapuje aktivní paměť RAM na adresu 0x040000.

Poslední vygenerovanou instrukcí je vždy skok na adresu, čímž se provede inicializace registru Program Counter. Aplikace díky tomu může začít přesně od místa, kde byl obraz uložen.

Pro urychlení načítání se obraz paměti načítá z karty pomoci SIMI DMA řadiče.

Formát Z80

Načtení souboru Z80 je velice podobné souboru SNA. Tento formát je však v mnoha ohledech sofistikovanější. Existuje několik verzí, které jsou však naštěstí zpětně kompatibilní. Zřejmě nejvýraznější změnou je podpora komprese souboru pomocí RLE (Run-length Encoding). Kvůli ní však není možné obraz kopírovat pomocí SIMI DMA řadiče, ale je nutné obraz načítat a dekomprimovat softwarově, což je výrazně pomalejší.

Formát TAP

Již jsme si vysvětlili, že formát TAP není obraz paměti počítače, ale obraz magnetofonové pásky a je proto nutné načítat jej zcela jiným způsobem, než formáty SNA a Z80.

Nejprve se pomocí SIMI DMA řadiče přenese soubor z SD/MMC karty do volného místa v SDRAM paměti, konkrétně na adresu 0x80000, která je zcela prázdná a k ničemu se nevyužívá. Pomocí knihovny task.c se vygeneruje kratičký kód, který pouze nastaví registr PC (Program Counter) procesoru na hodnotu 0, čímž dojde k softwarovému resetu procesoru. Při něm se opět spustí upravená ROM ZX Spectrum 128K, popsaná v kapitole 6.1.

V tuto chvíli již stačí v BASICu spustit příkaz LOAD, který zajistí korektní načtení magnetofonových dat z adresy 0x080000.

Formát TZX

Načtení formátu TZX je téměř stejné jako u TAP. Liší se pouze vnitřní struktura souboru, která v sobě obsahuje nejen TAP data, ale i další informace související s frekvencí tónu na kazetě, délkou pulzů a pod. Tyto informace je nutné během načítání přeskočit.

Kapitola 7

Výsledná fyzická realizace

V kapitole 4 jsem vysvětlil, proč jsem se pro implementaci rozhodl využít vývojový kit FITkit. Po ověření funkčnosti na platformě FITkit jsem se ale rozhodl, že pro emulátor vytvořím vlastní plošný spoj, který bude pokud možno co nejmenší a nejjednodušší a bude obsahovat všechny potřebné konektory pro připojení periferií. Osazený plošný spoj je na obrázku 7.1.

Při návrhu vlastního zapojení jsem vycházel ze schéma FITkitu [7]. Z něj jsem převzal především zapojení stabilizátorů napětí $3,3V$ a $1,2V$. Použil jsem však jiný typ FPGA, konkrétně Spartan XC3S400AN-FGG400 [22], protože toto FPGA v sobě obsahuje FLASH paměť pro uložení konfiguračního řetězce pro FPGA. Flash paměť je navíc natolik veliká, že je do ní možné uložit i uživatelská data. Díky tomu není nutné používat externí FLASH paměť.

Celé schéma se nachází v příloze D. Kvůli použitému FPGA, které má BGA pouzdro se 400 piny, bylo nutné použít čtyřvrstvý plošný spoj. Spodní vrstva je použita k rozvodu GND signálu. Horní vrstva obsahuje datové vodiče a SMD součástky. Vnitřní vrstvy slouží pro rozvod napájecího napětí a dalších datových vodičů.

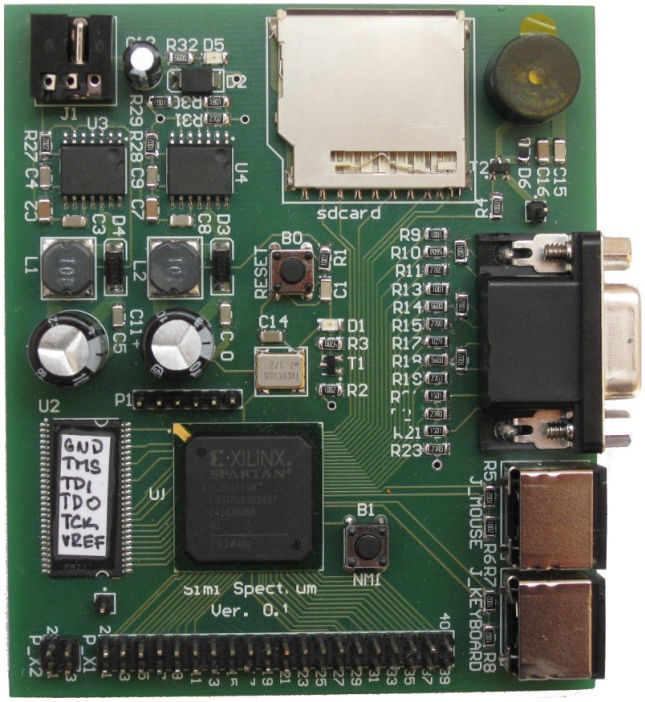
Pro možnost budoucího rozšiřování emulátoru jsem z FPGA vyvedl 40 univerzálních I/O pinů na externí konektor P_X1. Lze je využít např. k připojení joysticku či jiných periferií. Na konektoru P_X2 jsou vyvedena napájecí napětí $5V$ a $3,3V$ a také GND.

FPGA se programuje pomocí JTAG rozhraní vyvedeného na konektor P1. Na desce se také nachází tlačítko B0 pro RESET a dále univerzální tlačítko B1, které bylo původně zamýšleno pro nemaskovatelné přerušení NMI, ale v současné chvíli není využíváno a jeho funkce tedy může být v budoucnu jiná.

Hodinový kmitočet pro FPGA se generuje pomocí oscilátoru s frekvencí $7,372MHz$. Z tohoto kmitočtu se pomocí DCM obvodu v FPGA generuje vnitřní kmitočet 50MHz, který se používá jako hodinový kmitočet celého emulátoru.

Napájecí napětí 5V, které se používá jako napájení např. pro PS2 periferie, je přivedeno přímo ze vstupu a není nijak stabilizováno. Celé zařízení proto musí být napájeno přesně 5 volty. Při nedodržení může dojít k poškození, případně nesprávnému chování. To je bohužel značná nevýhoda tohoto zapojení. Praktičtější by bylo mít stabilizované i napětí 5V.

Další drobnou chybou zapojení jsou chybějící filtrační kondenzátory u FPGA. Plošný spoj sice funguje správně i bez nich, ale jistě by bylo vhodnější tyto kondenzátory zapojit.



Obrázek 7.1: Plošný spoj emulátoru ZX Simi

7.1 Externí paměť FLASH

Po otestování osazeného plošného spoje jsem zjistil, že pro uložení operačního systému ZX Simi OS a ZX Spectrum 128K ROM nemohu využít vnitřní FLASH paměť v FPGA. Programovací software Xilinx Impact totiž z nepochopitelného důvodu během programování vždy zamrzne a FPGA se nepodaří naprogramovat. Zřejmě se jedná o chybu v aplikaci Xilinx Impact. Toto chybu jsem nahlásil na oficiální podpoře firmy Xilinx, ale doposud jsem neobdržel žádnou odpověď. K chybě dochází na všech testovaných operačních systémech (Windows 7, XP, Linux) i na různých verzích aplikace Xilinx Impact (13.4, 12.1).

Z tohoto důvodu jsem byl nucen použít externí paměť FLASH AT45DB041D, kterou jsem připojil do konektoru P_X1. Jedná se o 4Mbit FLASH paměť, které pracuje na frekvenci až 66MHz. S pamětí se komunikuje pomocí SPI rozhraní (viz např. kap. 5.6).

Upravená ROM ZX Spectrum 128K je uložena od adresy 0 a její velikost je 128K. Ihned za ní je v paměti uložen operační systém ZX Simi OS ve formátu TAP. Po spuštění emulátoru se pomocí komponenty `FLASH_EXT` z FLASH paměti načte ROM ZX Spectrum 128K a uloží se do SDRAM paměti na fyzickou adresu 0. Dále se z FLASH paměti načte ZX Simi OS a uloží se do SDRAM paměti ihned za načtenou ROM (0x8000) Na tuto adresu je po resetu také nastaven obvod SIMI DMA.

Podívejme se nyní na rozhraní obvodu `FLASH_EXT`:

```
entity FLASH_EXT is
    port (
        CLK      : in  STD_LOGIC;
        RESET    : in  STD_LOGIC;
        WAIT_n   : out std_logic;
        -- externi FLASH
```

```
        MISO     : in std_logic;
        MOSI     : out std_logic;
        CSB      : out std_logic;
        SCLK     : out std_logic;
        -- rozhrani po komunikaci s~pameti
        DATA_OUT : out  STD_LOGIC_VECTOR (7 downto 0);
        ADDR     : out  STD_LOGIC_VECTOR (22 downto 0);
        WR       : out  STD_LOGIC;
        ACK      : in   STD_LOGIC
    );
end FLASH_EXT;
```

Pomocí signálu `WAIT_n` se pozastavuje činnost procesoru. Obvod díky tomu může procesor pozastavit do té doby, než do paměti uloží ROM ZX Spectrum 128K a operační systém ZX Simi OS.

Komunikace s externí FLASH pamětí probíhá pomocí signálů `MISO`, `MOSI`, `CSB` a `SCLK`, což jsou standardní signály SPI rozhraní. Dále je obvod připojen k paměti SDRAM pomocí signálů `DATA_OUT`, `ADDR`, `WR` a `ACK`.

Obvod je řízen pomocí konečného automatu. Nejdříve se do FLASH paměti odešle příkaz čtení z adresy 0: 0x0B, 0x00, 0x00, 0x00. Po odeslání příkazu se z FLASH přečte jeden „don't care" bajt a po něm již následuje sekvence čtených bajtů. Ty se postupně ukládají do SDRAM paměti od adresy 0.

7.2 Spuštění a ovládání emulátoru

Jak tedy probíhá spuštění celého emulátoru? Obvod `FLASH_EXT` po resetu pozastaví činnost procesoru signálem `WAIT_n`. Následně z FLASH paměti přečte ROM ZX Spectrum 128K a uloží ji do SDRAM paměti na adresu 0x020000. Na adresu 0x0x028000 uloží TAP obraz operačního systému ZX Simi OS. Po dokončení načítání je zrušen signál `WAIT_n`, čímž se opět povolí činnost procesoru T80, který spustí BASIC.

V tuto chvíli je již možné s počítačem normálním způsobem pracovat. Pokud uživatel zadá v BASICu příkaz LOAD, spustí se operační systém ZX Simi OS, který je uložený v SDRAM paměti ve formátu TAP a obslužná rutina v ROM jej načte a spustí jako klasickou uživatelskou aplikaci z magnetofonu. Tento princip byl vysvětlen v kapitole 6.1.

Pomocí operačního systému ZX Simi OS je možné spouštět aplikace z SD/MMC paměťové karty. Systém byl popsán v kapitole 6.2.

Kapitola 8

Ověření funkčnosti emulátoru

Abych ověřil funkčnost celého emulátoru, použil jsem k jeho otestování dobové aplikace a především hry. Především hry jsou pro testování ideální, protože využívali veškeré možnosti počítače „nadoraz“. Vzhledem k tomu, že jsou jich stovky, není pochopitelně v mých silách ověřit funkčnost všech. Vybral jsem si proto k otestování jen jejich malou podmnožinu, především mé oblíbené hry.

Základní funkčnost celého zařízení je zřejmá již z toho, že počítač dokáže spustit modifikovanou ROM ZX Spectrum 128KB a následně načíst a spustit operační systém ZX Simi OS. Vzhledem k tomu, že operační systém správně zobrazuje obsah souborů a složek na kartě a správně spouští aplikace, považuji i jeho implementaci za správnou.

Z testovaných her zmiňme např. hry Dizzy 1-9, Batman 1, Jetpack, The Simpsons, Boulderdas, Cauldron 1-2, Bionic Commando, F-19, Golden Axe, Pacmania, Indy 1-4, The Race Against Time, Rampage, Mortal Kombat a mnoho dalších. Funkčnost Kempston myši jsem prověřil na hrách Simcity, Wolfenstein, Lemmings, Ikari Warriors, Carrier Command a Arkanoid 2.

Bohužel jsem však narazil i na nefunkční hry. Např: Arkanoid 1, Batman 3, Prince of Persia, Space Crusader. Konkrétně u hry Arkanoid jsem se dočetl, že čte z nedefinovaných I/O portů, čímž zřejmě načítá náhodná data ze sběrnice. Vzhledem k tomu, že toto chování není nijak specifikované, nepodařilo se mi tuto hru zprovoznit.

Tehdejší hry se z důvodu vyšší rychlosti často programovaly přímo v assembleru a programátoři se snažili využít veškeré možnosti tehdejších počítačů - jeho zdokumentované chování, ale i to nezdokumentované či dokonce jeho chyby. Domnívám se, že právě to je důvodem, proč některé hry nefungují správně.

Konkrétně hra Arkanoid 1 ze stejného důvodu nefunguje ani na originálním ZX Spectrum 2+, což je důkaz, že některé hry opravdu byly šité přímo na míru danému počítači.

V celkovém měřítku však většina her funguje správně a proto považuji emulátor za funkční.

Kapitola 9

Závěr

Cílem práce bylo navrhnout a implementovat pomocí technologie FPGA emulátor počítače ZX Spectrum 48K. V úvodu práce jsem čtenáře seznámil s počítači ZX Spectrum - jejich historií, verzemi počítačů a jejich vnitřní architektuře.

K implementaci emulátoru jsem mohl využít libovolný vývojový kit dostupný na škole. Rozhodl jsem se pro platformu FITkit, se kterou jsem měl dobré zkušenosti z předešlého studia.

Zadání práce jsem si značným způsobem rozšířil. V první řadě jsem se rozhodl vytvořit emulátor počítače ZX Spectrum 128K, který je s počítačem ZX Spectrum 48K zpětně kompatibilní, avšak přináší velké množství zásadních změn - především rozšíření paměti ROM i RAM a s tím související změny. Mimo jiné používá i 3-kanálový zvukový generátor AY-3-8912, který jsem však z důvodu zjednodušení neimplementoval.

Po ověření funkčnosti emulátoru na platformě FITkit jsem nad rámec zadání navrhl vlastní plošný spoj, který je „šitý na míru“ emulátoru - obsahuje veškeré potřebné konektory pro připojení periferií a dovoluje používat emulátor i bez přítomnosti počítače, který by jej například musel po zapnutí naprogramovat.

Plošný spoj jsem navrhl tak, aby bylo do budoucna možné rozšiřovat funkce emulátoru připojením dalších periferií k FPGA pomocí rozšiřujícího portu. Toho by se dalo využít např. pro připojení joysticku, externího zvukového generátoru AY-3-8912 apod. Vzhledem k tomu, že je navržen univerzálně, bylo by ho možné použít k implementaci i dalších osmibitových počítačů nebo zařízení, jako je např. Atari nebo GameBoy.

Drobnou vadou na kráse navrženého plošného spoje je, že jsem nepoužil stabilizátor napětí na 5V, takže je nutné celé zařízení napájet již stabilizovaným napětím 5V, jinak by mohlo dojít k poškození PS/2 periferií.

Aby bylo možné v emulátoru přistupovat k SD/MMC paměťovým kartám, musel jsem také naprogramovat vlastní operační systém ZX Simi OS, který v současnosti podporuje souborový systém FAT32 a spouštění aplikací uložených ve formátu SNA, Z80, TAP a TZX.

Abych ověřil funkčnost celého emulátoru (plošný spoj, HW emulátor v FPGA a operační systém ZX Simi OS), otestoval jsem celou platformu pomocí dostupných dobových aplikací a her. Valná většina aplikací funguje správně. Vzhledem k tomu, že se tehdejší aplikace programovaly přímo na míru počítače (často přímo v assembleru) a programátoři se snažili využít každou vlastnost počítače (často i jeho nezdokumentované chování), existují i aplikace, které na mém emulátoru nefungují. Zmiňme například hru Arkanoid.

I přes zmíněné problémy se mi však podařilo vytvořit funkční hardwarový emulátor počítače ZX Spectrum 128K a jeho funkčnost náležitě otestovat na dobovém software.

V rámci dalších bakalářských a diplomových prací by bylo například možné rozšířit

funkce emulátoru o podporu dalších typů myší, joysticků, již zmíněného zvukového generátoru, případně přidat podporu tzv. ULA+ módu, což je rozšíření, které umožňuje na počítači ZX Spectrum zobrazovat až 64 barev. Také by bylo užitečné přidat podporu ukládání na SD/MMC kartu. To by znamenalo implementovat podporu zápisu na souborovém systému FAT32 v operačním systému ZX Simi OS a upravit obslužnou rutinu v ROM ZX Spectrum. Plošný spoj je navíc možné použít jako výchozí platformu pro implementaci zcela jiných emulátorů (Atari, GameBoy…).

Literatura

[1] BROWN, Stephen, ROSE, Jonathan. Architecture of FPGAs and CPLDs: A Tutorial. [online]. 41 [cit. 2011-12-29].
URL http://www.site.uottawa.ca/~rabielmo/ceg3156_s11/docs/FPGA_CPLDTutorial.pdf

[2] CHAPWESKE, Adam. The PS/2 Mouse Interface. [online]. [cit. 2012-01-30].
URL http://www.computer-engineering.org/ps2mouse/

[3] Clive Sinclair. In *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, [cit. 2011-11-23].
URL http://en.wikipedia.org/wiki/Clive_Sinclair

[4] Cygnusova stránka nejenom o ZX Spectru: Myši a ZX Spectrum. [online]. [cit. 2012-04-08].
URL http://cygnus.speccy.cz/popis_mysi.php

[5] Cygnusova stránka nejenom o ZX Spectru: UR-4 (paralelní port s 8255A). [online]. [cit. 2012-04-08].
URL http://cygnus.speccy.cz/popis_mhb8255.php

[6] Emulátory pod Linux. [online]. [cit. 2011-12-28].
URL http://www.ms.mff.cuni.cz/~masiv3am/data/Emulatory/Emulatory.htm

[7] FITkit. FIT VUT BRNO. [online]. [cit. 2011-12-30].
URL http://merlin.fit.vutbr.cz/FITkit/

[8] GAZDA, Martin. *Využití paměťové karty SD na platformě FITkit.* Brno, 2008. Bakalářská práce. FIT VUT v Brně. Vedoucí práce Zdeněk Vašíček.

[9] How to Use MMC/SDC. [online]. 2010-11-03 [cit. 2012-02-04].
URL http://elm-chan.org/docs/mmc/mmc_e.html

[10] JAHN, Claus. Overview ZX: Inofficial ZX-Spectrum emulator file formats site. [online]. [cit. 2012-04-09].
URL http://www.zxmodules.de/

[11] KŘEHLÍK, Karel. *Esej do předmětu P109 - Historie a vývojové trendy ve VT* [online]. 1999 [cit. 2011-11-23].
URL http://www.fi.muni.cz/usr/jkucera/pv109/xkrehlik.htm

[12] MAŤÁTKO, Jan. *Elektronika.* 5. vyd., V [nakl.] Idea servis 3., rozš. vyd. Praha: Idea servis, 2002, 327 s. ISBN 80-85970-42-2.

[13] MICROSOFT CORPORATION. Microsoft Extensible Firmware Initiative: FAT32 File System Specification. Version 1.03, December 6, 2000.
URL http://msdn.microsoft.com/en-us/windows/hardware/gg463080

[14] OpenCores: T80 CPU. [online]. [cit. 2012-03-02].
URL http://opencores.org/project,t80

[15] SMITH, Chris. *The ZX Spectrum ULA : How to Design a Microcomputer*. First Edition. United Kingdom : ZX Design and Media, 2010. 300 s. ISBN 978-0-9565071-0-5.

[16] *Amatérské radio: řada pro konstruktéry*. Praha: Amaro, 1989, XXXVIII(1).

[17] *SD Card Specification: Simplified Version of Part E1*. Version 1.0. 719 San Benito St. Suite C Hollister, CA 95023 USA: SD Association, 2001, 29 s.
URL http://www.sandisk.com/Assets/File/OEM/Manuals/SD_SDIO_specsv1.pdf

[18] *Z80 Family CPU User Manual* [online]. ZiLOG Worldwide Headquarters, revision 05, 2004-12 [cit. 2011-11-26].
URL http://www.zilog.com/docs/z80/um0080.pdf

[19] VELESOFT: Kempston Mouse Turbo 2008. [online]. [cit. 2012-01-31].
URL http://velesoft.speccy.cz/kmturbo2008-cz.htm

[20] VÍT, Vladimír. *Základy televizní techniky*. 1. vyd. Praha: SNTL, 1987, 358 s. ISBN 04-518-87.

[21] X. Výročí Sinclair ZX Spectrum. *FIFO : Microcomputer Magazín*. žáří 1992, ročník: 3, číslo: 17, s. 4-7.

[22] XILINX. *Spartan-3AN FPGA Family Data Sheet*. April 1, 2011.
URL http://www.xilinx.com/support/documentation/data_sheets/ds557.pdf

[23] Z88dk: z80 Development kit [online]. [cit. 2012-04-12].
URL http://www.z88dk.org

[24] ZX Spectrum 128 ROM Disassembly. [online]. [cit. 2012-03-17].
URL http://www.fruitcake.plus.com/Sinclair/Spectrum128/ROMDisassembly/Spectrum128ROMDisassembly.htm

[25] ZX Spectrum. In *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, [cit. 2011-11-24].
URL http://en.wikipedia.org/wiki/ZX_Spectrum

[26] ZX80. In *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, [cit. 2011-11-24].
URL http://en.wikipedia.org/wiki/ZX80

[27] ZX81. In *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, [cit. 2011-11-24].
URL http://en.wikipedia.org/wiki/ZX81

Příloha A

Překlad a zprovoznění emulátoru

A.1 Překlad hardwarové architektury pro FPGA

Zdrojové soubory emulátoru pro FPGA se nacházejí na CD v adresáři "fpga". V podsložce "FITkit"jsou zdrojové soubory, které jsem převzal z platformy FITkit. Jedná se o řadiče PS/2, VGA, SDRAM a správy hodinového signálu.

Protože pro překlad používám prostředí Xilinx ISE, umístil jsem na CD do složky "Xilinx"soubor Xilinx.xise. V tomto souboru jsou definovány cesty ke zdrojovým souborům a lze díky němu celý projekt otevřít v prostředí Xilinx ISE a přeložit jej.

A.2 Překlad systému ZX Simi OS

Operační systém ZX Simi OS jsem naprogramoval v jazyce C. Pro překlad jsem využil vývojové prostředí z88dk [23] ve verzi 1.9, které umožňuje překlad aplikací pro procesor Z80. Jeho součástí jsou i upravené standardní knihovny (stdlib.h, stdio.h) pro jednodušší tvorbu aplikací pro počítače ZX Spectrum. Díky nim lze například snadno přistupovat na obrazovku počítače pomocí instrukce printf apod.

Zdrojové soubory operačního systému jsou uloženy na CD v adresáři "zx_simi_os". Před překladem je nutné přidat cestu k překladači do systémové proměnné PATH. Aplikaci lze přeložit spuštěním scriptu make.bat. Výsledkem provedení scriptu je přeložená aplikace ZX Simi OS ve formátu TAP, tedy v obrazu magnetofonové pásky, pomocí kterého je možné aplikaci načíst a spustit.

A.3 Tvorba obsahu FLASH paměti

Na CD v adresáři "rom"se nacházejí soubory ROM počítače ZX Spectrum 48K a ZX Spectrum 128K. Dále se zde nacházejí i jejich modifikované verze s upravenou obsluhou rutiny LOAD z kapitoly 6.1. Tyto modifikované verze mají ve svém názvu slovo "_LOAD".

Na CD v adresáři "flash"se nachází utilita, která načte soubor ve formátu TAP a odstraní z něj nepotřebné hlavičky stejným způsobem, jako to dělá i operační systém ZX Simi OS během načítání TAP souboru a jeho ukládání do paměti. Modifikovaná obslužná rutina LOAD v BASICu totiž očekává právě takto předzpracovaná data.

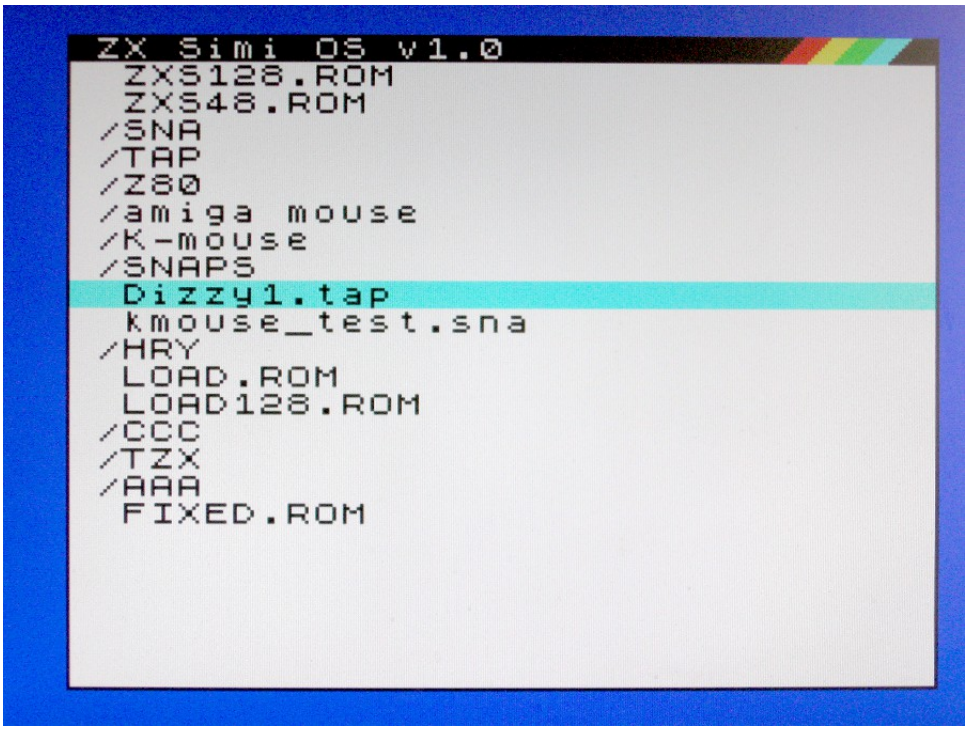
Ve stejném adresáři se pomocí scriptu "run.bat"vytvoří soubor "flash.bin", který se již následně může naprogramovat do FLASH paměti. Tento soubor vznikne spojením modi-

fikované ROM ZX Spectrum 128K a předzpracovaného TAP souboru operačního systému ZX Simi OS z minulého kroku.

Ve FLASH paměti tedy bude uložena ROM ZX Spectrum 128K a ihned za ní bude následovat operační systém ZX Simi OS tak, jak to očekává komponenta `FLASH_EXT` popsaná v kapitole 7.1.

Příloha B

Ovládání operačního systému ZX Simi OS



Obrázek B.1: Ukázka operačního systému ZX Simi OS

Na obrázku B.1 je ukázka ze spuštěného operačního systému. Pomocí kurzorových šipek nahoru a dolu se pohybuje v seznamu souborů a adresářů. Adresář je označen pomocí lomítka před názvem.

V případě, že je označen adresář, klávesou Enter se zobrazí jeho obsah. Z adresáře lze přejít o úroveň výš pomocí virtuálního adresáře .. (dvě tečky). Pokud je označen soubor TAP, TZX, ROM, SNA nebo Z80, klávesou Enter se tento soubor spustí.

Příloha C

Výpis obslužné rutiny LOAD z ROM ZX Spectrum 128K

```
;; LD-BYTES
L0556:  INC     D               ; reset the zero flag without disturbing carry.
        EX      AF,AF'          ; preserve entry flags.
        DEC     D               ; restore high byte of length.
        DI                      ; disable interrupts
        LD      A,$0F           ; make the border white and mic off.
        OUT     ($FE),A         ; output to port.
        LD      HL,L053F        ; Address: SA/LD-RET
        PUSH    HL              ; is saved on stack as terminating routine.
        IN      A,($FE)         ; read the ear state - bit 6.
        RRA                     ; rotate to bit 5.
        AND     $20             ; isolate this bit.
        OR      $02             ; combine with red border colour.
        LD      C,A             ; and store initial state long-term in C.
        CP      A               ; set the zero flag.

;; LD-BREAK
L056B:  RET     NZ              ; return if at any time space is pressed.

;; LD-START
L056C:  CALL    L05E7           ; routine LD-EDGE-1
        JR      NC,L056B        ; back to LD-BREAK with time out and no edge present on tape.
        LD      HL,$0415        ; set up 16-bit outer loop counter for approx 1 second delay.

;; LD-WAIT
L0574:  DJNZ    L0574           ; self loop to LD-WAIT (for 256 times)
        DEC     HL              ; decrease outer loop counter.
        LD      A,H             ; test for
        OR      L               ; zero.
        JR      NZ,L0574        ; back to LD-WAIT, if not zero, with zero in B.
        CALL    L05E3           ; routine LD-EDGE-2
        JR      NC,L056B        ; back to LD-BREAK if no edges at all.

;; LD-LEADER
L0580:  LD      B,$9C           ; set timing value.
        CALL    L05E3           ; routine LD-EDGE-2
        JR      NC,L056B        ; back to LD-BREAK if time-out
        LD      A,$C6           ; two edges must be spaced apart.
        CP      B               ; compare
        JR      NC,L056C        ; back to LD-START if too close together for a lead-in.
        INC     H               ; proceed to test 256 edged sample.
        JR      NZ,L0580        ; back to LD-LEADER while more to do.

;; LD-SYNC
L058F:  LD      B,$C9           ; initial timing value in B.
        CALL    L05E7           ; routine LD-EDGE-1
        JR      NC,L056B        ; back to LD-BREAK with time-out.
        LD      A,B             ; fetch augmented timing value from B.
        CP      $D4             ; compare
        JR      NC,L058F        ; back to LD-SYNC if gap too big, that is a normal lead-in edge gap.
        CALL    L05E7           ; routine LD-EDGE-1
        RET     NC              ; return with time-out.
        LD      A,C             ; fetch long-term mask from C
        XOR     $03             ; and make blue/yellow.
        LD      C,A             ; store the new long-term byte.
        LD      H,$00           ; set up parity byte as zero.
        LD      B,$B0           ; timing.
        JR      L05C8           ; forward to LD-MARKER
                                ; the loop mid entry point with the alternate
                                ; zero flag reset to indicate first byte is discarded.

;; LD-LOOP
L05A9:  EX      AF,AF'          ; restore entry flags and type in A.
        JR      NZ,L05B3        ; forward to LD-FLAG if awaiting initial flag which is to be discarded.
        JR      NC,L05BD        ; forward to LD-VERIFY if not to be loaded.
        LD      (IX+$00),L      ; place loaded byte at memory location.
        JR      L05C2           ; forward to LD-NEXT

;; LD-FLAG
L05B3:  RL      C               ; preserve carry (verify) flag in long-term state byte.Bit 7 can be lost
        XOR     L               ; compare type in A with first byte in L.
        RET     NZ              ; return if no match e.g. CODE vs DATA.
        LD      A,C             ; fetch byte with stored carry
        RRA                     ; rotate it to carry flag again
        LD      C,A             ; restore long-term port state.
        INC     DE              ; increment length ??
        JR      L05C4           ; forward to LD-DEC. but why not to location after ?
```

```
;; LD-VERIFY
L05BD:  LD      A,(IX+$00)    ; fetch byte from memory.
        XOR     L             ; compare with that on tape
        RET     NZ            ; return if not zero.

;; LD-NEXT
L05C2:  INC     IX            ; increment byte pointer.

;; LD-DEC
L05C4:  DEC     DE            ; decrement length.
        EX      AF,AF'        ; store the flags.
        LD      B,$B2         ; timing.

;; LD-MARKER
L05C8:  LD      L,$01         ; initialize as %00000001

;; LD-8-BITS
L05CA:  CALL    L05E3         ; routine LD-EDGE-2 increments B relative to gap between 2 edges.
        RET     NC            ; return with time-out.
        LD      A,$CB         ; the comparison byte.
        CP      B             ; compare to incremented value of B.
                              ; if B is higher then bit on tape was set.
                              ; if <= then bit on tape is reset.
        RL      L             ; rotate the carry bit into L.
        LD      B,$B0         ; reset the B timer byte.
        JP      NC,L05CA      ; JUMP back to LD-8-BITS
        LD      A,H           ; fetch the running parity byte.
        XOR     L             ; include the new byte.
        LD      H,A           ; and store back in parity register.
        LD      A,D           ; check length of
        OR      E             ; expected bytes.
        JR      NZ,L05A9      ; back to LD-LOOP while there are more.
        LD      A,H           ; fetch parity byte.
        CP      $01           ; set carry if zero.
        RET                   ; return in no carry then error as checksum disagrees.

;; LD-EDGE-2
L05E3:  CALL    L05E7         ; call routine LD-EDGE-1 below.
        RET     NC            ; return if space pressed or time-out.
                              ; else continue and look for another adjacent
                              ; edge which together represent a bit on the tape.

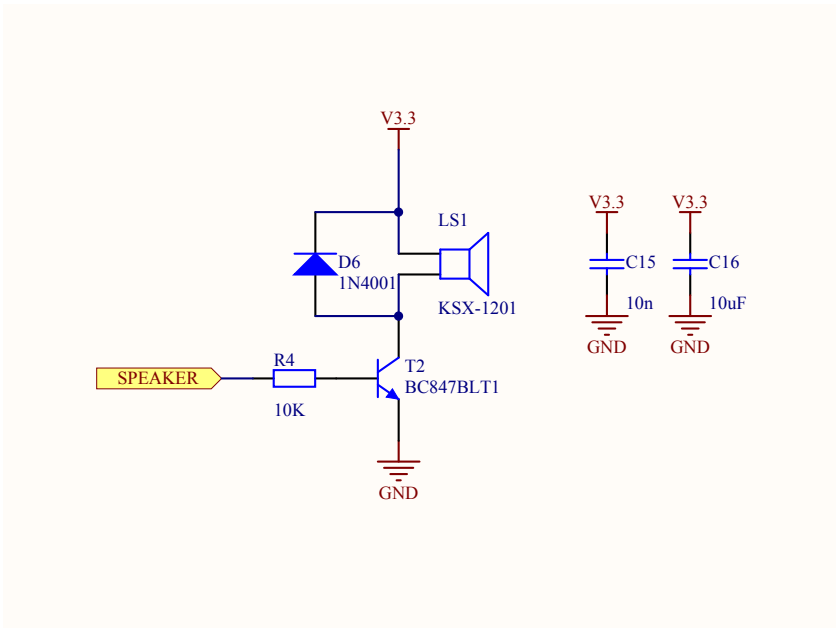
;; LD-EDGE-1
L05E7:  LD      A,$16         ; a delay value of twenty two.

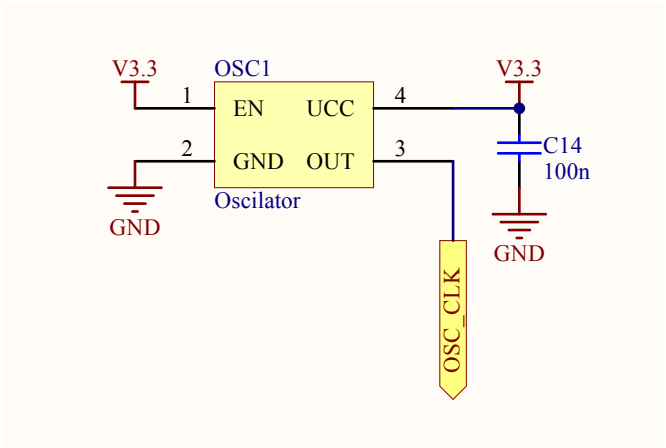
;; LD-DELAY
L05E9:  DEC     A             ; decrement counter
        JR      NZ,L05E9      ; loop back to LD-DELAY 22 times.
        AND      A            ; clear carry.

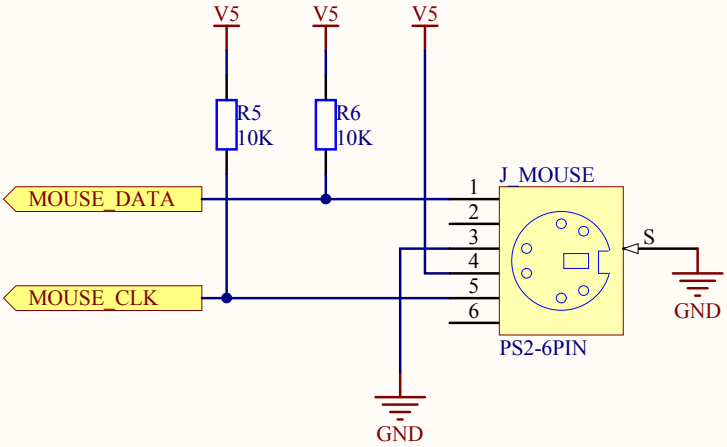
;; LD-SAMPLE
L05ED:  INC     B             ; increment the time-out counter.
        RET     Z             ; return with failure when $FF passed.
        LD      A,$7F         ; prepare to read keyboard and EAR port
        IN      A,($FE)       ; row $7FFE. bit 6 is EAR, bit 0 is SPACE key.
        RRA                   ; test outer key the space. (bit 6 moves to 5)
        RET     NC            ; return if space pressed.  >>>
        XOR     C             ; compare with initial long-term state.
        AND     $20           ; isolate bit 5
        JR      Z,L05ED       ; back to LD-SAMPLE if no edge.
        LD      A,C           ; fetch comparison value.
        CPL                   ; switch the bits
        LD      C,A           ; and put back in C for long-term.
        AND     $07           ; isolate new colour bits.
        OR      $08           ; set bit 3 - MIC off.
        OUT     ($FE),A       ; send to port to effect change of colour.
        SCF                   ; set carry flag signaling edge found within time allowed.
        RET                   ; return.
```

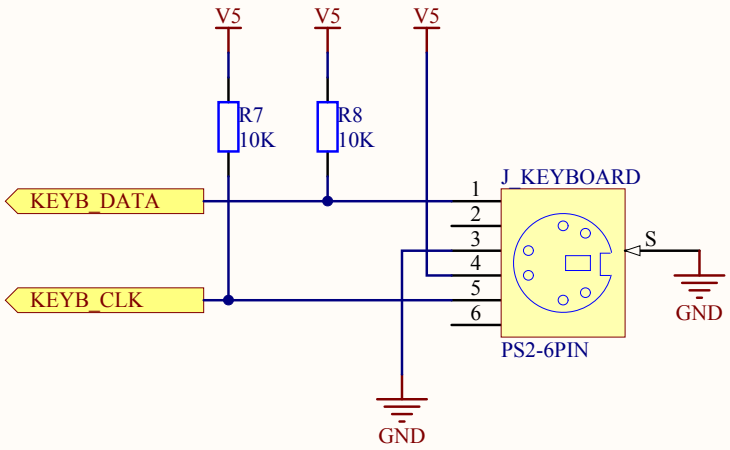
Příloha D

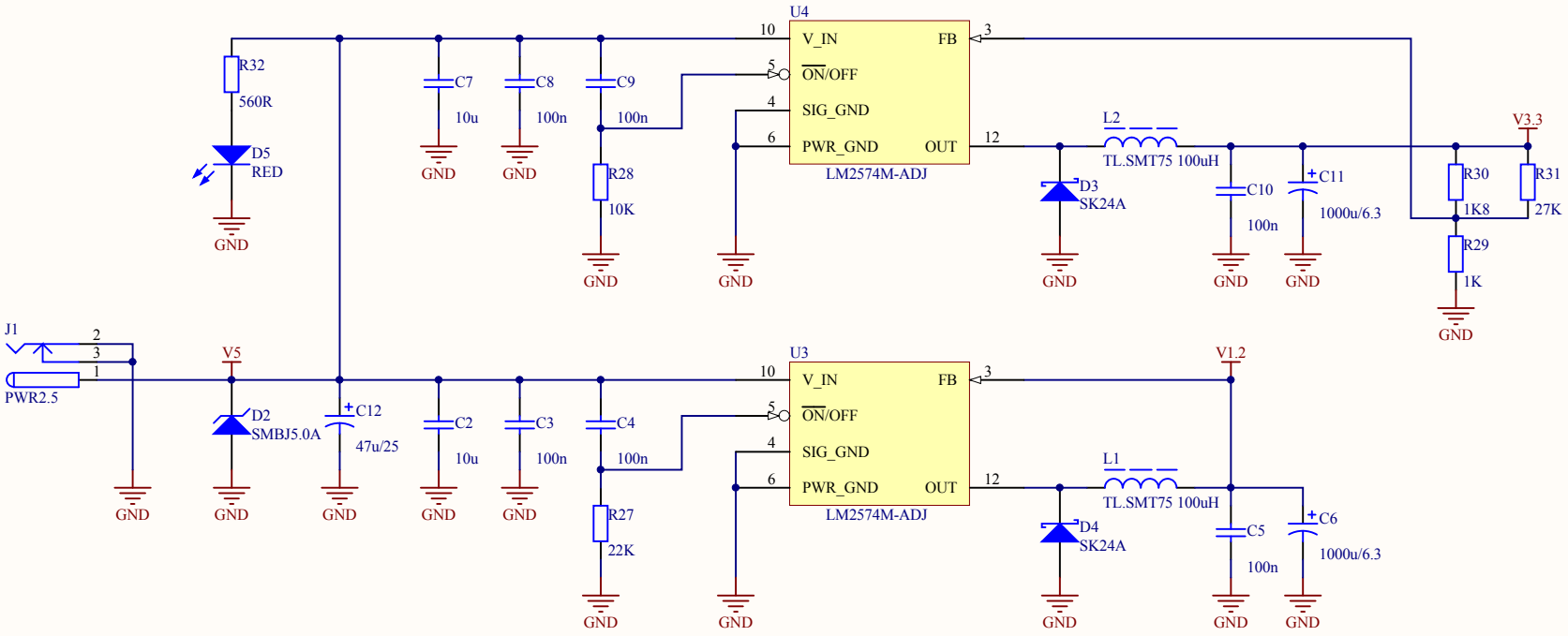
Schéma zapojení emulátoru

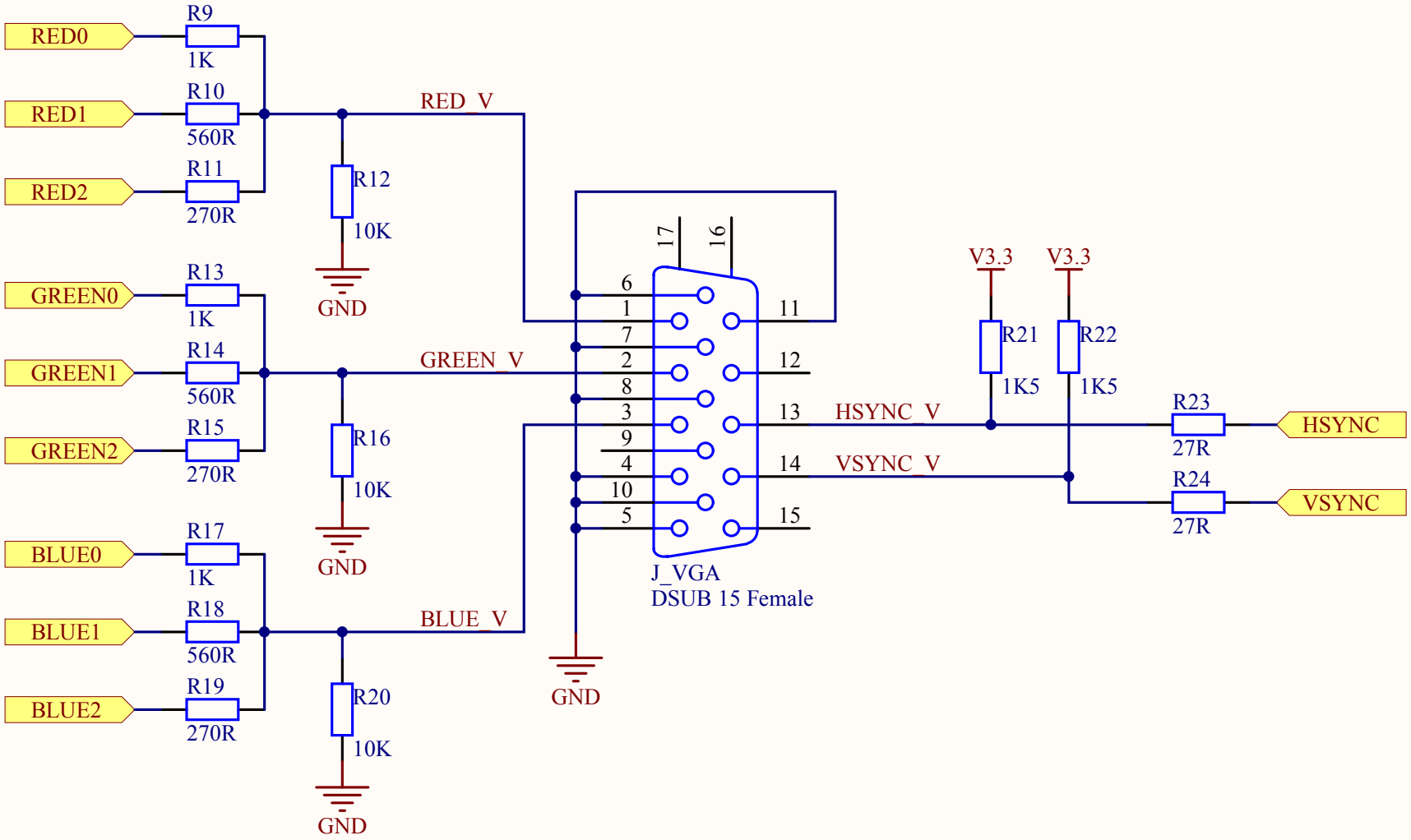


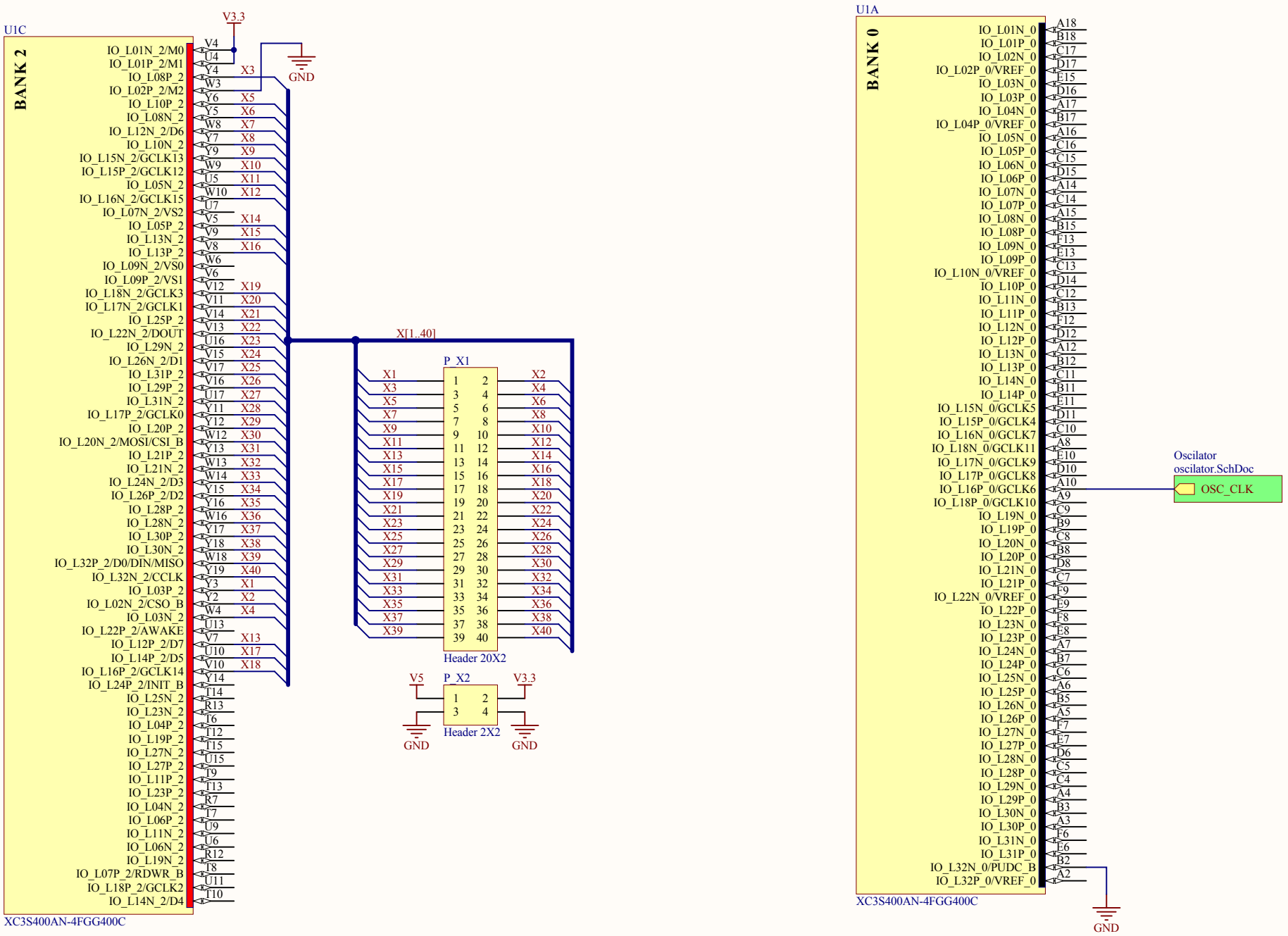


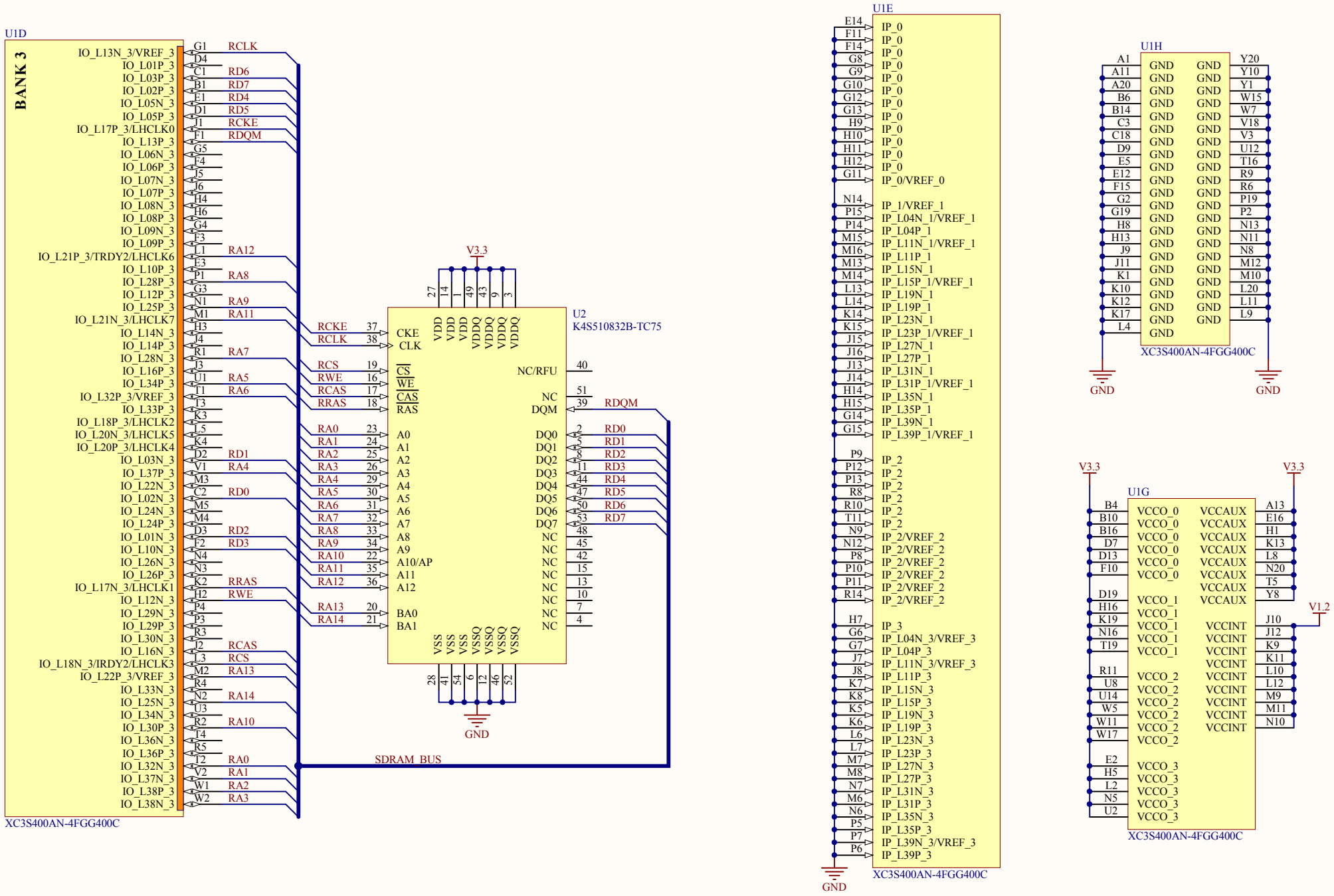

V5
V5
V5
R5
10K
R6
10K
MOUSE_DATA
MOUSE_CLK
J_MOUSE
1
2
3
4
5
6
S
GND
PS2-6PIN
GND

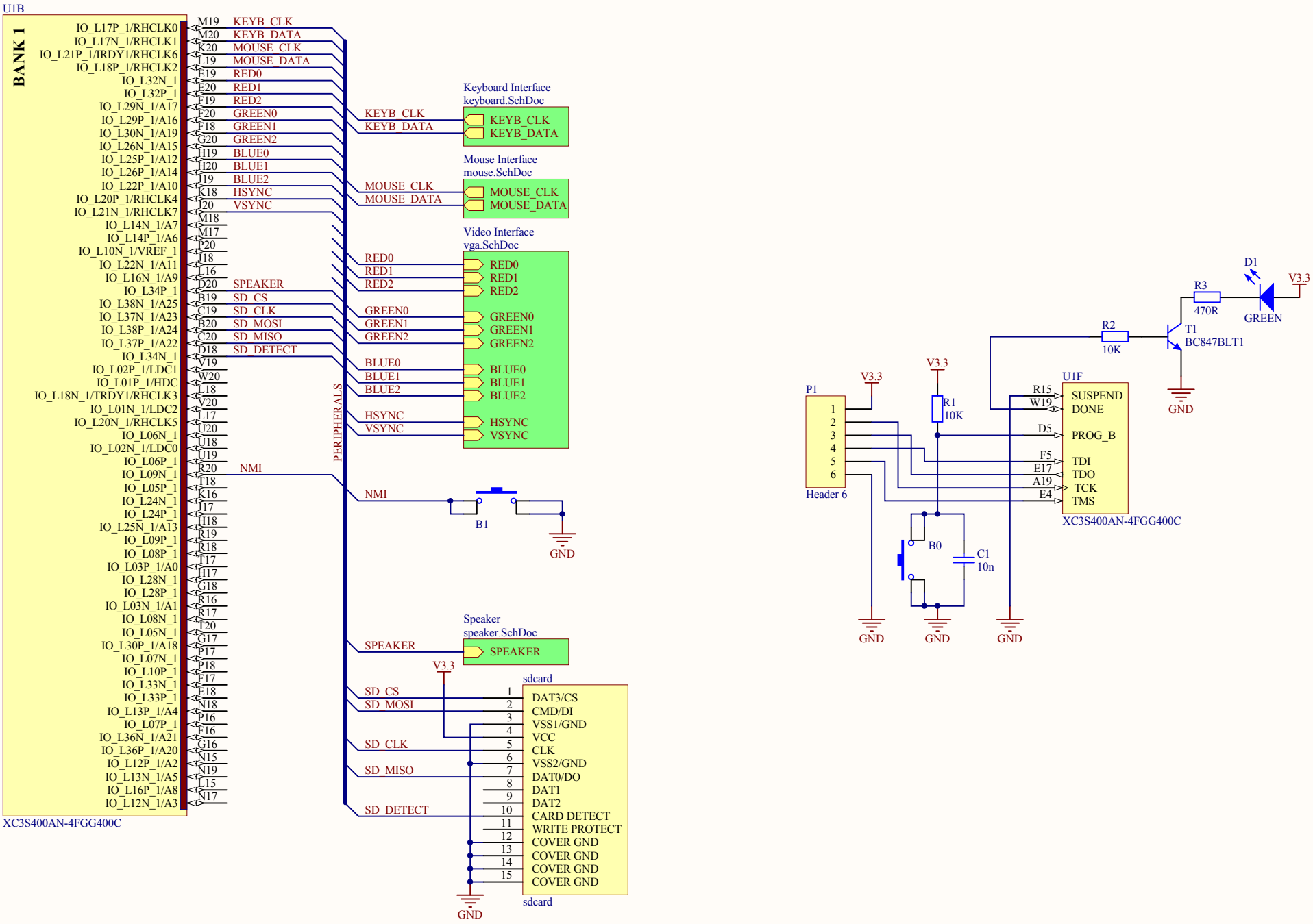

V5
V5
V5
R7
10K
R8
10K
KEYB_DATA
KEYB_CLK
J_KEYBOARD
1
2
3
4
5
6
S
GND
PS2-6PIN
GND


U4
V_IN
FB
ON/OFF
SIG_GND
PWR_GND
OUT
LM2574M-ADJ
U3
V_IN
FB
ON/OFF
SIG_GND
PWR_GND
OUT
LM2574M-ADJ
R32
560R
D5
RED
C7
10u
C8
100n
C9
100n
R28
10K
L2
TL.SMT75 100uH
D3
SK24A
C10
100n
C11
1000u/6.3
V3.3
R30
1K8
R31
27K
R29
1K
J1
PWR2.5
V5
D2
SMBJ5.0A
C12
47u/25
C2
10u
C3
100n
C4
100n
R27
22K
L1
TL.SMT75 100uH
D4
SK24A
C5
100n
C6
1000u/6.3
V1.2
GND


RED0
R9
1K
RED1
R10
560R
RED2
R11
270R
R12
10K
GND
RED_V
GREEN0
R13
1K
GREEN1
R14
560R
GREEN2
R15
270R
R16
10K
GND
GREEN_V
BLUE0
R17
1K
BLUE1
R18
560R
BLUE2
R19
270R
R20
10K
GND
BLUE_V
17
16
6
1
7
2
8
3
9
4
10
5
11
12
13
14
15
J_VGA
DSUB 15 Female
GND
V3.3
V3.3
R21
1K5
R22
1K5
HSYNC_V
VSYNC_V
R23
27R
R24
27R
HSYNC
VSYNC

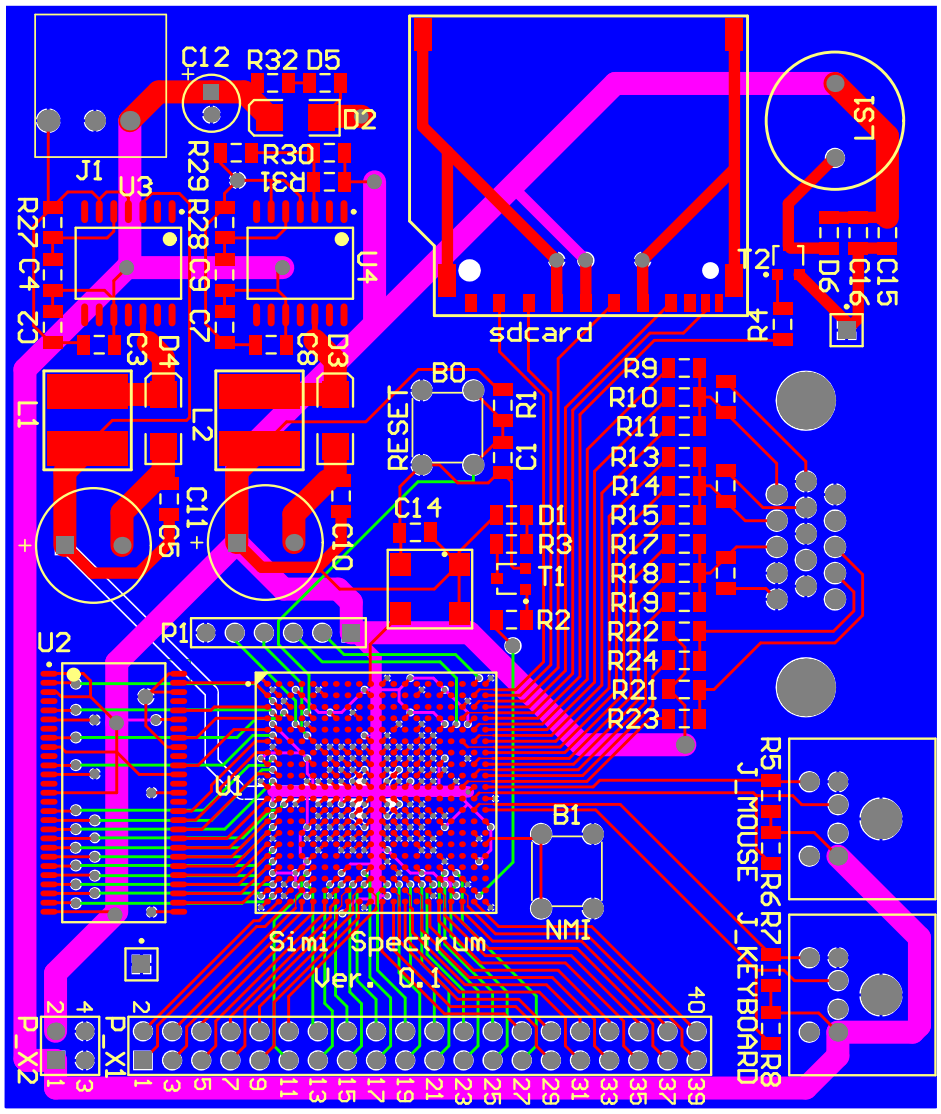

U1C
BANK 2
V3.3
GND
IO_L01N_2/M0 V4
IO_L01P_2/M1 U4
IO_L08P_2 Y4 X3
IO_L02P_2/M2 W3
IO_L10P_2 Y6 X5
IO_L08N_2 Y5 X6
IO_L12N_2/D6 W8 X7
IO_L10N_2 Y7 X8
IO_L15N_2/GCLK13 Y9 X9
IO_L15P_2/GCLK12 W9 X10
IO_L05N_2 U5 X11
IO_L16N_2/GCLK15 W10 X12
IO_L07N_2/VS2 U7
IO_L05P_2 V5 X14
IO_L13N_2 V9 X15
IO_L13P_2 V8 X16
IO_L09N_2/VS0 W6
IO_L09P_2/VS1 V6
IO_L18N_2/GCLK3 V12 X19
IO_L17N_2/GCLK1 V11 X20
IO_L25P_2 V14 X21
IO_L22N_2/DOUT V13 X22
IO_L29N_2 U16 X23
IO_L26N_2/D1 V15 X24
IO_L31P_2 V17 X25
IO_L29P_2 V16 X26
IO_L31N_2 U17 X27
IO_L17P_2/GCLK0 Y11 X28
IO_L20P_2 Y12 X29
IO_L20N_2/MOSI/CSI_B W12 X30
IO_L21P_2 Y13 X31
IO_L21N_2 W13 X32
IO_L24N_2/D3 W14 X33
IO_L26P_2/D2 Y15 X34
IO_L28P_2 Y16 X35
IO_L28N_2 W16 X36
IO_L30P_2 Y17 X37
IO_L30N_2 Y18 X38
IO_L32P_2/D0/DIN/MISO W18 X39
IO_L32N_2/CCLK Y19 X40
IO_L03P_2 Y3 X1
IO_L02N_2/CSO_B Y2 X2
IO_L03N_2 W4 X4
IO_L22P_2/AWAKE U13
IO_L12P_2/D7 V7 X13
IO_L14P_2/D5 U10 X17
IO_L16P_2/GCLK14 V10 X18
IO_L24P_2/INIT_B Y14
IO_L25N_2 T14
IO_L23N_2 R13
IO_L04P_2 T6
IO_L19P_2 T12
IO_L27N_2 T15
IO_L27P_2 U15
IO_L11P_2 T9
IO_L23P_2 T13
IO_L04N_2 R7
IO_L06P_2 T7
IO_L11N_2 U9
IO_L06N_2 U6
IO_L19N_2 R12
IO_L07P_2/RDWR_B T8
IO_L18P_2/GCLK2 U11
IO_L14N_2/D4 T10
XC3S400AN-4FGG400C
X[1..40]
P_X1
X1 1 2 X2
X3 3 4 X4
X5 5 6 X6
X7 7 8 X8
X9 9 10 X10
X11 11 12 X12
X13 13 14 X14
X15 15 16 X16
X17 17 18 X18
X19 19 20 X20
X21 21 22 X22
X23 23 24 X24
X25 25 26 X26
X27 27 28 X28
X29 29 30 X30
X31 31 32 X32
X33 33 34 X34
X35 35 36 X36
X37 37 38 X38
X39 39 40 X40
Header 20X2
V5
P_X2
V3.3
1 2
3 4
Header 2X2
GND
GND
U1A
BANK 0
IO_L01N_0 A18
IO_L01P_0 B18
IO_L02N_0 C17
IO_L02P_0/VREF_0 D17
IO_L03N_0 E15
IO_L03P_0 D16
IO_L04N_0 A17
IO_L04P_0/VREF_0 B17
IO_L05N_0 A16
IO_L05P_0 C16
IO_L06N_0 C15
IO_L06P_0 D15
IO_L07N_0 A14
IO_L07P_0 C14
IO_L08N_0 A15
IO_L08P_0 B15
IO_L09N_0 F13
IO_L09P_0 E13
IO_L10N_0/VREF_0 C13
IO_L10P_0 D14
IO_L11N_0 C12
IO_L11P_0 B13
IO_L12N_0 F12
IO_L12P_0 D12
IO_L13N_0 A12
IO_L13P_0 B12
IO_L14N_0 C11
IO_L14P_0 B11
IO_L15N_0/GCLK5 E11
IO_L15P_0/GCLK4 D11
IO_L16N_0/GCLK7 C10
IO_L18N_0/GCLK11 A8
IO_L17N_0/GCLK9 E10
IO_L17P_0/GCLK8 D10
IO_L16P_0/GCLK6 A10
IO_L18P_0/GCLK10 A9
IO_L19N_0 C9
IO_L19P_0 B9
IO_L20N_0 C8
IO_L20P_0 B8
IO_L21N_0 D8
IO_L21P_0 C7
IO_L22N_0/VREF_0 F9
IO_L22P_0 E9
IO_L23N_0 F8
IO_L23P_0 E8
IO_L24N_0 A7
IO_L24P_0 B7
IO_L25N_0 C6
IO_L25P_0 A6
IO_L26N_0 B5
IO_L26P_0 A5
IO_L27N_0 F7
IO_L27P_0 E7
IO_L28N_0 D6
IO_L28P_0 C5
IO_L29N_0 C4
IO_L29P_0 A4
IO_L30N_0 B3
IO_L30P_0 A3
IO_L31N_0 F6
IO_L31P_0 E6
IO_L32N_0/PUDC_B B2
IO_L32P_0/VREF_0 A2
XC3S400AN-4FGG400C
Oscilator
oscilator.SchDoc
OSC_CLK
GND


U1D
BANK 3
IO_L13N_3/VREF_3
IO_L01P_3
IO_L03P_3
IO_L02P_3
IO_L05N_3
IO_L05P_3
IO_L17P_3/LHCLK0
IO_L13P_3
IO_L06N_3
IO_L06P_3
IO_L07N_3
IO_L07P_3
IO_L08N_3
IO_L08P_3
IO_L09N_3
IO_L09P_3
IO_L21P_3/TRDY2/LHCLK6
IO_L10P_3
IO_L28P_3
IO_L12P_3
IO_L25P_3
IO_L21N_3/LHCLK7
IO_L14N_3
IO_L14P_3
IO_L28N_3
IO_L16P_3
IO_L34P_3
IO_L32P_3/VREF_3
IO_L33P_3
IO_L18P_3/LHCLK2
IO_L20N_3/LHCLK5
IO_L20P_3/LHCLK4
IO_L03N_3
IO_L37P_3
IO_L22N_3
IO_L02N_3
IO_L24N_3
IO_L24P_3
IO_L01N_3
IO_L10N_3
IO_L26N_3
IO_L26P_3
IO_L17N_3/LHCLK1
IO_L12N_3
IO_L29N_3
IO_L29P_3
IO_L30N_3
IO_L16N_3
IO_L18N_3/IRDY2/LHCLK3
IO_L22P_3/VREF_3
IO_L33N_3
IO_L25N_3
IO_L34N_3
IO_L30P_3
IO_L36N_3
IO_L36P_3
IO_L32N_3
IO_L37N_3
IO_L38P_3
IO_L38N_3
G1 RCLK
D4
C1 RD6
B1 RD7
E1 RD4
D1 RD5
J1 RCKE
F1 RDQM
G5
F4
J5
J6
H4
H6
G4
F3
L1 RA12
E3
P1 RA8
G3
N1 RA9
M1 RA11
H3
J4
R1 RA7
J3
U1 RA5
T1 RA6
T3
K3
L5
K4
D2 RD1
V1 RA4
M3
C2 RD0
M5
M4
D3 RD2
F2 RD3
N4
N3
K2 RRAS
T2 RWE
P4
P3
R3
J2 RCAS
L3 RCS
M2 RA13
R4
N2 RA14
U3
R2 RA10
T4
R5
T2 RA0
V2 RA1
W1 RA2
W2 RA3
XC3S400AN-4FGG400C
V3.3
27 14 1 49 43 9 3
VDD VDD VDD VDDQ VDDQ VDDQ VDDQ
RCKE 37 CKE
RCLK 38 CLK
RCS 19 CS
RWE 16 WE
RCAS 17 CAS
RRAS 18 RAS
RA0 23 A0
RA1 24 A1
RA2 25 A2
RA3 26 A3
RA4 29 A4
RA5 30 A5
RA6 31 A6
RA7 32 A7
RA8 33 A8
RA9 34 A9
RA10 22 A10/AP
RA11 35 A11
RA12 36 A12
RA13 20 BA0
RA14 21 BA1
NC/RFU 40
NC 51
DQM 39 RDQM
DQ0 2 RD0
DQ1 5 RD1
DQ2 8 RD2
DQ3 11 RD3
DQ4 44 RD4
DQ5 47 RD5
DQ6 50 RD6
DQ7 53 RD7
NC 48
NC 45
NC 42
NC 15
NC 13
NC 10
NC 7
NC 4
VSS VSS VSS VSSQ VSSQ VSSQ VSSQ
28 41 54 6 12 46 52
GND
U2
K4S510832B-TC75
SDRAM_BUS
U1E
E14 IP_0
F11 IP_0
F14 IP_0
G8 IP_0
G9 IP_0
G10 IP_0
G12 IP_0
G13 IP_0
H9 IP_0
H10 IP_0
H11 IP_0
H12 IP_0
G11 IP_0/VREF_0
N14 IP_1/VREF_1
P15 IP_L04N_1/VREF_1
P14 IP_L04P_1
M15 IP_L11N_1/VREF_1
M16 IP_L11P_1
M13 IP_L15N_1
M14 IP_L15P_1/VREF_1
L13 IP_L19N_1
L14 IP_L19P_1
K14 IP_L23N_1
K15 IP_L23P_1/VREF_1
J15 IP_L27N_1
J16 IP_L27P_1
J13 IP_L31N_1
J14 IP_L31P_1/VREF_1
H14 IP_L35N_1
H15 IP_L35P_1
G14 IP_L39N_1
G15 IP_L39P_1/VREF_1
P9 IP_2
P12 IP_2
P13 IP_2
R8 IP_2
R10 IP_2
T11 IP_2
N9 IP_2/VREF_2
N12 IP_2/VREF_2
P8 IP_2/VREF_2
P10 IP_2/VREF_2
P11 IP_2/VREF_2
R14 IP_2/VREF_2
H7 IP_3
G6 IP_L04N_3/VREF_3
G7 IP_L04P_3
J7 IP_L11N_3/VREF_3
J8 IP_L11P_3
K7 IP_L15N_3
K8 IP_L15P_3
K5 IP_L19N_3
K6 IP_L19P_3
L6 IP_L23N_3
L7 IP_L23P_3
M7 IP_L27N_3
M8 IP_L27P_3
N7 IP_L31N_3
M6 IP_L31P_3
N6 IP_L35N_3
P5 IP_L35P_3
P7 IP_L39N_3/VREF_3
P6 IP_L39P_3
GND
XC3S400AN-4FGG400C
U1H
A1 GND GND Y20
A11 GND GND Y10
A20 GND GND Y1
B6 GND GND W15
B14 GND GND W7
C3 GND GND V18
C18 GND GND V3
D9 GND GND U12
E5 GND GND T16
E12 GND GND R9
F15 GND GND R6
G2 GND GND P19
G19 GND GND P2
H8 GND GND N13
H13 GND GND N11
J9 GND GND N8
J11 GND GND M12
K1 GND GND M10
K10 GND GND L20
K12 GND GND L11
K17 GND GND L9
L4 GND
XC3S400AN-4FGG400C
GND
GND
V3.3
V3.3
U1G
B4 VCCO_0 VCCAUX A13
B10 VCCO_0 VCCAUX E16
B16 VCCO_0 VCCAUX H1
D7 VCCO_0 VCCAUX K13
D13 VCCO_0 VCCAUX L8
F10 VCCO_0 VCCAUX N20
VCCAUX T5
D19 VCCO_1 VCCAUX Y8
H16 VCCO_1
K19 VCCO_1
N16 VCCO_1 VCCINT J10
T19 VCCO_1 VCCINT J12
VCCINT K9
R11 VCCO_2 VCCINT K11
U8 VCCO_2 VCCINT L10
U14 VCCO_2 VCCINT L12
W5 VCCO_2 VCCINT M9
W11 VCCO_2 VCCINT M11
W17 VCCO_2 VCCINT N10
E2 VCCO_3
H5 VCCO_3
L2 VCCO_3
N5 VCCO_3
U2 VCCO_3
V1.2
XC3S400AN-4FGG400C


U1B
BANK 1
IO_L17P_1/RHCLK0
IO_L17N_1/RHCLK1
IO_L21P_1/IRDY1/RHCLK6
IO_L18P_1/RHCLK2
IO_L32N_1
IO_L32P_1
IO_L29N_1/A17
IO_L29P_1/A16
IO_L30N_1/A19
IO_L26N_1/A15
IO_L25P_1/A12
IO_L26P_1/A14
IO_L22P_1/A10
IO_L20P_1/RHCLK4
IO_L21N_1/RHCLK7
IO_L14N_1/A7
IO_L14P_1/A6
IO_L10N_1/VREF_1
IO_L22N_1/A11
IO_L16N_1/A9
IO_L34P_1
IO_L38N_1/A25
IO_L37N_1/A23
IO_L38P_1/A24
IO_L37P_1/A22
IO_L34N_1
IO_L02P_1/LDC1
IO_L01P_1/HDC
IO_L18N_1/TRDY1/RHCLK3
IO_L01N_1/LDC2
IO_L20N_1/RHCLK5
IO_L06N_1
IO_L02N_1/LDC0
IO_L06P_1
IO_L09N_1
IO_L05P_1
IO_L24N_1
IO_L24P_1
IO_L25N_1/A13
IO_L09P_1
IO_L08P_1
IO_L03P_1/A0
IO_L28N_1
IO_L28P_1
IO_L03N_1/A1
IO_L08N_1
IO_L05N_1
IO_L30P_1/A18
IO_L07N_1
IO_L10P_1
IO_L33N_1
IO_L33P_1
IO_L13P_1/A4
IO_L07P_1
IO_L36N_1/A21
IO_L36P_1/A20
IO_L12P_1/A2
IO_L13N_1/A5
IO_L16P_1/A8
IO_L12N_1/A3
XC3S400AN-4FGG400C
M19 KEYB_CLK
M20 KEYB_DATA
K20 MOUSE_CLK
L19 MOUSE_DATA
E19 RED0
E20 RED1
F19 RED2
F20 GREEN0
F18 GREEN1
G20 GREEN2
H19 BLUE0
H20 BLUE1
J19 BLUE2
K18 HSYNC
J20 VSYNC
D20 SPEAKER
B19 SD_CS
C19 SD_CLK
B20 SD_MOSI
C20 SD_MISO
D18 SD_DETECT
R20 NMI
PERIPHERALS
KEYB_CLK
KEYB_DATA
MOUSE_CLK
MOUSE_DATA
RED0
RED1
RED2
GREEN0
GREEN1
GREEN2
BLUE0
BLUE1
BLUE2
HSYNC
VSYNC
NMI
SPEAKER
SD_CS
SD_MOSI
SD_CLK
SD_MISO
SD_DETECT
Keyboard Interface
keyboard.SchDoc
KEYB_CLK
KEYB_DATA
Mouse Interface
mouse.SchDoc
MOUSE_CLK
MOUSE_DATA
Video Interface
vga.SchDoc
RED0
RED1
RED2
GREEN0
GREEN1
GREEN2
BLUE0
BLUE1
BLUE2
HSYNC
VSYNC
B1
GND
Speaker
speaker.SchDoc
SPEAKER
V3.3
sdcard
1 DAT3/CS
2 CMD/DI
3 VSS1/GND
4 VCC
5 CLK
6 VSS2/GND
7 DAT0/DO
8 DAT1
9 DAT2
10 CARD DETECT
11 WRITE PROTECT
12 COVER GND
13 COVER GND
14 COVER GND
15 COVER GND
sdcard
GND
P1
1
2
3
4
5
6
Header 6
V3.3
R1
10K
B0
C1
10n
GND
GND
GND
U1F
R15 W19 SUSPEND DONE
D5 PROG_B
F5 TDI
E17 TDO
A19 TCK
E4 TMS
XC3S400AN-4FGG400C
R2
10K
T1
BC847BLT1
R3
470R
D1
GREEN
V3.3
GND

Příloha E

Plošný spoj


C12
R32
D5
D2
R30
R31
J1
U3
R29
R28
C9
C2
U4
R27
C4
C3
D4
C8
D3
L1
L2
C11
C5
C10
C14
RESET
B0
C1
R1
D1
R3
T1
R2
P1
U2
U1
B1
NMI
Simi Spectrum
Ver. 0.1
sdcard
R9
R10
R11
R13
R14
R15
R17
R18
R19
R22
R24
R21
R23
R5
R6
R7
R8
J_MOUSE
J_KEYBOARD
P_X1
P_X2
LS1
T2
D6
C15
C16
R4