VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

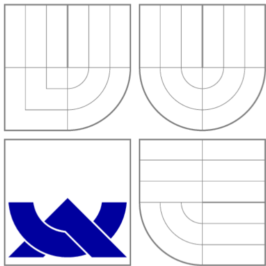
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ELEKTRONICKÁ ŠACHOVNICE NA FITKITU

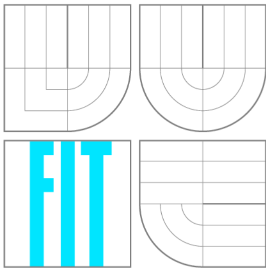
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAKUB KUBÍN
AUTHOR

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

ELEKTRONICKÁ ŠACHOVNICE NA FITKITU

ELECTRONIC CHESSBOARD ON THE FITKIT PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAKUB KUBÍN
AUTHOR

VEDOUCÍ PRÁCE Ing. JAN KAŠTIL
SUPERVISOR

BRNO 2012

Abstrakt

Tato bakalářská práce se zabývá analýzou, návrhem a implementací hry šachy na platformě FITkit. K platformě je připojen VGA monitor, na kterém je zobrazena šachovnice s figurami. Hra je ovládána pomocí klávesnice na FITkitu. Práce popisuje realizaci jednotky pro zobrazení šachovnice, implementovanou v programovatelném hradlovém poli. Software v mikrokontroléru řídí zobrazovací jednotku, generuje možné tahy a kontroluje tahy figur. Součástí kontrol je i zda král nemá šach a zdali hra neskončila matem nebo patem.

Abstract

This thesis deals with the analysis, design and implementation of chess on FITkit platform. The platform is connected to the VGA monitor, on which is shown the chessboard with the figures. The game is controlled by using the keyboard on FITkit platform. The work describes the realisation of the unit for the display of the checkerboard that is implemented in the programmable gate field. Software of the microcontroller controls the depictive unit, generates possible moves and checks strokes of figures. There is a control whether the King does not have the check and if the game is not over because of checkmate or stalemate.

Klíčová slova

FITkit, MCU, mikrokontrolér, FPGA, šachy, BRAM, bloková paměť

Keywords

FITkit, MCU, microcontroller, FPGA, chess, BRAM, block ram

Citace

Jakub Kubín: Elektronická šachovnice na FITKitu, bakalářská práce, Brno, FIT VUT v Brně, 2012

Elektronická šachovnice na FITKitu

Prohlášení

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

....
Jakub Kubín
14. května 2012

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu Ing. Janu Kaštilovi za vedení a poskytnuté rady při zpracování této práce.

© Jakub Kubín, 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 3

2 Teoretický úvod 4
2.1 FITkit . . . 4
2.1.1 Úvodní informace . . . 4
2.1.2 Platforma FITkit . . . 5
2.1.3 MCU . . . 7
2.1.4 FPGA . . . 8
2.1.5 SPI . . . 8
2.1.6 SPI na FITkitu . . . 9
2.1.7 LCD displej . . . 9
2.1.8 Klávesnice . . . 9
2.1.9 VGA rozhraní . . . 9
2.1.10 Řadič SDRAM . . . 10
2.2 Šachy . . . 10
2.2.1 Dějiny šachové hry . . . 11
2.2.2 Šach, Mat, Pat . . . 12
2.2.3 Pohyb figur . . . 12
2.3 Jazyk VHDL . . . 13
2.3.1 Základní prvky jazyka . . . 14
2.3.2 Behaviorální popis . . . 14
2.3.3 Strukturní popis . . . 15
2.3.4 Data flow popis . . . 16

3 Návrh řešení 17
3.1 Obecný úvod . . . 17
3.2 Struktura projektu . . . 17
3.3 Návrh FPGA . . . 18
3.3.1 Grafický návrh . . . 18
3.3.2 Zobrazovací jednotka . . . 18
3.3.3 Výběr paměti . . . 20
3.3.4 Video RAM . . . 21
3.3.5 Texture RAM . . . 21
3.4 Návrh MCU . . . 22
3.4.1 Úkoly MCU . . . 23
3.4.2 Datové struktury . . . 23
3.4.3 Ovládání hry . . . 23
3.4.4 Uložení hry . . . 23

4 Implementace **25**
4.1 Implementace FPGA . . . 25
4.1.1 Výběr top-level entity . . . 25
4.1.2 Použité komponenty . . . 25
4.1.3 SPI . . . 25
4.1.4 VRAM . . . 26
4.1.5 TEXRAM . . . 26
4.1.6 Proces `comp_vram_addr` . . . 27
4.1.7 Proces `comp_texture_addr` . . . 27
4.1.8 Proces `sig_delay` . . . 28
4.1.9 Popis práce zobrazovací jednotky . . . 28
4.2 Implementace MCU . . . 30
4.2.1 Terminál . . . 30
4.2.2 Ovládání hry . . . 30
4.2.3 Jádro hry . . . 30
4.2.4 Kontrola tahu . . . 31
4.2.5 Generátor možných tahů . . . 31
4.2.6 Kontrola šach . . . 32
4.2.7 Kontrola mat, pat . . . 32
4.2.8 VRAM . . . 32
4.2.9 Využití Flash paměťi . . . 33
4.2.10 TEXRAM . . . 33
4.2.11 Uložení hry . . . 33

5 Závěr **35**

A Blokový diagram FITkitu **38**

B Obsah CD **39**

Kapitola 1

Úvod

Platforma FITkit, pro kterou byla vytvořena aplikace v této práci, byla vyvinuta na Fakultě informačních technologií Vysokého učení technického v Brně. Jedná se o hardwarové zařízení umožňující studentům prakticky si ověřit znalosti z technicky zaměřených předmětů ve školních i vlastních projektech. Aplikace vytvořené pro platformu můžeme považovat za vestavěné systémy. Přesto, že o tom většina lidí dnes nemá ani tušení, jsou vestavěné systémy nedílnou součástí našich každodenních životů. Zařízení, jako například mobilní telefon nebo MP3 přehrávač, jsou typickými představiteli vestavěných systémů.

Šachy jsou klasickou deskovou logickou hrou s možností uplatnění důmyslných strategií a logického myšlení. Hra tak přispívá k rozvoji analytického myšlení, představivosti či paměti. Šachy nejsou ovlivněny prvkem náhody, v partii rozhodují pouze schopnosti a znalosti hráčů.

Cílem této bakalářské práce je vytvořit hru šachy pro platformu FITkit s generátorem možných tahů. Zadání jsem rozšířil o kontrolu šachu a konce hry. Vytvořený program navíc také umožňuje uložení hry a její pozdější opětovné načtení. Pro běh aplikace není vyžadováno PC, hru si je možné zahrát pouze na platformě s připojeným zobrazovacím zařízením.

Důvodem, proč jsem si vybral toto téma, byl můj pozitivní vztah ke hře šachy a dobré zkušenosti při vytváření projektů pro platformu FITkit. Také jsem chtěl více pochopit problematiku návrhu a implementace hardwaru pomocí jazyka VHDL.

Práce je rozdělena do pěti kapitol. Kapitola 2 blíže seznamuje s platformou FITkit. Popisuje možnosti platformy a významné periferie pro tuto práci. Popsán je LCD displej, klávesnice, VGA rozhraní a paměť SDRAM a jejich možnost napojení na komunikační rozhraní SPI. Dále pak kapitola rozebírá hru šachy a jazyk VHDL. V kapitole 3 je rozebrán návrh řešení zobrazovací jednotky v FPGA a kódu pro MCU. Kapitola 4 popisuje implementaci hardwaru a softwaru podle předchozí kapitoly. Závěrečná 5. kapitola obsahuje zhodnocení dosažených cílů a možností dalšího vylepšení a rozšíření projektu.

Kapitola 2

Teoretický úvod

Tato kapitola se věnuje podrobnějšímu popisu platformy FITkit a hře šachy, která na ní má být implementována. Kapitola je rozdělena na tři části: první se věnuje platformě FITkit, druhá je věnována hře šachy. V poslední části jsou diskutovány základní vlastnosti jazyka VHDL pro popis hardwaru v FPGA.

2.1 FITkit

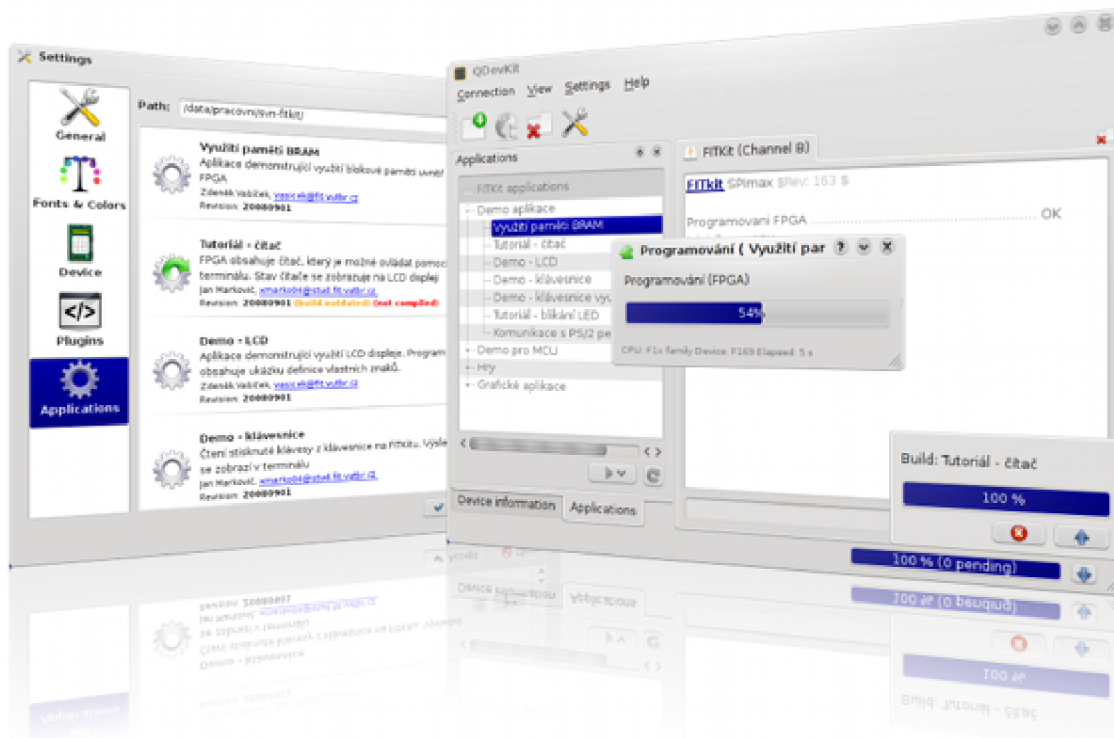
2.1.1 Úvodní informace

Platforma FITkit (obr. 2.1) byla vyvinuta na Fakultě informačních technologií VUT v Brně. Cílem tohoto projektu je umožnit studentům prakticky si ověřit teoretické znalosti z technicky zaměřených předmětů. Je možné navrhovat a realizovat softwarové, ale i hardwarové projekty nebo celé aplikace.

V dnešní době se často používají tzv. vestavěné systémy (anglicky Embedded Systems). V budoucnu se dá očekávat perspektiva a čím dál větší uplatnění této oblasti informatiky. Typicky jsou vestavěné systémy používány například v automobilovém průmyslu. Vestavěný systém je zařízení, které má v sobě nějakým způsobem vestavěn počítač (mobilní telefon, MP3 přehrávač, bankomat, autopilot, ABS v automobilech, klimatizace, kalkulačka atd.). Typické vestavěné systémy se skládají z procesorů, specializovaného hardwaru (např. MP3 kodér/dekodér) a aplikačního software.

Obrázek 2.1: Platforma FITkit [19]

Pro FITkit je vytvořena řada aplikací. Například generátor napětí, hodiny reálného času, videopřehrávač nebo hra tetris. Aplikace jsou volně přístupné (pod licencí open-source) na stránkách FITkitu [15]. K naprogramování a komunikaci s FITkitem slouží multiplatformní skriptovatelný terminál QDevKit (obr. 2.2). Ten umožňuje tzv. vzdálený překlad, díky kterému nemusíme složitě instalovat programy pro překlad projektu.



Obrázek 2.2: Program QDevKit [18]

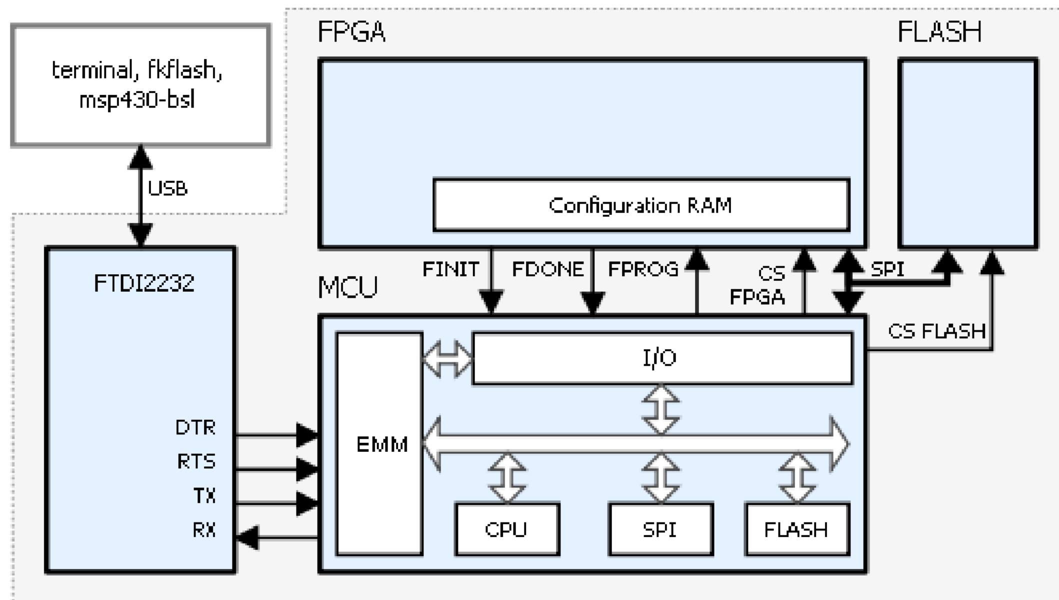
2.1.2 Platforma FITkit

> FITkit je samostatný hardware, který obsahuje výkonný mikrokontrolér s nízkým příkonem, hradlové pole FPGA (anglicky Field Programmable Gate Array) a řadu periferií. Důležitým aspektem je využití pokročilého reprogramovatelného hardwaru na bázi hradlových polí FPGA jenž lze, podobně jako software na počítači, neomezeně modifikovat pro různé účely dle potřeby – uživatel tedy nemusí vytvářet nový hardware pro každou aplikaci znovu.[1]

Pro komunikaci mezi komponentami v FPGA a MCU je využita sběrnice SPI (Synchronous Peripheral Interface). Jednoduché blokové schéma FITkitu je zobrazeno na obr. 2.3, komplexní blokový diagram zobrazující propojení MCU, FPGA, SPI a periferií je na obr. A.1.

V současné době je pro studenty k dispozici starší verze FITkitu (FITkit 1.x) nebo novější vylepšený FITkit 2.0. Obě verze jsou vybaveny MCU rodiny MSP430 firmy Texas Instruments, FPGA řady Spartan 3 firmy Xilinx a nasledujícími periferiemi:

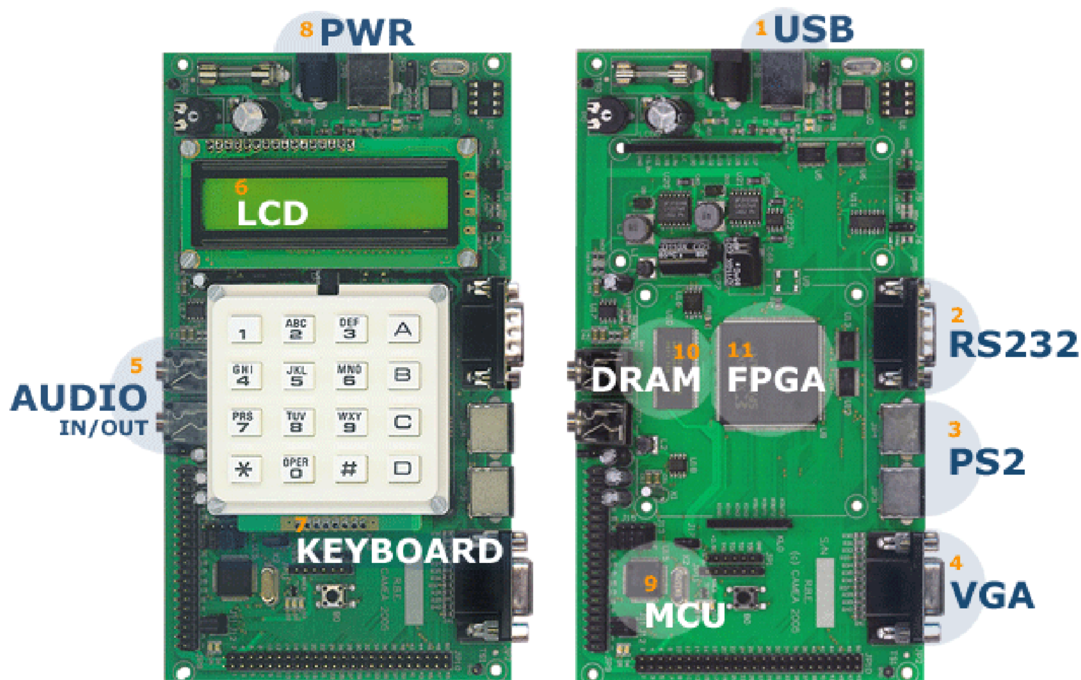
[1] Převzato z [19]



Obrázek 2.3: Blokové schéma FITkitu [17]

- USB převodníkem FT2232C
- Řádkovým LCD displejem
- Klávesnicí
- DRAM 8x8Mbit
- rozhraním VGA
- konektorem RS232
- konektory PS2
- rozšiřujícími konektory
- audio rozhraním

Nová verze kitu (FITkit 2.0) oproti svému předchůdci obsahuje řadu vylepšení a novinek. Mezi ně patří výkonnější model mikrokontroleru, který má více FLASH, ale hlavně RAM paměti. Nový mikrokontroler obsahuje vylepšené seriové rozhraní, které umí komunikovat na vyšších rychlostech a podporuje nezávislý běh SPI/I2C a UART na jednom kanálu. S tím však souvisí zpětná nekompatibilita na úrovni zdrojových kódů, kterou řeší transparentně knihovna libfitkit. Přibyl výkonný konfigurovatelný audio kodek TLV320AIC23B vybavený A/D a D/A stereo převodníky, propojka J5 pro povolení přerušení z FPGA a piezo reproduktor, který umožňuje generovat zvuk. FTDI obvod je nově osazen EEPROM pamětí, která kitu umožňuje identifikovat se pomocí vlastního seriového čísla.



Obrázek 2.4: Platforma FITkit s popisem periferií [16]

2.1.3 MCU

MCU (anglicky Micro-Controller Unit) na kitu je 16-bitový nízkopříkonový mikrokontrolér rodiny MSP430 firmy Texas Instruments. FITkit verze 1.x obsahuje mikrokontroler MSP430F168IPM, FITkit verze 2.0 výkonnější variantu MSP430F2617. Více informací lze nalézt v [13], resp. v [14]. Čipy mají následující vlastnosti:

- frekvence 8 MHz (resp. 16 MHz v případě FITkitu 2.0)
- nízké napájecí napětí (1,8 V – 3,6 V)
- nízký příkon (330 uA v aktivním režimu při 1 MHz a 2,2 V, 1,1 uA ve stand-by režimu, 0,2 uA v režimu vypnuto)
- 16-bitová RISC architektura, instrukční cyklus 125 ns
- paměť pro program typu FLASH 48 kB, paměť pro data typu RAM 2 kB (resp. 92 kB FLASH a 8 kB RAM v případě FITkitu 2.0)
- Integrované moduly:
  - 3-kanálové DMA (anglicky Direct Memory Access) umožňuje přímý přístup do paměti bez využití CPU
  - 12-bitové A/D a D/A převodníky
  - 2x 16-bitové časovače
  - sériové komunikační rozhraní USART0 (asynchronní UART, synchronní SPI, I2C), USART1 (asynchronní UART, synchronní SPI)
  - komparátor

2.1.4 FPGA

Všechny programovatelné součástky se souhrnně označují PLD (Programmable Logic Device). Na rozdíl od logických hradel, binární sčítačky, multiplexoru a jiných integrovaných obvodů mají programovatelné logické obvody tu výhodu, že nemají danou funkčnost již z výroby. Uživatel si tak může naprogramovat svůj vlastní logický obvod s požadovanou funkcionalitou. Mezi PLD můžeme zařadit i obvody FPGA, ty mají z této kategorie nejobecnější strukturu a obsahují nejvíce logiky. Více informací lze nalézt na [9]. Obě verze FITkitu jsou osazeny programovatelným hradlovým polem XC3S50-4PQ208C řady Spartan 3 firmy Xilinx. FPGA čip má tyto parametry:

- 124 vstupně/výstupních obvodů (I/O Block - IOB), možnost využít až 124 (I/O) pinů FPGA
- 192 konfigurovatelných logických bloků (CLBs - Configurable Logic Blocks) uspořádaných do matice o 16 řádcích a 12 sloupcích, 1728 logických buněk, 50000 logických hradel
- 4 blokové dvouportové paměti BRAM, každá s kapacitou 2 kB
- 12 kb distribuované paměťi RAM
- 4 násobičky 18x18 bitů
- 2 jednotky pro správu hodin (DCM - Digital Clock Manager)
- bloky DLL (Delay-Locked Loop), které slouží pro rekonstrukci a případné násobení či dělení vnějších taktovacích signálů

Více podrobností lze nalézt v [23]. Čip používá konfiguraci pomocí statické paměti RAM. Z toho vyplývá, že po připojení napájení je vždy nutné znovu nahrát konfiguraci. Tento problém je na FITkitu vyřešen použitím paměti Flash, do které je uložena konfigurace FPGA. Konfigurace je při spuštění aplikace nahrána mikrokontrolérem do FPGA.

2.1.5 SPI

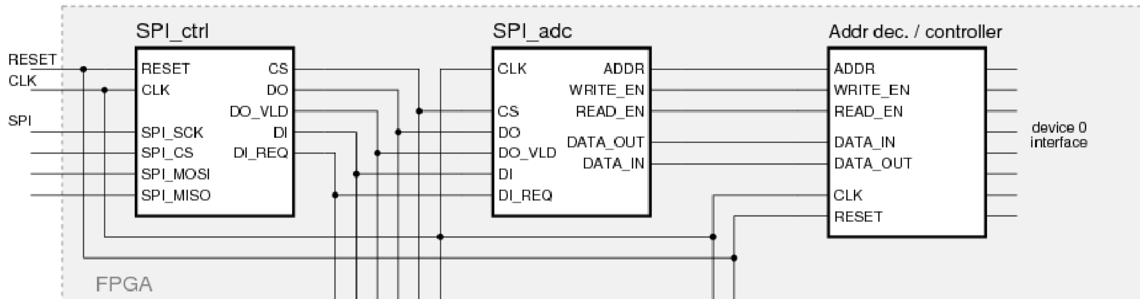
Jedná se o sériové vysokorychlostní rozhraní podporující plně duplexní obousměrnou komunikaci. To znamená, že zařízení připojené ke sběrnici může současně příjímat i vysílat data. Rozhraní SPI na FITkitu propojuje MCU, FPGA a paměť Flash. Komunikace mezi jednotkami probíhá na principu master-slave, tzn. že master (pán – nadřazené) zařízení řídí komunikaci na sběrnici a rozhoduje, s kterým ze slave (otrok – podřízených) zařízení bude komunikovat. Na FITkitu je tímto řídícím zařízením MCU. Komunikace na sběrnici probíhá pomoci následujících signálů:

- CS (Chip Select) - signál určuje platnost dat na sběrnici
- SCK (Clock) - hodinový signál sloužící k vzorkování dat
- MOSI (Master Out, Slave In) - přenos dat z master zařízení do slave zařízení
- MISO (Master In, Slave Out) - přenos dat z slave zařízení do master zařízení

Signály CS a SCK generuje master zařízení.

2.1.6 SPI na FITkitu

SPI na FITkitu zprostředkovává MCU komunikaci s Flash pamětí a periferiemi realizovanými v FPGA. Typicky přes SPI komunikujeme s řadičem LCD displeje, klávesnice, VGA rozhraní či SDRAM paměti. Signál SPI_CS umožňuje zvolit komunikaci s pamětí FLASH, signál SPI_FPGA_CS s komponentami uvnitř FPGA. SPI řadič zajišťuje převod protokolu SPI na interní sériovou sběrnici v FPGA. SPI dekodér propojuje SPI řadič s řadičem komponenty. SPI řadič i SPI dekodér jsou komponenty implementované v FPGA pomocí jazyka VHDL. Ukázka propojení SPI řadiče, SPI dekodéru a řadiče komponenty je na obr. 2.5.



Obrázek 2.5: Propojení SPI řadiče, SPI dekodéru řadiče komponenty [20]

2.1.7 LCD displej

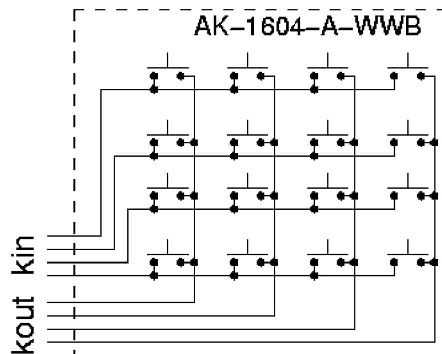
Součástí FITkitu 1.x je šestnáctiznakový jednořádkový (16x1) LCD displej CM1610NR-j2. FITkit 2.0 je osazen dvouřádkovým displejem (16x2) CM160224. Displej je fyzicky připojen přímo na piny FPGA. V FPGA je implementován (popsán v jazyce VHDL) řadič LCD displeje, který je ovládán MCU pomoci ovladače řadiče. Komunikace mezi MCU a dekodérem řadiče panelu LCD v FPGA probíhá přes rozhraní SPI.

2.1.8 Klávesnice

FITkit je vybaven alfanumerickou klávesnicí AK-1604-A-WWB s 16 tlačítky zapojenými do matice 4x4. Detekce zmáčknutí tlačítka probíhá tak, že na řádky (kin - vstupy klávesnice) postupně přivádíme signál a na sloupcích (kout - výstupech klávesnice) ověřujeme zmáčknutí. Data z klávesnice lze zpracovávat více způsoby. Prvním způsobem je zpracování již v FPGA. Druhým způsobem je zpracování v MCU. To je možné provést aktivním čtením dat z řadiče klávesnice přes dekodér řadiče klávesnice nebo čekat na přerušení od řadiče klávesnice, kdy není potřeba stále testovat jestli byla zmáčknuto nějaké tlačítko.

2.1.9 VGA rozhraní

Jedná se o jednosměrné (výstupní) analogové rozhraní pro připojení zobrazovacího zařízení. Základ rozhraní tvoří dvojice synchronizačních pulsů, tzv. vertikální a horizontální synchronizace, a zobrazovaná data reprezentovaná třemi základními barevnými složkami. Výsledná barva se skládá z červené (red), zelené (green) a modré (blue) složky - RGB model. Na FITkitu je každá barevná složka reprezentována 3 bity, z toho vyplývá, že je možné zobrazit maximálně $2^9 = 512$ barev. Pozice právě zobrazovaných dat je určena pomocí vertikálního a horizontálního synchronizačního pulsu. Parametry zobrazení (rozlišení



Obrázek 2.6: Zapojení tlačítek v klávesnici [12]

a obnovovací frekvence) jsou automaticky nastaveny zobrazovacím zařízením podle délky a trvání synchronizačních pulzů.

2.1.10 Řadič SDRAM

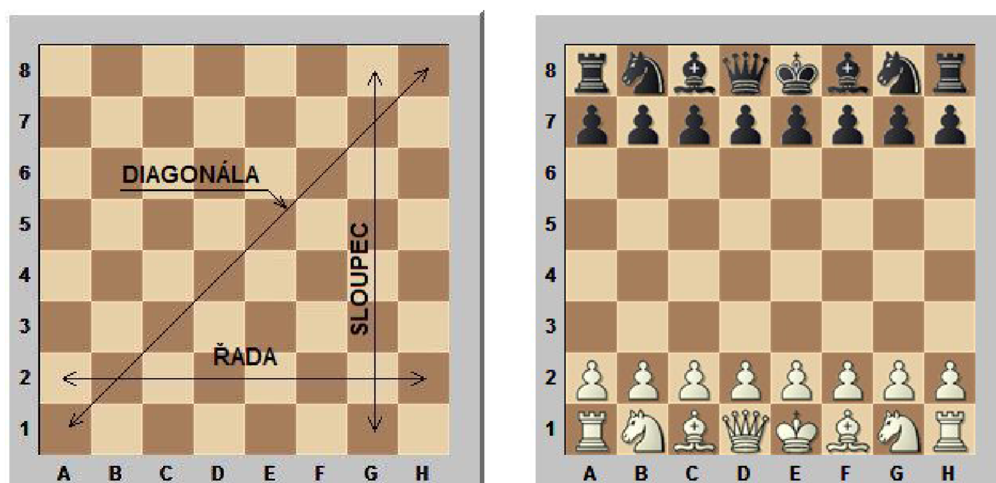
FITkit je osazen SDRAM pamětí s označením GM72V66841ET7K. Jedná se o synchronní dynamickou paměť typu RAM (Random Access Memory, paměť s přímým přístupem) s kapacitu 8 MB. Paměť je rozdělena na 4 banky, každý tvoří adresovatelný blok o velikosti 512 sloupců x 4096 řádků. Dynamické paměti jsou složeny z kondenzátorů, ty ale nedokáží udržet informaci příliš dlouho, a proto je potřeba provádět tzv. refresh. Celou paměť je nutné obnovit v čase 64 ms.

2.2 Šachy

Šachy nebo též šach jsou klasickou deskovou hrou pro dva hráče, která rozvíjí tvůrčí myšlení, paměť, představivost, přesnost, vůli. Staly se společenskou záležitostí, sportem, životním stylem, zábavou. Informace v této kapitole jsem čerpal z [1], [5] a [10].

Šachy se hrají na šachovnici, čtvercové desce pravidelně rozdělené na 8x8 čtvercových polí. Šachovnice je zobrazena na obr. 2.7(a). Pole jsou střídavě zbarvena světle (označovaná jako bílá pole) a tmavě (černá pole). Šachovnice je mezi hráči umístěna tak, že každý z nich má po své pravé ruce rohové pole bílé. Každá strana má na začátku hry celkem šestnáct kamenů šesti druhů. Jsou to král, dáma, dvě věže, dva střelci, dva jezdci (těmto kamenům říkáme figury) a osm pěšců. Na obr. 2.7(b) je zobrazena šachovnice se základním postavením kamenů.

Šachová partie je soubojem mezi dvěma soupeři (protihráči), kteří střídavě přemisťují kameny na šachovnici. Hru vždy zahajuje hráč s bílými kameny. Tento hráč se označuje jako *bílý* a jeho soupeř jako *černý*. Hráč je na tahu poté, co jeho soupeř provedl tah. Cílem hry, ke kterému směřují oba soupeři, je dát soupeřovu králi *mat*. Pod pojmem mat rozumíme takové ohrožení krále, které se již nedá odvrátit. Hráč, který dal soupeřovu králi mat, hru vyhrává. Podle pravidel Mezinárodní šachové federace FIDE [10] není dovoleno ponechat krále v ohrožení, vystavit krále do ohrožení nebo sebrat soupeřova krále.



(a) Šachovnice

(b) Šachovnice s kameny v základním postavení

Obrázek 2.7: Šachovnice [5]

2.2.1 Dějiny šachové hry

Kolem původu šachu panují dohady a není zcela jasně určeno kde a kdy hra vznikla. Za prapůvodce šachové hry je označována hra čaturanga z Indie. Název hry můžeme přeložit jako „čtyři součásti vojska“, kterými jsou pěchota, jízda, sloni a válečné vozy – předchůdci dnešních šachových pěšců, jezdců, střelců a věží. Součástí hry byl hod čtyřstěnnou kostkou, který hráči určil kterým kamenem má hrát. Jednalo se o hru určenou pro čtyři hráče, přičemž vždy dva a dva hráči tvořili dvojici, jejímž cílem bylo zajmout všechny kameny protivníků.

Strategická desková hra šatrandž pocházející z Persie vznikla někdy v 6. století ze hry čaturanga. Hra byla určena pouze pro dva hráče a ve hře se již nepoužívala hrací kostka. Ze hry tak odpadnul prvek náhody. Ve hře již věž, jezdec a král (s výjimkou rošády) táhli stejně jako v moderním šachu.

Počátkem 13. století se v Itálii a Španělsku pravidla šatrandže začala pomalu pozměňovat, až koncem 15. století hra dostala v zásadě tu podobu, v jaké je známá dnes. Právě konec 15. století můžeme považovat za počátek moderní hry. V té době také začal rozvoj ranné šachové teorie. Mistři šachu 16. a 17. století vytvořili základy teorie zahájení jako italská hra, královský gambit a španělská hra a začali analyzovat jednoduché koncovky. Centrum evropského šachového života se v 18. století přesunulo z jihoevropských zemí do Francie. Během 19. století se rychle rozvíjí šachový život. Vzniká množství šachových klubů, knih a časopisů.

První moderní šachový turnaj se konal v Londýně roku 1851 a lze jej považovat za počátek šachu jako sportu. V Paříži roku 1924 byla založena Mezinárodní šachová federace, která každoročně pořádá turnaj o titul mistra světa. Mezi nejznámější jména šachového světa patří Anatolij Karpov nebo Kavkazan Garri Kasparov. Informace v této kapitole byly čerpány z [3] a [22], kde lze nalézt i více podrobností.

2.2.2 Šach, Mat, Pat

Král, který je napaden soupeřovou figurou, je v `šachu`. Proti tomuto se musí bránit. Buď ústupem, nebo (pokud je to možné) braním šachující figury, nebo (opět, je-li to možné) představením vlastní figury. Šach, ze kterého už není úniku, se nazývá `mat` nebo také „šach-mat“, což v překladu znamená „král je mrtev“ (šach = persky král, mat = arabsky mrtev). Představte si zvláštní případ, kdy král strany na tahu není ohrožen šachem, ale nemůže táhnout, aniž by vstoupil do šachu a ani jeho ostatní figury nemají k dispozici možný tah. Takové situaci se říká `pat`. Partie končí nerozhodně.[2]

2.2.3 Pohyb figur

Pod pojmem *tah* rozumíme přesunutí jednoho kamene v souladu s pravidly. Výjimku tvoří rošáda, při které se současně přesune král i věž. Tah na pole s kamenem soupeře se nazývá *braní*. Soupeřův kámen je takovým tahem odstraněn ze šachovnice. Kameny není možné táhnout na pole, na kterém již je jiný kámen stejné barvy. Jediným kamenem, který smí provádět tah přes pole, které je obsazeno jiným kamenem, je jezdec. Kameny mohou táhnout podle následujících pravidel:

- *Král* - se pohybuje a bere soupeřovy kameny v libovolném směru, ale pouze na vzdálenost jednoho pole. Na rozdíl od ostatních kamenů se však nesmí postavit na pole, které je pod soupeřovou kontrolou. Z toho též vyplývá, že král nemůže brát soupeřův kámen, který je chráněn jinou figurou nebo pěšcem. Král se rovněž nesmí přiblížit k soupeřově králi, mezi oběma králi musí vždy zůstat alespoň jedno pole volné.

  Druhým způsobem tahu krále je tzv. **rošáda**. Král, kterým se při provádění rošády táhne nejdříve, se posune o dvě pole ve směru věže. Pak krále tato věž přeskočí a postaví se na sousední pole. Přitom rozlišujeme krátkou a dlouhou rošádu. Pokud táhneme králem a bližší věží na královském křídle, jedná se o krátkou rošádu. Při tahu krále a vzdálené věže na dámském křídle mluvíme o dlouhé rošádě. Rošádu je možné provést pokud králem a příslušnou věží ještě nebylo táhnuto, mezi králem a věží nestojí žádná figura, král nepřejde přes žádné pole, které ohrožuje soupeřova figura a král před provedením a po provedení rošády nemá šach. Provedení rošády je ukázáno na obr. 2.8(a) a 2.8(b).

- *Věž* - se pohybuje po řadách a sloupcích o libovolný počet polí, pokud ji nestojí v cestě jiný kámen.

- *Střelec* - se pohybuje diagonálně o libovolný počet polí, pokud mu nestojí v cestě jiný kámen. Střelec se po celou dobu partie pohybuje po polích stejné barvy. Proto střelce, který stál v základním postavení na bílém poli nazýváme bělopolný střelec a střelce, který stál v základním postavení na černém poli černopolný střelec.

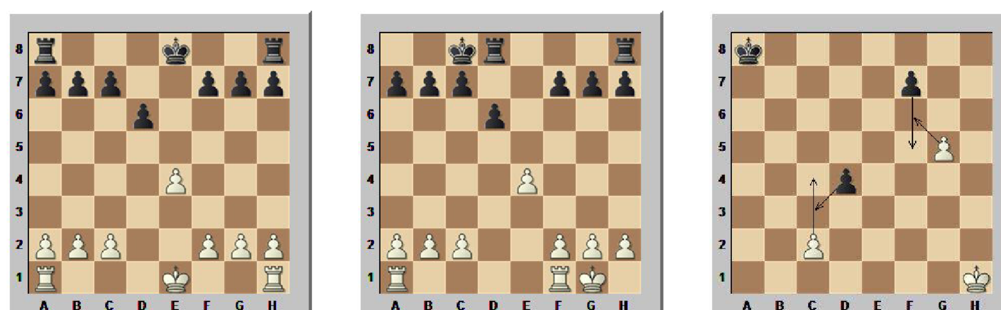
- *Dáma* - se může pohybovat po řadách, sloupcích i diagonálách o libovolný počet polí, pokud ji nestojí v cestě jiný kámen. Možnost táhnout ve všech směrech z ní činí nejsilnější figuru na šachovnici.

- *Jezdec* - se pohybuje skoky ve tvaru písmene L. Jezdec skáče vždy z bílého pole na černé a z černého na bílé, může přeskakovat vlastní i cízí figury.

[2] Převzato z [1]

- *Pěšec* - se pohybuje pouze směrem dopředu, a to o jedno pole, pokud je toto pole neobsazené. Pouze ze základního postavení se může (ale nemusí) posunout o dvě pole kupředu, pokud jsou obě pole volná. Pěšec může vzít soupeřův kámen, který je na úhlopříčně sousedícím poli před polem, na kterém stojí. Zvláštním tahem je tzv. **braní mimochodem** (francouzky en passant – doslovně *během míjení*), při kterém je možné vzít soupeřova pěšce, který se posunul ze svého základního postavení o dvě pole vpřed, a to stejným způsobem, jako kdyby se tento pěšec pohnul pouze o jedno pole. Tento tah je ukázán na obr. 2.8(c).

  Podaří-li se pěšci dojít na poslední řadu (tzn. bílý pěšec na osmou řadu, černý pěšec na první řadu), je ve stejném tahu odstraněn z desky a nahrazen na tomto poli dámou, věží, střelcem nebo jezdcem dle volby hráče (tzv. **proměna**). Přitom se kámen ihned účastní hry, může tedy např. dát soupeřovu králi šach. Díky proměně může mít hráč dvě i více dam.



(a) Postavení před rošádou

(b) Postavení po malé rošádě bílého a velké rošádě černého

(c) Braní mimochodem

Obrázek 2.8: Zvláštní tahy v šachu [5]

2.3 Jazyk VHDL

VHDL (VHSIC Hardware Description Language) je typovaný programovací jazyk sloužící k popisu hardware. Vznik jazyka VHDL je úzce spjat se vznikem projektu VHSIC (Very High Speed Integrated Circuit), který byl v roce 1980 iniciován ministerstvem obrany Spojených států amerických. Cílem projektu byl vývoj nových výkonných integrovaných obvodů s vysokou hustotou integrace. VHDL je standardem IEEE 1076 z roku 1987 (VHDL 87), byl revidován a v roce 1993 přijat jako standart IEEE 1164 (VHDL 93). Používá se k návrhu a simulaci digitálních integrovaných obvodů, např. hardwarových struktur pro programovatelná hradlová pole. Nejnovější verze jazyka je použitelná i pro návrh analogových obvodů.

Jazyk VHDL není vázán na žádnou konkrétní cílovou technologii. Konečná realizace navržené struktury je závislá až na syntéze VHDL kódu. Jednoduše řečeno – pomocí tohoto jazyka lze provádět návrhy pro hradlová pole většiny výrobců (Xiling, Altera, Lattice apod.) a následně použít vhodný kompilátor. Jazyk VHDL je určen k syntéze obvodů stejně tak jako k simulaci obvodů. Je však potřeba mít na mysli, že ačkoli VHDL kód je bez problému zcela simulovatelný, ne všechny konstrukce jsou syntetizovatelné. Informace v této kapitole jsem čerpal z [8], [4] a [11].

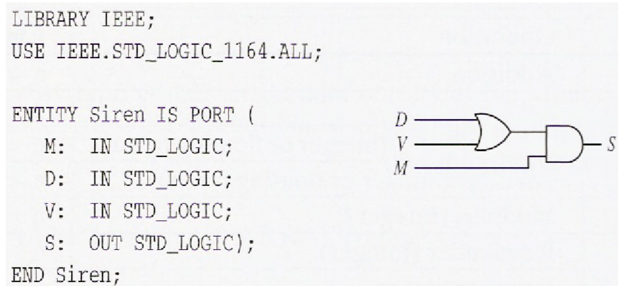
2.3.1 Základní prvky jazyka

V jazyce VHDL je hardware popisován pomocí komponent. K popisu komponent jsou používány knihovny (LIBRARY) zpřístupňující knihovní funkce, entity (ENTITY) definující rozhraní komponenty a architektury (ARCHITECTURE) definující strukturu či chování komponenty. Přitom jedna entita může mít více architektur.

Entita definuje rozhraní komponenty. Komunikační rozhraní tvoří porty reprezentované signály. Na základě módu signálu IN, OUT, INOUT nebo BUFFER je vybrán směr komunikace na portu. Datová šířka portu je určena šířkou signálu. Formální syntaxe entity je ukázána na obr. 2.9. U entity, která je nejvýše v hierarchii (zapouzdřuje projekt), jsou porty fyzicky připojeny na vývody hradlového pole. Definice entity pro funkci $F = (D\ or\ V)\ and\ M$ a ukázka logického obvodu této funkce je zobrazena na obr. 2.10.

```
ENTITY entity_name IS
   PORT (
      port_name : signal_mode signal_type;
      port_name : signal_mode signal_type;
      ...);
END entity_name;
```

Obrázek 2.9: Formální syntaxe entity [8]



Obrázek 2.10: Definice entity a logický obvod funkce [6]

Architektura určuje chování komponenty, vnitřní strukturu a její funkcionalitu. Architektura je vždy svázána s entitou, která určuje způsob komunikace s okolím. Formální syntaxe architektury je ukázána na obr. 2.11. Je rozdělena na deklarační a příkazovou část. Deklarační část je určena k deklaraci signálů a komponent. Příkazová část obsahuje implementaci architektury pomocí behaviorálního, strukturního nebo data flow popisu. Přitom můžeme využít kombinaci těchto tří popisů.

2.3.2 Behaviorální popis

Behaviorální popis popisuje obvod nebo jeho část algoritmem na základě jeho chování. Základním prvkem popisu jsou procesy, které mezi sebou komunikují pomocí signálů. K popisu procesů jsou použity algoritmy obsahující příkazy, podmínky, cykly a funkce. Po spuštění procesu jsou příkazy vykonávány sekvenčně. U procesu je nutné definovat seznam citlivých signálů, na které potom reaguje. Úkolem procesu je definovat výstupy na základě stavu nebo

```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

Obrázek 2.11: Formální syntaxe architektury [8]

změny vstupních signálů. Při použití tohoto popisu nemusí být zřejmá obvodová realizace. Behaviorální popis architektury je ukázán na obr. 2.12.

```
ARCHITECTURE Siren_Behavioral OF Siren IS
   SIGNAL term_1: STD_LOGIC;
BEGIN
   PROCESS (D, V, M)
   BEGIN
      term_1 <= D OR V;
      S <= term_1 AND M;
   END PROCESS;
END Siren_Behavioral;
```

Obrázek 2.12: Behaviorální popis architektury [6]

2.3.3 Strukturní popis

Strukturní popis popisuje architekturu pomocí vzájemně propojených kompenent. Komponenty se mohou skládat z dalších samostatně popsaných komponent. Je tak možné vytvářet hierarchii komponent. Popis tímto způsobem je blízký konečné obvodové realizaci. Strukturní popis architektury je ukázán na obr. 2.13.

```
ARCHITECTURE Siren_Structural OF Siren IS
   COMPONENT myOR PORT (
     in1, in2: IN STD_LOGIC;
     out1: OUT STD_LOGIC);
   END COMPONENT;
   SIGNAL term1: STD_LOGIC;
BEGIN
   U0: myOR PORT MAP (D, V, term1);
   S <= term1 AND M;
END Siren_Structural;
```

Obrázek 2.13: Strukturní popis architektury [6]

2.3.4 Data flow popis

Tento popis definuje chování uvnitř architektury pomocí přiřazovacích příkazů. U tohoto popisu můžeme použít také podmíněného přiřazení signálů. Data flow popis architektury je ukázán na obr. 2.14.

```
ARCHITECTURE Siren_Dataflow OF Siren IS
   SIGNAL term_1: STD_LOGIC;
BEGIN
   term_1 <= D OR V;
   S <= term_1 AND M;
END Siren_Dataflow;
```

Obrázek 2.14: Data flow popis architektury [6]

Kapitola 3

Návrh řešení

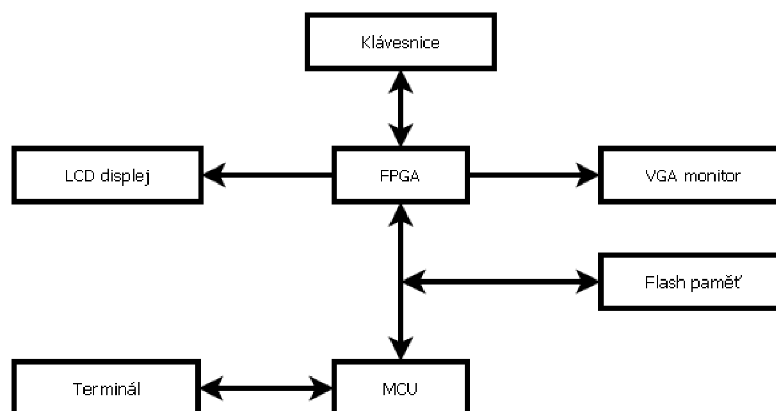
3.1 Obecný úvod

Návrh řešení je nedílným základem vývoje aplikace (software, hardware). Kvalitní návrh usnadňuje implementaci a přispívá k výsledné kvalitě produktu. Při návrhu a následné implementaci aplikace jsem se snažil docílit optimálního poměru mezi paměťovou a časovou složitostí programového kódu s mírnou váhou na paměťovou optimalizaci. Stejně tak jsem se snažil i o efektivní využití zdrojů FPGA.

Návrh projektu byl přizpůsoben požadavku pro funkčnost i bez propojení s počítačem. Tzn., že hra musí být funkční pouze na FITkitu s připojeným zobrazovacím zařízením. Výsledná aplikace je plně funkční na FITkitu 2.0 i starší verzi FITkitu 1.x.

3.2 Struktura projektu

Výsledný projekt je souborem navzájem propojených elementárních prvků. Propojení prvků je zobrazeno na obr. 3.1. Řídícím prvkem projektu je MCU. Uživatel s ním může přímo komunikovat za pomoci terminálu. Dalším velice důležitým prvkem je FPGA, přes které je připojen LCD displej, klávesnice a VGA monitor. FPGA zprostředkovává MCU výstup textu na LCD displej, příjem příkazů z klávesnice a komunikaci s jednotkou pro tvorbu obrazu. K ukládání hry MCU využívá Flash paměť.



Obrázek 3.1: Blokové schéma projektu

3.3 Návrh FPGA

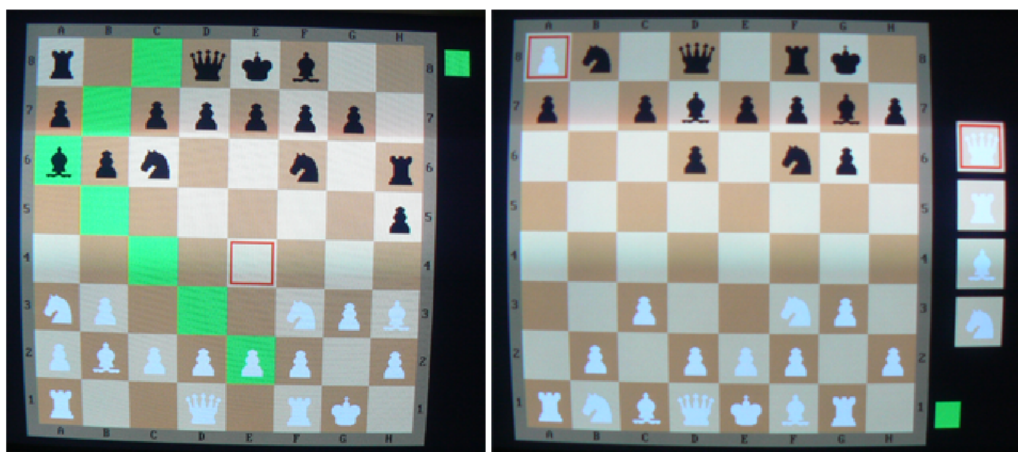
V této části práce je popsán návrh zobrazovací jednotky a potřebných prostředků pro její realizaci.

3.3.1 Grafický návrh

V této kapitole bude popsán postupný vývoj grafického návrhu pro hru šachy. Při vytváření návrhu byl kladen důraz na zobrazení co možná nejvíce podstatných informací v grafickém výstupu. Grafický výstup hry je zobrazen na obr. 3.2.

Navržený grafický návrh je postaven na upraveném principu grafiky hry Tetris [7]. Tato hra využívá rozlišení 640x480 bodů. Obrazovku dělí na 60 řádků a 64 sloupců, čímž vznikají pole o rozměrech 10x8 bodů, která jsou barvena. Konečný grafický výstup je tvořen odpovídajícím vybarvením polí.

Pro hru šachy se jako dostatečné ukázalo být stejné rozlišení tj. 640x480 bodů. Narozdíl od hry Tetris však obrazovka není rozdělena zcela pravidelně, ale na oblasti. Máme-li šachovnici o 8 řadách a 8 sloupcích, pak pole šachovnice bude mít rozměry 60x60 bodů ($480/8 = 60$ bodů). Tento rozměr pole je naprosto vyhovující pro zobrazení figur. V další fázi jsem použil běžné značení šachovnici písmeny A až H nahoře a dole, z levé a pravé strany pak číslicemi 1 až 8. Textury písmen a číslic mají rozměr 8x16 bodů (16 řádků x 8 sloupců). Po přidání orámování nad a pod šachovnicí ($16 + 16 = 32$ bodů navíc) bylo nutné upravit rozměry pole šachovnice na 56x56 bodů ($(480-32)/8 = 56$ bodů). Konečným rozšířením bylo zobrazení čtverečku vpravo nahoře nebo vpravo dole od šachovnice, který upozorňuje, kdo je právě na tahu a zobrazení figur pro výměnu za pěšce, došel-li pěšec na opačnou stranu šachovnice.

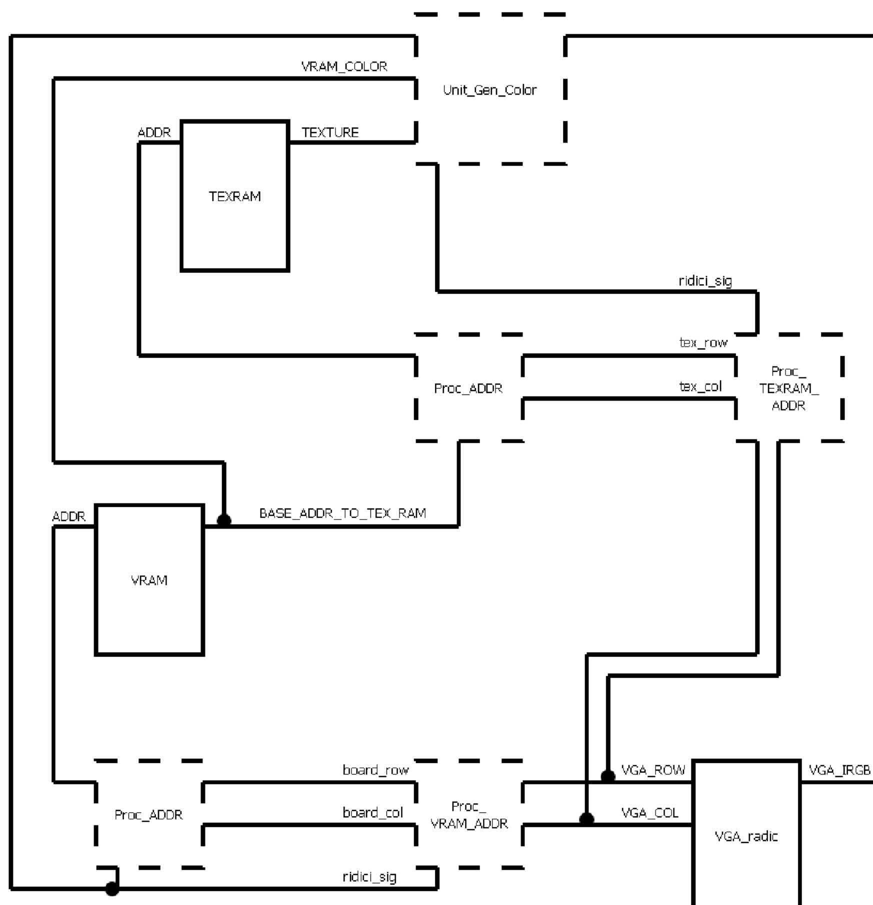


Obrázek 3.2: Grafický výstup hry na zobrazovacím zařízení

3.3.2 Zobrazovací jednotka

Úkolem zobrazovací jednotky je vytvářet grafický výstup pro zobrazovací zařízení. Struktura zobrazovací jednotky je zobrazena na obr. 3.3. Srdcem jednotky je VGA řadič, ten nastavuje signály VGA_ROW a VGA_COL. Tyto dva signály určují, pro který bod na zob-

razovacím zařízení se očekává na signálu VGA_IRGB barva pixelu. Při rozlišení 640x480 nabývají signály VGA_ROW a VGA_COL hodnot 0 – 639, respektive 0 – 479.



Obrázek 3.3: Struktura zobrazovací jednotky

Jednotku lze logicky rozdělit na čtyři části:

- část zobrazující šachovnici s figurami
- část zobrazující orámování šachovnice
- část zobrazující kdo je právě na tahu
- část zobrazující figury pro výměnu za pěšce

Princip zobrazení šachovnice a figur

Ve video RAM (dále jen VRAM) je pro každé pole šachovnice uložen záznam. Podrobnější popis záznamu lze nalézt v kapitole 3.3.4. Na základě čísla řádku a sloupce z VGA řadiče je vypočítáno číslo řady a sloupce šachovnice a dále pak číslo řádku a sloupce v rámci pole. Z čísla řady a sloupce šachovnice je sestavena adresa pro VRAM. Pokud záznam z VRAM oznamuje, že pole obsahuje figuru, je potřeba načíst pixel z textury uložené v texture RAM (dále jen TEXRAM). Adresa pro TEXRAM se sestavuje z bázové adresy textury (uloženo

v záznamu VRAM) a z čísla řádku a sloupce v rámci pole. Je-li pole šachovnice prázdné, z TEXRAM se nic nenačítá a o vše se již postará jednotka UNIT_GEN_COLOR. Jednotka po vyhodnocení VRAM záznamu nastaví barvu pixelu pro VGA řadič.

Princip zobrazení orámování šachovnice

Pokud má být zobrazena textura označející řadu nebo sloupec šachovnice, postupuje se obdobně jako při zobrazení textury figury. Sestaví se adresa pro VRAM, adresa pro TEXRAM z bázové adresy textury a pozice v rámci textury a nakonec jednotka pro nastavení barvy nastaví barvu. V případě, že má být zobrazeno pouze pozadí, je o tom jednotka informována speciálním řídícím signálem.

Princip zobrazení kdo je právě na tahu

Princip je stejný jako při zobrazení prázdného pole šachovnice.

Princip zobrazení figur pro výměnu za pěšce

Pokud dojde hráč pěšcem na protější stranu šachovnice, zobrazí se figury, za které je možno pěšce vyměnit. Figury mají vždy stejnou barvu pole jako pole, na kterém stojí pěšec. Princip zobrazení je shodný jako při zobrazení figury na šachovnici.

3.3.3 Výběr paměti

Pro VRAM paměť bylo vhodné použít blokovou paměť BRAM. Hlavním kritériem pro toto rozhodnutí byla možnost nezávislého čtení a zápisu dat z/do paměti. K uložení textur bylo možné použít buď pamět SDRAM nebo paměť BRAM. Použití paměti SDRAM by vyžadovalo návrh a implementaci vlastního inteligentního řadiče paměti. Ačkoliv není nezbytně nutné čtení a zápis dat textur, na začátku je nezbytné do paměti uložit textury figur. Fyzické parametry paměti SDRAM neumožňují souběžné čtení a zápis dat a proto by bylo nezbytné, aby námi implementovaný řadič byl schopný odstavit požadavky na čtení dat a umožnil tak nahrání textur do paměti. Přesto, že mají BRAM paměti oproti SDRAM paměti výrazně menší kapacitu, rozhodl jsem se pro uložení textur použít BRAM paměť.

Block RAM (BRAM) paměť je bloková dvouportová paměť o kapacitě 2 KB s volitelnou organizací dat. Datovou šířku pro paměť RAMB16_S[wA]_S[wB] zvolíme nahrazením výrazů wA a wB, které určují šířku dat portu A a B, za požadované hodnoty. Možná nastavení paměti jsou uvedena v tabulce na obr. 3.4.

Organization	Memory Depth	Data Width	Parity Width	DI/DO	DIP/DOP	ADDR	Single-Port Primitive	Total RAM Kbits
512x36	512	32	4	(31:0)	(3:0)	(8:0)	RAMB16_S36	18K
1Kx18	1024	16	2	(15:0)	(1:0)	(9:0)	RAMB16_S18	18K
2Kx9	2048	8	1	(7:0)	(0:0)	(10:0)	RAMB16_S9	18K
4Kx4	4096	4	-	(3:0)	-	(11:0)	RAMB16_S4	16K
8Kx2	8192	2	-	(1:0)	-	(12:0)	RAMB16_S2	16K
16Kx1	16384	1	-	(0:0)	-	(13:0)	RAMB16_S1	16K

Obrázek 3.4: Bloková paměť BRAM – organizace dat [24]

3.3.4 Video RAM

V paměťi video RAM (VRAM) jsou uloženy VRAM záznamy. Záznamy obsahují informace o výše zmíněných částech zobrazovací jednotky (poli šachovnice, atd.). Struktura VRAM záznamu pro pole šachovnice je zobrazena v tab. 3.1. Při zobrazování figur pro výměnu za pěšce se využívá stejný záznam jako pro pole šachovnice, jediný rozdíl je ve významu nejnižšího bitu. Ten slouží k určení, jestli se mají zobrazit figury, nebo mají být skryty. Pro orámování šachovnice jsou použity ve VRAM záznamu pouze 4 nejnižší (nejméně významné) bity, více v tab. 3.2.

Bit(y)	**Význam bitu(ů)**	**Hodnota**	**Význam hodnoty**
0	příznak prázdnosti pole	0	na poli je figura
		1	pole je prázdné
0	příznak zobrazení figur (při výměně za pěšce)	0	zobrazit figury
		1	nezobrazovat figury
1	barva figury	0	černá figura
		1	bílá figura
432	barva pole	000	bílé pole
		001	černé pole
		010	zelené pole
		011	oranžové pole
		100	červené pole
		101 – 111	nevyužito
765	bázová adresa textury	000	vež
		001	střelec
		010	dáma
		011	král
		100	pěšec
		101	jezdec
		110	nevyužito
		111	nevyužito
8	příznak aktivního pole	0	pole je aktivní
		1	pole není aktivní

Tabulka 3.1: Struktura a význam bitů VRAM záznamu pro pole šachovnice

3.3.5 Texture RAM

V paměti texture RAM (TEXRAM) jsou uloženy textury figur a textury čísel a písmen pro orámování šachovnice. Textury čísel a písmen byly přejaty z projektu [21].

Návrh uložení textur

Při ukládání textur do paměti jsem narazil na problém – nedostatek paměťi. Bylo potřeba navrhnout co nejefektivnější způsob uložení textur bez zbytečných redundantních dat. Figury věž, střelec, dáma, král a pěšec jsou osově souměrné podle vertikální osy. Toho můžeme efektivně využít a snížit tak paměťovou náročnost každé figury na polovinu. Postačí nám

Bity	Význam bitů	Možné hodnoty	Význam hodnoty
3210	bázová adresa textury	0000	1
		0001	2
		0010	3
		0011	4
		0100	5
		0101	6
		0110	7
		0111	8
		1000	A
		1001	B
		1010	C
		1011	D
		1100	E
		1101	F
		1110	G
		1111	H
7654	bity nejsou využity		

Tabulka 3.2: Struktura a význam bitů VRAM záznamu pro orámování šachovnice

totiž uložit pouze polovinu textury a druhou polovinu získáme ozrcadlením. Blíže popsáno v kap. 4.1.9. K uložení textur jsem se dále snažil použít nějaký druh komprese, např. RLE (Run Length Encoding) kompresi. Použití komprese se ale ukázalo být nepříliš efektivní. Důvodem neefektivity byla nutnost komprimovat příliš malé shluky dat.

Vezmeme-li v potaz fakt, že pole šachovnice může mít dvě barvy (světlou, tmavou) a figury jsou také dvou barev (bílé, černé), pak pro uložení těchto čtyř hodnot každého pixelu textury potřebujeme 2 bity. Pokud bychom ukládali všechny figury (bílé i černé) s dvěma barvami pole, pak bychom museli uložit 24 textur ($2barvyFigur*6figur*2barvyPolí= 24textur$). Navíc při generování tahů, popsáno v kap. 4.2.5, se u validních tahů barva pole mění na zelenou. Bylo by tak zapotřebí uložit dalších 12 textur navíc. Tento problém můžeme vyřešit uložením barvy pole a barvy figury pole šachovnice do VRAM záznamu. Textura tak bude obsahovat pouze pixely, které nám budou říkat kde je figura a kde je barva pole. Dosáhneme tak redukce. K uložení jednoho pixelu textury budeme potřebovat pouze 1 bit. Celkově budeme potřebovat 6 textur.

Jak již bylo zmíněno dříve, pole šachovnice má rozměry 56x56 pixelů a znak rámování 8x16 pixelů. Rámování nad (pod), resp. vlevo (vpravo) od pole, polem zabírá prostor 16x56 pixelů. V paměti je však uložena pouze textura znaku. O vykreslování pozadí se stará jednotka UNIT_GEN_COLOR. Tím došlo ke snížení paměťové náročnosti na 1/7.

3.4 Návrh MCU

Tato část práce popisuje potřebné datové struktury a úkoly MCU.

3.4.1 Úkoly MCU

MCU plní tyto úkoly:

- zajišťuje komunikaci s uživatelem
- inicializuje a řídí hru
- řídí zobrazovací jednotku
- generuje možné tahy
- provádí kontrolu tahu
- provádí uložení a načtení hry

3.4.2 Datové struktury

Pro ovládání a kontrolu hry je nutné mít informace o rozestavení figur na šachovnici. Bylo možné tyto informace číst z VRAM paměti. Pro účely kontroly hry však tyto informace nejsou vhodně strukturované a navíc velký počet přístupů k paměti by mohl zpomalit hru. Proto jsou informace o rozmístění figur na šachovnici ukládány do pole `chessBoard[8][8]`. Přitom pozice `[0][0]` je v levém horním rohu a pozice `[7][7]` v pravém dolním rohu šachovnice. Práci s polem ulehčilo a zpřehlednilo uložení figur jako symbolických konstant.

Ve struktuře `Game` 3.4 je uložen příznak toho, která ze stran je na tahu (proměnná `hrajeBila`). Je-li hodnota `true` na tahu je BILA. Nastavením proměnné `changePesec` se hra dostane do stavu, kdy čeká, až si uživatel vybere figuru za pěšce, kterým došel na opačnou stranu šachovnice. Konec hry je signalizován nastavením proměnné `gameOver` na hodnotu logická 1.

Struktura `BoardState` 3.3 obsahuje informaci o aktivním poli (proměnné `xAktiv`, `yAktiv`). Jedná se o aktuálně zaměřené pole šachovnice, které slouží k ovládání hry. Podrobnější popis můžeme nalézt v kapitole 3.4.3. Proměnná `takeFigure` je příznakem, jestli hráč zvednul figuru a hodlá s ní táhnout. Pozici figury určují proměnné `xTake`, `yTake`. Před provedením rošády jsou kontrolovány proměnné `kralMoved`, `leftVezMoved` a `rightVezMoved`. Tedy jestli nebylo již táhnuto králem a patřičnou věží. Poslední proměnná `xChangePesec` určuje pozici (figuru) v rámci nabídky výměny pěšce za figuru.

3.4.3 Ovládání hry

Při návrhu ovládání hry jsem se snažil, aby ovládání bylo jednoduché a intuitivní. Hra se ovládá pomocí klávesnice na FITkitu, tlačítky 1 – 9. Tlačítkem '2' se posouvá nahoru, '8' dolu, '4' doleva, '6' doprava a tlačítka '1', '3', '7' a '9' slouží k diagonálnímu posunu. Tlačítko '5' se používá pro tah figurou – zvednutí a položení figury. Hru lze uložit stisknutím tlačítka 'A'. Nahrání uložené hry provede tlačítko 'B'. Novou hru je možné začít hrát po stisknutí tlačítka 'D'.

3.4.4 Uložení hry

K uložení hry je použita paměť Flash. Hra ukládá stav figur na šachovnici, která ze stran je na tahu a pro obě strany aktivní pole a zda již bylo taženo králem a věžemi. Po nahrání hry tak nemůže dojít k situaci, že figurami již bylo táhnuto a přesto je možné provést rošádu.

```
typedef struct BoardState
{
 short xAktiv;
 short yAktiv;
 bool takeFigure;
 short xTake;
 short yTake;
 bool kralMoved;
 bool leftVezMoved;
 bool rightVezMoved;
 short xChangePesec;
} TBoardState;
```

Tabulka 3.3: Struktura BoardState

```
typedef struct Game
{
 bool hrajeBila;
 bool changePesec;
 bool gameOver;
} TGame;
```

Tabulka 3.4: Struktura Game

Hru je možné uložit pouze pokud není zvednuta nějaká figura. Vysvětlení tohoto omezení a implementačních podrobností je v kapitole 4.2.11.

Kapitola 4

Implementace

V této kapitole budou podrobněji specifikovány implementační detaily. Implementace vychází z předchozího návrhu projektu a je rozdělena na hardwarovou a softwarovou část. Implementace těchto dvou částí probíhala do značné míry odděleně. Implementaci zobrazovací jednotky však bylo nutné provádět současně v FPGA i MCU. Hra je implementována podle pravidel Mezinárodní šachové federace FIDE [10].

4.1 Implementace FPGA

Hardware pro FPGA je implementován v jazyce VHDL (popsán v kapitole 2.3). K popisu architektury jsem použil behaviorální a data flow popis. Součástí top-level entity je popis komponent s napojením na spi rozhraní a popis zobrazovací jednotky.

4.1.1 Výběr top-level entity

SVN repozitář k projektu FITkit poskytuje tři top-level entity. Dále pod pojmem entita bude myšlena top-level entita. Tyto entity integrují nezbytné komponenty (generátor hodinového signálu, řadič rozhraní SPI). Rozdíl mezi entitami je v tom, jaké poskytují rozhraní. Entita `bare` je základní entita, která je vhodná pro aplikace nevyužívající periferie na FITkitu ani port X. Port X je skupina vstupně-výstupních portů, které jsou fyzicky vyvedeny na propojovací pole JP10. Pokud aplikace potřebuje využít port X, vhodnou entitou je entita `gp`. Pro aplikace využívající periferie FITkitu (VGA port, PS/2, atd) je určena entita `pc`. Periferie sdílejí část portu X a pro jejich povolení je nutné aktivovat propojku JP6.

Volba top-level entity se provádí pomoci atributu `architecture` v souboru *project.xml* v sekci FPGA. Pro tento projekt byla použita entita `pc`, protože program využívá VGA port.

4.1.2 Použité komponenty

V rámci architektury top-level entity jsou popsány tyto komponenty: SPI řadič, klávesnice, LCD displej a BRAM paměti. Také jsou instancovány řadiče komponent a SPI adresové dekodéry. Významné komponenty budou postupně popsány.

4.1.3 SPI

SPI řadič umožňuje MCU komunikovat s řadiči komponent. Přitom je nutné dodat, že každý řadič komponenty je spárován se SPI adresovým dekodérem. SPI adresový dekodér

je potřeba nastavit tak, aby vytvářel disjunktní adresový prostor, díky kterému SPI řadič ví, s kterou komponentou má komunikovat. Tab. 4.1 zobrazuje nastavení SPI adresových dekodérů v tomto projektu.

Zařízení	**ADDR WIDTH**	**DATA WIDTH**	**ADDR OUT WIDTH**	**BASE ADDR**	**Adresový prostor**
LCD displej	8	16	1	0x00	0x00 – 0x01
klávesnice	8	16	1	0x02	0x02 – 0x03
VRAM	16	8	11	0x1000	0x1000 – 0x17FF
TEXRAM	16	8	14	0x4000	0x4000 – 0x7FFF

Tabulka 4.1: Nastavení SPI adresových dekodérů

4.1.4 VRAM

VRAM paměť je reprezentována BRAM pamětí s organizací dat RAMB16_S9_S9. Nastavení rozděluje paměť na 2048 buněk po 8 bitech s jedním paritním bitem navíc. Paritní bit je možné použít k uložení běžných dat, tudíž máme pro data k dispozici 9 bitů. Organizace dat ve VRAM je zobrazena v tab. 4.2.

00: pole[0][0]	01: pole[0][1]	- - -	07: pole[0][7]
08: pole[1][0]	- - -	- - -	15: pole[1][7]
- - -	- - -	- - -	63: pole[7][7]
64: border_1	65: border_2	- - -	71: border_8
72: border_A	73: border_B	- - -	79: border_H
80: change_Dama	81: change_Vez	82: change_Strelec	83: change_Jezdec
84: hraje_Bílá	85: hraje_Černá	86: Nevyužito.	- - -
- - -	- - -	- - -	2048: Nevyužito.

Tabulka 4.2: Organizace dat ve VRAM

Celkově je použito 86 z 2048 paměťových buněk. Na adresách 0 až 63 jsou uloženy VRAM záznamy polí šachovnice. Následuje 16 záznamů orámování šachovnice (8 vertikálních + 8 horizontálních) na adresách 64 až 79. Dále 4 záznamy pro zobrazení figur, které je možné vyměnit za pěšce, na adresách 80 až 83. Adresy 84 a 85 okupují dva záznamy označující, kdo právě hraje.

Adresace je založeno na procesu `comp_vram_addr` (popsán v kapitole 4.1.6), který nastavuje adresační signály a potřebné řídící signály.

4.1.5 TEXRAM

TEXRAM paměť je reprezentována BRAM pamětí s organizací dat RAMB16_S1_S1. Paměť je rozdělena na 16384 buněk po 1 bitu. Organizace dat v TEXRAM je zobrazena v tab. 4.3.

Proces `comp_texture_addr` (popsán v kapitole 4.1.7) nastavuje adresační signály v rámci dané textury a další nezbytné řídící signály. Adresace textury je zajištěna pomoci bázové

První adresa	**Poslední adresa**	**Textura**						
0	1567	věž	10976	11103	tex 1	12000	12127	tex A
1568	3135	střelec	11104	11231	tex 2	12128	12255	tex B
3136	4703	dáma	11232	11359	tex 3	12256	12383	tex C
4704	6271	král	11360	11487	tex 4	12384	12511	tex D
6272	7839	pěšec	11488	11615	tex 5	12512	12639	tex E
7840	10975	jezdec	11616	11743	tex 6	12640	12767	tex F
			11744	11871	tex 7	12768	12895	tex G
			11872	11999	tex 8	12896	13023	tex H

Tabulka 4.3: Organizace dat v TEXRAM

adresy textury získané z VRAM záznamu, adresačních signálů textury a řídících signálů z procesů `comp_vram_addr` a `comp_texture_addr`.

4.1.6 Proces `comp_vram_addr`

Proces nastavuje adresové signály (proměnné `board_row` a `board_col`) a řídící signály. Seznam řídících signálů s popisem významu je zanesen v tab. 4.4. Signál je pravdivý, jestliže nabývá logické hodnoty 1. V jednom okamžiku je vždy aktivní pouze jeden řídící signál. Vyjímkou jsou signály `sig_border_bg` a `sig_border_corner`, které jako jediné mohou být aktivní současně.

Signál	**Význam**
`out_of_bounds`	mimo vykreslované plochy
`sig_border_bg`	pozadí orámování šachovnice
`sig_border_corner`	roh orámování šachovnice
`sig_change`	textury figur pro výměnu za pěšce
`sig_change_bg`	mezery mezi texturami figur
`sig_on_move`	pole označující kdo je na tahu

Tabulka 4.4: Řídící signály

4.1.7 Proces `comp_texture_addr`

Úkolem procesu je nastavit adresové signály textury (proměnné `texture_row` a `texture_col`) a řídící signály `sig_border_h`, `sig_border_v`. Za tímto účelem používá signál `board_row`, `board_col` a `sig_border_corner`. Adresové signály nám udávají číslo řádku a sloupce pixelu, který má být z textury zobrazen. Tyto hodnoty jsou získány výpočtem z VGA vystavených signálů `vga_rrow` a `vga_rcol`. Řídící signály signalizují vykreslování horizontálních nebo vertikálních textur.

4.1.8 Proces `sig_delay`

Použitím BRAM pamětí při generování grafického výstupu vzniká požadavek na správné časování řídících signálů. Jinak řečeno je potřeba zajistit zpoždění řídících signálů o jeden až dva takty hodinového signálu `CLK`. Důvodem je sekvenční zpoždění vznikající na BRAM pamětech. Proces `sig_delay` se stará o nastavení správných hodnot do signálů uchovávajících zpožd̆ění řídících signálů.

4.1.9 Popis práce zobrazovací jednotky

Pro lepší pochopení zobrazovací jednotky bude následovat komplexní popis její funkcionality. Vše začíná u VGA řadiče. Ten vystaví na signálech `vga_rrow` a `vga_rcol` číslo řádku a sloupce pixelu, pro který na signálu `vga_irgb` očekává barvu, kterou zobrazí na zobrazovacím zařízení. Z čísla řádku a sloupce pixelu dále vychází proces `comp_vram_addr` a `comp_texture_addr`. Aby nedošlo k nějaké zkreslené představě, tak bych ještě rád upozornil, že vykreslování na výstupním zařízení probíhá pixel po pixelu, řádek po řádku. Nikoliv, jak by si možná někdo mohl chybně myslet, vykreslováním celých textur najednou. Pro lepší přehled je další postup rozdělen do několika scénářů:

1) Zobrazení šachového pole s figurou

Proces `comp_vram_addr` na základě čísla řádku a sloupce pixelu určí do kterého pole šachovnice tento pixel patří. Podle toho je nastaven řádek a sloupec do signálu `board_row` a `board_col`. Konkatenací těchto dvou signálů získáme adresu pro VRAM.

Proces `comp_texture_addr` použije dva výše zmíněné signály, s jejichž pomocí získá z řádku a sloupce pixelu na monitoru řádek a sloupec pixelu v textuře. Tomu odpovídá nastavení signálu `tex_row` a `tex_col`.

Nyní můžeme přejít k dalšímu kroku, kterým je analýza VRAM záznamu (ukázka v tab. 4.5). Bit 0 má hodnotu 0, což nám říká, že pixel, který máme zobrazit, je součástí pole šachovnice, na kterém je figura. Situace, kdy bit nabývá hodnoty 1, je rozebrána ve scénáři 4.1.9. Předpokládejme, že máme zobrazit černou dámu na světlém poli. Přikročme k získání informace o pixelu z textury. Z VRAM záznamu vyjmeme bázovou adresu textury. Ta má pro figuru dámy hodnotu 2, což nám definuje, že před texturou dámy jsou uloženy dvě textury. Bázová adresa má tudíž hodnotu 3136. Předpokládejme, že pixel který máme zobrazit leží na 20. řádku a ve 12. sloupci. Důležité je si uvědomit, že řádky a sloupce jsou číslovány od 0. Budeme tedy zobrazovat pixel z 19. řádku a 11. sloupce. Pro figury, které jsou osově souměrné a mají v TEXRAM uloženo jen polovinu textury je pro sloupce 29 – 56 (resp. 28 – 55) potřeba přepočítat číslo sloupce podle vzorce 4.1. Vzorec pro výpočet adresy pixelu v rámci textury je uveden v rovnici 4.2. Aplikací tohoto vzorce získáme adresu 543 (rovnice 4.3). Součtem bázové adresy figury (3136) s adresou pixelu v rámci textury získáme adresu 3679 v TEXRAM, na které leží požadovaná informace o zobrazovaném pixelu.

Posledním krokem je nastavit ve VGA řadiči barvu pixelu. Bit textury získaný z TEXRAM nese informaci o tom, jestli je pixel součástí figury (hodnota bitu je 0) nebo barvy pole (hodnota bitu je 1). Zobrazuje-li pixel část figury, je barva pixelu nastavena na černou. Volba barvy figury je závislá na hodnotě prvního bitu VRAM záznamu. Podle našeho zvoleného příkladu, kdy zobrazujeme figuru černé dámy, má bit hodnotu 0. Zobrazuje-li pixel barvu pole, pak podle našeho zvoleného příkladu bude pixelu nastavena barva světlého pole. Barvě

Bit	Význam
0	příznak prázdnosti pole
1	barva figury
432	barva pole
765	bázová adresa textury

bit	765	432	1	0
hodnota bin	010	000	0	0
hodnota dec	2	0	0	0

Tabulka 4.5: VRAM záznam – černá dáma na světlém poli

samozřejmě korespondují patřičné hodnoty ve VRAM záznamu.

$$spravne_cislo_sloupce = 55 - cislo_sloupce \quad (4.1)$$
$$adresa_pixelu = cislo_radku * pocet_pixelu_na_radek + cislo_sloupce \quad (4.2)$$
$$= 19 * 28 + 11 = 543 \quad (4.3)$$

2) Zobrazení prázdného šachového pole

Při zobrazování prázdného šachového pole se až po získání VRAM záznamu postup neliší od předchozího scénáře. Zde ovšem dochází ke změně. Analýzou VRAM záznamu zjistíme, že zobrazovaný pixel je součástí prázdného pole šachovnice. Informuje nás o tom 0. bit VRAM záznamu hodnotou 1. Můžeme tedy přejít k nastavení barvy pixelu, protože pro prázdné pole nepotřebujeme načíst data textury. Z 2. – 4. bitu VRAM záznamu je vyjmuta informace o barvě a ta je převedena na potřebnou 9 bitovou hodnotu.

3) Zobrazení aktivního pole

Pole slouží k určení aktivní pozice na šachovnici. Při nastavování barvy pixelu pro VGA řadič je kontrolován nejvyšší bit VRAM záznamu. Má-li bit hodnotu 1, pak tato hodnota je příznakem aktivního pole. Je-li pixel adresovaný VGA řadičem současní označení aktivního pole, pak je namísto barvy pozadí pole zobrazena (červená) barva označující aktivní pole.

4) Zobrazení orámování šachovnice

Princip zobrazení orámování šachovnice vychází z principu zobrazení šachového pole s figurou. Proto již nebudeme popisovat veškeré detaily a budeme se soustředit pouze na významné odlišnosti. Proces `comp_vram_addr` automaticky nastaví signál pro pozadí orámování šachovnice (`sig_border_bg`), pokud je pixel součástí orámování. Na základě signálů `board_row` a `board_col` proces `comp_texture_addr` rozpozná, zda-li je potřeba zobrazit texturu orámování a případně patřičně nastaví signály poukazující na nutnost zobrazení textury (`sig_border_h`, `sig_border_v`) a adresační signály v rámci textury.

Bylo by dobré poznamenat, že adresa pro VRAM je v případě horizontálního orámování (signalizováno signálem `sig_border_h`) složena konkatenací offsetu (čísla 8) a signálu `board_col`. V případě vertikálního orámování (signalizováno signálem `sig_border_v`) je adresa složena konkatenací offsetu (čísla 9) se signálem `board_row`.

Uveďme si jako příklad zobrazení pixelu z textury pro znak D. Bázová adresa textury s naším pixelem je uložena na spodních čtyřech bitech VRAM záznamu. Připomeňme si, že textury pro orámování začínají na adrese 10976. Adresu pixelu získáme podle vzorce 4.4. Princip nastavení barvy pixelu je obdobný jako u pole s figurou.

$$addr = addr_start + base_addr_tex * tex_size + addr_pix \quad (4.4)$$

addr - adresa pixelu v TEXRAM
addr_start - počáteční adresa textur
base_addr_tex - bázová adresa textury
tex_size - velikost textury (128 bitů)
addr_pix - adresa pixelu v rámci textury

5) Zobrazení figur pro výměnu za pěšce

Nijak výrazně se neliší od zobrazení šachového pole s figurou. Hlavní odlišností je adresace VRAM záznamu. Adresa se skládá z offsetu (čísla 10) a signálu `board_row`, který určuje řádek jedné ze čtyř figur. Další odlišností je význam 0. bitu VRAM záznamu, ten zde plní účel příznaku pro zobrazení figur.

4.2 Implementace MCU

Kód pro MCU je implementován v jazyce C s využitím prostředků knihovny libfitkit. Tato část práce se věnuje popisu implementace principů uvedených v předchozí části. Hlavní funkce programu obsahuje inicializaci hardwaru, datových struktur, VRAM paměti, TEXRAM paměti a hlavní smyčku programu. Hlavní smyčka programu zajišťuje obsluhu terminálu a klávesnice.

4.2.1 Terminál

Terminál je podstatnou částí programu, která zajišťuje komunikaci uživatele s aplikací. Na straně aplikace jsou příkazy zpracovány funkcí `decode_user_cmd()`. Terminál akceptuje následující příkazy:
`FLASH W TEX` - uloží textury ze souboru do flash paměti
`UPDATE TEX` - přenese textury z flash do BRAM
`NEW GAME` - spustí novou hru
`SAVE GAME` - uloží hru
`LOAD GAME` - načte hru
`DAMA` - vymění pěšce za dámu
`VEZ` - vymění pěšce za věž
`STRELEC` - vymění pěšce za střelce
`JEZDEC` - vymění pěšce za jezdce

4.2.2 Ovládání hry

K ovládání hry je použita klávesnice na FITkitu. Obsluhu řadiče klávesnice zajišťuje funkce `keyboard_idle()`. Tato funkce obsluhu zmáčknutého tlačítka předává funkci `serviceKey()`, která již zajistí správnou interpretaci příkazu.

4.2.3 Jádro hry

Jádro hry tvoří funkce `serviceKey()`, která zpracovává příkazy z klávesnice a patřičně na ně reaguje. Posun aktivního pole po šachovnici probíhá na základě směrových kláves (tlačítka 1 – 9, kromě tlačítka 5). Před přesunem aktivního pole jsou kontrolovány meze

šachovnice. Přesun je zaznamenán do struktury `boardState` a zanesen do VRAM záznamů, čímž se zajistí překreslení aktivního pole na zobrazovacím zařízení.

Tah figurou se provádí pomoci akčního tlačítka (tlačítko '5'). Nejdříve je třeba nastavit aktivní pole na pole s figurou, se kterou chceme táhnout. Figuru zvedneme akčním tlačítkem, na pole kde chceme figuru položit přesuneme aktivní pole a figuru na toto pole položíme akčním tlačítkem. Pokud jsme zvedli figuru, ale tah jsme si rozmysleli, je figuru možné položit nastavením aktivního pole na zvednutou figuru a stisknutím akčního tlačítka.

Při zvednutí figury hra pro figuru generuje možné tahy, které na šachovnici označují zelená pole. Generátor tahů je podrobněji popsán v kapitole 4.2.5. Při pokládání figury hra kontroluje, zda je požadovaný tah pro figuru přípustný (viz. kapitola 4.2.4). Nutností je také dodržet, že tahem se král nedostane do šachu nebo měl-li král šach, tímto tahem byl šach zrušen. Podrobnosti o kontrole na šach lze nalézt v kapitole 4.2.6. Po provedení tahu hra kontroluje konec hry (kapitola 4.2.7).

4.2.4 Kontrola tahu

Kontrolu tahů kamenů provádí funkce `checkMove()`. Nutnou podmínkou korektního tahu je, že na cílovém poli nesmí být moje figura. Tuto podmínku zajišťuje již funkce `serviceKey()`.

Pro kontrolu tahu věže byly implementovány funkce `horizontalCheck()` a `vertikalCheck()`, které v závislosti na požadovaném horizontálním nebo vertikálním tahu věží kontrolují jestli mezi počátečním a cílovým polem není žádná figura. Pro kontrolu tahu střelce je implementována funkce `crossCheck()`. Funkce kontroluje diagonální přesun figury a pracuje na stejném principu jako funkce pro kontrolu tahu věže. Při kontrole tahu dámy jsou použity funkce pro kontrolu tahu věže a střelce.

U tahu jezdcem je kontrolován horizontální a vertikální rozdíl pozic polí. Podmínkou korektního tahu je, aby jeden rozdíl pozic byl 2 a druhý 1. Táhneme-li jezdcem z B8 na C6 jedná se o korektní tah, neboť horizontální rozdíl pozic je 2 a vertikální 1. Táhneme-li pěšcem, je kromě běžných tahů implementována i kontrola pro tah nazývaný braní mimochodem. Ke kontrole tahu králem je určena funkce kontrolující šach pro zadané pole, více je popsáno v kapitole 4.2.6.

4.2.5 Generátor možných tahů

Generování možných tahů provádí funkce `markPosibleMove()`, která má jisté podobnosti s funkcí pro kontrolu tahu. Obě funkce se ovšem liší v jednom zásadním principu. Generátor tahů generuje pro figuru všechny teoreticky možné tahy, nebere v úvahu šach krále. Kdežto kontrola tahu provádí kontrolu skutečné proveditelnosti tahu. Tahy pro pěšce jsou generovány s využitím funkce pro kontrolu tahu. Při generování tahů pro krále generátor využívá funkci kontrolující tahy k generování tahu nazývaného rošáda. Pro jezdce jsou podle jednoduchých pravidel ověřeny a vyznačeny platné tahy.

Pro efektivní generování tahů ostatních figur, nebyla použita funkce pro kontrolu tahu, ale byly implementovány funkce `horizontalMark()`, `verticalMark()` a `crossMark()`. Funkce prochází řádek, sloupec, respektive diagonálu dokud nenarazí na figuru nebo mez šachovnice. Generátor tahů zajišťuje generování tahů pro věž použitím dvou prvně jmenovaných funkcí. Tahy pro střelce jsou generovány poslední z jmenovaných funkcí. Ke generování tahů pro dámu je použito všech tří funkcí.

4.2.6 Kontrola šach

Funkce `checkSachForPoz()` zjišťuje zda král na daném poli má šach nebo by ho měl, kdyby na něj táhnul. Algoritmus je založen na organizovaném prohledávání šachovnice, při němž se snažíme nalézt figuru soupeře která ohrožuje ověřované pole. Nenajdeme-li žádnou protivníkovu figuru, která se může přesunout na námi ověřované pole, můžeme prohlásit, že na tomto poli král nemá šach.

Podle tohoto principu provádíme kontrolu figur na řádku, sloupci a diagonálách ověřovaného pole. Řádky a sloupce ověřujeme na možný útok soupeřovou věží nebo dámou. Útok z diagonál může přijít od střelce nebo dámy. Dalším krokem je kontrola všech polí, z nichž by na krále mohl zaútočit jezdec či pěšec. Poslední kontrolovanou figurou je protivníkův král, ke kterému se svým tahem nesmím přiblížit do těsné blízkosti.

4.2.7 Kontrola mat, pat

Kontrolu matu a patu provádí funkce `checkMatPat()`. Pozice krále na šachovnici je určena pomocí funkce `findPozKral()`. Při kontrole matu je nejdříve ověřeno, jestli má král šach. Pokud je král ohrožován, hra zjišťuje, zda může ustoupit na některé z vedlejších polí. Nastane-li situace, že král má šach a nemůže ustoupit, funkce `checkMoveOtherFigure()` ověří možnost představit do cesty šachující figury jinou ze svých figur. Další ověřovanou možností je možnost vzít ohrožující figuru některou ze svých figur.

Představme si, že se šachová partie dostala do stavu, kdy král nemá šach, ale nemůže táhnout na žádné z vedlejších polí. V takovém případě program s využitím funkce `checkPat()` ověřuje, zda-li hra nezkončila patem. Tato funkce na šachovnici hledá jinou vhodnou figuru, která může táhnout aniž bych tahem ohrozila krále. Pokud již na šachovnici není jiná figura, která by mohla táhnout, hra končí patem.

4.2.8 VRAM

Video RAM paměť je řídícím prvkem zobrazovací jednotky. Implementací návrhu jednotky jsme docílili efektivity a flexibility zobrazovaného grafického výstupu. Změny grafického výstupu jako například přesun figury jsou realizovány téměř v nulovém čase.

Nízkoúrovňová funkce `setFigure()` je určena k nastavení pole s figurou a společně s funkcí `setEmptyPole()`, která slouží k nastavení prázdného pole tvoří logický most mezi VRAM pamětí a funkcí vyšší úrovně. Díky těmto funkcím je jednoduše možné přesunem figur na šachovnici (v proměnné `chessBoard[][]`) realizovat změnu grafického výstupu. K tomuto účelu slouží funkce vyšší úrovně `vRamSetFigure()`, která zajišťuje nastavení VRAM záznamu dle pole šachovnice.

Hlavní myšlenkou při návrhu zobrazovací jednotky a jejího ovládání bylo, že veškerou logiku a potřebné informace bude obsahovat program pro MCU. Tímto jsem chtěl docílit pouze přenosu dat z MCU do VRAM. Implementace návrhu však ukázala, že dodržením tohoto principu by došlo ke zvětšení paměťové náročnosti a složitosti programu. Důvodem pro zavedení čtení dat z VRAM byla potřeba získat informaci o barvě pole. Generátor možných tahů označuje možné tahy figury zelenou barvou pole. Ukládání pozic možných tahů by vedlo k dalším požadavkům na paměť a proto jsou možné tahy zobrazeny přímo v grafickém výstupu. V rámci programu není třeba časté čtení dat z paměti VRAM ani čtení velkého bloku dat a proto se nemusíme obávat zpomalení aplikace.

4.2.9 Využití Flash paměťi

Důležité parametry pro práci s Flash pamětí jsou shrnuty na obr. 4.1. Kapacita 528 kB je rozdělena do 2048 stránek po 264 B. Pro uživatelská data je k dispozici 1833 stránek, zbytek stránek je určen k uložení konfigurace FPGA.

```
FLASH INFO
  kapacita flash [kB]:    528
  velikost stranky [B]:   264
  velikost bufferu [B]:   264
  pocet stranek:          2048
  pocet bloku:            256
  konfigurace FPGA [pg]: 208 (8 - 215)
  uzivatelska data [pg]: 1833 (216 - 2047)
```

Obrázek 4.1: Informace o Flash paměťi

Flash paměť je v aplikaci použita pro uložení textur a hry. Tabulka 4.6 ukazuje uložení dat v paměti.

První stránka	Počet stránek	Použití
216	1	uložení hry
217	50	uložení textur

Tabulka 4.6: Uložení dat ve Flash paměti

4.2.10 TEXRAM

Tato paměť poskytuje textury figur, čísel a znaků. Uložení dat do paměťi probíhá ve dvou krocích. V prvním kroku jsou po zadání příkazu `FLASH W TEX` v terminálu data uložena ze souboru do Flash paměti. Ve Flash paměti jsou k uložení textur vyhrazeny stránky 217 – 266, celkem tedy 50 stránek. Počet potřebných stránek k uložení dat je možné vypočítat podle rovnice 4.5. Druhým krokem je přesunutí dat z Flash paměti do TEXRAM zadáním příkazu `UPDATE TEX` do terminálu. Tento krok zajišťuje funkce `initTexRam()`. Součástí funkce je jednoduchá kontola dat, při které je na začátku dat očekáván můj školní login (řetězec `XKUBIN16`). Odpojení FITkitu od terminálu způsobí ztrátu dat v TEXRAM paměti. Tento nepříjemný efekt způsobuje, že při každém spuštění aplikace je potřeba znovu uložit data do TEXRAM paměti. V aplikaci je tento problém řešen automatickým updatem TEXRAM paměti, při kterém je použita již zmíněná kontrola dat.

$$pocet_stranek \doteq velikost_dat \;/\; velikost_stranky \tag{4.5}$$

4.2.11 Uložení hry

K uložení hry ve Flash paměti je použita stránka 216. Hra ukládá 64 polí šachovnice (64 B), informaci o tom, který z hráčů je na tahu (1 B). Pro bílého i černého je ukládána pozici aktivního pole (2 B, dohromady 4 B) a informace jestli již hráč táhnul králem a věžemi

(3 B, celkově 6 B). Celkem je k uložení hry tedy použito 75 z 264 B. Hru není možné uložit v těchto případech:

- **Hráč drží figuru** - důvodem tohoto omezení je, že by bylo nutné navíc ukládat pozici zvednuté figury. Především by ale nastal problém při načítání hry. Bylo by problematické zajistit správný stav hry a vygenerování možných tahů. Hru je možné uložit po provedení tahu nebo po položení figury.
- **Výměna pěšce za figuru** - hru je možné uložit až po výměně pěšce za požadovanou figuru. Ukládání a načítání hry v tomto mezistavu by bylo problematické.
- **Konec hry** - po ukončení hry již hru není možné uložit. Uložení a načtení hry by bylo složité.

Kapitola 5

Závěr

Cílem bakalářské práce bylo vytvořit hru šachy pro platformu FITkit. Součástí zadání bylo navrhnout jednotku pro zobrazování šachovnice na VGA monitoru a generátor možných tahů. A dále potom provést jejich implementaci. Tyto cíle byly splněny. Realizace zobrazovací jednotky byla provedena v FPGA a generátor možných tahů byl implementován v MCU.

Nad rámec zadání byly implementovány funkce kontrolující šach a konec hry. Dalším rozšířením je možnost uložení a načtení hry.

Jednou z možností rozšíření této práce by byla možnost ukládání a načítání hry z počítače. K tomu by bylo potřeba modifikovat funkce pro uložení a načtení hry z Flash. Hlavní komplikace je však na straně QDevKitu, který neumožňuje ukládání dat z Flash paměti do souboru. Bylo by tak potřeba implementovat plugin nebo potřebnou funkci pro QDevKit, která by to umožnila. Načítání hry ze souboru by umožnilo řešení šachových úloh. Po zprovoznění ukládání dat z flash do souboru by bylo možné ukládat průběh hry pomocí šachové notace.

Pro vylepšení grafického výstupu hry by bylo možné použít místo jedné BRAM tři paralelně zapojené BRAM. Tím bychom pro informaci o pixelu získali místo 1 bitu 3 bity, které by umožňovaly uložit až osm barev. Druhou možností by bylo k uložení textur použít paměť SDRAM, pro tu by bylo však nezbytné napsat řadič.

Pro větší komfort hráčů by hru bylo možné rozdělit na dva FITkity a ty propojit. Každý z hráčů by tak měl vlastní FITkit s monitorem, na kterém by hrál. Pravděpodobně nejzajímavějším a implementačně nějnáročnějším by bylo hru opatřit umělou inteligencí. Všechna tato výše uvedená vylepšení jsou časově i pracovně značně náročná a mohla by být předmětem další práce.

Literatura

[1] Alster, L.: *Šachy – hra královská*. Praha: Práce, 1987, 296 s.

[2] Fucik, O.; Bardas, R.: FITkit verze 1.0. online, 19. října 2008 [cit. 2012-03-29]. URL http://merlin.fit.vutbr.cz/FITkit/download/schematic_v10.pdf

[3] Gižycki, J.: *Šachy všech dob a zemí*. Praha: Práce, 1975, 480 s.

[4] Halba: Stručný popis jazyku VHDL. online, Rev. 17.5.2002. URL www.hepin.cz/storage/1245608020_sb_vhdl.pdf

[5] Herejk, P.: Základní šachový výcvik začátečníků. online, 2009 [cit. 2012-03-29]. URL http://www.chess.cz/www/mladez/metodika/zakladni-sachovy-vycvik.html

[6] Hwang, E. O.: *Digital Logic and Microprocessor Design with VHDL*. Toronto, Canada: Nelson, 2006, ISBN 0-534-46593-5, 588 s.

[7] Matušov, I.: Hra Tetris - FITkit. online, Rev. 31.1.2011 [cit. 2012-03-29]. URL http://merlin.fit.vutbr.cz/FITkit/docs/aplikace/apps_games_tetris.html

[8] Padroni, V. A.: *Circuit Design with VHDL*. London, England: MIT Press, 2004, ISBN 978-0-262-16224-1, 363 s.

[9] Pech, J.: Nebojte se FPGA. online, 2002. URL http://www.hw.cz/teorie-a-praxe/dokumentace/nebojte-se-fpga.html

[10] Šachový svaz České republiky: Pravidla šachu FIDE. online, 2009. URL http://www.chess.cz/www/assets/files/informace/legislativa/PravidlaSachuFIDE2009.pdf

[11] Sekanina, L.: Úvod do jazyka VHDL. online, Rev. 15.10.2007. URL https://www.fit.vutbr.cz/study/courses/INP/private/cvic/inp_vhdl_opora.pdf

[12] Slaný, K.; Markovič, J.: Klávesnice 4x4 - FITkit. online, Rev. 19.3.2009 [cit. 2012-03-29]. URL http://merlin.fit.vutbr.cz/FITkit/docs/firmware/fpga_keyboard.html

[13] Texas_Instruments: MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller. online, 2002, Updated 2-March-2011. URL http://www.ti.com/lit/ds/symlink/msp430f168.pdf

[14] Texas_Instruments: MSP430F241x, MSP430F261x Mixed Signal Microcontroller. online, 2007, Updated 12-December-2011.
URL http://www.ti.com/lit/ds/symlink/msp430f2617.pdf

[15] Vašíček, Z.: Aplikace - FITkit. online, [cit. 2012-03-29].
URL http://merlin.fit.vutbr.cz/FITkit/aplikace.html

[16] Vašíček, Z.: Hardware - FITkit. online, [cit. 2012-03-29].
URL http://merlin.fit.vutbr.cz/FITkit/hardware.html

[17] Vašíček, Z.: Programování a bootování FITkitu - FITkit. online, [cit. 2012-03-29].
URL http://merlin.fit.vutbr.cz/FITkit/docs/hardware/hw_boot.html

[18] Vašíček, Z.: QDevKit - Windows - FITkit. online, [cit. 2012-03-29].
URL http://merlin.fit.vutbr.cz/FITkit/docs/navody/qdevkit.html

[19] Vašíček, Z.: Úvod - FITkit. online, [cit. 2012-03-29].
URL http://merlin.fit.vutbr.cz/FITkit/uvod.html

[20] Vašíček, Z.: Komunikační systém - FITkit. online, Rev. 17.9.2009 [cit. 2012-03-29].
URL
http://merlin.fit.vutbr.cz/FITkit/docs/firmware/fpga_interconect.html

[21] Vašíček, Z.: Textový režim - FITkit. online, Rev. 26.3.2009 [cit. 2012-03-29].
URL
http://merlin.fit.vutbr.cz/FITkit/docs/aplikace/apps_vga_textmode.html

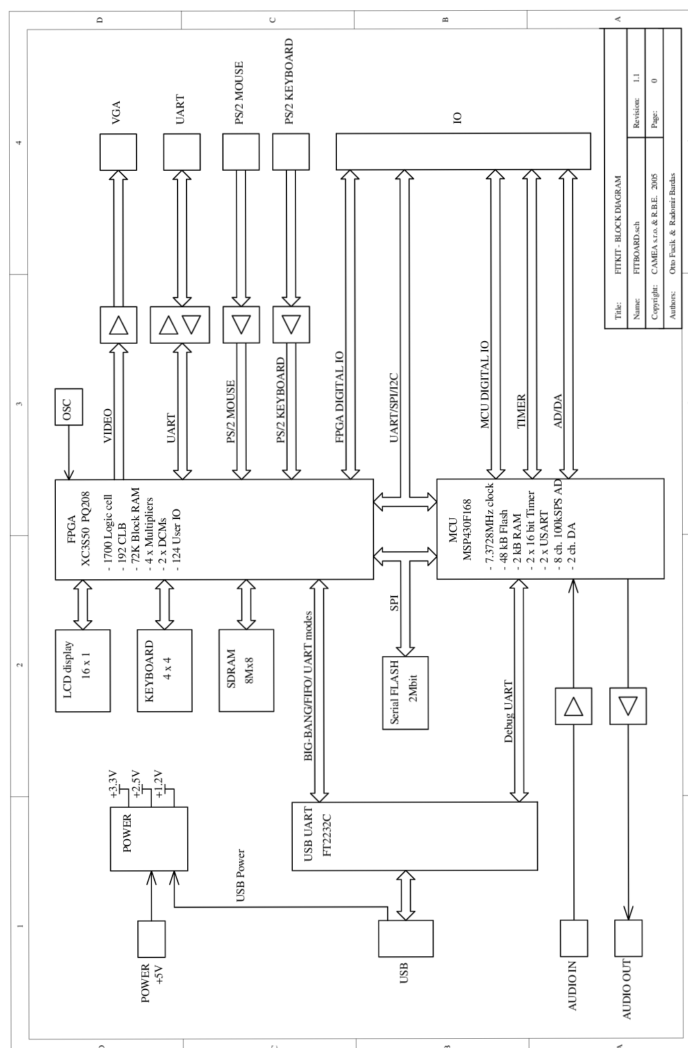
[22] Wikipedie: Dějiny šachové hry. online, Rev. 8.5.2012 [cit. 2012-03-29].
URL http://cs.wikipedia.org/wiki/D%C4%9Bjiny_%C5%A1achov%C3%A9_hry

[23] Xilinx: Spartan-3 FPGA Family Data Sheet. online, 2003, Updated 4-December-2009.
URL http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf

[24] Xilinx: Spartan-3 Generation FPGA User Guide. online, 2006, Updated 13-June-2011.
URL http://www.xilinx.com/support/documentation/user_guides/ug331.pdf

Příloha A

Blokový diagram FITkitu



Obrázek A.1: FITkit - blokový diagram [2]

Příloha B

Obsah CD

Přiložené CD obsahuje následující složky:

- **src** - obsahuje zdrojové kódy projektu pro FITkit
- **technicka zprava** - adresář obsahuje tuto práci ve formátu *pdf* a dále pak zdrojové kódy v LaTeXu pro opětovné sestavení zprávy.