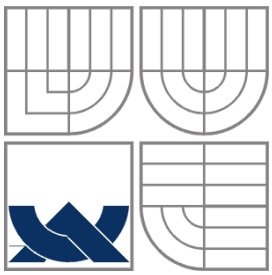
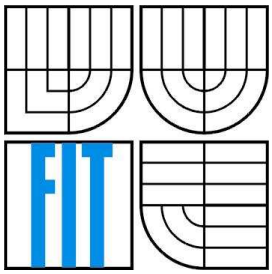


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

IMPLEMENTACE CELL MATRIX V FPGA

AN IMPLEMENTATION OF CELL MATRIX IN FPGA

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — JAN MARTINÁK
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Lukáš Sekanina, Ph.D.
SUPERVISOR

BRNO 2007

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Jan Martinák**
Id studenta: 84251
Bytem: Na Valtické 666/36, 691 41 Břeclav
Narozen: 19. 10. 1984, Valtice
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Implementace CellMatrix v FPGA
Vedoucí/školitel VŠKP: Sekanina Lukáš, doc. Ing., Ph.D.
Ústav: Ústav počítačových systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	..... Maršík Jan .....
Nabyvatel	Autor

Zadání bakalářské práce/4922/2006/xmarti33

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2006/20

Zadání bakalářské práce

Řešitel: **Martinák Jan**
Obor: Informační technologie
Téma: **Implementace CellMatrix v FPGA**
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s platformou CellMatrix (http://www.cellmatrix.com/).
2. Seznamte se s FitKitem.
3. Popište buňku CellMatrix a skupinu buněk CellMatrix ve VHDL.
4. Realizujte CellMatrix ve výukovém kitu FITkit.
5. Navrhněte a realizujte vhodné uživatelské rozhraní.
6. Shrňte dosažené výsledky.

Literatura:

- www.cellmatrix.com

Při obhajobě semestrální části projektu je požadováno:

- Bod 1 a 2.

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

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% 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 paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Sekanina Lukáš, doc. Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

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

Hardware s možností rekonfigurace představuje moderní trend ve vývoji nových obvodů. Stále rostoucí požadavky na takovou architekturu vyústily ve snahy vytvořit obvod, který dokáže rekonfiguraci provádět paralelně a lokálně. Jednou z takových technologií je architektura Cell Matrix, založená na principech celulárních automatů. Tato práce si klade za úkol seznámit čtenáře s architekturou Cell Matrix a ukázat její výhody, funkce a možnosti implementací v programovatelném poli FPGA na vývojovém kitu FITkit.

Klíčová slova

Cell Matrix, celulární automaty, rekonfigurace, FPGA, FITkit.

Abstract

Reconfigurable hardware architectures represent a modern trend in development of new circuits. Growing demands on such architectures have lead to creating a circuit, that is able to reconfigure itself in parallel and locally. One of such technologies is the Cell Matrix which is based on principles of cellular automata. The purpose of this thesis is to describe this Cell Matrix architecture. It also shows its advantages, functions and potential when implemented in the FPGA on FITkit development kit.

Keywords

Cell Matrix, cellular automata, reconfiguration, FPGA, FITkit.

Citace

Jan Martinák: Implementace Cell Matrix v FPGA, bakalářská práce, Brno, FIT VUT v Brně, 2007

Implementace Cell Matrix v FPGA

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doc. Ing. Lukáše Sekaniny, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jan Martinák
19.4.2007

Poděkování

Chtěl bych poděkovat doc. Ing. Sekaninovi, Ph.D. za užitečné rady při řešení projektu a odbornou pomoc při vypracování této bakalářské práce.

Obsah

Úvod

Termínem hardware s možností rekonfigurace máme na mysli zařízení s pevnou fyzickou strukturou, ale s možností změny vykonávané funkce, která je prováděna až v době po ukončení výrobního procesu [1].

Myšlenka rekonfigurovatelného hardware není zdaleka tak nová, jak se může zdát. Sahá až do 60. let minulého století, kdy byly nastíněny principy počítače s programovatelným polem a procesorem řídícím jeho rekonfiguraci [2]. Tehdy ovšem byl takový nápad daleko za technologickými možnostmi té doby. Velký rozmach této myšlenky byl zaznamenán až v posledním desetiletí spolu s radikálním rozvojem křemíkové technologie, který umožňoval umístnit komplexní architekturu na jeden čip.

Velká část práce v oblasti programovatelných obvodů se dnes soustřeďuje především ve vývoji FPGA (Field Programmable Gate Arrays). Stejně jako ostatní zařízení tohoto typu se FPGA skládá z homogenních funkčních jednotek, které jsou uspořádány do pravidelné struktury (např. matice), jejíž rekonfigurace je řízena externí jednotkou. Tento přístup však naráží na jisté limity. Nejvýznamnějším omezením je obtížná rozšiřitelnost a časová náročnost rekonfigurace, která je závislá na velikosti struktury.

Platforma Cell Matrix je do jisté míry podobná FPGA, nesdílí s ní však omezení popsaná výše. Hlavním důvodem je, že obvod provádějící rekonfiguraci je složen ze stejných logických jednotek jako obvod, který je rekonfigurován. Díky této vlastnosti je Cell Matrix do jisté míry nezávislá na svém okolí, neboť může ovlivňovat svou vlastní funkci. Cell Matrix je tedy architektura nejen se schopností rekonfigurace, ale také se schopností seberekonfigurace. Neméně důležitým důsledkem je, že veškeré operace uvnitř Cell Matrix mohou probíhat paralelně, což platí jak pro rekonfiguraci, tak pro zpracování dat.

Cíl práce

Mým úkolem a cílem této práce je prezentovat Cell Matrix jakou možnou alternativu ke stávajícím zařízením umožňujícím rekonfiguraci, označit její výhody a nevýhody, popsat vlastnosti a schopnosti, a hlavně provést její implementaci. Ta je realizována popisem buňky a matice buněk Cell Matrix v jazyce VHDL a syntézou do programovatelného pole FPGA řady Spartan-3 od firmy Xilinx. Tento čip je umístněn na vývojovém kitu FITkit, který vznikl na Fakultě informačních technologií VUT jako výsledek projektu „Zvýšení konkurenceschopnosti IT odborníků - absolventů pro Evropský trh práce“. Nakonec bylo mým úkolem navrhnout a vytvořit vhodné uživatelské rozhraní pro použití a experimentování s Cell Matrix.

Struktura práce

V následující kapitole rozeberu strukturu, principy a obecné vlastnosti Cell Matrix, následovat bude kapitola, která se věnuje vývojovému kitu FITkit, specielně těm komponentám, které byly využity v rámci tohoto projektu. Třetí kapitola se bude zabývat vlastní implementací Cell Matrix, realizací principů na úrovni programovacího jazyka VHDL a syntézou za použití příslušných nástrojů. V předposlední kapitole jsou prezentovány některé jednoduché obvody vytvořené konfigurací Cell Matrix. Poslední kapitola zhodnocuje dosažené výsledky, zvažuje perspektivy v použití Cell Matrix v praxi a další oblasti práce na podobném projektu.

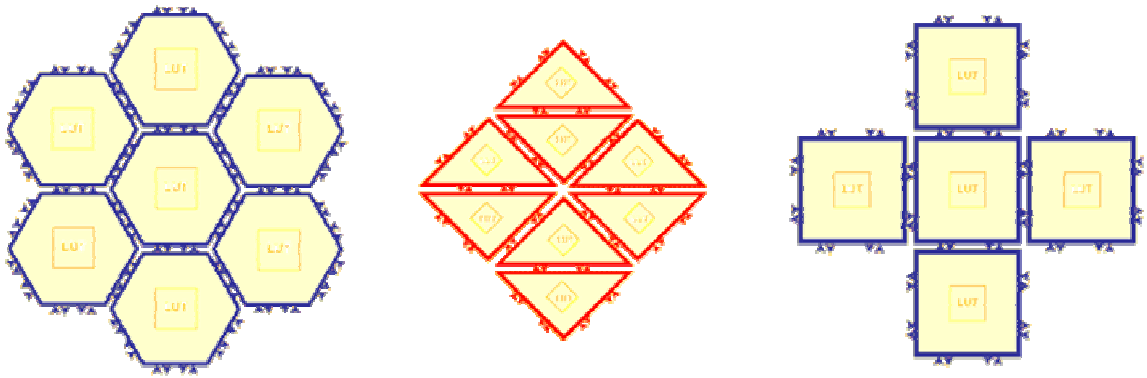
1 Cell Matrix

Cell Matrix je výpočetní architektura, která je založená na celulárním výpočetním modelu a vychází z principů celulárních automatů [3].

Celulární automat je systém, tvořený výhradně tzv. buňkami (kapitola *Celulární automaty* v [1]). Každá buňka v celulárním systému má v určitém diskrétním okamžiku určitý stav, přičemž stav celého automatu je dán stavy jednotlivých buněk. Jednotlivé buňky přecházejí do následujícího stavu v diskrétních krocích v závislosti na předchozích hodnotách buněk sousedních. Tak jako v celulárním automatu se ve struktuře Cell Matrix dosahuje specifického chování interakcí sousedních buněk.

1.1 Struktura Cell Matrix

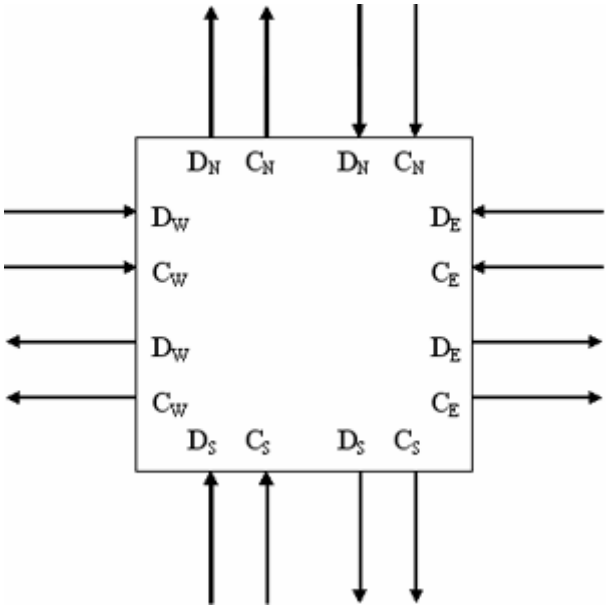
Strukturu Cell Matrix představuje vícerozměrná mřížka propojených buněk. Nejčastějším typem, se kterým se budeme setkávat při experimentování, jsou buňky čtvercové, uspořádané do dvourozměrné matice. Teoreticky však mohou mít různé typy sousedství - tedy různý tvar (obrázek 1.1), a je možné je uspořádat až do třírozměrných struktur.



Obr. 1.1 Různé topologie Cell Matrix (převzat z [4])

V dalším textu uvažujeme čtyrstrannou buňku, která byla implementována v rámci tohoto projektu.

Ke komunikaci se svým okolím slouží buňce dva typy signálů: tzv. C a D signály. Čtyři strany buňky názvem odpovídají příslušným světovým stranám: severní, jižní, západní a východní (anglicky North, South, West, East). Každé straně buňky přísluší dvojice signálů C a D, jedna vstupní a jedna výstupní tak, jak je vyznačeno na obrázku 1.2.



Obr. 1.2 Rozhraní čtyrstranné buňky Cell Matrix

Každá buňka v takovém systému vykonává svoji specifickou funkci. Tato funkce je v buňce uložena ve formě pravdivostní tabulky, jejíž velikost závisí na typu sousedství. Tabulka 1.1 ukazuje případ pro čtyrstrannou buňku, která plní funkci 1bitové úplné sčítačky. V závislosti na hodnotách vstupních signálů D a C může buňka pracovat ve dvou módech.

Vstupy				**Výstupy**							
D_N	D_S	D_W	D_E	C_N	C_S	C_W	C_E	D_N	D_S	D_W	D_E
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	1	0	0
0	1	0	1	0	0	0	0	0	0	1	0
0	1	1	0	0	0	0	0	0	1	0	0
0	1	1	1	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	1	0	0
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	0	0	1	0
1	1	0	0	0	0	0	0	0	0	1	0
1	1	0	1	0	0	0	0	0	1	1	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	1	1	0

Tab. 1.1 Pravdivostní tabulka čtyrstranné buňky

1.2 D-mód

Obecně můžeme říci, že buňka se nachází v D-módu, pokud žádný ze vstupních signálů C není v logické 1. V tom případě funguje čistě jako kombinační logika a v závislosti na hodnotách vstupů D nastavuje hodnoty na osmi výstupních signálech C a D podle vestavěné pravdivostní tabulky. Jinými slovy, buňka plní Booleovu funkci definovanou pravdivostní tabulkou. Např. pro čtyřstrannou buňku platí, že tato funkce má čtyři parametry, jimiž jsou D vstupy z jednotlivých stran. Fyzicky jsou tyto vstupy použity jako adresa do paměti, v níž je uložena pravdivostní tabulka.

Jelikož může jedna buňka plnit jakoukoliv Booleovskou funkci, není problém realizovat jednoduché funkce jako NAND, XOR, ale i např. jednobitovou sčítačku. Pomocí množiny takových buněk pak lze sestavovat složitější obvody, např. čítače, klopné obvody atd. Originální buňka Cell Matrix pracuje v D-módu asynchronně.

1.3 C-mód

Protože je Cell Matrix strukturou schopnou rekonfigurace a specifické chování buňky není určeno při výrobě, ale pravdivostní tabulkou v paměti, musí existovat způsob, jak tuto pravdivostní tabulku měnit. Takovou schopnost má buňka, která se nachází v C-módu.

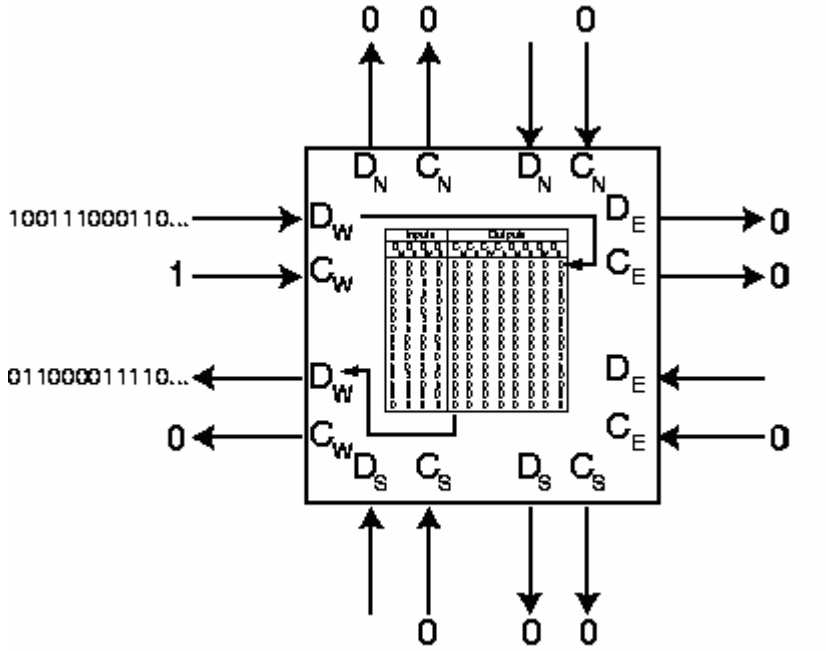
Změnu tabulky ovlivňují okolní buňky a to pouze ty, které poskytují aktivní signál C buňce v C-módu. Ta strana buňky v C-módu, na které je aktivní signál C, se nazývá aktivní stranou. Pokud je buňka v C-módu, pak její výstupy mají následující hodnoty:

- Všechny C výstupy mají hodnotu logická 0.
- Všechny D výstupy na neaktivních stranách mají hodnotu logická 0.
- Všechny D výstupy na aktivních stranách předávají hodnoty odpovídající hodnotě posledního bitu pravdivostní tabulky (ta je neustále posouvána, protože v C-módu pracuje paměť buňky jako posuvný registr).

Hodnota nového bitu, načítaná do pravdivostní tabulky, je rovna logickému součtu OR všech D-vstupů na aktivních stranách (z kapitoly *C-mód* v [5]).

Programování pravdivostní tabulky určují systémové hodiny. Během jednoho taktu je na první pozici posunut nový bit a z poslední pozice je bit poskytnut na vstupy buněk na aktivních stranách (obrázek 1.3). V tomto procesu jsou ve vzorových implementacích využity obě hrany hodinového signálu, nástupná hrana k posunu bitu do tabulky, sestupná pak k posunu celé pravdivostní tabulky a vystavení posledního bitu. V tomto projektu je však využita pouze nástupná hrana (více viz kapitola *Implementace Cell Matrix*).

Právě C-mód je unikátní vlastností Cell Matrix, kterou se mj. liší od celulárních automatů. Zajišťuje schopnost seberekonfigurace, neboť umožňuje definovat funkci buněk, které dokáží replikovat jiné buňky.



Obr. 1.3 Buňka při změně pravdivostní tabulky (převzat z [5] a upraven)

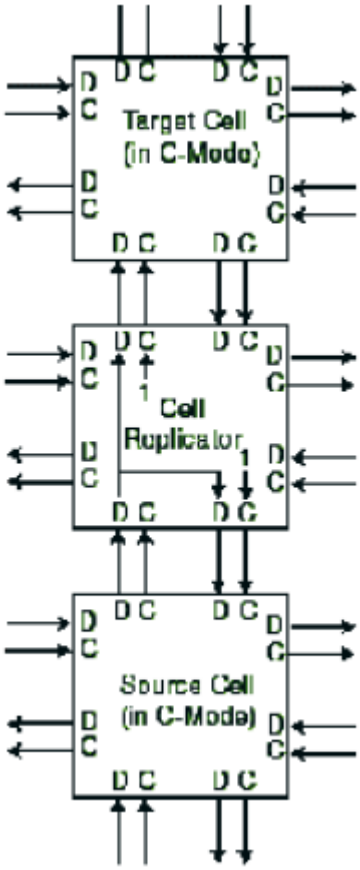
1.4 Obecné vlastnosti Cell Matrix

Cell Matrix disponuje vlastnostmi, které nejsou u moderních rekonfiguračních obvodů běžné. V následujícím souhrnu jsou tyto vlastnosti průběžně srovnávány s podobnou technologií FPGA. Informace v této kapitole jsou čerpány z [6].

1.4.1 Seberekonfigurace

Na rozdíl od FPGA můžeme v Cell Matrix vytvořit obvody, které nějakým způsobem tvoří, či mění jiné obvody resp. jiné části sebe samotného. Novější typy FPGA sice umožňují dynamickou rekonfiguraci [7], FPGA však není principielně na seberekonfiguraci založená.

Jediná buňka Cell Matrix sice může přímo ovlivňovat pouze své bezprostřední okolí, skupina buněk je však schopna díky vzájemné spolupráci nepřímo působit na jiné, neomezeně vzdálené, části obvodu. Způsob, jakým to provádí, ilustruje obrázek. 1.4.



Obr. 1.4. (převzat z [5])

Prostřední buňka na obrázku představuje tzv. replikátor. Replikátor čte pravdivostní tabulku zdrojové buňky (source cell) a zároveň ji nedestruktivně vrací zpět a kopíruje do cílové buňky (target cell). Provádí to tak, že zdrojovou i cílovou buňku uvede do C-módu a ve své pravdivostní tabulce zajistí patřičné propojení D vstupů a výstupů (viz obrázek 1.4.).

Důsledkem vlastnosti seberekonfigurovatelnosti je fakt, že můžeme v Cell Matrix tvořit komplexní obvody, které samy sebe analyzují a dokáží reagovat na případné poruchy, stejně jako učit se a vyvíjet v čase.

1.4.2 Lokální konfigurace

V Cell Matrix může být každá část matice konfigurována nezávisle na zbytku obvodu, na rozdíl od většiny typů FPGA, u kterých je nutné při změně části provést konfiguraci celé struktury znovu. Obvod, či alespoň jeho části, tak mohou být rekonfigurovány souběžně při prováděni své funkce, paralelní přístup tak celý proces výrazně urychluje.

1.4.3 Škálovatelnost

Tvorba složitějších a větších obvodů nepředstavuje pro Cell Matrix problém z hlediska rozšiřitelnosti. Jelikož jediné výpočetní a řídící zdroje jsou buňky, stačí pouze matici rozšířit o další buňky jejich připojením k okrajům stávající matice. Pokud bychom podobným způsobem chtěli rozšířit FPGA, narazíme přinejmenším na problém dvou různých adresových prostorů. Větší FPGA si také vyžadují komplexnější řídící obvody.

1.4.4 Levná výroba

Výroba struktur typu Cell Matrix je potenciálně levnější než výroba ASIC. Toto tvrzení je podloženo hlavně skutečností, že struktura tohoto hardware je jednoduchá, protože se sestává z identických buněk a specifické chování určují jejich paměti. Na druhou stranu jsou kladeny přísné požadavky na efektivitu návrhu buňky, kde hlavní problém spočívá v implementaci pravdivostní tabulky.

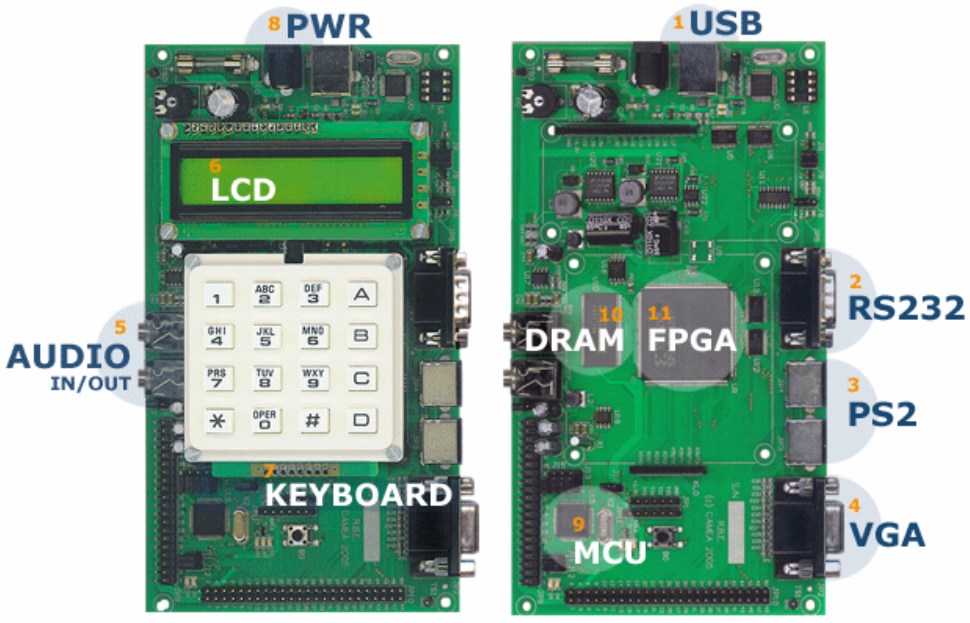
Příčinou relativně levné výroby je také existence určité tolerance při výrobě matic. Jak již bylo zmíněno, Cell Matrix se dokáže sama od sebe vyvíjet, a proto pro ní není problém izolovat ty části matice, které obsahují výrobní defekt.

Ačkoliv se může zdát, že Cell Matrix je dokonalá technologie budoucnosti, má i své nevýhody. Největší nevýhoda spočívá v uložení pravdivostní tabulky. Jelikož paměť, ve které je uložena, musí být schopna posunu obsahu, nabízí se implementace pomocí posuvného registru. Tento způsob je však neefektivní z hlediska velikosti výsledného obvodu buňky. Další nevýhodou je fakt, že buňka je správně přeprogramována pouze tehdy, pokud dojde k výměně celé pravdivostní tabulky, tzn. krátkodobé přerušení C-módu vede k použití nesprávně naprogramované buňky. O optimalizacích návrhu buňky je více popsáno v patentových dokumentech, zejména pak v [8].

2 FITkit

Vývojová platforma FITkit vznikla na Fakultě informačních technologií VUT za účelem zapojení vestavěných systémů do běžné výuky a zlepšení praktických schopnosti studentů při práci s hardware [9].

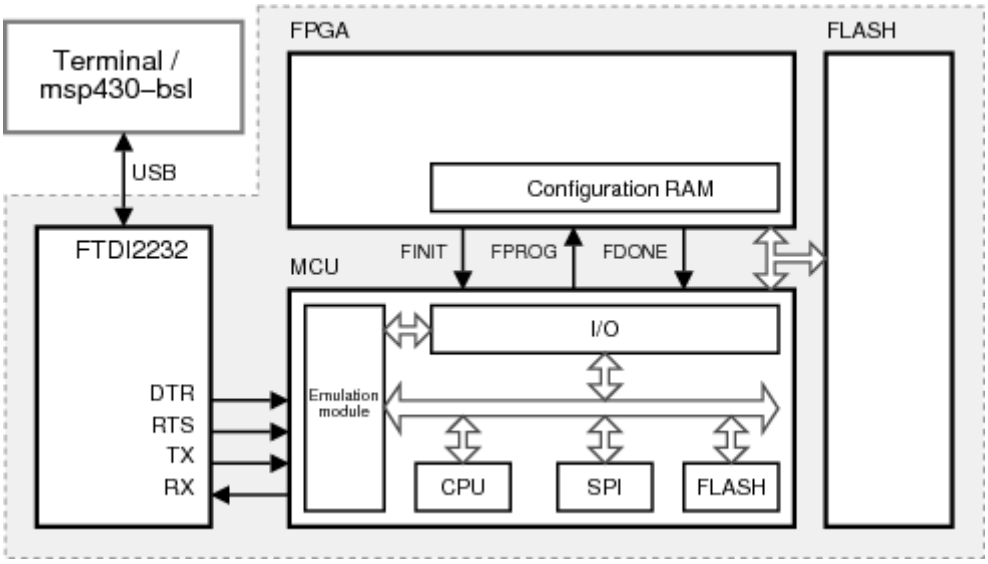
Obsahuje výkonný mikrokontrolér od firmy Texas Instruments a řadu periferií umístěných na desce. Nejvýznamnějším obvodem pro tento projekt je programovatelné hradlové pole (FPGA) řady Spartan-3 od firmy Xilinx, které bylo využito k implementaci Cell Matrix. Mezi další periferie patří např. paměť DRAM, klávesnice a řádkový displej. K dispozici je také řada rozhraní pro komunikaci s externími periferiemi, z nichž nejdůležitější je sériové rozhraní pro naprogramování mikrokontroléru z PC (řízeno USB-UART převodníkem), dále např. rozhraní VGA, RS232 atd. Podrobnější informace a schémata lze nalézt v [10]. Obrázek 2.1. zobrazuje pohledy na FITkit (vlevo pohled seshora, vpravo pohled na desku s plošnými spoji).



Obr. 2.1 Pohledy na FITkit (převzato z [10])

2.1 Princip systému bootování

FITkit disponuje dvěmi programovatelnými jednotkami a sice mikroprocesorem (MCU) řady MSP430 a již zmíněným hradlovým polem. Obě jednotky je možné programovat pomocí rozhraní JTAG, avšak FITkit nabízí i jednodušší variantu umožňující programovat obě zařízení přes USB kabel [11].



Obr. 2.2 Zjednodušené schéma FITkitu (převzato z [11])

Veškerá komunikace PC - FITKit je realizována sériovými protokoly. Komponenta FTDI2232 slouží k převodu mezi asynchronní komunikací podle standardu RS232, kterým komunikuje s MCU, a komunikací přes USB rozhraní.

Mikroprocesor se programuje pomocí programu *msp430-bsl*, který spolu se zavaděčem (*bootstrap loader* v mikroprocesoru) zavádí program, tedy posloupnost instrukcí vygenerovanou překladačem, do programové paměti FLASH.

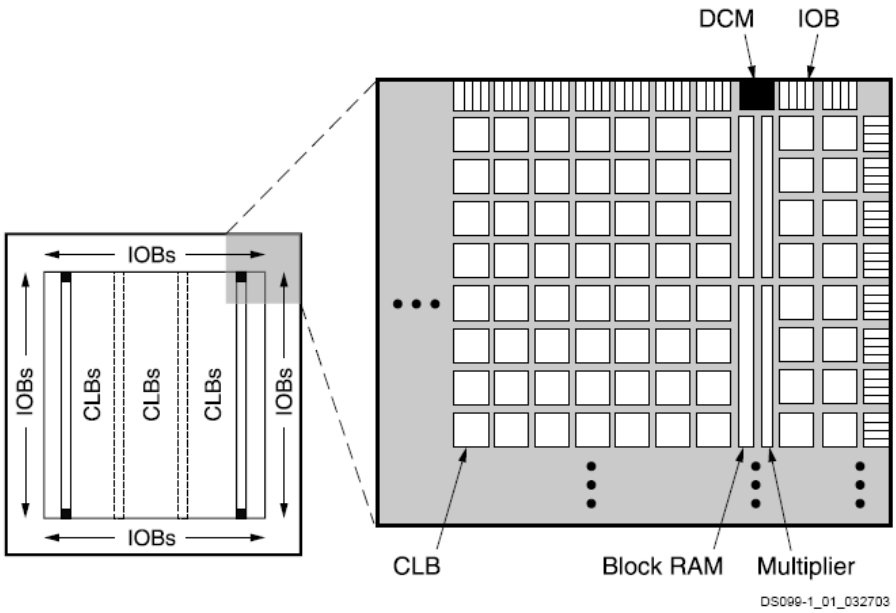
Programování FPGA provádíme v zásadě dvěma způsoby. První způsob počítá s využitím externí FLASH (kapacita 2Mbity), kam je uložena konfigurační informace pro FPGA. Zavádění této informace do FLASH je proces, ve kterém CPU přijímá konfigurační soubor z terminálu přenosovým protokolem *Xmodem*, a řadič SPI v MCU řídí její ukládání do externí FLASH přes vnitřní rozhraní SPI. Programování FPGA z FLASH proběhne vždy po resetu MCU, nebo zadáním příkazu v terminálu a provádí jej CPU za účasti SPI řadiče, který čte informace z FLASH. Druhý způsob se liší od prvního tím, že konfigurační data pro FPGA jsou po příjmu z terminálu rovnou použita k naprogramování FPGA.

Zásadní rozdíl mezi těmito dvěma způsoby je ten, že při použití prvního způsobu nemusíme po odpojení FITkitu od napájení znovu zasílat konfigurační soubor. Naprogramování se automaticky provede po startu a použijí se data v energeticky nezávislé FLASH paměti. Při ladění dizajnu našeho obvodu však postačí používat druhý způsob.

2.2 Struktura FPGA

Rodina FPGA Spartan-3 obsahuje pět programovatelných elementů, jejichž umístnění na čipu ilustruje obrázek 2.3:

- CLB (Configurable Logic Block). CLB představuje hlavní zdroj pro implementaci synchronních i kombinačních obvodů. Uvnitř CLB jsou dva páry jednotek (*Slice*). *Slices* obsahují generátory funkcí (look-up tabulky LUT), prvky k uchování dat (klopné obvody), sadu multiplexorů, které umožňují kombinovat ostatní prvky za účelem sestavení složitější logiky, a další vedlejší logiku.
- IOB (Input/Output Block). IOB je jednotka pro kontrolu toku dat mezi vývody pouzdra a vnitřní logikou. Podporuje obousměrný tok dat i třístavovou logiku včetně dalších vstupně výstupních standardů.
- Bloková RAM (BRAM). Její velikost se liší v závislosti na třídě FPGA.
- Násobičky 18 krát 18 bitů.
- DCM (Digital Clock Manager). Jednotka, která umožňuje distribuci hodinového signálu uvnitř FPGA.



Obr. 2.3 Struktura FPGA rodiny Spartan-3 (převzato z [12])

FPGA čip na FITkitu je zástupcem nejnižší třídy rodiny Spartan-3. Obsahuje 192 konfiguračních logických bloků, organizovaných do matice 16 krát 12, dále čtyři bloky BRAM o kapacitě 72 kilobitů, organizované do jednoho sloupce, čtyři násobičky a dvě jednotky DCM.

2.3 Knihovna libfitkit

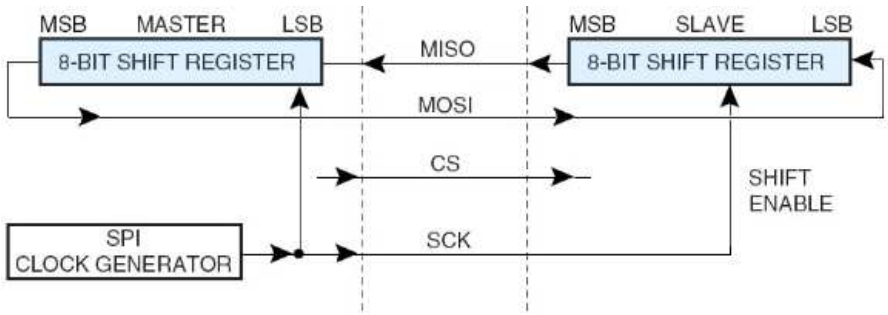
Knihovna *libfitkit* představuje operační jádro FITkitu. Je napsána v jazyce C a sdružuje základní funkce nezbytné pro správnou funkci mikrokontroléru. Poskytuje funkce pro správnou inicializaci komponent FITkitu, funkce pro komunikaci s terminálem na PC přes USB a UART MCU, funkce pro konfiguraci FPGA, pro komunikaci FPGA s externí FLASH pamětí přes vnitřní SPI rozhraní a další podpůrné funkce.

V knihovně je implementována komunikace přes sériové rozhraní, které je připojeno na port B čipu FT2232C (port A připojen přímo k FPGA). Nad sériovou komunikací je realizován jednoduchý příkazový interpret, do kterého lze zadávat příkazy pro MCU, které knihovna zpracuje. Pokud se jedná o příkaz knihovny, je jí vykonán. V opačném případě je zpracování ponecháno na programátorovi. Pro sériovou komunikaci s FITkitem přes USB kabel je nutné mít v operačním systému nainstalované ovladače pro čip FT2232C a správně nastavený terminál (ve Windows např. HyperTerminál).

Seznam s krátkým popisem některých funkcí užitečných při programování MCU je možné nalézt v dokumentu [13], ze kterého bylo také čerpáno v této kapitole. Pro podrobnější charakteristiku doporučuji hledat v příslušných dokumentacích k MCU (*datasheet*).

2.4 Vnitřní rozhraní SPI

Rozhraní SPI slouží na FITkitu jako propojovací systém komponent MCU, FPGA a externí FLASH paměti. Umožňuje obousměrnou komunikaci řízenou tzv. master (nadřazeným) zařízením s libovolným počtem připojených slave (podřízených) zařízení.



Obr. 2.4 Princip funkce SPI (převzato z [14])

Komunikace přes SPI spočívá ve výměně obsahů dvou osmibitových registrů. Přenos započne automaticky zapsáním osmibitové informace do posuvného registru master zařízení, které poté generuje signál CS, indikující zahájení transakce přechodem do logické 0. Master zařízení poté generuje hodinový signál SCK, jehož příslušná hrana určuje povolovací signál oběma registrům. Po

osmi taktech je obsah registrů vyměněn. Datové bity se přenášejí po signálech MISO (od slave zařízení do master zařízení) a MOSI (naopak) podle vzoru na obrázku 2.4.

Na FITkitu je master zařízení mikrokontrolér. SPI řadiče podřízených zařízení FLASH a FPGA sdílejí veškeré signály a rozlišení adresovaného zařízení se provádí na úrovni přenášených bitů v rámci operačního kódu.

Aby bylo možné přes SPI adresovat nejen FPGA, ale i zařízení uvnitř FPGA, byla vytvořena interní sériová sběrnice a protokol, který umožňuje komunikaci s těmito zařízeními. Komunikace probíhá v následujících posloupnostech:

1. **Odeslání operačního kódu**. Operační kód je osmice bitů, která obsahuje identifikaci slave zařízení, podle které rozpozná SPI řadič, že následující data jsou určena právě jemu. Hodnoty prvních dvou bitů operačního kódu určují, zda jde o operaci čtení nebo zápis.
2. **Odeslání adresy zařízení v FPGA**. Šířka adresy je parametr, který se nastavuje u SPI dekodérů jednotlivých zařízení.
3. **Odeslání dat**. Šířka datového bloku je parametr dekodéru stejně jako v případě adresy. Počet datových bloků je neomezen.

Základ vnitřní sběrnice tvoří jeden řadič SPI, který obsahuje 8bitový slave registr, pomocí něhož komunikuje SPI protokolem s MCU. Na něj je napojeno neomezené množství SPI dekodérů. Řadič provádí konverzi SPI protokolu na vnitřní sériový protokol, kterým komunikuje s dekodéry, jejichž úkolem je dekódovat adresu cílového zařízení, poskytnout mu parametrem daný počet adresových bitů z adresy a zejména umožnit jednoduché obousměrné paralelní spojení. Podrobný popis signálů rozhraní jednotlivých komponent se nalézá v dokumentu [14]. Demonstrace použití je popsána v následující kapitole.

Knihovna *libfitkit* poskytuje užitečné funkce pro komunikaci přes SPI rozhraní. Pro programátora má význam zejména funkce pro řízení signálu CS a funkce pro výměnu jednoho či více bajtů s požadovaným slave zařízením.

2.5 Překladový systém

Pro snadný vývoj aplikací na FITkitu bez nutnosti použít jakéhokoliv speciálního vývojového prostředí je vytvořen překladový systém. Stačí mít pouze nainstalované potřebné nástroje prostředí POSIX (překladač jazyka C, utility…) a nástroje pro překlad a syntézu VHDL. Systém využívá standardní GNU Makefile soubory pro překlad zdrojových kódů, naprogramování MCU a překlad, syntézu a vytvoření konfiguračního souboru pro FPGA. Seznamy překladových pravidel pro software a firmware, stejně jako globální konfigurační soubor, uchovávající parametry závislé na typu a konfiguraci operačního systému, je možné nalézt v adresáři `FITKIT/base`. Více informací o parametrizaci překladového systému v dokumentu [15].

3 Implementace Cell Matrix

Implementace architektury Cell Matrix na FITkitu je tvůrčí proces, který se skládá ze dvou do jisté míry nezávislých etap. Logicky je nejprve nutné navrhnout firmware, který bude vysyntetizován do FPGA, a v druhé fázi pak přistoupit k tvorbě vhodného uživatelského rozhraní pro komunikaci s obvodem v FPGA.

Vlastnosti softwaru a hardwaru se dnes popisují do značné míry obdobným způsobem a to vhodným programovacím jazykem. Zdrojové kódy popisující firmware jazykem VHDL (VHSIC Hardware Description Language) se nacházejí v adresáři `CELLMATRIX/top`, uživatelské rozhraní je naprogramováno v jazyce C za použití knihovny *libfitkit* a jeho zdrojové kódy jsou umístněny v adresáři `CELLMATRIX/sw`.

3.1 Synchronní D-mód

V teoretickém úvodu o Cell Matrix bylo zmíněno, že buňka v D-módu funguje jako kombinační logika, tedy asynchronně. Jelikož však tento projekt má být spíše demonstračního rázu se zaměřením na experimentování se strukturou, přistoupilo se ke zjednodušení v podobě synchronní implementace D-módu. Prakticky to znamená, že buňka v D-módu provádí změnu hodnot na výstupních signálech s náběžnou hranou hodinového signálu. Důsledkem je to, že do různých částí Cell Matrix trvá propagace signálu různý počet taktů hodin, můžeme však s přihlédnutím k velikosti matice s jistotou říct, za kolik taktů se konečná hodnota v určitém místě (např. na okrajích matice) objeví.

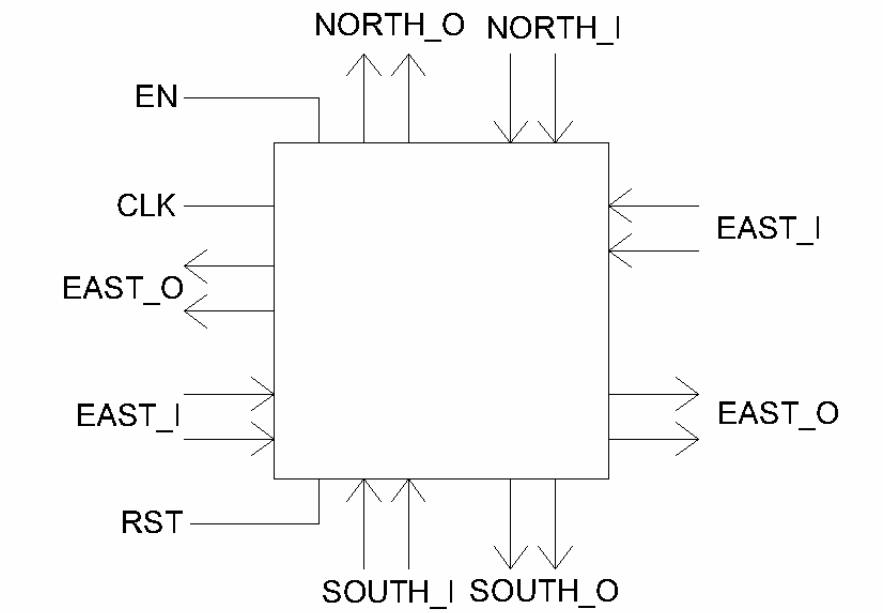
3.2 Návrh firmware

Při návrhu je vhodné postupovat v určité hierarchii. V této implementaci je zvolen přístup zdola nahoru. Nejprve tedy navrhneme entitu buňky, kterou poté několikanásobně použijeme při popisu celé matice.

3.2.1 Návrh a popis buňky

Rozhraní buňky již bylo dříve popsáno, je však nutné jej doplnit o některé další signály (obrázek 3.1). Těmi jsou:

- RST (asynchronní reset). Slouží k vynulování registru pravdivostní tabulky a výstupů buňky.
- CLK (hodinový signál). V C-módu slouží k posunu pravdivostní tabulky, v D-módu jeho náběžná hrana aktualizuje výstupy buňky.
- EN (povolení funkce). Tento signál je nutný k řízení činnosti buňky uživatelským rozhraním.



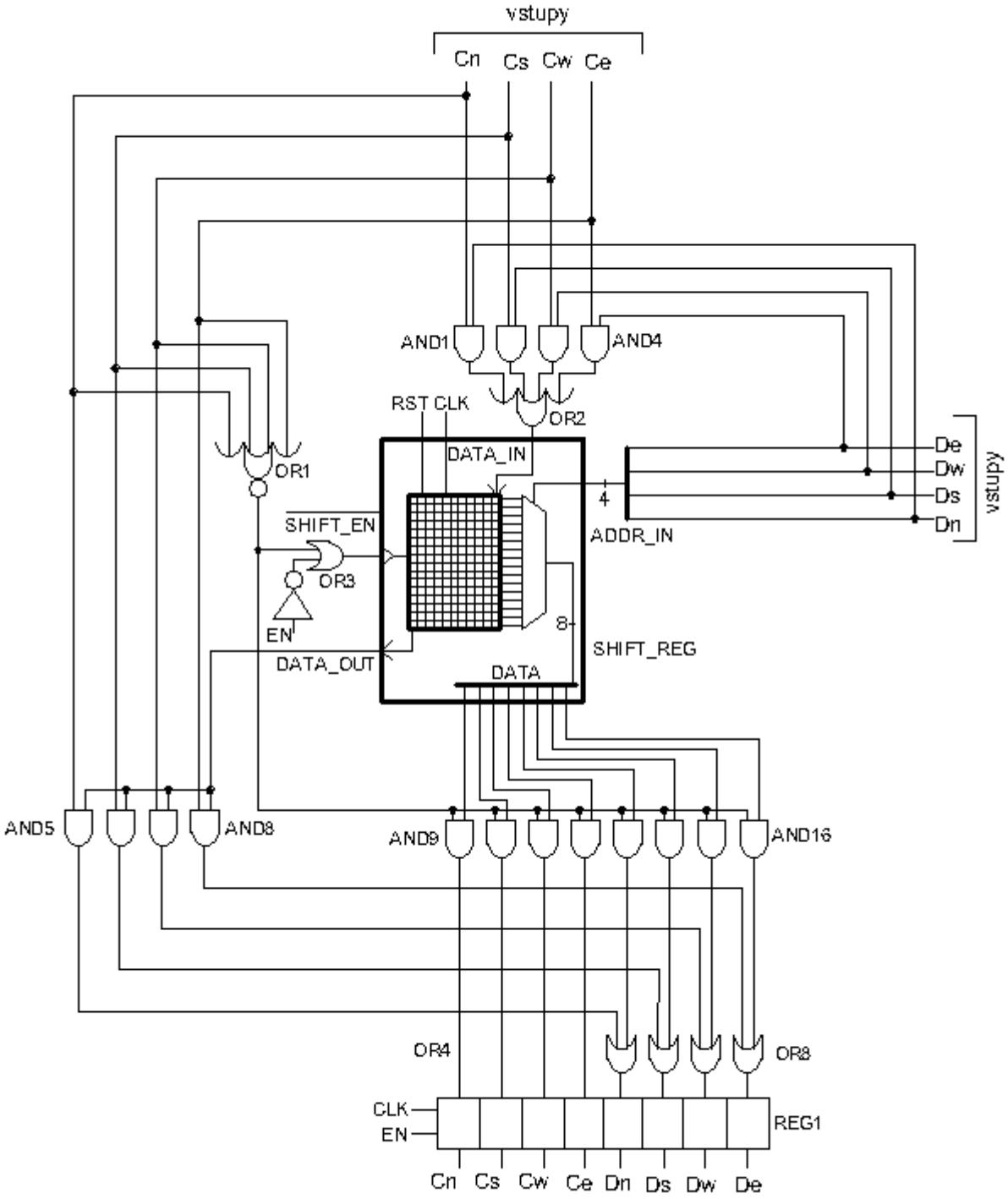
Obrázek 3.1 Rozhraní buňky

O vnitřní struktuře víme to, že se skládá z paměti k uchování pravdivostní tabulky a vedlejší logiky, která zajišťuje správnou funkci rozhraní tak, jak bylo definováno v kapitole 1.1. Návrh buňky popisuje obrázek 3.2. Je nutné říct, že obrázek je do jisté míry abstraktní a odpovídá spíše VHDL popisu ve zdrojovém kódu. Nelze spoléhat na postupy, které zvolí syntetizační nástroj, protože ty jsou podřízeny možnostem FPGA a zdrojům v programovatelných jednotkách, na které je popsaný obvod mapován.

Součástí vnitřní struktury je komponenta pravdivostní tabulky, která je popsána v souboru `shift_truth_table.vhd`. Tuto komponentu tvoří jeden 128bitový posuvný registr k uchování tabulárních dat a jeden multiplexor. Multiplexor slouží v D-módu k výběru té osmice bitů v posuvném registru, která odpovídá řádku v pravdivostní tabulce adresovaném čtveřicí vstupních D-signálů (vstupní vektor *ADDR_IN*). Tato osmice je potom vložena na výstup *DATA*. Vstup *DATA_IN* je platný při C-módu a má vždy hodnotu bitu, který bude vsunut do tabulky s náběžnou hranou hodinového signálu *CLK* za předpokladu, že je aktivní signál *SHIFT_EN* (aktivní v log. 0). Naproti tomu výstup *DATA_OUT* má vždy hodnotu posledního bitu v tabulce.

3.2.1.1 Tok dat v buňce

Signály D_N až D_E vstupují do komponenty `SHIFT_REG` s informací o adrese řádku v tabulce, pokud je buňka v D-módu. Jelikož signály C_N až C_E mají tehdy hodnoty log. 0, propustí hradlo `OR_1` log. 1, která aktivuje hradla `AND_9` až `AND_16`. Přes ně putuje osmice hodnot bitů z adresovaného řádku, která je následně zachycena skupinou klopných obvodů `REG_1`. Pokud je aktivní povolovací signál *EN* (aktivní v log. 1), je výsledek s náběžnou hranou signálu *CLK* propuštěn z buňky.



Obr. 3.2 Vnitřní struktura buňky

Pokud je buňka v C-módu, propouští hradlo `OR_1` hodnotu log. 0, která způsobí povolení hodinového signálu *SHIFT_EN* pro komponentu `SHIFT_REG` v případě, že je aktivní signál *EN*. Dále tato log. 0 způsobí uzavření hradel `AND_9` až `AND_16`. Jak již bylo popsáno, hodnota nového bitu posunutého do tabulky je rovna logickému součtu hodnot D-vstupů na aktivních stranách matice. Selekci aktivních stran provádí členy `AND_1` až `AND_4`, podmínku logického součtu pak hradlo `OR_2`. Takto vypočítaný bit je poskytnut na vstup klopného obvodu typu D nejvyššího bitu v posuvném registru pravdivostní tabulky. Na druhé straně posuvného registru je hodnota nejnižšího bitu propouštěna pouze přes ta hradla z množiny `AND_5` až `AND_8`, která odpovídají aktivním stranám. Následuje proces, kdy ve 128 hodinových taktech dochází k nasouvání nové pravdivostní tabulky. Členy `OR_4` až `OR_8` udávají, že D výstupy mohou být na rozdíl od výstupních C signálů měněny buď v C-módu nebo v D-módu.

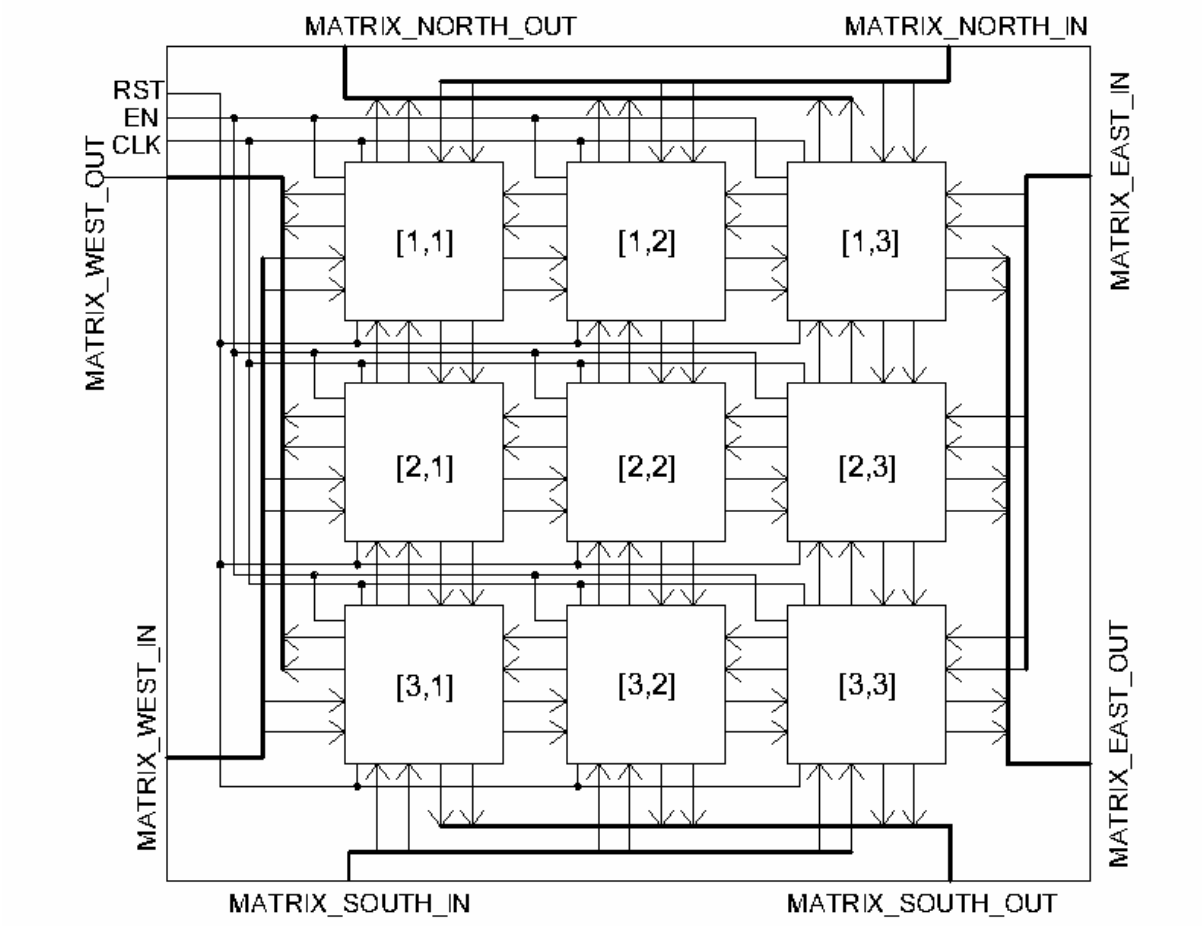
Popis rozhraní buňky a její architektura se nachází v souboru `single_cell.vhd`.

3.2.2 Popis matice buněk

Pro vytvoření matice buněk využijeme její pravidelné struktury a možnosti generického popisu ve VHDL. Generický popis umožňuje navrhnout obvod dynamicky tak, aby jeho strukturu bylo možné ovlivňovat generickými parametry. Abychom definovali matici, potřebujeme znát její rozměry, tedy počet buněk na šířku i na délku, což budou právě tyto parametry. Druhá možnost, jak vytvořit matici, je „na pevno“ pouhým pospojováním signálů několika buněk. Tento přístup však není efektivní, protože zatím nevíme, jak velkou matici budeme moci vysyntetizovat do FPGA, navíc kvůli přenositelnosti kódu na různě velká FPGA bychom měli být schopni vytvářet matice libovolných rozměrů.

Rozhraní entity je popsáno v souboru `cell_matrix.vhd`. Šířka vstupních a výstupních vektorů závisí na rozměrech matice (jejich velikost se rovná dvojnásobku počtu buněk strany příslušné vektoru), přitom vždy platí, že dva nejvýznamnější bity patří první buňce v řádku resp. sloupci, dva nejméně významné poslední buňce v řádku resp. sloupci, každý lichý bit je dedikován kanálu C a každý sudý bit kanálu D. Tato situace je naznačena na obrázku 3.3.

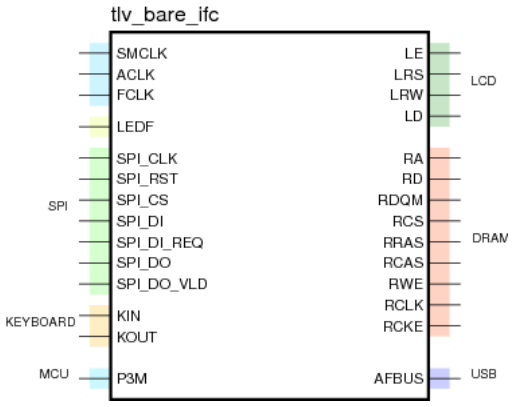
Generování matice je provedeno ve dvojnásobném cyklu přes všechny řádky a sloupce, přičemž odděleně od vnitřních buněk se generují okrajové a rohové buňky, jejichž D a C kanály nejsou spojeny s další buňkou, ale jsou propojeny s rozhraním matice. Vnitřní mezibuněčné signály jsou uloženy ve dvou dvojrozměrných polích vektorů, signály severojižní v jednom poli, východozápadní v poli druhém.



Obr. 3.3 Rozhraní a vnitřní struktura matice buněk pro rozměry 3 krát 3

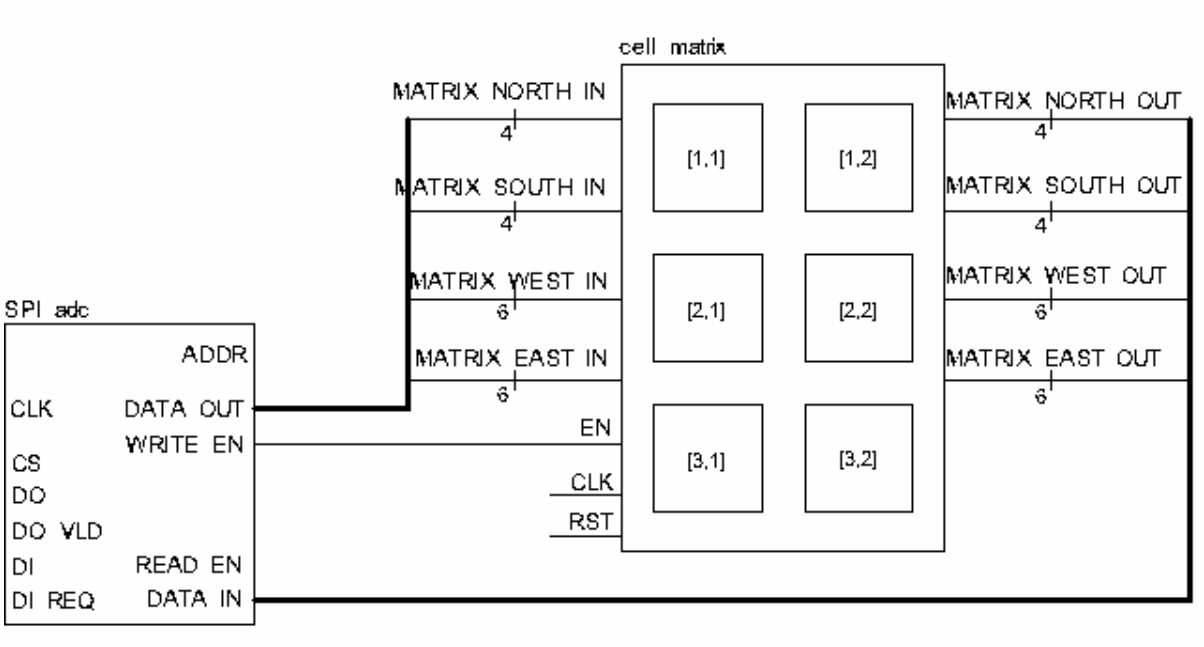
3.2.3 Top-level entita

Nejvýše v hierarchií entit VHDL projektu stojí vždy tzv. top-level entita. Její signály korespondují s jednotlivými vývody FPGA. V rámci vývoje firmware FITkitu byla vytvořena sada základních entit, které je možné v projektech použít. Při návrhu aplikace je úkolem zvolit vhodnou top-level entitu a doplnit její architekturu podle potřeb aplikace. V tomto projektu není nutné využívat sběrnici *X* (propojení s konektory VGA, RS232, PS2 případně IDE), stačí nám proto základní entita `tlv_bare_ifc` (obrázek 3.4).



Obr. 3.4 Top-level entita (převzato z [16])

K tomu, abychom mohli matici ovládat vně FPGA, je nutné využít komunikační rozhraní SPI. Jak je z obrázku 3.5 patrné, do top-level entity jsou přivedeny signály vnitřního protokolu SPI. Abychom vytvořili paralelní rozhraní pro přímou komunikaci s komponentou `cell_matrix`, připojíme na tyto signály dekodér. S výhodou můžeme využít již připravenou entitu SPI_adc ([14]).



Obr. 3.5 Schéma architektury top-level entity pro případ matice 3 krát 2

Šířka vektoru signálu *DATA_OUT* , potažmo *DATA_IN* u dekodéru je v konečném důsledku závislá na šířce vstupních resp. výstupních vektorů matice, tedy na rozměrech matice.

Aby bylo možné odstínit parametrizaci dekodéru, kterou je nutné provádět zároveň se změnou rozměrů matice, určují parametry celého obvodu konstanty ROWS_C (počet řádků matice) a COLS_C (počet sloupců matice), definované v balíku CM_PCKG (`cell_matrix_pkg.vhd`). V architektuře top-level entity (`top_level.vhd`) je pak definována konstanta INPUTS_CNT, která uchovává počet vstupů resp. výstupů matice. Její hodnota je vypočtena z konstant rozměrů matice v balíku a přiřazena generickému parametru dekodéru DATA_WIDTH, který určuje šířku vektorů *DATA_OUT* a *DATA_IN*. Na generické parametry matice jsou pouze namapovány konstanty ROWS_C a COLS_C.

3.2.4 Simulace

Souběžně s laděním architektur klíčových entit tohoto projektu byly prováděny četné simulace, které měly za úkol testovat správné chování obvodu. K tomu, abychom mohli testovat entity VHDL modelu, potřebujeme tzv. *testbench*, což není nic jiného než další zdrojový soubor ve VHDL, který však nemá definovanou entitu. *Testbench* má za úkol instanciovat komponenty a stimulovat jejich

vstupy. Programátor pak odečítá odezvy na tyto vstupy z vygenerovaného průběhu a porovnává je s očekávanými hodnotami.

Simulace se provádějí po každé fázi implementace firmware, aby bylo možné nejen ladit požadované chování obvodu (behaviorální simulace), ale také odhalit chyby, které se mohou projevit s různým zpožděním logických členů („post-place and route“ simulace). V tomto projektu nebylo nutné provádět přesnější simulace kvůli správnému časování, byly však provedeny behaviorální testy rozhraní buňky a matice buněk. Projekty testovacího prostředí *ModelSim XE* a příslušné *testbenche* lze nalézt v adresáři `CELLMATRIX/sim`.

3.2.5 Výsledky implementace dizajnu

K syntéze projektu byl použit program *xst*. Tento nástroj slouží ke kompletní analýze zdrojového kódu ve VHDL. Postupně provádí jeho překlad, sestavení hierarchie jednotlivých entit s přihlédnutím ke generickým parametrům návrhu a analýzu konečných automatů (pokud je obvod obsahuje), ale také optimalizace a základní analýzu časování. Výsledkem jsou jednak dílčí zprávy o výsledcích v jednotlivých etapách, z nichž stěžejní je odhad využití prostoru FPGA s předběžnou zprávou o časování, a jednak tzv. *netlist*, který obsahuje logický popis dizajnu na úrovni meziregistrových přenosů spolu s omezeními (*constraints*).

V druhé fázi je provedena implementace *netlistu*, která zahrnuje překlad na reprezentaci primitivními logickými hradly, kterou provádí nástroj *ngdbuild*. V další etapě program *map* provede mapování hradlové reprezentace na elementy specifické technologie, v našem případě na prvky uvnitř *slices* (LUT). Až v této části jsou známy přesné hodnoty utilizace FPGA. V předposlední fázi (place and route) je rozhodnuto, kam se mapované prvky umístní na čipu, a jak budou propojeny. Nakonec je vytvořen konfigurační soubor, který je nahrán do FPGA.

V našem případě bylo zjištěno, že kapacita FPGA stačí na vysyntetizování maximálně šesti buněk Cell Matrix, přitom celý obvod pokrývá 87 procent kapacity konfiguračních jednotek (*slices*) a může být synchronizován signálem o frekvenci přibližně 115 MHz. Podrobné výsledky syntézy a implementace dizajnu poskytují úseky logů v příloze.

3.3 Návrh software

Aby bylo možné experimentovat s maticí implementovanou v FPGA, je potřeba navrhnout vhodné uživatelské rozhraní a sadu příkazů pro výkon nejrůznějších funkcí. Veškerou komunikaci s maticí řídí mikrokontrolér. Jako prostředník mezi uživatelem a maticí interpretuje a vykonává příkazy uživatele přijaté přes sériové rozhraní z terminálu a zpět na terminál zasílá návratové hodnoty.

3.3.1 Práce se vstupy a výstupy matice

Vstupy a výstupy matice jsou na straně MCU ukládány do pole. Je zřejmé, že velikost tohoto pole se mění v závislosti na rozměrech matice. Konkrétně je použito pole prvků typu `unsigned char` velikosti 8 bitů. Jednotlivé bity odpovídají vstupním resp. výstupním datovým signálům rozhraní matice, potažmo jednotlivým bitům vektoru *DATA_OUT* resp. *DATA_IN* v rozhraní SPI dekodéru. Uspořádání signálů v poli naznačuje tabulka 3.1.

Strana matice	NORTH								WEST						EAST									
Index buňky	1		2		1		2		1		2		3		1		2		3					
Typ signálu	c	d	c	d	C	d	c	d	c	d	c	d	c	d	c	d	C	d	c	d				
Index bitu	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Prvek pole	data[0]								data[1]						data[2]									

Tab. 3.1. Uspořádání signálů rozhraní matice velikosti 3 krát 2

Se znalostmi principů vyhodnocování výrazů v jazyce C můžeme pro výpočet velikosti takového pole použít vzorec

```
INPUT_ARRAY_SIZE = INPUTS_CNT/8 + ((INPUTS_CNT % 8) > 0),
```

kde `INPUTS_CNT` je počet vstupů nebo výstupů matice, získaný ze vzorce

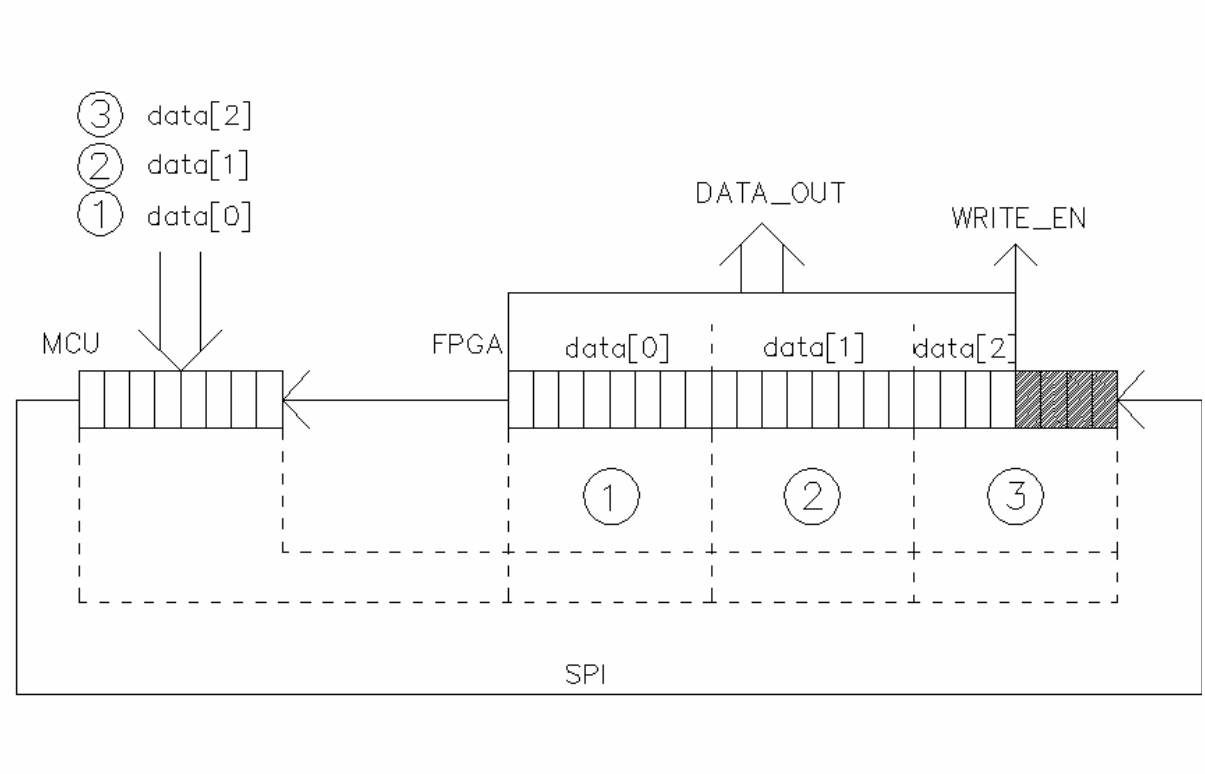
```
INPUTS_CNT = 4*(MATRIX_ROWS + MATRIX_COLS).
```

`MATRIX_ROWS` je zde počet řádků a `MATRIX_COLS` počet sloupců matice.

Ke čtení a zápisu jednotlivých bitů v poli slouží speciální funkce, jejichž jeden parametr je takto uspořádané pole a druhý index bitu v něm.

Návrh datové struktury k uložení vstupně výstupní informace byl účelný vzhledem k funkcím dostupným pro komunikaci s FPGA přes SPI v knihovně *libfitkit*. Konkrétně se jedná o funkci `FPGA_SPI_RW_A8_DN(...)`.

První parametr této funkce udává, zda se jedná o operaci čtení (symbolická konstanta SPI_FPGA_ENABLE_READ) nebo zápis (SPI_FPGA_ENABLE_WRITE), následuje parametr bázové adresy SPI dekodéru, který musí korespondovat s adresou zadanou při generickém mapování komponenty `SPI_adc` ve VHDL zdroji (pro adresu o šířce 8 bitů má hodnotu např. 00h), třetí parametr je ukazatel na pole určené k výměně dat a poslední udává počet bloků dat o délce *n* krát 8 bitů.



Obr. 3.6 Schéma operace zápis pro matici 3 krát 2

Na obrázku 3.6 vidíme postup při operaci zápis. Po příjmu a rozpoznání operačního kódu a adresy dekodérem odesílá MCU postupně 8bitové bloky a SPI dekodér na straně FPGA je přijímá do vstupního datového registru velikosti parametru DATA_WIDTH. Pokud je tento registr zaplněn, je aktivován signál *WRITE_EN*, který je napojen na vstup EN matice, což způsobí povolení hodinového signálu. Matice si tak v dalším taktu převezme data. Se zaplněním datového registru dekodéru se může stát, že z posledního 8bitového bloku zbývá poslat ještě několik bitů. Ty jsou sice do registru přeneseny, obecně však mají nedefinovanou hodnotu a nenesou žádný význam. Ukončení zápisu dat udává signál CS, který resetuje registry dekodéru.

Operace čtení probíhá v našem případě bez účasti signálu *READ_EN*, který je dekodérem vystaven při příchodu žádosti o čtení. Při čtení jsou hodnoty na sběrnici *DATA_IN* paralelně nahrány do výstupního datového registru a v dalších taktech vysouvány do MCU.

3.3.2 Funkce uživatelského rozhraní

Pokud bychom implementovali pouze nízkoúrovňovou komunikaci, stačí vytvořit funkce pro změnu vstupů, přečtení výstupů a posun hodinového signálu matice. Pro snadnou práci zejména s konfigurací matice však byly vytvořeny i příkazy pro vložení posloupnosti bitů a pro provedení sekvence operací.

Funkce, které používá MCU pro implementaci uživatelských příkazů, jsou definovány v modulu `config.c`. Prototypy těchto funkcí jsou deklarovány v hlavičkovém souboru `config.h`,

který vedle exportu některých globálních proměnných (knihovna, konfigurační sekvence, viz dále) definuje symbolické konstanty pro parametrizaci chování uživatelského rozhraní. Pro uživatele mají význam především konstanty MATRIX_ROWS (počet řádků matice) a MATRIX_COLS (počet sloupců). Je nutné si uvědomit, že pro správnou funkci programu musíme zachovávat vzájemnou koincidenci těchto parametrů na obou stranách komunikace. Změnu konstant rozměrů v balíku `cell_matrix_pkg.vhd` proto provádíme souběžné se změnou v `config.h`.

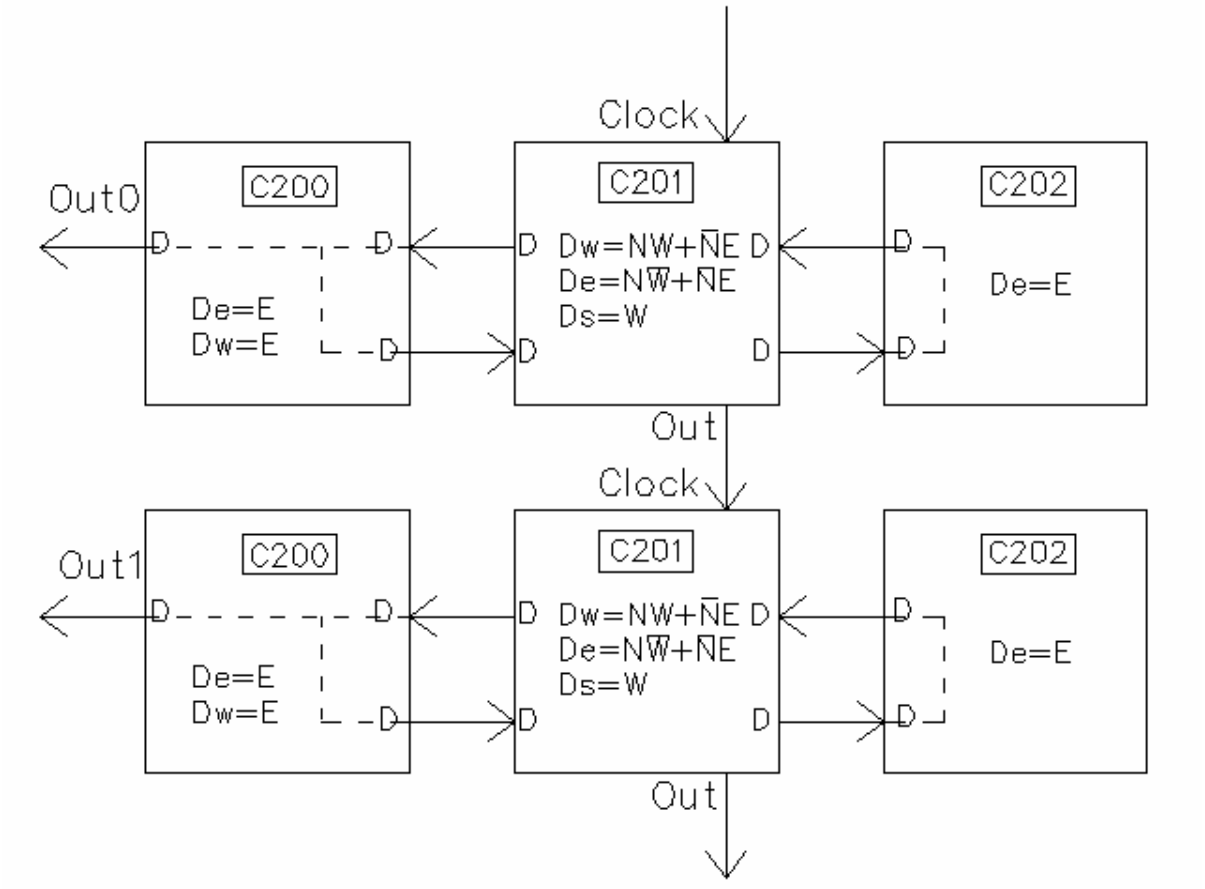
Pro podrobnější informace o implementaci odkazuji na programovou dokumentaci, vygenerovanou systémem *doxygen* v adresáři `CELLMATRIX/sw/doc/html`. Syntaxe jednotlivých příkazů s přesným popisem se nachází v příloze a v souboru uživatelské příručky `README.txt` v adresáři `CELLMATRIX/sw/doc`.

4 Ověření funkčnosti

Abychom si byli jisti, že vytvořená implementace skutečně funguje podle specifikace dané architekturou Cell Matrix, bylo připraveno několik ukázek konfigurací matice, které demonstrují funkčnost jednoduchých číslicových prvků.

4.1 Čítač

Čítač je komponenta, která v závislosti na hodnotě vstupního signálu obdélníkovitého průběhu čítá posloupnost binárních čísel, jejichž jednotlivé bity se objevují na výstupu. Rozsah čítané posloupnosti je v případě realizace takové komponenty v Cell Matrix úměrně závislý na rozsahu matice, a to v poměru 1:3 (za každý bit výstupu tři buňky Cell Matrix). Připomeňme, že v našem případě stačí kapacita FPGA na 6 buněk Cell Matrix, což odpovídá čítači, který zvládne čítat 2bitovou posloupnost.



Obr. 3.7 Konfigurace ‚čítač'

Vstup signálu obdélníkového průběhu je označen *CLOCK*. Při nástupné hraně je výsledek vstupu prostřední buňky propagován do buňky napravo odkud je vrácen, při sestupné hraně je propagován na

výstup *OUTn*, kde *n* je n-tý bit výsledku, a opět vrácen zpět do prostřední buňky. Při sestupné hraně vstupního signálu je také generována hrana na výstup *OUT*, která slouží jako vstupní signál pro další řádek čítače, který obstarává vyšší bit čísla. Nejlépe lze pochopit funkci čítače použitím simulátoru na adrese http://www.cellmatrix.com/entryway/products/software/jpearn.html.

Matici do funkce čítače můžeme nakonfigurovat trojicí příkazů. Pravdivostní tabulky již máme nachystané v knihovně (názvy pravdivostních tabulek viz obrázek 3.7):

```
>b c200 ns 1
>b c201 ns 2
>b c202 ns 3
```

Čítání budeme provádět změnou logických hodnot vstupu D prostřední buňky. Je nutné si uvědomit, že D mód buněk pracuje v naší implementaci synchronně a propagace signálu v buňkách trvá určitý počet taktů, se kterým musíme počítat při krokování příkazem `s`. Teoreticky je možné určit kolik kroků při jaké hodnotě *CLOCK* stačí provést, abychom získali konečné hodnoty dalšího prvku posloupnosti na výstupech čítače, nám však bude stačit, že provedeme „dostatečně velký“ počet kroků na to, aby se propagace maticí ustálila (např. 10). Generování periody obdélníkového signálu si můžeme uložit do sekvence příkazů a příkazem `bs` pouze zjišťovat další prvek posloupnosti:

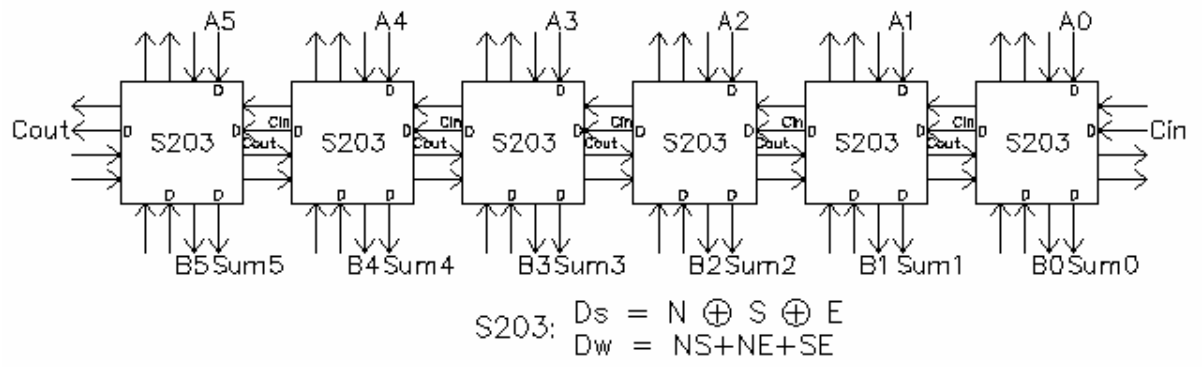
```
>w n 2 d  //zápis log. 1 na D vstup 2. buňky severní strany matice
>s 10  //provedení dostatečného počtu kroků k propagaci sig. maticí
>w        //vynulování vstupů matice
>s 10
>r o   //výpis výstupů
```

Prvky čítané posloupnosti se objevují na západní straně matice.

4.2 Úplná sčítačka

Sčítačka představuje jednodušší konfiguraci než čítač. Vše, co potřebujeme, je vytvořit pravdivostní tabulku, která realizuje funkci 1bitové úplné sčítačky (viz tabulka 1.1). Buňky naprogramované takovou pravdivostní tabulkou pak pouze zapojíme vedle sebe do kaskády přes signály přenosů. Protože můžeme vysyntetizovat až 6 buněk, naše sčítačka může být až 6bitová. Matici o rozměrech 1 krát 6 můžeme nakonfigurovat např. příkazem

```
>b s203 n 1,2,3,4,5,6 c
```



Obr. 3.8 Konfigurace ‚úplná sčítačka‘

Matice seřazená podle vzoru na obrázku 3.8 produkuje ve své nejlevější buňce nejvýznamnější bit výsledku (Sum5), naproti tomu buňka na opačné straně dává nejméně významný bit. Stejné uspořádání musí platit i pro bity sčítanců A a B. Demonstračně provedeme sečtení čísel 38 a 14, kdy nehrozí přetečení rozsahu. Vložení čísel na vstupy provedeme těmito příkazy:

```
>w n 1,4,5 d      //A = 100110b
>w s 3,4,5 d      //B = 001110b
```

Poté provedeme např. 10 kroků pomocí příkazu `s`. Výsledek můžeme odečíst z hodnot výstupů na jižní straně matice. Pokud na vstupy vystavíme čísla, jejichž rozsah součtu přesahuje 6 bitů, je u buňky na pozici nejvýznamnějšího bitu navíc generován výstupní přenos Cout.

5 Závěr

Architektura Cell Matrix má do budoucna velký potenciál. Díky jejím vlastnostem je možné využití zejména v oblastech nanotechnologií a hardwaru se schopností autonomního vývoje (evolvable hardware). Byly vytvořeny aplikační modely, využívající rekonfigurační schopnosti obvodů, jako např. model rozbití algoritmu DES, ve kterých Cell Matrix výkonově porážejí stejné aplikace postavené na FPGA [17]. Technologické možnosti a nedostatek zájmu však zatím neumožňují masové využití, ani nástroje pro dostatečně efektivní programování takových obvodů nejsou zatím dostupné.

Jako hlavní přínos tohoto projektu obecně spatřuji v možnostech experimentování s Cell Matrix ač na velmi omezeném prostoru. Oblast budoucího rozšiřování projektu je však takřka neomezená. Další práce na tomto projektu je nutné zaměřit především na optimalizaci návrhu a to zejména z hlediska plochy. Velkou výzvou je také práce na uživatelském rozhraní pro ovládání Cell Matrix v FPGA, zahrnující zejména rozšíření stávající sady příkazů, popř. vytvoření grafické nadstavby nad nimi.

Při práci na tomto projektu jsem si zdokonalil znalosti jazyka VHDL a vyzkoušel si práci s platformu FITkit, zejména programování mikrokontroléru v jazyce C.

Literatura

[1] Janíček, F.: Vlastnosti a použití architektury Cell Matrix [diplomový projekt]. Brno, FIT VUT v Brně, 2004.

[2] Wikipedia: Reconfigurable computing [online]. Last modified 15:00, 23 April 2007. URL: <http://en.wikipedia.org/wiki/Reconfigurable_computing>.

[3] Cell Matrix corporation website [online]. URL: <www.cellmatrix.com>.

[4] Durbeck, L.: Introduction to cells [online]. Cell Matrix corporation, c2002 [cit. 2007-05-02]. URL: <http://www.cellmatrix.com/entryway/products/concepts/intro3.html>.

[5] Janíček, F.: Stavební bloky pro architekturu Cell Matrix [ročníkový projekt]. Brno, FEI Ústav elektrotechniky a informatiky v Brně, 2002.

[6] Durbeck, L.: Comparison to design features of an FPGA: Not an FPGA [online]. Cell Matrix corporation, c1999 [cit. 2007-05-02]. URL: <http://www.cellmatrix.com/entryway/products/concepts/comparison1.html>.

[7] Xilinx, Inc.: Products & Services: Xilinx Virtex-II Pro Platform FPGA [online]. c1994-2007. URL: <http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex_ii_pro_fpgas/index.htm>.

[8] Durbeck, L., Macias, N.: Self-configurable parallel processing system made from self-dual code/data processing cells utilizing a non-shifting memory, US Patent #6,222,381, 2001. URL: <http://www.cellmatrix.com/entryway/products/pub/publications.html>.

[9] Stránky projektu FITkit [online]. Brno, FIT VUT v Brně, c2006. URL: <http://merlin.fit.vutbr.cz/FITkit>.

[10] FITkit: Popis kitu [online]. Stránky projektu FITkit, Brno, c2006 [cit. 2007-05-02]. URL: <http://merlin.fit.vutbr.cz/FITkit/hardware.html>.

[11] Markovič, J.: Princip systému bootování [online]. Stránky projektu FITkit, Brno, c2006 [cit. 2007-05-01]. URL: <http://merlin.fit.vutbr.cz/FITkit/docs/hardware/20060205a.html>.

[12] Xilinx, Inc.: Xilinx DS099 Spartan-3 Complete data sheet [online]. c2003-2006, April 26, 2006 [cit. 2007-05-02]. URL: <http://www.xilinx.com/bvdocs/publications/ds099.pdf>.

[13] Markovič, J.: FITkit firmware: Knihovna libfitkit [online]. Stránky projektu FITkit, Brno, c2006 [cit. 2007-05-02]. URL: <http://merlin.fit.vutbr.cz/FITkit/docs/firmware/20060501lib.html>.

[14] Vašíček, Z.: FITkit firmware: Propojovací systém FITkitu [online]. Stránky projektu FITkit, Brno, c2006 [cit. 2007-05-02]. URL: <http://merlin.fit.vutbr.cz/FITkit/docs/firmware/spifitkit.html>.

[15] Vašíček, Z.: FITkit návody: Překladový systém [online]. Stránky projektu FITkit, Brno, c2006 [cit. 2007-05-02]. URL: <http://merlin.fit.vutbr.cz/FITkit/docs/navody/makefile.html>.

[16] Vašíček, Z.: FITkit návody: VHDL projekt od začátku [online]. Stránky projektu FITkit, Brno, c2006 [cit. 2007-05-02]. URL: <http://merlin.fit.vutbr.cz/FITkit/docs/navody/toplevel.html>.

[17] Durbeck, L., Macias, N.: The Cell Matrix: An Architecture for Nanocomputing. Nanotechnology 12 (2001) 217–230.
URL: <http://www.cellmatrix.com/entryway/products/pub/publications.html>.

Seznam použitých zkratek a symbolů

FPGA – Field-Programmable Gate Array

VHDL – VHSIC Hardware Description Language

VHSIC – Very-High-Speed Integrated Curcuit

DRAM – Dynamic Random Access Memory

VGA – Video Graphics Array

USB – Universal Serial Bus

UART – Universal asynchronous receiver-transmitter

RS232 – Recommended Standard 232

MCU – Microcontroller Unit

JTAG – Joint Test Action Group

SPI – Serial Peripheral Interface

CPU – Central Processing Unit

CLB – Configurable Logic Block

IOB – Input-Output Block

LUT – Look-up Table

BRAM – Block Random Access Memory

DCM – Digital Clock Manager

MISO – Master-In Slave-Out

MOSI – Master-Out Slave-In

POSIX – Portable Operating System Interface

GNU – GNU’s Not Unix

PS2 – IBM Personal System/2

IDE – Integrated Drive Electronics

DES – Data Encryption Standard

Seznam příloh

Přílohy

Příloha 1. Entity VHDL popisu

```
--entita posuvneho registru uvnitr bunky
entity SHIFT_REG is
     port(
          DATA_IN: in std_logic; --vstup pri posunu
          DATA_OUT: out std_logic; --vystup bitu pri posunu
          RST: in std_logic; --reset
          CLK: in std_logic; --hodinovy signal
          SHIFT_EN: in std_logic; --povoleni hodin, clock enable
          --adresa radku tabulky
          ADDR_IN: in std_logic_vector (3 downto 0);
          DATA: out std_logic_vector (7 downto 0) --vystup tabulky
         );
end SHIFT_REG;

--entita bunky
entity single_cell is
    port (
     -- vektory vstupu a vystupu bunky pro vsechny ctyri strany
          NORTH_I: in std_logic_vector (1 downto 0);
          NORTH_O: out std_logic_vector (1 downto 0);

          SOUTH_I: in std_logic_vector (1 downto 0);
          SOUTH_O: out std_logic_vector (1 downto 0);

          WEST_I: in std_logic_vector (1 downto 0);
          WEST_O: out std_logic_vector (1 downto 0);

          EAST_I: in std_logic_vector (1 downto 0);
          EAST_O: out std_logic_vector (1 downto 0);

          CLK : in std_logic;
          RST : in std_logic;
          EN : in std_logic);
end single_cell;
```

```
--entita matice
entity cell_matrix is
      generic (
      --parametry urcujici rozmer matice
            ROWS : integer;
            COLS : integer
      );
      port (
      -- signaly, pres ktere se konfiguruji okrajove bunky matice
            MATRIX_NORTH_IN: in std_logic_vector ((COLS*2 - 1) downto 0);
            MATRIX_NORTH_OUT: out std_logic_vector ((COLS*2 - 1) downto 0);

            MATRIX_SOUTH_IN: in std_logic_vector ((COLS*2 - 1) downto 0);
            MATRIX_SOUTH_OUT: out std_logic_vector ((COLS*2 - 1) downto 0);

            MATRIX_WEST_IN: in std_logic_vector ((ROWS*2 - 1) downto 0);
            MATRIX_WEST_OUT: out std_logic_vector ((ROWS*2 - 1) downto 0);

            MATRIX_EAST_IN: in std_logic_vector ((ROWS*2 - 1) downto 0);
            MATRIX_EAST_OUT: out std_logic_vector ((ROWS*2 - 1) downto 0);

            CLK : in std_logic; --hodinovy signal pro celou matici
            RST : in std_logic; --reset pro celou matici

            EN: in std_logic); --povoleni funkce matice
end cell_matrix;
```

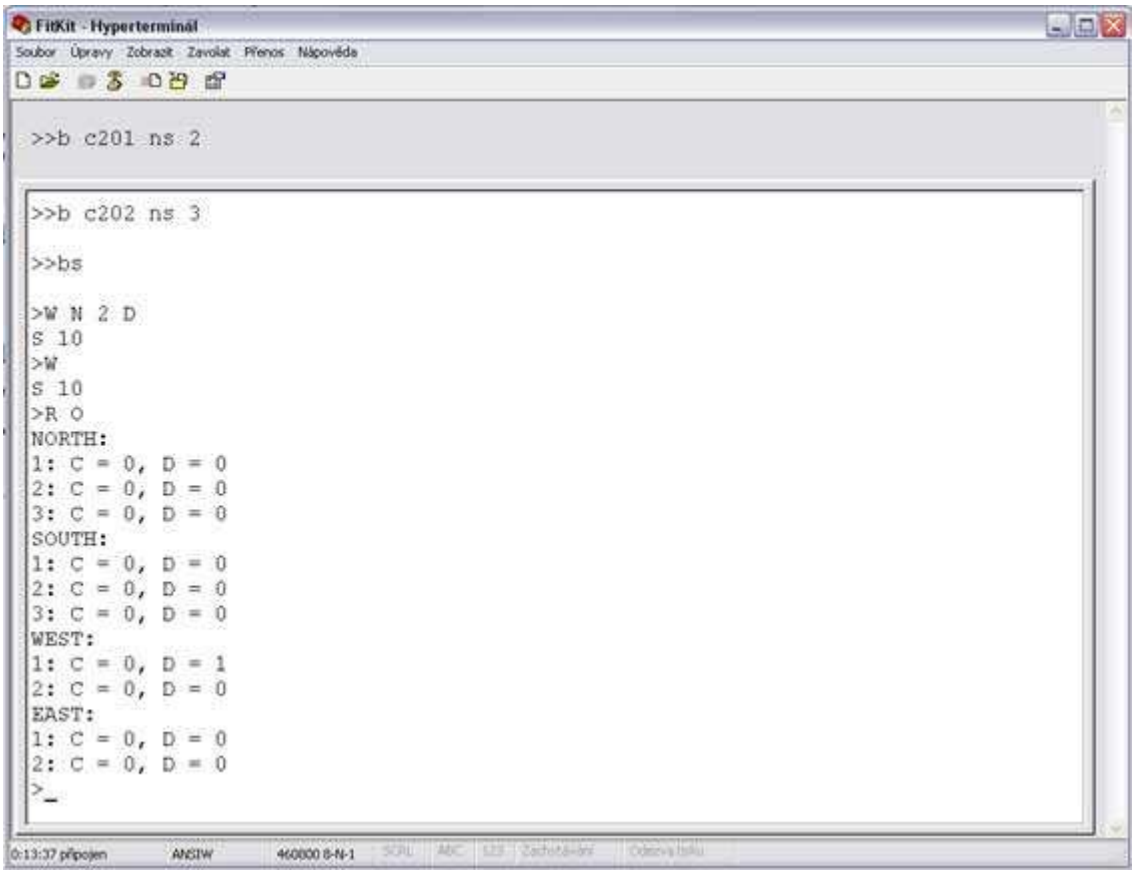
Příloha 2. Úseky logů z překladu VHDL projektu matice 6 buněk

```
Synthesizing Unit <SHIFT_REG>.
    Related source file is "C:/FitKit
sources/fitkit/FITkit/apps/cellmatrix/top/shift_truth_table.vhd".
    Found 8-bit 16-to-1 multiplexer for signal <DATA>.
    Found 128-bit register for signal <reg>.
    Summary:
      inferred 128 D-type flip-flop(s).
      inferred   8 Multiplexer(s).
Unit <SHIFT_REG> synthesized.

Synthesizing Unit <single_cell>.
    Related source file is "C:/FitKit
sources/fitkit/FITkit/apps/cellmatrix/top/single_cell.vhd".
    Found 4-bit register for signal <code_out>.
    Found 4-bit register for signal <data_out>.
    Summary:
      inferred   8 D-type flip-flop(s).
Unit <single_cell> synthesized.
```

```
HDL Synthesis Report

Macro Statistics
# Counters                                             : 2
 5-bit up counter                                      : 1
 8-bit up counter                                      : 1
# Registers                                            : 23
 1-bit register                                        : 2
 128-bit register                                      : 6
 2-bit register                                        : 1
 20-bit register                                       : 2
 4-bit register                                        : 12
# Multiplexers                                         : 6
 8-bit 16-to-1 multiplexer                             : 6
# Tristates                                            : 37
 1-bit tristate buffer                                 : 37
```

```
Advanced HDL Synthesis Report

Macro Statistics
# FSMs                                              : 2
# Counters                                          : 2
 5-bit up counter                                   : 1
 8-bit up counter                                   : 1
# Registers                                         : 186
 Flip-Flops                                         : 186
# Multiplexers                                      : 48
 1-bit 16-to-1 multiplexer                          : 48

Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 8.747ns (Maximum Frequency: 114.325MHz)
   Minimum input arrival time before clock: 4.434ns
   Maximum output required time after clock: 9.329ns
   Maximum combinational path delay: No path found

Design Summary:
----------------
Logic Utilization:
  Number of Slice Flip Flops:         881 out of   1,536   57%
  Number of 4 input LUTs:             541 out of   1,536   35%
Logic Distribution:
  Number of occupied Slices:          669 out of     768   87%
Total Number of 4 input LUTs:         541 out of   1,536   35%
  Number of bonded IOBs:               89 out of     124   71%
  Number of GCLKs:                      1 out of       8   12%

Total equivalent gate count for design:  11,307
Additional JTAG gate count for IOBs:  4,272
Peak Memory Usage:  143 MB
```

Příloha 3. Komunikace s FITkitem přes program HyperTerminál



Příloha 4. Programová dokumentace ve formátu HTML generovaná systémem doxygen

Programová dokumentace se nachází na přiloženém CD v adresáři `/sw/doc/html`.

Příloha 5. Uživatelská příručka v textovém formátu

Uživatelská příručka (README.txt) obsahující informace o zprovoznění projektu a ovládání uživatelského rozhraní se nachází na přiloženém CD v adresáři `/sw/doc`.

Příloha 6. CD, adresářová struktura projektu

```
CD
|----/CELLMATRIX  (zdrojové soubory projektu a programová dokumentace)
|     |----/sw
|     |     |----/doc
|     |     |     |----/Doxyfile
|     |     |     |----/README.txt
|     |     |     |----/html
|     |     |     |     |----  Soubory programové dokumentace
|     |     |----/config.c
|     |     |----/config.h
|     |     |----/main.c
|     |     |----/Makefile
|     |     |----/cygwin1.dll
|     |----/top
|     |     |----/shift_truth_table.vhd
|     |     |----/single_cell.vhd
|     |     |----/cell_matrix.vhd
|     |     |----/top_level.vhd
|     |     |----/cell_matrix_pkg.vhd
|     |     |----/Makefile
|     |----/sim
|     |     |----/cell_matrix
|     |     |     |---- Simulační projekt matice pro ModelSim XE
|     |     |----/single_cell
|     |     |     |---- Simulační projekt buňky pro ModelSim XE
|     |----/cellmatrix.ppr
|----/techreport  (technická zpráva ve zdrojovém tvaru i PDF)
|     |----/ibp.doc
|     |----/ibp.pdf
```