VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

TESTOVÁNÍ SPOJŮ A EXTERNÍCH PAMĚŤOVÝCH KOMPONENT V FPGA

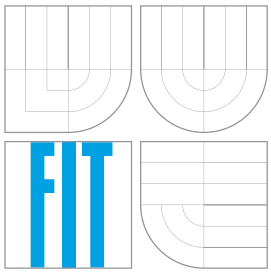
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN LOUDA

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

TESTOVÁNÍ SPOJŮ A EXTERNÍCH PAMĚŤOVÝCH KOMPONENT V FPGA

TESTING OF WIRES AND EXTERNAL MEMORY COMPONENTS IN FPGA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN LOUDA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ MARTÍNEK

BRNO 2008

Abstrakt

Tato práce se zabývá testováním propojů a paměťových prostředků na kartě COMBO2. Začátek práce je věnován představení problematiky testování propojů a pamětí typu RAM. V hlavní části práce je představen návrh obecné architektury testu propojů a testu paměťových prostředků, který je soustředěn na cílovou platformu FPGA.

Klíčová slova

testování spojů, testování pamětí RAM, March test, FPGA, VHDL, Handel-C, COMBO2

Abstract

This work deals with COMBO2 card interconnect and memory devices testing. In the beginning of the paper, some existing testing algorithms for interconnect and RAM memories testing are introduced. This work is devoted to proposal of generic architecture for interconnect and memory devices testing. The proposed architecture is optimized for FPGA implementation.

Keywords

interconnect testing, wires testing, RAM memory testing, March test, FPGA, VHDL, Handel-C, COMBO2

Citace

Martin Louda: Testování spojů a externích paměťových komponent v FPGA, diplomová práce, Brno, FIT VUT v Brně, 2008

Testování spojů a externích paměťových komponent v FPGA

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně pod vedením Ing. Tomáše Martínka. Další informace mi poskytli kolegové z projektu Liberouter. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Louda
16. května 2008

Poděkování

Chtěl bych na tomto místě poděkovat vedoucímu své diplomové práce Ing. Tomáši Martínkovi za poskytnuté informace a čas strávený konzultacemi. Dále bych rád poděkoval kolegům z projektu Liberouter.

Obsah

Kapitola 1

Úvod

S rozvojem síťových technologií a zvyšováním rychlosti, na které dnešní síťová zařízení pracují, vzrůstají požadavky na specializované síťové platformy. Pro tyto síťové platformy je specifická zejména potřeba rychlého přenosu dat mezi i uvnitř zařízení a dále potřeba paměťových prostředků, které umožňují zaznamenat informace o jednotlivých paketech či datových tocích.

Tato práce se konkrétně zabývá testováním propojů a paměťových prostředků na kartě COMBO2. Tato karta se používá převážně v síťových aplikacích, například pro hardwarové filtrování či směrování paketů. Ke kartě COMBO2 lze pomocí speciálních propojů připojit kartu síťových rozhraní nebo i jiná zařízení. Navíc pro podporu síťových aplikací je karta osazena několika typy paměťových prostředků.

Metody testování propojů jsou vyvíjeny již po několik desetiletí. S rostoucími požadavky na rychlost přenosu dat je potřeba vyvíjet efektivnější metody testování, které dokáží odhalit více typů poruch v co nejkratším čase. Se zvyšující se rychlostí přenosu se také objevují nové problémy, které nebylo potřeba dříve řešit.

Testováním paměťových prostředků se zabývá samostatné odvětví diagnostiky. První vyvinuté testy pamětí byly schopné detekovat pouze malou podmnožinu chyb, které se mohly v paměti vyskytnout. Ukázalo se, že je zapotřebí vyvinout nové metody testování pamětí, protože těmito testy procházelo mnoho chybných pamětí. Výsledkem výzkumů v oblasti testování pamětí bylo zavedení chybových modelů, které zohledňují strukturu paměti a možnosti vzájemného ovlivňování jednotlivých buněk paměti. Pro potřeby testování moderních pamětí byly vyvinuty testy typu March, které detekují mnoho typů chyb a jejich časová složitost je lineární.

V druhé kapitole jsou představeny některé metody testování propojů, jsou uvedeny algoritmy s největší diagnostickou schopností. Třetí kapitola se zabývá metodikou testování paměťových prostředků. Dále je zde představena problematika testování bitově a slovně orientovaných pamětí. V následujících kapitolách jsou představeny spolu s testovanou architekturou také navržené architektury pro testování propojů a paměťových prostředků na kartě COMBO2. Na závěr jsou nastíněny možnosti pokračování práce.

Kapitola 2

Metody testování propojů

Základy testování propojovacích sítí byly položeny již v sedmdesátých letech dvacátého století. S novými technologiemi a narůstající složitostí elektrických obvodů vzniká potřeba stále rychlejšího přenosu dat. S rostoucí přenosovou rychlostí vznikají další typy poruch, které je nutné diagnostikovat. Metody pro testování spojů se neustále vyvíjejí. Zlepšuje se jejich diagnostická schopnost a zároveň se zmenšuje doba potřebná pro testovací proceduru.

2.1 Poruchy propojů

Dva základní typy poruch vodičů (propojů) jsou zkraty (shorts) a přerušené vodiče (opens). Dále můžeme uvažovat poruchy jednoduché a násobné. Jednoduchá porucha se projeví chybou na jednom vodiči, zatímco násobná porucha reprezentuje chyby na více vodičích zároveň – relevantní pouze u propojů sběrnicového typu.

Nyní se blíže seznámíme se základními typy poruch. Pokud jsou dva (nebo více) vodiče zkratované, jejich výsledná logická hodnota může být určena jako logický OR hodnot vodičů, logický AND, anebo hodnotu určuje jeden „nejsilnější“ vodič – říkáme že dominuje. V každém případě bude hodnota na všech zkratem ovlivněných vodičích stejná. Pokud je vodič zkratován s napájecím vodičem, bude vykazovat hodnotu trvalá 1 (stuck-at 1). Analogicky při zkratu se zemnícím vodičem bude hodnota trvalá 0 (stuck-at 0). Pokud je vodič přerušený, může být jeho hodnota buď logická 0 nebo logická 1 (závisí na implementaci přijímače), což se označuje pojmem soft stuck-at 0, resp. soft stuck-at 1. Důležité je si uvědomit, že logická hodnota všech přerušených vodičů bude stejná. Na jednom vodiči se mohou vyskytnout oba typy poruch, tedy zkrat i přerušení. Za této situace je logická hodnota na vodiči určena kombinovaným vlivem obou poruch.

Pro modelování konkrétních druhů testů se používá tato terminologie: [8]

PTV – Parallel Test Vector Vektor logických hodnot aplikovaný paralelně na všechny vodiče.

STV – Sequential Test Vector Vektor logických hodnot aplikovaný sekvenčně na jeden vodič.

TS – Test Set Soubor všech STV. Každý řádek TS je STV a každý sloupec TS je PTV.

SRV – Sequential Response Vector Vektor na přijímací straně po odeslání STV vysílací stranou.

Syndrome SRV chybného vodiče.

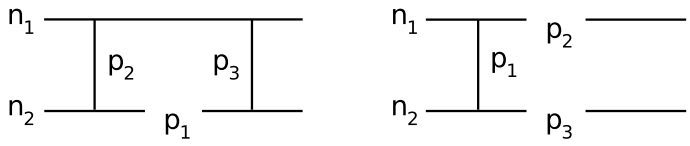
Může nastat situace, kdy nelze přesně identifikovat poruchy vodičů. Například když je odezva špatných vodičů stejná jako odezva dobrých vodičů. Tento jev je označován jako *aliasing* a lze demonstrovat na následujícím příkladu. Mějme testovací vektory (TS) v tabulce 2.1. Předpokládejme, že na vodičích n_3 a n_4 nastal zkrat a model chování těchto vodičů je typu OR. Aplikací TS dostaneme na příslušných vodičích SRV odpovídající hodnotě 0111, který ale odpovídá správnému SRV vodiče n_7. V tomto případě tedy nelze určit, mezi kterými vodiči je zkrat.

	v_1	v_2	v_3	v_4
n_1	0	0	0	1
n_2	0	0	1	0
n_3	0	0	1	1
n_4	0	1	0	0
n_5	0	1	0	1
n_6	0	1	1	0
n_7	0	1	1	1
n_8	1	0	0	0
n_9	1	0	0	1
n_{10}	1	0	1	0

Tabulka 2.1: Příklad testovacích vektorů TS.

Další možná komplikace identifikace poruchy je problém rozlišení chyby (*confounding syndrome*), kdy z množiny nezávislých chyb získáme identické syndromy. Jako příklad lze opět použít Test Set v tabulce 2.1. Předpokládejme, že model chování zkratovaných vodičů je typu OR. Pokud jsou zkratovány vodiče n_4 a n_{10} ale také n_6 a n_8, výsledný SRV je ve všech čtyřech případech roven 1110. Nejsme tedy schopni rozlišit, zda se jedná o dva nezávislé zkraty vodičů n_4, n_{10} a n_6, n_8, anebo jeden zkrat čtyř vodičů n_4, n_6, n_8 a n_{10}. [4]

Dále zavedeme pojem neidentifikovatelné poruchy vodiče. Poruchu označíme za neidentifikovatelnou, pokud neexistuje Test Set a algoritmus takový, že by při aplikaci na síť propojů bylo možné takovou poruchu odhalit. [8] Na obrázku 2.1 jsou znázorněny dva typické příklady neidentifikovatelné chyby. Pokud jsou dva vodiče zkratovány a jeden je navíc přerušen tak, jak je vidět na obrázku 2.1 vlevo, přerušení vodiče nelze díky zkratu odhalit. Také pokud jsou dva vodiče zkratovány a oba dva přerušeny tak, jako na obrázku 2.1 vpravo, nelze díky přerušení vodičů odhalit zkrat.



Obrázek 2.1: Příklady neidentifikovatelné poruchy.

Metody a algoritmy pro testování propojů lze rozdělit do několika kategorií podle toho, jakou mají diagnostickou schopnost, která je známa pod pojmem *diagnostické rozlišení* (DR, Diagnostic Resolution). Následující výčet zahrnuje šest základních kategorií, seřazených vzestupně podle diagnostického rozlišení [8].

DR1 Rozhodne, zda je množina vodičů bez poruchy.

DR2 Identifikuje všechny vodiče s poruchou.

DR3 Pro každý vodič rozhodne, zda je bez poruchy, bez jakékoli informace o ostatních vodičích.

DR4 Identifikuje všechny vodiče s poruchou. Navíc na sběrnici bez zkratů zjistí přítomnost přerušených vodičů. Na sběrnici bez přerušených vodičů identifikují všechny zkratované vodiče.

DR5 Identifikuje všechny vodiče s poruchou. Navíc identifikuje všechny poruchy, které lze diagnostikovat (jsou identifikovatelné).

DR6 Identifikuje všechny vodiče s poruchou. Navíc identifikuje všechny zkratované a přerušené vodiče.

Metody s diagnostickým rozlišením DR1 jsou schopny rozhodnout, zda je množina propojů (sběrnice) bez poruchy či nikoli. DR2 již identifikují všechny chybné vodiče. Neříkají ale nic o povaze poruchy. DR3 se od DR2 liší tím, že pro rozhodnutí o poruchovosti vodiče jim stačí pouze SRV onoho konkrétního vodiče. Vzhledem ke skutečnosti, že některé poruchy nejsou identifikovatelné (bez opravy), není možné dosáhnout diagnostického rozlišení DR6.

Metody pro testování spojů lze rozdělit na statické, neboli jednokrokové, a dynamické, také označované jako adaptivní či dvoukrokové.

2.2 Statické metody

Při použití statické metody testování spojů se na množinu vodičů aplikuje celý Test Set a poté jsou vyhodnoceny výsledky testu. Algoritmus statického testu by se dal popsat následujícími kroky:

1. Generování testovacích vektorů,
2. Přiřazení vektorů PTV testovaným vodičům,
3. Uchování SRV vektoru,
4. Opakování předchozích dvou kroků pro všechny generované PTV vektory,
5. Vyhodnocení získaných SRV vektorů.

Následují příklady známých statických algoritmů pro testování propojů, které dosahují diagnostického stupně DR5.

2.2.1 Walking one's (zero's) algorithm

Název algoritmu se často překládá jako Algoritmus rotující jedničky (nuly). Testovací vektory algoritmu rotující jedničky vzniknou umístěním jedničky do diagonály matice a přidáním nulového vektoru, jak je znázorněno v tabulce 2.2. Analogicky u rotující nuly. Složitost algoritmu je lineární a je bezesporu nejjednodušší na automatické generování.

Aby test splňoval požadavky kategorie DR5, je nutné aplikovat algoritmus rotující jedničky a nuly sekvenčně za sebou. Algoritmus rotující jedničky totiž umí identifikovat

Vodič	Vstupní vektory				
n_1	0	0	0	0	1
n_2	0	0	0	1	0
n_3	0	0	1	0	0
n_4	0	1	0	0	0

Tabulka 2.2: Walking one's algorithm.

všechny poruchy na vodičích pouze pokud je model chování při zkratu vodičů typu OR. Pokud neznáme model chování při zkratu, je nutné použít navíc algoritmus rotující nuly. Takto vzniklý algoritmus byl pojmenován jako *Universal Test Set*. Délka algoritmu poté narůstá na $2(n+1)$. Další vybrané algoritmy mají stejné identifikační schopnosti jako algoritmus rotující jedničky a nuly, liší se pouze délkou algoritmu, tedy délkou testovacích vektorů STV.

Vodič	Vstupní vektory									
n_1	1	1	1	1	0	0	0	0	0	1
n_2	1	1	1	0	1	0	0	0	1	0
n_3	1	1	0	1	1	0	0	1	0	0
n_4	1	0	1	1	1	0	1	0	0	0

Tabulka 2.3: Universal Test Set.

2.2.2 $n+1$ algorithm

Tento algoritmus [11] byl vyvinut s cílem minimalizace doby potřebné pro úplnou diagnózu (kategorie DR5) propojů. Používá testovací vektory STV menší délky než algoritmus Universal Test Set (UTS). Délka STV vektorů je dokonce poloviční než u algoritmu UTS. V literatuře [11] je uveden formální důkaz příslušnosti algoritmu $n+1$ do kategorie DR5. Hlavním přínosem algoritmu je redukce doby trvání testu při zachování malé náročnosti automatického generování testovacích vektorů.

Vodič	Vstupní vektory				
n_1	0	0	0	0	1
n_2	0	0	0	1	1
n_3	0	0	1	1	1
n_4	0	1	1	1	1

Tabulka 2.4: Algoritmus $n+1$.

2.2.3 Interleaved True/Complement test

Algoritmus je založený na principu, který představil Kautz již v r. 1974. A sice na přiřazení jednoznačného kódového vektoru jednotlivým vodičům. Pokud je na některém vodiči porucha, bude kódové slovo po přenosu porušené. Takovýto test (nazvaný Counting Sequence)

má délku $\ln n$ při počtu n vodičů. Je schopný detekovat zkraty a přerušené vodiče, ale není schopen poruchy lokalizovat – provést úplnou diagnózu. Bylo navrženo několik modifikací tohoto algoritmu (viz [7]) a jednou z nich je také Interleaved True/Complement test.

Vodič	Vstupní vektory	
n_1	0	0
n_2	0	1
n_3	1	0
n_4	1	1

Tabulka 2.5: Algoritmus Counting Sequence.

Způsob modifikace původního algoritmu je zřejmý již z názvu. Pojem True/Complement značí, že původní hodnoty vektorů Counting Sequence se v algoritmu vyskytují navíc ještě v invertované podobě. To proto, aby byl v každém vektoru vyvážený počet jedniček a nul. Takto modifikovaný algoritmus má výrazně lepší detekční schopnosti, ale stále trpí na aliasing poruch. Pokud bude ovšem zajištěno, že se budou v každém vektoru alespoň jednou vyskytovat oba dva přechody logických úrovní ($0 \rightarrow 1$ a $1 \rightarrow 0$), lze se problému aliasingu vyhnout. Testovací vektory PTV je tedy nutné přeskupit tak, aby vždy po sobě jdoucí vektory PTV měly hammingovu vzdálenost alespoň 2. Odtud tedy pojem Interleaved. Výsledné vektory algoritmu jsou znázorněny v tabulce 2.6. [7]

Vodič	Vstupní vektory			
n_1	1	0	1	0
n_2	1	0	0	1
n_3	0	1	1	0
n_4	0	1	0	1

Tabulka 2.6: Interleaved True/Complement Test.

Takto zkonstruovaný algoritmus dosahuje opět diagnostické úrovně DR5, zatímco délka testu je pouze $2\lceil \log_2(N) \rceil$.

2.3 Dynamické metody

Dynamické (adaptivní) metody testování propojů jsou založené na zpětné vazbě sítě propojů. Testovací vektory jsou generovány v závislosti na odezvě propojů na předchozí testovací vektory. Často jsou označované jako metody dvoukrokové, protože testování a vyhodnocování poruch probíhá ve dvou krocích:

1. Generování statických testovacích vektorů a otestování sítě propojů,
2. Na základě odezvy systému propojů na statické vektory jsou vygenerovány další testovací vektory.

Konečné vyhodnocení je možné až po získání odezvy na obě sady testovacích vektorů.

Označení adaptivní algoritmus plyne ze schopnosti přizpůsobit se odezvám na počáteční testovací vektory. Cílem této skupiny algoritmů je opět minimalizace délky testovacích

vektorů, maximální diagnostická schopnost v porovnání se statickými (jednokrokovými) algoritmy zůstává stejná.

Vzhledem k relativně složité implementaci adaptivních algoritmů v hardware se tato práce zaměřuje na statické testovací metody.

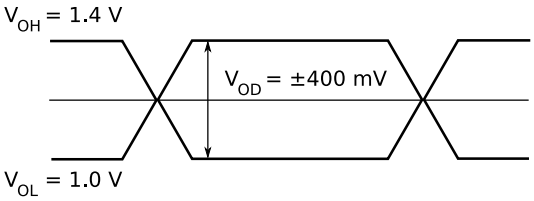
2.4 Technologie diferenciálních propojů LVDS

Se zvyšujícím se výkonem procesorů, multimédií a obecně zařízení zpracovávajících data, vzrůstají požadavky na rychlost přenosu dat. LVDS (Low Voltage Differencial Signaling) je aplikačně nezávislý standard pro rychlý přenos dat na úrovni point-to-point.

LVDS definují dva standardy ANSI/TIA/EIA-644 a IEEE 1596.3 SCI-LVDS. První zmiňovaný je považován za obecnější, jelikož definuje pouze vstupní charakteristiku vysílače a výstupní charakteristiku přijímače a nezahrnuje tedy specifikaci protokolů pro přenos dat [6].

LVDS technologie používá diferenciální schéma přenosu dat. Diferenciální schéma je oproti jednoduchému (single-ended) schématu přenosu méně náchylné na běžné rušení prostředí. Běžný šum prostředí pouze přispívá k modulaci diferenciálního signálu a je přijímačem ignorován.

Tato technologie je také nezávislá na velikosti napájecího napětí. Při libovolné velikosti napájecího napětí je udržován stále stejný odstup jednotlivých úrovní signálu. Aby bylo možné dosáhnout velkých přenosových rychlostí, musí být odstup jednotlivých úrovní signálu malý. Standardem udávaná teoretická maximální přenosová rychlost jednoho páru je 1.9 Gbit/s a doporučená mez je 655 Mbit/s. Avšak samotná přenosová rychlost je závislá také na propojovacím médiu a jeho délce. Na obrázku 2.2 jsou znázorněny napěťové úrovně - je vidět, že rozdíl napěťových úrovní je 400 mV.



Obrázek 2.2: Napěťové úrovně LVDS.

Pokud koncové zařízení na přijímací straně propoje (receiver) podporuje tzv. failsave mód, je zaručeno, že hodnota výstupního signálu bude známa i za některých chybových stavů. Konkrétně se jedná o situace, kdy jsou vstupy přijímače nezapojeny, zkratovány, nebo pokud není dodáváno napětí vysílačem. V uvedených případech je výstupní hodnota přijímače rovna napěťové úrovni logické 1.

2.4.1 Metody testování

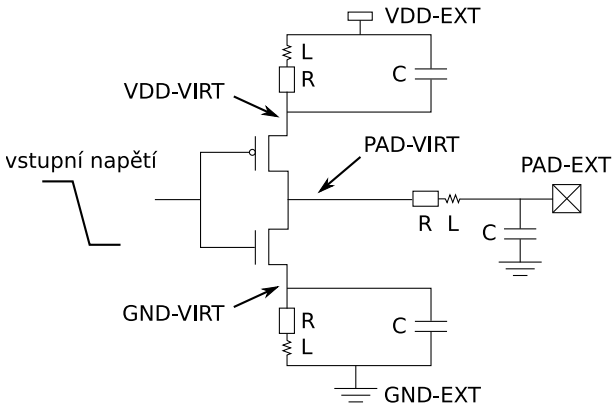
Pro testování LVDS spojů se hojně užívá metoda Bit Error Rate Testing (BERT) [6]. Hodnotu Bit Error Rate určuje počet detekovaných chyb při přenosu za vhodně dlouhý časový úsek. Přesněji BER = (počet chybných bitů)/(celkový počet přenesených bitů). Běžný výklad hodnoty BER je následující: $1 \times 10^{-14} \Rightarrow$ jedna nebo méně chyb v 10^{14} přenesených bitech.

Pro generování bitových vektorů, které jsou při testu použity, se používají pseudonáhodné generátory. Je vhodné poznamenat, že metoda testování BERT je časově náročná. Pokud je přenosová kapacita spoje 50 Mbit/s a chceme dosáhnout BER 1×10^{-14} nebo lepší, je doba potřebná k běhu testu 23,15 dní (2 000 000 vteřin).

2.5 Ground Bounce

Ground Bounce je termín označující jev oscilace napětí, který způsobuje nestabilitu v chování integrovaného obvodu. Oscilace napětí je způsobena přítomností parazitních indukčností a kapacit na přívodu napájecího napětí a uzemnění integrovaného obvodu.

Tento jev lze ilustrovat na obrázku 2.3, na kterém je znázorněn jednoduchý invertující obvod s externím výstupem (pad). Zde je přívod napájecího napětí (VDD) a uzemnění (GND) rozdělen do dvou úrovní – vnější připojení (EXT) a vnitřní připojení (VIRT). Spojení mezi vnějšími a vnitřními napětími lze znázornit pomocí RLC obvodu (parazitní indukčnost a kapacita). Jako příklad možného chybného chování obvodu uvažujme změnu hodnoty na vstupu invertoru z logické 1 do logické 0. Proud odebíraný z VDD způsobí oscilaci RLC obvodu. VDD-EXT má stálou hodnotu, ale VDD-VIRT po krátkou dobu osciluje. To způsobuje nestabilitu na PAD-VIRT a následně PAD-EXT. Pokud takové oscilace přetrvají delší dobu, můžou způsobit čtení chybné logické hodnoty z PAD-EXT a tedy chybu obvodu. Nežádoucí oscilace napětí se projeví tím více, čím větší je frekvence integrovaného obvodu. Se zvyšující se rychlostí integrovaných obvodů se tomuto problému věnuje více pozornosti.



Obrázek 2.3: Znázornění parazitních indukčností a kapacit na přívodech napětí integrovaného obvodu.

Pojem Ground Bounce se také spojuje s parazitními kapacitami na sousedících vodičích – propojích. Napětí na vodiči může být při vysoké frekvenci přepínání ovlivněno parazitními kapacitami na okolních vodičích do té míry, že se na krátkou dobu změní napětí na vodiči. Při vysokých přenosových rychlostech to může znamenat opět přečtení chybné hodnoty z vodiče. [10]

Jev Ground Bounce lze pozorovat zejména na výstupech integrovaného obvodu, ale při použití velmi vysokých frekvencí se může objevit i uvnitř. U některých součástek (např. technologie Advanced Shottky TTL [3]) je definován tzv. práh přepínání – maximální počet výstupů integrovaného obvodu, jejichž logickou hodnotu lze naráz změnit. V případě pro-

pojů je takovéto omezení často nežádoucí, protože chceme být schopni přenášet jakékoli hodnoty nezávisle na hodnotě předchozí. V této práci se tedy zaměříme na zjištění, zda k tomuto jevu může docházet, popřípadě na jeho vyvolání. Jev Ground Bounce se u propojů projevuje s rostoucím počtem zároveň přepínaných vodičů. Pokud se tedy budou po propoji posílat střídavě testovací vektory se všemi bity v log. 1 a se všemi bity log. 0, je pravděpodobnost objevení Ground Bounce největší.

2.5.1 Dopad na diferenciální propoje

Jak lze vyvodit z definice propoje typu LVDS, díky diferenciálním hodnotám signálu je tento typ propoje velmi tolerantní vůči různým typům šumu okolí, projevujícímu se jako nárůst napětí na obou diferenciálních vodičích. Uvážíme–li jeden diferenciální propoj, je jen malá možnost ovlivnění logické hodnoty spoje jevem Ground Bounce. Pokud ovšem máme sběrnicové zapojení, tedy několik párů vodičů, kde každý z nich přenáší jeden diferenciální signál, nelze zaručit konstantní nárůst napětí na všech vodičích. Proto je důležité zabývat se tímto jevem i při použití technologie LVDS, i když je obecně známo, že LVDS jevem Ground Bounce netrpí. Tím spíše, když chceme přenášet data na vysoké frekvenci. Vliv jevu Ground Bounce je pak určen do jisté míry rychlostí přenosu a použitou technologií (materiálem).

Kapitola 3

Metody testování pamětí

Testování pamětí je jednou ze základních oblastí diagnostiky. První testy pamětí nezohledňovaly výrobní technologii ani organizaci dat na čipu. Později byla prokázána souvislost mezi strukturou paměťového čipu a poruchami pamětí. Paměť se začala zkoumat z hlediska vlastního umístění tranzistorů a možnosti vzájemného ovlivňování. Byly objeveny další typy poruch pamětí a navrženy možnosti jejich detekce.

3.1 Poruchy pamětí

Poruchy pamětí s náhodným přístupem (RAM) lze rozdělit do několika kategorií podle místa jejich vzniku:

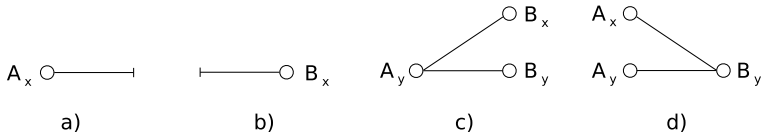
- adresový dekodér,
- datové pole,
- datové cesty,
- řídicí signály.

3.1.1 Poruchy adresového dekodéru

Při poruše adresového dekodéru může nastat následující situace:

- pro některou adresu se nevybere žádná paměťová buňka (3.1a),
- některá paměťová buňka není přístupná (3.1b),
- pro jednu adresu je vybráno několik paměťových buněk (3.1c),
- k jedné paměťové buňce lze přistoupit více adresami (3.1d).

Z obrázku 3.1 je patrná závislost mezi poruchami a, c a b, d – tyto typy poruch se vyskytují většinou současně.



Obrázek 3.1: Poruchy adresového dekodéru.

3.1.2 Poruchy paměťového pole

Poruchy paměťového pole lze rozdělit podle počtu operací s pamětí, které je potřeba provést k vyvolání poruchy. Statické poruchy jsou vyvolány provedením nejvýše jedné paměťové operace. Mezi statické poruchy patří například poruchy typu trvalá 0/1 (stuck-at 0/1) nebo pokud čtení paměťové buňky vyvolá inverzi její hodnoty. V prvním případě není potřeba žádná, v druhém případě jedna paměťová operace pro vyvolání poruchy. Možné typy poruch paměťového pole budou podrobněji popsány dále. Oproti tomu dynamické poruchy jsou vyvolány posloupností několika paměťových operací.

Jak statické, tak dynamické poruchy se dále dělí na single-cell (ovlivňují pouze jednu buňku paměti) a multi-cell (ovlivňuje se několik buněk mezi sebou).

Pokud si paměť představíme jako matici paměťových buněk, můžeme u poruch typu multi-cell rozlišit okolí (sousedství) ovlivňovaných buněk. Existuje několik typů okolí:

- sousední buňky v řádku a/nebo ve sloupci,
- sousední buňky v řádku, ve sloupci a/nebo v diagonálách,
- všechny buňky v řádku nebo ve sloupci,
- všechny buňky paměti.

3.1.3 Poruchy datových cest

Tyto typy chyb mají původ na plošném spoji paměťového modulu. Nejčastěji je to přerušený spoj nebo propojení (zkrat) sousedních vodičů. Pokud to umožňuje rozmístění vodičů na plošném spoji, mohou být datové či adresové vodiče spojeny s vodičem napájecího napětí či uzemnění. Poruchy datových cest patří mezi snadno detekovatelné.

3.1.4 Poruchy řídicích signálů

Obecně nelze určit chování paměti při výskytu tohoto typu poruchy. Záleží na komunikačním protokolu paměti, na typu poruchy a na konkrétním provedení daného čipu. Jiné chování lze například očekávat od paměti synchronní a paměti asynchronní. U synchronní paměti je rozhodující hodnota řídicího signálu v době náběžné hrany hodin. Asynchronní paměť je oproti tomu přímo řízena hodnotou (změnou hodnoty) řídicího signálu. Reakce konkrétní paměti na chyby řídicích signálů by měly být popsány v komunikačním protokolu dané paměti.

3.2 Modely paměťových poruch

K analýze chování chybného paměťového obvodu a vývoji technik pro detekci chyb se používají abstraktní chybové modely. Jejich pomocí lze vytvářet testy, které chybné obvody

rozpoznají. Při modelování chyb pamětí jsou fyzické chyby modelovány jako chyby logické. To je možné díky tomu, že pro každou fyzickou poruchu jsme schopni určit její dopad na logické chování obvodu. Kvůli rozměrům a konstrukci čipu nejsme totiž schopni přesně lokalizovat fyzické chyby a proto je třeba použít test, který je založen na porovnání logických hodnot obvodu.

Funkčnost každé jednotlivé paměťové buňky by měla být ideálně ověřena pro všechny možné kombinace hodnot v ostatních buňkách. Ovšem jen zřídka kdy je to opravdu potřeba, navíc pro paměti s velkou kapacitou je to časově velmi náročné.

3.2.1 Statické poruchy

Statické poruchy paměťového pole jsou takové poruchy, které lze vyvolat jednou a méně operacemi s pamětí. Rozlišují se chyby typu single-cell a multi-cell (většinou jen two-cell). U multi-cell poruch definujeme dva typy buněk – řídicí a závislé.

State fault (SF) Buňka je ve stavu trvalé 0 nebo trvalé 1 ihned po inicializaci paměti.

Transition fault (TF) Buňka je schopna uchovávat libovolnou hodnotu, nezvládne však přechod z určité logické úrovně do druhé (buď z 0 do 1 nebo z 1 do 0). Dejme tomu, že buňka neumí vykonat přechod z 0 do 1. Pokud je pak po inicializaci paměti v buňce hodnota 1, zápis hodnoty 0 proběhne v pořádku, ovšem tato hodnota už v buňce zůstane.

Write disturb fault (WDF) Zápis stejné hodnoty do buňky vede ke změně hodnoty. Například zápisem hodnoty 0 do buňky, která obsahuje 0, vyvoláme inverzi hodnoty na 1.

Read destructive fault (RDF) Operace čtení invertuje hodnotu uloženou v buňce. Přečte se změněná hodnota.

Deceptive read destructive fault (DRDF) Operace čtení vrátí správnou hodnotu, ovšem hodnota v buňce se invertuje.

Incorrect read fault (IRF) Operace čtení nezmění hodnotu uloženou v buňce, přečte se však její invertovaná hodnota.

State coupling fault (CFst) Je-li v řídící buňce jistá hodnota, do závislé buňky je vnucena určitá hodnota. Jinými slovy, hodnota závislé buňky přímo závisí na hodnotě řídící buňky a to ihned po inicializaci paměti.

Disturb coupling fault (CFdst) Operace (zápis nebo čtení) v řídicí buňce změní hodnotu v závislé buňce.

Transition coupling fault (CFtr) Bude-li v řídící buňce určitá hodnota, nepovede se invertující zápis do závislé buňky.

Write destructive coupling fault (CFwd) Bude-li v řídící buňce určitá hodnota, dojde k inverzi hodnoty v závislé buňce při jakémkoli zápisu do této závislé buňky.

Read destructive coupling fault (CFrd) Bude-li v řídící buňce určitá hodnota, pak čtení v závislé buňce vyvolá inverzi hodnoty a tato hodnota se přečte.

Deceptive read destructive coupling fault (CFdrd) Bude-li v řídící buňce určitá hodnota, operace čtení v závislé buňce vyvolá inverzi hodnoty, ale přečte se správná hodnota.

Incorrect read coupling fault (CFir) Bude-li v řídící buňce určitá hodnota, operace čtení v závislé buňce nezmění její hodnotu, ale přečte se invertovaná hodnota.

3.2.2 Dynamické poruchy

Výzkum zaměřený na dynamické poruchy ukazuje, že tento typ poruch se často vyskytuje u dynamických pamětí DRAM [5]. Jejich existence se nejčastěji vysvětluje propojením adresového či datového vodiče přes parazitní impedanci přímo s paměťovou kapacitou. Pak opakované impulsy na vodiči po malých kvantech elektrického náboje postupně mění (snižují/zvyšují) hodnotu napětí na kapacitě, až tato změní svůj stav.

Dynamic read destructive fault (dRDF) Operace čtení bezprostředně po operaci čtení nebo zápisu invertuje hodnotu v buňce a tato invertovaná hodnota je přečtena.

Dynamic deceptive read destructive fault (dDRDF) Operace čtení bezprostředně po operaci čtení nebo zápisu invertuje hodnotu v buňce, je však přečtena správná hodnota.

Dynamic incorrect read fault (dIRF) Operace čtení bezprostředně po operaci čtení nebo zápisu vrátí invertovanou hodnotu, hodnota v buňce se ale nemění.

Dynamic transition fault (dTF) Invertující zápisová operace bezprostředně po operaci čtení nebo zápisu selže, hodnota buňky se tedy nemění.

Dynamic write disturb fault (dWDF) Neinvertující zápisová operace bezprostředně po operaci čtení nebo zápisu vede k inverzi hodnoty v buňce.

Two-cell chyb je teoreticky 242 a dělí se do čtyř kategorií podle posloupnosti aplikace operací na buňky. Buňka, která vyvolá chybu, se označuje S_a (*aggressor*). Buňka, ve které se chyba projeví, se značí S_v (*victim*). K vyvolání dynamické two-cell chyby je potřeba aplikovat posloupnost operací buď pouze na některou z buněk S_a a S_v nebo na obě buňky. Podle počtu a pořadí operací aplikovaných na každou z buněk se tyto chyby dělí do čtyř kategorií:

1. S_{aa}: dvě operace se aplikují na buňku S_a.
2. S_{vv}: dvě operace se aplikují na buňku S_v.
3. S_{av}: první operace se aplikuje na buňku S_a, druhá na buňku S_v.
4. S_{va}: první operace se aplikuje na buňku S_v, druhá na buňku S_a.

3.3 Testy March

Pro testování pamětí typu RAM se používají zejména testy March. Dokáží detekovat mnoho typů poruch pamětí (záleží na konkrétním algoritmu testu) a mají lineární časovou složitost.

March test se skládá z konečné posloupnosti kroků March testu. Krok March testu je konečná posloupnost operací, které se aplikují na (každou) paměťovou buňku před

přechodem na další buňku. Pořadí procházení paměťových buněk je dáno pořadím testovaných adres, které může být vzestupné (značí se $\Uparrow$) nebo sestupné (značí se $\Downarrow$). Jestliže nezáleží na pořadí průchodu paměťových buněk (značí se $\Updownarrow$), použije se v testu vzestupné nebo sestupné pořadí. Základní operace prováděné v testu jsou:

w0 zápis hodnoty 0 do buňky,

w1 zápis hodnoty 1 do buňky,

r0 přečtení buňky s tím, že se očekává hodnota 0,

r1 přečtení buňky s tím, že se očekává hodnota 1.

Celý March test je v zápise ohraničený hranatými závorkami, přičemž kroky March testu jsou ohraničeny kulatými závorkami a jsou od sebe odděleny středníkem. Jednotlivé operace jsou potom odděleny čárkami.

Například test MATS+ by se zapsal takto: $\{\Updownarrow (w0); \Uparrow (r0, w1); \Downarrow (r1, w0)\}$. V prvním kroku se v jakémkoli pořadí (vzestupném nebo sestupném) naplní celá paměť nulami. Ve druhém kroku se postupuje od nejnižší adresy paměti. Z každé buňky se vyčte hodnota a zapíše se hodnota 1 před přechodem na další adresu. V posledním kroku se postupuje od nejvyšší adresy. Nejdříve se vyčte hodnota buňky a poté se zapíše hodnota 0 před přechodem na další buňku.

3.3.1 Testy March pro slovně orientované paměti

Slovně orientované paměti jsou takové paměti, které obsahují více bitů v jednom slově (většinou je počet bitů ve slově roven mocnině dvou). Operace čtení přečte najednou všechny bity slova, operace zápisu zapíše do všech bitů hodnoty, které jsou na sobě v principu nezávislé.

Většina existujících algoritmů testů pamětí je navržena s předpokladem, že paměť je bitově orientovaná, tzn. že čtecí nebo zápisová operace ovlivní pouze jeden bit paměti (z historických důvodů). Slovně orientované paměti se pak tradičně testují opakovaným použitím testů pro bitově orientované paměti za použití různých datových vzorů (v závislosti na použitém testu). Tento přístup je ale časově náročný a schopnost detekce chyb je omezená [12].

V literatuře [12] je popsán postup konverze March testů pro bitově orientované paměti na slovně orientované. Jedná se o připojení navrženého March testu pro detekci chyb uvnitř slova k vybranému March testu pro detekci chyb mezi slovy (klasické March testy pro bitově orientované paměti). Algoritmus dokáže detekovat všechny chyby typu *CFst*, *CFin*, *CFid* a *CFds* mezi dvěma buňkami (two-cell) uvnitř slova. Algoritmus vychází z dvoubitové posloupnosti datových vzorů pozadí (datová báze):

$$\mathrm{S}_t = S_{00}, S_{11}, S_{00}, S_{01}, S_{10}, S_{01}$$

a posloupnosti čtecích a zápisových operací:

$$\Omega_t = w11, r11, r11, w00, r00, r00, w01, w10, r10, r10, w01, r01, r01.$$

Rozšíření datového vzoru ze dvou bitů na B bitů lze provést podle následujících kroků:

1. Na všechny dvojice buněk (c_i, c_{i+1}) aplikujeme dvoubitový datový vzorek. To může být provedeno aplikací sekvence S_t.

2. Na všechny dvojice buněk (c_i, c_{i+2}) aplikujeme sekvenci $01, 10, 01$. Tato sekvence je dostačující, protože sekvence $00, 11, 00$ již byla aplikována v předchozím kroku.

3. Na všechny dvojice buněk (c_i, c_{i+4}) aplikujeme sekvenci $01, 10, 01$.
   ...

$\log_2 B$. Na všechny dvojice buněk $(c_i, c_{i+2^{\log_2 B-1}})$ aplikujeme sekvenci $01, 10, 01$.

Obdobně je možné rozšířit sekvenci operací ze dvou na B bitů. V literatuře [12] je dále probírána problematika optimalizace takto vytvořených testů March pro slovně orientované paměti, která zde pro stručnost není podrobněji popsána.

3.3.2 Testy March a dynamické chyby pamětí RAM

Studie chování chybných pamětí vede k definici chybových modelů. Většina teoretických prací se ovšem věnuje pouze statickým chybám. Bylo zjištěno [5], že u pamětí typu DRAM se objevují také dynamické chyby. Většina testů pamětí RAM je navržena k detekování statických chyb a tudíž nemusí detekovat chyby dynamické.

V tabulce 3.1 je znázorněno pokrytí single-cell dynamických chyb běžnými testy March [5]. Pokrytí two-cell dynamických chyb těmito testy je obdobné, jako u single-cell dynamických chyb. Je zřejmé, že klasické testy March, navržené pro detekci statických chyb, nedokáží detekovat všechny dynamické chyby. Pro detekci dynamických chyb se proto používají speciální March testy.

Chyba	MATS+	March C-	March B	PMOVI	March U	March SR	March LA
dRDF	0%	0%	50%	50%	50%	50%	50%
dDRDF	0%	0%	0%	50%	0%	0%	50%
dIRF	0%	0%	50%	50%	50%	50%	50%

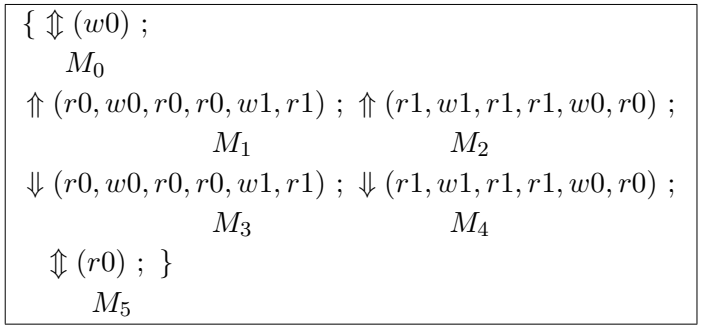
Tabulka 3.1: Pokrytí single-cell dynamických chyb March testy

Pro detekci single-cell dynamických chyb byl vyvinut test March RAW1 [5] (read-after-write), algoritmus je popsán na obrázku 3.2. Test má délku $13n$ a dokáže detekovat všechny zmiňované single-cell dynamické chyby. Jelikož provádí nejvýše dvě operace v každém kroku testu, detekuje dynamické chyby vyvolané maximálně dvěma operacemi.

$$\{ \Updownarrow (w0) \; ; \; \underset{M_0}{\Updownarrow (w0, r0)} \; ; \; \underset{M_1}{\Updownarrow (r0)} \; ; \; \underset{M_2}{\Updownarrow (w1, r1)} \; ; \; \underset{M_3}{\Updownarrow (r1)} \; ; \; \underset{M_4}{\Updownarrow (w1, r1)} \; ; \; \underset{M_5}{\Updownarrow (r1)} \; ; \; \underset{M_6}{\Updownarrow (w0, r0)} \; ; \; \underset{M_7}{\Updownarrow (r0)} \; ; \; \underset{M_8}{} \; \}$$

Obrázek 3.2: Test March RAW1

Test March RAW, jehož algoritmus je popsán na obrázku 3.3, byl navržen pro detekci two-cell dynamických chyb. Délka testu je $26n$, přičemž tento test dokáže detekovat některé dynamické chyby vyvolané více než dvěma operacemi. Test March RAW detekuje jak two-cell, tak single-cell dynamické chyby, proto má širší uplatnění než test March RAW1.



Obrázek 3.3: Test March RAW

3.4 Paměť typu DDR2 DRAM

V dnešní době jsou nejčastěji ve výpočetní technice využívány dynamické paměti (DRAM). DRAM paměti nabízejí velkou kapacitu a používají se většinou jako systémová paměť.

Dynamické paměti ukládají každý bit do samostatného kondenzátoru. Informace uložená v paměti musí být pravidelně obnovována (tzv. refresh cyklus), k tomu účelu bývá paměť vybavena obnovovacím obvodem. Při čtecí operaci se celý řádek, ve kterém leží požadovaná paměťová buňka, přečte přes snímací zesilovač, kde se rozliší uložená hodnota. Následně se hodnota z požadované buňky vystaví na výstup. Poté se celý řádek opět nahraje do paměti, jelikož při čtení byly uložené hodnoty zapomenuty (zničeny). Při zápisové operaci se vyčte celý řádek, požadované místo je připojeno ke snímacím vodičům a je přepsáno novou hodnotou. Poté se celý řádek nahraje zpět do paměti.

DDR (Double Data Rate) je protokol, pomocí něhož probíhají datové přenosy mezi pamětí a řadičem. Při DDR přenosu se data přenáší na vzestupnou i sestupnou hranu hodin, což vede ke zvýšení propustnosti datové sběrnice.

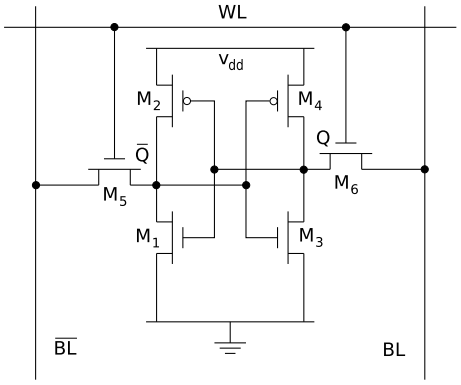
Technologie DDR2 je druhou verzí standardu DDR. Hlavní změnou oproti předchůdci je zvýšení frekvence externí paměťové sběrnice na dvojnásobek, zatímco operační frekvence paměťových buněk je zachována. Paměťové buňky tedy pracují na poloviční frekvenci oproti paměťové sběrnici – za referenční frekvenci paměti se považuje frekvence paměťové sběrnice. Pokud tedy porovnáme paměti DDR2 a DDR pracující na stejné frekvenci, budou mít stejnou propustnost, ale paměť DDR2 bude mít výrazně větší latenci (až 2x). Výhodou technologie DDR2 je bezesporu fakt, že paměťové moduly s touto technologií operují na mnohem vyšší frekvenci, než moduly DDR. Ve výsledku, pokud porovnáme paměť DDR2 pracující na dvojnásobné frekvenci paměti DDR, budou latence pamětí stejné, ale paměť DDR2 bude mít 2x vyšší propustnost.

3.5 Paměť typu QDR SRAM

Statické paměti (SRAM) jsou díky nízké latenci využívány hlavně jako vyrovnávací paměti (cache).

Statické paměti jsou schopny uchovat uloženou informaci po libovolně dlouhou dobu, pokud je ovšem přivedeno napájecí napětí. Nepotřebují tedy žádné obnovovací cykly. K uložení jednoho bitu informace je potřeba alespoň čtyř tranzistorů, které fungují jako klopný obvod (viz schéma 3.4). Symetrická struktura paměti umožňuje rychlé čtení uložené informace. K malé latenci také přispívá fakt, že adresové bity nebývají multiplexované. To umožňuje

paměti přijmout celou adresu najednou.



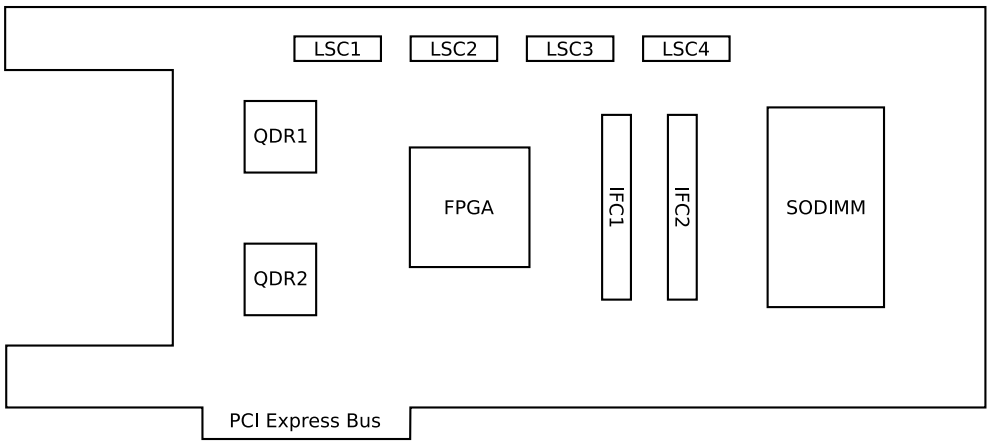
Obrázek 3.4: Schéma buňky CMOS SRAM.

Paměti QDR (Quad Data Rate) jsou optimalizovány pro využití v síťových aplikacích. Zkratka protokolu QDR vyjadřuje až čtyřnásobný přenos dat oproti SDR (Single Data Rate), čehož je dosaženo pomocí nezávislého čtecího a zápisového portu – na obou portech přenos pomocí protokolu DDR. Maximální operační frekvence těchto pamětí se pohybuje okolo 333 MHz (standard QDR2), tudíž nabízejí vysokou propustnost.

Kapitola 4

Testovaná architektura

Testovaná karta byla vyvinuta pro potřeby projektu Liberouter. Karta je osazena FPGA čipem Virtex-5, pro komunikaci se software využívá rozhraní na sběrnici PCI Express x8. Karta obsahuje čtyři low-speed konektory pro komunikaci s pomalejšími zařízeními, dva high-speed konektory pro komunikaci s přídavnou kartou, dále dvě statické paměti QDRII SRAM a SODIMM slot pro dynamickou paměť DDR2 SDRAM. [2]



Obrázek 4.1: Blokové schéma karty COMBO2.

4.1 High-speed propoje

High-speed (nebo IFC) propoje jsou určeny pro rychlý přenos dat mezi kartou COMBO2 a kartou síťových rozhraní. Konektor obsahuje celkem 44 diferenciálních párů s následujícím rozdělením:

- 2 páry pro JTAG rozhraní,
- 8 párů pro přenos dat pomocí RocketIO – propustnost až 20 Gbit/s v obou směrech,
- 36 LVDS párů pro přenos dat – propustnost až 36 Gbit/s v jednom směru,
- 1 pár pro detekci připojení a identifikaci karty.

4.2 Low-speed propoje

Tento typ propoje je určen pro komunikaci s pomalejšími zařízeními, jako například SATA disk apod. Low-speed konektor obsahuje celkem 12 LVDS párů, z toho jsou dva páry rezervované pro hodiny a reset. Pro přenos dat tedy zbývá 10 LVDS párů, z nichž každý může teoreticky dosáhnout přenosové rychlosti až 1 Gbit/s. To znamená přenosovou rychlost 10 Gbit/s jedním směrem a 5 Gbit/s oběma směry pro jeden low-speed konektor.

4.3 Dynamická paměť DDR2 DRAM

Karta je vybavena SODIMM slotem, pomocí kterého lze připojit dynamickou paměť DDR2 SDRAM do velikosti 1 GB. Maximální pracovní frekvence použitého čipu je 267 MHz, šířka datové sběrnice je 64 bitů. Data lze přenášet v režimu burst či pomocí oddělených čtecích a zápisových operací. Teoretická propustnost paměti při 250 MHz je 4 Gbit/s.

4.4 Statická paměť QDR SRAM

Je použit čip firmy Cypress o velikosti 72 Mbit a maximální pracovní frekvenci 250 MHz. Šířka datové sběrnice použité paměti je 18 bitů. Datové přenosy probíhají pouze v režimu burst o délce 4. Teoretická propustnost paměti je 9 Gbit/s.

Kapitola 5

Návrh testu propojů

Navržený postup testování propojů na kartě COMBO2 lze rozdělit do sekvence tří kroků:

1. Diagnostika statických a dynamických chyb – V první řadě je nutné odhalit a identifikovat statické a dynamické poruchy propojů, jako jsou zkraty a přerušené vodiče. K tomu lze využít statické nebo adaptivní algoritmy. Tato práce se zaměřuje na testování sběrnic obsahujících maximálně desítky propojů. Takové množství propojů lze efektivně otestovat statickými algoritmy ve velmi krátkém čase. Z toho důvodu není nutné použití adaptivních algoritmů, které jsou navíc náročnější na implementaci v hardware. K odhalení všech detekovatelných statických a dynamických poruch propojů je nutné použít některý ze statických algoritmů s nejvyšší diagnostickou schopností. V tomto případě byl vybrán algoritmus Interleaved True/Complement Test (zkratka ITCT).

2. Ground Bounce test – U sběrnic diferenciálních propojů se lze setkat s jevem Ground Bounce a je proto nutné možnost jeho výskytu prověřit. Pravděpodobnost výskytu tohoto jevu vzrůstá s přenosovou rychlostí a počtem propojů, současně měnících svoji logickou hodnotu. Do algoritmu testu se tedy přidají datové vektory, jejichž bity obsahují samé 1 a samé 0. Sekvence jedničkových a nulových vektorů se bude několikrát opakovat, aby byly zajištěny všechny možné přechody logických hodnot na každém propoji. Takto bude jednoduše prověřena možnost výskytu jevu Ground Bounce.

3. BER test – Jako měřítko schopnosti přenosu propoje o určité přenosové rychlosti bývá nejčastěji použit Bit Error Rate test. Jako další krok testu propojů na kartě COMBO2 bude tedy použita tato metoda tak, že budou přenášené náhodně vygenerované testovací vektory (viz dále) a bude zaznamenán počet chyb při přenosu. Hodnota BER je následně spočtena z doby trvání testu a počtu chyb, které nastaly při přenosu.

V této kapitole jsou dále uvedena řešení některých obecných problémů testování propojů, vztažená na cílovou architekturu – kartu COMBO2.

5.1 Rychlost přenosu dat

Cílem testování je obecně detekovat a pokud možno identifikovat nalezené poruchy. V případě testování propojů jde o detekci statických a dynamických poruch, které se na daném typu propoje mohou vyskytnout. Důležité je si uvědomit, že projev poruchy – výskyt chyby – může být závislý také na intenzitě přepínání hodnot napětí na vodičích a tedy na rychlosti

přenosu dat. Proto je nutné testovat funkčnost propojů za použití různých pracovních frekvencí. V reálné aplikaci se poté použije nejspíše nejvyšší pracovní frekvence, při které se neprojevila porucha. Pomocí testování je možné takovou frekvenci nalézt.

Pro testování propojů na kartě COMBO2 je požadován synchronní přenos dat, kdy platnost dat na datových vodičích určuje hodnota hodinového signálu na vyhrazeném vodiči. Pro změnu rychlosti přenosu dat je zapotřebí změnit frekvenci hodinového signálu, což lze provést změnou parametrů příslušného hodinového generátoru – v technologii Xilinx Virtex-5 jde o obvod DCM (Digital Clock Manager).

Technologie programovatelných hradlových polí FPGA nabízí několik způsobů provedení změny v obvodu (rekonfigurace). Nejjednodušší je bezpochyby provést potřebnou změnu ve zdrojovém souboru, provést syntézu a vygenerovat novou konfiguraci, kterou lze poté nahrát do FPGA. Tato metoda se nazývá *statická rekonfigurace*. Prakticky to znamená udržovat potřebné množství konfigurací FPGA a podle potřeby je nahrávat do programovatelného obvodu. Zásadní problém je v udržovatelnosti takového řešení (oprava chyby ve zdrojovém kódu atd.).

Další variantou je *dynamická rekonfigurace* FPGA, která probíhá bez přerušení provádění operací programovatelného obvodu. Jedná se o změnu některého logického bloku programovatelného obvodu (částečná rekonfigurace) za chodu. Částečná dynamická rekonfigurace obvodu Virtex-5 může být provedena pomocí standardních rozhraní JTAG, ICAP nebo SelectMAP. Konfiguraci některých logických funkčních bloků uvnitř FPGA Virtex-5 (jako například DCM) lze provést také interně z čipu FPGA pomocí speciálního konfiguračního rozhraní [14].

5.1.1 Rekonfigurace DCM na čipu Xilinx Virtex-5

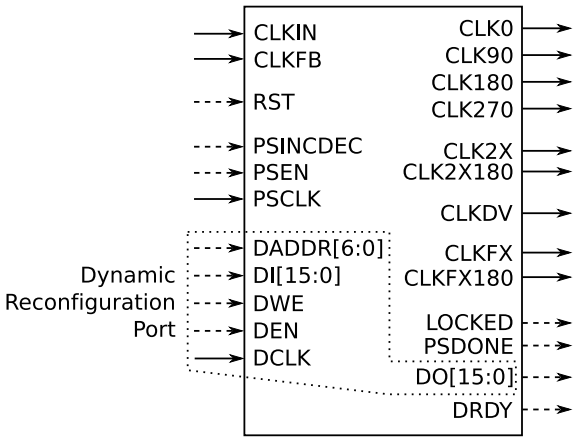
Hodinový signál je na kartě generován pomocí krystalu, který má samozřejmě fixní frekvenci. Pro testování různých přenosových rychlostí propojů je ale třeba frekvenci hodinového signálu měnit podle požadované přenosové rychlosti. Obvod DCM je schopen realizovat frekvenční syntézu hodinového signálu, tedy dle zadaných parametrů změnit frekvenci výstupního hodinového signálu. Frekvenční syntéza je řízena dvěma parametry – Multiply a Divide. Pokud požadujeme výstupní hodinový signál například o poloviční frekvenci, než má signál vstupní, stačí nastavit parametry M a D na hodnoty 0, resp. 1 (od požadovaných hodnot M a D se vždy odečítá jednička).

Technologie Xilinx Virtex-5 umožňuje interní částečnou dynamickou rekonfiguraci obvodu DCM pomocí rozhraní DRP (*Dynamic Reconfiguration Port*). Postup dynamické rekonfigurace je následující:

1. Nejdříve je nutné uvést příslušné DCM do stavu resetu.
2. Pro změnu některé z hodnot M nebo D je nejdříve potřeba přečíst jedno slovo z adresy $0x50$ přes rozhraní DRP.
3. Poté se požadovaný parametr M namaskuje do horního bajtu a parametr D do spodního bajtu vyčteného 16-ti bitového slova.
4. Celé slovo je pak zapsáno zpět na adresu $0x50$ přes rozhraní DRP.
5. Při dynamické změně parametrů ovlivňujících frekvenci výstupního hodinového signálu je někdy potřeba změnit také atributy DFS_FREQUENCY_MODE a DLL_FREQUENCY_MODE. Přesné požadavky pro nastavení těchto parametrů lze nalézt v [13], adresy pro přístup přes DRP lze nalézt v [14].

6. Pro obnovení stavových výstupů DCM obvodu je nutné na závěr provést čtení z adresy $0x00$ rozhraní DRP.

7. Po resetu a po nastavení výstupního signálu LOCKED začne DCM na výstupu CLKFX generovat hodinový signál podle nově nastavených parametrů M a D.



Obrázek 5.1: Signály rozhraní bloku DCM technologie Xilinx Virtex-5 s rozhraním pro dynamickou rekonfiguraci. [15]

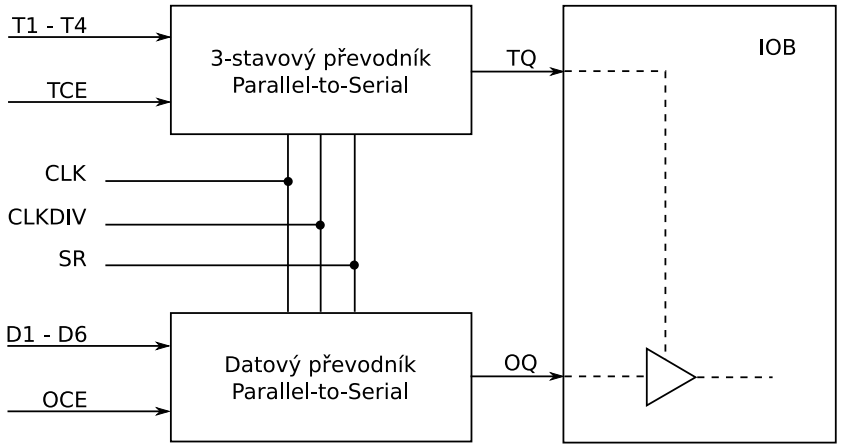
Interní rekonfigurace nabízí velmi efektivní mechanismus změny frekvence hodinového signálu přímo z čipu FPGA, vhodný pro potřeby testování propojů při různých pracovních frekvencích.

5.2 Serializace dat

Teoretická maximální přenosová rychlost jednoho LVDS páru je 1 Gbit/s, což odpovídá frekvenci 1 GHz. Jelikož maximální dosažitelná frekvence obvodu Xilinx Virtex-5 je 550 MHz, je nutné odesílaná data serializovat a přijímaná data opět deserializovat. K tomuto účelu slouží v technologii Virtex-5 speciální obvody OSERDES, resp. ISERDES [15]. Tyto obvody se dají nakonfigurovat na požadovaný poměr serializace, který může být až 6:1 (v základním režimu). Na obrázku 5.2 je znázorněno blokové schéma výstupního serializéru OSERDES. Ten je rozdělen na dva bloky, jeden slouží pro serializaci dat a druhý pro serializaci třístavového řízení.

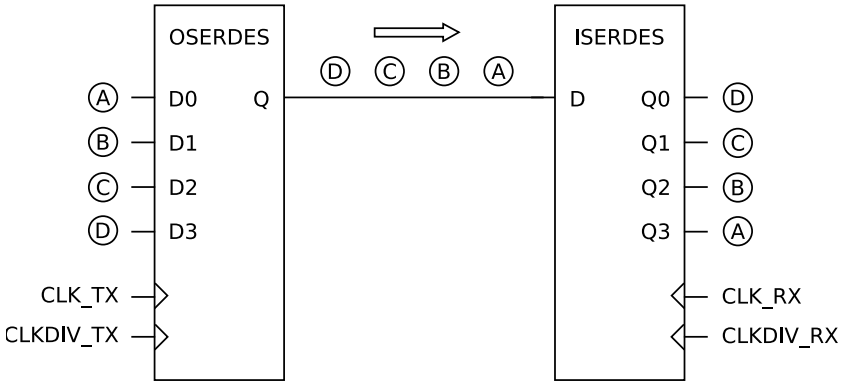
Serializačním obvodům je nutné podle nastaveného poměru připojit na vstup také oba hodinové signály. Tím do designu zavádíme podmínku relace frekvencí hodinových signálů pro odesílání a generování dat. To znamená, že spolu se změnou frekvence odesílání dat se změní frekvence jejich generování. Generování dat musí tím pádem probíhat v jiné hodinové doméně, než ve které pracuje zbytek systému (řízení, komunikace se software). To samé platí pro příjem dat, deserializaci a porovnání s referenčními testovacími vektory.

Na přijímací straně propoje jsou přenesené testovací vektory opět deserializovány, aby mohlo proběhnout porovnání s referenčními vektory. K deserializaci vstupních dat rekonfigurovatelného zařízení slouží obvody ISERDES. Tyto obvody je možné nakonfigurovat na stejný serializační poměr jako výstupní serializéry OSERDES. Je ovšem nutné počítat s tím, že při serializaci a zpětné deserializaci dojde k přehození pořadí vysílaných bitů. Data jsou



Obrázek 5.2: Blokové schéma bloku OSERDES technologie Virtex-5. [15]

serializérem vysílána od nejnižšího bitu, na přijímací straně jsou ale ukládány do nejvyšších bitů. Celý proces přehození pořadí vysílaných bitů je znázorněn na obrázku 5.3.



Obrázek 5.3: Přehození bitů při deserializaci. [15]

5.3 Generování testovacích vektorů

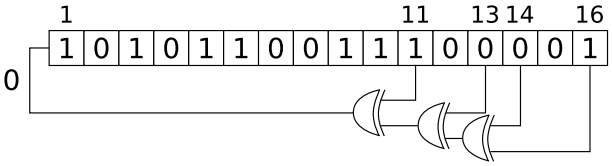
Jak je uvedeno výše, test propojů je rozdělen do tří fází a každá vyžaduje generování specifických testovacích vektorů. Pro test statických a dynamických poruch ITCT je možné testovací vektory uchovat v registrech hardware jako konstanty, jelikož se nemění s časem a díky jejich nízkému počtu tak spotřebují málo zdrojů. Groud Bounce test spočívá pouze ve střídání nulových a jedničkových vektorů, proto je také vhodné tyto hodnoty uchovat v hardware jako konstanty.

Pro BER test je nutné vygenerovat dostatečné množství pseudonáhodných vektorů, aby byl co nejvěrněji simulován reálný provoz. Testovací vektory lze generovat v software nebo přímo v hardware. Po vygenerování vektorů v software se vektory přesunou do hardware a uloží se do blokové paměti. Teprve poté je možné testovací vektory odesílat na výstup propojů. Při generování testovacích vektorů v hardware lze pomocí *Linear Feedback Shift*

Register (LFSR) generovat každý takt jeden testovací vektor. Testovací vektory jsou ihned po vygenerování odeslány na výstup, tudíž není zapotřebí bloková paměť pro jejich uložení.

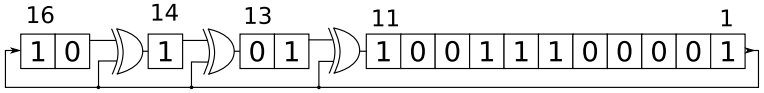
LFSR je posuvný registr, jehož vstupní bit je lineární funkcí jeho předchozího stavu.[1] LFSR je deterministický, tedy posloupnost hodnot výstupů registru je vždy dopředu známa (při znalosti počáteční hodnoty registru). Jelikož je počet stavů (hodnot) registru konečný, výstupní hodnoty registru se periodicky opakují. Délka periody pak určuje kvalitu LFSR jako generátoru pseudonáhodných čísel. Pokud je vhodně zvolena přechodová funkce registru, nabývá registr postupně všech možných hodnot (kromě hodnoty, kdy jsou všechny bity rovné nule). Takový LFSR se nazývá maximální.

Rozlišujeme dva druhy LFSR: Fibonacciho a Galoisův. Na obrázku 5.4 je znázorněn příklad Fibonacciho LFSR. Bity, které jsou použity k výpočtu hodnoty vstupního bitu, se označují jako *tap* nebo *tap sequence*. Tap bity lze vyjádřit jako polynom o základu 2. Tap sekvenci z obrázku 5.4 lze zapsat jako $x^{16}+x^{14}+x^{13}+x^{11}+1$, kde binární operátor + značí funkci XOR. Hodnota 1 neodpovídá žádnému tap bitu, ale hodnotě x^0. Tento nultý bit je vždy přiveden na vstup LFSR. U Fibonacciho LFSR jsou tap bity přivedeny na vstupy kombinační logiky, která vypočítá hodnotu vstupního bitu.



Obrázek 5.4: Fibonacciho LFSR.

Druhým typem je Galoisův LFSR, znázorněný na obrázku 5.5. Hlavní rozdíl oproti Fibonacciho LFSR spočívá v tom, že jednotlivé kombinační elementy jsou vloženy přímo mezi jednotlivé bity registru. Tap bity jsou zde chápány jako výstupy kombinačních elementů. Tento přístup umožňuje paralelní výpočet všech tap bitů, což znamená rychlejší a efektivnější hardwarové zpracování. Galoisův LFSR generuje stejné výstupní vektory, pouze v opačném pořadí než Fibonacciho LFSR.



Obrázek 5.5: Galoisův LFSR.

Pro potřeby testování low-speed a ifc konektorů je potřeba najít generující polynomy LFSR pro různé délky testovacích vektorů. Konkrétně pro low-speed konektory jsou to vektory o délce 4 bity (komunikace oběma směry - full duplex) a 10 bitů (komunikace jedním směrem - simplex). Pro ifc konektory je to pak 16, resp. 34 bitů. Dále je žádoucí, aby byly dané polynomy maximální, tedy aby jimi definovaný LFSR generoval všechny možné hodnoty vektoru. Potřebné LFSR polynomy jsou vypsány tabulce 5.1, počítá se s použitím kombinačních elementů XNOR. Pravdivostní tabulka funkce XNOR je uvedena v tabulce 5.2.

Generované testovací vektory budou před odesláním serializovány. Proto je potřeba použít více LFSR, které budou pracovat paralelně. Testovací vektory, vygenerované LFSR

[1] Za jediné lineární funkce jsou považovány funkce XOR a XNOR.

Délka vektoru	LFSR polynom
4	$x^4 + x^3 + 1$
10	$x^{10} + x^7 + 1$
16	$x^{16} + x^{15} + x^{13} + x^4 + 1$
34	$x^{34} + x^{27} + x^2 + x^1 + 1$

Tabulka 5.1: Maximální LFSR polynomy pro různé délky vektorů. [1]

A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

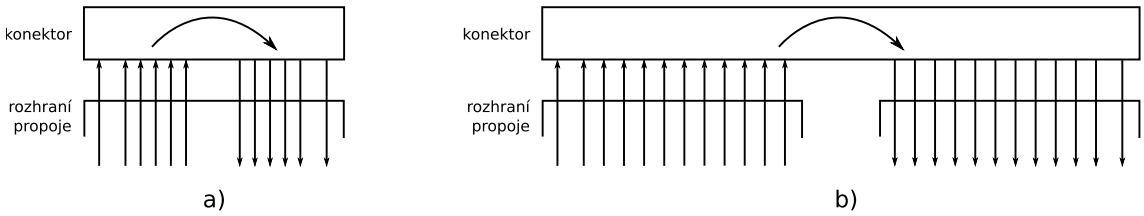
Tabulka 5.2: Pravdivostní tabulka funkce XNOR.

registry v jednom taktu, budou po serializaci odeslány po propoji ihned po sobě. Jednotlivé LFSR by tudíž měly být inicializovány různými počátečními hodnotami, aby nedocházelo k opakování testovacích vektorů jdoucích krátce po sobě.

5.4 Konektory

Pro účely testování propojů na kartě COMBO2 byly navrženy speciální konektory usnadňující diagnostiku. Cílem je možnost testování propojů na kartě samostatně, tj. bez nutnosti druhé karty či jiného zařízení připojeného na rozhraní propoje. Pro každé rozhraní karty jsou použity dva typy konektorů:

1. Konektor propojující páry v rámci jednoho rozhraní (5.6a),
2. Konektor propojující páry dvou rozhraní (5.6b).



Obrázek 5.6: Schéma konektorů rozhraní propoje.

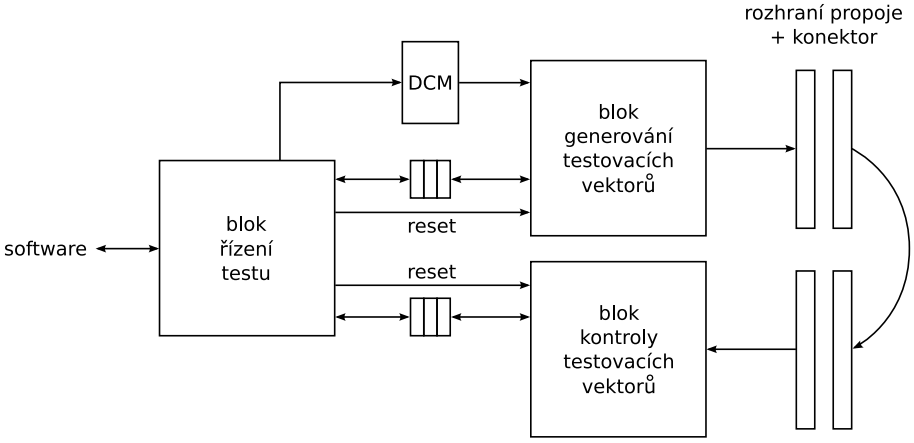
Pokud propojíme konektorem páry jednoho rozhraní, simulujeme tak oboustrannou komunikaci s externím zařízením, kdy jsou pro každý směr vyhrazeny zvláštní páry. Schéma propojení párů takovýmto konektorem je na obrázku 5.6a. Druhý typ konektoru propojuje dvě různá rozhraní. Tento typ zapojení odpovídá jednosměrné komunikaci, např. posílání dat z karty síťových rozhraní na mateřskou kartu (schéma 5.6b).

Navržený test propojů by proto měl být obecný ve smyslu zapojení konkrétních párů daného rozhraní, aby bylo možné testovat oba výše uvedené modely komunikace.

5.5 Architektura

Následující část práce se podrobně zabývá návrhem architektury testu propojů na kartě COMBO2. V návrhu jsou zohledněny obecné teoretické poznatky o testování propojů z kapitoly 2, návrh se také opírá o řešení některých konkrétních problémů ze začátku kapitoly 5. Součástí návrhu architektury jsou schémata a detailní popisy jednotlivých funkčních bloků, s ohledem na následnou implementaci navržené architektury.

Modul testování propojů se skládá ze tří základních funkčních bloků. Jedná se o blok *generování testovacích vektorů*, blok *kontroly testovacích vektorů* a blok *řízení testu*. Na obrázku 5.7 je celkové blokové schéma testu propojů obsahující výše uvedené bloky a prostředky pro komunikaci mezi nimi.



Obrázek 5.7: Blokové schéma testu propojů.

5.5.1 Blok generování testovacích vektorů

Tento funkční celek se stará o generování požadovaných testovacích vektorů a o jejich správné odesílání na rozhraní propojů. Karta COMBO2 obsahuje dva typy propojů (LSC a IFC), každý z nich sestává z různého počtu diferenciálních LVDS párů propojů. Blok generování a odesílání musí být dostatečně obecný, jinými slovy nezávislý na počtu testovaných propojů, aby jej bylo možné efektivně využít na testování rozdílných typů propojů karty COMBO2. Se změnou počtu testovaných propojů se mění např. počet serializačních elementů OSERDES, délka LFSR registrů pro generování testovacích vektorů a také se mění počet a délka testovacích vektorů algoritmu ITCT. Tato variabilita je v návrhu funkčního bloku zahrnuta.

Samotné odesílání dat obstarávají elementy OSERDES, které jsou nutné kvůli požadavku testování vysokých přenosových rychlostí. K elementům OSERDES je přiváděn hodinový signál serializovaných dat CLK_TX o frekvenci, která odpovídá požadované přenosové rychlosti propoje. Například pokud požadujeme přenosovou rychlost 200 Mbit/s, bude mít hodinový signál CLK_TX frekvenci 200 MHz, tedy periodu 5 ns. Dále je k těmto elementům přiváděn hodinový signál neserializovaných dat CLK_GEN. Jeho frekvence závisí na nastaveném serializačním poměru. V tomto případě je použit serializační poměr 4:1 což znamená, že za jeden takt hodin CLK_GEN budou serializovány 4 bity dat. Frekvence CLK_GEN se tedy rovná čtvrtině frekvence CLK_TX. Pro usnadnění serializace budou testovací vektory

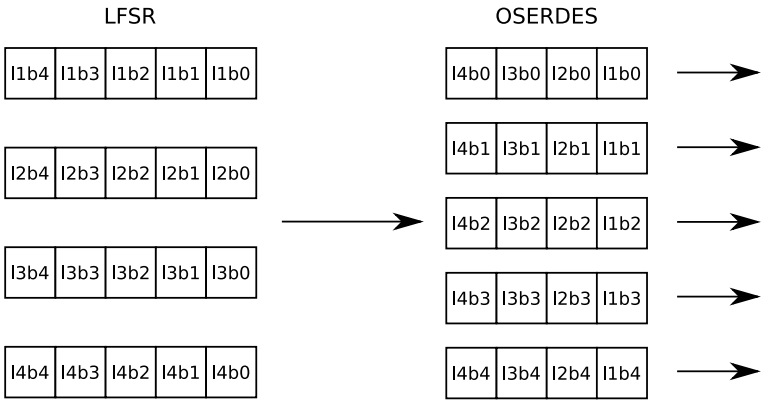
generovány na frekvenci CLK_GEN. Jelikož se ale v průběhu testování bude měnit rychlost odesílání dat, bude se měnit také rychlost generování testovacích vektorů. Za změnu taktu hodin za běhu designu zodpovídá blok řízení testu. Aby bylo možné data jednoduše přijmout, je jeden diferenciální pár vyhrazen právě pro hodinový signál s frekvencí, na které jsou odesílána data.

Pro detekci statických a dynamických chyb propojů je použit algoritmus ITCT. Testovací vektory algoritmu budou uloženy v registrovém poli a odesílány přímo do serializéru v prvním kroku testu. K testovacím vektorům ITCT je přidáno několik nulových a jedničkových vektorů, které slouží k detekci jevu Ground Bounce. V prvním kroku testu jsou odeslány staticky uložené testovací vektory. V tabulce 5.3 jsou uvedeny testovací vektory v případě, že je testováno 10 propojů.

Vodič	Vstupní vektory											
n_1	1	0	1	0	1	0	1	0	1	0	1	0
n_2	1	0	1	0	1	0	1	0	1	0	0	1
n_3	1	0	1	0	1	0	1	0	0	1	1	0
n_4	1	0	1	0	1	0	1	0	0	1	0	1
n_5	1	0	1	0	1	0	0	1	1	0	1	0
n_6	1	0	1	0	1	0	0	1	1	0	0	1
n_7	1	0	1	0	1	0	0	1	0	1	1	0
n_8	1	0	1	0	1	0	0	1	0	1	0	1
n_9	1	0	1	0	0	1	1	0	1	0	1	0
n_{10}	1	0	1	0	0	1	1	0	1	0	0	1

Tabulka 5.3: Testovací vektory algoritmu ITCT (pořadí vektorů zprava) se čtyřmi přidanými vektory pro detekci Ground Bounce. Takto lze otestovat 10 propojů.

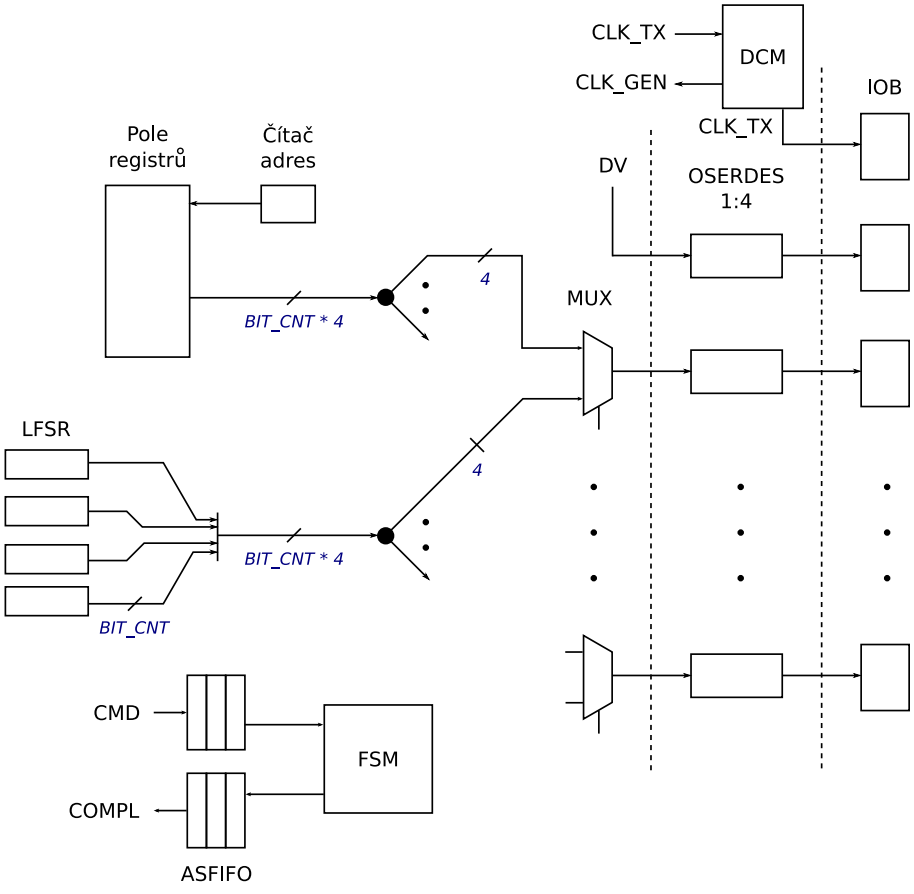
Druhou fázi testu propojů představuje BER test. Jako generátor pseudonáhodných testovacích vektorů bude použit LFSR registr, jehož délka a generující polynom bude určen na základě testovaného počtu propojů. Vhodný výběr polynomu zaručí maximální délku sekvence generovaných vektorů.



Obrázek 5.8: Příklad přeskládání bitů testovacích vektorů pro účely serializace (5 propojů, serializace 4:1).

Díky serializaci testovacích vektorů 4:1 před odesláním je potřeba v jednom taktu hodin vygenerovat 4 testovací vektory, které budou odeslány bezprostředně po sobě. Proces generování je tedy nutné paralelizovat, tj. použít 4 LFSR registry. Všechny použité LFSR registry budou mít stejný polynom (funkci), bude se lišit pouze jejich počáteční hodnota. Takto nebude docházet k odesílání shodných testovacích vektorů v bezprostředně následujících taktech. Dopad na rozložení generátoru je nulový, jelikož na pořadí hodnot vektorů nezáleží. Každý LFSR registr generuje v jednom taktu hodin jeden testovací vektor. Jednotlivé bity vektoru reprezentují hodnoty pro každý propoj na sběrnici. Paralelně vygenerované testovací vektory jsou nejdříve přeskládány tak, aby bylo možné je serializovat. Princip tohoto přeskládání bitů vektorů je znázorněn na obrázku 5.8.

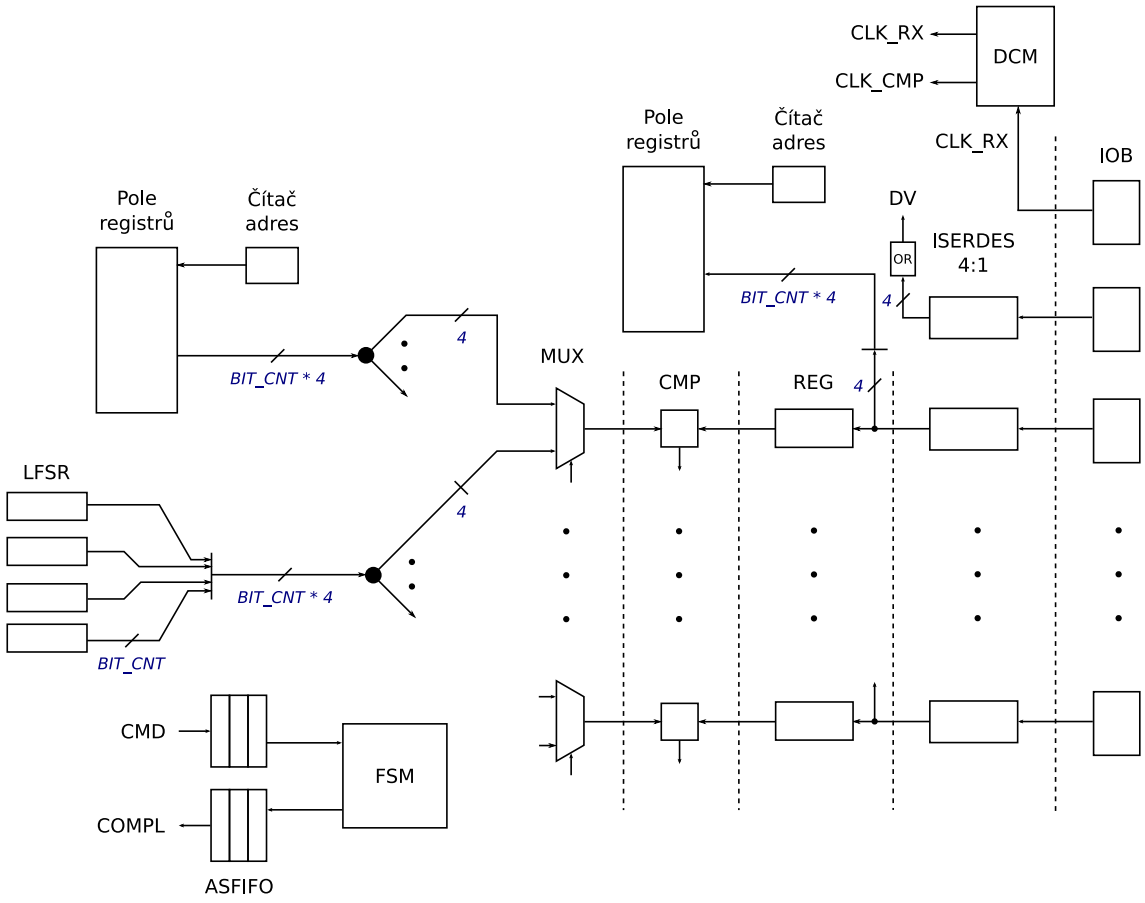
Celkové schéma bloku generování testovacích vektorů je popsáno na obrázku 5.9. V závislosti na fázi testu jsou na vstup serializérů přiváděny testovací vektory uložené v registrovém poli, nebo vygenerované LFSR registry. Dále je spolu s testovacími vektory a hodinovým signálem přenášen signál DV, který udává platnost odesílaných dat. Řízení bloku je kvůli požadavku na relaci vysílacích hodin CLK_TX a hodin CLK_GEN realizováno pomocí asynchronní fronty ASFIFO. Průběh testu řídí konečný automat, který nastavuje řídicí signály jednotlivých komponent na základě požadavků generovaných blokem řízení testu.



Obrázek 5.9: Blokové schéma části testu pro generování a odesílání testovacích vektorů.

5.5.2 Blok kontroly testovacích vektorů

Tento blok se stará o příjem testovacích vektorů, deserializaci a porovnání s referenčními testovacími vektory. Hodinový signál pro přijímání dat je přenášen po jednom vyhrazeném propoji spolu s daty. Z tohoto hodinového signálu CLK_RX je také restaurován hodinový signál CLK_CMP pro celý blok. Frekvence signálu CLK_CMP je rovna frekvenci signálu CLK_RX/4. Frekvence jednotlivých hodinových signálů pro odesílání a příjem dat jsou stejné, ovšem díky nenulovému zpoždění způsobenému přenosem po propoji mají rozdílné fáze. Tedy i fáze hodin CLK_TX a CLK_RX jsou různé a bloky vysílání a přijímání dat jsou v odlišných hodinových doménách. Z tohoto důvodů je nutná asynchronní komunikace mezi jednotlivými funkčními bloky.



Obrázek 5.10: Blokové schéma části testu pro příjem a porovnávání testovacích vektorů.

Deserializaci provádí bloky ISERDES 1:4. Pokud je po deserializaci aktivní signál DV, byly přijaty platné testovací vektory a proběhne jejich porovnání s referenčními testovacími vektory. Ty se opět vyčítají z registrového pole, nebo generují pomocí LFSR registrů v závislosti na kroku testu. Aby bylo možné přijaté testovací vektory porovnat s referenčními, musí být jednotlivé bity referenčního vektoru přerovnány jako důsledek přehození bitů při serializaci a zpětné deserializaci (viz obrázek 5.3). Bylo by možné místo referenčních vektorů přerovnávat přijaté vektory, pak by ale bylo obtížnější identifikovat na kterém propoji se vyskytla chyba.

V jednom taktu jsou porovnány paralelně všechny bity všech čtyř vektorů. Pokud se bity přijatého a referenčního vektoru neshodují, je zaznamenána chyba přenosu. Pro každý testovaný propoj je použit samostatný čítač chyb. V první fázi testu je navíc každý přijatý testovací vektor uložen do paměti, bez ohledu zda nastala chyba při přenosu či nikoliv. Uložené vektory je možné vyčíst do software a následně zobrazit či uložit do souboru. Je tak umožněna kontrola testovacích vektorů algoritmu ITCT v software. Identifikace všech možných typů poruch propojů pomocí vektorů algoritmu ITCT v hardware vede na složitou implementaci. Úloha hardware spočívá především v zjištění, zda nastala chyba, a v uchování potřebných informací pro identifikaci chyby v software.

S přijímáním testovacích vektorů souvisí problematika zarovnání deserializovaných dat na výstupu jednotky ISERDES. Aby mohly být porovnávány všechny 4 deserializované testovací vektory v jediném taktu, musí být zaručeno správné zarovnání přijatých vektorů do výstupního deserializovaného vektoru. To lze jednoduše zaručit shodným resetováním obvodů OSERDES a ISERDES. Pokud by byl například reset signál obvodů ISERDES nastaven i po deaktivaci signálu reset obvodů OSRDES, mohlo by dojít k nesprávné synchronizaci serializované a deserializované sekvence. Mohla by tedy nastat situace, kdy by ze všech čtyř deserializovaných testovacích vektorů byly platné jen některé. Tím by bylo znemožněno porovnání všech deserializovaných a referenčních vektorů paralelně.

5.5.3 Blok řízení testu

Blok řízení testu zprostředkuje pomocí řídicích a stavových registrů komunikaci se software. Také se stará o správné načasování operací testu (komunikace s vysílací a přijímací částí) a o dynamickou rekonfiguraci elementu DCM.

Řídicí blok obsahuje jako jediný rozhraní pro komunikaci se software. Pokud přijde na toto rozhraní čtecí požadavek, k jehož dokončení jsou potřeba data z jiného bloku testu, je vygenerován příkaz. Ten je směrován do asynchronní fronty FIFO příslušného bloku. Dotazovaný blok poté autonomně shromáždí požadovaná data a vytvoří odpověď. Ta je prostřednictvím asynchronní fronty v opačném směru předána řídicímu procesu, který data vloží na příslušný port softwarového rozhraní a nastaví signál informující o připravenosti dat. Podobně probíhá i zápis dat – většinou řídicích. Základní informace o aktuálním stavu modulu testu jsou obsaženy ve stavovém registru. Tabulka 5.4 udává význam jednotlivých bitů stavového registru.

Bit	Význam
0	Nastaven pokud je test spuštěn
1	Krok testu
2	Nastaven pokud se vyskytla chyba při přenosu (krok 1)
3	Nastaven pokud se vyskytla chyba při přenosu (krok 2)
4-7	MUL parametr DCM generujícího hodinový signál pro přenos dat
8-11	DIV parametr DCM generujícího hodinový signál pro přenos dat
12-15	Nepoužito
16-31	Počet chyb při přenosu (krok 2)

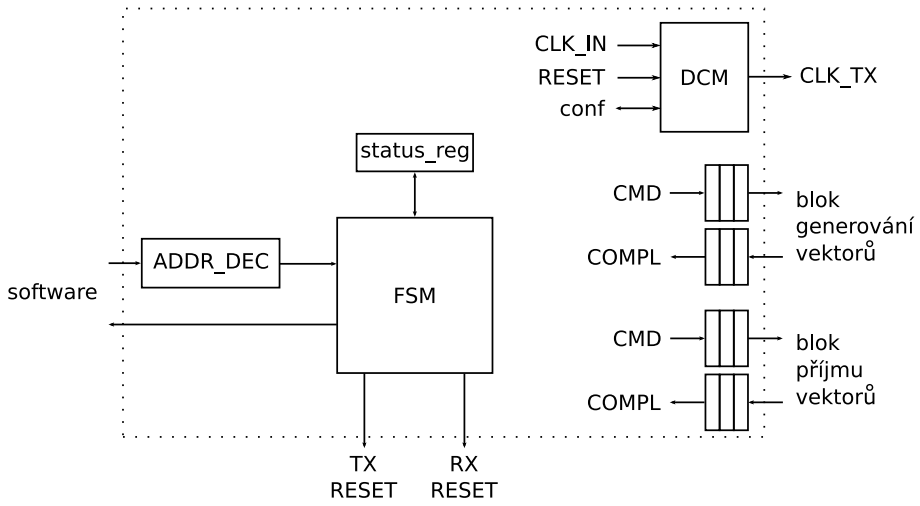
Tabulka 5.4: Význam bitů stavového registru testu propojů.

Důležitou úlohou řídicího bloku je proces dynamické rekonfigurace DCM pro generování hodinového signálu pro přenos dat po propojích. Postup samotné rekonfigurace DCM je

popsán v části 5.1.1. Úkolem řídicího bloku je provést korektní rekonfiguraci, jejíž postup lze popsat pomocí následujících kroků:

1. Příchod požadavku na rekonfiguraci spolu s novými hodnotami MUL a DIV.
2. Řídicí blok nastaví resetovací signál bloků pro vysílání a příjem dat. Nastaví také resetovací signál příslušného DCM.
3. Jsou provedeny všechny potřebné kroky pro rekonfiguraci DCM pomocí konfiguračního rozhraní.
4. Resetovací signál DCM je nastaven do 0.
5. Po dostatečně dlouhé časové prodlevě (je nutné počkat na nastavení signálu LOCKED příslušného DCM) lze nastavit resetovací signál bloků pro příjem a vysílání dat do 0.

Schéma bloku řízení testu je na obrázku 5.11. Jsou zde znázorněny asynchronní fronty pro komunikaci s ostatními bloky testu. Podle požadavků přicházejících ze software nastavuje centrální konečný automat řídicí signály a posílá příkazy ostatním blokům.



Obrázek 5.11: Blokové schéma části řízení testu.

5.6 Postup testování

Uvedený návrh testu propojů umožňuje jednoduché otestování propojů na kartě COMBO2. Pomocí následujícího postupu lze provést testování:

1. Před zahájením samotného testování je potřeba propojit rozhraní testovaného propoje pomocí vybraného konektoru (viz 5.4).
2. Poté stačí do FPGA čipu nahrát design s požadovanou testovací komponentou.
3. Nakonfigurováním parametrů elementu DCM přes softwarové rozhraní je nastavena přenosová rychlost propojů.

4. Pak je možné zápisem do řídicího registru spustit první fázi testu – test statických a dynamických poruch a Ground Bounce.

5. Po dokončení je možné výsledky vyčíst opět přes softwarové rozhraní a zobrazit případné nalezené chyby.

6. Po dokončení první fáze testu lze stejným způsobem spustit BER test. Informace o nalezených chybách lze vyčítat v průběhu testu. Test je možné kdykoli přerušit.

Pro testování za použití jiné přenosové rychlosti musí být test zastaven a musí být provedena rekonfigurace DCM obvodu, generujícího hodinový signál pro přenos. To lze provést jednoduchým zápisem požadovaných atributů hodinového signálu do řídicího registru testu. Konkrétní postup rekonfigurace DCM je uveden v části 5.1.1. Blok řízení testu se postará o celý proces rekonfigurace DCM a o správné načasování potřebných operací. Po dokončení rekonfigurace je možné spustit test bez nutnosti přehrávání designu FPGA.

Kapitola 6

Návrh testu pamětí

Pro testování pamětí RAM se nejvíce používají testy March. K důkladnému otestování dynamických a statických pamětí RAM mohou být potřeba vždy různé testy March. Bylo by proto vhodné, aby bylo možné zvolit testovací algoritmus v závislosti na typu testované paměti. Další důvod pro možnost volby testovacího algoritmu je čas potřebný k dokončení testu. Mohou nastat situace, kdy požadujeme pouze jednoduchý test paměti. Například k otestování správného zapojení datových vodičů vedoucích k paměti. V takovém případě by bylo nežádoucí čekat i několik desítek minut na dokončení sofistikovaného testu dynamické paměti RAM s kapacitou několik GB. Základním požadavkem na obecný test pamětí RAM je tedy možnost volby testovacího algoritmu.

6.1 Délka testovacích vektorů

Při návrhu testu několika různých typů pamětí RAM je nutné dbát na dostatečnou obecnost testovacího modulu. Například u paměti DDR DRAM je možné v rámci jedné operace přistoupit minimálně ke dvěma paměťovým slovům. Toto chování je dáno protokolem DDR, kdy se za jeden hodinový takt přenesou dvě slova. Operace s datovým slovem na adrese n vždy vyvolá stejnou operaci nad datovým slovem $n + 1$. Test se proto musí ke slovům na dvou po sobě jdoucích adresách chovat tak, jako by šlo o jedno paměťové slovo. Pokud by tomu tak nebylo, byl by porušen základní princip testu March, tj. že operace probíhají sekvenčně vždy právě nad jedním paměťovým slovem.

Paměť QDR SRAM podporuje datové přenosy také pouze pomocí protokolu DDR. Navíc je možné provádět operace s pamětí pouze v režimu burst pevné délky (u pamětí na kartě COMBO2 jde o burst délky 4). Režim burst značí shlukový přenos dat, tento režim je efektivnější z pohledu síťových aplikací, u kterých hraje důležitou roli propustnost paměťové sběrnice. Pokud paměť podporuje burst délky 4, znamená to přenos 4 datových slov při každé operaci. Spolu s použitím protokolu DDR jde tedy o přenos čtyř datových slov během dvou taktů při každé operaci s pamětí.

Délka testovacího vektoru algoritmu March musí být dvakrát a čtyřikrát větší než délka datového slova paměti pro paměť typu DDR, respektive QDR. Operace s jedním datovým slovem totiž ovlivňuje i jedno (DDR) nebo tři (QDR) sousední slova. Pro testování paměti na provozní frekvenci (at-speed) by se měla každý takt provádět nějaká operace s pamětí. At-speed testy jsou navrženy tak, že neobsahují žádné zbytečné čekací stavy a dovolují tak odhalit i poruchy, které by se projevily jen za plného vytížení paměťového modulu. Pro testování paměti DDR musí být zaručeno vybavení každého testovacího vektoru v jediném

taktu, pro paměti QDR stačí polovina testovacího vektoru za takt. Testovací vektory budou uloženy přímo na čipu FPGA v blokových pamětech, které mají konstantní latenci 1 takt.

Jelikož se s typem testované paměti mění délka testovacího vektoru, je potřeba tento fakt v návrhu zohlednit a zavést generické parametry testovacího modulu. Následuje výčet těchto parametrů spolu se stručným popisem:

DATA_WIDTH Udává šířku datového slova paměti.

VECT_LENGTH Udává délku testovacího vektoru (může se lišit od šířky datového slova paměti).

BURST Říká, zda datové přenosy probíhají v burst režimu.

BURST_LENGTH Udává počet datových slov přenesených v burst režimu.

MAX_ADDR Udává nejvyšší adresu paměti, která bude testována. U běžných pamětí udává vztah $2^{MAX\_ADDR} * DATA\_WIDTH$ velikost paměti, u DDR a QDR pamětí se může lišit díky různým adresovým módům.

6.1.1 Rozhraní pamětí RAM

Pro testování různých typů pamětí je také potřeba univerzální rozhraní řadiče paměti RAM. Řadiče pamětí RAM poskytují abstrakci složitého řídicího mechanismu paměťových modulů v podobě základní sady obecnějších řídicích signálů. I toto rozhraní se ale může lišit v závislosti na typu paměti či řadiče. Pro potřeby testu pamětí RAM je proto použito univerzální rozhraní, které je v obálce testovací komponenty konvertováno na rozhraní použitého řadiče konkrétní paměti. Obálkou je v tomto případě myšlena komponenta, která ve své architektuře obsahuje obecný test pamětí RAM a řadič konkrétního typu použité paměti. Signály univerzálního paměťového rozhraní jsou uvedeny v tabulce 6.1.

Jméno signálu	Směr	Význam
addr	výstup	Adresa paměti
di	výstup	Data zapisovaná do paměti
enable	výstup	Povolovací signál operací čtení/zápisu
write	výstup	Požadavek na zápis do paměti
read	výstup	Požadavek na čtení z paměti
do	vstup	Data vyčtená z paměti
do_dv	vstup	Signál do obsahuje platná data
write_busy	vstup	Indikátor připravenosti paměti na zápis
read_busy	vstup	Indikátor připravenosti paměti na čtení

Tabulka 6.1: Univerzální paměťové rozhraní testu pamětí RAM.

6.2 Testovací algoritmus

V části 3.3 je uveden zápis testu March a základní popis způsobu testování. Ve zkratce jde o několik sekvencí operací (čtení, zápis) postupně prováděných pro každou adresu paměti. Po bližším prostudování metodiky testování March testy byl navržen způsob dynamického

řízení March testu. V principu se jedná o uložení významných informací o algoritmu testu do blokové paměti a jejich postupné vyčítání přímo za běhu testu. Takto je možné test řídit a měnit jeho algoritmus bez nutnosti rekonfigurace celého čipu FPGA či jeho částí.

Konkrétně je potřeba o algoritmu testu uchovat informace jako celkový počet kroků testu a pro každý krok počet jeho operací, směr čítání adres a zda se jedná o tzv. čekací krok. Při čekacím kroku nejsou prováděny žádné paměťové operace po dostatečně dlouhou dobu. Čekací kroky slouží k odhalení dynamických poruch pamětí. Pro každou operaci je uveden typ operace (čtení nebo zápis) a odkaz na příslušný testovací vektor. Testovací vektory jsou také uloženy v blokové paměti uvnitř FPGA a lze je libovolně měnit za běhu designu.

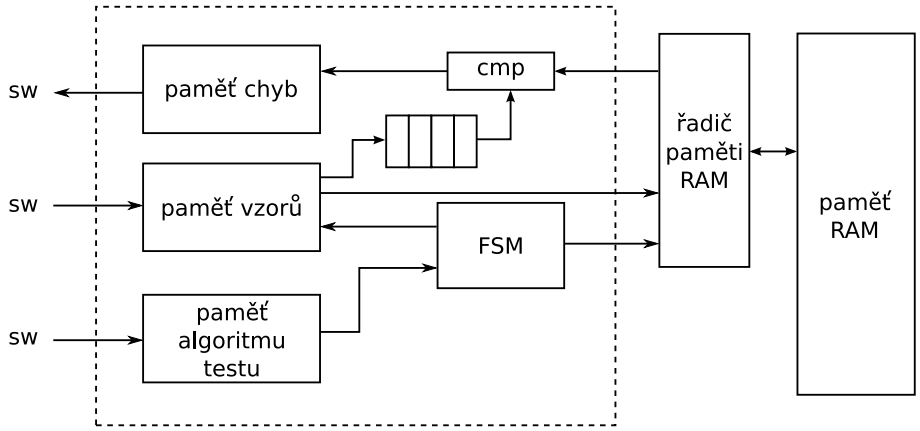
Pro zjednodušení přístupu ke konfiguračním pamětem algoritmu testu ze strany software byl v jazyce C napsán jednoduchý parser. Ten převede algoritmus z podoby čitelné člověkem (konfiguračního souboru) na posloupnost zápisů do konfiguračních pamětí algoritmu testu a umožní tak velmi jednoduchou změnu algoritmu testu. Navíc odpadá nutnost znalosti adresového prostoru testu pro změnu algoritmu testu. První řádek konfiguračního souboru obsahuje zakódovaný algoritmus testu March, uzavřený do hranatých závorek. Jednotlivé kroky testu jsou od sebe odděleny středníkem a jsou uvozeny směrem čítání adres (*up* a *down* pro vzestupný, resp. sestupný směr). Operace v kroku jsou uzavřeny do kulatých závorek a skládají se ze symbolů operace (r pro čtení a w pro zápis) a číselného indexu testovacího vektoru. Zakódovaný algoritmus testu March by mohl vypadat např. takto:

$$\{up(w0); down(r0, r0, w1, r1); down(r1)\}$$

Následuje jeden prázdný řádek a poté vždy jeden testovací vektor (zapsán v hexadecimální soustavě) na řádek. První testovací vektor má index 0, následující má index 1 atd.

6.3 Architektura

Základní architektura testu pamětí RAM vychází z práce [9]. Tato architektura obsahuje přímo v hardware zakódovaný algoritmus testu March. Pokud upravíme architekturu testu podle návrhu uvedeného výše, lze algoritmus nahrávat do hardware za běhu. Pak lze použít stejnou architekturu pro testování statických i dynamických pamětí, pouze použitý algoritmus testu se bude lišit.



Obrázek 6.1: Blokové schéma testu pamětí RAM.

Na obrázku 6.1 je znázorněno blokové schéma architektury testu pamětí RAM. Pomocí software je nahrán algoritmus testu a datové vzory. Po spuštění testu vyčítá postupně řídicí automat kroky testu (algoritmus). Posloupnost paměťových operací je tedy určena uživatelem definovaným algoritmem testu.

6.3.1 Stavový automat

Nejkomplexnějším prvkem architektury testu pamětí je řídicí stavový automat. Hlavní funkcí stavového automatu je interpretace algoritmu testu, kterou zajišťuje dekódováním informací z konfiguračních blokových pamětí a prováděním požadovaných operací s pamětí.

Architekturu stavového automatu je možné popsat behaviorálně, pro případ implementace ve vysokoúrovňovém jazyce pro popis hardware (např. Handel-C). Následující popis architektury v pseudokódu se skládá ze dvou bloků – hlavního bloku testu:

```
 1 for each step
 2    fetch step info (wait, cnt_dir, op_cnt)
 3    if wait
 4       wait for some time
 5    else
 6       if cnt_dir = up
 7          act_addr := 0
 8       else
 9          act_addr := MAX_ADDR
10       for each addr
11          for each op
12             if op = write
13                write to ram
14             else
15                write to ref_fifo
16                read from ram
```

a bloku porovnávání testovacích vektorů:

```
17 if ram data valid
18    compare ram data, ref_fifo
19    if !data_ok
20       error count ++
21       save error info
```

Pro každý krok algoritmu se opakuje cyklus na řádcích 2 – 15. Pokud je aktuální krok testu čekací, je aktivována časová prodleva (řádek 4). V opačném případě je podle směru čítání adres nastavena počáteční adresa (řádky 6 – 9). Poté následují dva vnořené cykly, které zaručují provedení všech operací (řádek 11) postupně pro každou adresu testované paměti (řádek 10). Pokud se jedná o operaci zápisu, je proveden zápis příslušného testovacího vektoru (řádek 13). Pokud jde ale o operaci čtení, je testovací vektor, který by měl být v případě bezporuchovosti paměti vyčten, uložen do fronty (řádek 15). Také je iniciováno čtení z paměti (řádek 16).

Blok porovnávání testovacích vektorů je prováděn nezávisle na hlavním bloku. Poté co je vyčten testovací vektor z paměti (řádek 17) je porovnán s referenčním testovacím vektorem

z fronty (řádek 18). Pokud nejsou oba vektory shodné, je zaznamenána chyba inkrementací čítače chyb a uložením nezbytných informací pro identifikaci poruchy paměti (řádek 21).

Stavový automat interpretuje konfiguraci testu uloženou v konfiguračních blokových pamětech na čipu FPGA. Konkrétně se jedná o tři blokové paměti: paměť algoritmu testu, kde každý záznam reprezentuje jeden krok algoritmu, paměť operací, kde každý záznam reprezentuje konkrétní operací s pamětí, a paměť testovacích vektorů.

Bity	Význam
0-3	Počet operací v daném kroku.
4	Směr čítání adres (0 = vzestupně, 1 = sestupně).
5	Identifikace čekacího kroku.

Tabulka 6.2: Význam bitů konfigurační blokové paměti algoritmu testu.

Bity	Význam
0	Identifikátor paměťové operace (0 = čtení, 1 = zápis).
1-5	Identifikátor testovacího vektoru.

Tabulka 6.3: Význam bitů konfigurační blokové paměti operací.

6.4 Identifikace poruchy

Testování paměti je prováděno pomocí zápisu a následného čtení testovacích vektorů z paměti. Pokud je algoritmem požadována zápisová operace, je proveden zápis příslušného testovacího vektoru do paměti. Při operaci čtení je zaznamenán referenční testovací vektor uložením do fronty typu FIFO. Po vyčtení dat z paměti je porovnán vyčtený a referenční testovací vektor. V případě shody testovacích vektorů proběhly operace zápisu a čtení dat z paměti v pořádku. V opačném případě nastala chyba a je potřeba uložit potřebné informace pro pozdější identifikaci poruchy. Konkrétně je uložen příslušný krok algoritmu March, pořadí operace v rámci kroku a adresa paměti, při jejíž vyčítání nastala chyba. Také je uložen vektor identifikující bity, jejichž hodnota byla chybná. Tento vektor se získá operací XOR referenčního a vyčteného vektoru. Uvedené informace jsou dostačující pro pozdější přesnou identifikaci nalezené poruchy paměti.

Informace o poruchách jsou ukládány do tří blokových pamětí přímo na čipu FPGA. Jako index do blokové paměti slouží pořadí nalezené chyby. Pokud bylo např. nalezeno 5 chyb, blokové paměti budou obsahovat 5 záznamů, uložených od adresy 0 (první nalezená chyba) do adresy 4. Struktura ukládaných dat je uvedena v tabulkách 6.4, 6.5 a 6.6. Délka vektorů identifikujících chybné bity je vždy rovna délce testovacího vektorů konkrétní testované paměti. Nevyužité horní bity blokové paměti neobsahují platná data. Tyto blokové paměti lze vyčítat pomocí softwarového rozhraní. Adresy (offsety) jednotlivých pamětí lze nalézt v popisu adresového prostoru testu pamětí RAM (tabulka A.1).

Bity	Význam
0-31	Adresa výskytu chyby testované paměti.

Tabulka 6.4: Paměť pro ukládání adresy výskytu chyby.

Bity	Význam
0-4	Číslo kroku algoritmu.
5-11	Pořadí prováděné operace v příslušném kroku testu.

Tabulka 6.5: Paměť pro ukládání doplňujících informací o výskytu chyby.

Bity	Význam
0-143	Vektor identifikující chybné bity (max. 144 bitů).

Tabulka 6.6: Paměť pro ukládání vektorů chybných bitů.

6.5 Postup testování

Architektura testu RAM pamětí byla navržena obecně, je tedy možné použít téměř shodné testovací moduly k otestování libovolné paměti RAM na kartě COMBO2. Metodika testování i softwarové rozhraní zůstává vždy stejné, proto je možné pomocí níže uvedeného postupu otestovat všechny podporované typy pamětí RAM. Předpokládejme, že do čipu FPGA byla nahrána konfigurace obsahující potřebný testovací modul pamětí RAM. Další postup je následující:

1. Nejdříve je potřeba provést konfiguraci testu, tedy pomocí software nahrát algoritmus testu. To lze provést nadefinováním algoritmu v textovém souboru a nahráním do FPGA pomocí připraveného software, který se mj. postará o nastavení konfiguračního bitu v řídicím registru.

2. Zapsáním hodnoty $0x1$ do řídicího registru je test spuštěn.

3. Stav testu lze kdykoli vyčítat ze stavového registru, který informuje např. o počtu nalezených chyb a o aktuálním kroku testu. Registr průběhu obsahuje adresu paměti, na kterou byl právě proveden přístup. Úplné informace o adresovém prostoru testu a významu jednotlivých registrů jsou v tabulce A.1.

4. Pokud test odhalil chybně vyčtená data z paměti, je možné získat další informace o konkrétních chybách. Jedná se o krok algoritmu testu, číslo operace a vektor chybných bitů.

5. Po skončení testu je možné jej opětovně spustit, stavové registry a čítače chyb se pak automaticky inicializují na počáteční hodnoty. Před opětovným spuštěním testu je samozřejmě možné nahrát (jiný) algoritmus testu.

Kapitola 7

Závěr

Cílem této práce byl návrh a implementace testů propojů a paměťových prostředků na kartě COMBO2. Po nastudování nejnovějších poznatků z oblasti testování propojů a pamětí byly vybrány nejúčinnější metody testování, jejichž teoretické základy jsou uvedeny v první části práce. S ohledem na cílovou platformu byl poté proveden návrh architektur testů propojů a paměťových prostředků. Při návrhu byl kladen velký důraz na vytvoření obecné architektury.

Výsledkem této práce je architektura testu propojů a paměťových prostředků RAM. Podle návrhu byla provedena implementace obecného testu propojů a testu pamětí za použití jazyků pro popis hardware VHDL a Handel-C. Problém testování propojů za použití různých přenosových rychlostí je vhodně vyřešen pomocí dynamické rekonfigurace části testu. Díky obecnému paměťovému rozhraní je možné použít test pamětí pro testování různých typů pamětí RAM. Navíc možnost změny testovacího algoritmu zajišťuje značnou flexibilitu testování. Funkčnost byla ověřena simulacemi. Byla také provedena statická časová analýza, která prokázala bezproblémové nasazení testů v hardware.

Pro potřeby testování propojů byly navrženy dva typy propojovacích konektorů, které umožňují propojení rozhraní propojů na kartě COMBO2 a zjednodušují tak proceduru testování.

Jako možné další rozšíření této práce by bylo ověření funkčnosti navržených testů na cílové platformě COMBO2.

Literatura

[1] Peter Alfke. *Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators*. Xilinx, Inc, July 1996. http://www.xilinx.com/bvdocs/appnotes/xapp052.pdf.

[2] CESNET, z.s.p.o. *COMBO2: Specifikace karty*, říjen 2007.

[3] Bob Colwell. Ground bounce. *Computer*, pages 11–13, March 2003.

[4] Tomáš Filip. Testování spojů na desce combo6. Bakalářská práce. FIT VUT v Brně, 2005.

[5] Said Hamdioui, Zaid Al-Ars, and Ad J. van de Goor. Testing static and dynamic faults in random access memories. In *Proceedings of the 20th IEEE VLSI Test Symposium*, 2002.

[6] Syed B. Huq and John Goldie. *Application Note 971: An Overview of LVDS Technology*. National Semiconductor, July 1998. http://www.national.com/an/AN/AN-971.pdf.

[7] Artur Jutman. At-speed on-chip diagnosis of board-level interconnect faults. In *Ninth IEEE European Test Symposium*, pages 2–7, 2004.

[8] Jung-Cheun Lien and Melvin A. Breuer. Maximal diagnosis for wiring networks. In *Proceedings of the IEEE International Test Conference on Test: Faster, Better, Sooner*, pages 96–105. IEEE Computer Society, 1991.

[9] Martin Louda. Testování paměťových prostředků na desce combo6. Bakalářská práce. FIT VUT v Brně, 2006.

[10] Amitava Majumdar, Michio Komoda, and Tim Ayres. Ground bounce considerations in dc parametric test generation using boundary scan. In *16th IEEE VLSI Test Symposium*, 1998.

[11] Sungju Park. A new complete diagnosis patterns for wiring interconnects. In *33rd Annual Conference on Design Automation*, pages 203–208, 1996.

[12] Ad J. van de Goor and Issam B. S. Tlili. Tests for word-oriented memories. Technical report, Delft University of Technology, Department of Electrical Engineering, Delft, The Netherlands, 1997.

[13] Xilinx, Inc. *Virtex-5 Data Sheet: DC and Switching Characteristics*, April 2008. http://www.xilinx.com/support/documentation/data_sheets/ds202.pdf.

[14] Xilinx, Inc. *Virtex-5 FPGA Configuration User Guide*, March 2008. http://www.xilinx.com/support/documentation/user_guides/ug191.pdf.

[15] Xilinx, Inc. *Virtex-5 FPGA User Guide*, March 2008. http://www.xilinx.com/support/documentation/user_guides/ug190.pdf.

Dodatek A

Adresový prostor testu pamětí RAM

Offset (bajty)	Délka (bajty)	Přístup	Popis
$0x0$	4	RW	Řídicí registr
$0x4$	4	R	Stavový registr
$0x8$	4	R	Registr průběhu
$0x10$	4	RW	Registr počtu kroků testu
$0x14$	236	-	Nevyužito
$0x100$	72	W	Paměť testovacích vektorů #0
$0x148$	72	W	Paměť testovacích vektorů #1
$0x190$	72	W	Paměť testovacích vektorů #2
$0x1D8$	72	W	Paměť testovacích vektorů #3
$0x220$	72	W	Paměť testovacích vektorů #4
$0x268$	152	-	Nevyužito
$0x300$	128	W	Paměť algoritmu testu
$0x380$	128	-	Nevyužito
$0x400$	512	W	Paměť operací
$0x600$	512	-	Nevyužito
$0x800$	512	R	Paměť adresy výskytu chyby
$0xA00$	512	R	Paměť informací o výskytu chyby
$0xC00$	512	R	Paměť vektorů chybných bitů #0
$0xE00$	512	R	Paměť vektorů chybných bitů #1
$0x1000$	512	R	Paměť vektorů chybných bitů #2
$0x1200$	512	R	Paměť vektorů chybných bitů #3
$0x1400$	512	R	Paměť vektorů chybných bitů #4

Tabulka A.1: Adresový prostor testu pamětí RAM.