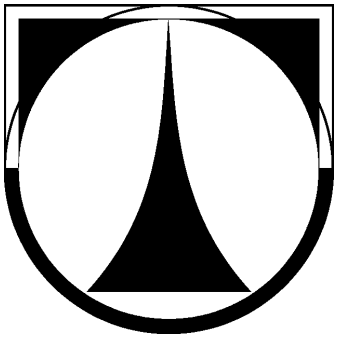
Technická Univerzita v Liberci

Fakulta mechatroniky, informatiky a mezioborových studií

Ústav informačních technologií a elektroniky



Aplikačně závislé testování FPGA obvodu

Application dependent FPGA testing

Disertační práce

2011 **Ing. Martin Rozkovec**

Aplikačně závislé testování FPGA obvodu
Application dependent FPGA testing

Autor: Ing. Martin Rozkovec

Studijní program: P2612 Elektrotechnika a informatika

Studijní obor: 2612V045 – Technická kybernetika

Školící pracoviště: Ústav informačních technologií a elektroniky

Fakulta mechatroniky, informatiky a mezioborových studií

Technická Univerzita v Liberci

Studentská 2/1402, 461 17, Liberec

Školitel: Prof. Ing. Ondřej Novák, CSc.

Rozsah práce:

Počet stran: 90

Počet obrázků: 51

Počet tabulek: 22

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Abstrakt

Tato práce se zabývá testováním obvodů FPGA. V textu jsou představeny základní pojmy testování obvodů ASIC a vybrané techniky testování obvodů FPGA. Jádrem textu je představení nové metody, která je určena k testování FPGA obvodu.

Představená metoda pracuje s modulárně rozděleným obvodem, který je implementován s pomocí částečné dynamické rekonfigurace. Změna okolí testovaného modulu – dynamické vytváření testerů a analyzátoru testu umožňuje otestovat celý obvod. Ověřovány jsou analyzátory na bázi MISR a skenovacího řetězce a testery na bázi LFSR, generátoru deterministického testu a dekompresoru zkomprimované sekvence.

Na různých experimentech jsou ověřeny dílčí části metody – schopnost dělení obvodu, transformace popisu obvodu pro ATPG, kvalita testovacích vektorů a schopnost detekce poruch SEU.

Abstract

This work is focused on the FPGA testing. The basics of the ASIC testing are presented in the text as well as some of the FPGA test methods. The core of the report is focused on presentation of the novel application dependent FPGA test method.

Presented method splits the tested circuit into several modules that are implemented using partial reconfiguration design flow. Partial reconfiguration allows changing the role of modules surrounding the tested module to testers and analyzers. All the functional modules can be tested using this approach. Analyzers based on a MISR and scan chains are evaluated as well as testers based on a LFSR, a deterministic test generator and a compressed sequence decompressor.

Circuit splitting, FPGA netlist to ASIC netlist transformation, quality of test patterns and ability of SEU detection are evaluated in various experiments within the text.

Poděkování

Děkuji rodičům a mým nejbližším za významnou podporu během studia a při psaní této práce. Zároveň bych chtěl poděkovat svému školiteli, prof. Ing. Ondřeji Novákovi, CSc. za cenné rady a zkušenosti.

Práce byla financována grantem GAČR GA102/09/1668 s názvem „Zvyšování spolehlivosti a provozuschopnosti v obvodech SoC 2009-2011“.

Obsah

Seznam obrázků

Seznam tabulek

Použité zkratky

ASIC	Application Specific Integrated Circuit
ATE	Automated Test Equipment
ATPG	Automatic Test Pattern Generator
BIST	Built-In Self-Test
BRAM	Block RAM
CLB	Complex Logic Block
CMOS	Complementary Metal-Oxide-Semiconductor
COMPAS	Compressed Test Pattern Sequencer
CPLD	Complex Programmable Logic Device
CUT	Core Under Test
DSP	Digital Signal Processing
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
ICAP	Internal Configuration Access Port
IOB	Input-Output Block
IP	Intellectual Property
LFSR	Linear Feedback Shift Register
LUT	Look Up Table
MISR	Multiple Input Signature Register
ORA	Output Response Analyzer
PAR	Place And Route
PLB	Processor Local Bus
RESPIN	Reusing Scan Chains for Test Pattern Decompression
SAT	Boolean satisfiability
SET	Single Event Transient
SEU	Single Event Upset
So(P)C	System on a (Programmable) Chip
SRAM	Static RAM
SRL	Shift Register LUT
TPG	Test Pattern Generator
TTL	Transistor-transistor-logic
VHDL	Very High speed integrated circuits Description Language

1 Úvod

Integrovaný obvod patří mezi nejsložitější lidské výtvory. Od svého vynalezení v 50. letech minulého století stačil proniknout snad do všech myslitelných zařízení, jež nás obklopují.

Integrovaný obvod je vyroben z monokrystalického křemíkového plátku, na kterém jsou vytvořeny tranzistory, které jsou vzájemně propojeny pomocí několika metalických vrstev vodičů oddělených tenkými vrstvami izolantů. Motivy propojovacích vrstev jsou velké řádově desítky nanometrů a vrstvy jsou silné stovky atomů. Vlivem nedokonalostí při výrobě, nepříznivých provozních podmínek nebo degradace materiálů může v obvodu dojít k celé řadě poruch. Ty mohou mít na systém, jehož je obvod součástí nepříznivé následky. Ať už se jedná o řízení pracího cyklu nebo o zajištění bezpečnosti leteckého provozu, vždy jsou na obvody kladeny nemalé nároky na spolehlivost.

Spolehlivost obvodů lze zajistit nejen pomocí produkčních testů, ale i průběžným testováním obvodů v místě provozu. Cílem práce je vyvinout metodu testování obvodů FPGA, která umožní testování obvodu bez dodatečného přístrojového vybavení a za použití běžně dostupných nástrojů.

V první kapitole práce jsou představeny pojmy a techniky testování ASIC obvodů. Jsou popsány modely poruch a základy testování kombinačních a sekvenčních obvodů. V druhé kapitole jsou popsány obvody FPGA a základní metody jejich testování. V kapitole třetí se nachází samotné jádro práce – metoda strukturního testování obvodu implementovaného v FPGA. Detailně je popsána každá dílčí část metody. Čtvrtá kapitola prezentuje celou řadu experimentů, které byly s dílčími částmi metody prováděny. Závěrečná kapitola shrnuje dosažené výsledky, analyzuje je a navrhuje směr, kterým se může metoda dále rozvíjet.

1.1 Trvalé poruchy

Trvalé poruchy vznikají většinou při výrobě obvodu nebo degradací materiálu integrovaného obvodu. Jejich vliv na obvod je trvalý a po vzniku poruchy je nelze odstranit. Jelikož by pro účely testování bylo velmi náročné poruchy modelovat na fyzikální úrovni, používají se tzv. modely poruch [35], jež poruchy popisují na vyšších úrovních abstrakce.

Historicky nejstarším modelem poruch číslicových obvodů je model poruch trvalá 0/1 (stuck-at (SA) [41], [22], který modeluje poruchy, které se objevovaly na deskách plošných spojů a v integrovaných obvodech typu TTL a NMOS. Model trvalá 0/1 rozlišuje dva typy poruch – trvalá 1 - stuck-at 1(SA1) a trvalá 0 - stuck-at 0(SA0). V případě trvalé 1 se může jednat např. o přerušený vodič a poruchu modelujeme jako trvalé připojení vodiče na logickou 1. Trvalá 0 může být způsobena např. průrazem tranzistoru uvnitř hradla a modeluje se jako připojení vodiče na logickou 0.

Model zkratu (bridging fault) [41] modeluje poruchy, které vznikají spojením dvou sousedních vodičů v obvodu. V obvodech CMOS je lze modelovat jako montážní součet, v případě technologie TTL jako montážní součin. Model vyžaduje podrobné informace o struktuře obvodu, především pak umístění vodičů, bez kterého je velmi obtížné určit, ke kterému spojení vodičů může dojít. Informace o fyzické implementaci jsou vyžadovány i modely poruch, které popisují trvalou vodivost tranzistoru (stuck-on) nebo přerušené vodiče (open) v obvodech CMOS.

1.2 Dočasné poruchy

Dočasné poruchy vznikají především vlivem provozních podmínek a po jejich odeznění mohou zaniknout. Dočasnou poruchu může způsobit například přehřátí obvodu, vystavení silnému elektromagnetickému poli, anebo radioaktivita prostředí. Dočasné poruchy mohou být někdy modelovány jako trvalé poruchy (např. přerušení vodiče vlivem elektromigrace [44]).

1.2.1 Poruchy SEU

Při průchodu těžkého iontu, protonu nebo neutronu křemíkem dochází k přenosu energie částice do prostředí, který se projeví vznikem nosičů náboje (děr). Akumulovaný náboj vytvoří přechodný proudový zákmit, který může negativně působit na funkci obvodu.

Narazí-li částice do tranzistoru uvnitř hradla, může dojít ke krátkodobé změně výstupu. Tento efekt se označuje jako SET [29] (Single Event Transient). Po vyčerpání náboje vzniklého ionizací efekt samovolně odezní. V případě, že částice narazí do tranzistoru, který je součástí paměťové buňky může dojít k inverzi uložené hodnoty. Tento efekt se nazývá SEU (Singe Event Upset) a může být způsoben i SET, pokud k SET dojde v okamžiku změny stavu sekvenčního obvodu. Při vysokém větvení kombinační logiky může vlivem jedné SET dojít ke změně na více větvích obvodu.

Pokud jsou v postižených signálových cestách klopné obvody, může dojít k mnohonásobnému přepnutí – Multiple Bits Upset (MBU). V případě SEU nebo MBU nabývá porucha charakteru trvalé poruchy do doby, než je detekována a opravena.

Pravděpodobnost, že k SET nebo SEU dojde je odvislá od míry radioaktivity prostředí (sluneční aktivita – počet a energie částic), geografické pozice (nadmořská výška, pozice vůči geomagnetickému poli), rozměrů tranzistoru (napájecí napětí, velikost přepínacího proudu) a ploše čipu [32]. Pro obvody vyrobené technologií nad 100 nm byly zaznamenány výskyty SEU v kosmickém prostoru [25], výjimečně v letových hladinách běžných komerčních letů [29]. Předpokládá se, že pro menší rozměry tranzistorů bude k výše popsaným efektům docházet i v nízkých nadmořských výškách.

1.3 Testování

Testování obvodu se provádí přivedením testovacího vektoru (test pattern) z testeru na primární vstupy testovaného obvodu (CUT – circuit under the test), odečtením odezvy obvodu z primárních výstupů a jejím následným vyhodnocením. Tato procedura se nazývá krokem testu. Doba testu je počet kroků testu vynásobený dobou trvání kroku testu.

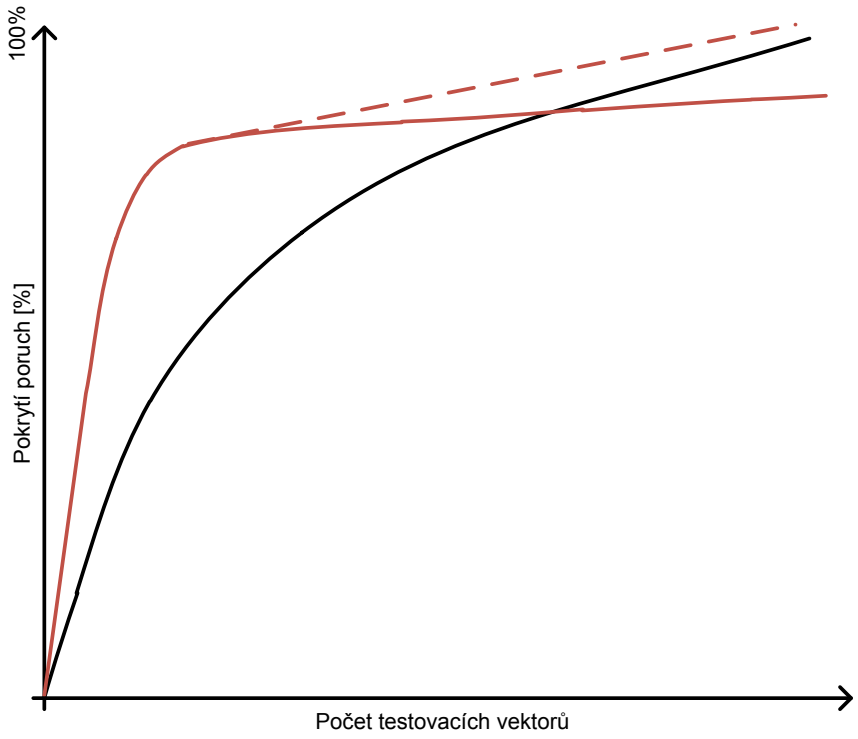
V případě produkčních testů bývá testovaný obvod zapojen do zařízení, které testování provádí samočinně. Tato zařízení se souhrnně nazývají ATE (Automatic Test Equipment). Obsahují řadič testu, vnitřní paměť, do které se ukládají testovací vzorky a paralelní rozhraní k fyzickým pinům obvodu (test access). Na parametrech testeru, především na rychlosti a kapacitě přístupu k CUT závisí produkční náklady na test. Část nebo celé testovací vybavení se proto přesouvá do testovaného obvodu, ve kterém může běžet o několik řádů rychleji. Včlenění testovacího vybavení do obvodu bývá označováno jako BIST [41] (Built-In Self-Test).

Testování s BIST se redukuje na výzvu ATE k zahájení testu a odečtení výsledku ve formě „v pořádku“, nebo „detekována porucha“. Začlenění testovacího vybavení do obvodu navíc v určitých případech umožňuje průběžnou diagnostiku obvodu, která zvyšuje spolehlivost systému přímo v místě provozu (např. na osazené desce plošných spojů řídící technologický proces).

1.4 Testování kombinačních obvodů

Definujme kombinační obvod jako elektrický obvod plnící booleovskou (často vektorovou) funkci $y = F(x)$. X nazvěme vstupním vektorem funkce a y výstupním vektorem. Prvky vektoru x bývají nazývány primárními vstupy obvodu a mají podobu vstupních pinů obvodu. Prvky vektoru y se nazývají primárními výstupy obvodu a mají fyzickou podobu výstupních pinů obvodu.

Triviální ověření funkce obvodu spočívá v systematickém přivádění všech kombinací vektoru x a ověření, že vektor y nabývá správné hodnoty. Takový test je nazýván úplným nebo triviálním testem a detekuje všechny poruchy, které nemění chování obvodu na sekvenční. Doba trvání triviálního testu je závislá pouze na počtu primárních vstupů.



Obrázek 1: Obecná závislost pokrytí na počtu přivedených vektorů

Některé obvody lze rozdělit na podobvody - tzv. kužely. Triviální testy jednotlivých kuželů mohou probíhat paralelně a test, který se označuje jako pseudotriviální, bude trvat podstatně kratší dobu nežli triviální test celého obvodu.

Přivedeme-li na vstupy obvodu náhodný vektor, pak pravděpodobnost, že se bude jednat o vektor detekující určitou poruchu je $\frac{m}{2^n}$, kde n je počet primárních vstupů

obvodu a *m* je počet vektorů, které danou poruchu detekují. Budeme-li počet přivedených náhodných vektorů navyšovat, pak se bude zvyšovat i pravděpodobnost detekce poruchy. Snadno testovatelné poruchy budou detekovány s vyšší pravděpodobností, než poruchy obtížně testovatelné. Empiricky bylo toto ověřeno [35] (viz obrázek 1). Černá závislost v grafu zobrazuje variantu jeden deterministický vektor – jedna porucha. V praxi jeden vektor detekuje více poruch, takže závislost stoupá strměji, nežli závislost značená červenou barvou.

Náhodný test nemůže pokaždé dosahovat stejného pokrytí poruch (na základě náhodnosti) a jeho implementace ve formě BIST je složitá. V praxi používají namísto náhodných pseudonáhodné testy. Ty jsou generovány buď v celulárních automatech, anebo v posuvných registrech s lineární zpětnou vazbou (LFSR). LFSR bývá někdy doplněn o kombinační logiku (např. phase shifter [35]), která upravuje rozložení nul a jedniček na výstupech generátoru. Jelikož je vytvoření LFSR výhodnější co do rychlosti i plochy, často se LFSR používají i při triviálních nebo pseudotriviálních testech namísto binárních čítačů. Aby bylo zaručeno, že budou vygenerovány všechny možné kombinace, je nutné nastavit (zapojením zpětných vazeb a hradel XOR) LFSR tak, aby realizoval primitivní generující polynom [10]. V některých případech je registr doplněn logikou, která umožňuje vygenerování nulového vektoru.

1.5 Strukturní deterministický test

Při sestavování fyzické implementace obvodu se booleovská funkce dekomponuje na kombinaci jednodušších funkcí. Tento proces se nazývá syntézou obvodu a jeho výstupem je orientovaný graf, který se nazývá netlist. Uzly grafu odpovídají technologickým primitivům (např. hradla) a hranám odpovídají vodiče mezi jednotlivými prvky. Prvek netlistu, který chceme testovat na určitou poruchu, musíme správným způsobem stimulovat a informaci o případné poruše přivést na primární výstupy obvodu, ze kterých informaci odečteme.

Za tímto účelem se sestaví pomyslná signálová cesta vedoucí z primárních vstupů obvodu přes místo výskytu poruchy. Algoritmus vytváření cesty je možné nalézt v [18]. Výpočetní náročnost algoritmu zcitlivění cesty exponenciálně stoupá s vnitřním větvením obvodu (významně roste počet možných návratových cest). Využívají se proto nejrůznější heuristické techniky (např. FAN [18]), které se strukturu obvodu učí a tím snižují počet návratů v algoritmu. Programy, které generují testovací vektory, se souhrně nazývají ATPG (Automatic Test Pattern Generator). Jejich nekomerčními

představiteli jsou např. Atalanta [31] nebo MILF [19], komerčními pak např. TetraMAX [46] nebo TestKompres [45].

Začlenění deterministického strukturního testu do testovaného obvodu je problematické, poněvadž vyžaduje uložení testovacích vektorů a tím navyšuje nároky na vybavení vnořeného testeru. Strukturní test se proto provádí především při procesu výroby obvodu, kde paměťové omezení u externího testeru nehraje tak významnou roli, jako u testeru vnořeného. Výhodou deterministických vektorů je předem dobře odhadnutelné pokrytí poruch a dobrý poměr pokrytí k počtu vektorů (jeden vektor testuje většinou víc poruch).

1.6 Kombinovaný test

Pseudonáhodný test odhalí rychle velký počet snadno testovatelných poruch a jeho režie je nevelká. Deterministický test oproti tomu potřebuje poměrně malý počet vektorů na odhalení všech testovatelných poruch, ale nároky na vybavení testeru jsou vysoké a rychlost testu je řádově nižší. V praxi se využívá kombinace obou metod.

Tabulka 1: Porovnání deterministického a pseudonáhodného testu

test	počet vektorů	rychlost testu	Režie BIST	efektivita testu
deterministický	nízký	pomalá	vysoká	vysoká
	desítky až stovky vektorů	rozhraní ATE CUT je řádově pomalejší nežli rychlost CUT	nutnost interní paměti	testuje všechny testovatelné poruchy
pseudonáhodný	vysoký	rychlá	nízká	nízká
	tisíce vektorů	CUT běží na vysokých frekvencích	postačuje zpětnovazebný registr	nízká pravděpodobnost pro obtížně testovatelné poruchy

LFSR obsažený v obvodu se v pravidelných intervalech inicializuje deterministickými testovacími vektory [30]. Inicializace protestuje obtížně testovatelné poruchy a samotný běh LFSR vygeneruje pseudonáhodné posloupnosti pokrývající velký počet poruch. Protože není třeba přivést všechny deterministické vektory, sníží se jejich počet a vektory je možné snadno přenést z ATE nebo uložit přímo do paměti testeru v čipu. Výše popsaného principu využívá například ATPG TestKompress nebo v modifikované podobě algoritmus komprese testovacích vektorů COMPAS [26].

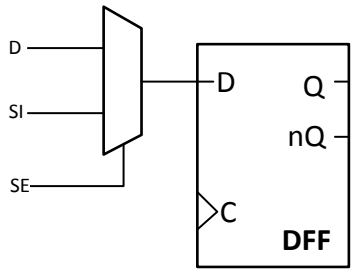
1.6.1 Sekvenční obvody

Cílem funkčního testování sekvenčního obvodu je odlišit daný automat od jiných automatů, které mohou vzniknout působením poruchy v obvodu [22]. Na základě přechodových a výstupních funkcí automatu se sestavují sekvence testovacích vektorů tzv. kontrolní posloupnosti. Délka kontrolních posloupností roste v přímé úměře s počtem vnitřních stavů automatu a přechodů mezi nimi. Ruku v ruce s tím roste i možnost, že vůbec nebudeme schopni některé posloupnosti sestavit. K funkčnímu testu se přistupuje v případě jednoduchých automatů, nebo když není známa struktura obvodu. V ostatních případech se využívá strukturního testu.

1.7 Strukturní testování sekvenčních obvodů

Přechodová i výstupní funkce automatu je implementována jako kombinační logika. Stav automatu se uchovává v klopných obvodech. Jsme-li schopni řídit a pozorovat stavy klopných obvodů, pak můžeme kombinační logiku automatu testovat za pomocí strukturního testu. Přístup ke klopným obvodům se nejčastěji řeší následujícím způsobem.

Nalezneme všechny klopné obvody a přidáním multiplexoru na jejich vstupy (viz Obrázek 2) vytvoříme skenovací řetězec – posuvný registr s paralelní předvolbou, sériovým vstupem a paralelním a sériovým výstupem. Zavedením skenovacího řetězce pomyslně rozpojíme zpětné vazby v automatu. Ze vstupů klopných obvodů vytvoříme dočasně primární výstupy a z výstupů primární vstupy.



Obrázek 2: Standardní skenovací buňka s multiplexorem

Pro sériový vstup a výstup řetězce vytvoříme nový primární vstup, resp. primární výstup obvodu. Primární vstupy obvodu obohatíme dále o signály předvolba (capture), přepnutí mezi funkčním a testovacím režimem (test mode) a sériový posuv (shift). Testování obvodu poté probíhá sériovým načítáním vektorů do obvodu, sejmutím odezvy kombinační logiky do klopných obvodů a jejím sériovým vyčtením a vyhodnocením.

Vzhledem k tomu, že přístup do skenovacích řetězců je sériový a tudíž relativně pomalý bývá povětšinou namísto jednoho dlouhého řetězce v obvodu vytvořeno několik kratších, paralelních, skenovacích řetězců. I s několika paralelními řetězci je sériový přístup k obvodu limitujícím faktorem doby testu. Z tohoto důvodu se vkládá část ATE do obvodu a část testu je realizována přímo v obvodu. Někdy (viz [17]) se využívá paralelních řetězců k zmenšení nároků na přenosovou kapacitu testovacího přístupu a testovací vzorky jsou do obvodu přiváděny zkomprimované.

2 Obvody FPGA

V předchozích kapitolách byly popsány základy testování obvodů s neměnnou funkcí označovaných jako ASIC (Application Specific Integrated Circuit – aplikačně specifický integrovaný obvod). Vývoj těchto obvodů je náročný na čas, zkušenosti návrhářů a především na finance – vývoj špičkového ASIC obvodu může trvat i řadu let a stát řádově miliony dolarů. Rozpočet je důvodem k tomu, že se ASIC produkují pouze ve velkých sériích. Pro malé nebo prototypové série je vhodnější využít obvody, které se jako alternativa objevili na počátku 80. let minulého století a jež se obecně označují jako PLD (programable logic devices – programovatelné obvody).

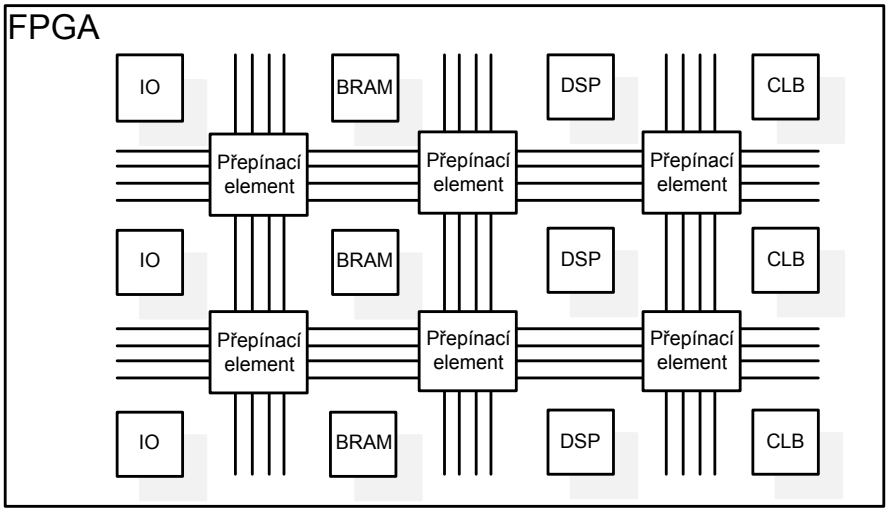
Vnitřní architektura, která je neměnná, je navržena jako ASIC obvod producentem těchto obvodů a funkci, kterou lze často opakovaně měnit, si programuje sám zákazník. Vysoké náklady na vývoj ASIC jsou přesunuty k producentovi obvodů, ale jsou vykoupeny vysokou cenou samotného programovatelného obvodu. Zákazník programovatelné obvody vnímá jako samostatnou kategorii obvodů, zatímco výrobce jako zástupce „druhu“ ASIC.

Mezi hlavní zástupce programovatelných obvodů v dnešní době patří obvody CPLD (Complex PLD) a obvody FPGA (Field Programmable Gate Array). Právě FPGA obvody se v poslední dekádě dostaly do pozornosti širší návrhářské veřejnosti. Cena a komplexita tradičních ASIC obvodů totiž raketově stoupá a pro malosériovou výrobu přestávají být ASIC návrhy rentabilní. Naproti tomu se produkce výrobně jednodušších FPGA (z pohledu struktury) každoročně zvyšuje, což umožňuje snižovat jejich cenu. Zatím neutuchající technologický pokrok v podobě miniaturizace integrovaných obvodů se pozitivně projevuje zejména na výkonu a spotřebě FPGA, jejichž parametry se jeví pro řadu aplikací více než dostatečné.

2.1 Struktura obvodů FPGA

Přestože určitá základní podobnost mezi různými rodinami obvodů FPGA existuje, není možné jejich strukturu zcela generalizovat. Z tohoto důvodu se v následujícím textu zaměříme na FPGA obvody firmy Xilinx a to především na hradlová pole řady Virtex-4 [54].

FPGA obvod tvoří do matice uspořádaná soustava různých programovatelných bloků vzájemně spojených přepínatelnou propojovací maticí.

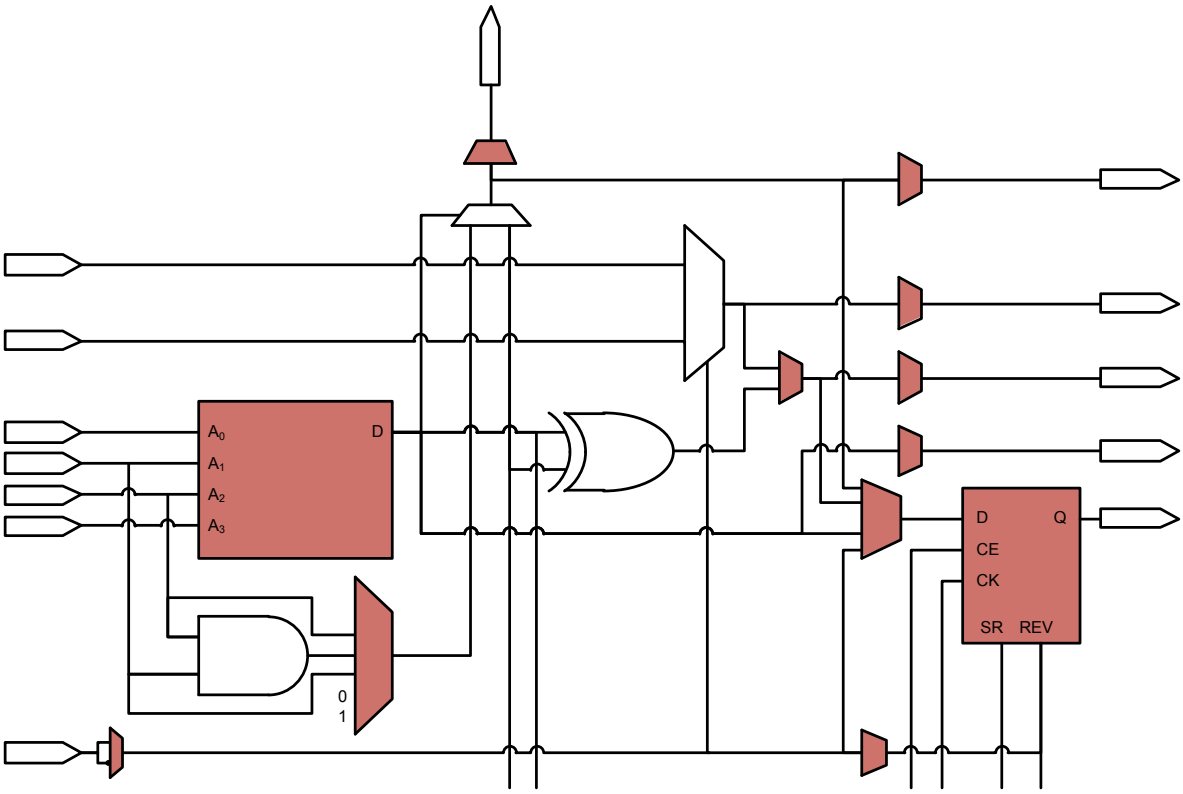


Obrázek 3: Struktura hradlového pole

Největší procento programovatelných bloků je zastoupeno makrobloky CLB (Complex Logic Block). Ty se skládají ze dvou dvojic podbloků nazývaných SLICE-L (obrázek 4) a SLICE-M. Každý SLICE je tvořen dvojicí 16 bitových vyhledávacích tabulek LUT (Look Up Table), výstupními multiplexory, dedikovanou aritmetickou logikou a konfigurovatelnými výstupními klopnými obvody typu D.

LUT může pracovat v dvou různých režimech. V SLICE typu L je ji možné použít pouze jako generátor libovolné logické funkce čtyř vstupních proměnných nebo jako paměť ROM. V SLICE typu M vyjma generátorického režimu může LUT fungovat též jako 16 bitová paměť typu RAM nebo posuvný registr s volitelnou délkou.

V moderních obvodech FPGA, rodinu Virtex-4 nevyjímaje, tvoří další podstatné zdroje složitější obvodové prvky, které by v případě, že by byly vytvářeny z vyhledávacích tabulek, zabíraly buď příliš mnoho zdrojů obvodu, nebo by nedosahovaly konkurenceschopných výkonů. Těmito bloky bývají (seřazeno dle míry zastoupení) konfigurovatelné bloky paměti RAM/FIFO (BRAM), hardwarové MAC (Multiply-ACcumulate) bloky, rychlá sériová rozhraní a v některých obvodech dokonce procesory. Při popisu hradlových polí nelze vynechat konfigurovatelné vstupně-výstupní bloky splňující širokou škálu napěťových standardů.

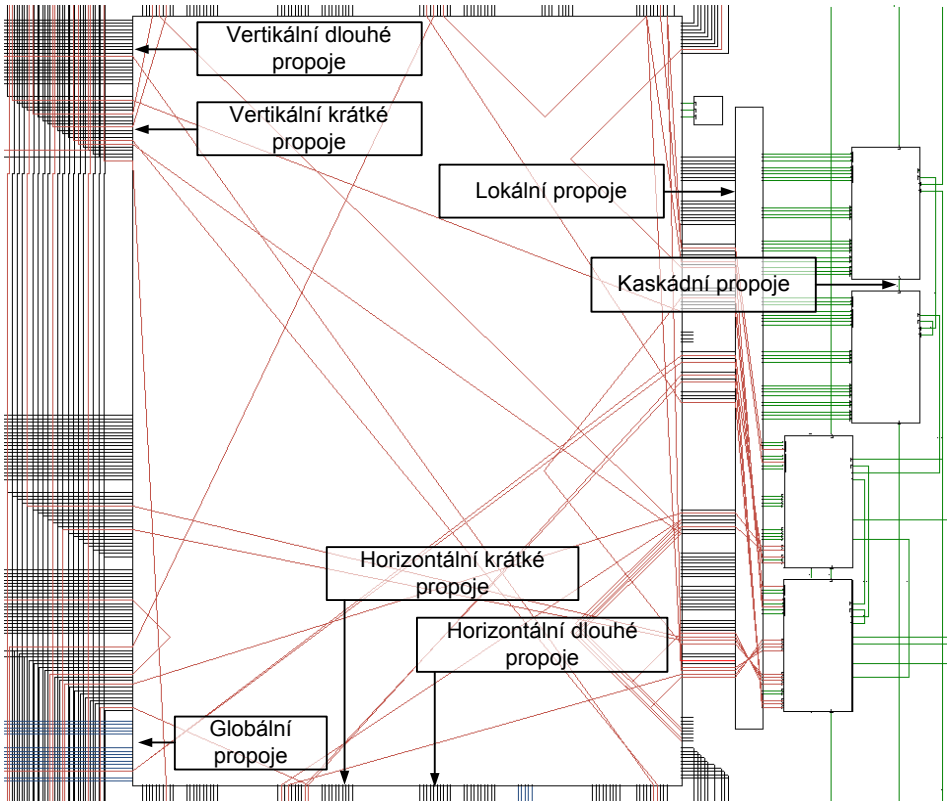


Obrázek 4: Část SLICE-L

2.2 Propojovací matice

Vodiče v hradlových polích jsou seskupovány a rovnoměrně, po řádcích a sloupcích rozmístěny po celém čipu. V místech, kde se vertikální a horizontální propoje protínají jsou umístěny konfigurovatelné přepínače (obrázek 5) umožňující omezené propojení vodičů. Aby nemohlo dojít k situaci, že nebude možno návrh realizovat kvůli nedostatku vodičů, bývá počet vodičů v obvodu značně předimenzován. V obvodech řady Virtex-4 lze nalézt celou řadu programovatelných propojů:

- rychlé a krátké kaskádní propoje mezi sousedními LUT sloužící k realizaci aritmetických funkcí
- lokální propoje spojující blízké CLB
- dlouhé a globální propoje spojující vzdálené CLB
- dedikované rozvody hodinových signálů



Obrázek 5: Přepínací element FPGA se znázorněnými druhy propojů

2.3 Konfigurační paměť

Funkce obvodu implementovaného v FPGA je dána nastavením programovatelných bloků a jejich vzájemným propojením. V případě rodin Virtex-4 je nastavení obvodu ovládáno obsahem volatilní konfigurační paměti typu SRAM. Data, která jsou do paměti ukládána, se vytváří v poslední fázi procesu implementace obvodu a ukládají se do konfiguračního souboru, označovaného jako bitstream. Konfigurační soubor obsahuje jak konfigurační data, tak příkazy pro řadič konfigurační paměti. Lze rozlišovat bitsream – soubor příkazů a konfiguračních dat a samotná konfigurační data, která jsou výrobcem označována jako raw bitsream.

Kromě konfigurace programovatelných prvků je konfigurační paměť též využívána k implementaci vnořené paměti (BRAM nebo LUT v režimu RAM). V takovém případě nabývá konfigurační paměť charakteru zdroje obvodu a konfigurační soubor slouží k jeho počáteční inicializaci. Je-li hodnota vnitřních pamětí změněna, mění se zároveň i obsah konfigurační paměti. Konfigurovatelné klopné obvody tvoří v tomto pravidle výjimku – bitstream slouží k jejich inicializaci (nastavení hodnoty po

resetu), avšak jejich aktuální hodnotu nelze opětovným vyčtením konfiguračních dat získat nebo ji zápisem změnit.

2.4 Rekonfigurace obvodu

V případě kompletní změny konfigurační paměti hovoříme o úplné rekonfiguraci. Takový případ nastane nejčastěji při zapnutí obvodu a je podmíněn volatilním charakterem konfigurační paměti.

V případě, že nepřepíšeme celou konfigurační paměť, ale pouze její část, změníme pouze část funkce, která je příslušnou pamětí ovládána. Takovou konfiguraci nazýváme částečnou konfigurací.

SRAM obvodů řady Virtex-4 je bezzákmitová. Je-li paměťová buňka přepsána stejnou hodnotou, nedochází na výstupu této buňky při přepisování k žádnému zákmitu. Toho lze využít a změnit část konfiguračních dat bez toho aniž by došlo k ovlivnění zbytku obvodu. Je-li měněna část konfigurační paměti za běhu zbytku obvodu, mluvíme o částečné dynamické rekonfiguraci.

Mění-li se základní elementy obvodu (vyhledávací tabulky, vzájemné propojení apod.) hovoříme o jemnozrnné rekonfiguraci. O hrubozrnnou rekonfiguraci se jedná v případě změny větších celků, například oblastí CLB i s propojením.

2.5 Poruchy FPGA založených na SRAM

Ačkoliv se FPGA řadí mezi obvody programovatelné, jedná se o obvody typu ASIC a jako takové je mohou postihovat všechny defekty ASIC obvodů. Většina trvalých poruch FPGA je způsobena nedokonalostmi ve výrobním procesu a je odhalena při výrobních testech. Defektní obvody jsou vyřazeny nebo jsou ve spolupráci se zákazníkem využity pro specifické aplikace. Z dlouhodobých měření výrobců [50] lze usuzovat, že jsou FPGA obvody pro běžné aplikace dlouhodobě spolehlivé.

V poslední dekádě se začaly FPGA používat v oblasti letectví, kosmonautiky, medicínské techniky, apod., kde jsou nároky na spolehlivost podstatně vyšší. FPGA zde získávají uplatnění hlavně kvůli nízké ceně vývoje, dobrému poměru výpočetního výkonu ke spotřebě a rekonfigurovatelnosti umožňující určitou míru samoopravitelnosti, která je v dlouhodobých „misích“ výhodná. Právě rekonfigurovatelnost FPGA založených na statické paměti je Achillovou patou jejich spolehlivosti. V leteckých a vesmírných aplikacích dochází v mnohem větší míře k

poruchám typu SEU [29]. Výskyt a projevy těchto poruch jsou u FPGA obvodů založených na SRAM markantnější, nežli u ASIC obvodů:

- Největší část obvodu FPGA (plochu obvodu) zabírá konfigurační paměť. Pravděpodobnost vzniku SEU je tak vyšší než u obvodů, ve kterých převládá kombinační logika.
- SEU v konfigurační paměti není možné okamžitě detekovat. U ASIC obvodů lze paměť opatřit kontrolními a opravnými kódy, u hradlových polí se toto většinou nevyužívá.
- Porucha typu SEU se v FPGA může projevit jako trvalá porucha, protože SRAM paměť řídí konfiguraci obvodu, na rozdíl od ASIC, kde bývá použita především k uchování dat.
- Za určitých okolností může dojít vlivem SEU k trvalé poruše obvodu – např. zkratováním výstupů LUT. U ASIC hrozí „pouze“ ztráta dat.

Význam jednotlivých bitů konfigurační paměti není přímo znám (ochrana duševního vlastnictví), ale z prosté úvahy (podpořené např. [16]) lze usoudit, že nejvíce poškozenou částí FPGA bude propojovací matice. Se zmenšující se velikostí tranzistorů v FPGA obvodech je pravděpodobné, že se SEU začnou vyskytovat i za běžných podmínek [29].

2.6 Techniky testování FPGA

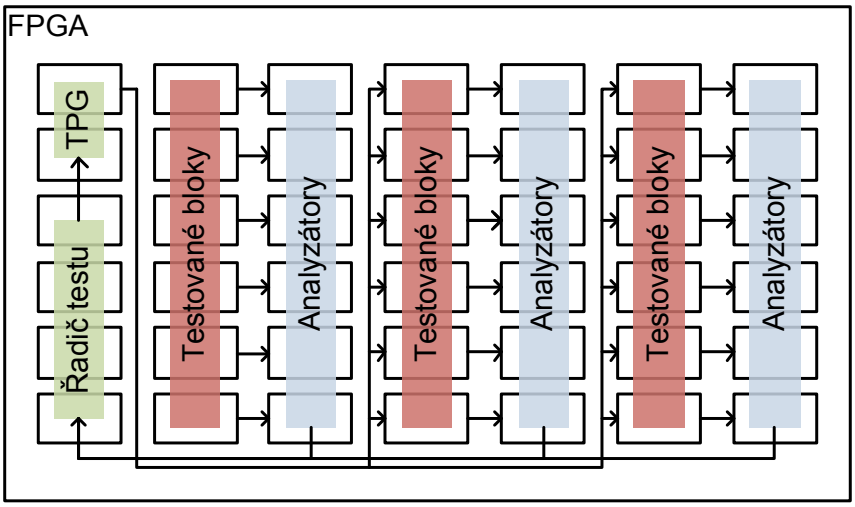
Metody testování lze rozdělit do několika kategorií [48]. Testy, které pro testování obvodu vyžadují zastavení činnosti obvodu, bývají označovány jako off-line testy. Testy označované jako on-line testy testují obvod za běhu tak, aby nebyla jeho funkce narušena. Off-line testy často využívají úplnou rekonfiguraci obvodu, zatímco online testy využívají částečnou rekonfiguraci. Testy nezávislé na struktuře implementovaného obvodu (aplikačně nezávislé) testují všechny prvky FPGA bez ohledu na to, zdali jsou využity implementovaným obvodem, zatímco testy závislé na struktuře implementovaném obvodu (aplikačně závislé) testují pouze ty zdroje FPGA, které implementovaný obvod využívá.

2.6.1 Aplikačně nezávislé testy

Testy nezávislé na implementovaném obvodu jsou založeny na podobném principu (např. [9]). Obvod je pomyslně rozdělen na řádky nebo sloupce viz obrázek 4. Každý řádek může být konfigurován jako generátor testovacích vektorů (TPG),

testovaný řádek a analyzátor odezev (ORA). V průběhu testu se funkce řádků střídá tak, aby byl otestován celý obvod.

Konfigurace testovaného řádku zpravidla umožňuje testovat pouze jeden z mnoha programovatelných bloků v FPGA (např. CLB). Vzhledem ke komplexnosti bloků jedna konfigurace daného prvku nepostačuje a je nutné provést více testů při různých nastaveních programovatelného bloku, což podstatně navyšuje režii testu v podobě nároků na paměť a čas testu [43].



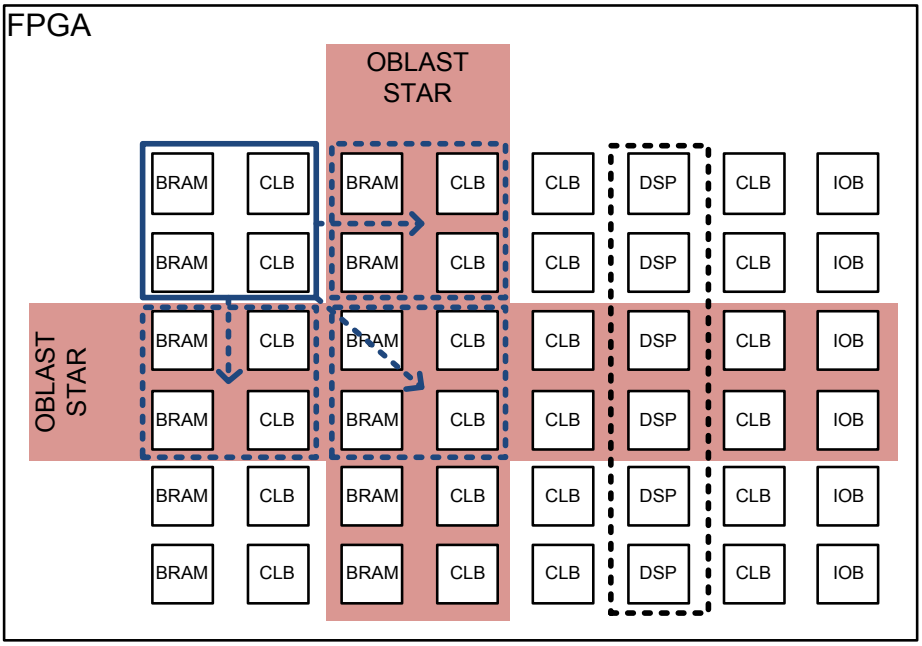
Obrázek 6: Možná implementace offline strukturního testu

Pro testování propojovací matice [38] se užívá obdobného principu. Při rozdělení obvodu na řádky lze testovat vertikální a šikmé propoje, pro test horizontálních propojů je nutné použít sloupcové rozdělení obvodu. Chceme-li testovat dlouhé propoje, je nutné TPG a ORA vzdálit o více než jeden sloupec nebo řádek.

Mezi představitele online testů patří např. roving STAR algoritmus [8]. FPGA obvod je rozdělen na stejně široké sloupce a řádky tak, že alespoň jeden sloupec a řádek zůstává neobsazen. Do neobsazeného prostoru je umístěn STAR (Self Testing ARea). V oblasti STAR je možné provádět testování, které je velmi podobné testu popsanému v předchozím textu a zbytek obvodu, který není testováním ovlivněn, může vykonávat původní funkci. Test jiné části obvodu je realizován tak, že se STAR přesune do jiného sloupce obvodu. Při přesouvání STAR je nutné činnost obvodu pozastavit. Postupným přesouváním oblasti STAR je teoreticky možné otestovat celý obvod, aniž by byla narušena jeho funkce.

Roving STAR algoritmu je obtížně implementovatelný. Moderní FPGA obsahují mnoho různorodých prvků, kterou jsou rozložené ve sloupcích, a tak není zaručeno, že bude možné oblasti STAR posouvat libovolně. Situace je naznačena na

obrázku 7. Černým čárkováním je označena oblast jediného sloupce FPGA, ve kterém se nacházejí DSP bloky, které mohou být buď využity funkční částí obvodu, nebo mohou být testovány. Obdobným problémem je propojení rozdělených oblastí obvodu, které vyžaduje alokaci vodičů v oblasti STAR a tím znemožňuje jejich otestování. Obdobně jako v případě off-line testu zmíněného výše je paměťová režie testu značná, poněvadž je nutno uložit nejméně čtyři různé konfigurace pro každý blok FPGA – viz modře označené oblasti v obrázku 7.



Obrázek 7: Roving STAR algoritmus

2.6.2 Aplikačně závislé testy

Na obrázku 4 jsou vybarveny ty prvky ze SLICE-L, které jsou přímo ovládané z konfigurační paměti, nevybarvené jsou ty, které nejsou přímo konfigurovatelné, ale na nastavení vybarvených prvků závisí. Lze tvrdit, že není možno nakonfigurovat SLICE tak, aby byly použity všechny prvky, poněvadž by docházelo ke konfliktům (např. zkraty mezi budiči). Právě to je kamenem úrazu aplikačně nezávislých testů. Chceme-li všechny prvky v programovatelném bloku otestovat, musí být provedeno několik rekonfigurací obvodu, které jsou časově náročné.

Aplikačně závislé testování tento problém potírá tím, že testuje pouze ty prvky, které implementovaný obvod využívá. Míra redukce testovaných prvků je značná, uvážíme-li, že žádný obvod nemůže najednou využít 100% zdrojů obvodu a v potaz vezmeme fakt, že většina „konfigurovatelnosti“ hradlového pole leží v programovatelné propojovací matici, jejíž využití je velmi nízké.

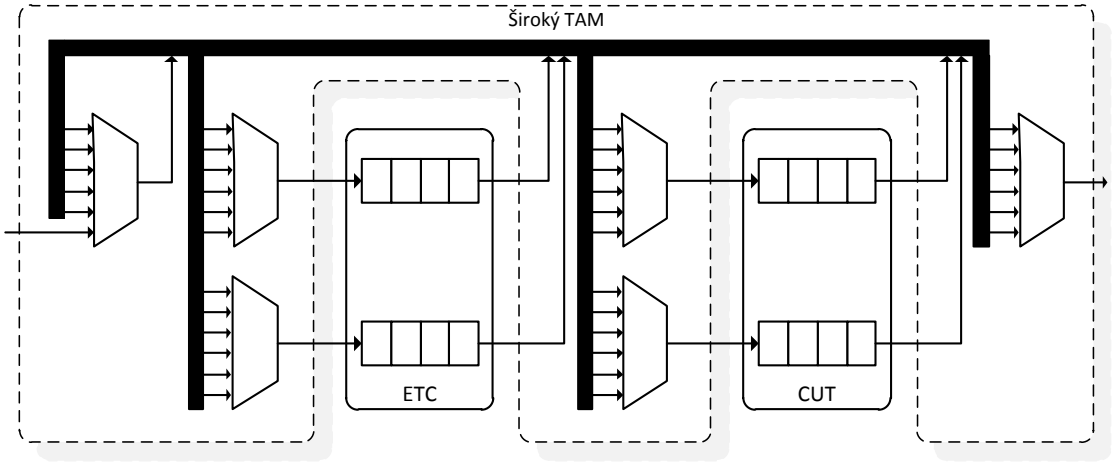
Základním aplikačně závislým testem je jednoduchý test konfigurační paměti, který se nazývá bitstream scrubbing [16]. Je založen na systematickém vyčítání konfiguračních dat z paměti zařízení a jejich porovnávání s referenčním vzorkem uloženým v paměti chráněné proti radiaci. Někdy je z dat pouze vypočítán kontrolní součet a ten je porovnáván s referenčním součtem. Metoda umožňuje omezenou detekci poruch SEU, detekci trvalých poruch umožňuje pouze v případě, že je nějakým způsobem porušen hardware související s konfigurační paměti. Scrubbing selhává i pro testování těch částí konfigurační paměti, které jsou použity jako uživatelsky přístupná paměť RAM, poněvadž není možné uvažovat případnou změnu této paměti při porovnávání s referenčním vzorkem/příznakem. Pro svou jednoduchost bývá bitstream scrubbing implementována přímo v FPGA [11].

K testování FPGA se vybízí využít technik BIST důvěrně známých z obvodů ASIC. S pomocí těchto technik by bylo možné testovat obvod jak na některé poruchy SEU, tak na trvalé strukturní poruchy. Do obvodu implementovaného v FPGA je možné vložit skenovací řetězce a jako generátor testu použít zpětnovazebný registr, celulární automat nebo paměť uchovávající strukturní testovací vektory. Potíž je jak se skenovacími řetězci, tak i s testovacími daty. Vkládáním skenovacích řetězců se značně zhoršují parametry časování a až dvojnásobně [37] se zvyšuje velikost obvodu. Tím, že je struktura FPGA výrobcem tajena je velmi obtížné byť jen ověřit účinnost libovolného testu, natož sestavit test strukturní. Bez značných úprav CAD nástrojů není vhodné BIST do hradlových polí vkládat a ani to nebývá využíváno.

2.6.3 RESPIN, DyRESPIN

RESPIN [17] je testovací architektura navržená pro testování vícejádrových obvodů na čipu (System on a Chip - SoC) s pomocí komprimovaných testovacích vektorů. Každé z jader v obvodu je vybaveno řídící obálkou (wrapper) dle standardu IEEE 1500 [23]. Jádro, respektive jeho obálka, může být nastaveno ve třech režimech - funkčním, testovaném (Core Under Test - CUT) a testovacím (Embedded Tester Core - ETC). Zdaleka nejzajímavější je testovací režim, ve kterém jsou skenovací řetězce uvnitř jádra využity k dekompresi testovacích vzorků s vysokou mírou kompakce. V obvodu není potřeba vytvářet dedikovaný hardware určený pouze k dekompresi testovacích vektorů a tak dochází k výrazné úspoře zdrojů obvodu. Tím že testovací vzorky dekomprimujeme až v obvodu, snížíme dobu/cenu testu a zároveň zvýšíme i dostupnost obvodu.

V architektuře RESPIN existují dva testovací přístupy (Test Access Mechanism; TAM) – široký a úzký. Úzký testovací přístup je určen pro přivádění zkomprimovaných testovacích vektorů z ATE do ETC a jeho přenosová kapacita je nízká. Široký testovací přístup se nachází mezi testujícím a testovaným jádrem a jeho přenosová kapacita je podstatně vyšší. Časová úspora architektury RESPIN spočívá v tom, že se zkomprimovaná data do řetězců mohou přenášet poměrně pomalu, zatímco dekomprimovaný test je možno díky rychlému, často paralelnímu TAM přenést k testovanému jádru velmi rychle.



Obrázek 8: Široký TAM založený na multiplexorech

Jedním z důvodů proč není vhodné původní RESPIN implementovat v obvodech FPGA je způsob, kterým je realizován široký testovací přístup (viz obrázek 8). Široký TAM je vytvořen pomocí sběrnice založené na multiplexorech, která je pro větší počty skenovacích řetězců k implementaci na FPGA obvodech nevhodná nejen kvůli značné spotřebě zdrojů obvodu, ale i kvůli vysokému kombinačnímu zpoždění.

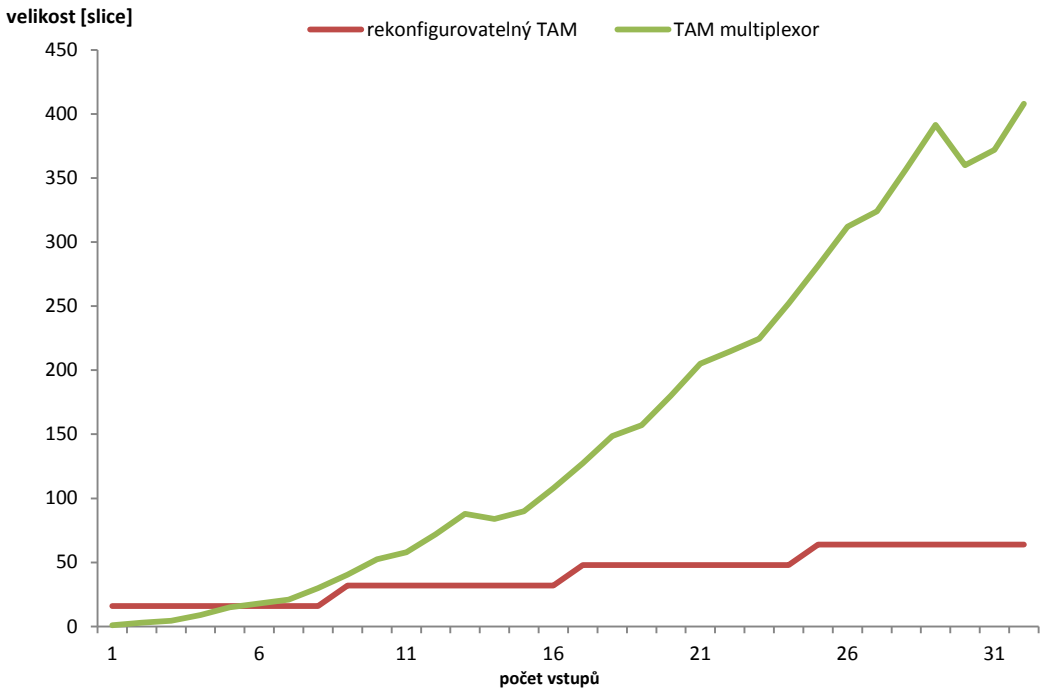
DyRESPIN [34] je testovací architektura určená pro programovatelné obvody, která vychází z konceptu architektury RESPIN a je obohacená o mechanismus rekonfigurace obvodu. V systému DyRESPIN je široký TAM nahrazen rekonfigurovatelným blokem o n vstupech a n výstupech, kde n odpovídá počtu skenovacích řetězců v obvodu. Namísto změny signálů ovládajících multiplexoru je dynamicky měněno propojení vstupů a výstupů uvnitř rekonfigurovatelného modulu. Změna je realizována prostým přeprogramováním propojovací matice, která je sama o sobě druhem konfigurovatelné sběrnice. Nespornou výhodou přímého využití propojovací matice je vysoká a pro různé varianty TAM prakticky konstantní frekvence, na které může sběrnice operovat, poněvadž v rekonfigurovatelné bloku

TAM prakticky neexistuje kombinační logika (nepočítáme-li oddělení rekonfigurovatelného modulu od statické části systému). DyRESPIN implementovaný na platformě Virtex-4 byl prezentován v [3].

Nevýhodou rekonfigurovatelné sběrnice je dlouhá doba přestavování obvodu daná rychlostí přístupu do konfigurační paměti, velikostí rekonfigurovatelné oblasti a u sběrnice paradoxně značná paměťová náročnost, protože pro každé možné propojení testovaného a testovacího jádra musí být uložen unikátní konfigurační soubor v paměti.

Jak doba rekonfigurace, tak paměťová náročnost rekonfigurovatelné sběrnice narůstá společně s počtem skenovacích řetězců v obvodu. Přímé porovnání se sběrnicí založenou na multiplexorech se nachází na obrázku 9. Oproti exponenciálnímu nárůstu u sběrnice založené na multiplexorech, lze u sběrnice rekonfigurovatelné pozorovat nárůst pouze lineární.

Je nutno poznamenat, že v systému DyRESPIN je ATE nahrazeno vnitřní nebo vnější pamětí uchovávající zkomprimované testovací vzorky. Výhoda komprese ale zůstává zachována – je zapotřebí výrazně méně paměti, než kdybychom uchovávali kompletní testovací data, a není k nim třeba tak často přistupovat.



Obrázek 9: Závislost velikosti TAM na počtu skenovacích řetězců

DyRESPIN neodstraňuje nevýhody spojené se vkládáním skenovacích řetězců do hradlového pole a žádným způsobem nezohledňuje rozdíly mezi implementací

testovaného obvodu v technologii ASIC nebo FPGA. Tyto nevýhody by mělo vyvažovat znovuvyužití hardware v podobě dekompresoru vytvořeného ze skenovacích řetězců a implementace TAM s pomocí částečné dynamické rekonfigurace.

Kontroverzní je způsob, kterým je rekonfigurace FPGA využita, poněvadž je sporné, zda dynamický TAM skutečně navyšuje funkční hustotu programovatelného obvodu. Drtivá většina hardware je v obvodu implementována staticky, dynamicky je implementovaná pouze testovací sběrnice, která tvoří jen nepatrný zlomek testovacího vybavení systému a je navíc v systému přítomna vždy.

3 Navržená metoda testování

Metody testování FPGA představené v předchozím textu mají několik společných charakteristik. Shrňme je v několika stručných bodech:

- Aplikačně nezávislé metody:
    - Vyžadují detailní znalosti struktury obvodu, která nebývá u nových generací FPGA k dispozici. Strukturní závislost omezuje přenositelnost metod mezi různými architekturami hradlových polí.
    - Časová a paměťová náročnost způsobená nutností vícenásobných rekonfigurací. Se zvyšující se heterogenitou obvodů nároky stoupají a test se prodlužuje a zabírá více místa v paměti.
    - Předimenzované testy. Testovány jsou všechny zdroje obvodu, bez ohledu na jejich reálné využití. Některé typy poruch (např. SEU, zpoždění) není možné testovat.
- Aplikačně závislé metody:
    - Metody založené na vyčítání konfigurační paměti mají nízkou závislost na struktuře FPGA, jsou snadno implementovatelné a lehce přenositelné na jiná FPGA. Netestují však strukturní poruchy, ani všechny poruchy SEU.
    - Metody založené na BIST vyžadují přístup k testovanému obvodu, který na hradlových polích významně zvyšuje spotřebu zdrojů obvodu a zhoršuje časovací charakteristiky. Strukturní závislost BIST je úměrná modelu FPGA obvodu, časová a paměťová náročnost je závislá na způsobu implementace BIST.

Určit, která z metod testování FPGA obvodů je vhodnější samozřejmě nelze. Produkční testy nelze provádět jinak než s pomocí aplikačně nezávislých metod, zatímco test vysoce dostupného systému nemůže sestávat z desítek [48] kompletních rekonfigurací obvodu. Určitou váhu je nutno přiřadit i ceně testu. Představa řídící jednotky satelitu, která k testu vyžaduje průmyslové ATE je absurdní.

Představená metoda, jak její název napovídá, spadá do kategorie aplikačně závislých testů a je tudíž cílena na testy FPGA nasazených na konkrétní aplikaci. Metoda ideově vychází z architektury DyRESPIN, ze které si bere myšlenku znovuvyužití hardware pro testovací účely. DyRESPIN však dostatečně nerozvíjí

vlastnosti, které FPGA odlišují od obvodů ASIC. Především schopnost dynamické rekonfigurace může přispět k testování obvodu v daleko větší míře. Částečnou rekonfiguraci metoda využije k dynamickému vytváření vnořeného testovacího vybavení, k zajištění přenosu testovacích vzorků a ke zvýšení kontrolovatelnosti a pozorovatelnosti testovaného obvodu.

I přes určité odlišnosti bude metodu možno klasifikovat jako variantu aplikačně závislého BIST. Metoda se tak bude potýkat se závislostí na struktuře testovaného obvodu. Domnívám se, že tuto závislost je možné z velké míry odbourat použitím vhodného modelu obvodu; i při jen částečné znalosti struktury lze vytvořit model obvodu dostatečně kvalitní pro potřeby testování. Založení metody na BIST by však mělo přinést nesporné výhody v podobě postupů, nástrojů a technik, které jsou dlouhodobě aplikovány a prověřeny na obvodech ASIC. Pro specifické prostředí obvodů FPGA je však bude třeba přizpůsobit.

Podstatnou výhodou metody je i fakt, že většina obdobných prací se zaměřuje buď na zvýšení zabezpečení obvodu (např. [40]), anebo na aplikačně nezávislé testování specifického zdroje hradlového pole (např. [38]). Nevyužitý prostor, který tak vzniká, může metoda inovativně zaplnit.

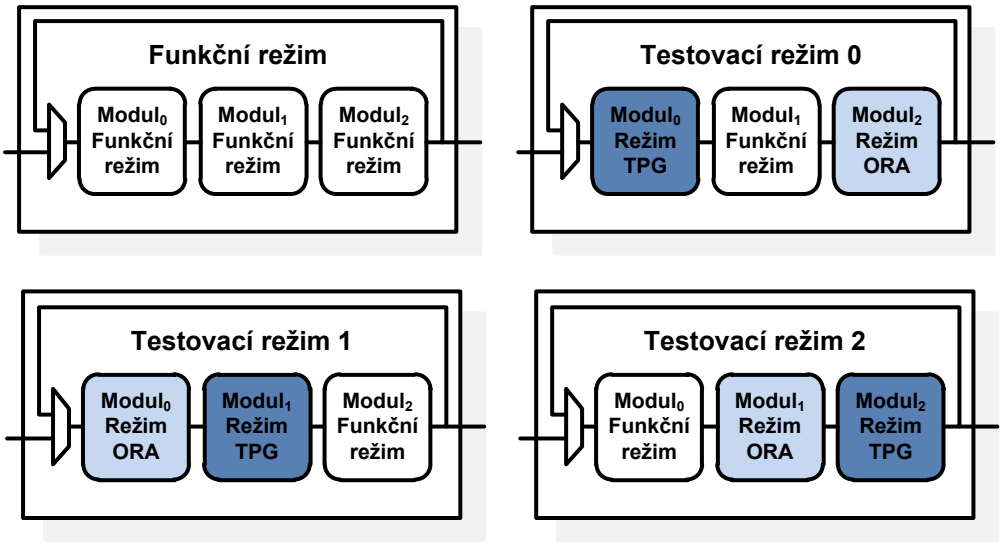
3.1 Přehled metody

Zjednodušený princip metody je následující: Návrh obvodu implementovaný v FPGA musí být rozdělen na nejméně tři části. Každá část je upravena a implementována jako rekonfigurovatelný modul. Moduly jsou doplněny o obálku, která umožňuje minimalistické řízení testu a zároveň sjednocuje přístup ke každému jádru. Obálka zajistí, že je ke každému modulu možné jednoduše vytvořit další obvody – generátor testovacích vektorů (TPG) a analyzátor odezev (ORA). Pro samotný princip metody není vnitřní zapojení TPG ani ORA klíčové.

Testování jednoho z modulů ve funkčním režimu probíhá tak, že je jeho „pravý“ soused překonfigurován na ORA a posléze „levý“ soused na TPG (viz obrázek 10). Po ukončení testu jsou okolní moduly navráceny do funkčního režimu. Cyklickým posouváním generátoru a analyzátoru po rozděleném obvodu je možné celý obvod otestovat. Na hradlovém poli může být takovým způsobem implementováno více obvodů, ve kterých může testování probíhat paralelně.

Přístup k testovanému obvodu je řešen přes zachované paralelní vstupy a výstupy. Mezi primárními vstupy a výstupy obou krajních sousedů je vytvořena zpětná

vazba tak, aby se mohly vzájemně testovat. Každá z obálek je doplněna o sériové řízení testu, aby bylo možné testování zahájit a po ukončení testu vyčíst příznak nebo informaci o úspěšnosti testu. Testovací signály jsou spolu s řadičem testu cyklicky propojeny, podobně, jako tomu bývá například u rozhraní JTAG [24].



Obrázek 10: Přesouvání modulů v obvodu

Protože ke generování testovacích vektorů chceme využít nástroje běžné pro obvody ASIC, bude nutné popis obvodů s FPGA prvky přepsat do strukturního netlistu na bázi ASIC prvků. Deterministický test může být oproti pseudonáhodnému testu výhodnější co do počtu testovacích vektorů a tím pádem bude obvod testován kratší dobu.

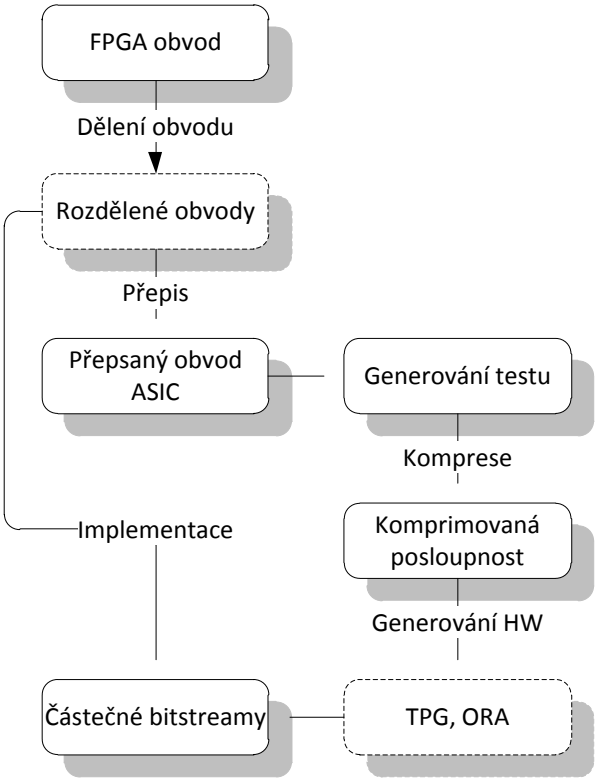
Paměťovou režii, která využitím deterministických vektorů vzroste, je možné kompenzovat využitím komprese testovacích vektorů, se kterou máme dlouhodobé zkušenosti [26]. Testovací vektory lze uložit do některého z paměťových prvků FPGA ve formě inicializace konfiguračních dat. Využitím částečné dynamické rekonfigurace se tak sníží nároky testeru na paměť, ale i na zdroje FPGA, poněvadž se v FPGA bude trvale nacházet jen minimum logiky určené pro testování a pro režii rekonfigurace.

Rekonfigurace okolních modulů bude fungovat podobně jako metoda bitstream scrubing, s tím rozdílem, že namísto porovnávání aktuálního obsahu konfigurační paměti s referenčním se bude při vracení okolních modulů do funkčního režimu konfigurační paměť obnovovat a tím bude docházet k odstranění přechodných poruch.

Tím, že v testovaném modulu (modul ve funkčním režimu) nezměníme konfigurační paměť, umožníme částečnou detekci SEU. Metoda odhalí pouze ty poruchy v konfigurační paměti, které vedou ke změně funkce implementovaného

obvodu. Je zdokumentováno (např. v [16], [25]), že většina poruch konfigurační paměti na funkci obvodu vliv nemá.

Implementace metody sestává z několika dílčích kroků, které budou v následujícím textu představeny. Celý postup aplikace metody je znázorněn na obrázku 11.



Obrázek 11: Dílčí kroky metody

3.2 Dělení a přepis obvodu

Navržená metoda ke své funkčnosti vyžaduje rozdělení obvodu na pravidelné, podobně veliké části. Na rozdíl od architektury RESPIN jsou jednotlivé části obvodu podstatně menší. Systém není rozdělen na jádra, ale vzájemné testování probíhá na úrovni podobvodů jednotlivých jader.

Velké množství současných návrhů pro obvody FPGA využívá tzv proudového zpracování (pipelining). V pipeline návrhu je kombinační logika rozdělena různým počtem vnořených klopných obvodů. Ty zabraňují vytvoření příliš dlouhé kombinační cesty a tím umožňují běh obvodu na vyšší frekvenci. Cenou za proudové zpracování je latence, která je rovna počtu stupňů pipeline.

Klopné obvody v pipeline vytváří přirozenou hranici, podél níž se vyplatí obvod dělit. Obvod bývá navržen tak, aby žádná z částí nebyla podstatně větší než ostatní. Tím by mělo být zajištěno i podobné kombinační zpoždění všech stupňů

pipeline. Pokud tomu tak není, je možné využít např. syntézní volby register retiming [55][53], která posouvá registry v obvodu tak, aby bylo dosaženo vylepšení pracovní frekvence.

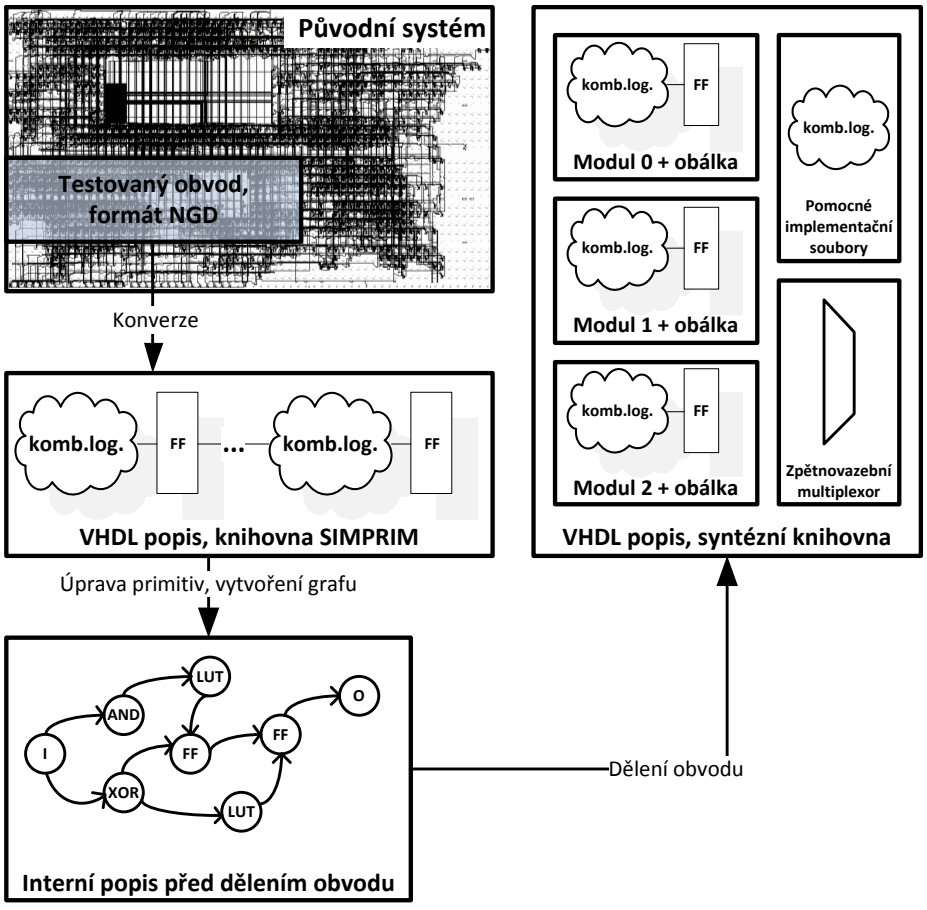
Problém při dělení obvodu vyvstává v případě, když obvod, nebo jeho část, obsahuje stavový automat a tím pádem i zpětné vazby mezi registry. Zpětné vazby je možno virtuálně odstranit vložením skenovacího řetězce nebo sestavit sekvenční test. Vložením skenovacího řetězce dochází ke zhoršení provozních parametrů a navýšení režie testování. Pokud by bylo zpětných vazeb v obvodu příliš velké množství, pak by došlo k extrémnímu navýšení režie a metodu by nebylo vhodné aplikovat. Lze však spoléhat na návrhářská pravidla, která doporučují oddělení řadiče od datové cesty. Počet registrů se zpětnou vazbou je v „dobrém“ návrhu nižší než počet registrů v pipeline. Při vložení skenovacího řetězce ke zhoršení parametrů a režie dojde, ale zhoršení nebude kritické.

Před samotným dělením je nutné implementovaný obvod převést z interního databázového formátu na (počítačem) čitelný strukturní netlist. Informace o struktuře lze získat ze souborů vytvářených v průběhu procesu implementace. Ta probíhá v několika fázích – syntéza, mapování, umístění a vytváření konfiguračního souboru a jsou při ní vytvářeny tři typy netlistů. Výstupem syntézy je netlist ve formátu NGC, který je složený z obecných prvků, které nemusí mít se skutečnými technologickými primitivy nic společného. Takto popsaný obvod je dále zpracován programy, které mapují (MAP) syntézní primitiva na primitiva technologická a následně je rozmisťují a propojují uvnitř návrhářem zvoleného hradlového pole (PAR: Place And Route). Meziprodukty procesů MAP a PAR jsou databázové soubory NCD (Native Circuit Description). Namapovaný netlist je možné procesem zpětné syntézy přepsat [28] do obecného popisu obsahujícího přesné informace o umístění a typu primitiv.

Způsobů, jak přepis z interních databázových formátů získat ve zpracovatelné podobě je několik. V [21] je uvedeno řešení s využitím strukturovaného, lidsky čitelného souboru XDL, který lze vygenerovat pomocí programu XDL translator a příslušného databázového souboru. XDL soubor obsahuje detailní, hierarchicky členěný popis modulů použitých v implementovaném obvodu. Každý modul je rozepsán až na úroveň prvků SLICE a jsou v něm zastoupeny i prvky, které nejsou ve slice využity (jsou vypnuty). Popis je možné použít k velmi komplexnímu strukturnímu popisu obvodu, případně jako pomocný popis v rámci druhé metody přepisu. Soubor XDL

není možno syntetizovat a nelze jej vytvořit z databázových souborů popisujících rekonfigurovatelné moduly.

Druhý způsob přepisu nemá omezení týkající se modulárních návrhů. Přepis lze získat programem Netgen [49], který je původně určen k vytváření HDL popisů pro časovou simulaci. V obvodu, který přepisem vznikne, nejsou zastoupeny všechna primitiva, ale pouze ta, která jsou využita. Na rozdíl od prvního řešení ani nebývá k přepisu přiložena tak detailní informace o umístění prvku v obvodu. Přepsaný soubor není dovoleno syntetizovat, pouze simulovat - za účelem syntézy je na souboru nutno provést několik úprav. V práci je pro přepis využíván právě tento postup, ačkoliv by první mohl být při vytváření modelu obvodu výhodnější.



Obrázek 12: Přepis a dělení testovaného obvodu

Rozdělení obvodu provádí skript *split*, který byl za tímto účelem vytvořen. Skript ke své činnosti vyžaduje VHDL popis obvodu získaný z přepisu modulárně implementovaného obvodu, který chceme testovat. Skript načte ze zdrojového souboru všechna primitiva a jejich vzájemné propoje a v paměti vytvoří orientovaný graf obvod reprezentující. S původním obvodem provádí skript následující operace:

- Odstraňuje všechny ryze simulační konstrukty.
- Odstraňuje izolované prvky.
- Odstraňuje některé zdroje trvalých logických nul a jedniček.
- Odstraňuje šíření hodinového signálu a nahrazuje jej vlastním šířením.
- Nahrazuje simulační primitiva primitivy syntézními nebo vytváří funkčně shodné prvky.

Jako příklad nahrazení uveďme nahrazení nedělitelných posuvných registrů implementovaných pomocí vyhledávací tabulky na běžný posuvný registr tvořený z klopných obvodů.

Po modifikaci obvodových prvků se pomocí algoritmů prohledávání do hloubky a do šířky skript v grafu snaží nalézt všechny cesty mezi primárními výstupy a klopnými obvody, které oddělují jednotlivé stupně pipeline. Po nalezení všech cest a odstranění redundantních prvků, které se na nich nacházejí, jsou cesty a prvky, které na nich leží zapsány do nového souboru a odstraněny z původního obvodu. Nově vytvořený modul je doplněn o unifikované signály řízení testu. Registry na hranici obvodu jsou interně přetypovány na primární výstupy. Algoritmus prohledávání se opakuje tak dlouho, dokud nenarazí na primární vstupy. Při každé iteraci je testována konzistence obvodu, především pak výskyt cyklů.

Poslední a první stupeň pipeline je nutno zpracovat odlišně, než zbytek obvodu. Výstupy posledního a vstupy prvního stupně jsou vzájemně porovnány. Je-li zjištěna neshoda, je obálka výstupního modulu doplněna tak, aby byly moduly vzájemně propojitelné. Prakticky to znamená, že skript doplní dodatečné primární výstupy posledního modulu, právě když je jejich počet nižší než počet primárních vstupů prvního modulu. Okrajové stupně obvodu jsou doplněny o multiplexor, který umožňuje cyklické testování všech modulů rozděleného obvodu. Modifikaci obálek a vytváření zpětnovazebního multiplexoru zajišťuje samostatný skript *make_feedback*, který navazuje na skript předchozí. *Make_feedback* k modifikovaným modulům vytváří pomocné soubory, které slouží ke zjednodušení procesu implementace rekonfigurovatelného systému (viz 3.7).

3.3 Přepis pro ASIC ATPG

Po přepsání, rozdělení a modifikaci prvků strukturního popisu testovaného obvodu je možné přistoupit k procesu, jehož výstupem je deterministický strukturní test „šitý na míru“ implementovanému obvodu a použitému hradlovému poli.

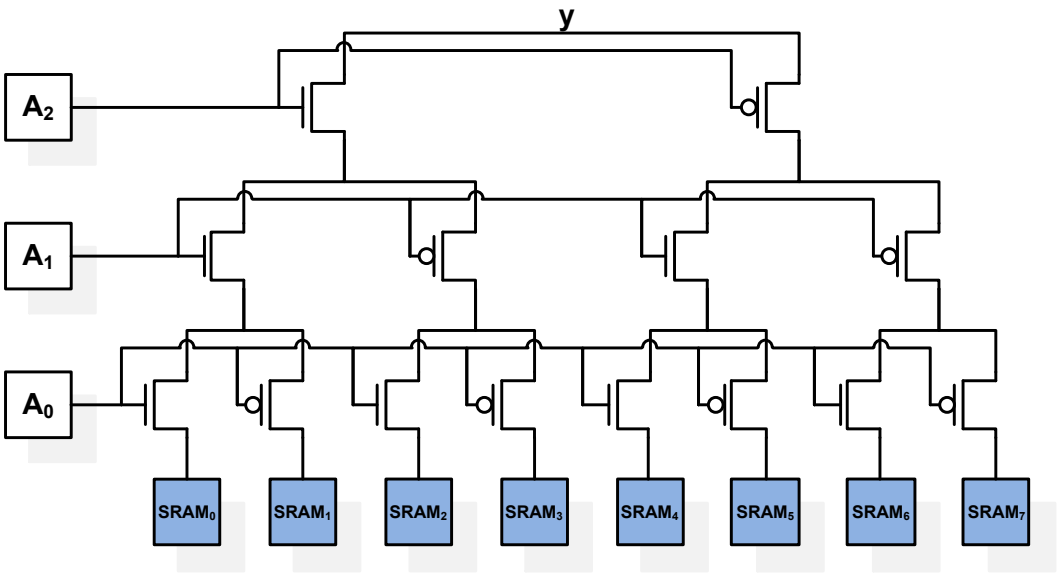
Před samotnou implementací je nutné všechny moduly rozděleného obvodu syntetizovat. Standardní nastavení syntézních voleb je pro generování testu nevhodné a je nutné zvolit poněkud neobvyklé volby; především ty, které se týkají optimalizace sekvenčních prvků. Při běžném nastavení se snaží syntézní nástroj využít synchronních vstupů set a reset klopných obvodů tak, aby nebylo nutné vytvářet dodatečnou kombinační logiku. Pro ilustraci si představme obvod, který při splnění určité podmínky na svém, registrem opatřeném výstupu nastavuje nulu. Namísto vytvoření multiplexoru doplněného klopným obvodem použije syntéza signál vyhodnocení podmínky k řízení vstupu reset registru. Tímto sice ušetří zdroje obvodu a sníží kombinační zpoždění, ale prakticky znemožní testování, poněvadž testovaný obvod ovládá sekvenční prvky, což je problematické nejen při posouvání vzorků ve skenovacím řetězci, ale i při pozorování obvodu u návrhu bez skenovacích řetězců [15]. Pro potřeby testu je nutné volby použití synchronních set/reset signálů vypnout.

Po syntéze obvodu a jeho implementaci (viz 3.7.3) je obvod konvertován do strukturního popisu způsobem uvedeným v předchozí kapitole. Strukturní popis je dále nutné přepsat do formátu čitelného některým z ATPG. V práci použité generátory používají formáty složené buď ze základních hradel (CIR, BENCH) nebo z knihovních, strukturně popsaných prvků (Verilog), které jsou opět složeny ze základních hradel. U obou formátů je tak nutno všechna FPGA primitiva popsat pomocí hradel.

Jelikož není kvůli ochraně duševního vlastnictví možné zjistit přesný popis primitiv, jsou jednotlivá primitiva přepisována podle informací, které je možné nalézt v oficiálních dokumentech výrobce [54] nebo v nejrůznějších publikacích [29], [38]. Náznaky informací o struktuře jsou k nalezení i ve zdrojových souborech knihoven UNISIM a SIMPRIM dodávaných společně s návrhovým prostředím. Jelikož jsou tyto knihovny určeny k simulaci, nemůže jim být při sestavování modelů přikládán velký význam.

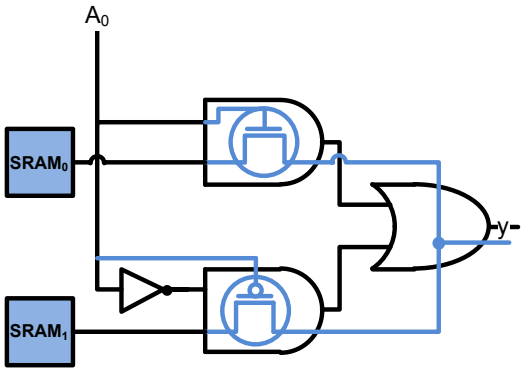
FPGA primitiva lze rozdělit na několik skupin. V první skupině se nacházejí ta primitiva, jejichž přepis je zcela zřejmý a jejichž struktura naprosto odpovídá vytvořenému modelu. Prakticky se jedná pouze o hradla, která slouží k implementaci aritmetických funkcí (AND, XOR). Druhá skupina zahrnuje prvky, jejichž strukturu lze s pomocí výše uvedených zdrojů odhadnout, ale jejich model je pouze přibližný. Do této skupiny lze zařadit struktury budičů, multiplexorů (jak infrastrukturních, tak funkčních), klopných obvodů a vyhledávací tabulky. Třetí skupina obsahuje ty prvky, jejichž vnitřní zapojení je prakticky nezjistitelné nebo jsou tak komplexní, že je

nereálné pro ně vygenerovat deterministický strukturní test. Prvky z třetí skupiny (DSP bloky, BRAM, přepínače propojovací matice apod.) přepisovány nejsou.



Obrázek 13: Strukturní model třívstupého LUT v generátorickém režimu.

Pravidla přepisu první skupiny jsou triviální. Instance prvku se nahradí příslušnou konstrukcí v syntaxi daného formátu. U prvků z druhé množiny je načtena konfigurace FPGA prvku, kterou je instance jeho modelu inicializována. Příkladem budiž přepis vyhledávací tabulky (obrázek 13). Jednotlivé transistory jsou v modelu nahrazeny hradly AND a invertory, spojení vodičů je realizováno jako hradlo OR (odkaz). LUT je tak modelován jako 16vstupý multiplexor se čtyřmi adresními vstupy. Hodnoty uložené v konfigurační paměti jsou nahrazeny konstantními signály log. 0 a log. 1. Dle podobného klíče jsou modelovány i další prvky z druhé skupiny a k nim příslušející část konfigurační paměti. V případě formátů BENCH a MILEF musí být do obvodu přidán globální rozvod logických 0 a 1. Ty jsou získány z rovnic $l_0 = x\bar{x}$ a $l_1 = x + \bar{x}$. Vstupní proměnnou x je jeden z primárních vstupů. Jazyk Verilog využití konstant podporuje, globální rozvody vytvářeny být nemusejí. Doplním, že specifikace Verilogu podporují i využití transistorových konstrukcí, kterými by šlo zapojení primitiv modelovat přesněji, použité ATPG (TetraMAX) je však interně nepodporuje kvůli nekompatibilitě s modelem poruch trvalá 0/1.



Obrázek 14: Model tranzistorového multiplexoru.

Transformované prvky jsou propojeny stejným způsobem jako originální obvod. Do struktury obvodu jsou dále přidány části obvodových struktur sloužících pro oddělení rekonfigurovatelných modulů (viz 3.7.3). Pro transformaci obvodu za účelem generování testu byl vytvořen skript *trans*.

Výsledky přepisu vybraných benchmarkových obvodů řady ISCAS85 a ISCAS89 jsou zaneseny v tabulkách 2 a 3. V prvním řádku obou tabulek je počet hradel původního obvodu. Ten byl získán ze strukturního popisu ve formátu BENCH, který byl po konverzi do VHDL pro experiment použit. Druhý řádek udává počet hradel po implementaci a přepisu obvodu. Nárůst počtu hradel po přepisu ASIC→FPGA→ASIC je v průměru 30 násobný: při vynechání extrémně malých obvodů c17 a s27 jsou přepsané obvody v průměru 21 krát větší. Pro další krok (generování testu) je významným parametrem předposlední řádek tabulky – počet konstant v obvodu. Tento parametr udává kolik vstupů hradel je připojeno na zdroj log. 0 a log. 1. Poslední řádek tabulky je ilustrativní a udává počet SLICE, které benchmarkový obvod v FPGA vyžaduje.

Tabulka 2: charakteristiky přepsaných obvodů ze sady ISCAS85

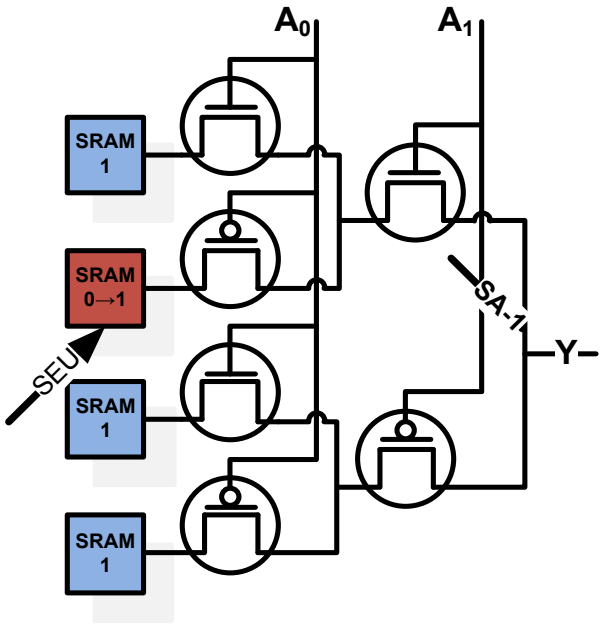
Obvod	**c1355**	**c17**	**c1908**	**c3540**	**c432**	**c499**	**c880**
ASIC hradel	**546**	**6**	**880**	**1669**	**160**	**202**	**383**
FPGA přepsaných hradel	**7487**	**447**	**8020**	**20225**	**5870**	**7480**	**9589**
Počet konstant v obvodu	**3708**	**247**	**3589**	**7999**	**2557**	**3764**	**4555**
FPGA SLICE	**45**	**2**	**58**	**181**	**43**	**45**	**60**

Tabulka 3: charakteristiky přepsaných obvodů ze sady ISCAS89

Obvod	**s1494**	**s208**	**s27**	**s298**	**s344**	**s349**	**s386**	**s832**	**s1488**
ASIC hradel	647	96	11	119	169	170	159	287	653
FPGA přepsaných hradel	14374	2586	792	3244	4418	4420	3867	7602	14606
Počet konstant v obvodu	5715	1304	424	1652	2310	2324	1736	3318	5723
FPGA SLICE	134	12	3	15	17	17	27	57	136

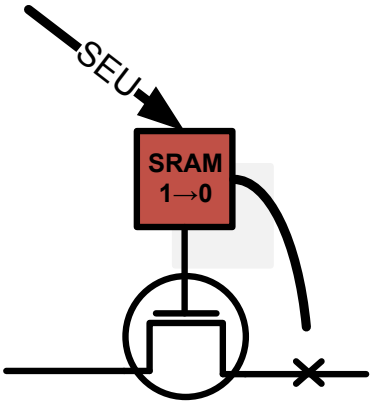
3.4 Generování testu

V práci použité ATPG (Atalanta, MILEF a TetraMax) podporují generování testů pro nejrůznější modely poruch. Společným pro všechny jmenované je model trvalá 0/1. Ačkoliv není tento model pro moderní CMOS obvody nejvhodnější, jedná se o nejrozšířenější model poruch, který podchytí celou řadu poruch jiných modelů [33]. V nejhorším případě dojde při použití modelu trvalá 0/1 ke zcitlivění cesty a alespoň se zvýší šance, že se porucha projeví.



Obrázek 15: Projev SEU jako Stuck-at 1

Jednou z tezí práce je, že pomocí modelu trvalá 0/1 lze do jisté míry modelovat i poruchy SEU. SEU jsou v přepsaném obvodu modelovány jako poruchy trvalá 0/1 na vodičích vedoucích z buněk konfigurační paměti. Je-li v buňce uložena hodnota log. 1, pak je případná SEU modelována jako trvalá 0, v případě log. 0 jako trvalá 1. Ukázka korektního a nekorektního použití modelu trvalá 0/1 u SEU je zobrazena na obrázku 15, respektive obrázku 16. Na prvním obrázku je ilustrován projev poruchy SEU ve vyhledávací tabulce jako poruchy trvalá 1 na výstupním vodiči. V druhé ilustraci způsobuje SEU na výstupu budiče poruchu, která není s modelem trvalá 0/1 kompatibilní. I v případě testování druhé poruchy pomocí deterministického testu dojde ke zcitlivění cesty – porucha se může projevit.



Obrázek 16: Projev SEU jako stuck-open

Ani nejnovější ATPG si s přepsanými obvody na výbornou nevedlo. Přepis prvků se nepříznivě projevuje na velikosti obvodu. Každý LUT se přepíše na 60 hradel, zatímco booleovská funkce, kterou realizuje, může být vyrobena podstatně levněji (viz tabulka 2 a 3 v předchozí kapitole). Vzhledem k tomu, že jsou moderní ATPG na velké obvody připraveny, není toto nejzávažnější komplikace. Problémem spočívá především v zanesení velkého počtu konstant do struktury obvodu. Na ty ATPG při budování citlivých cest často naráží a rychle vyčerpává limit návratů, který je nutné navýšit i o několik řádů. Navyšování limitu se negativně projeví na době běhu ATPG, která je několikanásobně delší, než pro obvody ASIC se srovnatelným počtem hradel. Řadu přepsaných obvodů nebyly starší ATPG ani schopny zpracovat. Řešením dlouhých běhů ATPG může být redukce redundantních prvků v přepsaném obvodu [39]. Výhodnější by mohlo být i využití ATPG založeném na principu jiném, než je zcitlivění cesty, např. ATPG založeného na SAT [47], [12].

Pro ověření vlastností testovacích vektorů je zapotřebí vygenerované vektory odsimulovat. Realistické ověření by muselo využít přesný model struktury FPGA (tj. ASIC netlist příslušného FPGA obvodu), na kterém by se vektory vygenerované pro přepsaný (redukovaný) model simulovaly. Úplný model má k dispozici pouze výrobce. Použít simulátor poruch a ověřovat testy na modelu, ze kterého byly generovány, nepřináší informaci o skutečné kvalitě testu. Na druhou stranu taková simulace může poskytnout srovnání jednotlivých testovacích sad. Porovnání vektorů vygenerovaných pro přepsaný obvod s vektory, které byly vygenerovány pro původní obvody, je zaznamenáno v tabulce 4.

Označení sloupců v tabulce 4 má následující význam. „FPGA“ jsou testy vygenerované pro obvody přepsané dle metodiky uvedené v předchozím textu. Označením „ASIC“ jsou rozuměny testy vygenerované pro původní strukturu obvodu (konvertovanou z BENCH formátu do verilogu). Zkratka ND v pátém sloupci znamená nedetekovatelné poruchy. Poslední dva sloupce udávají dobu, po kterou ATPG generovalo test včetně spuštění prostředí a zápisu výsledků. Hodnoty posledního sloupce jsou uvedeny v sekundách. Rovnice 1,2,3 definují pokrytí testu, pokrytí poruch a efektivitu ATPG tak, jak jsou uvedeny v [42].

Tabulka 4: Srovnání testovacích vektorů

Obvod	Počet vektorů		Počet poruch v obvodu		Max. pokrytí poruch [%]	Pokrytí testu [%]		Efektivita ATPG [%]		Generování [h:m:s]	
	FPGA	ASIC	Celkem	ND		FPGA	ASIC	FPGA	ASIC	FPGA	ASIC
c1355	211	123	41712	23607	56,60	90,08	69,03	95,69	86,56	03:46	0,898
c17	16	4	2476	1629	65,79	81,94	53,25	93,82	84,01	00:01	0,771
c1908	275	148	44544	22836	51,27	89,75	83,19	95,01	91,81	02:50	0,931
c3540	447	162	111730	50373	45,08	85,27	78,38	91,91	88,13	01:13	1,183
c432	206	47	32288	16981	52,59	82,52	68,65	91,71	85,14	12:10	0,938
c499	171	101	41602	24330	58,48	88,36	68,13	95,17	86,77	01:58	1,090
c880	145	37	53234	30116	56,57	86,91	75,18	94,31	89,22	00:03	0,809
s1488	283	120	81028	39056	48,2	86,43	80,91	92,97	90,11	00:03	1,047
s1494	294	122	79844	38301	47,97	85,12	79,53	92,26	89,35	00:03	0,843
s208	105	28	14456	9169	63,43	86,23	73,86	94,96	90,44	00:01	0,774
s27	20	8	4350	2942	67,63	82,60	73,44	94,37	91,40	00:01	0,763
s298	58	28	18082	10966	60,65	84,49	77,25	93,89	91,05	00:01	0,773
s344	49	18	24502	15276	62,35	82,10	72,33	93,26	89,58	00:01	0,778
s349	47	19	24534	15488	63,13	82,73	73,99	93,63	90,41	00:01	0,779
s386	134	68	21566	11779	54,62	84,59	76,35	93,01	89,27	00:01	0,778
s832	243	113	42328	23208	54,83	84,56	77,55	93,03	89,86	00:02	0,821

$$Pokrytí\ testu = \frac{detekované\ poruchy}{detekovatelné\ poruchy} \quad (1)$$

$$Pokrytí\ poruch = \frac{detekované\ poruchy}{všechny\ poruchy} \quad (2)$$

$$Efektivita\ ATPG = \frac{řešitelné\ poruchy}{všechny\ poruchy} \quad (3)$$

Všechny testy byly generovány v ATPG TetraMax s použitím modelu trvalá 0/1 na počítači s Intel Xeon L5640 @ 2.27-2.8GHz s 32GB RAM. ATPG bylo nastaveno na generování testu se 100 % pokrytím, vysokou prioritou slučování vektorů a s limitem návratů 2000000. Porovnání ASIC a FPGA testovacích sad nemá v principu

za cíl srovnávat jejich efektivitu – je zřejmé, že test vytvořený pro naprosto odlišnou strukturu obvodu nemůže v porovnání obstát. Podstatná je spíše doba generování, která je v případě přepsaného obvodu v průměru o téměř dva řády vyšší. Nejpravděpodobnějšími důvody jsou navýšení prvků v obvodu a značné množství konstant na jejich vstupech (viz tabulka 2 a 3).

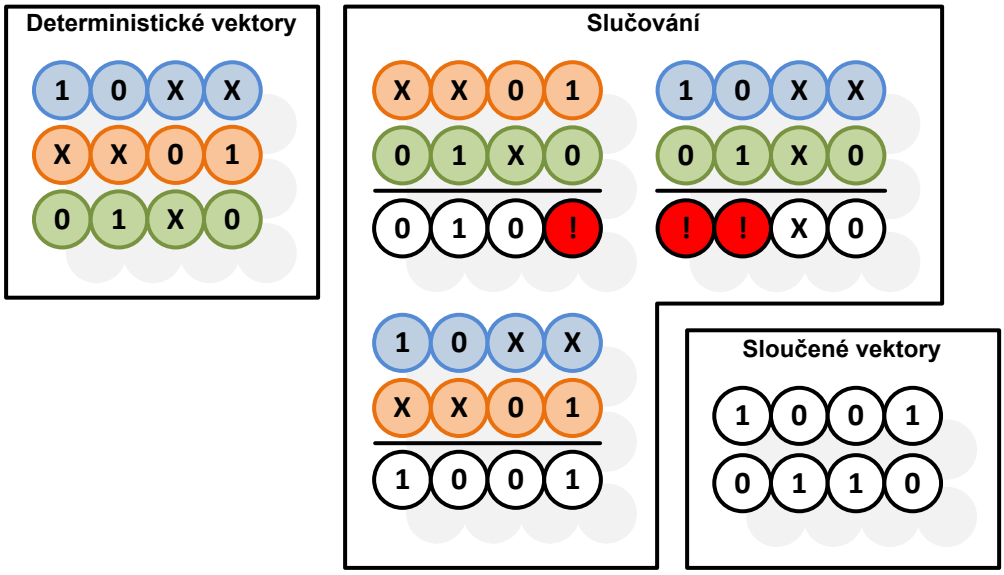
3.5 Komprese testovacích dat

Cílem dělení obvodu je rozdělit ho na takové části, které umožní implementaci TPG i ORA v místech sousedících s testovanou částí obvodu bez nutnosti okolním blokům přidělovat větší prostor nebo vytvářet mechanismy, které umožní doplnění testovacích dat do testeru. Podmínku lze snadno splnit, je-li TPG implementován jako generátor pseudonáhodných vektorů. LFSR (ale i celulární automat) jsou v FPGA snadno realizovatelné a je vyloučeno, že by vyčerpaly zdroje sousedního modulu, poněvadž je počet klopných obvodů v „sousedovi“ vždy větší nebo roven počtu primárních vstupů a výstupů testovaného modulu a ke každému klopnému obvodu přísluší jedno hradlo XOR (LUT), které je pro implementaci generátoru potřebné.

Velikost okolních modulů je omezující pro deterministický generátor. Ten lze nejjednodušším způsobem navrhnout jako binární čítač adresy připojený na paměti ROM tvořené LUT, do níž jsou uloženy testovací vektory. Kapacita ROM v TPG nemusí být dostatečná k uložení všech vektorů a v některých případech je ji nutné navýšit. Toho lze docílit zvětšením oblasti TPG, zajištěním přístupu do systémové paměti nebo rozdělením testu na několik fází a rozdělením vektorů do jednotlivých modulů TPG – navýšením počtu rekonfigurací obvodu. Abychom přístup do paměti nebo počet rekonfigurací omezili, je možné testovací vektory komprimovat.

Pro snížení objemu testovacích dat byl použit kompresní algoritmus COMPAS, který testovací data komprimuje pomocí překládání vektorů. Princip překládání vektorů je založen na empirickém zjištění, že většina testovacích vektorů bývá z velké většiny vyplněna nedefinovanými hodnotami. Při budování citlivé cesty není vždy nutné ošetřit všechny primární vstupy obvodu. Běžné kompresní techniky tohoto faktu využívají a kompakci vektorů provádějí překrýváním vektorů. Například interní komprese v ATPG TetraMax využívá metody překrývání vektorů (pattern merging) [42], která hledá takové vyplnění nedefinovaných hodnot v testovacích vektorech, aby výsledný, úplně definovaný vektor detekoval co možná největší počet poruch. Pattern merging se snaží nalézt vektory, které mají na shodné pozici buď shodné hodnoty, nebo nedefinované

hodnoty a ty logicky sčítá. Generátor se zároveň snaží vytvořit takových vektorů minimum. Příklad slučování je na obrázku 17.



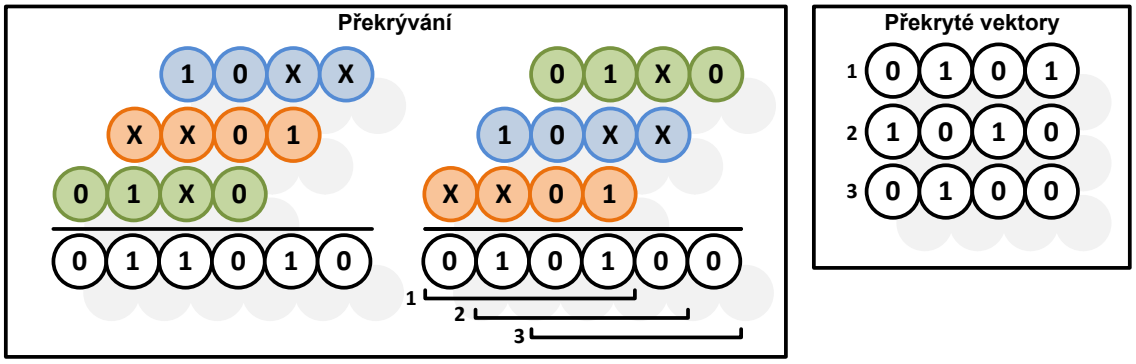
Obrázek 17: Pattern Merging

Na rozdíl od techniky pattern merging, která vektory slučuje pouze v prostoru se COMPAS snaží vektory sloučit i v čase. COMPAS ke své činnosti vyžaduje vstupní data – seznam poruch v obvodu a k nim příslušejících testovacích vektorů, které jsou, pokud možno, zaplněny nespecifikovanými hodnotami. Nejprve se, stejně jako TetraMax, snaží jednotlivé vektory slučovat. Pokud ve slučování neuspěje, posouvá vzájemně vektory (viz obrázek 18) tak, aby mohl překrýt alespoň jejich části. Při každém kroku překrytí je původní vektor posunut v pomyslném skenovacím řetězci o jednu pozici. Nově vzniklý vektor (s nespecifikovanou hodnotou na svém konci) je možné sloučit s dalším vektorem ze seznamu. Když algoritmus s hledáním vhodného vektoru uspěje, překryté vektory a detekované poruchy odstraňuje ze seznamu.

Může dojít k situaci, že nebude možné vektory nijak překrýt. Stane-li se tak, pak COMPAS generuje nové vektory, aby v co nejméně krocích dospěl k možnosti překrytí dalším vektorem. Nové vektory mají charakteristiky podobné s pseudonáhodnými vektory; sekvence se stává pomyslnou kombinací deterministického a náhodného testu, která je pro testování velice výhodná.

Počet vektorů, které se překrýváním vytvoří, je vyšší, než v případě metody Pattern Merging, ale vektory vygenerované COMPASem mají zajímavou vlastnost – každý vektor se od předchozího liší pouze v jednom bitu. *N* testovacích vektorů o délce *m* je ve vhodném dekompresním obvodu možné rekonstruovat z binární sekvence o

délce $m+N$ tak, že každý vektor bude vytvořen z předchozího vektoru posunutého o jeden bit a na konci doplněného nový bit ze zkomprimované sekvence. K dekompresi je zapotřebí posuvného registru o délce m, případně nevyužitého skenovacího řetězce se zpětnou vazbou o stejné délce (viz architektura DyRESPIN v 2.6.3).



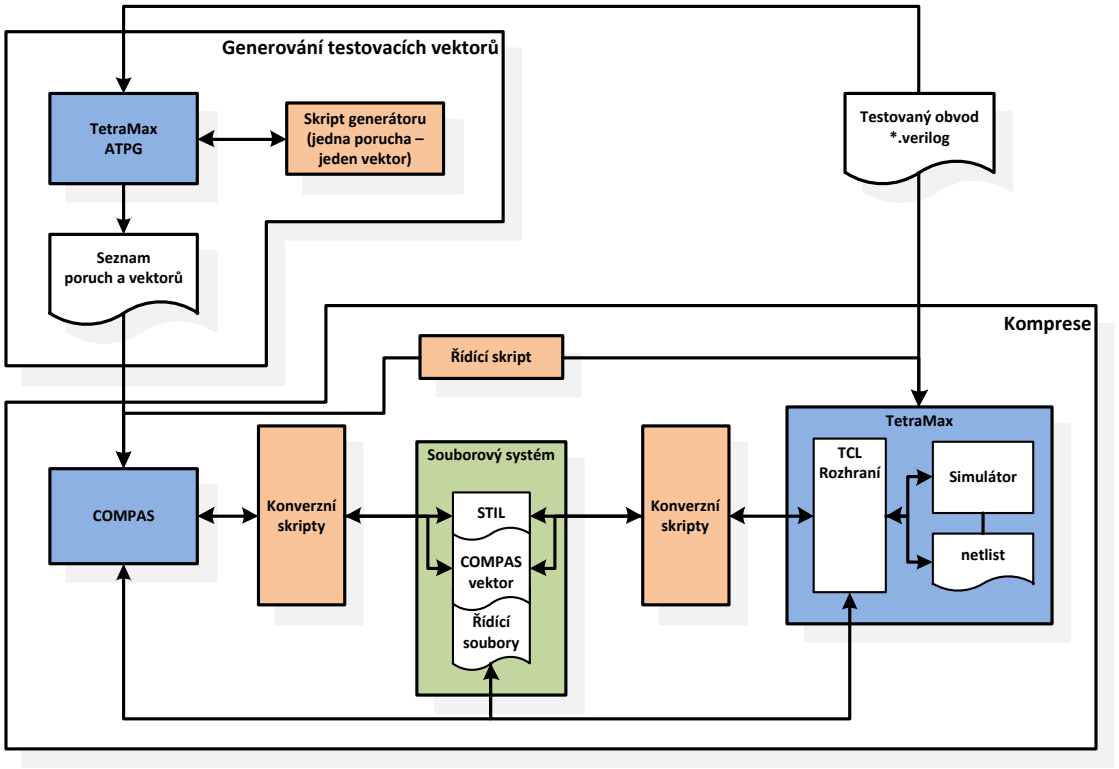
Obrázek 18: COMPAS ilustrace možných překrytí a výsledných vektorů

COMPAS může pracovat ve dvou režimech – se simulátorem poruch a bez něj. Je-li simulátor k dispozici, pak je každý nově vytvořený vektor simulován a poruchy, které detekuje, jsou odstraněny ze seznamu společně s testovacími vektory, které jim příslušely. Použití simulátoru významně ovlivňuje délku zkomprimované sekvence. Interně obsahuje COMPAS simulátory FSIM a HOPE, které slouží pro práci se soubory formátu BENCH, potažmo pro spolupráci s ATPG Atalanta. Jelikož Atalanta řadu obvodů nebyla schopna zpracovat, bylo nutno vytvořit rozhraní mezi COMPAS a interním simulátorem TetraMax.

Spolupráce COMPAS s ATPG probíhá následovně. Pro testovaný obvod je nejprve vygenerován seznam všech poruch (blok Generování testovacích vektorů na obrázku 19). Seznam je automaticky zpracován a dle něj je vytvořen skript generující vektory pro interpret TCL jazyka, který je integrován v ATPG. Skript je řízen výsledky běhu ATPG. To je v počátku nastaveno na generování vektorů s nespecifikovanými hodnotami s nízkým limitem návratů. Nedaří-li se generátoru vektor sestavit, limit návratů skript navyšuje tak dlouho, dokud není překročen uživatelem zadaný limit. Poté se opouští od vektorů s nespecifikovanými bity a skript nastaví ATPG pro generování standardních vektorů. Představu o výpočetní náročnosti si lze vytvořit porovnáním doby generování standardního a COMPAS testu (bez komprese), která je vynesena v tabulce 6.

Samotná komprese je realizována ve fázi druhé. COMPAS postupně předkládá konverzním skriptům vektory (kandidáty na překrytí), které jsou konvertovány,

v TetraMax simulovány a COMPASU je (opět přes konverzní skripty) navrácen seznam detekovaných poruch. Doba trvání komprese je oproti době generování testu zanedbatelná.



Obrázek 19: Blokové schéma rozhraní COMPAS – TetraMax

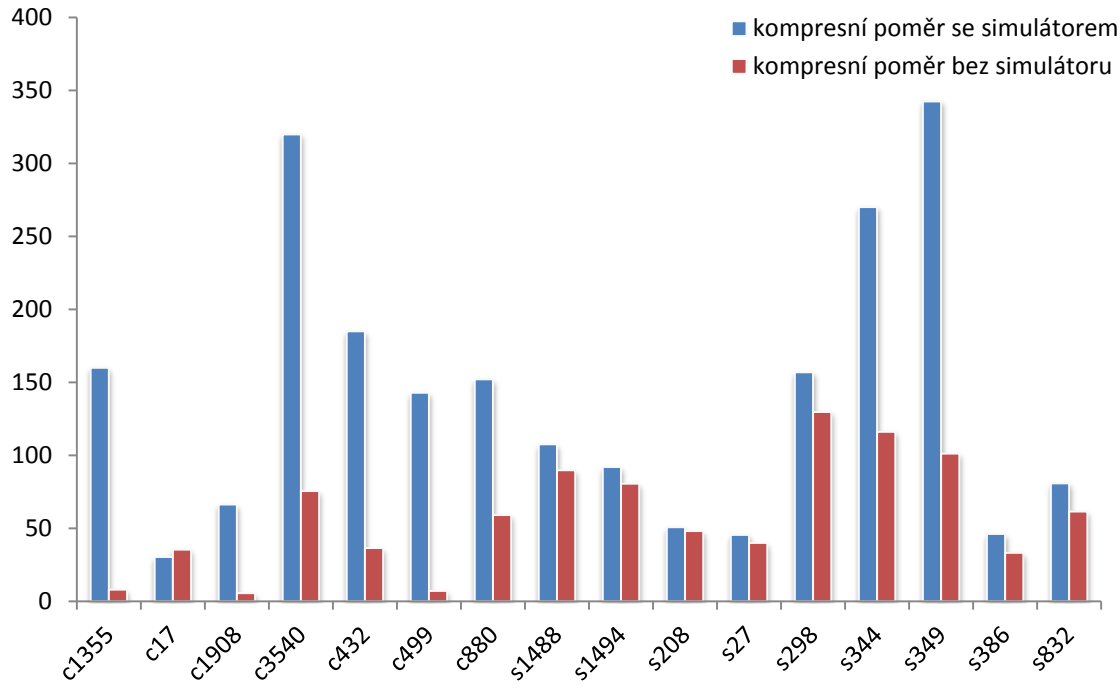
V tabulce 5 jsou zaneseny délky zkomprimovaných sekvencí pro některé obvody ISCAS85 a ISCAS89. V druhém, třetím a čtvrtém sloupci tabulky jsou zaneseny hodnoty původních dat a dat zkomprimovaných bez a s použitím simulace. Objem původních dat je odvozen od počtu vygenerovaných vektorů a počtu primárních vstupů příslušného obvodu.

Tabulka 5: Srovnání objemů testovacích dat

Obvod	Počet vektorů	Objem testovacích dat			Kompresní poměr	
		Původní	Bez sim.	Se sim.	Bez sim.	Se sim.
c1355	3563	146083	18020	913	8,1	**160,0**
c17	128	640	18	21	**35,5**	30,4
c1908	3210	105930	18893	1596	5,6	**66,3**
c3540	5542	277100	3667	866	75,5	**319,9**
c432	1352	48672	1335	263	36,4	**185,0**
c499	2956	121196	17045	848	7,1	**142,9**
c880	2741	164460	2779	1081	59,1	**152,1**

Obvod	Počet vektorů	Objem testovacích dat			Kompresní poměr	
		Původní	Bez sim.	Se sim.	Bez sim.	Se sim.
s1488	6808	95312	1060	886	89,9	**107,5**
s1494	6214	86996	1080	945	80,5	**92,0**
s208	828	15732	327	309	48,1	**50,9**
s27	189	1323	33	29	40,0	**45,6**
s298	1229	20893	161	133	129,7	**157,0**
s344	1317	31608	272	117	116,2	**270,1**
s349	1456	34944	345	102	101,2	**342,5**
s386	1718	22334	671	484	33,2	**46,1**
s832	3453	79419	1292	981	61,4	**80,9**

Ve sloupci „kompresní poměr“ jsou vyneseny kompresní poměry při a bez použití simulátoru. Až na případ extrémně malého obvodu c17 je patrné, že využití simulátoru významně redukuje délku zkomprimované sekvence. Na obrázku 20 je graficky znázorněno porovnání kompresních poměrů z posledních dvou sloupců tabulky 5.



Obrázek 20: Porovnání kompresních poměrů při a bez použití simulace

V tabulce 6 je zaneseno srovnání standardní (sloupce označené TMAX), nekomprimované a zkomprimované (označení COMPAS) sekvence testovacích dat.

Porovnáme-li dobu generování, pak je průměrná hodnota generování nekomprimované sekvence téměř 20× vyšší, než u sekvence standardní. Celkové navýšení doby generování oproti době generování testu původních obvodů (viz tabulka 4) je témě o tři řády vyšší. Určitý podíl na navýšení má bezpochyby režie řízení ATPG, ovšem tento podíl bude zanedbatelný. Největší zpomalení je způsobeno vynecháním fáze generování pseudonáhodných vektorů, která při běžném generování rychle odhalí velký počet poruch. Obě sady testů (TMAX a COMPAS) jsou srovnatelné co do pokrytí testu a efektivity ATPG.

Poněkud zarážející se mohou jevit nízké hodnoty pokrytí testu a efektivity ATPG u nekomprimovaného testu, ačkoliv je ten generován pro každou poruchu obvodu. Problém spočívá v redukci logiky, která je při vytváření modelu obvodu v ATPG prováděna. Poruchy, které jsou při generování označeny jako testovatelné, jsou při následné simulaci hodnoceny jako netestovatelné.

Tabulka 6: Tabulka pokrytí testů

	Pokrytí testu [%]			**Efektivita ATPG [%]**			**Čas generování [h:m:s]**	
Obvody	**TMAX**	**Nekomp.**	**COMPAS**	**TMAX**	**Nekomp.**	**COMPAS**	**TMAX**	**Nekomp.**
c1355	90.08	87.37	90.08	95.69	94.52	95.69	0:03:46	1:03:21
c17	81.94	81.94	81.94	93.82	93.82	93.82	0:00:01	0:00:13
c1908	89.75	87.16	89.77	91.81	93.74	95.02	0:02:50	0:18:25
c3540	85.27	64.37	84.33	91.91	80.43	91.39	0:01:13	0:47:53
c432	82.52	41.50	81.21	91.71	72.27	91.09	0:12:10	0:56:37
c499	88.36	86.70	88.36	95.17	94.48	95.17	0:01:58	0:01:44
c880	86.91	53.43	86.40	94.31	79.77	94.10	0:00:03	0:01:35
s1488	86.43	82.99	86.19	92.97	91.19	92.85	0:00:03	0:02:25
s1494	85.12	81.16	84.80	92.26	90.20	92.09	0:00:03	0:02:41
s208	86.23	78.48	83.62	94.96	92.13	94.01	0:00:01	0:00:11
s27	82.60	72.66	81.53	94.37	91.15	94.02	0:00:01	0:00:03
s298	84.49	82.59	84.49	93.89	93.15	93.89	0:00:01	0:00:10
s344	82.10	78.84	82.10	93.26	92.03	93.26	0:00:01	0:00:18
s349	82.73	80.54	82.73	93.63	92.83	93.63	0:00:01	0:00:14
s386	84.59	83.74	84.53	93.01	92.62	92.98	0:00:01	0:00:19
s832	84.56	81.73	84.36	93.03	91.75	92.93	0:00:02	0:00:37

3.6 Generátory a analyzátory

S pomocí testovacích dat získaných v předchozí kapitole a rozdělených a upravených obvodů (kapitola 3.2) je možné vytvořit pro každý obvod unikátní generátor testovacích vektorů a analyzátor odezev. Představená metoda explicitně nespecifikuje, jakým způsobem mají být TPG a ORA implementovány. Pro účely diagnostiky je kupříkladu vhodnější mít k dispozici každou odezvu z testovaného obvodu zvlášť, nežli až po testu analyzovat kompaktní odezvu; ORA implementovaný jako prostý skenovací řetězec v takovém případě vyhoví lépe, než složitější MISR. Obdobně platí, že generátory mohou být upraveny tak, aby splňovaly rozličné požadavky, Například modelu trvalá 0/1 poruch lépe vyhoví dekompresor COMPAS, pro model se hodí generátor s deterministickými vektory uloženými v paměti ROM.

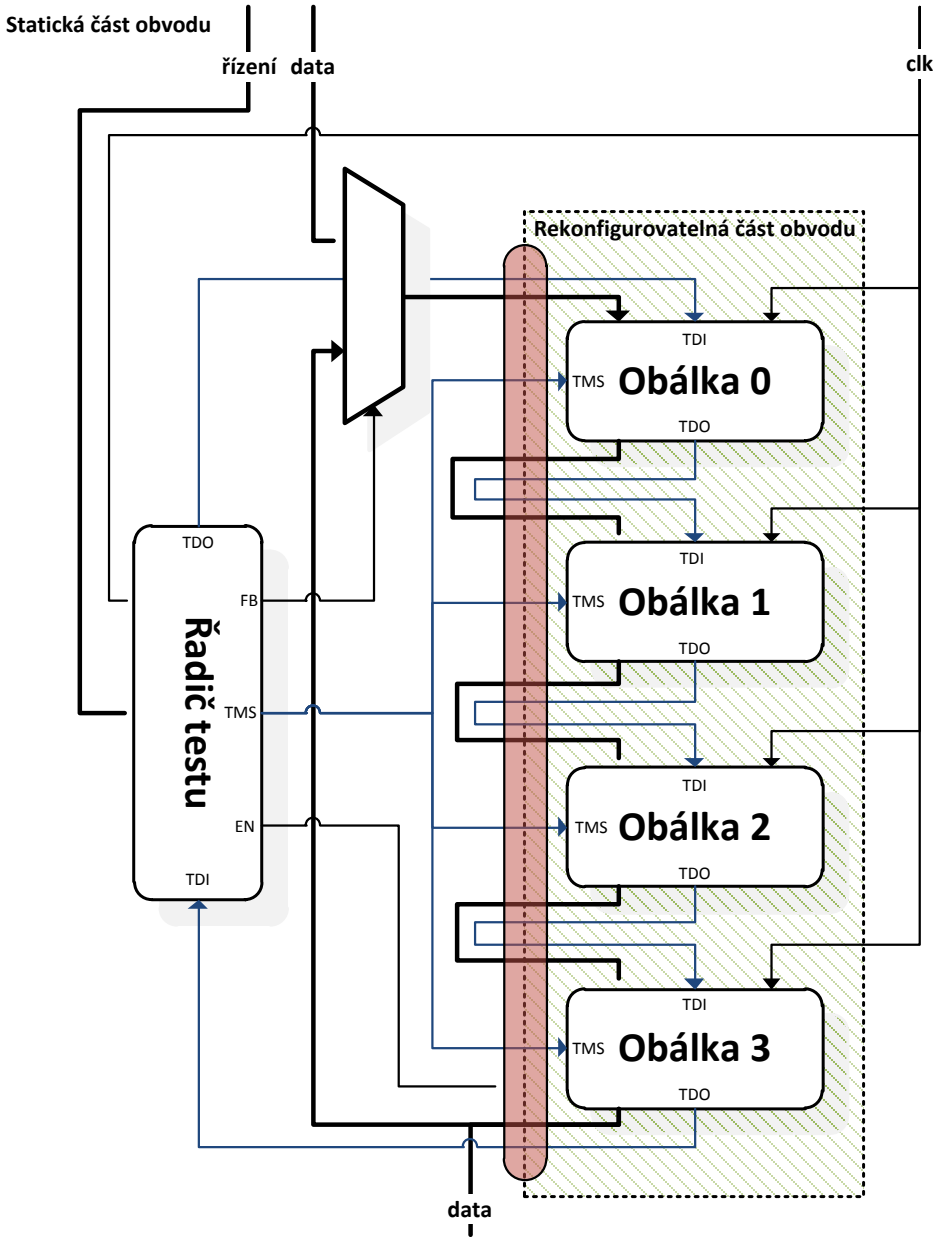
Dynamicky rekonfigurovatelné programovatelné obvody díky své flexibilitě poskytují oproti obvodům s pevnou strukturou při takto volném zadání nespornou výhodu. Různé varianty generátorů je možné ukládat do systémové paměti a dynamicky je dle potřeb testu měnit. V průběhu životního cyklu výrobku je dokonce možné navrhnout jejich nové varianty a bez větších potíží je do systému včlenit. Při návrhu je nutno dodržet pouze pravidlo, že obě součásti vnořeného testeru musí být možno vložit do obvodu jako rekonfigurovatelný modul, tj. musí splňovat základní parametry, jako jsou velikost, časování a pinová kompatibilita s původním obvodem a musí je být možno definovaným způsobem ovládat z nadřazeného řadiče testu. Všechny ostatní detaily jsou zcela na návrháři.

3.6.1 Obálka modulu

Rozdělené obvody jsou doplněny o obálku, která unifikuje přístup k jednotlivým částem obvodu. Rozhraní každé obálky je tvořeno původními primárními vstupy a výstupy a signály řízení testu. Řídící signály (na obrázku 21 jsou znázorněny modře) jsou tři a obdobně jako u standardu JTAG jsou pojmenovány názvy TMS (Test Mode Select), TDI (Test Data In) a TDO (Test Data Out).

Počet řídících signálů nebyl zvolen náhodně. Prvním důvodem je zachování zažitých konvencí, druhým důvodem je režie, která vzniká na každém vstupu nebo výstupu do rekonfigurovatelného obvodu (červená oblast na obrázku 21). Větší počet řídících signálů by přinesl rychlejší průběh testu, protože by bylo možno řídit test paralelně, ale přinesl by neúměrně velkou režii. Menší počet řídicích vstupů by režii

zdánlivě snížil, ale testování by se stalo pomalým a velikost řadiče uvnitř modulů by úsporu prostředků dozajista vyrovnala.

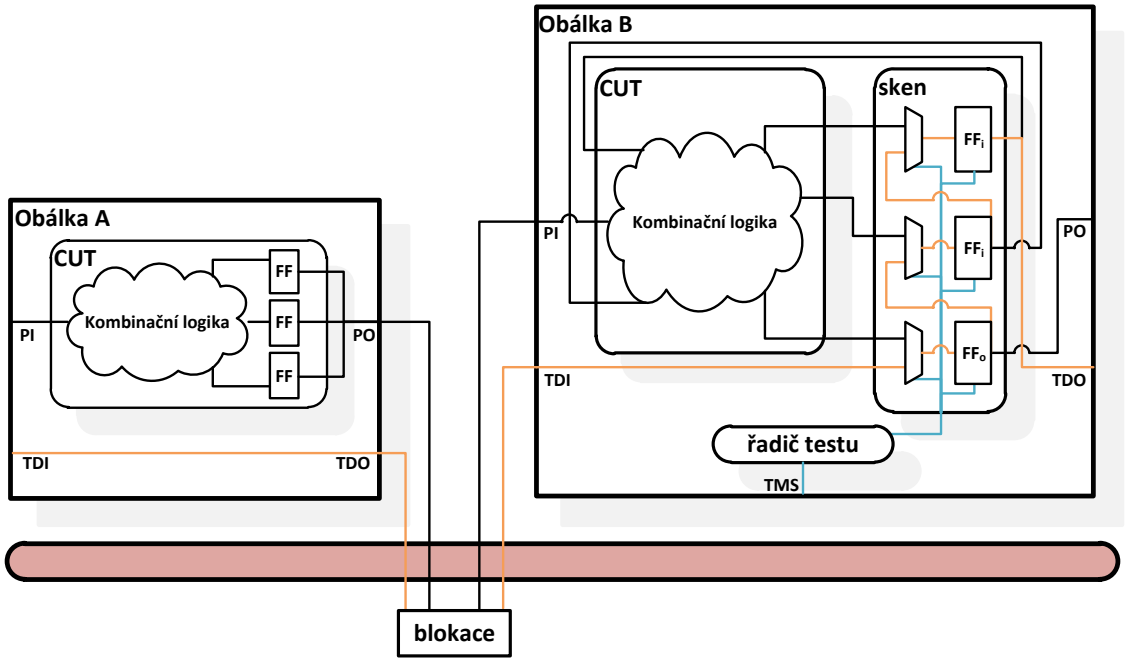


Obrázek 21: Propojení obálek rekonfigurovatelných modulů

Vstup TDI představuje sériové datové rozhraní do obvodu, výstup TDO je jeho výstupním ekvivalentem. Řídící vstup TMS řídí stavové automaty uvnitř decentralizovaného testeru. Signály testovacího přístupu jsou synchronizovány na náběžnou hranu hodinového signálu, který je společný pro všechny části testovaného obvodu a může být řízený externím prvkem.

Na obrázku 22 je zobrazeno principiální schéma propojení obálky obvodu ve dvou variantách rekonfigurovatelných modulů ve funkčním režimu. V případě obálky

A není testovaný modul opatřen skenovacím řetězcem a tak jsou vstupy a výstupy obálky přímo napojeny na příslušné vstupy a výstupy modulu. Takové propojení nepřidává žádnou logiku a dochází při něm pouze k přejmenování signálů. Signály TDO a TDI jsou vzájemně propojeny tak, aby bylo možno testovat poslední modul. Druhá část ilustrace (obálka B) demonstruje situaci, ve které je testovaný modul opatřen skenovacím řetězcem. V takovém případě musí být obálka doplněna o jednoduchý stavový automat, který bude řídit snímání a posuv odezev ve skenovacím řetězci. Jak řadič, tak skenovací řetězec by měly být před implementací hierarchicky odděleny od samotného funkčního obvodu (např. tak jak jsou odděleny části CUT a „sken“ na obrázku 22 uvnitř obálky B), jinak půjde jen obtížně obvod dělit; syntéza může sloučit některý logický blok testovaného obvodu s multiplexorem posuvného registru. Doplňme, že zkratky PI a PO odpovídají primárním vstupům, respektive výstupům a indexy *i* a *o* u obálky B odpovídají interním a výstupním klopným obvodům.



Obrázek 22: Různé implementace obálek testovaného modulu

3.6.2 Generátor testovacích vektorů

Generátorem testovacích vektorů je myšlen obvod, který dodává stimuly testovanému obvodu. Generátor ovládá vstupy do testovaného modulu vyjma vstupu TMS. Generátor se nachází „vlevo“ od testovaného modulu, pouze v případě testu

prvního modulu rozděleného obvodu se nachází vpravo od něj (viz obrázek 10). V práci byly navrženy, implementovány a nasazeny tři verze generátoru testovacích vektorů:

- Generátor pseudonáhodných vzorků s využitím posuvného registru s lineární zpětnou vazbou
- Deterministický generátor s řadičem a pamětí ROM pro
  - Vzorky komprimované algoritmem COMPAS
  - Nekomprimované vzorky

Generátor založený na posuvném registru s lineární zpětnou vazbou je vytvářen skriptem *tpg_lfsr*. Argumenty programu jsou VHDL popisy rozděleného obvodu. Program vyžaduje alespoň tři soubory: soubor s testovaným modulem, jeho předchůdce a následníka. Pro každý vstupní soubor je generován unikátní generátor.

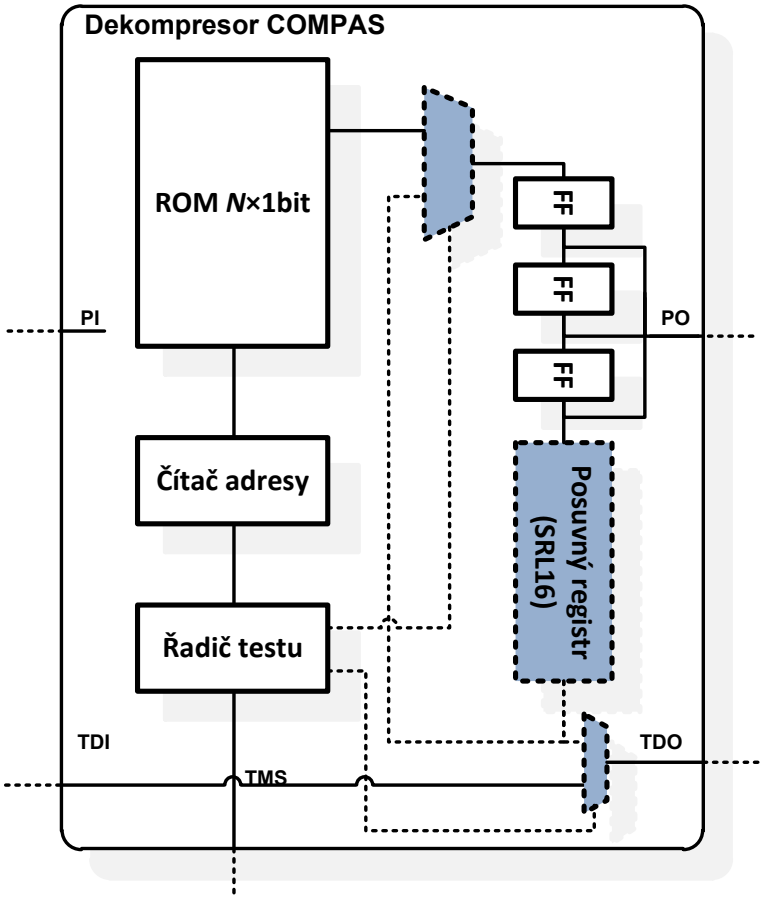
Tpg_lfsr analyzuje počet primárních vstupů modulů a dle něj vybírá z tabelovaných hodnot primitivní polynom, jehož binární reprezentaci použije k zavedení zpětných vazeb v generické šabloně VHDL souboru, kterou v obálce nahradí funkční popis jádra. Společně se zpětnovazebním posuvným registrem je do obvodu vložen i distribuovaný řadič testu.

Primitivní polynomy byly vybírány z [10] a mají tu zajímavou vlastnost, že s jejich použitím se vytvoří LFSR s nejmenším počtem zpětných vazeb. Výhodou je úspora hardwarových prostředků a vysoká frekvence, na které běží. Nevýhodou je více uniformní sekvence, kterou generátory založené na takových polynomech produkují.

Vytváření obvodu z tabelovaných polynomů je limitováno maximálním řádem polynomu (konkrétně řádem 168). Pro vyšší řády lze buď doplnit tabulku polynomů (např. z [56]) nebo využít jiného (i neprimitivního) polynomu, např. $x^n + x + 1$. Pro účely práce a velikosti rekonfigurovatelných oblastí byl maximální řád primitivních polynomů naprosto dostačující.

Druhým typem generátoru je deterministický generátor implementovaný jako dekompresor sekvence COMPAS (viz 3.5). Dekompresor se skládá ze čtyř hlavních komponent – generické paměti ROM, čítače adres, dekompresního posuvného registru a stejně jako u pseudonáhodného generátoru z řadiče testu. Paměť ROM je implementována pomocí vyhledávacích tabulek a je v konfiguraci 1 bit × *N*, kde N je

počet bitů sekvence. Obsah paměti je inicializován přímo ve VHDL kódu z *teststream* souboru, který byl vygenerován programem COMPAS.

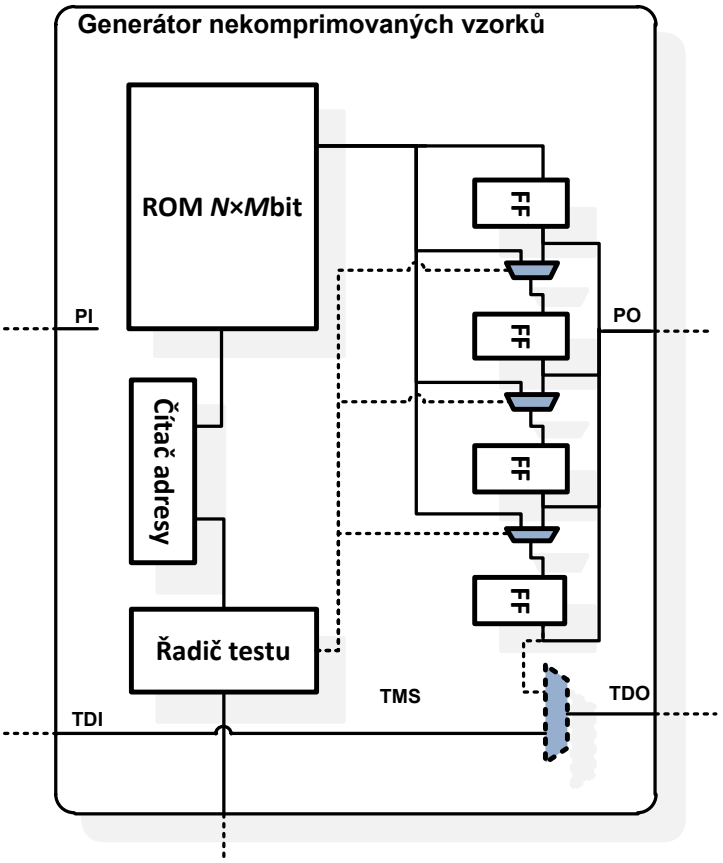


Obrázek 23: Dekompresor COMPAS

O vytváření dekompresoru se stará skript *tpg_compas*, který vyžaduje alespoň tři VHDL popisy testovaných modulů, stejně jako skript *tpg_lfsr* popsaný v předchozím textu.

Tpg_compas pracuje s generickou VHDL šablonou, která byla navržena pouze pro obvody, které neobsahují zpětné vazby a není do nich třeba vkládat skenovací řetězce. Dekompresor je generován bez zpětnovazebního multiplexoru, dodatečného posuvného registru a bypass multiplexoru (všechny jsou znázorněny na obrázku 23 modře). Obsahoval-li by testovaný modul skenovací řetězce, bylo by nutno „modré“ prvky do generické šablony doplnit, což je poměrně nenáročný úkol; zpětnovazebný a bypass multiplexor jsou triviální VHDL konstrukty, stejně jako instance posuvného registru o pevné délce, který je tvořený vyhledávacími tabulkami v režimu posuvných registrů.

Podobně jako dekompresor COMPAS je řešen i generátor nekomprimovaných deterministických testovacích vektorů. Obsahuje stejný řadič testu, ale čítač adresy, registr a paměť uchovávající testovací vzorky jsou odlišné. Paměť je v konfiguraci *M*bit ×*N*, kde *N* je počet testovacích vektorů a *M* je počet primárních vstupů testovaného modulu (případně může být *M* větší o počet buněk skenovacího řetězce testovaného souseda). Paměť vzorků je tvořena výhradně z vyhledávacích tabulek.



Obrázek 24: Generátor nekomprimovaných vzorků

V publikaci [5] bylo uvažováno o uchovávání testovacích dat v posuvných registrech. Od tohoto konceptu bylo opuštěno kvůli polovičnímu počtu vyhledávacích tabulek schopných SRL módu oproti tabulkám v režimu paměti ROM. Složitější konstrukce řadiče (přidání čítače adresy) je kompenzována navýšením paměťové kapacity generátoru.

3.6.3 Analyzátory odezev

Analyzátor odezev je připojen na výstupy testovaného modulu jako „pravý“ soused (viz obrázek 10). Obdobně jako v případě generátoru i zde existuje výjimka: při

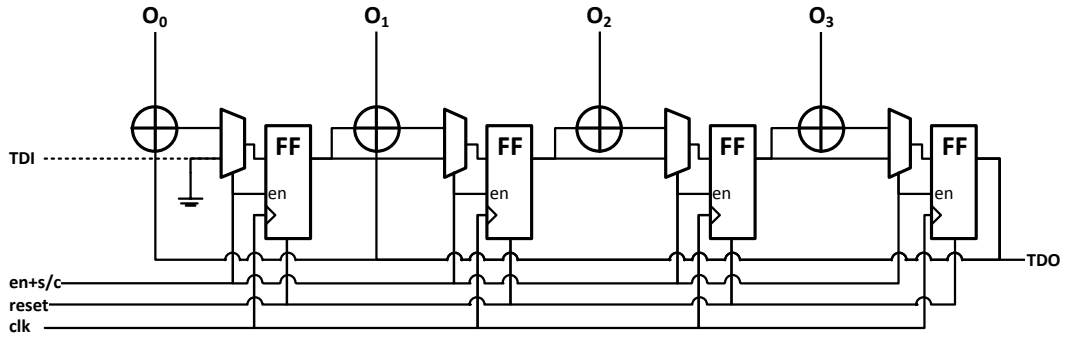
testování posledního modulu se analyzátor nachází nalevo od testovaného modulu. V práci byly navrženy a nasazeny dva typy analyzátoru odezev:

- Posuvný registr s paralelní předvolbou
- Vícevstupový příznakový registr (MISR)

První obvod není analyzátorem v pravém slova smyslu, poněvadž neprovádí příznakovou analýzu, ale pouze snímá a vysouvá sejmuté odezvy. Vzhledem k pomalému sériovému přístupu není vhodný pro reálné nasazení, vyjma ladících účelů, kvůli kterým byl vytvořen.

Druhý ORA je implementován jako MISR – je založen na výpočtu zbytku po dělení dvou polynomů – odezvy obvodu a charakteristického polynomu LFSR, na kterém je MISR vybudován. Zbytek po dělení je použit jako příznak testu.

Přibližné schéma MISR je zobrazeno na obrázku 25. Skládá se z posuvného registru s lineární zpětnou vazbou, do kterého jsou paralelně přes hradla XOR přivedeny výstupy (O_0až O_3 na obrázku 25) testovaného modulu. Implementovaný MISR pracuje ve dvou režimech: v režimu příznakové analýzy a režimu vysouvání vzorku. Obvod je řízen hodinovým signálem, signály reset, clock_enable a shift. V zájmu zachování přehlednosti jsou signály shift a clock_enable na obrázku 25 sloučeny v jeden vodič.



Obrázek 25: Přibližné schéma MISR

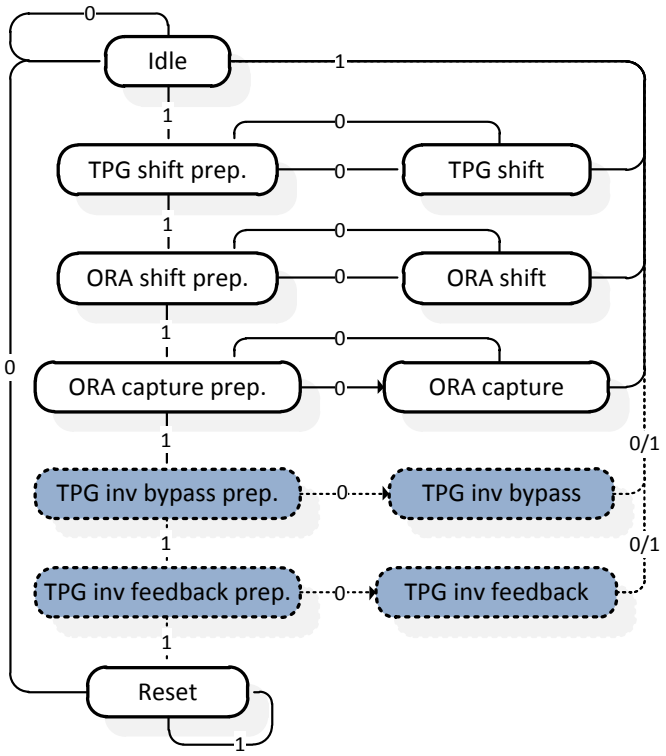
Obdobně jako generátory jsou oba analyzátory vytvářeny skripty s podobným rozhraním. Skript generující posuvný registr se nazývá *ORA_scan* a skript generující

ORA s vícevstupovým příznakovým registrem se jmenuje *ORA_misr*. Skript *ORA_misr* využívá tabelovaných primitivních polynomů, stejných jako v případě *TPG_lfsr*.

3.6.4 Distribuovaný řadič testu

Řadič testu je shodný jak pro generátory testovacích vektorů, tak pro analyzátory odezev. Ačkoliv nebyl řadič použit ve funkčním jádře jako řadič skenovacího řetězce (obvody se skenovacím řetězcem nebyly v experimentech nasazeny) je navržen tak, aby tuto úlohu mohl plnit.

Vstupy řadiče jsou hodinový signál a řídící signál testu (TMS) synchronizovaný na náběžnou hranu hodinového signálu. Výstupy řadiče jsou signály *TPG_shift*, *ORA_shift*, *ORA_capture* a *chain_reset*. Výstupy jsou řízeny stavovým automatem typu „Moore“ zobrazeným na obrázku 26.



Obrázek 26: Stavový automat generátoru testovacích vektorů a analyzátoru odezev

Výstup *TPG_shift* ovládá adresní čítač nebo posuvný registr generátoru vektorů. Výstup řídí generování nového testovacího vektoru – v analyzátoru zapojen není. *TPG_shift* je přiřazena hodnota log. 1 pouze ve stavu TPG shift, který lze vyvolat přivedením sekvence 10X na vstup TMS.

Výstupy *ORA_shift* a *ORA_capture* jsou zapojeny pouze v modulu analyzátoru odezev a ovládají „ukládání“ nového vektoru do příznakového registru v případě

ORA_capture a vysunutí odezvy z registru do nadřazeného řadiče testu v případě výstupu *ORA_shift*. Výstupy jsou zapojeny v modulu ORA, v modulu TPG zapojeny nejsou. ORA_capture i ORA_shift mohou být využity ve funkčním modulu, pokud je opatřen skenovacím řetězcem. Vyvolání posuvu je provedeno přivedením sekvence 110X, sejmutí odezvy pak s pomocí sekvence 1110X.

Výstup *chain_reset* je společný pro generátor i analyzátor nastavuje počáteční hodnoty všech komponent – např. nastavení adresy na 0 v čítači adresy TPG, nebo inicializace LFSR registru v případě ORA. Reset je vyvolán sekvencí jedniček, která je úměrná počtu stavů automatu. Automat se vrací do výchozího stavu po přivedení log. 0 po této sekvenci.

Automat je navržen tak, aby bylo možno výše popsané povely provádět opakovaně, nebo sestavovat zajímavé uživatelské sekvence. Příkladem opakování může být vysouvání odezvy, kterou lze vyvolat sekvencí $11\{00\}_n1$, kde n je délka příznakového registru. Obdobně lze i sestavit sekvenci příkazů „vygeneruj vektor a ulož odezvu“ a to jako sekvenci 100110X.

Pro testování obvodů se skenovacím řetězcem by bylo nutno doplnit řadič o dva výstupy (zapojené na T klopné obvody), ze kterých by byl řízen zpětnovazebný multiplexor a bypass multiplexor (např. obrázek 23). Doplněné stavy jsou na obrázku 26 označeny modře.

3.7 Implementační detaily

Představená metoda je z velké části založená na schopnosti hradlových polí dynamicky měnit obsah konfigurační paměti. Částečná dynamická rekonfigurace nefiguruje v práci pouze jako bezrozměrný prostředek k dosažení výměny modulů, ale tvrdě limituje možnosti nasazení metody v praxi. Rekonfigurace významnou měrou ovlivňuje nejen způsob implementace návrhu do FPGA, ale i kvalitu vzniklého obvodu, především spotřebu zdrojů a výslednou rychlost. V této kapitole bude čtenář stručně seznámen s problematikou návrhu rekonfigurovatelných obvodů.

3.7.1 Konfigurační paměť podrobně

Konfigurační paměť v obvodech Virtex-4 je organizována po blocích rovnoměrně rozprostřených po celém obvodu. Nejmenší adresovatelnou jednotkou paměti je tzv. rámec - frame [53]. Existuje několik typů rámců: typ 0 konfigurující CLB, DSP, IOB, typ 1 propojení BRAM, typ 2 obsah BRAM atd. Všechny rámce mají shodnou velikostí 1312 bitů (41 32bitových slov). Polohu rámce v obvodu určuje adresa rámce, která se skládá ze dvou údajů: pozice rámce v obvodu a typu rámce. Systém fyzické adresace rámce je následující.

FPGA obvody rodiny Virtex4 jsou rozděleny na dvě poloviny – horní a dolní. Každá polovina je dělena na řádky výšky odpovídající 16 CLB (8 DSP, 32 IOB) nebo 4BRAM. V rámci řádku je prostředním slovem rámce konfigurována část hodinových rozvodů (v [53] označovány jako HCLK). Řádky jsou dále děleny na sloupce o šířce jednoho prvku a každý sloupec je jemně rozdělen pomocí vedlejší adresy sloupce. Počet vedlejších adres sloupce v rámci „hlavní“ adresy sloupce je dán typem rámce; Jako příklad uveďme 22 rámců konfigurujících sloupec 16 CLB, 21 rámců oblast konfigurujících 4×1 DSP, apod.

Tabulka 7: Adresa rámce

segment	polovina FPGA	typ rámce	řádek	sloupec	vedlejší sloupec
bit	22	21 .. 19	18 .. 14	13 .. 6	5 .. 0

Fyzickou adresu rámce znázorňuje tabulka 7, ve které první řádek označuje typ segmentu adresy a druhý řádek polohu a počet bitů vyhrazených danému segmentu. Konfigurace zařízení začíná od horní poloviny směrem nahoru doprava a poté pokračuje osově symetricky směrem dolů. V dolní polovině FPGA jsou bity rámců reverzovány, až na prostřední slovo rámce konfigurující HCLK.

Z podrobné znalosti kompozice rámců v bitstreamu lze odhadnout poměr dat určených ke konfiguraci logiky a propojení. Xilinx v [16] uvádí, že více než 60% konfigurační paměti je vyhrazeno pro konfiguraci propojovací matice. V [13] je uváděno, že prvních 20 slov rámce typu 0 je určeno pro konfiguraci propojení. Tvrzení lze ověřit: dva rámce o kapacitě 80 slov (2 slova pro HCLK) nastavují 16 CLB = 128 LUT = 2048 konfiguračních bitů. Z kapacity rámců zbývá 512 bitů, z nichž připadá 8 bitů na každý SLICE. Tyto bity by měly nastavovat 2 klopné obvody, parametry LUT v režimu distribuované paměti a budiče uvnitř SLICE. Hrubou vizuální kontrolou (viz obrázek 4) počet bitů odpovídá.

Poměr konfiguračních dat propojení a logiky se dle výše uvedených zdrojů nachází v intervalu 60 – 90%, přičemž se domnívám, že dolní mez intervalu platí pro celý obvod a horní mez lépe vystihuje situaci v oblasti složené z CLB.

3.7.2 Přístup do konfigurační paměti, řadič ICAP

Externí přístup do konfigurační paměti je zajištěn pomocí sériových (např. SPI, JTAG) nebo paralelních rozhraní (BPI, SelectMAP). Zevnitř obvodu lze k řadiči konfigurační paměti přistupovat přes interní kopii rozhraní SelectMAP tzv. ICAP (Internal Configuration Access Port). ICAP umožňuje snadné vytváření systémů autonomně se rekonfigurujících systémů.

Rozhraní ICAP je celkem standardní: řadič komunikuje pomocí dvou 32 bitových datových portů *I* a *O*, obsahuje hodinový vstup *CLK*, řídící signály povolení hodin *CE*, povolení zápisu *WRITE* a kontrolní port *BUSY*. Protokol komunikace sestává z víceslovných příkazů. První slovo volí typ příkazu (jednoslovný / dávkový) a adresu registru, druhá část příkazu obsahuje data. Popis registrů s příklady komunikačního protokolu lze nalézt v [53].

Vložení ICAP do rekonfigurovatelného systému je snadné, pro SoPC systémy nabízí výrobce dokonce IP komponentu XPS_HWICAP. Komponenta sestává z instance ICAP, jednoduchého stavového automatu řízeného uživatelskými registry a z vyrovnávacích pamětí FIFO o kapacitě 1024 byte pro čtení a 4096 byte pro zápis. XPS_HWICAP je připojitelný na systémovou sběrnici pomocí jako PLB slave modul s podporou dávkových přenosů dat.

Systémová sběrnice, na kterou je komponenta připojena výrazně omezuje rychlost rekonfigurace a tím i aplikovatelnost představované metody testování. Při PIO módu se rychlost rekonfigurace pohybuje okolo 6 MB/s, s použitím řadiče DMA [27]

lze dosáhnout rychlostí přes 55 MB/s. Jelikož ani tato rychlost není dostatečná, bylo přistoupeno k návrhu vlastního řadiče ICAP.

XCL_ICAP, jak byla komponenta řadiče nazvána, je minimalisticky navržený řadič, jehož cílem je maximalizovat rychlost rekonfigurace a tím zkrátit dobu testu. Komponenta obsahuje jednoduchý stavový automat, instanci prvku ICAP a základní slave rozhraní sběrnice PLB. Krom řídícího rozhraní obsahuje XCL_ICAP i samostatný komunikační kanál protokolu XCL (Xilinx Cache Link), který je připojen na datový port řadiče paměti SRAM.

Samostatný kanál má teoretickou propustnost 400MB/s, která je shodná s maximální možnou propustností ICAP. Režie na stavovém automatu a protokolu XCL snižují šířku pásma na 198,7MB/s, ale stále se jedná o téměř čtyřnásobné zrychlení oproti způsobu uvedenému v [27]. Řadič podporuje pouze zápis do konfigurační paměti, čtení paměti podporováno není, jelikož není v práci používáno. Implementace čtení by rychlost rekonfigurace nesnižovala.

Tabulka 8: uživatelské registry XCL_ICAP

offset	Jméno registru
0x00	Řídící registr
0x04	Stavový registr
0x08	Registr adresy
0x0C	Registr délky

XCL_ICAP je ovládán čtveřicí uživatelských registrů uvedených v tabulce 8. Řídící registr ovládá softwarový reset komponenty a řízení rekonfigurace, stavový registr informuje o stavu probíhající rekonfigurace. Do registru adresy se zadává adresa bitstreamu v systémové paměti a do registru délky jeho délka v násobcích 32 bit slov. Délka je omezena na 65535 slov, což odpovídá přibližně polovině použitého FPGA (Virtex4-FX12). Ke komponentě byly vytvořeny ovladače v jazyce C.

3.7.3 Hrubozrnné rekonfigurovatelné systémy

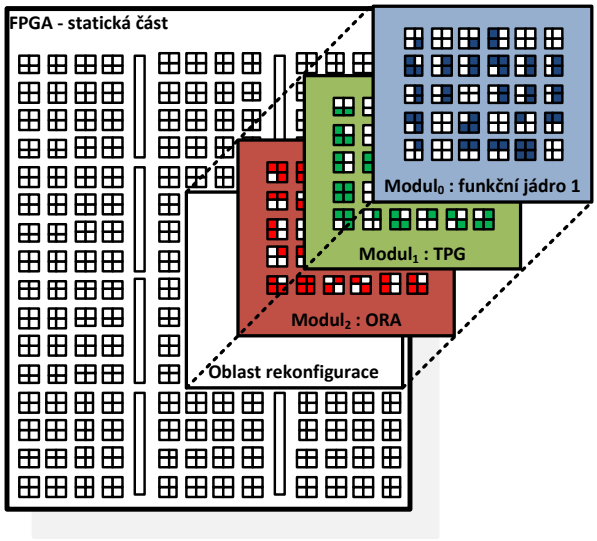
Se znalostí přístupu do konfigurační paměti a adresace rámců by bylo teoreticky možné přímo provádět jemnozrnnou dynamickou rekonfiguraci. Drobný detail v podobě neznalosti souvislostí mezi prvky FPGA a strukturou konfigurační paměti posouvá myšlenku rovnou do kategorie nerealizovatelných. Na starších architekturách (Virtex, VirtexII) byl k dispozici nástroj Jbits [20], který dával uživateli kontrolu nad konfigurací logiky i propojení a tím i možnost jemnozrnné rekonfigurace, ale verze pro novější architektury zřejmě nikdy nevznikne. V oblasti jemnozrnné

rekonfigurace pro obvody Virtex4 existuje pouze možnost volat předkompilované knihovny dodávané s ovladači XPS_HWICAP, které umožňují měnit bity CLB. Uzavřená architektura FPGA Virtex4 uživatelům dovoluje využít prakticky jen hrubozrnnou rekonfiguraci.

Návrh hrubozrnného rekonfigurovatelného systému je pro většinu uživatelů paradoxně výhodnější, poněvadž nevyžaduje tolik znalostí o obvodu, jako rekonfigurace jemnozrnná. Postup návrhu rekonfigurovatelného systému se v mnohém podobá postupu používanému při modulárnímu návrhu.

Na počátku je obvod rozdělen na nejméně dva celky – statickou a dynamickou část. Statická část obsahuje všechny prvky, které nebudou měněny: krom programovatelných bloků se do statické části umísťují především instance IO bloků nebo managementu hodin (DCM, PLL). Umístění statické části v FPGA nemusí být nikterak specifikováno, ačkoliv to zakázáno není.

Pro lepší orientaci v následujícím textu definujme pojmy *rekonfigurovatelná oblast* a *rekonfigurovatelný modul*. Oblastí budeme označovat fyzickou část FPGA ve které bude docházet k dynamické rekonfiguraci. Modulem budeme chápat konkrétní konfiguraci příslušející jedné oblasti; V našem případě jimi budou moduly funkčního jádra, TPG, ORA a další. Ke každé rekonfigurovatelné oblasti přísluší více modulů, zatímco jeden konkrétní modul přísluší právě jedné oblasti. Oba termíny jsou ilustrovány na obrázku 27.



Obrázek 27: Oblasti a moduly

Rekonfigurovatelná oblast musí splňovat několik základních podmínek, které se týkají umístění, velikosti a způsobu zacházení se signály procházejícími mezi statickou a dynamickou částí obvodu.

Umístění rekonfigurovatelné oblasti se řídí logickou adresou, která se odlišuje od adresy fyzické představené v 3.7.1. Logická adresace nedělí obvod na rámce, ale na bloky programovatelných prvků adresovaných pomocí kartézských souřadnic $\{x, y\}$ s počátkem souřadného systému v levém dolním rohu obvodu. Souřadnice x se zvyšuje směrem doprava, y směrem nahoru. Nejmenším adresovatelným prvkem logické adresy jsou celé SLICE, BRAM, DSP, IOB apod.

Rekonfigurovatelné oblasti v obvodech Virtex4 mohou být umístěny libovolně tak, aby nedělily žádný blok CLB. Prakticky to znamená, že oblast rekonfigurace může tvořit jakýkoliv pravoúhlý mnohoúhelník s levými a dolními okraji umístěnými na sudých adresách SLICE a pravými a horními okraji umístěnými na lichých adresách SLICE. Deklarace rekonfigurovatelné oblasti se uvádí v UCF souboru a má podobu:

INST "*jméno instance* " AREA_GROUP = "*jméno oblasti*";
AREA_GROUP "*jméno oblasti*" MODE = RECONFIG;
AREA_GROUP "*jméno oblasti*" RANGE = SLICE_X0Y0:SLICE_X1Y1;

Všechny prvky CLB v oblasti jsou vyhrazeny pro rekonfigurovatelné moduly a zapovězeny užití v oblasti statické. Kromě managementu hodin může rekonfigurovatelná oblast zahrnovat i jiné zdroje než CLB, ale ty musí být v UCF specifikovány a jejich umístění musí podléhat obdobným pravidlům jako u oblastí tvořených výlučně z CLB. Oblasti mohou být specifikovány obdobně, jako na následujícím příkladu:

AREA_GROUP "*jméno oblasti*" RANGE = DSP48_X0Y0:DSP48_X1Y1;
AREA_GROUP "*jméno oblasti*" RANGE = BRAM_X0Y0: BRAM _X1Y1;

Přiřazením parametru RANGE dojde k rezervaci programovatelných prvků, ale ne k rezervaci vodičů propojovací matice. Vodiče uvnitř rekonfigurovatelné oblasti jsou využívány nejen rekonfigurovatelnými moduly, ale i statickou částí. Zamezení užití vodičů statickou částí může vést k horším výsledkům časování ve statické části obvodu,

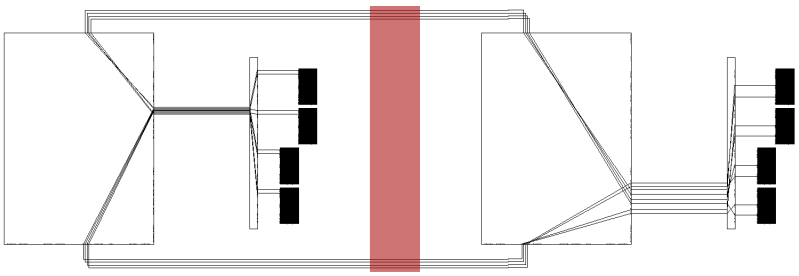
ale pro určitý typ aplikací je nevyhnutelné. Omezení, které není v dodávané dokumentaci uváděno, má tvar:

AREA_GROUP "*jméno oblasti*" ROUTING = CLOSED;

Znalost fyzické adresace rámců nám umožní najít nejvýhodnější umístění rekonfigurovatelné oblasti. Adresa A_l dolního okraje ideální oblasti musí splňovat podmínku $A_l \, mod \, 16 = 0$ a adresa horního okraje A_h podmínku $A_h \, mod \, 16 = 15$. Oblasti, které podmínky splňují, jsou složeny z nejmenšího počtu konfiguračních rámců, což je důležité nejen kvůli velikosti částečného bitstreamu a rychlosti rekonfigurace, ale i při návrhu systému s větším počtem rekonfigurovatelných oblastí.

Více rekonfigurovatelných oblastí se v obvodu vytvoří stejným způsobem jako oblast jedna; v souboru UCF je nutno doplnit záznamy s dalšími jmény instancí a oblastí. Vícenásobné oblasti musí být umístěny tak, aby mezi nimi nedocházelo ke sdílení konfiguračních rámců, což je snadné splnit, pokud návrhář při jejich umisťování dodrží předchozí podmínky.

Speciální péče je věnována signálům, které prochází mezi statickou a dynamickou částí obvodu. Každý z nich musí procházet speciální konstrukcí, kterou budeme označovat jako *oddělovací element* (v [51] je označen termínem *bus macro*). Oddělovací element je dodáván v podobně fyzického makra, které se skládá z dvojice CLB umístěné dílem ve statické a dílem v rekonfigurovatelné oblasti. Signály procházející mezi vstupními a výstupními LUT jsou v makru předem definovány a opatřeny atributem „*Allow Boundary Cross: true*“, který jim umožní průchod hranicí statické a dynamické oblasti. Každý oddělovací element obhospodařuje osmici signálů. Oddělovací element společně s hranicí statické a dynamické oblasti je znázorněn na obrázku 28.



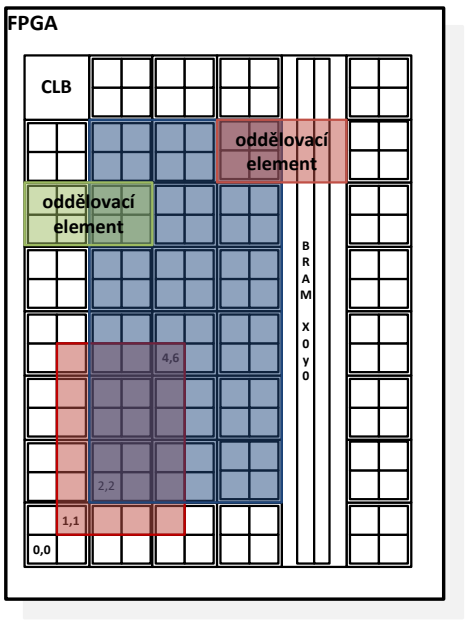
Obrázek 28: Oddělovací element

Užití oddělovacího elementu má dvojí význam. Primárně je elementu zapotřebí kvůli algoritmu PAR, který jej využívá k jednoznačnému určení počátku a konce šíření

signálu. Sekundárním účelem je zabránit šíření zákmitů, které vznikají při přepisu konfigurační paměti a z rekonfigurovatelné oblasti pronikají do statické části obvodu. Druhý důvod je uplatnitelný pouze u elementu vedoucích signály z rekonfigurovatelné oblasti. Doplňme, že elementy nemusí procházet globální signály typu VCC, GND nebo hodinové signály.

Oddělovací element je ve verzi pro Virtex4 dodáván v 32 variantách dle směru šíření signálu {zleva doprava, zprava doleva, seshora dolů, zdola nahoru}, instance klopného obvodu {synchronní, asynchronní}, šířky {úzké, široké} a dle přítomnosti blokovací signálu. Při návrhu obvodů uvedených v 4.1 a 4.2.2 byly použity výhradně úzké asynchronní oddělovací elementy vedoucí signál zleva doprava.

Instance oddělovacího elementu se vkládá do statické části obvodu. V případě většího počtu signálů je tato činnost doslova úmorná – vkládání elementů je v práci řešeno pomocí skriptů *make_feedback* nebo *PRParser*. Podobně jako rekonfigurovatelnou oblast i element je zapotřebí fyzicky umístit. Umístění je deklarováno v souboru UCF a podléhá jednoduchým pravidlům: levý dolní roh elementu je umístěn na sudých adresách *x* a *y*. Umístění elementu do statické nebo dynamické části obvodu je závislé na směru šíření signálu a užití oddělovacího elementu (vstup do/z rekonfigurovatelné oblasti).



Obrázek 29: Rekonfigurovatelné oblasti a oddělovací elementy

Adresní prostor CLB není spojitý: sloupce CLB jsou v obvodu přerušovány sloupci s jinými programovatelnými bloky. Úzký oddělovací element, který propojuje

sousední bloky CLB proto není možné umístit na hranice sloupců CLB. Příklad správného a špatného umístění oddělovacích elementů a rekonfigurovatelných oblastí je ilustrován na obrázku 29. Modrou barvou je znázorněna korektně, ale nevýhodně umístěná rekonfigurovatelná oblast, červenou nesprávně umístěná oblast. Zeleně je označen správně umístěný oddělovací element, červeně je označen element umístěný špatně.

Jsou-li definovány rekonfigurovatelné oblasti a umístěny oddělovací elementy je možno rekonfigurovatelný obvod implementovat. Implementace statické a dynamické části probíhá zčásti odděleně, většinou odspoda vzhůru; všechny moduly jsou syntetizovány samostatně, včetně hierarchicky nejvyššího, ve kterém jsou jejich instance uvedeny jako „černé skříňky“. Moduly jsou s hierarchicky nejvyšším složeny odděleně – nejprve je složena statická část obvodu, posléze část dynamická. Implementace obou části vyžaduje modifikované nástroje MAP, PAR, Bitgen a několik skriptů dodávaných společně s opravnou záplatou, která částečnou rekonfiguraci umožňuje [51].

Požadavky na rekonfigurovatelné moduly nejsou nikterak svazující: Moduly musí mít shodné rozhraní a musí se vejít do rekonfigurovatelné oblasti. Modul nesmí být popsán jako prázdná obálka a musí obsahovat alespoň nějaký programovatelný prvek.

Velikost a umístění rekonfigurovatelných oblastí a oddělovacích elementů je pro algoritmy MAP a PAR poměrně restriktivní a u rekonfigurovatelných modulů může docházet k významné degradaci parametrů časování. Obecně lze tvrdit, že čím je oblast menší, tím je dosaženo horších výsledků. Podobné pravidlo platí i pro umístění oddělovacích elementů. Zpoždění signálu je přímo úměrné vzdálenosti oddělovacího elementu od budiče signálu (roli hraje především zpoždění signálu v propojovací matici).

Výsledkem postupu představeného v této kapitole je sada částečných konfiguračních souborů vytvořených pro každý unikátní modul. Každý konfigurační soubor je opatřen ochranným CRC kódem. Chceme-li data v částečném konfiguračním souboru měnit je nutno kontrolu CRC vypnout, nebo při každé změně souboru hodnotu CRC znovu vypočítat. Vypnutí docílíme přepsáním parametru (původně hodnoty CRC), který následuje po příkazu 30000001_{16} (zapiš do CRC registru) na hodnotu $0000DEFC_{16}$.

Postup návrhu hrubozrnného rekonfigurovatelného obvodu popsaný v předchozím textu byl kvůli značné náročnosti výrazně zautomatizován. Byly vytvořeny skripty, které vkládají rekonfigurovatelné moduly do obvodu, skripty, které vytvářejí instance oddělovacích elementů a propojují statickou část a rekonfigurovatelné oblasti a skripty, které automaticky spouštějí procesy implementace jednotlivých částí návrhu.

Prezentovaný postup je, až na výjimky, uplatnitelný pouze pro nástroje verze 9.2PR7(8) a obvody Virtex4. Pro novější obvody a verze návrhového systému 11 a vyšší je postup rekonfigurace změněn [52], například odpadá nutnost vkládat oddělovací elementy, tvorba systému je realizována v grafickém návrhovém prostředí apod. Daní za uživatelskou přívětivost je omezená možnost manuálně vstupovat do procesu návrhu, čímž mírně klesá využitelnost částečné rekonfigurace pro atypické aplikace.

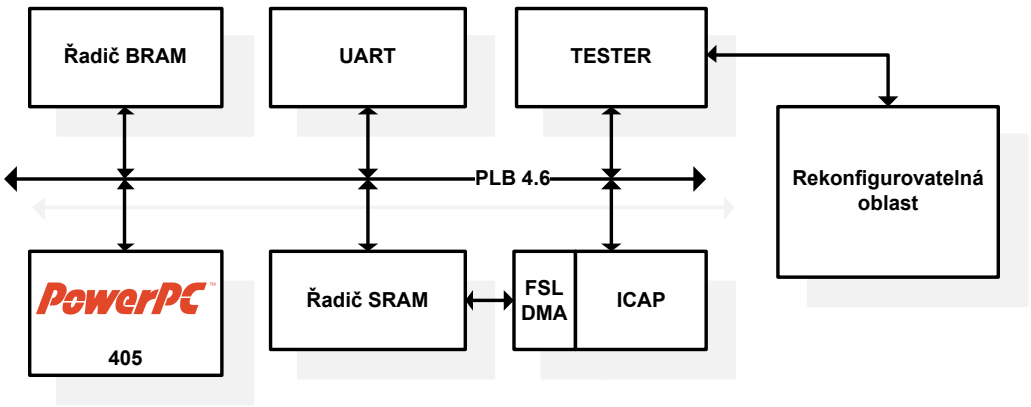
4 Experimenty

V následujícím textu jsou popsány experimenty, které byly s dílčími částmi metody prováděny. Experimenty jsou seřazeny chronologicky tak, jak se práce na prezentované metodě postupně vyvíjela.

4.1 Injektor poruch – experiment 1

V úvodu kapitoly 3 bylo zmíněno, že by testovací vektory měly detekovat jak strukturní poruchy, tak poruchy konfigurační paměti způsobené SEU. Ověření detekce strukturních poruch je možné pouze tak, jak bylo prezentováno v 3.3. Za účelem ověření detekce poruch způsobených SEU bylo nutno vytvořit experimentální emulátor.

Hlavním cílem, kvůli kterému emulátor vznikl, bylo rychlé ověření vlivu změny konfigurační paměti na testovaný obvod a ověření schopnosti vygenerovaných testovacích vektorů případné poruchy odhalit. Stav práce v okamžiku provádění experimentu nebyl zdaleka ve fázi, která je popisována v předchozím textu. Z hardwarového pohledu nebylo implementováno testování okolními moduly, autonomní řízení testu nebo dekomprese testovacích vektorů. Softwarová část postrádala spolupráci algoritmu COMPAS se simulátorem poruch, ATPG byl spouštěn s celou řadou odlišných nastavení a pracoval na odlišném modelu FPGA. Pokus byl realizován hlavně proto, aby se ověřily základní postupy metody, tj. aby se zjistilo, zdali je vůbec možné s dostupnými prostředky metodu realizovat, případně zjistit, jaká omezení pro metodu z experimentu vyplynou.



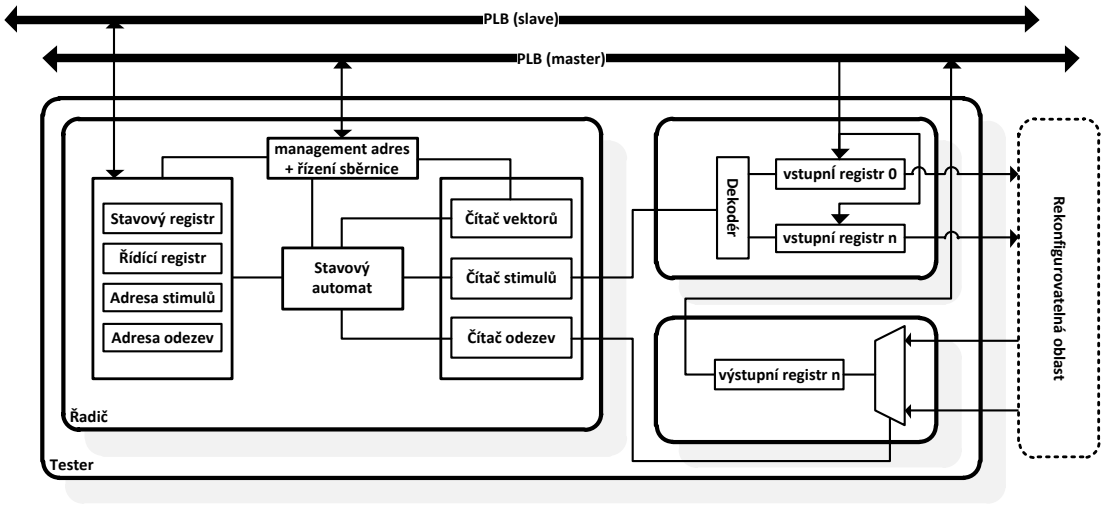
Obrázek 30: Blokové schéma injektoru SEU poruch

Systém injektoru SEU poruch je systémem na programovatelném čipu (SoPC) a skládá se z procesoru PowerPC 405, řadičů interní paměti BRAM, externí paměti

SRAM, řadiče paměťové karty Compact Flash, modifikovaného řadiče ICAP a modulu testeru. Komunikace systému s PC je zajištěna přes rozhraní RS232. Systém byl realizován na vývojové desce ML403 osazené FPGA Virtex-4 FX 12 a pro jeho tvorbu bylo užito software XPS 9.2.02 a syntézní nástroje verze 9.2.04i s opravnou záplatou PR7/8. Blokové schéma injektoru je vyobrazeno na obrázku 30.

4.1.1 Tester a testovaný obvod

Úlohou testeru (obrázek 31) je řízení testu, přivádění testovacích vektorů a snímání odezev. Tester disponuje třemi rozhraními. První dvě – master a slave jsou zapojena k 32 bitové sběrnici PLB taktované na 100 MHz a slouží ke konfiguraci parametrů probíhajícího testu a přenosu testovacích dat. Tester je ovládán přes paměťově mapované registry slave rozhraní. Význam registrů je uveden v tabulce 9. Master rozhraní umožňuje testeru číst testovací data a ukládat odezvy testu do sdílené systémové paměti bez zapojení procesoru. Mechanismus samostatného přístupu do paměti byl zvolen ze dvou důvodů. Prvním důvodem je vyšší rychlost testu, která snižuje dobu, po kterou je testovaný obvod vystaven vlivu poruchy a druhým je nízká kapacita BRAM zvoleného FPGA, která neumožňuje kompletní uložení nezkomprimovaného testu. S autonomním přístupem do paměti je navíc možné ukládat i odezvy, které mohou sloužit k účelům diagnostiky.



Obrázek 31: Principiální schéma modulu testeru

Třetí rozhraní přivádí testovací data do testovaného obvodu. Je tvořeno dvojicí registrů – vstupním a výstupním. Oba registry jsou řízeny stavovým automatem řadiče

testeru, který do nich ukládá data přiváděná z master PLB rozhraní a posléze z nich vyčítá odezvy. Registry jsou napojeny na testovaný obvod přes oddělující element. Vstupní registr má kapacitu 64 bitů a výstupní registr 32 bitů.

Tabulka 9: Uživatelské registry

Jméno	**0x0 Stavový registr**				
Bit	0 … 4	5	16 … 31		
Popis	Stavový index automatu	Bit indikace dokončení	Stav čítače vektorů		
Jméno	**0x4 Řídící registr**				
Bit	0	1	4 … 7	8 … 11	16 … 31
Popis	Reset	Spuštění	Délka dávky stimulů	Délka dávky odezev	Počet vektorů
Jméno	**0x8 Registr adresy stimulů**				
Bit	0 … 31				
Popis	Fyzická adresa počátku testovacích vektorů v systémové paměti				
Jméno	**0xC Registr adresy odezev**				
Bit	0 … 31				
Popis	Fyzická adresa systémové paměti alokované pro ukládání odezev				

Rekonfigurovatelná oblast má velikost čtyř rekonfiguračních rámců, což odpovídá 512 LUT (cca 5% z celkového objemu zdrojů FPGA). Uvnitř rekonfigurovatelné oblasti se nachází i osm DSP48 bloků, které nejsou využity. Ačkoliv může mít testovaný obvod až 128 vstupů a stejný počet výstupů, byl jejich počet omezen na 64 vstupů a 32 výstupů kvůli omezení doby testu a objemu zdrojů rekonfigurovatelné oblasti. Počet primárních vstupů a výstupů obvodů ovlivňuje využitelnost logiky uvnitř oblasti, poněvadž každý vstup nebo výstup požaduje jednu vyhledávací tabulku na straně rekonfigurovatelné oblasti. IO režie ubírá 18,75% z celkového množství dostupných zdrojů v rekonfigurovatelné oblasti a to i v případě, že testovaný obvod takový počet vstupů a výstupů nemá – nevyužité vstupy jsou součástí unifikované obálky, kterou je každý testovaný obvod při implementaci obalen.

Z benchmarkových obvodů byly vybrány takové, která uvedená kritéria splňují. Mezi tyto obvody patří již zmíněná podmnožina obvodů ISCAS85 a ISCAS89. Ze sady ISCAS89 byly odstraněny klopné obvody. Každý obvod byl implementován jako částečně rekonfigurovatelný modul v prostředí ISE 9.2.04i PR7 (platforma win32).

4.1.2 Vkládání poruch

Testování obvodů probíhalo následujícím způsobem: Nejprve byla do záložní kopie částečného bitsteramu procesorem vnořena porucha – jeden z bitů byl přepsán na inverzní hodnotu. Posléze byla poškozená konfigurační data nahrána přes modifikovaný ICAP řadič do paměti zařízení a zápisem do konfiguračních registrů testeru byl započat

test. Po ukončení testu byl do konfigurační paměti FPGA nahrán záložní nepoškozený bitstream. Obvod byl znovu otestován, zdali se v něm nevyskytují trvalé poruchy. Testování jedné poruchy bylo limitováno časovým rámcem 3ms. Pokud byl počet vektorů příliš vysoký, pak se testování rozdělilo na několik fází. Odezvy byly vyhodnoceny po samotném testu – bylo tak možno poruchu nejen detekovat, ale zároveň zjistit, který vektor ji detekuje. Velikost všech částečných bitstreamů (s vypnutou kompresí a odstraněnými hlavičkami) byla 144320 bitů, což odpovídá počtu všech možných SEU poruch v konfigurační paměti, která přísluší testovanému obvodu.

Ověřováno bylo několik sad testovacích vektorů. První sada byla vygenerována v ATPG TetraMAX se standardními nastaveními a vysokým stupněm interní komprese (nastavení „merge high“). Druhá sada byla generována v témže ATPG s tím, že byl pro každou poruchu vygenerován jeden testovací vektor, obsahující hodnoty don't care (nastavení „fill -X“). Tyto vektory byly zkomprimovány v dřívější verzi programu COMPAS, který v tomto experimentu nepracoval se simulátorem poruch. Další sada deterministických vektorů byla vygenerována v ATPG Atalanta pro původní variantu obvodu (sada označovaná ASIC). Čtvrtou sadou byla předem připravená posloupnost pseudonáhodných testovacích vektorů (v tabulkách jsou označeny jako NÁHODNÉ). Poslední sada byla vytvořena ze čtvrté sady tak, že byl počet vektorů v sadě redukován tak, aby se shodoval s počtem vektorů testu COMPAS. Tyto vektory jsou uváděny jako KRÁTKÉ NÁHODNÉ.

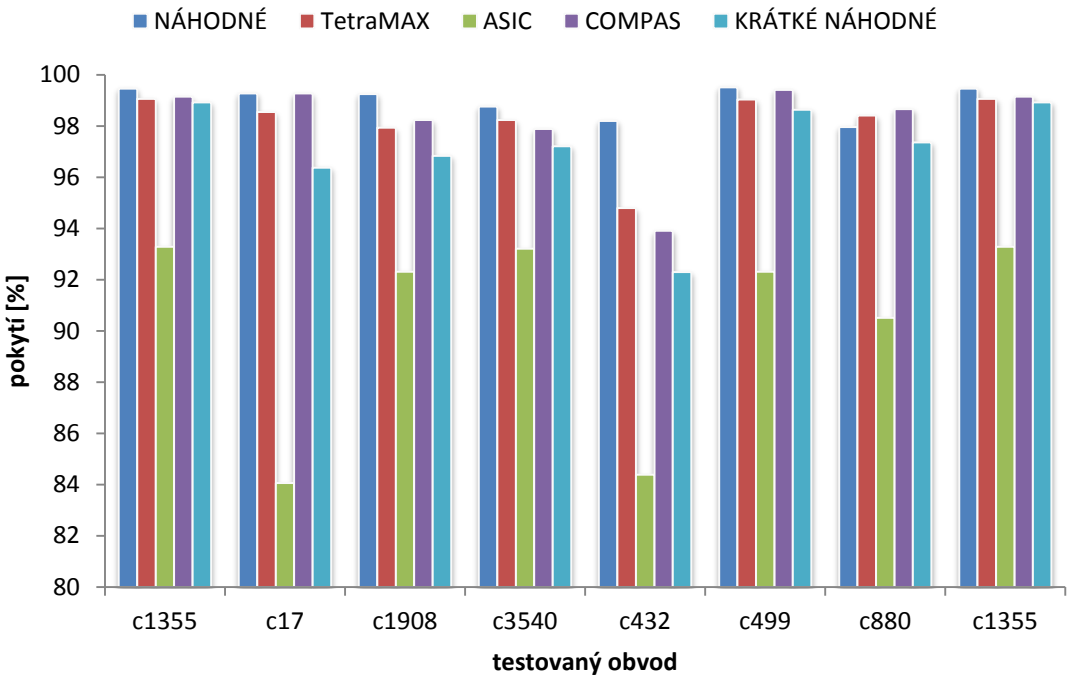
Z prvních výsledků testu byl vytvořen seznam poruch, které byly detekovány všemi testy (druhý řádek tabulek 5 a 6) a ten byl postupně porovnáván s výstupy všech testů. Pokud byla nalezena porucha, kterou jeden z testů nedetekoval, byla zařazena do seznamu podezřelých poruch. Tyto poruchy byly opakovaně (200 krát) na emulátoru testovány, aby bylo vyloučeno, že se jedná o poruchy s nahodilým chováním. Pokud se poruchy chovaly nahodile (tj. tak, že je některý test detekoval v méně než 100% případů) byly poruchy smazány ze seznamů poruch detekovaných všemi testy, ale zůstaly zachovány v seznamu detekovatelných poruch. Počet poruch s náhodným chováním je vynesen ve čtvrtém řádku tabulek 10 a 11 a je v nich označen jako náhodné poruchy. Třetí řádek tabulky udává poměr mezi počtem poruch, které byly detekovány (včetně poruch s náhodným chováním) a počtem všech poruch vložených do bitstreamu (hodnota je u všech obvodů stejná - 144320). Výsledky experimentu jsou zaneseny v tabulkách 10 a 11 a v grafech na obrázcích 32, 33, 34, 35, 36 a 37.

Tabulka 10: Výsledky experimentu na ISCAS 85 obvodech

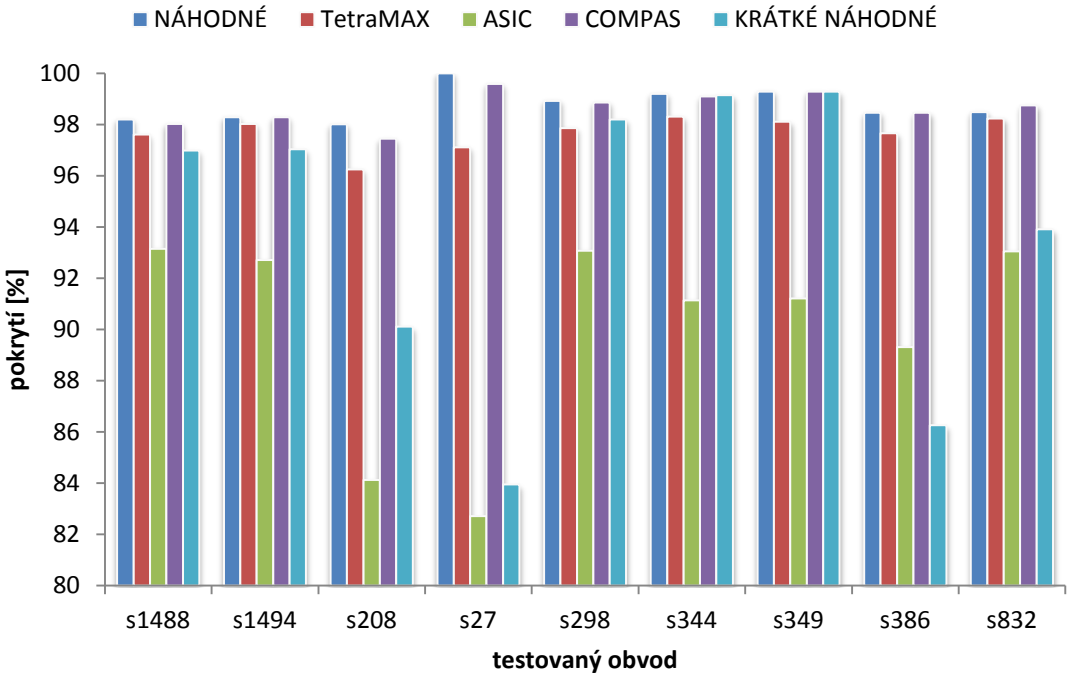
Obvod		**c1355**	**c17**	**c1908**	**c3540**	**C432**	**c499**	**c880**	**c1355**
Detekované poruchy		4100	138	5444	17461	3729	3965	5714	4100
Detekované / Všechny poruchy[%]		2,84	0,1	3,77	12,1	2,58	2,75	3,96	2,84
Náhodné poruchy		22	1	35	134	56	18	63	22
NÁHODNÉ	**pokrytí[%]**	99,46	99,28	99,25	98,76	98,2	99,52	97,97	99,46
	vektorů	10000	10000	10000	10000	10000	10000	10000	10000
TetraMAX	**pokrytí[%]**	99,07	98,55	97,94	98,24	94,8	99,04	98,42	99,07
	vektorů	212	17	277	460	201	179	156	212
	délka [bity]	8692	85	9141	23000	7236	7339	9360	8692
ASIC	**pokrytí[%]**	93,29	84,06	92,32	93,22	84,39	92,31	90,51	93,29
	vektorů	124	7	173	231	80	91	72	124
	délka [bity]	5084	35	5709	11520	2880	3731	4320	5084
COMPAS	**pokrytí[%]**	99,15	99,28	98,24	97,89	93,91	99,42	98,67	99,15
	vektorů	1099	26	1668	2182	423	824	5926	1099
	délka [bity]	1099	26	1668	2182	423	824	5926	1099
KRÁTKÉ NÁHODNÉ	**pokrytí[%]**	98,93	96,38	96,84	97,22	92,3	98,64	97,37	98,93
	vektorů	1099	26	1668	2182	423	824	5926	1099

Tabulka 11: Výsledky experimentu na ISCAS 89 obvodech

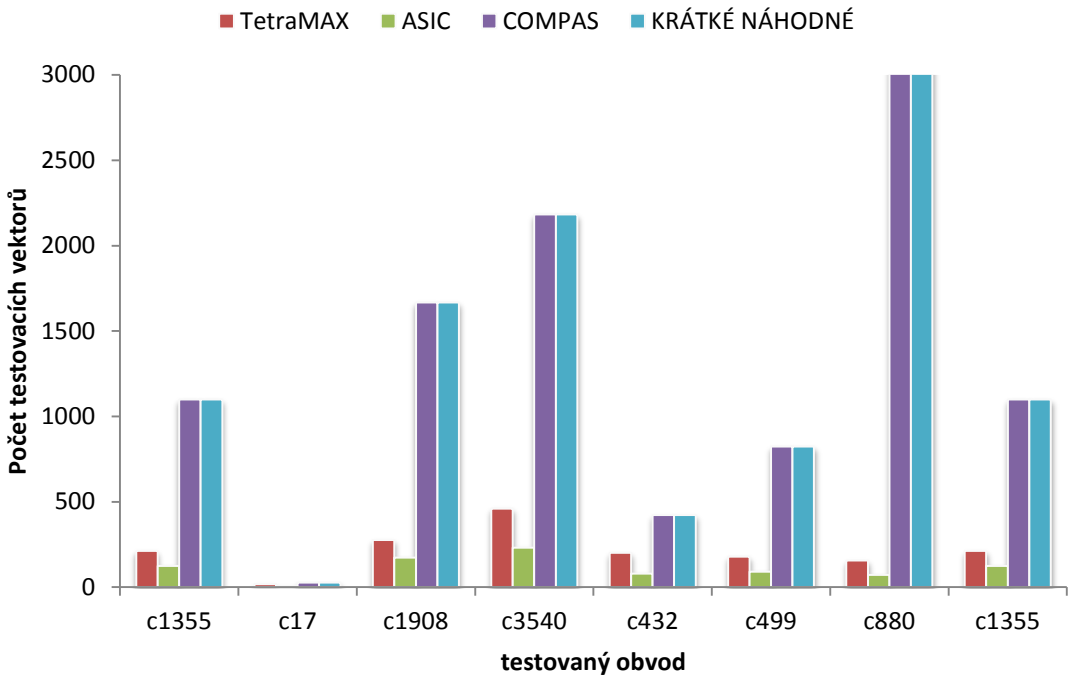
Obvod		**s1488**	**s1494**	**s208**	**s27**	**s298**	**s344**	**s349**	**s386**	**s832**
Detekované poruchy		10488	10323	1254	243	1502	2010	1959	2396	4647
Detekované / Všechny poruchy[%]		7,27	7,15	0,87	0,17	1,04	1,39	1,36	1,66	3,22
Náhodné poruchy		165	151	19	0	16	16	14	32	33
NÁHODNÉ	**pokrytí[%]**	98,2	98,29	98,01	100	98,93	99,2	99,29	98,46	98,49
	vektorů	10000	10000	10000	10000	10000	10000	10000	10000	10000
TetraMAX	**pokrytí [%]**	97,61	98,03	96,25	97,12	97,87	98,31	98,11	97,66	98,24
	vektorů	283	294	100	21	61	55	50	140	250
	délka [bity]	3962	4166	1900	147	1037	1320	1200	1820	5750
ASIC	**pokrytí [%]**	93,15	92,71	84,13	82,72	93,08	91,14	91,22	89,32	93,05
	vektorů	186	172	38	8	45	30	30	79	145
	délka [bity]	2604	2408	722	56	765	720	720	1027	3335
COMPAS	**pokrytí [%]**	98,03	98,29	97,45	99,59	98,87	99,1	99,29	98,46	98,75
	vektorů	5926	1747	1850	579	48	213	277	288	959
	délka [bity]	5926	1747	1850	579	48	213	277	288	959
KRÁTKÉ NÁHODNÉ	**pokrytí [%]**	96,99	97,04	90,11	83,95	98,2	99,15	99,29	86,27	93,91
	vektorů	1747	1850	579	48	213	277	288	959	2045



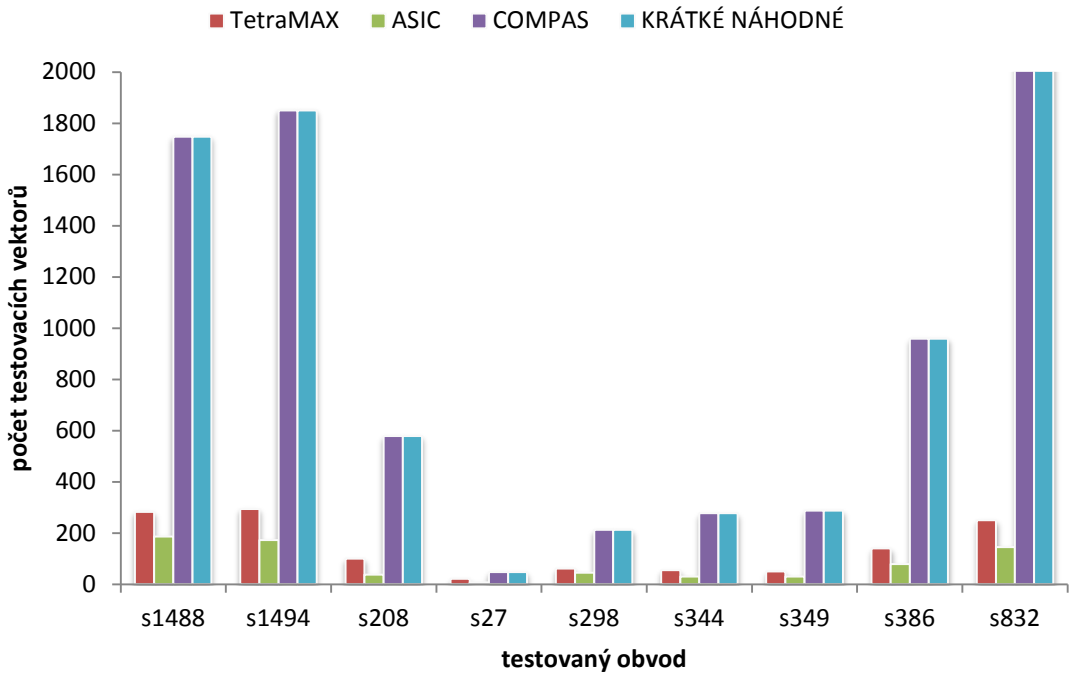
Obrázek 32: Srovnání pokrytí SEU pro různé sady testovacích vektorů a obvodů



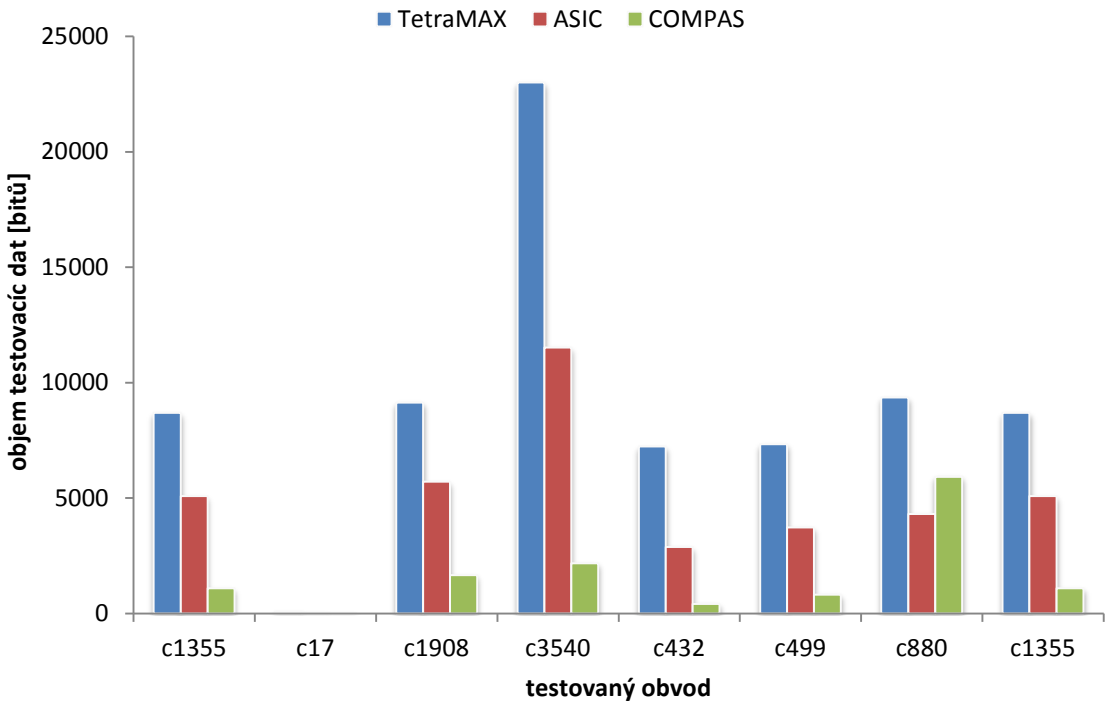
Obrázek 33: Srovnání pokrytí SEU pro různé sady testovacích vektorů a obvodů



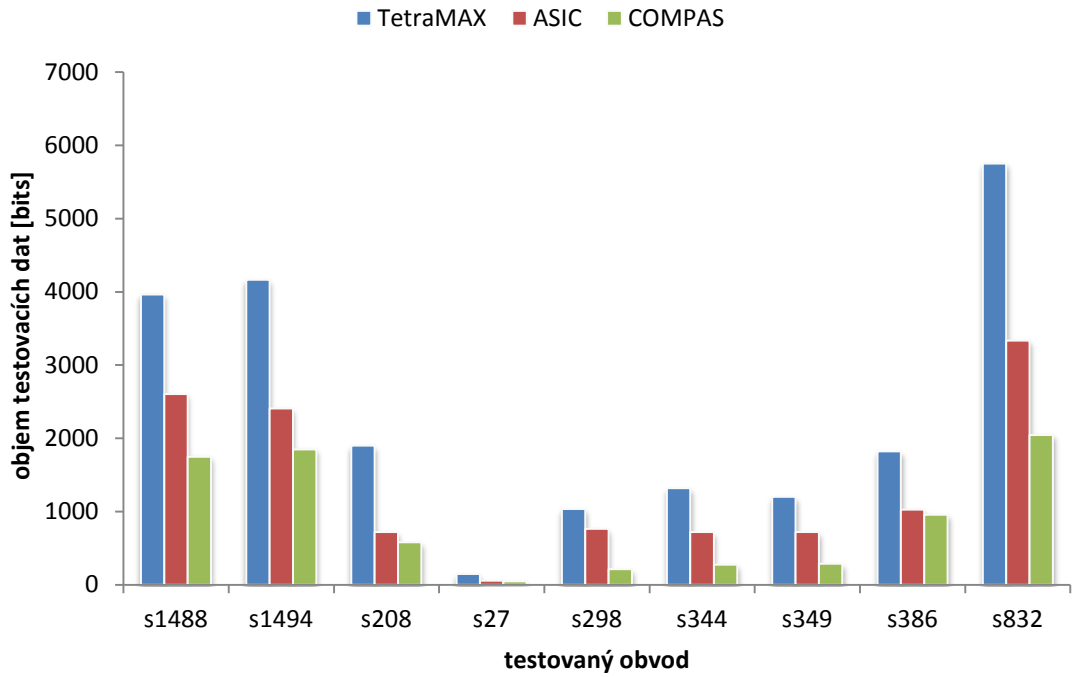
Obrázek 34: Srovnání počtu vektorů pro různé sady testovacích vektorů a obvodů



Obrázek 35: Srovnání počtu vektorů pro různé sady testovacích vektorů a obvodů



Obrázek 36: Srovnání objemu testovacích dat pro různé sady testovacích vektorů a obvody



Obrázek 37: Srovnání objemu testovacích dat pro různé sady testovacích vektorů a obvody

4.1.3 Zhodnocení výsledků

Zcela v souladu s očekáváním je nízká hodnota poměru detekovaných a injektovaných poruch u všech zkoumaných obvodů (řádek 2 v tabulkách 10 a 11). Údaj, jenž v průměru dosahuje hodnoty 3,26%, je oproti údajům, které poskytuje výrobce poměrně nízký (Xilinx v [16] udává, že 5% až 10% bitů konfigurační paměti má přímý vliv na chování obvodu). Přepočítáme-li naměřené hodnoty dle poměru využitých bloků vůči všem dostupným zdrojům v celé rekonfigurovatelné oblasti, pak se naměřené údaje shodují s továrními daty. Přepočet, kterým eliminujeme chybu měření poruch vložených do nevyužitých částí rekonfigurovatelné oblasti, je uveden v tabulkách 12 a 13. Do propočtu využití zdrojů jsou zahrnuty i vstupní a výstupní oddělovací elementy. Údaje potřebné pro výpočet byly získány z tabulek 2 a 3.

Tabulka 12: Přepočet poruch detekovaných testy

obvod	**c1355**	**c17**	**c1908**	**c3540**	**c432**	**c499**	**c880**	**průměr**
využití zdrojů [%]	31,84	2,15	33,98	84,77	25,20	31,84	40,23	
detekované poruchy [%]	2,84	0,10	3,77	12,10	2,58	2,75	3,96	4,01
přepočtený údaj [%]	8,92	4,45	11,10	14,27	10,26	8,63	9,84	9,64

Tabulka 13: Přepočet poruch detekovaných testy

obvod	**s1488**	**s1494**	**s208**	**s27**	**s298**	**s344**	**s349**	**s386**	**s832**	**průměr**
využití zdrojů [%]	60,74	59,96	10,35	3,32	13,09	16,41	16,41	15,63	31,45	
detekované poruchy [%]	7,27	7,15	0,87	0,17	1,04	1,39	1,36	1,66	3,22	2,68
přepočtený údaj [%]	11,96	11,93	8,39	5,07	7,95	8,49	8,27	10,63	10,24	9,22

Druhým údajem, který se nevztahuje k žádnému konkrétnímu testu, je počet náhodných poruch. Ty jsou nejspíše způsobeny zkraty nebo změnou kombinačního chování obvodu na sekvenční. Druhou možnost lze považovat za málo pravděpodobnou, vzhledem ke struktuře SLICE a jeho napojení na propojovací matice.

Nad rámec experimentu byl s náhodnými poruchami proveden pokus, jehož cílem bylo ověřit, zda počet detekovaných poruch alespoň přibližně odpovídá realitě. Experiment je časově náročný a nebyl proveden pro všechny obvody a testovací sady, ale pouze pro největší obvod c3540 a sadu testovacích vektorů zkomprimovaných algoritmem COMPAS. Pokus spočíval v mnohonásobné aplikaci téhož testu (200x) na každou injektovanou poruchu. Při pokusu bylo odhaleno 138 náhodně se chovajících poruch, což je přesně o 4 více, než v původním experimentu. Jedná se o 3% z původně zjištěných náhodných a 2,5‰ ze všech detekovaných poruch. Vzhledem k tomu, že

testovací sekvence pro obvod c3540 patří mezi nejdelší, využití zdrojů je ze všech obvodů nejvyšší a změna oproti původním údajům je prakticky nevýznamná, nebylo nutné experiment opakovat i pro další obvody. Informace o výskytu náhodných poruch lze prohlásit za hodnověrné.

Hlavním cílem experimentu bylo srovnání schopnosti různých testovacích dat detekovat poruchy způsobené změnou konfigurační paměti. V průměru nejvyššího pokrytí dosahují dlouhý náhodný test a sekvence COMPAS. Rozdíl mezi testem COMPAS a náhodným dlouhým testem je pouze v desetinách procent (často navíc ve prospěch pseudonáhodného testu) a to navzdory předpokladu, že bude deterministický test pokrývat větší procento poruch.

Jedním z faktorů, který způsobil nižší pokrytí poruch u deterministických testů, bylo to, že pro přepis obvodu byla použita starší verze algoritmu, která z obvodu odstraňuje budiče ovládané z konfigurační paměti. Vzhledem k tomu, že budiče mají v konfigurační paměti téměř stejné zastoupení jako vyhledávací tabulka, má jejich odstranění na kvalitu testu nezanedbatelný vliv. V nové verzi algoritmu již budiče zahrnuty jsou (viz 4.2).

Při porovnávání kvalit testů je nutné brát v potaz i délku testovací sekvence, která se odrazí v době, po kterou bude testovaný obvod nedostupný. Ta je v případě dlouhých náhodných testů několikanásobně delší, než v případě testů zkomprimovaných COMPASem nebo testů TMAX. Adekvátní srovnání nabízí krátký náhodný test, který svou délkou odpovídá délce testovací sekvence COMPAS. Porovnání pokrytí deterministických testů s krátkým pseudonáhodným testem hovoří v prospěch deterministických testů.

Testovací sady je vzhledem ke způsobu implementace generátoru testu nutno porovnávat i co se týče objemu testovacích dat. Na spotřebu zdrojů vychází lépe generátory pro oba pseudonáhodné testy, které vyžadují pouze zdroje potřebné pro implementaci zpětnovazebného registru a jednoduché řídící logiky. Zpětnovazebný posuvný registr je možné vložit do sousedního obvodu vždy.

Deterministické testy vyžadují komplexnější řadič (adresaci paměti vzorků), ale především paměť ROM, do které se budou ukládat. Vzhledem k tomu, že je kapacita ROM značně omezena, je objem dat velmi podstatným parametrem, na kterém závisí i doba testu (provádění dodatečných rekonfigurací testeru). Srovnání objemu potřebných dat je uvedené v grafech na obrázku 36 a obrázku 37. COMPAS dosahuje jednoznačně

nejlepších výsledků a to i při lepším pokrytí poruch, nežli test komprimovaný slučováním vektorů.

Závěrem experimentu je, že všechny testy, které byly generovány pro strukturní testování FPGA obvodu jsou schopny odhalit dostatečný počet poruch způsobených změnami v konfigurační paměti. Z hlediska navržené metody se nejvhodnějším testem jeví test COMPAS. Vysoký počet testovacích vektorů protestuje i poruchy, které nejsou zahrnuty v modelu obvodu, pro který byl test vytvořen. Způsob rekonstrukce vektorů je hardwarově nenáročný a paměťové nároky generátoru testeru jsou nízké. Experiment zdárně odpověděl i na otázku, zdali je možno testování obvodu FPGA pomocí představené metody realizovat.

4.2 Testování IP jader – experiment 2

Výrobci FPGA ke svým obvodům dodávají i vývojový software, který umožňuje implementaci uživatelských návrhů. Součástí těchto softwarových balíků bývají knihovny obsahující nejrůznější hardwarové prvky, které urychlují a zlevňují návrh obvodu. Prvky těchto knihoven bývají shodně označovány jako IP jádra (Intelecual Property Core).

Jádra dosahují různého stupně složitosti – od poměrně jednoduchých, které uživateli ulehčují nastavení vnořených struktur (konfigurace BRAM, DSP a jiné), až po velmi komplexní obvodové struktury (procesory, sběrnice, řadiče pamětí, matematické operace aj.). Jádra, jak placená, tak i bezplatná, bývají dobře zdokumentována, otestována a dosahují poměrně dobrých výkonů.

Běžnou součástí moderních systému na čipu se stále více stává jednotka výpočtů v plovoucí řádové čárce (FPU - Floating Point Unit), která akceleruje výpočetně náročné operace ve formátech definovaných IEEE754. Běžná FPU se skládá hned z několika IP jader: sčítačky, násobičky, z převodu do a z celočíselného formátu, v některých případech i z děličky či jiných operací.

Druhý experiment je kvůli rozšíření FPU jednotek v moderních SoC orientován na aplikaci představované metody na tato jádra. Na rozdíl experimentů s benchmarkovými obvody byly ověřovány všechny vlastnosti metody představené v kapitole 3.

4.2.1 Testovaná jádra

Testovaná jádra byla vytvořena generátorem floating_point_v3_0 z balíku Xilinx ISE 9.2.04i PR8 (platforma linux x86). Z generátoru byla zvolena jádra floatové sčítačky, násobičky, děličky a druhé odmocniny. Byla generována výlučně jádra se čtyřmi stupni pipeline. Generátor byl nastaven tak, aby do jader nevkládal DSP bloky a aby výpočet operací probíhal bez cyklů – tj. tak, aby byla vytvořena čistě datová cesta bez zpětných vazeb.

Netlisty, které generátor produkuje, byly implementovány jako samostatný rekonfigurovatelný modul a dle postupu uvedeného v kapitole 3.2 byly přepsány a následně rozděleny skriptem *split*. Detailní statistiky původních, rozdělených a implementovaných obvodů jsou zobrazeny v tabulce 14. Tabulka obsahuje parametry původního a čtyř nově vytvořených obvodů ve dvou variantách. Liché čtveřice řádků popisují parametry modulů po přepisu (index P), sudé čtveřice obsahují statistiky získané až po implementaci (index I).

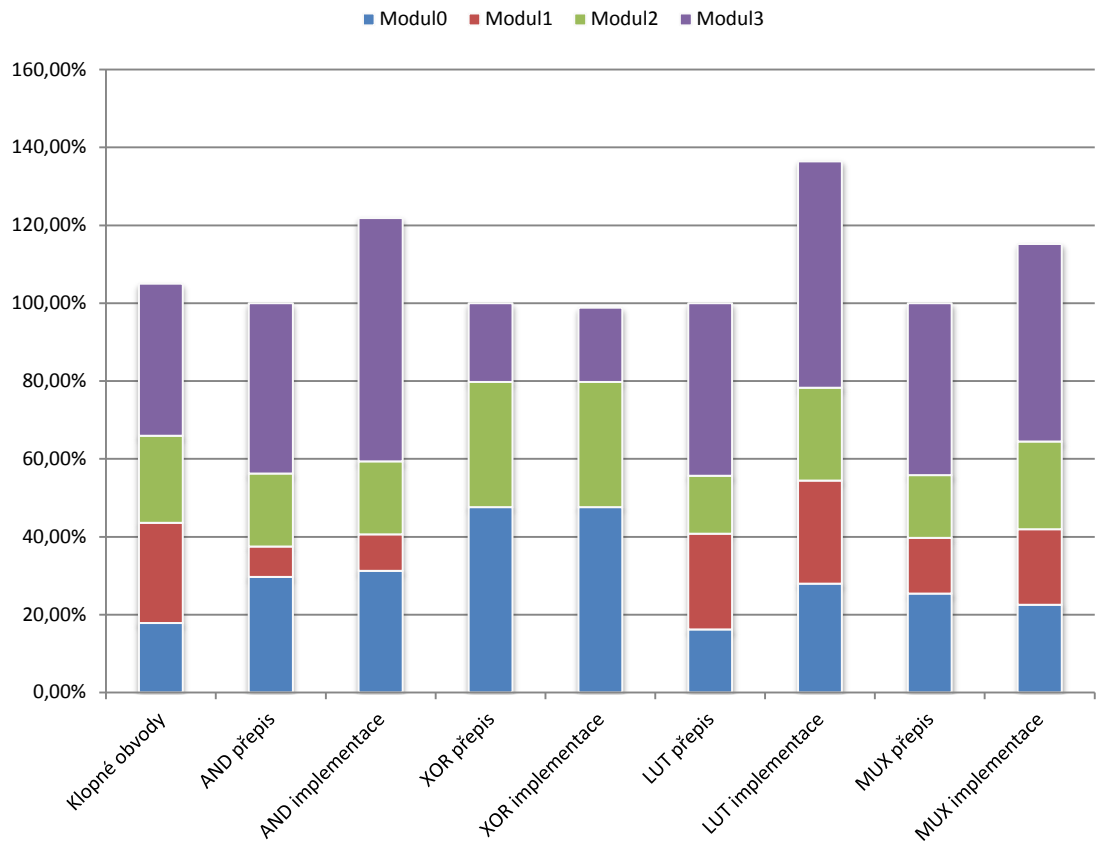
Sloupce FF, sFF a SRL označují počty sekvenčních prvků původního obvodu – klopný obvod, klopný obvod se synchronním vstupem set a vyhledávací tabulku v režimu posuvného registru. V rozdělených obvodech se vyskytuje pouze první varianta klopného obvodu, tudíž není rozdělení uváděno. Indexy názvů modulů jsou číslovány podle pořadí, dle kterého byly vytvářeny. Nejnižší index modulu odpovídá modulu umístěnému nejblíže původním primárním výstupům testovaného obvodu (poslednímu stupni pipeline), nejvyšší index odpovídá prvnímu stupni pipeline.

Tabulka 14: Parametry rozdělených obvodů

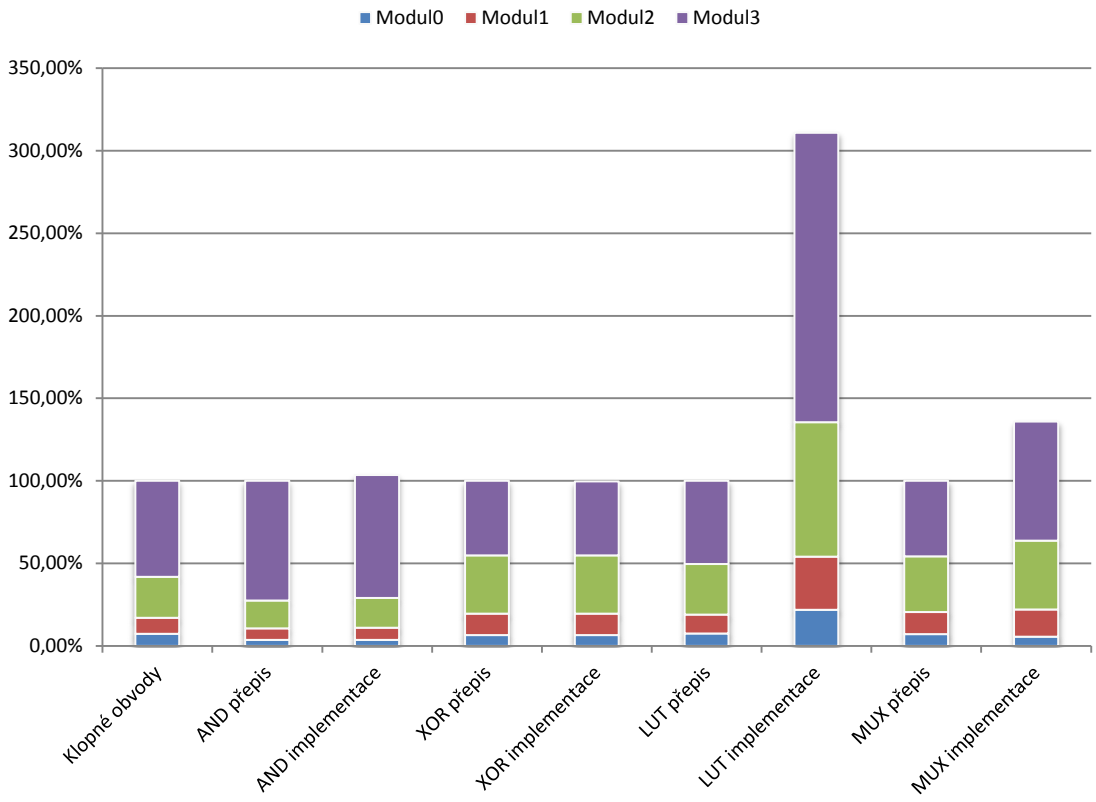
obvod		**FF**	**sFF**	**SRL**	**AND**	**XOR**	**LUT**	**MUX**
sčítačka	**nerozdělený**	136	31	12	64	84	544	315
	Modul$_{0P}$	32			19	40	88	80
	Modul$_{0I}$				20	40	152	71
	Modul$_{1P}$	46			5	0	134	45
	Modul$_{1I}$				6	0	144	61
	Modul$_{2P}$	40			12	27	81	51
	Modul$_{2I}$				12	27	130	71
	Modul$_{3P}$	70			28	17	241	139
	Modul$_{3I}$				40	16	316	160
násobička	**nerozdělený**	159	264	11	435	477	589	931
	Modul$_{0P}$	32			16	31	44	66
	Modul$_{0I}$				16	31	129	52
	Modul$_{1P}$	42			30	62	68	125
	Modul$_{1I}$				32	62	189	153
	Modul$_{2P}$	108			74	169	180	314
	Modul$_{2I}$				78	169	480	388
	Modul$_{3P}$	252			315	215	297	426
	Modul$_{3I}$				324	213	1033	672
dělička	**nerozdělený**	139	32	28	308	666	748	1277

obvod		FF	sFF	SRL	AND	XOR	LUT	MUX
	$Modul_{0P}$	32			16	31	38	66
	$Modul_{0I}$				16	31	127	49
	$Modul_{1P}$	38			88	176	200	357
	$Modul_{1I}$				93	176	539	450
	$Modul_{2P}$	77			99	225	225	414
	$Modul_{2I}$				101	225	634	517
	$Modul_{3P}$	68			105	234	285	440
	$Modul_{3I}$				106	234	709	542
odmocnina	**nerozdělený**	111	32	11	180	374	484	772
	$Modul_{0P}$	32			11	23	29	47
	$Modul_{0I}$				11	23	11	34
	$Modul_{1P}$	35			79	142	176	320
	$Modul_{1I}$				84	141	456	403
	$Modul_{2P}$	49			65	145	154	283
	$Modul_{2I}$				64	145	409	346
	$Modul_{3P}$	38			25	64	125	122
	$Modul_{3I}$				25	63	225	147

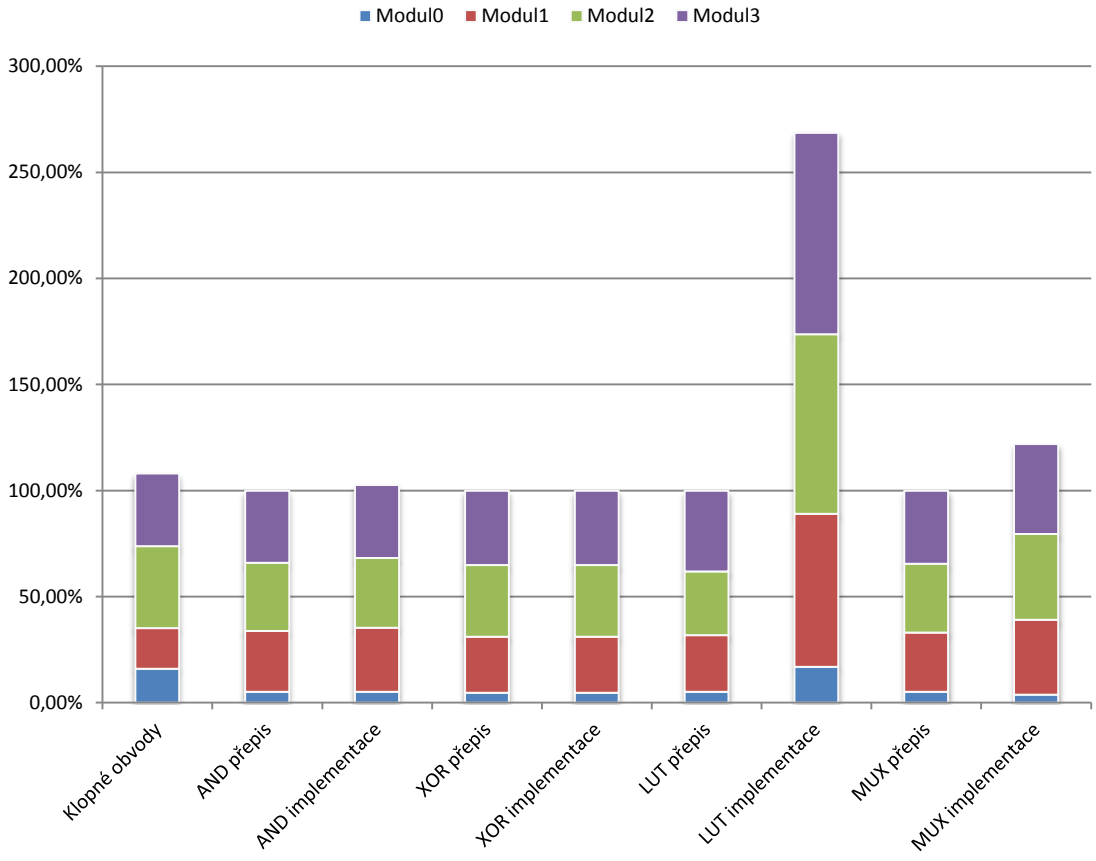
Poměry zastoupení primitiv jednotlivých částí rozděleného obvodu před a po implementaci jsou zobrazeny v následujících grafech.



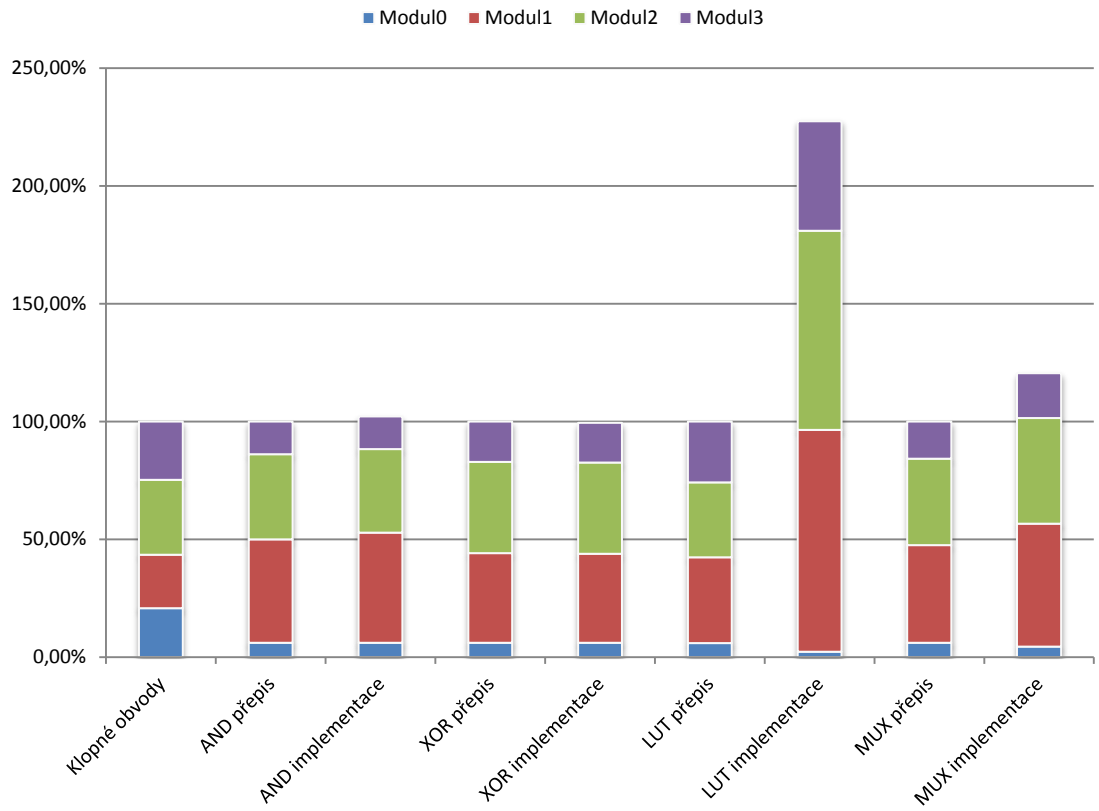
Obrázek 38: Rozdělení primitiv v pipeline u float sčítačky



Obrázek 39: Rozdělení primitiv v pipeline u float násobičky



Obrázek 40: Rozdělení primitiv v pipeline u float děličky



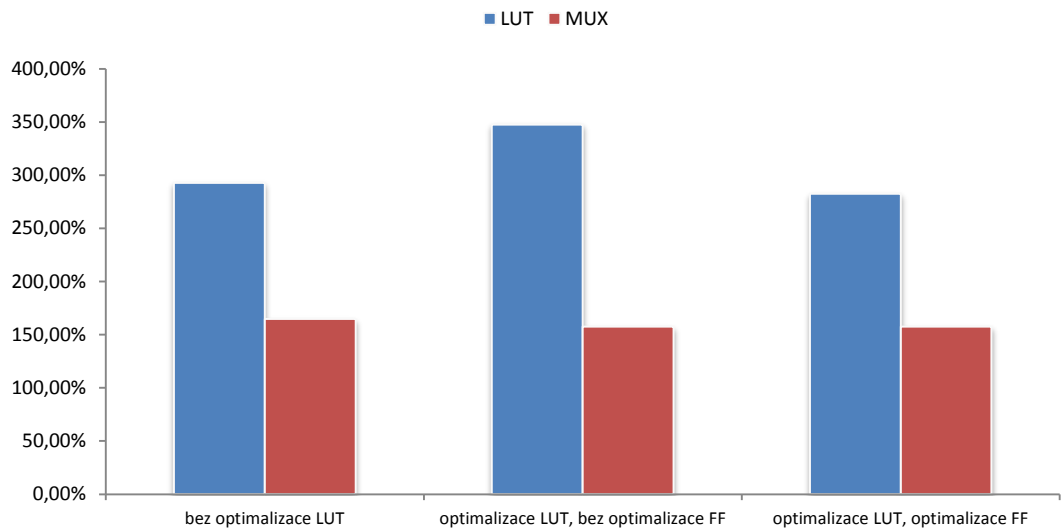
Obrázek 41: Rozdělení primitiv v pipeline u float odmocniny

Z grafů na obrázku 38 až 41 je na první pohled patrné, že po implementaci obvodů dochází k enormnímu nárůstu spotřeby zdrojů FPGA, především vyhledávacích tabulek. Zdrojem problému je s největší pravděpodobností syntézní nástroj v kombinaci se skriptem pro dělení a přepis obvodu. VHDL popisy modulů jsou složeny téměř výhradně ze standardních primitiv jako LUT4, MUXCY, XORCY apod. z knihovny UNISIM. Po syntéze dojde k mírnému navýšení počtu primitiv typu LUT4, ale především se v obvodech objeví velký počet instancí LUT1, LUT2 a LUT3, ačkoliv se v původním, de facto strukturním popisu vůbec nevyskytovaly. Tato primitiva zůstávají zachována i po implementaci a ve výsledném modulu se objevují jako instance vyhledávací tabulky (LUT4). Příčin vytváření dodatečných vyhledávacích tabulek může být několik a lze je rozdělit na dvě kategorie:

- Syntéza:
    - Instance primitiv, které nejsou typu LUT (např. prvku XORCY), implikuje vytváření budičů ve formě výstupu vyhledávací tabulky
    - Nárůst zdrojů mírně závisí na nastavení syntézy. Určitý vliv má nastavení optimalizace vložených primitiv a použití synchronních vstupů SET a

RESET klopných obvodů. Rozdíly post implementačních dat čtvrtého stupně násobičky jsou zobrazeny v grafu na obrázku 42. „Nejlepších“ výsledků je dosaženo se zapnutou optimalizací primitiv a povolením používat synchronní SET/RESET. Optimalizace klopných obvodů (v grafu označovaných jako FF) se bohužel neslučuje s možností vkládat skenovací řetězce

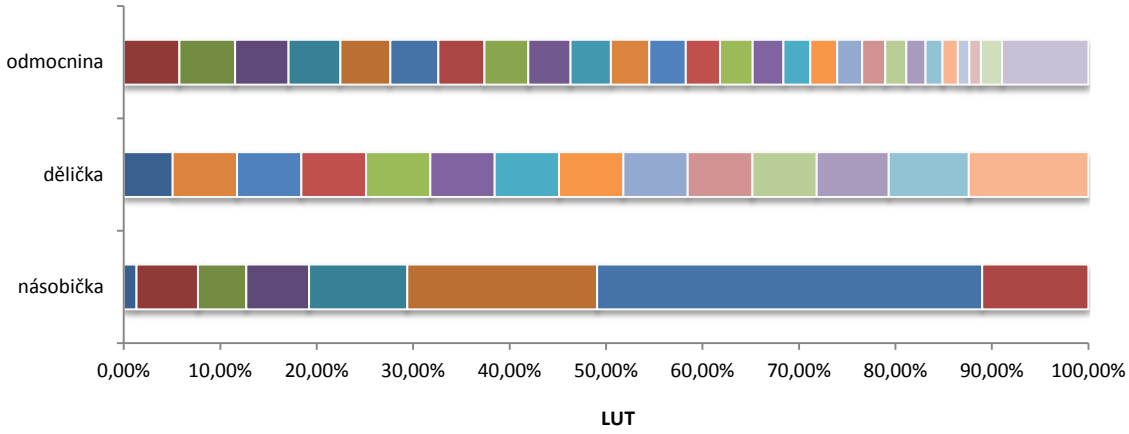
- Přepisovací skript:
  - Nejsou použity správné verze výše zmíněných primitiv. Například LUT4 je instancí vyhledávací tabulky s obecným výstupem, ale existují i verze LUT4_D s výstupem přivedeným na klopný obvod nebo LUT4_L s lokálním výstupem. Přepisovací skript mezi nimi nerozlišuje, protože by musel analyzovat vzájemné umístění prvků obvodu
  - Skriptem nejsou odstraňovány tzv. route through vyhledávací tabulky, které vznikají při použití samostatného hradla XOR a jsou v obvodu přítomny již před rozdělením. V kombinaci s prvním bodem v kategorii syntéza může docházet k jejich dvojité instanci



Obrázek 42: Vliv nastavení syntézy na velikost obvodu $modul_{31}$ násobičky

Ze statistik lze vyčíst ještě jeden zajímavý údaj. Poměr rozdělení zdrojů původního obvodu mezi rozdělené moduly není stejný, ale téměř bez výhrady jsou počáteční stupně pipeline největší. Při delších pipeline je tento nepoměr ještě patrnější, jak dokazuje graf na obrázku 33, ve kterém nejsou dokonce vidět některé stupně pipeline, protože jsou tvořeny pouze klopnými obvody. První modul rozděleného obvodu se v grafu nachází vždy napravo.

Jako podklad pro graf byly použity statistiky skriptu *split*. Graf zobrazuje obvody násobičky, děličky a odmocniny s 8, 24 a 28 stupni pipeline. Uváděny jsou pouze poměry počtů vyhledávacích tabulek - ostatní parametry jsou vynechány.



Obrázek 43: Rozložení LUT v pipeline obvodů odmocnina, dělička a násobička

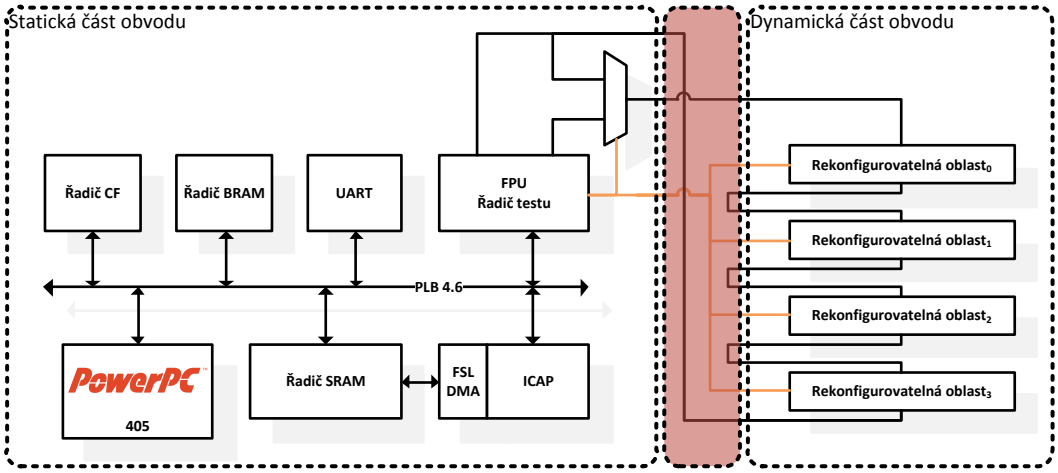
Nestejnoměrná velikost modulů může komplikovat vytváření generátorů testovacích vektorů: Jsou-li okolní moduly podstatně menší než modul testovaný může se stát, že nebude v okolních modulech dostatek vyhledávacích tabulek na vytvoření paměti ROM.

Nárůst zdrojů obvodu v kombinaci s nejmenším hradlovým polem rodiny Virtex4-FX vedl k omezení možností implementace IP jader v dostupném hardware. V reálném systému, který bude představen v další kapitole, bylo možno testovat pouze jádro sčítačky, jelikož ostatní IP jádra nebylo možno implementovat. Jedinou příčinou bylo to, že velikost jednoho modulu rozděleného obvodu byla větší než velikost rekonfigurovatelné oblasti. Doplňme, že výsledky implementace v této kapitole byly získány implementací na větším obvodu stejné řady.

4.2.2 Rekonfigurovatelný systém

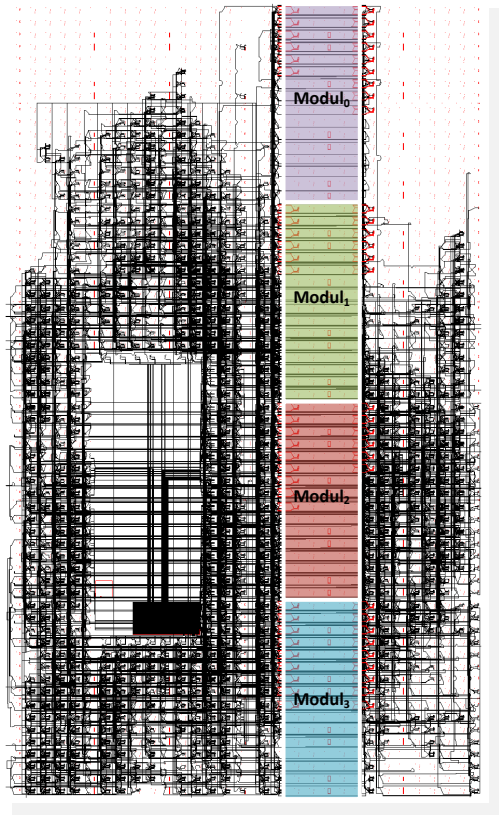
Systém (viz obrázek 44), na kterém bylo testování jader prováděno, byl sestaven podobně jako systém v předchozím experimentu. Shodně se skládal z procesorového jádra PowerPC405, sběrnice PLB, a rozličných systémových komponent jako řadiče pamětí (externích SRAM a CompactFlash karty, interní BRAM) nebo systémový časovač. Systém komunikuje s PC prostřednictvím standardního rozhraní RS232. Přístup do konfigurační paměti obvodu je realizován modifikovaným řadičem ICAP, který je připojen přímo na řadič externí paměti SRAM. Oproti původnímu

systému chybí komponenta testeru, která byla nahrazena jádrem FPU s integrovaným řadičem testu.



Obrázek 44: Blokové schéma systému

Změnilo se i fyzické rozložení obvodu. Namísto jedné rekonfigurovatelné oblasti byly vytvořeny čtyři oblasti, každá se shodnou velikost 4×16 CLB + 1×8 DSP. Oblasti byly v obvodu rozmístěny nad sebe v jednom sloupci tak, jak ilustruje obrázek 45 získaný ze software FPGA Editor.



Obrázek 45: Rozvržení obvodu. Barevně jsou označeny rekonfigurovatelné oblasti a moduly

4.2.3 Rozhraní FPU s integrovaným řadičem testu

Rozhraní FPU s integrovaným řadičem testu v systému nahrazuje komponentu testeru představenou v experimentu popsaném v kapitole 4.1. FPU umožňuje funkční i testovací přístup k IP jádru a je připojena jako slave modul na sběrnici PLB (Processor Local Bus) s možností jednoslovných datových přenosů. V reálných systémech by byla FPU napojena na vhodnější rozhraní, například na takové, které by komunikovalo přímo s procesorem nebo by umožňovalo dávkové datové přenosy. Rozhraní FPU je řízeno přes šestici uživatelských registrů uvedených v tabulce 15.

Tabulka 15: Uživatelské registry FPU

offset	Jméno registru
0x00	Stavový a řídící registr
0x04	Registr hodnot A
0x08	Registr hodnot B
0x0C	Registr výsledku
0x10	Registr vektorů
0x14	Registr posuvů/odezvy

Zápisem do libovolného registru hodnot dvou dojde k zahájení výpočtu. Výsledek lze po čtyřech taktech vyčíst z registru výsledku.

Ostatní registry ovládají hardware příslušející testu: Registr vektorů slouží k nastavení počtu testovacích vektorů. Registr posuvu v režimu zápisu nastavuje počet posuvů, který se provede při spuštění příkazu „vyčti odezvu“. V režimu čtení obsahuje registr 32bitovou hodnotu aktuálně vyčtené odezvy. Pro kompletní vyčtení odezvy, která je delší než 32 bitů je nutno povel spustit vícekrát.

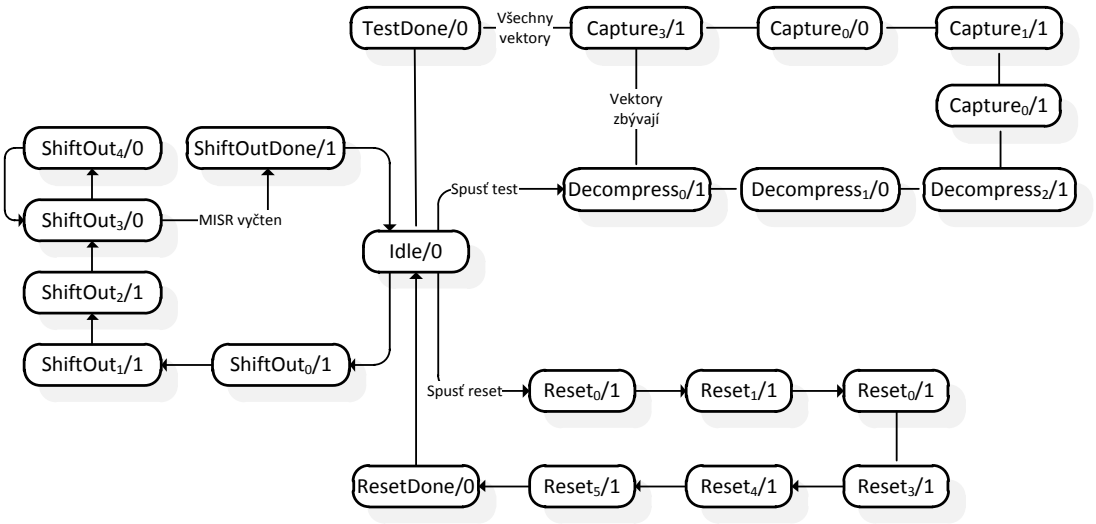
Stavovým a řídícím registrem je možno ovládat distribuovaný tester, oddělovací elementy a nastavení zpětnovazebného registru. Význam jednotlivých bitů registru je v tabulce 16.

Tabulka 16: Význam bitů stavového a řídícího registru

Bit	Význam zápis	Význam čtení
0	Reset komponenty po zapsání log. 1	Bez významu – vždy log. 0
1	Spuštění testu zápisem log. 1	Indikace stavu testu – v log. 1 indikuje běžící test. Ukončení operace je indikováno log. 0.
2	Spuštění resetu TPG a ORA	Indikace resetu – v log. 1 indikuje probíhající operaci reset. Ukončení operace je indikováno log. 0.
3	Vyčtení odezvy z ORA	Indikace stavu vyčítání – v log. 1 indikuje probíhající vyčítání. Ukončení operace je indikováno log. 0.
16 .. 16+(N-1)	Ovládání oddělovacích elementů. Každý bit ovládá jeden element. V log. 1 je povoleno šíření signálu,	Indikace stavu oddělovacích elementů. Význam viz „význam zápisu“.

Bit	Význam zápis	Význam čtení
	v log. 0 zakázáno. Parametr N odpovídá počtu oddělovacích elementů, resp. počtu rekonfigurovatelných oblastí a je limitován na 15.	
16+N	Ovládání zpětnovazebního multiplexoru. V log. 0 je testovaný obvod nastaven do funkčního režimu a vstupy prvního modulu jsou řízeny z registrů hodnot. V log. 1 je jádro nastaveno v testovacím režimu. Výstup posledního modulu řídí vstupy prvního modulu.	Indikace nastavení zpětnovazebného multiplexoru.

Řízení testu neprobíhá přímo, ale přes jednoduchý řadič *TAP_interface*, který se skládá ze stavového automatu a dvou čítačů – čítače vektorů a čítače posuvů. Přechodový graf automatu je zobrazen na obrázku 46. Automat je typu „Moore“ a distribuovaný tester ovládá řízením výstupu TMS. Hodnota výstupu TMS je zobrazena ve stavech na přechodovém grafu. Automat je navržen k řízení testu modulů, které neobsahují skenovací řetězce; začlenění skenovacích řetězců by si vyžádalo úpravu automatu v souladu s návrhem na obrázku 26.



Obrázek 46: Přechodový graf automat *TAP_interface*

K rozhraní FPU byly vytvořeny softwarové ovladače, které umožňují snadno ovládat základní funkce FPU a řadiče testu.

4.2.4 Testy

V souladu s navrženou metodou byly pro každý modul rozděleného jádra sčítačky vytvořeny deterministické testy. Byly vygenerovány čtyři sady testů, které

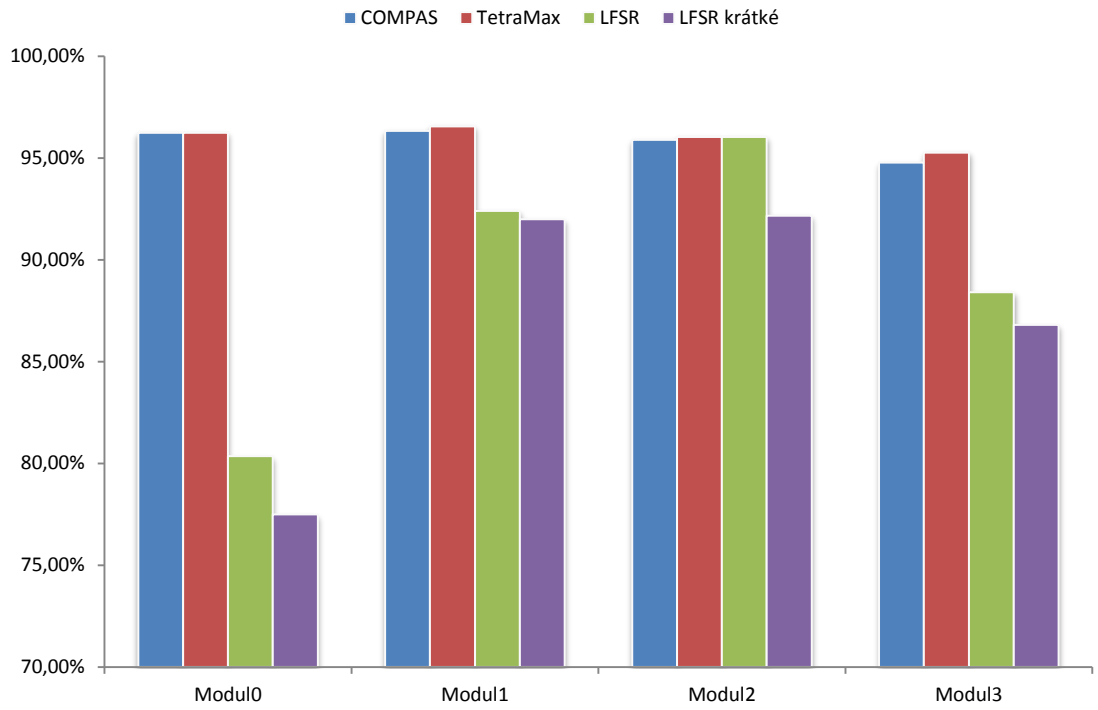
budou v textu označovány následovně: sada COMPAS bude označovat zkomprimovanou testovací sekvenci, sada TetraMax testovací sekvenci vytvořenou pomocí standardního postupu v ATPG TetraMAX, sada LFSR bude označovat vektory, které generuje posuvný registr s lineární zpětnou vazbou a jménem „LFSR krátké“ bude označena sada vektorů, která vychází ze sady LFSR, ale počet jejích vektorů je nižší – shodný s počtem vektorů sekvence COMPAS.

Testovací vektory jednotlivých sad byly nejprve podrobeny simulaci poruch na modelu obvodu v interním simulátoru TetraMax. Tabulka s výsledky obsahuje údaje o pokrytí testu, pokrytí poruch a efektivitě ATPG tak, jak byly definovány v rovnicích (1), (2) a (3) v kapitole 3.4.

Tabulka 17: Strukturní pokrytí, délka a objem testovacích sekvencí

Obvod		COMPAS	TetraMax	LFSR	LFSR krátké
$Modul_0$	**Pokrytí testu [%]**	96,25	96,25	80,35	77,49
	Pokrytí poruch [%]	90,5	90,5	75,55	72,85
	Efektivita ATPG [%]	96,48	96,48	81,53	78,83
	Počet vektorů	1835	101	1835	10001
	Velikost ROM [bit]	1835	4646	0	0
$Modul_1$	**Pokrytí testu [%]**	96,33	96,56	92,41	91,99
	Pokrytí poruch [%]	92,75	92,97	88,97	88,57
	Efektivita ATPG [%]	96,47	96,69	92,69	92,29
	Počet vektorů	312	63	312	10001
	Velikost ROM [bit]	312	2520	0	0
$Modul_2$	**Pokrytí testu [%]**	95,89	96,03	96,03	92,17
	Pokrytí poruch [%]	92,18	92,31	92,31	88,6
	Efektivita ATPG [%]	96,05	96,18	96,18	92,47
	Počet vektorů	152	45	152	10001
	Velikost ROM [bit]	152	3150	0	0
$Modul_3$	**Pokrytí testu [%]**	94,78	95,27	88,41	86,8
	Pokrytí poruch [%]	89,99	90,45	83,94	82,41
	Efektivita ATPG [%]	95,05	95,51	88,99	87,46
	Počet vektorů	1993	309	1993	10001
	Velikost ROM [bit]	1993	19776	0	0

Graf pokrytí testu pro každý modul a každou sadu testovacích vektorů je vynesen na obrázku 47.



Obrázek 47: Pokrytí testu (strukturních poruch) modulů rozdělené sčítačky

V další fázi experimentu byly testovací sady ověřovány na injektoru poruch. Pro každý modul byly vytvořeny částečné bitstreamy ve variantách funkční jádro, generátory testovacích vektorů (COMPAS a LFSR) a analyzátory odezev (skenovací řetězec a MISR).

Testování probíhalo následovně. Částečné bistreamy a inicializační soubor ACE byly uloženy na paměťové kartě, ze které bylo zařízení inicializováno. Konfigurační soubory byly při startu systému nahrány do systémové paměti SRAM, odkud byly dále zpracovávány. Do funkčních částečných bitstreamů byly injektovány poruchy, okolí testovaných modulů bylo rekonfigurováno do testovacího režimu, byl spuštěn test a analyzována sejmutá odezva. Pokud byl výsledek testu negativní, pak byla informace o pozici poruchy v částečném bitstreamu předána přes sériový port počítači, který výsledky zpracoval. Před vkládáním poruch byl proveden test na každém „nepoškozeném“ modulu za účelem získání korektní odezvy. Při provádění experimentu byly používány pouze ORA ve variantě MISR. Analyzátory ve variantě skenovací řetězec byly použity pouze pro doladění funkce systému.

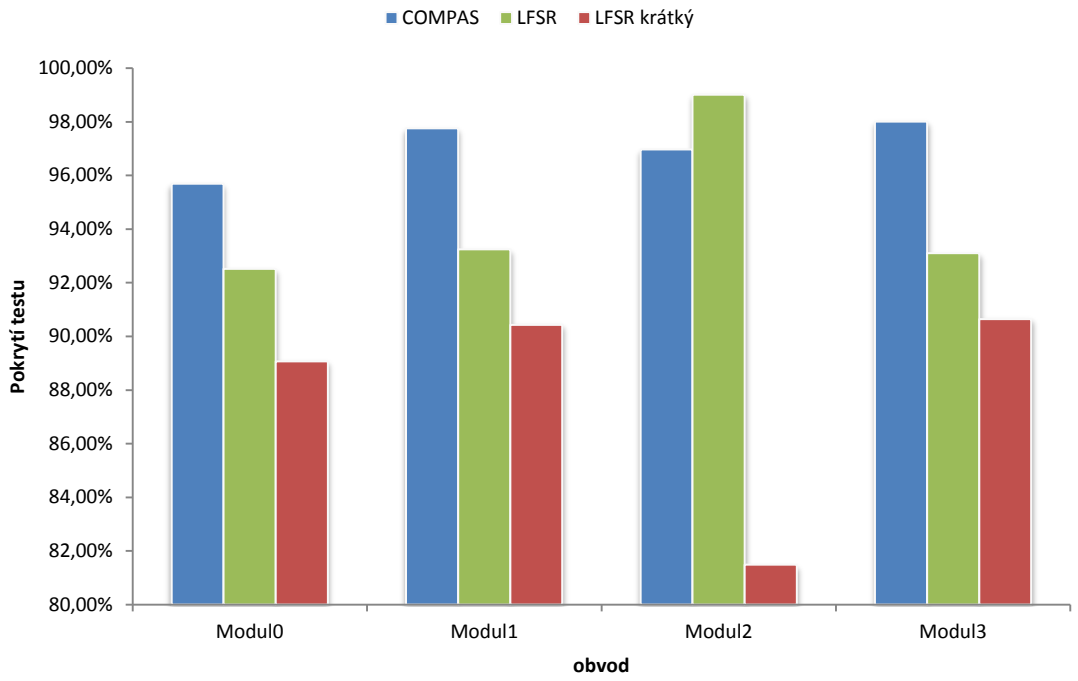
Ověřovány byly tři testovací sady – sada COMPAS, LFSR a LFSR krátké. Zpracované výsledky testu jsou zaznamenány v tabulce 18.

Tabulka 18: Pokrytí poruch konfigurační paměti

		$Modul_0$	$Modul_1$	$Modul_2$	$Modul_3$
Všechny testy	**Všechny poruchy**	5803	8247	8503	16521
	Náhodné poruchy	61	88	75	140
	Poruchy testu	697	766	1526	1553
	Pravidelné poruchy	5045	7393	6902	14828
COMPAS	**Všechny poruchy**	5608	8146	8308	16311
	Náhodné poruchy	55	84	63	118
	Poruchy testu	508	669	1343	1365
LFSR krátký	**Všechny poruchy**	5199	7527	6982	15078
	Náhodné poruchy	30	69	53	102
	Poruchy testu	124	65	27	148
LFSR	**Všechny poruchy**	5405	7769	8494	15489
	Náhodné poruchy	36	79	75	107
	Poruchy testu	324	297	1517	554
Pokrytí poruch	**COMPAS**	95,69%	97,76%	96,97%	98,01%
	LFSR krátký	89,07%	90,43%	81,49%	90,65%
	LFSR	92,52%	93,25%	99,01%	93,11%
Maximální pokrytí poruch		98,95%	98,93%	99,12%	99,15%
Efektivita pokrytí	**COMPAS**	96,71%	98,81%	97,83%	98,85%
	LFSR krátký	90,02%	91,41%	82,21%	91,42%
	LFSR	93,50%	94,25%	99,89%	93,90%

Kompletní testy byly provedeny vícenásobně a na základě výsledků byly vytvořeny tři kategorie poruch: „pravidelné poruchy“, „náhodné poruchy“ a „poruchy testu“. Mezi „pravidelné poruchy“ byly zařazeny poruchy, které byly detekovány vždy všemi testy a lze je považovat za poruchy detekované. Poruchy, které byly detekovány nepravidelně alespoň jedním testem, byly zařazeny do kategorie „náhodné poruchy“ (např. COMPAS 9(detekovaných)/10(provedených pokusů), LFSR 10/10, LFSR krátké 10/10) a jsou považovány za poruchy nedetekované. Do poslední kategorie patří poruchy, které byly detekovány při všech opakováních experimentu, ale ne u všech testů (např. COMPAS 10/10, LFSR 10/10, LFSR krátké 0/10). Poslední kategorie poruch byla zařazena do kategorie detekované.

Pokrytí testu je počítáno v souladu s rovnicí (2). Maximální pokrytí poruch je vypočítáno z parametrů vzniklých průnikem všech testů. Efektivita pokrytí je počítána jako poměr maximálního pokrytí poruch a pokrytí poruch každého testu. Graf pokrytí poruch je vynesen na obrázku 48.



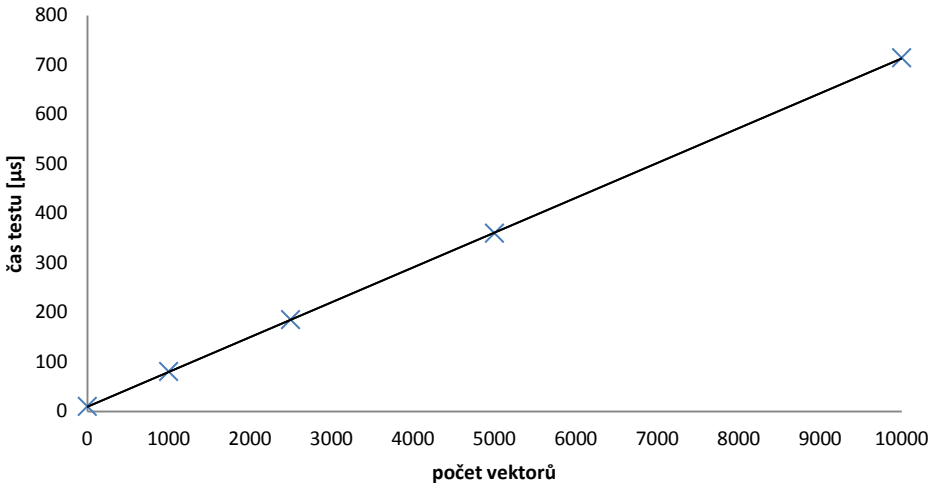
Obrázek 48: Pokrytí poruch (konfigurační paměti) modulů rozdělené sčítačky

Společně s měřením kvality testovacích sad byla provedena i měření dostupnosti FPU. Měřena byla časová závislost testu na počtu aplikovaných testovacích vektorům, doba rekonfigurace okolních modulů a průměrná doba kompletního testu všech obvodů při použití TPG COMPAS. Údaje byly získány s pomocí systémového časovače s časovým rozlišením 10ns.

V tabulce 19 a grafu na obrázku 49 je zanesena naměřená závislost doby samotného testu (generování vektorů + vyčtení odezvy) na počtu testovacích vektorů. Ačkoliv graf zobrazuje lineární závislost, může se doba generování vektorů skokově zvyšovat, pokud bude jako TPG použit COMPAS dekompresor a kapacita jeho interní paměti nebude dostatečná. V případě generátoru založeného na LFSR bude závislost lineární vždy.

Tabulka 19: Časová závislost testu na počtu vektorů

Počet vektorů	1	1000	2500	5000	10000
Čas testu [µs]	10	80	185	360	714



Obrázek 49: Graf časové závislosti testu na počtu testovacích vektorů

Průměrná doba rekonfigurace – výměny obou okolních modulů – trvala *200µs*, kompletní test FPU pak v průměru *1943µs* a jeho jednotlivé fáze, tj. testy $modulu_0$ až $modulu_3$ trvaly *540*, *432*, *421* a *550µs* respektive.

4.2.5 Režie testu

Aplikace prezentované metody vnáší do testovaného obvodu určitou míru nežádoucí režie. Lze předpokládat, že majoritní podíl na nárůstu spotřebovaných zdrojů je způsoben začleněním částečné rekonfigurace. Její vliv je dvojí. Za prvé se projeví vložení oddělovacích elementů (viz pravidelně se opakující červené prvky na vertikálních hranách modulů na obrázku 45) a za druhé se projeví omezení daná umístěním a geometrií rekonfigurovatelných oblastí (způsobená mechanismem přístupu do konfigurační paměti hradlového pole).

Pro vyčíslení míry režie bylo provedeno srovnávací měření spotřeby zdrojů tří odlišných systémů – systému bez rekonfigurace, systému se standardní rekonfigurací a systému s modifikovanou rekonfigurací, který je v této kapitole prezentován.

Všechny systémy sdílí ve statické části obvodu prvky PowerPC 405, BRAM 4KB, časovač, UART, SystemACE a řadič SRAM. Systémy se liší v modulech FPU a v komponentách zajišťujících rekonfiguraci.

Systém bez rekonfigurace neobsahuje v komponentě FPU řadič testu, v ostatních systémech je řadič do FPU vložen. Standardní rekonfigurovatelný systém obsahuje komponenty „xps_hwicap“ a „xps_centra_dma“, které jsou v modifikovaném systému nahrazeny vlastní komponentou „xcl_icap“ a dodatečným portem na řadiči SRAM. Ve standardním rekonfigurovatelném systému je z důvodu porovnání IP jádro

vloženo pouze do jedné oblasti rekonfigurace. Statistiky všech variant systému jsou uvedeny v tabulce 20.

Tabulka 20: Režie rekonfigurace v různých variantách systému

	varianta systému					
	bez rekonfigurace		standardní rekonfigurace		modifikovaná rekonfigurace	
	počet SLICE	% FPGA	počet SLICE	% FPGA	počet SLICE	% FPGA
statická část	2107	38,51	2383	43,55	2479	45,30
oddělovací elementy	0	0	48	0,88	240	4,39
oblast rekonfigurace	0	0	512	9,36	1024	18,71
IP jádro	339	6,20	339	6,20	457	8,35
režie rekonfigurace	0	0	197	3,60	805	14,71

Výsledky měření potvrzují, že režie na statické části obvodu je srovnatelná s řešením Xilinx, zatímco na části dynamické přesahuje tolerovatelné meze. Lze předpokládat, že se režie navyšuje tím víc, čím více rekonfigurovatelných oblastí je v systému vytvořeno. Parametrem, který nejzřetelněji vypovídá o míře nadbytečných zdrojů je *režie rekonfigurace (4)* uvedená v posledním řádku tabulky 20:

$$režie\ rekonfigurace = \frac{1}{2}BM + PR - IP \qquad (4)$$

V (4) *BM* odpovídá oddělovacím elementům, *PR* oblast rekonfigurace a *IP* označuje velikost původního nerozděleného IP jádra.

V systému, který lze chápat jako prototypové nasazení prezentované metody do reálných podmínek, dosahuje režie více než dvojnásobku spotřeby zdrojů původního IP jádra.

Srovnání se systémem bez rekonfigurace a se standardní rekonfigurací je však neobjektivní, uvážíme-li, že oba jmenované systémy nelze testovat a první systém není ani schopen opravy poruch konfigurační paměti pomocí rekonfigurace. Pro korekci porovnání režií je třeba srovnat systémy, které testování a opravy umožňují.

Do testovaného IP jádra byly proto vloženy skenovací řetězce a k celkovému počtu spotřebovaných zdrojů byl připočten odhad velikosti staticky implementovaného dekompresoru COMPAS. Výsledné hodnoty jsou vyneseny v tabulce 21, přičemž výpočet odhadu byl realizován dle vztahu (5):

$$odhad = \frac{(\frac{FF - PO}{16} + PI)_1 + \log_2(FF - PO + PI)_2 + PO}{2} \quad (5)$$

V (5) představují *PO*, *PI* a *FF* primární výstupy, vstupy a počet klopných obvodů v testovaném jádru respektive, přičemž do parametru *FF* jsou započteny i klopné obvody na primárních výstupech.

Myšlenka výpočtu odhadu je následující. Dekompresor COMPAS lze realizovat posuvným registrem délky *FF – PO + PI*, přičemž část řetězce odpovídající délce skenovacího řetězce *FF - PO* lze úsporně vytvořit s pomocí LUT v režimu SRL. Druhý součtový člen výrazu symbolizuje zdroje potřebné k implementaci řídícího čítače a poslední člen má význam zdrojů potřebných pro implementaci MISR. Ostatní prostředky jsou zanedbány. Pro vyjádření odhadu v SLICE je celý výraz dělen dvěma.

Tabulka 21: korekce režie testu pro systém se standardní rekonfigurací

PI	PO	oddělovací elementy	klopné obvody	IP jádro [SLICE]	odhad COMPAS + MISR [SLICE]	oblast rekonfigurace	režie [SLICE]	režie [%]
64	32	48	188	590	58	768	487	8,90

Vložení skenovacích řetězců zvětšilo IP jádro téměř dvojnásobně, kvůli čemuž musela být o třetinu zvětšena i oblast rekonfigurace. Její zvětšení a přidání statického TPG a ORA se negativně projevilo na celkové režii, která se oproti původnímu systému více než zdvojnásobila, ale je stále zhruba poloviční oproti systému, na kterém byla aplikována představovaná metoda testování.

Nižší režie sytému se skenovacími řetězci je vykoupena zvýšením nedostupnost FPU během testu: doba dekomprese jednoho testovacího vektoru se kvůli sériovému přístupu k testovanému obvodu zvýší minimálně (*FF – PO + PI*) krát.

Režie se neprojevuje pouze na zvýšení spotřeby zdrojů hradlového pole, ale i na časovacích charakteristikách obvodu. Statická část obvodu je u libovolného systému z představené trojice schopna pracovat na žádané frekvenci 100MHz. U obou rekonfigurovatelných systémů se dodatečná logika projevuje nežádoucím zpožděním na kritických cestách testovaného jádra. Statistiky časové analýzy získané programem *trace* z postimplementačních dat jsou vyneseny v tabulce 22.

Kritická cesta v systému se skenovacími řetězci vede od vstupního oddělovacího elementu a končí na registru první části pipeline IP jádra. Nárůst kritické cesty je oproti původnímu obvodu dán z menší části zpožděním na oddělovacích

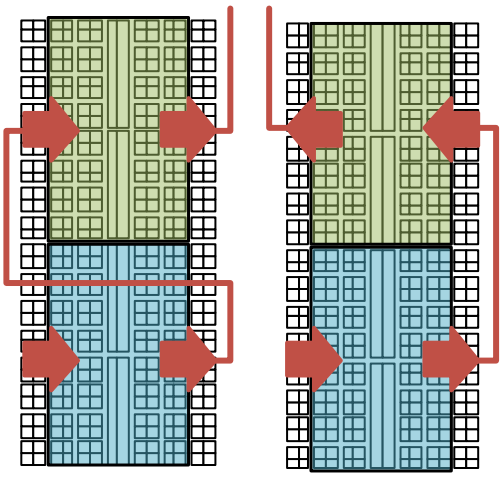
elementech a z větší části zpožděním vzniklým na kombinační logice multiplexorů skenovacích buněk.

Tabulka 22: Kritické cesty

kritická cesta					
standardní rekonfigurace		**standardní rekonfigurace + scan**		**modifikovaná rekonfigurace**	
prvek	**zpoždění [ns]**	**prvek**	**zpoždění [ns]**	**prvek**	**zpoždění [ns]**
vstupní BM	3,829	vstupní BM	3,139	BM_0;mux;BM_3	10,003
IP jádro	7,764	IP jádro	15,220	IP jádro	13,105
celkem	11,593	celkem	18,359	celkem	23,108

Kritická cesta systému s modifikovanou rekonfigurací se nachází mezi výstupem řadiče testu FPU, který ovládá oddělovací elementy první části rozděleného modulu, prochází zpětnovazebním multiplexorem a končí ve výstupních registrech prvního modulu. Zpoždění v IP jádru je pravděpodobně způsobeno přepisem obvodu a jeho druhou syntézou. Větší část celkového zpoždění je způsobena propojením mezi oddělovacími elementy a neregistrovaným multiplexorem: signál v propojovací matici doslova procestuje FPGA zdola nahoru a zpět.

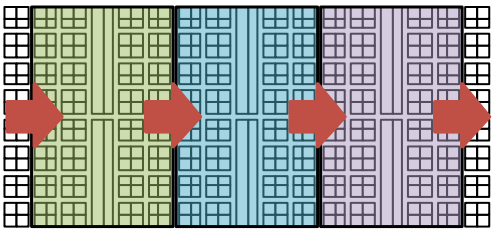
V rekonfigurovatelném systému jsou užívány výlučně oddělovací elementy vedoucí signál zleva doprava. Jelikož oblasti částečné rekonfigurace zabírají celý sloupec FPGA musí být signály spojující jednotlivé moduly vedeny pomocí dlouhých propojů z pravé části FPGA do části levé. Konfiguraci užitou v modifikovaném systému ilustruje levá část obrázku 50. Pravá část obrázku ilustruje situaci, kdy by byl u každého lichého modulu vyměněn směr šíření signálu. Experiment, který byl s takto pozměněným návrhem proveden, ukázal, že orientace oddělovacích elementů má na zpoždění na kritické cestě nulový vliv.



Obrázek 50: Modifikace umístění oddělovacích elementů 1

Zpoždění na oddělovacích elementech lze snížit s pomocí synchronních oddělovacích elementů, ovšem za cenu navýšení latence výpočtu. Další varianta, která je vyobrazena na obrázku 51, je založena na sdílení oddělovacích elementů mezi sousedními oblastmi rekonfigurace. Tato varianta neumožňuje využít oddělovací elementy s blokováním výstupu, jelikož signál, který blokování řídí, musí být veden statickou částí obvodu

I kdyby se však podařilo zpoždění na oddělovacích elementech odstranit, stále bude výkon IP jádra v obou rekonfigurovatelných systémech přibližně poloviční oproti jeho staticky implementované variantě.



Obrázek 51: Modifikace umístění oddělovacích elementů 2

5 Závěr

Prezentovaná metoda aplikačně závislého testování FPGA obvodů se skládá z celé řady dílčích kroků, které dohromady vytvářejí velmi komplexní návrhový postup. V následujícím textu budou dílčí kroky shrnuty a bude analyzován jejich vliv na kvalitu aplikace metody.

Metoda pracuje s částmi testovaného obvodu, které jsou v textu označovány jako moduly a vznikají procesem dělením obvodu dodaného ve formě strukturního netlistu. Nástroje, které byly pro potřeby práce vyvinuty, buď umožňují dělit sekvenční obvody, které obsahují registry bez zpětných vazeb, anebo umožňují vkládat skenovací řetězce do obvodů, které obsahují meziregistrové zpětné vazby. Kombinace obou přístupů prozatím realizována není.

Samotné dělení obvodu je prováděno korektně – chování opětovně složeného obvodu odpovídá původnímu návrhu nejen dle behaviorálních simulací, ale i při nasazení na hradlovém poli. Dělení obvodů více než dostačuje potřebám práce, problematické je až další zpracování rozdělených modulů.

Při opětovné syntéze a implementaci dochází k podstatnému nárůstu spotřeby zdrojů hradlového pole, jehož příčiny byly diskutovány v 4.2.1. Režie, která dvojí implementací vzniká, silně limituje užití metody v menších hradlových polích; vznikají absurdní situace, kdy je část rozděleného obvodu větší než původní obvod, ačkoliv při dělení dochází k redukci obvodových prvků. Ve zkvalitnění způsobu dělení obvodů je možno hledat jeden ze směrů vylepšení práce, přičemž důraz by měl být kladen především na kvalitnější klasifikaci prvků užitých v nerozděleném obvodu a na analýzu jejich vzájemného rozmístění.

Implementované moduly jsou v dalším kroku metody podrobeny automatickému zpracování v nástrojích ATPG. Za tímto účelem jsou moduly přepsány ze strukturního popisu, který sestává ze standardních FPGA primitiv do popisu, který je tvořen primitivy ASIC.

Kvůli limitaci dané uzavřenou architekturou FPGA obvodů je vytvářen pouze přibližný strukturní model FPGA a i zvolený model poruch (trvalá 1/0) je limitován strukturní závislostí. Z analýzy experimentů se však ukazuje, že původní teze, že i přibližný model může být pro tvorbu kvalitních testovacích vektorů dostatečný, byla správná.

Model obvodu byl vytvářen s vědomím, že konfigurační paměť hradlového pole je nedílnou součástí jeho struktury. Jelikož výsledky pokrytí poruch strukturního testu korelují s výsledky, které byly získány na emulátoru poruch konfigurační paměti, lze model prohlásit za celkem věrohodný a lze předpokládat, že detekuje i strukturní poruchy, přestože to není možné dokázat.

Reprezentativnost výsledků emulace poruch konfigurační paměti, o kterou se předchozí myšlenku opírá, podporují výsledky nezávislých výzkumů (např. [36]) a údaje výrobce, které udávají poměr konfiguračních bitů ovlivňujících implementovaný obvod. Tento poměr se shoduje s výsledky, které byly získány na základě vlastních experimentů. Vytvořené testy detekují přesně ty poruchy konfigurační paměti, které mají co dočinění s funkcí obvodu, což lze považovat za úspěšné naplnění principu aplikačně závislého testování.

Vytvořené nástroje umožňují generování popisů pro celou řadu ATPG, ačkoliv se z výsledků experimentů (viz 3.4) ukazuje, že komerční ATPG nabízejí o něco vyšší výkon než jejich nekomerční představitelé. S drobnými úpravami by bylo možné přenést metodu na jinou architekturu hradlových polí, užít odlišného modelu poruch či využít ATPG, která nejsou založena na vytváření citlivých cest. Prvotní pokusy s nástroji prezentovanými v [12] již proběhly.

V rámci práce byly navrženy a úspěšně ověřeny různé implementace generátorů testovacích vektorů a analyzátorů testovacích odezev společně s nadřazeným řadičem testu. Experimentálně byl vytvořen systém na programovatelném čipu, který umožňuje pomocí navržené metody testovat běžná IP jádra. Nejen pro účely experimentálního systému byl zkonstruován výkonný řadič přístupu do konfigurační paměti hradlového pole, jež nabízí několikanásobně vyšší přenosovou rychlost, než řadič standardně dodávaný výrobcem. Experimenty provedené na prototypovém systému dokazují realizovatelnost a funkčnost navržené metody, ale poukazují i na některé nedostatky.

Je to především režie spojená s nasazením částečné dynamické rekonfigurace, která se nepříznivě projevuje na celkové využitelnosti metody v praxi. Aplikací částečné rekonfigurace vznikají požadavky na dodatečné zdroje obvodu a zhoršuje se jeho celkový výkon. Omezení režie není zcela v kompetenci návrháře systému – je značná, i když je dodržena řada pravidel, které ji redukují (viz 3.7.3). Využití novějších architektur FPGA redukci režie neposkytne, naopak ji paradoxně zvýší. Je nutno si uvědomit, že režií způsobenou částečnou dynamickou rekonfigurací netrpí pouze

prezentovaná metoda, ale i jiné metody, jež jsou na částečné rekonfiguraci založeny, ať už účel rekonfigurace souvisí s testováním, anebo ne.

Závěrem lze konstatovat, že veškeré cíle zadané v úvodu kapitoly 3 se podařilo splnit. Byla vytvořena metoda aplikačně závislého testování FPGA obvodu, jež kombinuje výhody moderních programovatelných obvodů s nástroji, postupy a technikami BIST pro obvody ASIC, které byly pro potřeby práce upraveny. Nástroje a postupy, které byly pro metodu vytvořeny, jsou opakovatelně a univerzálně použitelné.

Bylo by však naivní konstatovat, že metoda, tak jak je prezentována, je úplná a práce na ní může ustat. Práce může být rozšířena a vylepšena v celé řadě směrů. Lze zkoumat využití moderních metod vytváření testovacích vektorů, aplikaci jemnozrnné rekonfigurace pro lepší kontrolovatelnost a pozorovatelnost testovatelného obvodu, vytváření strukturně podrobnějších modelů FPGA nebo možnosti jiných modelů poruch.

5.1 Přínos navržené metody

Metoda aplikačně závislého testování FPGA obvodů, která byla představená v předcházejícím textu, nabízí inovativní způsob testování programovatelných obvodů. Testování pomocí metody stojí na několika pilířích, z nichž nejvýznamnější jsou částečná dynamická rekonfigurace a užití zkomprimovaných deterministických testovacích vektorů.

Částečná dynamická rekonfigurace je v metodě využita k několika nezávislým činnostem: umožňuje přístup k testovanému obvodu, slouží jako transportní kanál přivádějící testovací vzorky, umožňuje provádění pravidelných oprav testovaného modulu na úrovni konfigurační paměti a dovoluje omezenou míru oprav implementovaného obvodu na strukturní úrovni. Metoda zásadně zvyšuje dostupnost a řiditelnost testovaného obvodu bez omezení činnosti ostatních částí obvodu, které testovány nejsou; je umožněno paralelní nezávislé testování několika obvodových celků v rámci jednoho FPGA. Metoda užitím částečné rekonfigurace pro test účelně navyšuje funkční hustotu programovatelného obvodu.

Zkomprimované deterministické testovací vektory, přinášejí nesporné výhody v podobě zkrácení doby testu, navýšení dostupnosti zařízení a snížení počtu nutných rekonfigurací; využití zkomprimovaných testovacích sekvencí poskytuje nejlepší poměr kvality testu vůči době testu, což dokazují provedené experimenty. Proces vytváření vektorů je uživatelsky ovlivnitelný a není omezen výběrem softwarového vybavení

v podobě ATPG. Test je možno cílit na širokou paletu modelů poruch nebo lze různé testy kombinovat.

Vlastní publikace

[1] Rozkovec, M.: Přechod DyRESPIN architektury na platformu Xilinx Virtex-4, Proc. Of PAD 2007, Září 2007, Srní, ISBN 978-80-7043-605-9

[2] Rozkovec, M.: Koncept rekonfigurovatelné testovací architektury pro FPGA obvody. Proc. Of PAD 2008, Září 2008, Hejnice, Czech, pp. 37-44, ISBN 978-80-7372-378-1

[3] Rozkovec, M.: Implementation of Dynamically Reconfigurable Test Architecture for FPGA Circuits. Proc. of DDECS 2008, April 2008, Bratislava, Slovakia, pp.182-185, ISBN 978-1-4244-2276-0

[4] Rozkovec, M.; Novák, O.: Structural test of programmed FPGA circuits, Proceedings of the 2009 IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2009, art. no. 5012114, pp. 136-139

[5] Rozkovec, M.; Jeníček, J.; Novák, O.: "Application dependent FPGA testing method using compressed deterministic test vectors," On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International, pp.192-193, 5-7 July 2010

[6] Rozkovec, M.; Jeníček, J.; Novák, O.: "Application Dependent FPGA Testing Method," Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on, pp. 525-530, 1-3 Sept. 2010

[7] Jenicek, J.; Rozkovec, M.; Novak, O.: "Test vector overlapping based compression tool for narrow test access mechanism," Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2011 IEEE 14th International Symposium on, vol., no., pp.387-392, 13-15 April 2011

Použitá literatura

[8] Abramovici, M.; Stroud, C. E.; Hamilton, C.; Wijesuriya, S.; Verma,V.: Using roving STARs for on-line testing and diagnosis of FPGAs in fault-tolerant applications, IEEE International Test Conference (TC), pp. 973-982

[9] Abramovici, M.; Stroud, C.E.: BIST-based test and diagnosis of FPGA logic blocks, IEEE Transactions on Very Large Scale Integration Systems 9, pp. 159-172

[10] Alfke, P., Xilinx Inc.: Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators, online [2011-07-08]

[11] Altera Corp.: SEU mitigation in Stratix III devices, Stratix III Device Handbook [online]

[12] Balcárek, J.; Fišer, P.; Schmidt, J.; "Test Patterns Compression Technique Based on a Dedicated SAT-Based ATPG," Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on, pp. 805-808, 1-3 Sept. 2010

[13] Becker, T.; Luk, W.; Cheung, P.Y.K.; , "Enhancing Relocatability of Partial Bitstreams for Run-Time Reconfiguration," Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on, pp. 35-44, 23-25 April 2007

[14] Carmichael, C.,Tseng C. W., Xilinx Inc: “Correcting Single-Event Upsets in Virtex-4Platform FPGA Configuration Memory”, online [2011-07-08]

[15] Crouch, A. L.: Design-for-test for digital IC's and embedded core systems, Prentice Hall, 1999, ISBN: 0-13-084827-1

[16] Chapman, K., Jones, L.: SEU strategies for Virtex-5 devices, Xilinx Inc., XAPP864, [online]

[17] Dorsch, R. and Wunderlich, H-J:Reusing Scan Chains for Test Pattern Decompression.Proc. IEEE ETW, 2001, pp.24-32

[18] Fujiwara, H., Shimono, T.:On The Acceleration of Test Generation Algorithms, IEEE Transactions on Computers C-32 (12), pp. 1137-1144

[19] Glaser, U., Vierhaus, H.T.: MILEF: an efficient approach to mixed level automatic test pattern generation. Proc. Of the Conference on European Design Automation, Los Alamitos, CA, pp. 318-321

[20] Guccione, S., Levi, P., Sundararajan, P.: ”JBits: A Java based Interface for Reconfigurable Computing”, 2nd Annual Militarz and Aerospace Applications of Programmable Devices and Technologies Conference, 1999

[21] Heron, O.; Arnaout, T.; Wunderlich, H.-J.; , "On the reliability evaluation of SRAM-based FPGA designs," Field Programmable Logic and Applications, 2005. International Conference on, vol., no., pp. 403- 408, 24-26 Aug. 2005, doi: 10.1109/FPL.2005.1515755

[22] Hlavička, J.: Diagnostika a spolehlivost, Skripta ČVUT, 1998, ISBN 80-01-01846-6

[23] IEEE Computer Society. IEEE Standard Testability Method for Embedded Core-based Integrated Circuits -IEEE Std 1500-2005. IEEE, New York, 2005.

[24] „IEEE Standard Test Access Port and Boundary - Scan Architecture," IEEE Std 1149.1-1990 , vol., no., pp.0_1, 1990

[25] Jacobs, A., George, A.D., Cieslewski, G.: Reconfigurable fault tolerance: A framework for environmentally adaptive fault mitigation in space, FPL 09: 19th International Conference on Field Programmable Logic and Applications, art. no. 5272313, pp. 199-204

[26] Jeníček, J.: Komprese testovacích dat založená na překrývání vzorků. Disertační práce. TU Liberec 2008.

[27] Kafka, L.; , "Analysis of Applicability of Partial Runtime Reconfiguration in Fault Emulator in Xilinx FPGAs," Design and Diagnostics of Electronic Circuits and Systems, 2008. DDECS 2008. 11th IEEE Workshop on , vol., no., pp.1-4, 16-18 April 2008

[28] Kafka, L., Novak, O.: FPGA based fault simulator, Proceeding of the 2006 IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems.2006, pp. 274-278.

[29] Kastensmidt, F.L., Carro, L., Reis, R.: Fault-Tolerance Techniques for SRAM-based FPGAs, Springer, 2006, ISBN-10 0-387-31068-1

[30] Krishna, C.V.; Touba, N.A.: Reducing Test Data Volume Using LFSR Reseeding with Seed Compression, Proc. Of VLSI Test Symp., 2001.

[31] Lee, H.K., Ha D.S.: Atalanta: an Efficient ATPG for Combinational Circuits, Technical Report, 93-12, Dep't of Electrical Eng., Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1993

[32] Lesea, A., Drimer, S., Fabula, J.J., Carmichael, C., Alfke, P.: The rosetta experiment: Atmospheric soft error rate testing in differing technology FPGAs, IEEE Transactions on Device and Materials Reliability 5 (3), pp. 317-328

[33] Ma, S., Shaik, I., Fetherson,R.S.: Comparison of bridging fault simulation methods, IEEE International Test Conference (TC), pp. 587-595

[34] Mader, Z.: Diagnostický systém jádrově založených SoC obvodů s nízkými nároky na paměť. Disertační práce. TU Liberec 2008

[35] Novák, O., Gramatová, E., Ubar, R. and coll.: Handbook of testing electronic systems. Nakladatelství ČVUT, srpen 2005, 395 stran, ISBN 80-01-03318-X

[36] Pratt, B.H., Wirthlin, M.J., Caffrey, M., Graham, P., Morgan, K.: Noise impact of single-event upsets on an FPGA-based digital filter, FPL 09: 19th International Conference on Field Programmable Logic and Applications, art. no. 5272554, pp. 38-43

[37] Renovell, M., Faure, P., Portal. J.M., Figueras, J., Zorian, Y.: IS-FPGA : a new symmetric FPGA architecture with implicit scan, International Test Conference, 2001. Proceedings, Page(s):924-931

[38] Renovell, M., Figueras, J., Zorian, Y.: Testing the interconnect of RAM-based FPGAs, IEEE Design and Test of Computers 15 , pp. 45-50

[39] Renovell, M., Portal, J.M., Faure, P., Figueras, J., Zorian, Y.: Some Experiments in Test Pattern Generation for FPGA-Implemented Combinational Circuits, IEEE Brazilian Symp. Integrated Circuit Design SBCCI'OO, pp. 3-8

[40] Straka, M.; Kastil, J.; Kotasek, Z.; , "Fault Tolerant Structure for SRAM-Based FPGA via Partial Dynamic Reconfiguration," Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on , vol., no., pp.365-372, 1-3 Sept. 2010

[41] Stroud,C.:A Designer's Guide to Built-In Self-Test, Kluwer Academic Publishers, 2002, ISBN 1-4020-7050-0

[42] Synopsys: TetraMAX® ATPG User Guide, Version D-2010.03, March 2010

[43] Tahoori, M.B, McCluskey, E.J., Renovell, M., Faure, P.: A Multi-Configuration Strategy for an Application Dependent Testing of FPGAs VLSI Test Symposium, 2004. Proceedings. 22nd IEEE, Page(s):154 – 159

[44] Tan, C.M., Roy, A.: Electromigration in ULSI interconnects, Materials Science and Engineering R: Reports 58 (1-2), pp. 1-75.

[45] TestKompress, [online], URL:http://www.mentor.com/products/silicon-yield/products/testkompress/

[46] TetraMAX, [online], URL: www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/TetraMAXATPG.aspx

[47] Tille, D., Drechsler, R.: A Fast Untestability Proof for SAT-based ATPG, Proc. of the 2009 IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2009

[48] Wang, L.T., Stroud, C.E., Touba, N.A.: System on chip test architectures, Nanometer design for testability, Morgan Kaufmann Publishers, 2008, ISBN: 978-0-12-373973-5

[49] Xilinx Inc.: Development System Reference Guide 9.2i (ug628), [online] [2009-12-10]

[50] Xilinx Inc.: Device Reliability Report, third Quarter 2009[online] [2010-01-04].

[51] Xilinx Inc.: Early Access Partial Reconfiguration User Guide [online] [2009-12-10]

[52] Xilinx Inc.: Partial Reconfiguration Guide (ug 702). [online] [2010-10-05]

[53] Xilinx Inc.: Virtex-4 Configuration Guide (ug071), [online] [2009-12-10]

[54] Xilinx Inc.: Virtex-4 FPGA User Guide (ug070),[online] [2009-12-10]

[55] Xilinx Inc.: XST User Guide for Virtex-4, Virtex-5, Spartan-3, and Newer CPLD Devices, [online] [2010-14-12]

[56] Živković, M.: A Table of Primitive Polynomials, Mathematics of Computation Journal, Volume 62 Issue 205, Jan. 1994