TECHNICKÁ UNIVERZITA V LIBERCI

FAKULTA MECHATRONIKY A MEZIOBOROVÝCH
INŽENÝRSKÝCH STUDIÍ

KOMPRESE TESTOVACÍCH DAT ZALOŽENÁ NA PŘEKRÝVÁNÍ VZORKŮ

DISERTAČNÍ PRÁCE

2008 Ing. Jiří Jeníček

KOMPRESE TESTOVACÍCH DAT ZALOŽENÁ NA PŘEKRÝVÁNÍ VZORKŮ

DISERTAČNÍ PRÁCE

Disertant: Ing. Jiří Jeníček

Studijní program: P 2612 Elektrotechnika a informatika

Studijní obor: 2612V045 Technická kybernetika

Pracoviště: Ústav informačních technologií a elektroniky

Fakulta mechatroniky, informatiky a mezioborových studií

Technická univerzita v Liberci

Školitel: Prof. Ing. Ondřej Novák, CSc.

Rozsah práce a příloh:

Počet stran: 114

Počet obrázků: 25

Počet grafů: 11

Počet tabulek: 13

Počet příloh: 1

Prohlášení

Tuto práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací se svým školitelem.

V Liberci dne 29. 12. 2008

Ing. Jiří Jeníček

Poděkování

Rád bych poděkoval vedoucímu práce prof. Ing. Ondřeji Novákovi, CSc. za cenné rady a odborné připomínky. Dále bych chtěl poděkovat týmu na naší univerzitě za umožnění vypracování práce, ochotu a přístup při její realizaci.

V neposlední řadě chci poděkovat své rodině za všestrannou podporu při studiu.

Výzkum byl podpořen z výzkumného grantu NPV 1QS 108/04/0510 „Technologie pro zlepšení testovatelnosti moderních číslicových obvodů 2005-2008“ Akademie věd ČR.

Abstrakt

Tato práce pojednává o cílech a metodách komprese testovacích vektorů. Cílem práce je vytvoření nového algoritmu komprese testovacích vektorů tak, aby bylo možné urychlit výpočet, s důrazem na urychlení především u rozsáhlých obvodů. Práce vychází z již funkčního kompresoru testovacích vektorů COMPAS.

Nový algoritmus efektivně využívá struktury vstupních dat pro snížení časových i paměťových nároků komprese. Zlepšenou analýzou vstupních dat je zároveň dosaženo zlepšení kompresního poměru. Součástí práce je ověření efektivity použitých algoritmů na testovacích obvodech ISCAS 85, ISCAS 89 a ITC99.

Uvedené úpravy výrazně snížily dobu běhu programu i jeho systémové nároky při současném zkrácení délky komprimované modifikační posloupnosti a zachování pokrytí poruch.

Abstract

The thesis deals with methods and objectives of test pattern compression. The main target is to design a new compression algorithm, which will speed up the compression, especially in case of large circuits. The work is based on functional test pattern compressor called COMPAS.

The new algorithm effectively uses the input data structure to lower both compression time and memory requirements. Improved analysis of the input data allows further shortening of the compressed sequence. The thesis also includes algorithm efficiency comparison based on measuring of the time consumed for compression of test vectors for ISCAS 85, ISCAS 89 and ITC99 benchmark circuits.

The introduced modifications have significantly decreased program runtime, lowered program system requirements, shortened compressed sequence length and maintain fault coverage.

Obsah

Seznam obrázků

Seznam grafů

Seznam tabulek

Seznam použitých zkratek a značek

ATPG	Automatic Test Pattern Generator
ATE	Automatic Test Equipment
BIST	Built-in Self Test
COMPAS	Compressed test Pattern Sequencer
CUT	Circuit Under Test
DUT	Device Under Test
ETC	Embedded Tester Core
FA	Future Array
FL	Fault List
IDDQ	Idd (supply current) in the Quiescent state
ITE	Ústav Informačních technologií a elektroniky
LFSR	Linear Feedback Shift Register
OpenMP	Open Multi-Processing
RESPIN	REusing Scan chains for test Pattern decompressIoN
SC	Scan Chain
SEU	Single Event Upset
SoC	System on a chip
SSBDD	Structurally Synthesized Binary Decision Diagram
TAM	Test Access Mechanism
TPL	Test Pattern List
UFL	Undetected Fault List

1. Úvod

Ústav informačních technologií a elektroniky (ITE) na fakultě mechatroniky, informatiky a mezioborových studií Technické univerzity v Liberci se dlouhodobě zabývá problematikou testování číslicových obvodů. Jedním ze zaměření výzkumu je problém vestavěných generátorů testů.

Tato práce se zabývá návrhem nového algoritmu komprese testovacích dat, který je založený na překrývání testovacích vektorů. Nový algoritmus urychlí a zlepší kompresi a sníží systémové nároky. Práce staví na již funkčním programu COMPAS (COmpressed test Pattern Sequencer), který byl již dříve členy týmu vyvinut a který úzce spolupracuje se simulátorem poruch.

Práce je strukturována následujícím způsobem:

Kapitola 1 je věnována úvodnímu slovu.

Kapitola 2 seznamuje čtenáře s problematikou testování a komprimace testovacích vektorů.

Kapitola 3 popisuje celkovou strukturu systému COMPAS.

Kapitola 4 krátce popisuje stávající algoritmus a návrh nového kompresního algoritmu.

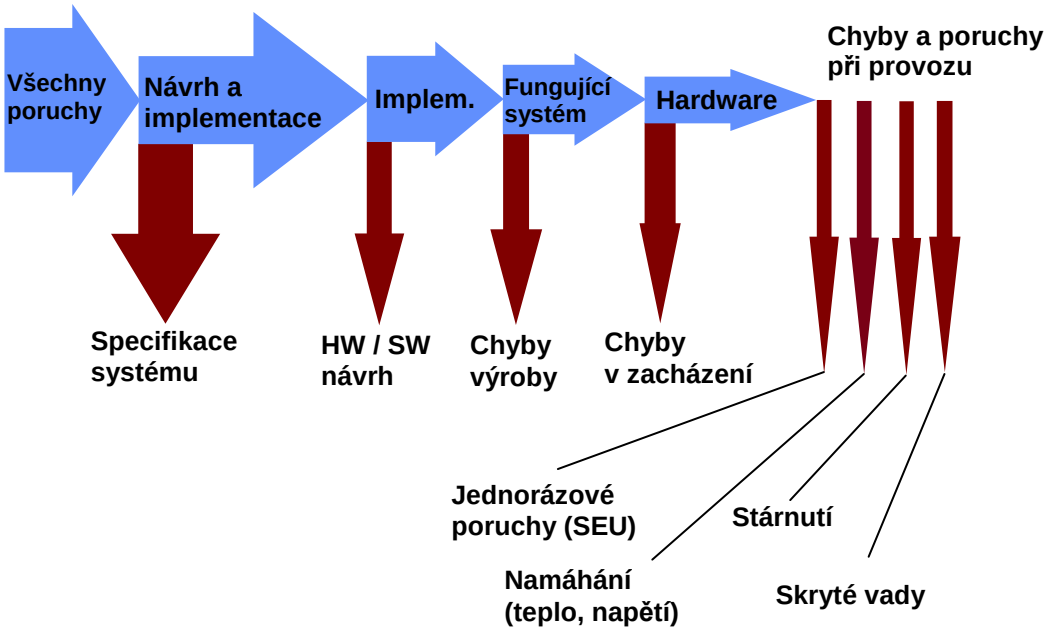
Kapitola 5 tvoří hlavní část práce a je věnována podrobnému popisu jednotlivých částí kompresního algoritmu a jejich vlivu na rychlost programu.

Kapitola 6 hodnotí výsledky dosažené v této práci a porovnává navržený algoritmus s ostatními řešeními.

2. Obecně o diagnostice

V dnešní době zákazník, který kupuje jakýkoli produkt, chce mít jistotu, že tento produkt funguje tak jak má. Výrobce tedy musí zajistit, aby jeho produkt byl bezchybný. Nejinak je tomu i v případě číslicových obvodů.

Chyby v obvodu se mohou vyskytnout v libovolném okamžiku jeho života (viz Obrázek 1: Druhy poruch [51]). Chybná funkce obvodu může být způsobena chybou vzniklou už při navrhování obvodu. Tyto chyby jsou odstraňovány v průběhu návrhu opakovanou verifikací. Jiné chyby mohou vzniknout při výrobě obvodu, další možností jsou chyby vzniklé v průběhu používání obvodu, způsobené stárnutím a vnějšími vlivy. Zatímco chyby návrhu a chyby vzniklé při výrobě jsou vždy trvalého charakteru, chyby vzniklé při používání obvodu mohou být jak trvalé (zničení obvodu přepětím), tak i dočasné (chybná hodnota způsobená nedostatečným napájením, radiací apod.).



Obrázek 1: Druhy poruch

Každý vyrobený obvod a následně testovaný obvod (CUT), by měl projít testem, který prokáže, že obvod je v pořádku a pracuje správně. Jedna z možností, jak test

provést je, že v jednom kroku testu jsou na primární vstupy číslicového obvodu přivedeny hodnoty tvořící vstupní vektor, odezva obvodu na jeho primárních výstupech potom tvoří výstupní vektor. Ten přitom musíme znát předem, abychom mohli říci, zda tento krok testu proběhl v pořádku. Vhodně zvolená skupina vstupních testovacích vektorů (a jim odpovídajících vektorů výstupních) potom tvoří test obvodu.

Test obvodu je možné provádět opakovaně i při používání obvodu, buď přímo při provozu (tzv. online testování), nebo jednou za čas přerušit funkci obvodu, provést test a pokračovat v práci (tzv. offline test).

Tato práce se zabývá chybami trvalého charakteru, vzniklými jak při výrobě tak při provozu, které budou testovány offline testem.

Obtížnější úkol je provést diagnostiku obvodu. Zatímco při testu obvodu je jako dostačující výsledek prosté konstatování, že obvod testem prošel v pořádku, při diagnostice je nutné zjistit co nejpřesněji, proč obvod nepracuje podle předpokladů. K tomu je většinou nutné použít větší množství speciálně generovaných dat a diagnostika je tedy časově i výpočetně náročnější.

2.1. Základní pojmy

Objektem zkoumaným při testování může být jednotlivé jádro, celý integrovaný obvod, ale i osazená deska plošných spojů nebo celý systém. V našem případě uvažujeme jen testovaný obvod. Ten pro komunikaci s okolím vyžívá svoje vstupy a výstupy. Přivedeme-li na vstupy obvodu testovací logické bity při současné znalosti aktuálního stavu obvodu, můžeme následným porovnáním odezvy obvodu s předem známými bezchybnými hodnotami výstupu určit funkčnost obvodu.

Testování může být prováděno na libovolné úrovni obvodu, od tranzistorů přes hradla, funkční bloky, jádra, obvody až k celým systémům. Ve všech případech je nutné na vstup přivést známý testovací vektor. Testovacím vektorem se nazývá soubor vstupních testovacích bitů. Pro číslicové obvody může vektor obsahovat

hodnoty log. 0 a log. 1. Počet bitů vektoru je shodný s počtem vstupů obvodu. Testovací vektor může být aplikován v simulátoru, přímo testerem nebo jiným blokem v obvodu. Pokud vektor vygenerovaný automatickým generátorem testu obsahuje nespecifikované hodnoty (obvykle se používá znak X), musí se všechny nespecifikované hodnoty před aplikací vektoru na vstupy obvodu nahradit buď log. 1 nebo log. 0. Schopnost přivádět známé testovací vektory na vstupy obvodu se nazývá řiditelnost.

Aby měl test vypovídací schopnost, je nutné aplikovat testovací vektory při známém vnitřním stavu testovaného obvodu. Ten může být navozen buď přivedením počátečních nastavovacích hodnot nebo vynulováním obvodu. V případě neznámého vnitřního stavu není možné předpovědět odezvu na vstupní testovací vektor. V průběhu aplikace testu se proto nikde nesmí vyskytnout neznámá nebo náhodná hodnota a odezva na známý vnitřní stav a známý vstupní vektor musí být vždy stejná. Test se potom nazývá deterministický. Výjimkou povolující existenci náhodných nebo neznámých hodnot je možnost identifikovat a ignorovat tyto hodnoty v průběhu testu.

Znalost správných odezev je nutná pro vyhodnocení správné funkce obvodu. Možnost zachytit odezvy skutečného testovaného obvodu se nazývá pozorovatelnost. Odezva obvodu je poté srovnána s předem známými správnými odezvami. Pro výstupy obvodu je, stejně jako pro vstupy, nutné, aby neobsahovaly žádné náhodné nebo neznámé hodnoty. Pokud je obsahují, je nutné mít možnost je ignorovat nebo zamaskovat.

Obvod generující testovací vektory se nazývá generátor vektorů, jeho výstupy jsou přivedeny na vstupy testovaného obvodu. Na vstup generátoru vektorů se přivádí testovací (někdy též modifikační) posloupnost. Testovací posloupnost je skupina bitů, které řídí generátor testovacích vektorů. Délka testovací posloupnosti je definovaná počtem modifikačních bitů. Pojmem defekt (defect) je míněn fyzický problém vzniklý např. přítomností nečistot při výrobě obvodu z křemíkového substrátu. Běžné defekty CMOS technologie jsou např. nedostatečné dopování, přerušení kovového propoje, spojení kovových propojů

(navzájem, proti napájecímu napětí nebo proti zemi), nepřítomné vertikální propoje (vias), apod. Protože fyzický defekt není možné nějak přímo matematicky popsat, je nutné zavést zjednodušený popis - model poruch. Model poruch (fault model; např. model trvalých poruch) musí existovat, aby bylo možné vyhodnocovat množství poruch. Termín porucha (fault) je modelový stav vyjadřující nesprávnou funkci obvodu, která nastává při přítomnosti defektu v obvodu. Termín chyba (error) značí rozdíl mezi správnou a skutečnou hodnotou. Chyba je vždy důsledek poruchy, ale ne každá porucha se musí projevit jako chyba. Aby se porucha mohla projevit chybou, musí být možné poruchu vybudit a pozorovat. Poruchy, které se nedají otestovat se nazývají nedetekovatelné poruchy.

Krok testu je dvojice tvořená jedním vstupním vektorem a jemu odpovídajícím výstupním vektorem. Test je potom množina dvojic vzájemně přiřazených vstupních a výstupních vektorů. Délka testu je rovna počtu kroků testu. Poměr počtu detekovatelných a skutečně detekovaných poruch obvodu vyjadřuje termín pokrytí poruch, který je vždy vztažený k určitému testu. Pokud tedy test detekoval všechny detekovatelné poruchy, je jeho pokrytí 100%. (Někdy se ovšem jako pokrytí bere poměr detekovaných a všech možných poruch v obvodu. Potom je místo termínu pokrytí ve smyslu první definice používán termín efektivita testu.) Optimální je test se stoprocentním pokrytím a minimální délkou.

2.2. Model poruchy

Model poruch, tedy zjednodušený popis skutečného fyzického defektu, je nutný především kvůli snadnějšímu matematickému popisu chování obvodu. Proto by měl model poruch dostatečně přesně popisovat chování defektů, a zároveň být dostatečně výpočetně efektivní. S použitím modelu poruch je pak následně vygenerován test obvodu, vyhodnocena kvalita toho testu a případně diagnostikovány poruchy.

Jedním z nejstarších a zároveň nejpoužívanějších modelů poruch je model trvalých poruch (stuck-at faults). Jedná se nízkoúrovňový strukturně založený

model. Používá se především na úrovni hradel, je ale možné adaptovat ho i na úroveň tranzistorů, registrů atd.

Tato práce předpokládá použití modelu trvalých poruch, jak je následně popsán. Trvalá porucha je v čase neměnná porucha, která se projeví neměnnou logickou hodnotou na vodiči. V obvodu je v průběhu celého testu předpokládán výskyt žádné nebo pouze jedné trvalé poruchy (kvůli snížení náročnosti). Model trvalých poruch je časově nezávislý, neuvažuje tedy taktovací frekvenci obvodu nebo zpoždění na jednotlivých prvcích. Existují dva typy trvalých poruch: trvalá jedna (t1) a trvalá nula (t0). Porucha t1 znamená zkratování vodiče na zdroj log. 1 (napájecí napětí) a projevuje se hodnotou log. 1 na vodiči. Porucha t0 je pak zkrat proti zdroji log. 0 (zemi) a projeví se hodnotou log. 0.

Uvažované trvalé poruchy se mohou vyskytnout v libovolném místě na vodičích propojujících logická hradla. Není tedy uvažován vnitřní tranzistorový model hradel a případné poruchy uvnitř. Logické hradlo tedy vždy plní předem danou funkci. To sice oproti např. tranzistorovému modelu dále snižuje komplexitu modelu, ale zároveň zhoršuje možnost modelu zachytit různé defekty.

V současných vysoce integrovaných obvodech jsou kvůli rychlosti většinou používána jednoduchá hradla. Obvod proto obsahuje velké množství napájecích vodičů, které musí být přivedeny ke každému hradlu. Protože je tak poměr napájecích vůči signálovým vodičům poměrně velký, je velké množství defektů, které se projevují spojením dvou vodivých cest v obvodu ve skutečnosti totožných s modelem trvalých poruch. To opravňuje použití poměrně jednoduchého modelu trvalých poruch i pro nové obvody [51].

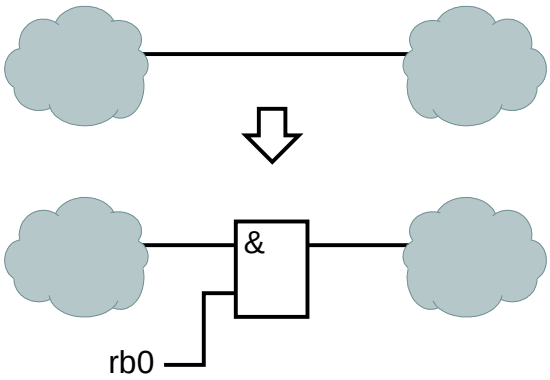
2.3. Metody zlepšující testovatelnost

Dvě základní metody pro zlepšení testovatelnosti jsou vkládání testovacích bodů a metody strukturovaného návrhu.

Testovací body jsou používány především při konstrukci větších obvodových celků jako desky plošných spojů apod. Protože se jedná o ad-hoc metodu, tedy

metodu která je použita až po vytvoření návrhu, a protože její výsledek závisí především na zkušenostech pracovníka, v podstatě není v dnešních vysoce integrovaných obvodech používána. Místo toho je nahrazována metodami strukturovaného návrhu, především plným a částečným skenem.

Testovací body jsou do obvodu umisťovány proto, aby se zvýšila pozorovatelnost nebo řiditelnost v daném místě. Existují dva typy testovacích bodů: pozorovací body a řídicí body. Řídicí bod může být na desce plošných spojů realizován například mechanickou propojkou, která je při testování rozpojena a místo ní je připojen na jeden konec propojky tester a na druhý konec propojky externí signál tak, aby byla zvýšena řiditelnost obvodu. Pozorovací bod pak může být také realizován jen absencí laku na spoji, kam se přitiskne měřicí hrot. Protože testování pomocí manipulací s mechanickými propojkami je obtížné, je možné je nahradit propojky vhodnými logickými prvky.

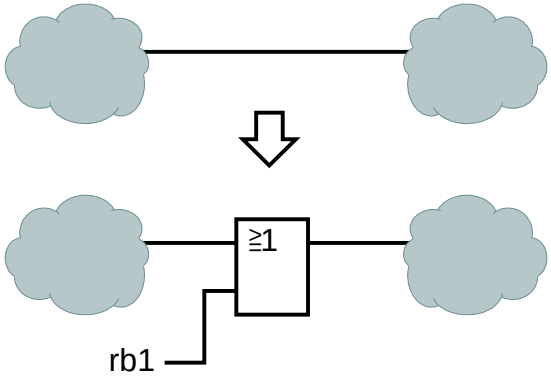


Obrázek 2: Vkládání řídicího bodu 0

Vkládání testovacích bodů pomocí přidaných logických prvků je také prakticky jediná ekonomicky realizovatelná cesta pro integrované obvody. Vkládání pozorovacího bodu je jednoduché, sledovaný signál se vyvede jako jeden z výstupů obvodu, a je tak následně možné jej uvažovat jako část výstupního vektoru.

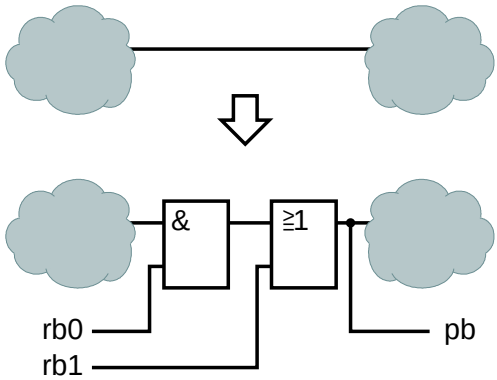
Řídicí bod v logickém obvodu je dvojího druhu, umožňuje buď vkládání hodnoty log. 0 nebo log. 1. Pro vložení log. 0 se používá dvojvstupé hradlo AND (viz

Obrázek 2: Vkládání řídicího bodu 0), pro vkládání log. 1 dvojvstupé hradlo OR (Obrázek 3: Vkládání řídicího bodu 1).



Obrázek 3: Vkládání řídicího bodu 1

Nastavením řídicího signálu rb0 do log. 1 je umožněn průchod běžné funkční hodnoty, nastavením do nuly je pak libovolný signál blokován a v řídicím bodě je nastavena log. 0. Obdobně při potřebě nastavení log. 1 v druhém řídicím bodě je na rb1 přivedena log. 1. Je-li potřeba mít možnost nastavovat na daném místě obvodu libovolnou hodnotu, je nutné použít oba druhy řídicího bodu najednou (viz Obrázek 4: Vkládání obou řídicích bodů).

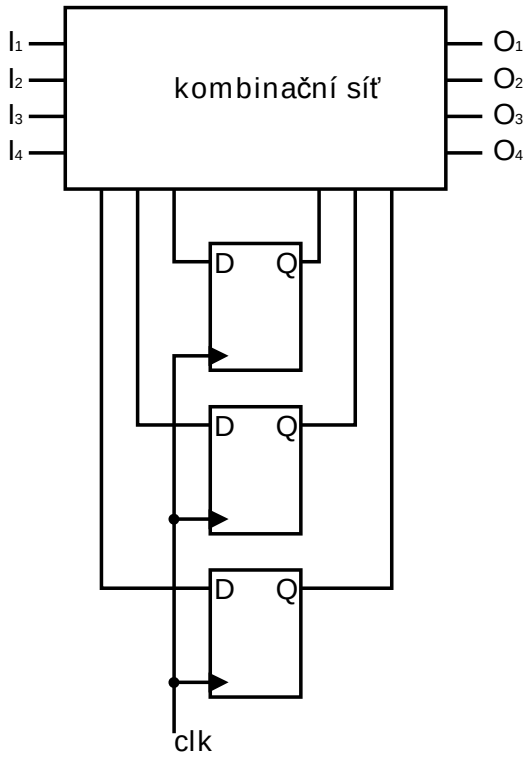


Obrázek 4: Vkládání obou řídicích bodů a pozorovacího bodu

Pro každý jednotlivý řídicí bod je potřeba vytvořit pomocný vstup obvodu, na který je pak možné přivádět data z testovacího vektoru. Je tedy zřejmé, že pro větší počet testovacích bodů narůstá potřebná plocha čipu (zabíraná dodatečnými hradly), ale především neúnosně narůstá počet vývodů obvodu. Množství

pomocných výstupů je možné řešit (za cenu dalšího růstu velikosti čipu a za cenu prodloužení doby testu) přidáním multiplexoru na pomocné výstupy.

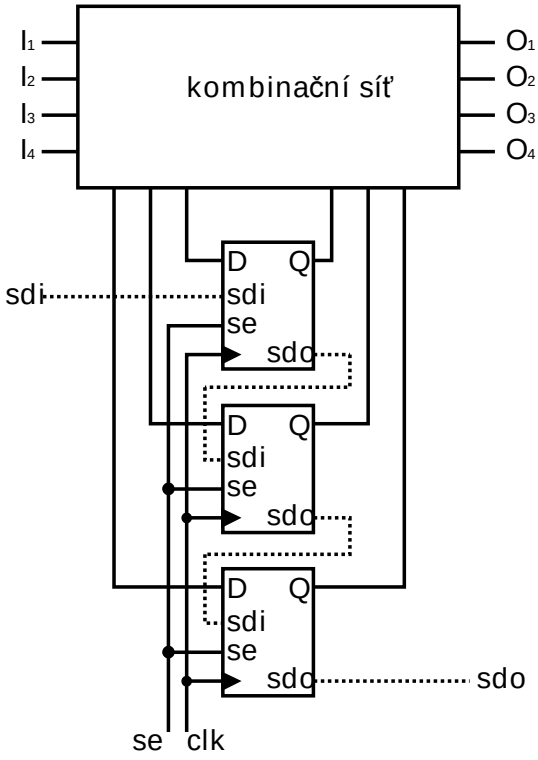
Strukturovaný návrh vychází z myšlenky, že testovaný obvod je možné rozdělit na dvě části: část obsahující čistě kombinační logiku a část obsahující pouze sekvenční (paměťové) prvky vyjadřující stav obvodu v čase.



Obrázek 5: Typizovaný sekvenční obvod

Běžně by se sekvenční obvod testoval pomocí nulovacích a nastavovacích sekvencí, to je ale náročné na tvorbu testu, test obtížněji dosahuje vysokého pokrytí a je časově i paměťově náročný. Řešením je přidaná logika a modifikace zapojení paměťových prvků v sekvenční části obvodu tak, aby bylo možné nastavit je do libovolné hodnoty.

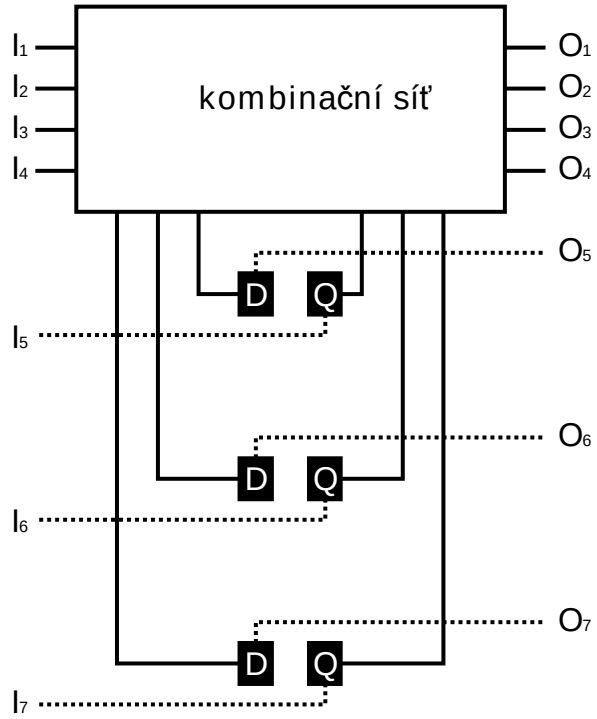
Ve chvíli, kdy je možné libovolně nastavovat všechny paměťové prvky, hovoříme o plném skenu, pokud lze nastavit jen některé, jde o sken částečný. Nejběžnější metodou zapojení je tzv. skenovací řetězec, kdy se všechny paměťové prvky zřetězí a vytvoří tak posuvný registr (existují i jiné, např. Random Access Scan).



Obrázek 6: Obvod s plným skenem

Plný sken je tak vlastně možné brát jako způsob, kterým do obvodu přidáváme ke všem sekvenčním prvkům řídicí i pozorovací bod, resp. vstup a výstup obvodu. Tím se výrazně zlepšuje testovatelnost obvodu, a tedy i snižuje náročnost na generovaní testu (viz Tabulka 1: Zlepšení testovatelnosti obvodu vložením skenu).

Po implementaci skenovacího řetězce jsou pak rozlišovány dva základní režimy činnosti obvodu: funkční režim, kdy obvod plní navrženou funkci, a režim testovací. V testovacím režimu je vstup i výstup skenovacího řetězce vyveden jako primární vstup a výstup, sekvenční prvky jsou vyřazeny z běžné funkce a obvod je možné brát jako čistě kombinační síť (viz Obrázek 7: Obvod s plným skenem z pohledu ATPG).



Obrázek 7: Obvod s plným skenem z pohledu ATPG

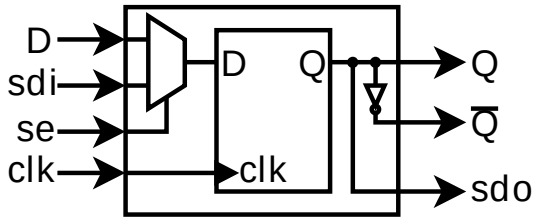
Krok testu se následně skládá ze sériového naplnění skenovacího řetězce (scan load-unload) při kterém zároveň dochází k přečtení předchozího obsahu řetězce, přivedení vstupních hodnot na primární vstupy (scan data apply), následného taktu funkčních hodin, který do paměťových elementů a na primární výstupy zapíše odezvu kombinační sítě (scan sample). Poslední krok je zároveň shodný s funkčním režimem. Skenovací řetězec tak umožňuje nastavit vnitřní stavová data obvodu a vyčíst následná stavová data pomocí jediného posuvného registru.

Tabulka 1: Zlepšení testovatelnosti obvodu vložením skenu

	Bez skenovacího řetězce	**Se skenovacím řetězcem**
Počet hradel	100 000	100 000
Počet poruch	500 000	500 000
Počet D klop. obvodů	5 000	efektivně 0
Max. sekvenční hloubka (počet klopných obvodů v signálové cestě)	500	0
Pinů	50	50 + 5 000
Hradel na pin	2 000	cca 19,8
Největší délka nastavovací sekvence	$2^M=2^{500}$ hluboce sekvenční obvod	$2^M=2^0=1$ kombinační obvod

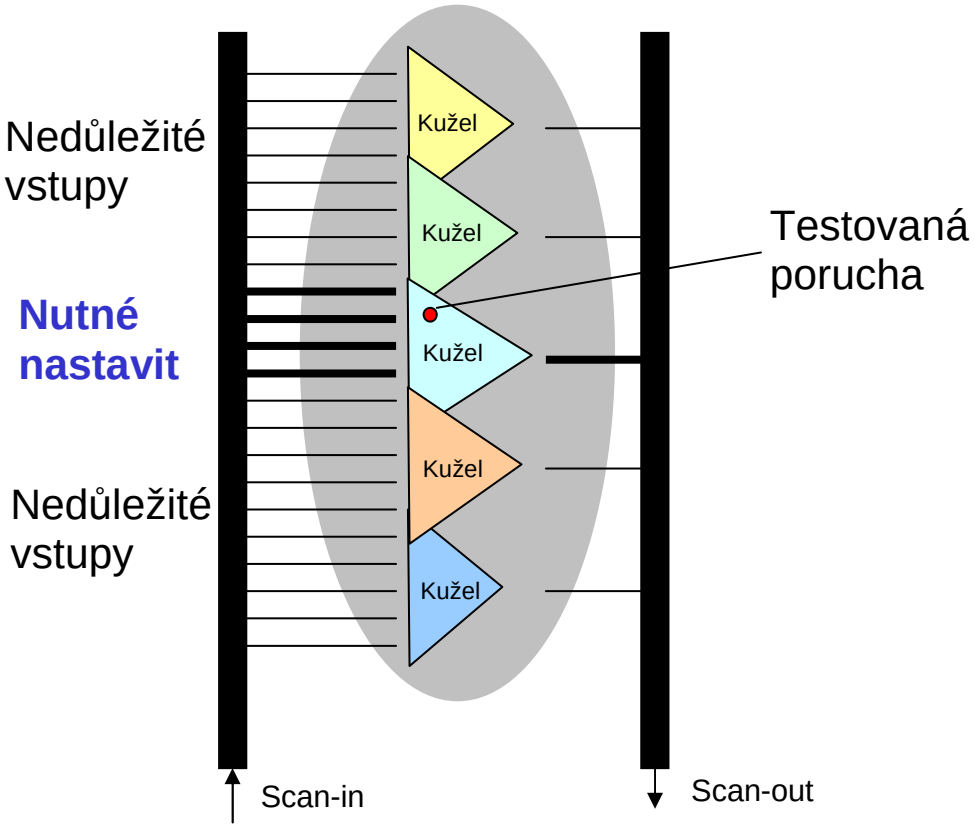
Vytváření skenovacího řetězce v obvodu probíhá postupným nahrazováním jednotlivých klopných obvodů skenovatelnými klopnými obvody a jejich řetězením. Příklad skenovatelného klopného obvodu ukazuje Obrázek 8: Skenovatelný klopný obvod typu Mux-D. Na obrázku značí *D* běžný vstup klopného obvodu, *Q* a ┐*Q* běžný a negovaný výstup, *sdi* a *sdo* značí sériový vstup a výstup, *se* volbu skenovacího režimu (scan-enable) a *clk* hodinový vstup.

Pořadí spojování může být při automatické náhradě řízeno například fyzickou vzdáleností jednotlivých klopných obvodů na čipu apod. Vkládání skenovacího řetězce klade dodatečné nároky na plochu čipu díky přidané logice a přidaným propojům.



Obrázek 8: Skenovatelný klopný obvod typu Mux-D

Ty je možné snížit například částečným skenem, kdy jsou do skenovacího řetězce zařazeny jen některé klopné obvody. Při částečném skenu je možné postupovat dvěma způsoby. V prvním případě je snaha o minimální velikost skenovacícho řetězce, kdy jsou klopné obvody do řetězce zařazovány z důvodu zlepšení testovatelnosti obvodu. Druhým přístupem je naopak snaha zařadit do řetězce všechny klopné obvody a vytvořit tak plný sken, ale z různých důvodů je počet prvků skenu zmenšován (například o klopné obvody v časově kritickým cestách, nebo rozsáhlé soubory registrů). Oba přístupy ale kladou zvýšené nároky na návrháře i na návrhové prostředky, protože je třeba správně zvolit, které klopné obvody do skenovacího řetězce zařadit a které ne. Výsledný obvod je také z hlediska ATPG stále obvod sekvenční, a proto je tvorba testu náročnější než pro obvod s plným skenem.

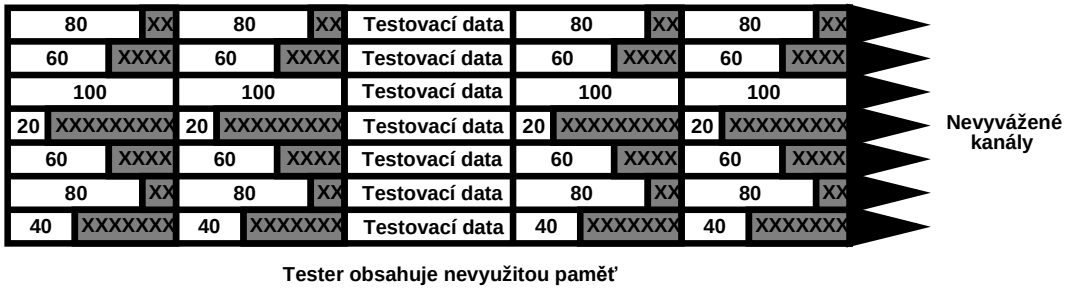


Obrázek 9: Testování poruchy v obvodu se skenem

Při testování pomocí skenovacího řetězce může nastat problém s časovou náročností. V průběhu jednoho testovacího kroku je totiž potřeba sériově načíst a zároveň vysunout všechna data z řetězce. To např. pro obvod s 1000 klopnými obvody představuje 1000 hodinových taktů. Jednoduchým řešením je rozdělit jeden dlouhý skenovancí řetězec na více kratších, potom ale vzrůstá počet vstupů a výstupů. Pokud bychom řetězec s 1000 buňkami rozdělili na 10 kratších řetězců, pro načtení a vyčtení dat bude potřeba jen 100 hodinových cyklů a test se tak 10x zrychlí. Jeden řetězec ale potřebuje jen 3 piny (sériový vstup, sériový výstup a scan-enable), zatímco 10 skenovacích řetězců potřebuje 21 pinů (10 sériových vstupů, 10 sériových výstupů a jeden společný scan-enable). Při volbě více skenovacích řetězců je tedy vždy potřeba hledat kompromis mezi délkou testu a počtem využitých testovacích pinů.

Obrázek 10: Obsazení paměti testeru při použití jednoho skenovacího řetězce

Při volbě, zda využít více skenovacích řetězců, je třeba si uvědomit, že se jedná pouze o způsob dopravy testovacích dat z a do obvodu, a že z pohledu generování testu se jedná o stále stejný obvod. Náročnost vytvoření testu pomocí ATPG, počet testovacích vektorů a celkový objem dat bude úplně stejný.



Obrázek 11: Využití paměti testeru při použití více nevyvážených skenovacích řetězců

Z hlediska nejkratšího času pro krok testu je také vhodné, aby skenovací řetězce v obvodu byly vyvážené, tedy aby měly všechny stejnou délku. Toho lze vždy dosáhnout, s výjimkou posledního skenovacího řetězce. Vyvážené skenovací řetězce jsou kromě rychlosti optimální i z hlediska spotřebované paměti testeru. Testovací vektor je v paměti testeru rozdělen na paralelní části příslušné jednotlivým skenovacím řetězcům. Ty jsou při testu zároveň nasouvány do sériových vstupů. Pokud jsou všechny řetězce stejně dlouhé, je paměť testeru využitá optimálně. V případě, že je některý z řetězců kratší než ostatní, testovací data musí být v paměti testeru zarovnána tak, aby byl do všech řetězců nasunut poslední bit zároveň. Při zarovnání dat je v paměti testeru pro všechny řetězce rezervována stejně velká paměť jako pro ten nejdelší z nich, a nevyužité bity na

začátku kratších řetězců jsou vyplněny náhodnými bity. Je proto lepší snažit se, aby všechny skenovací řetězce byly stejně dlouhé, protože potom nedochází k plýtvání paměti testeru. Skenovací řetězce je také vhodné navrhnout tak, aby případné zbylé klopné obvody vytvořily jeden kratší řetězec, který je v paměti testeru doplněn. Pokud by totiž byly zbylé klopné obvody přidány k poslednímu vyváženému řetězci, došlo by k doplňování paměti testeru u všech předchozích vektorů, což vede ke zbytečnému plýtvání paměti a prodloužení doby testu [53].

Obrázek 12: Využití paměti testeru při použití více vyvážených skenovacích řetězců

Existují různé typy skenovacích buněk (Mux-D, LSSD, Clocked-scan) určené pro testování trvalých poruch. Existují ale i architektury skenovacích řetězců schopné práce na provozní frekvenci zařízení, které jsou použitelné pro testování poruch zpoždění. Hlavními rozdíly je pak odlišné zapojení skenovacího řetězce a fakt, že pro testování se používají dvojice testovacích vektorů, kde první nastaví citlivou cestu obvodem a druhý způsobí změnu logické hodnoty v celé cestě. Na základě srovnání skutečné a teoretické rychlosti šíření změny signálu je pak možné usoudit na existenci různých defektů.

Testování pomocí jednoho nebo více skenovacích řetězců je metoda, která je standardizovaná, opakovatelná a jednoduše automatizovatelná jak z pohledu vkládání skenovacího řetězce, tak pro vygenerování testovacích dat. Zároveň zvyšuje pokrytí poruch a zrychluje generování testu a tím výrazně snižuje cenu testu. Jedná se proto o pravděpodobné nejčastěji používanou metodu. Možnými nevýhodami je v závislosti na kvalitě návrhu skenu případný nárůst plochy čipu, prodloužení doby testu, zvýšení spotřeby energie apod.

Stále častěji používanou metodou návrhu je využívání již hotových bloků zvaných vnořená jádra. Pokud pak celý elektronický systém umístíme na jediný čip, označujeme ho SoC. Jednotlivá jádra mohou být tvořena jinými vnořenými jádry, celý SoC tedy může mít hierarchickou strukturu. Testování takového systému přináší výhody, protože je při vhodném návrhu možné testovat jednotlivá jádra. Zároveň ale přináší problémy se zajištěním dostatečně rychlého komunikačního kanálu pro testovací data, protože jádra nejsou přímo přístupná. Záleží tedy především na testovací infrastruktuře uvnitř SoC.

Pro návrháře, který opakovaně používá jádra, je nevýhodné pokaždé přizpůsobovat testovací rozhraní jader novému návrhu. Je proto vhodné využívat standardy zavedené pro testování vnořených jader, především normu IEEE 1500 [43]. Testovací rozhraní jádra pak tvoří normovanou obálku jádra, která umožňuje standardním způsobem nastavovat různé provozní režimy vstupů a výstupů jádra.

2.4. Metody generování testu

Nejjednodušší na vytvoření je test, který prověří všechny možné vstupy obvodu, tzv. triviální test (v případě kombinačního číslicového obvodu testovací vektory obsahující všechny možné kombinace vstupu, v případě sekvenčního obvodu všechny možné sekvence všech kombinací), ten by ale v případě složitých obvodu byl příliš časově náročný a také objem dat (především odezvy, vstupní vektory je možné generovat algoritmicky přímo při testu) potřebný k otestování obvodu by byl neúnosný. Pokud je aplikována pouze podmnožina všech možných kombinací, jedná se o tzv. pseudotriviální test. Pro generování tohoto testu není nutné znát vnitřní strukturu obvodu.

Podobným testem je algoritmicky generovaný test. Tento test je sice jednoduché vygenerovat, ale dají se efektivně použít jen na jednoduché nebo pravidelné struktury (například sběrnice nebo paměti). Pro sestavení algoritmu, který bude generovat testovací vektory je nezbytně nutná znalost vnitřní struktury obvodu.

Dalším způsobem je funkční test, který u daného obvodu testuje jestli plní definovanou funkci (sčítačka sčítá čísla). Při tom je nutné otestovat všechny

možné kombinace vstupů, které mohou nastat při reálném provozu obvodu. Funkční testy ale vytváří návrhář, proto se u složitých obvodů obtížně a zdlouhavě vytvářejí. Pro vytváření tohoto testu také není nutné znát vnitřní strukturu obvodu, pouze jeho předpokládané chování. Pokud ale nejsou testovací vektory ohodnoceny z hlediska pokrytí poruch, jedná se o verifikaci funkce, nikoli o strukturní test, proto není tak efektivní pro testování defektů. Tento typ testu by tedy měl být prováděn především před výrobou obvodu, ne až po ní.

Další používaný způsob je vytvořit test, který prověří skutečnou topologii vyrobeného obvodu. Tento test pracuje s vnitřní strukturou obvodu a nazývá se proto strukturní test. Při použití modelu trvalých poruch metoda pracuje tak, že nastavuje vstupy obvodu na hodnoty, které přivádí k defektnímu místu v obvodu jeho opačnou hodnotu (např. přivede log. 0 k poruše t1) a zároveň nastavuje vstupy tak, aby se správná hodnota mohla šířit až k výstupu obvodu. Pokud je v pozorovacím bodě detekována hodnota odlišná od předpokládané, potom je detekována porucha. Základní metodou generování strukturního testu je metoda zcitlivění cesty. Principem je sestavení citlivé cesty od místa poruchy na libovolný pozorovací bod. Citlivá cesta je sled vzájemně propojených hradel, který je schopen přenášet změnu signálu od počátku do konce. Pro zcitlivění cesty je nutné nastavit ostatní vstupy hradel, kterými cesta prochází, tak, aby nebyl blokován průchod signálu o obou logických hodnotách. Pro hradla AND a NAND je tedy potřeba nastavit ostatní vstupy do log. 1, ostatní vstupy hradel OR a NOR pak do log. 0. Invertorem, bufferem a nonekvivalencí prochází citlivá cesta vždy. Pro zcitlivění cesty se používá množství algoritmů, např. D algoritmus, PODEM, SOCRATES, FAN a další [38].

Ne všechny poruchy se musí projevit navenek chybou. Poruchu, která se nikdy neprojeví chybou, tedy nebude možné zjistit, a proto pro ní není třeba generovat test. Je třeba tedy nalézt takovou sadu testovacích vektorů, která bude co nejmenší, a pomocí níž dosáhneme maximální pokrytí poruch. Pokud pokrytí bude 100%, dokázali jsme vytvořit úplný test.

2.5. Problém velikosti testu

Díky narůstající komplexnosti integrovaných obvodů, lepší výrobní technologii a většímu počtu jader na jeden čip rostou i nároky na testování, zvětšují se objemy testovacích dat a prodlužují se časy potřebné k otestování obvodu. Doba, kterou trvá otestování jednoho obvodu je přitom hlavní položkou ceny testu. Proto je potřeba hledat nové techniky pro snížení objemu testovacích dat a tím zkrácení času potřebného k vykonání testu.

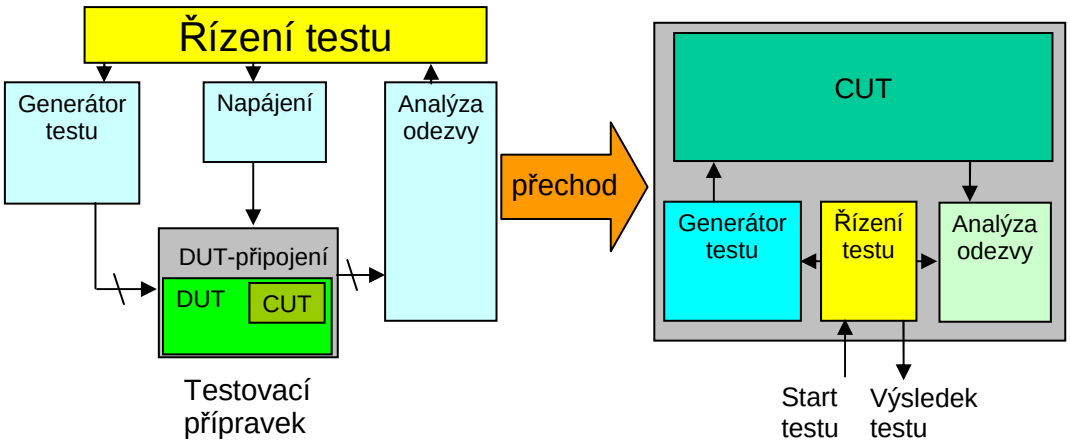
Se zvětšujícím se testem je potřeba řešit především následující problémy:

Omezená velikost paměti testeru: Paměť testeru musí být velmi rychlá, a proto je i drahá. Čím více máme testovacích vektorů, tím se zvyšují nároky na paměť. Při použití testeru, který nemá dostatečně velkou paměť, však dojde k tomu, že provedený test bude neúplný (neodhalí všechny poruchy) a tím nemůžeme mít jistotu, že testovaný obvod je v pořádku, nebo je nutné v průběhu testu postupně nahrávat části dat. To ale vede k podstatnému prodloužení doby testu.

Čas přesunu dat do testeru: Čím je objem dat potřebných k provedení testu obvodu větší, tím déle trvá přesun těchto dat do testeru. Je zřejmé, že pokud se podaří tento čas výrazně zkrátit, sníží se i cena celého testu, zejména pokud se objem testu podaří snížit tak, že se vejdou do testeru všechna testovací data najednou a nebude tak docházet k opakovanému načítání částí testu.

Úzký TAM: I když máme tester s dostatečně velkou pamětí pro uložení testovacích dat, čas přenosu těchto dat do testovaného obvodu je stále dlouhý kvůli úzkému TAMu. Propustnost testovacího mechanizmu je daná počtem testovacích pinů a jejich frekvencí. Omezení je dané především nízkém počtem testovacích pinů, protože čistě testovací piny jsou velmi drahé (nejsou v běžném provozu používány). Problém lze někdy řešit tím, že jsou piny sdílené pro funkční i testovací hodnoty, pak ale zase mají přidanou logiku, která způsobuje dodatečné zpoždění.

Alternativou k testům používajícím externí tester je tzv. Built-In Self-Test (BIST) (viz Obrázek 13: Přechod od externího testování k BISTu [51]). Použití BIST jako vestavěného generátoru testovacích vektorů je jednoduché především pro algoritmicky generované testy (především pro pravidelné struktury jako paměti, sběrnice apod.) nebo testy pseudonáhodné, použití pro testování obecných obvodů je složitější. Problémem je jednoduše a rychle navrhnout dostatečně malý generátor vektorů, který zvládne vygenerovat testovací vektory schopné detekovat poruchy odolné náhodnému testu. Proto se často používá smíšený test (mixed-mode), který kromě pseudonáhodné fáze použije k detekci poruch odolných náhodnému testu předem vygenerované deterministické testovací vektory. Ty mohou být buď uloženy v paměti (ať již mimo vlastní testovaný obvod, nebo přímo v něm) nebo vytvářeny za běhu technikami jako překlápění nebo fixace bitů. Při pseudonáhodném testu jsou testovací vektory většinou vytvářeny zpětnovazebním registrem, jedná se tedy ve skutečnosti o algoritmicky generovaný test řízený generujícím polynomem.



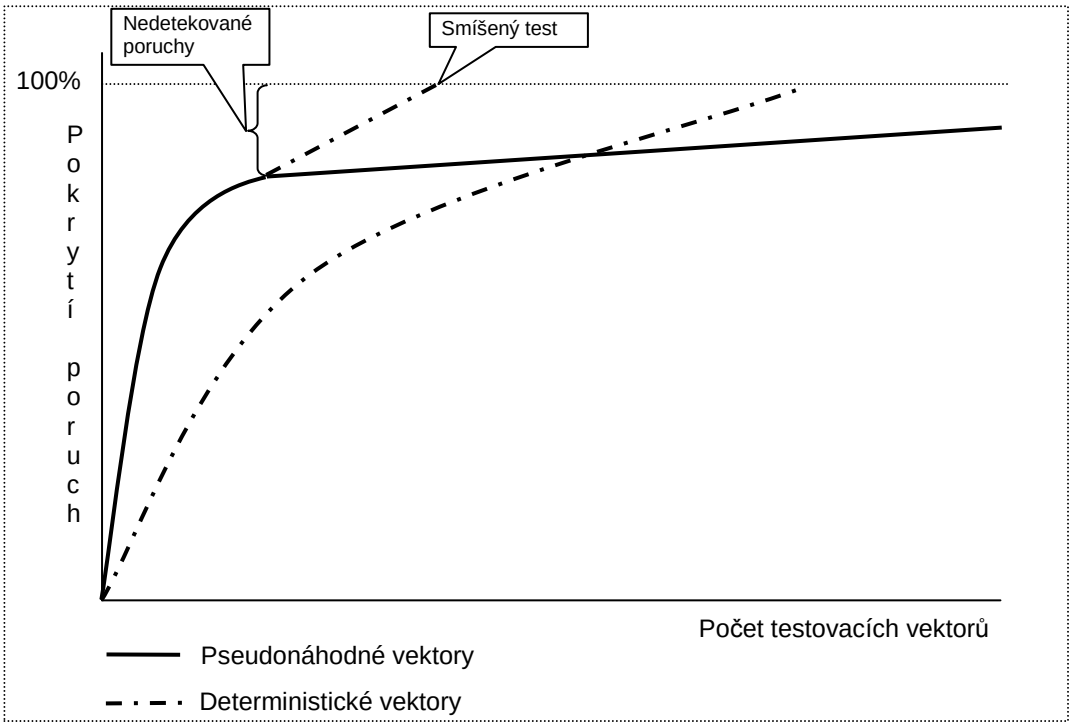
Obrázek 13: Přechod od externího testování k BISTu

Pseudonáhodná fáze generování testovacích vektorů sice významně snižuje počet deterministických vektorů, které musí být uloženy v paměti, nevýhodou ale je, že významně prodlužuje dobu testu a spotřebovanou energii. Smíšený test tak v závislosti na nastavení může být dobrým kompromisem mezi velikostí

hardwaru, množstvím paměti potřebným pro uložení testovacích dat a rychlostí testu.

2.6. Pseudonáhodné a deterministické vektory

Při testování pseudonáhodnými vektory velice rychle vzrůstá pokrytí poruch, zvláště při malé kombinační hloubce obvodu. To je užitečné pro současné rychlé sekvenční obvody, kde jsou tenké vrstvy kombinační logiky oddělovány registry. Pokud je následně pro testování použit skenovací řetězec, je kombinační logika velmi dobře řiditelná i pozorovatelná (viz Obrázek 9: Testování poruchy v obvodu se skenem). Po určité době jsou ale otestovány všechny poruchy testovatelné náhodným testem a zbývají jen poruchy odolné náhodnému testu. Generátor náhodných vektorů sice může být schopen postupně vygenerovat testovací vektory, které detekují i takové poruchy, ale většinou to trvá nepřípustně dlouho.



Graf 1: Rychlost nárůstu pokrytí poruch při použití různých druhů vektorů

Při použití deterministických vektorů bez nějakého druhu komprese je sice vždy dosaženo cíleného pokrytí poruch, nárůst pokrytí je ale pomalejší (viz Graf 1:

Rychlost nárůstu pokrytí poruch při použití různých druhů vektorů). Graf 1 vychází z experimentů prováděných pomocí programu BIST-Analyzer [50], je proto mírně odlišný od grafu uvedeném v [38].

Zrychlení nárůstu pokrytí poruch je možné buď pomocí použití některé z metod smíšeného testu nebo kompresí deterministických dat.

2.7. Komprese testu

Problémy popisované v předchozí kapitole se snaží zmírnit řada metod. Hlavním způsobem urychlení testu a tím snížení jeho ceny je komprimace. Různé metody komprese se s různou úspěšností snaží potlačovat hlavní problémy, tedy množství testovacích dat v testeru, dobu přenosu mezi testerem a obvodem a dobu samotného testu. Metody pro kompresi testovacích dat lze obecně dělit na dvě hlavní skupiny:

Metody, které nepotřebují znát strukturu obvodu: Do této skupiny patří především různé bezeztrátové komprimační techniky, např. Huffmanovy kódy, Golombovy kódy [27], FDR kódy [28]. Tyto metody kódují testovací vektory s použitím různých slovníkových a statistických metod. Protože nepotřebují znát vnitřní strukturu obvodu, je možné příslušný dekodér použít jako univerzální blok generátoru testovacích vektorů. Dekodér je ale většinou poměrně rozsáhlý. Mezi tuto metodu komprese lze zařadit i kompakci testovacích vektorů. Ta pracuje tak, že slučuje testovací vektory, přičemž využívá toho, že zpravidla obsahují velké množství nespecifikovaných bitů. Testovací vektory jsou potom slučovány do jediného vektoru tak, aby nenastala kolize mezi specifikovanými bity. Výsledkem je testovací vektor, který je schopen detekovat větší množství poruch.

Metody, které využívají strukturu obvodu: Při použití některé z těchto metod je nutné už na počátku návrhu generátoru testovacích vektorů využít simulátor poruch, automatický generátor vektorů ap. Metody využívají např. zpětnovazební registry (LSFR) [29] s případnou reinicializací jejich počátečního stavu, nebo i změnou generujícího polynomu. Dalšími metodami

jsou např. celulární automaty, rozkládající čítače (folding counters) [30], architekturu RESPIN++ [31] a metody pracující na základě překrývání testovacích vektorů [16].

Dalším problémem je pak komprese odezev obvodu. Ta může být prováděna jak v čase (výstupní vektor v několika po sobě jdoucích taktech zkombinujeme do jediného příznakového vektoru), tak v místě (z delšího výstupního vektoru obvodu vytváříme kratší příznakový vektor), například zpětnovazebním registrem nebo pomocí speciálních kombinačních obvodů jako X-Compact [48]. Zpravidla není nutné, aby byl při testu kontrolován každý stav výstupu, je možné až na konci testu načíst koncový příznak, a tak zjistit, zda se v průběhu testu některá z hodnot nelišila od hodnoty očekávané.

Vzájemné srovnání komprimačních technik je obtížné, protože je nutné brát v úvahu velké množství parametrů. Různé metody se obecně liší dosahovaným pokrytím, počtem bitů v paměti pro uložená testovací data, počtem hodinových taktů nutných pro vykonání testu, velikostí přidaného dekompresního hardware, složitostí řízení průběhu testu, znovupoužitelností testovacího hardware a dalšími parametry. Výběr vhodné komprimační metody je proto vhodné zvažovat už v počátečních fázích návrhu obvodu jako jednu z částí zásad návrhu pro snadnou testovatelnost.

3. Systém COMPAS

COMPAS – Compressed Test Pattern Sequencer je software pro kompresi testovacích vektorů pro obvody se sériovým diagnostickým přístupem. Tento software je vyvíjen pod vedením Prof. Ing. Ondřeje Nováka, CSc. Při komprimaci se používá metody pracující na základě překrývání testovacích vzorků [16].

Program slouží pro kompresi testovacích dat pro kombinační obvody nebo pro obvody s plným skenem. Je zaměřen na zpracování testovacích dat pro model trvalých poruch, přičemž využívá popis obvodu na úrovni hradel a simulátor poruch. Vstupní data mají formu dvojic, které jsou tvořeny nekomprimovaným testovacím vektorem a příslušnou poruchou. Je vhodné, aby testovací vektor obsahoval co největší množství nespecifikovaných bitů, protože ty lze nejsnáze překrýt.

Algoritmus obecně umožňuje kompresi libovolných testovacích vektorů. Vektory mohou obsahovat jen specifikované bity, pak je ale obtížné je překrýt a komprese nedosahuje dobrých výsledků. Vektory mohou být vytvořeny pro testování jiných než trvalých poruch. V tom případě ale není možné využít úzké spolupráce se simulátorem poruch, a je tak dosaženo horší komprese. Simulátor není možné použít ani v případě, že není k dispozici popis obvodu.

Kompresor COMPAS je platformě nezávislý, testován byl na platformách x86, x86-64 a sparc, pod operačními systémy Windows XP, Linux a SunOS ve verzích pro 32 i 64 bitů. Zdrojový kód je v jazyce C. K vývoji programu bylo použito vývojové prostředí Microsoft Visual Studio a editor Vim. Pro kompilaci byl podle použité platformy používán buď Microsoft Compiler nebo GCC.

3.1. Komunikační rozhraní

COMPAS je konzolový program, který s okolím komunikuje pomocí příkazového řádku, standardního vstupu, standardního a chybového výstupu a pomocí souborů. Program je stavěn pro dávkové spouštění, kdy jsou potřebné informace dodány na

příkazovém řádku, případně v datových souborech. To umožňuje zlepšit automatizované zpracování více souborů, případně ladění volených parametrů.

V průběhu programu nejsou vyžadovány na standardním vstupu žádné informace. Nutná data jsou buď načtena ze souborů a příkazové řádky, nebo je program automaticky vhodně zvolí. Jediný možný vstup je stisk CTRL-C, neboli zaslání signálu SIGINT (předčasné ukončení programu). Program předčasné ukončení detekuje, korektně uzavře všechny otevřené soubory a ukončí se.

Na standardní výstup jsou při běhu programu vypisovány běhové informace v závislosti na zvolené detailnosti výpisů. Program umožňuje plně tichý běh, kdy nejsou vypisovány žádné údaje (ani případné chyby a varování), přes souhrnné údaje až po velmi detailní výpis každého kroku programu použitelný pro ladění.

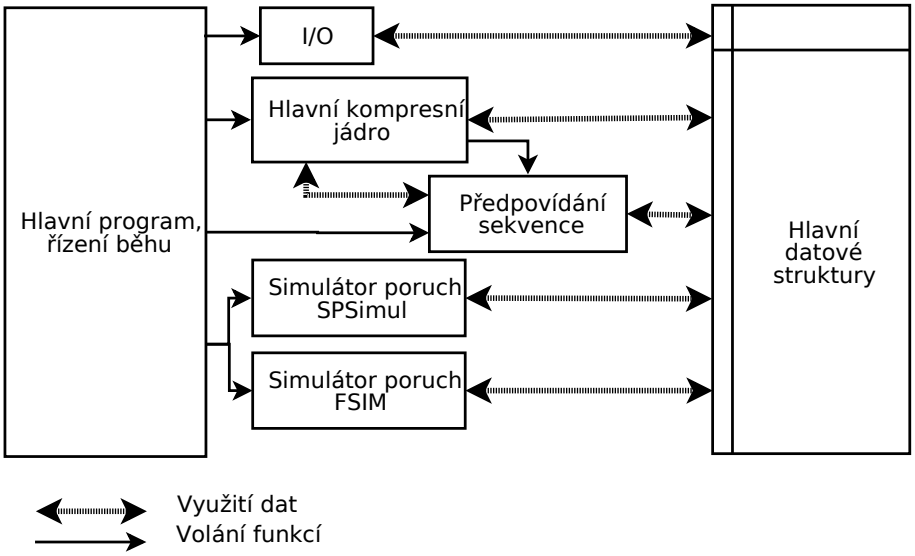
Na chybový výstup jsou vypisována varování a chyby programu. Zatímco při varování se program pokusí nastalou situaci řešit a pokračuje v běhu, při kritické chybě jsou vypsány údaje o nastalé chybě a program je ukončen.

Hlavní datová komunikace je realizována pomocí souborů. Program pracuje se soubory ve třech základních skupinách formátů: formáty dané programy ATALANTA, FSIM a HOPE, formáty dané programovým balíkem TurboTester a interní datové formáty. Pro převod z formátu EDIF do vhodných datových formátů existují vhodné programy, není proto problém importovat téměř libovolný obvod. Při načítání vstupních souborů je prováděna základní kontrola dat. Při zjištění chyb ve vstupních datech se pak program pokouší data vhodně opravit (např. při různé délce testovacích vektorů). Pokud to není možné, program končí.

Jaké soubory mají být načítány a zapisovány, množství výpisů, režim komprese a další parametry jsou nastavovány při startu programu na příkazové řádce (viz Dodatek A – Přehled nastavení příkazové řádky).

3.2. Základní struktura programu

Program se skládá z šesti základních bloků (viz Obrázek 14: Základní struktura programu). Každá část má přístup k hlavním sdíleným datovým strukturám, které přímo využívá. Vedle toho mají jednotlivé části svoje vlastní rozsáhlé statické i dynamické struktury, ke kterým nemají ostatní části programu přímý přístup.



Obrázek 14: Základní struktura programu

3.3. Interní datové struktury

Program využívá jednu hlavní trvalou globální dynamickou datovou strukturu jménem *patterns* a několik pomocných globálních struktur. Struktura patterns uchovává načtené testovací vektory a jejich statistické informace, jméno poruchy příslušející danému vektoru společně s přímým odkazem na poruchu v interních datových strukturách simulátorů poruch, a různé informace o předchozích krocích programu související s daným vektorem. Navázané globální struktury tvoří dvě pole ukazatelů na jednotlivé prvky struktury patterns. První pole je seřazené podle abecedního pořadí názvů poruch a slouží k rychlému binárnímu vyhledávání detekovaných poruch, pokud je použit externí simulátor poruch. Druhé pole pak obsahuje ukazatele seřazené podle hodnoty kritéria a slouží k rychlému výběru vhodných vektorů. Další globální struktury obsahují aktuální stav skenovacího

řetězce, celou výslednou komprimovanou posloupnost, struktury pro spekulativní odstraňování poruch, množství statistických informací apod.

Další rozsáhlé dynamické struktury jsou obsaženy v kompresním jádře programu a v bloku předpovídání bitové sekvence.

3.4. Komprese

Při kompresi dat systémem COMPAS s běžným nastavením nedochází ke ztrátě hlavní testovací informace, tedy pokrytí poruch, z hlediska pokrytí poruch se jedná o bezeztrátovou kompresi. Po dekompresi komprimovaných dat nedostaneme všechny původní vektory, z pohledu testovacích vektorů je tedy komprese ztrátová.

Jako vstupní data pro program slouží nekomprimované testovací vektory obsahující nespecifikované vstupní bity (don't care bit). Vzorky mají formu dvojice porucha – testovací vektor, tedy každé poruše přísluší právě jeden testovací vektor s co nejvíce nespecifikovanými bity. Tento typ testovacích vektorů není vždy snadné vygenerovat, protože je komerční generátory testů (např. programy fy. Mentor Graphics) málo podporují – na pozice nespecifikovaných bitů dosazují jedničky nebo nuly.

Výstupem programu je posloupnost nul a jedniček neobsahující žádné nespecifikované bity, která tvoří komprimovanou testovací posloupnost. Po kompresi programem COMPAS může být výsledná komprimovaná testovací posloupnost využita např. pro testování SoC.

Metoda komprese spočívá v postupném hledání testovací posloupnosti po jednom bitu z předem připravené sady testovacích vektorů – úplného testu. Pro kombinační obvody tato posloupnost při vlastním testu vstupuje do skenovací smyčky, jejíž velikost je stejná jako počet vstupů obvodu, pro sekvenční obvody je vhodné použít některou z testovacích architektur, např. RESPIN [52].

Na začátku algoritmu máme seznam zatím nedetekovaných poruch obvodu, seznam všech testovacích vektorů tvořících úplný test a ve skenovací smyčce jsou

samé nuly. Krok algoritmu pak probíhá tak, že je nalezen další jeden bit posloupnosti, dále je provedena simulace a ze seznamu poruch jsou vyškrtnuty ty, které jsou simulací detekovány. Algoritmus končí tehdy, když je seznam nedetekovaných poruch prázdný.

Kompresní algoritmus COMPASu začíná ze známého stavu, kterým je většinou vynulovaný skenovací řetězec. Dá se totiž předpokládat, že test obvodu začíná zapnutím napájení a vynulováním. Komprese ale může v případě nutnosti začít i s jiným stavem skenovacího řetězce.

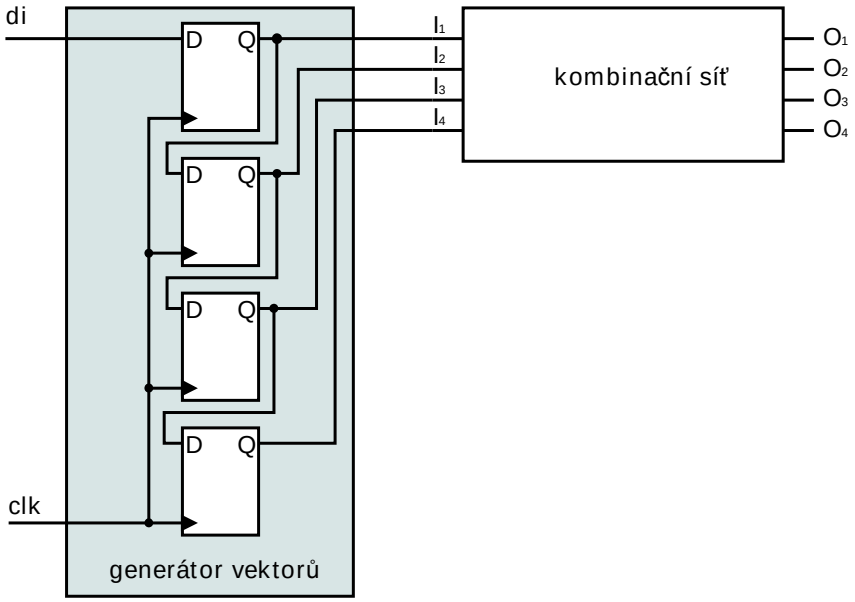
Použitý algoritmus také nepoužívá zpětné prohledávání, nevrací se k bitům vygenerovaným v předchozích krocích. Jakmile je tedy jednou komprimovaný bit vygenerován, je zařazen do výsledné posloupnosti a už není nikdy změněn. Toto omezení bylo zavedeno kvůli rychlosti. Při prvotních experimentech byla vyzkoušena i metoda se zpětným prohledáváním, z hlediska rychlosti byla ale naprosto nepoužitelná (komprese dat trvala hodiny pro obvody o desítkách hradel). Obecně by kompresní algoritmus mohl vyzkoušet všechna myslitelná pořadí seřazení vektorů, tím by se kompletně prohledal stavový prostor a bylo by nalezeno optimální řešení. Hledání by ale trvalo příliš dlouho, protože by bylo nutné vyzkoušet všechny permutace pořadí vektorů.

Tím, že nedochází k návratu k předchozím bitům, je výrazně omezen prohledávaný stavový prostor a dochází k výraznému zrychlení algoritmu. Algoritmus pracuje tak, že v každém kroku ohodnotí kritériem všechny možnosti, a vybere aktuální lokální minimum s tím, že existuje šance, že trvalým vybíráním lokálního minima bude nalezeno minimum globální. Jedná se tedy o hladový algoritmus (greedy search). Nalezené řešení není nejmenší možné, je ale dostatečně kvalitní.

3.5. Dekomprese

Dekomprese testu v kombinačních obvodech potom tak, že posloupnost bitů vstupuje po jednom bitu do řetězce klopných obvodů, tvořených např. Boundary

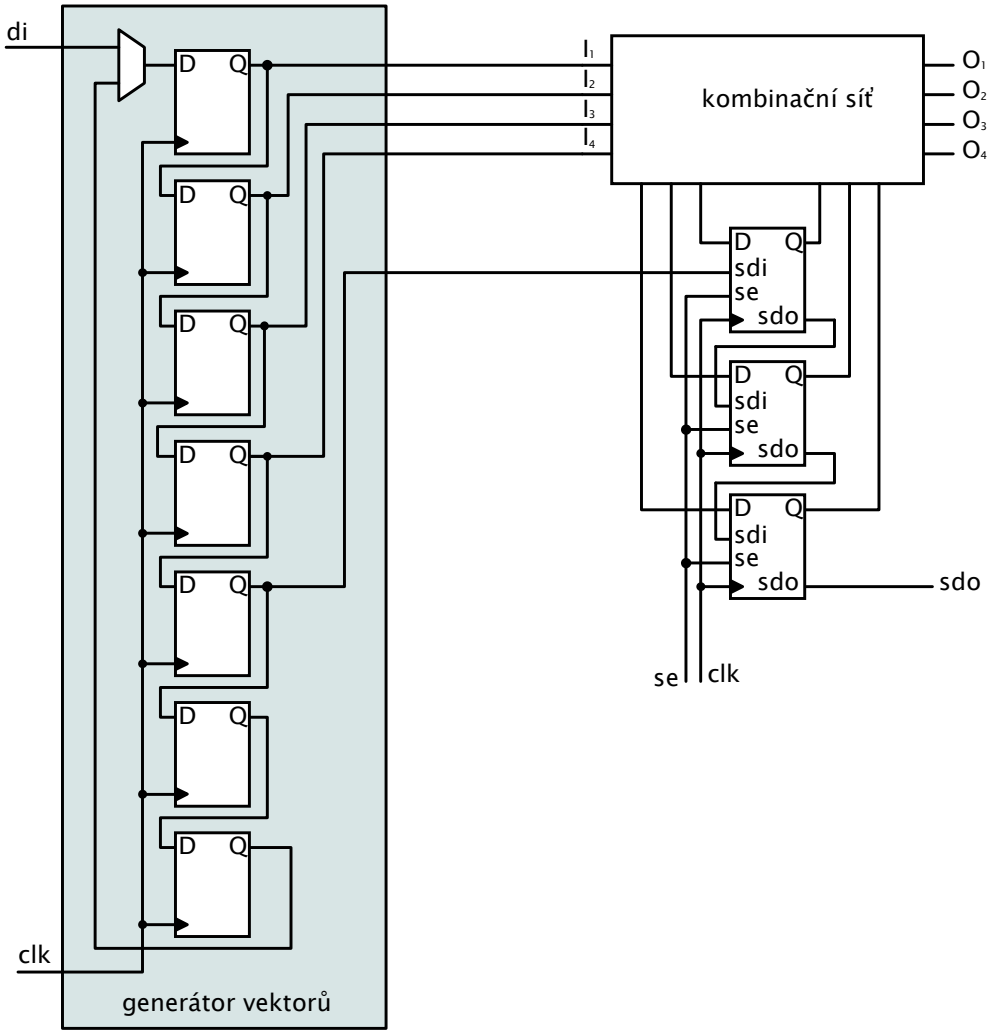
Scanem. Při posunutí posloupnosti o jeden bit je přitom vygenerován celý testovací vektor, čímž dochází ke značné úspoře paměti testeru.



Obrázek 15: Příklad generátoru vektorů pro kombinační obvod

Dekomprese při použití sekvenčního obvodu vyžaduje složitější dekompresní hardware, jedním z řešení je např. RESPIN architektura [26]. V obou případech je dekompresní hardware tvořen jednoduchým řetězcem klopných obvodů, viz Obrázek 15: Příklad generátoru vektorů pro kombinační obvod a Obrázek 16: Příklad generátoru vektorů pro sekvenční obvod.

Architektura RESPIN umožňuje dekomprimovat pomocí vestavěného testovacího jádra ETC výslednou testovací posloupnost na testovací vzorky pro testované jádro CUT, a je plně kompatibilní s normou IEEE P1500.



Obrázek 16: Příklad generátoru vektorů pro sekvenční obvod

3.6. Výhody

Výhodou této metody jsou především vysoká komprese použitých testovacích vektorů. Při kompresi se nevyužívá pouze kompakce testovacích vektorů, ale zároveň jejich posuv, což značně zvyšuje stupeň komprese. Dalšími výhodami jsou rychlé a snadné testování a velmi jednoduchý testovací hardware.

4. Zaměření disertační práce

4.1. Cíle

Prvním úkolem bylo z důvodu rychlosti integrovat do COMPASu simulátor poruch, protože spouštění externího simulátoru má příliš velkou režii.

Následným cílem bylo vyvinout lepší kompresní metodu a optimalizovat systém COMPAS pro dosažení maximální rychlosti komprese testovacích dat, zároveň je nutné minimalizovat délku výsledné testovací posloupnosti. Tím bude umožněna komprese testovacích dat pro velké obvody.

Dalším úkolem bylo přizpůsobit kompresní systém COMPAS více skenovacím řetězcům, které jsou v RESPINu používány. Větší obvody zpravidla pro testování používají více skenovacích řetězců, což je výhodné z hlediska rychlosti testu.

Dalším cílem bylo ověřit schopnost komprimovat testovací data i pro jiný model poruch než trvalé poruchy, a ověřit nezávislost kompresního algoritmu na použitém ATPG.

Posledním úkolem bylo zlepšit výkonnost COMPASu při využití moderních víceprocesorových výpočetních systémů, protože vícejádrové procesory se už staly standardním vybavením počítače.

Disertabilní jádro bude spočívat v novém kompresním algoritmu.

4.2. Původní stav COMPASu

Systém COMPAS využíval testovacích vektorů vygenerovaných automatickým generátorem testovacích vektorů ATALANTA [11] pro testování trvalých poruch. Pro simulaci poruch využíval program HOPE [11] ze stejného balíku. Především program ATALANTA je ale značně omezený, protože neumožňuje generovat testovací vektory ani pro středně velké obvody (30 tisíc hradel). Nebylo tedy zatím možné vyzkoušet metodu systému COMPAS na současných velkých

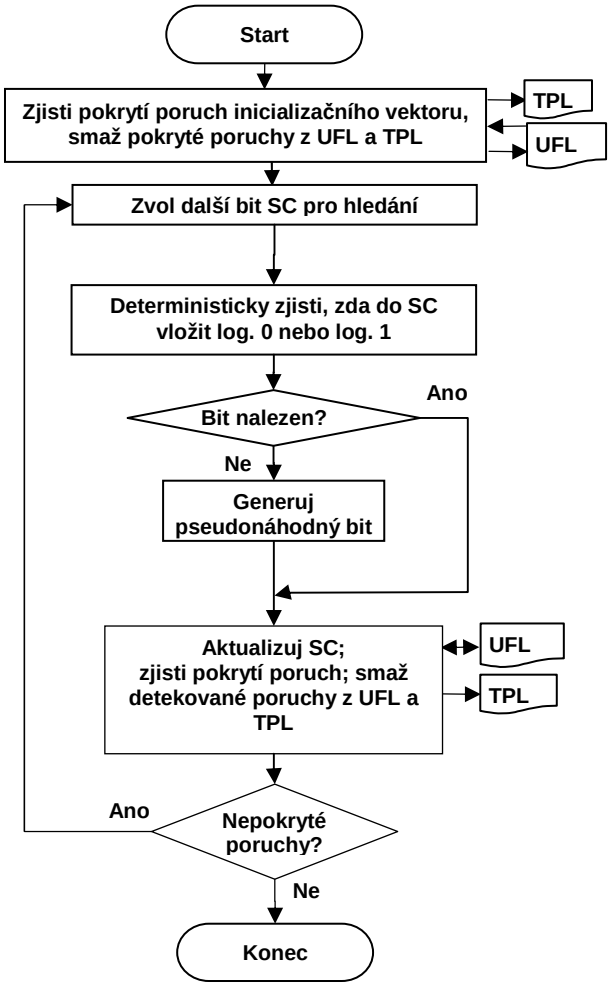
obvodech (více než 100 tisíc hradel). Samotný systém COMPAS je omezen v podstatě jen velikostí operační paměti.

Do operační paměti byla ukládána nekomprimovaná data ve dvojicích tak, že ke každé poruše náleží právě jeden testovací vektor. Ten byl uložen jako pole znaků „0“, „1“ a „X“, tak jak jsou načteny ze souboru vygenerovaného pomocí programu ATALANTA. V každém kroku byl vygenerován jeden bit výsledné komprimované testovací posloupnosti, vytvořeny soubory pro externí simulátor poruch HOPE, ten byl následně spuštěn, podle jeho textového výstupu odstraněny detekované poruchy a cyklus pokračoval, dokud zbývala alespoň jedna nedetekovaná porucha. Program využívá základní verzi algoritmu překrývání testovacích vektorů.

4.3. Stručný popis nového algoritmu

Prvním krokem je vygenerování souboru testovacích vektorů TPL (Test Pattern List) společně se souvisejícím seznamem nedetekovaných poruch UFL (Undetected Fault List) pro daný testovaný obvod. K tomu je nutné použít takový ATPG, který je schopen generovat nekompaktní testovací vektory. Pro každou poruchu, kterou chceme detekovat, je nutné vygenerovat nejméně jeden testovací vektor obsahující hodnoty ‘0’, ‘1’ a ‘X’, kde ‘X’ znamená nespecifikovanou hodnotu. Tímto způsobem je možné jednoznačně přiřadit kterému testovacímu vektoru náleží která porucha. Popis možností generování dat různými ATPG a jejich vliv na kompresní poměr je možné nalézt v kapitole 5.9. V kapitole 5.7 je popsán problém velkého množství vstupních dat a popsáno jeho řešení.

Dalším krokem provedeným před samotnou kompresí je analýza testovacích dat. Při ní jsou určeny obtížně detekovatelné poruchy a testovací vektory jsou ohodnoceny podle schopnosti detekovat obtížně detekovatelné poruchy. Využití této dodatečné informace vede k zlepšení kompresního poměru. Podrobný popis metody obsahuje kapitola 5.10.



Obrázek 17: Algoritmus komprese testovacích vektorů

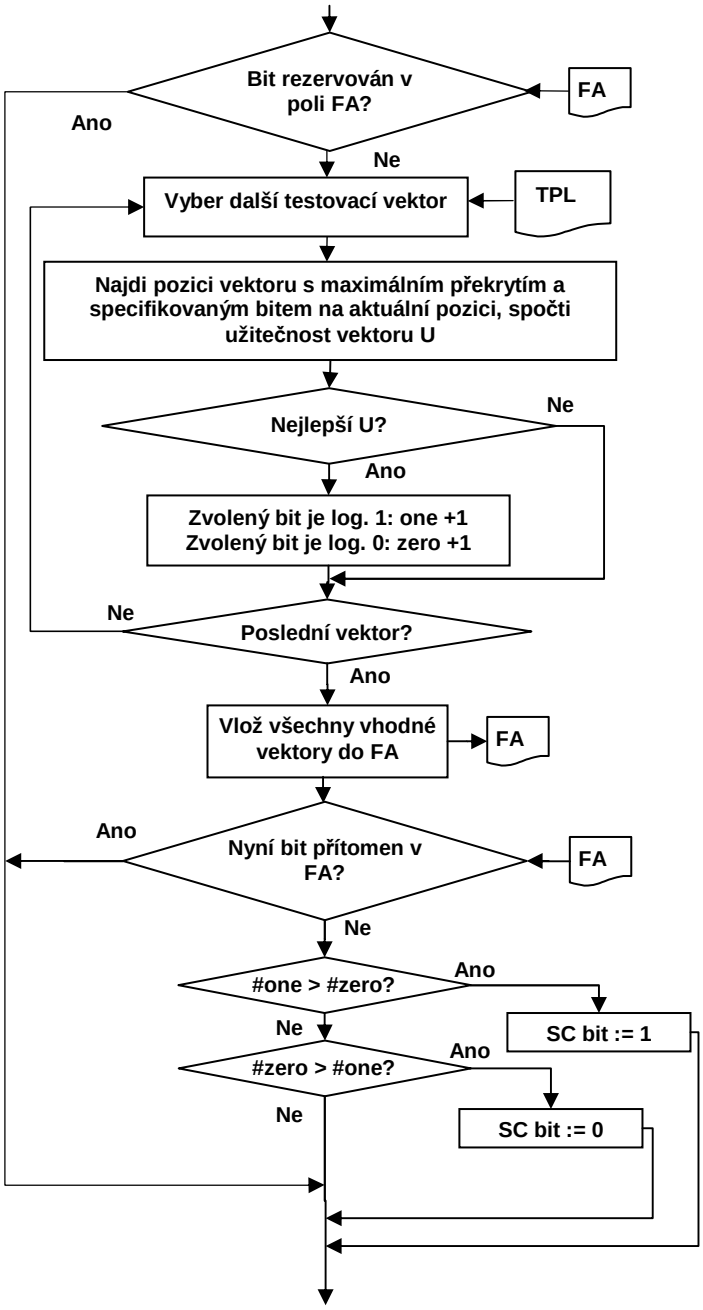
Hlavní cyklus algoritmu pro vyhledávání řetězce komprimovaných bitů, které budou následně uloženy do paměti ATE, popisuje Obrázek 17: Algoritmus komprese testovacích vektorů.

Předpokládejme (bez újmy na obecnosti), že skenovací řetězec je před samotným testováním vynulován, a první testovací vektor je tedy tvořen samými log. nulami (algoritmus umožňuje začít kompresi s libovolným známým stavem skenovacího řetězce). Tento první testovací vektor je odsimulován simulátorem poruch, detekované poruchy jsou odstraněny z UFL a jim příslušné testovací vektory jsou odstraněny z TPL.

Dále se algoritmus snaží zkomprimovat testovací data postupným překrýváním zbývajících testovacích vektorů s aktuálním stavem skenovacího řetězce. Algoritmus nejprve deterministicky hledá, jestli je jako další bit, který se má nasunout do skenovacího řetězce, vhodnější logická 1 nebo logická 0 (viz Obrázek 18: Detail deterministického hledání bitu).

Při hledání nového bitu je kontrolováno, jestli není kvůli dříve zvoleným testovacím vektorům nutné nastavit některé budoucí bity. Tyto bity jsou ukládány v poli budoucích hodnot FA společně s ohodnocením jejich užitečnosti a identifikačním číslem testovacího vektoru.

Pokud je v FA některá pozice rezervována pro konkrétní logickou hodnotu, je tato okamžitě použita jako výsledek a není nutné jí vypočítávat prohledáváním všech zbývajících testovacích vektorů. Podrobný popis metody předpovídání bitů posloupnosti obsahuje kapitola 5.3.



Obrázek 18: Detail deterministického hledání bitu

Pokud není v FA rezervován žádný bit, je nutné použít časově náročnější způsob pro rozhodnutí, zda je lepší zvolit jako výsledek log. 1 nebo log. 0. Algoritmus musí maximálně překrýt všechny zbývající testovací vektory s aktuálním stavem skenovacího řetězce. Z takto překrytých vektorů vyhledat všechny takové, které

na aktuální pozici bitu, který by měl být zvolen jako nový bit, má specifikovaný bit (tedy 1 nebo 0). Podrobnější popis překrývání vektorů poskytuje kapitola 5.8.

Poté co jsou vhodné vektory vybrány, jsou všechny ohodnoceny kritériem užitečnosti U, kde menší hodnota znamená lepší ohodnocení Volbou kritéria se zabývá kapitola 5.2.

Po ohodnocení testovacích vektorů je následně spočítán počet vektorů s nejlepší (tedy nejmenší) hodnotou U, které nabízí jako aktuální řešení log. 1 a počet vektorů, které nabízí log. 0. Pokud je počet vektorů s log. 1 větší než počet vektorů s log. 0, je jako řešení zvolena jednička. Pokud je menší, je zvolena nula. Tento způsob výběru řešení garantuje, že je vybráno a zakódováno největší množství nejužitečnějších testovacích vektorů. Výsledný bit a všechny ostatní specifikované bity náležící vybrané skupině vektorů jsou zapsány do pole FA obsahujícího budoucí řešení, takže následující vyhledávání bitů může být výrazně urychleno.

Pokud je počet překrytých vektorů se specifikovaným bitem na aktuální pozici nulový, nebo je jich shodný počet pro log. 1 i pro log. 0, není možné přímo rozhodnout, jaká hodnota by měla být zvolena jako výsledek. Jako výsledek je proto zvolena hodnota generovaná pseudonáhodným váženým generátorem kontrolovaným počtem jedniček a nul ve zbývajících dosud nezkomprimovaných testovacích vektorech. Metodu výběru pseudonáhodného bitu popisuje podrobněji kapitola 5.4.

Po zvolení výsledného bitu je tento nasunut do skenovacího řetězce, přičemž všechny stávající bity řetězce jsou posunuty o jednu pozici. Poté je spuštěna poruchová simulace. Detekované poruchy a jim příslušné testovací vektory jsou vymazány se seznamů TPL a UFL. Pokud již v UFL nebývá žádná nedetekovaná porucha, algoritmus končí. Vlivem simulátoru na výsledek a dobu komprese se zabývají kapitoly 5.5 a 5.6.

5. Rozbor nového algoritmu

Všechny části nového algoritmu popsané v následujících kapitolách jsem samostatně realizoval a experimentálně ověřil. Při návrhu metod v kapitolách 5.2, 5.3 a 5.4 částečně navazuji na výzkum ostatních členů týmu, především [33] a [49]. Nově navržené metody jsou ale kompletně přepracovány především z hlediska použitelnosti s rozsáhlými obvody (více než 200 tisíc hradel).

5.1. Testované obvody

Každá úprava v programu je opakovaně otestována na změny v rychlosti a v kompresním poměru. Pro testování jsou použity vybrané obvody testovací sady ISCAS85 [12], ISCAS89 [13] a ITC99 [14], celkem je testováno 83 různých obvodů ve velikostech od šesti hradel do přibližně 225 tisíc hradel.

5.2. Kritérium ohodnocení

Určuje vhodnost každého nekomprimovaného testovacího vektoru k nasazení do výsledné komprimované posloupnosti v aktuálním kroku kompresního algoritmu. Kritérium je vyjádřeno celým číslem větším než nula a je zvoleno tak, že menší hodnota značí užitečnější vektor. Čím nižší je tedy jeho hodnota, tím vyšší je šance, že vektor bude vybrán k nasazení do posloupnosti a že se prosadí jeho aktuální bit. Užitečnost testovacího vektoru U je spočítána podle následujícího vzorce:

$$U = t * (overlapped\_dontcares + shift) + all\_dontcares$$

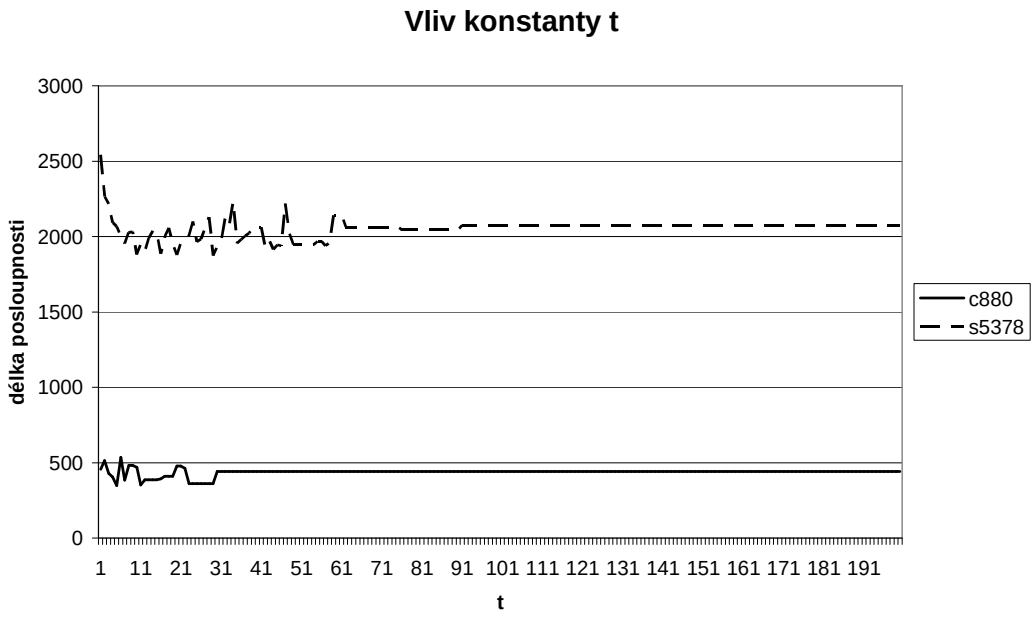
Kde značí:

overlapped_dontcares	počet nespecifikovaných bitů v překryté části testovacího vektoru
shift	počet nepřekrytých bitů vektoru
all_dontcares	celkový počet nespecifikovaných bitů v testovacím vektoru

t uživatelský parametr, dobrých výsledků je dosahováno při *t = počet_vstupů_obvodu / 2*

Kritérium obsahuje jediný uživatelsky volitelný parametr. Parametr by měl být volen tak, aby vhodně charakterizoval vlastnosti daného obvodu. Kritérium hodnotí jako vhodné ty vektory, které obsahují velké množství specifikovaných bitů a je tedy těžší je prosadit. Je totiž pravděpodobné, že při nasazení vektorů s vyšším počtem specifikovaných bitů budou pokryty i vektory více neurčité.

Parametr *t* pak hraje roli váhy důležitosti specifikovaných bitů v celém vektoru proti specifikovaným bitům v překryté části vektoru. Vhodnou volbou parametru *t* je možné dosáhnout vyšší komprese.



Graf 2: Vliv konstanty kritéria t na výslednou délku posloupnosti

5.2.1. Volba parametru t

Zpracováním všech tří sad testovacích obvodů (ISCAS 85, ISCAS 89, ITC 99) bylo zjištěno, že při nízkých hodnotách parametru *t* výsledná komprese značně kolísá, při postupném zvyšování jeho hodnoty se délka výsledné komprimované posloupnosti stabilizuje na přijatelné hodnotě a již se dále nemění. Tato

stabilizace nastává pro různé obvody v různý okamžik, s dostatečnou rezervou je však možné říct, že jako vhodná hodnota je počet vstupů obvodu dělený dvěma (Viz Graf 2: Vliv konstanty kritéria t na výslednou délku posloupnosti. Obvod c880 má 60 vstupů, obvod s5378 má 214 vstupů.).

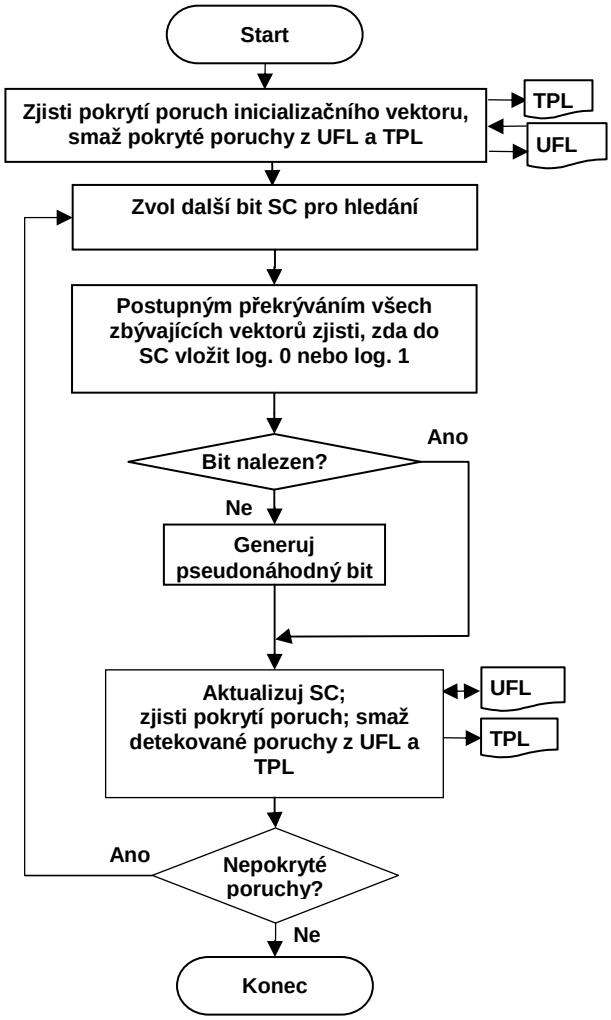
Parametr *t* je možné v případě nutnosti volit z příkazové řádky. Obvyklé ale je, že jeho hodnota není zadána, a program pak sám spočítá vstupy obvodu a nastaví správně jeho hodnotu.

Hledáním vhodného kritéria ohodnocení, volbou parametrů, jejich vlivem na výslednou kompresi, vztahem mezi velikostí obvodu a hodnotami parametrů a dalšími souvisejícími otázkami se zabývají především práce [15] a [49].

5.3. Predikce bitů komprimované posloupnosti

Hlavní myšlenkou metody předpovídání následujících bitů komprimované posloupnosti je využití informací dostupných v aktuálním kroku algoritmu i pro další kroky. Tím by mohlo být dosaženo jak zlepšení kompresního poměru z důvodu lepšího výběru vektorů, tak urychlení běhu programu.

Metoda spočívá v předpovědi budoucího řešení na základě aktuálního ohodnocení užitečnosti vektoru, tedy myšlence, že je-li vektor nejvhodnější v současném kroku, bude pravděpodobně nejvhodnějším i v krocích následujících. Pro lepší předpověď je bráno v úvahu větší množství vektorů a ne jen ten nejlepší. K nim jsou ukládány dodatečné informace jako pořadí nebo hodnota kritéria, které umožňují vybrat vhodný vektor.



Obrázek 19: Základní verze algoritmu

Tím, že jsou do výsledného řešení stabilně prosazovány hodnoty nejlepších vektorů, dochází ke zlepšení kompresního poměru. Vedlejším důsledkem zkrácení délky komprimované posloupnosti je, že program běží rychleji, protože ve svém průběhu spouští méněkrát simulátor poruch, a musí vyhledat méně bitů. Zrychlení ale nastává i jako přímý důsledek předpovídání bitů, protože v předchozích krocích předpovězený bit je využit bez výpočetně náročného překrývání všech zbývajících vektorů.

Hlavní části algoritmu předpovídání bitů ukazuje Obrázek 20: Kompresní algoritmus rozšířený o předpovídání bitů. Oproti kompresi bez predikce bitů (viz Obrázek 19: Základní verze algoritmu) jsou přidány následující kroky:

- *Údržba datových struktur*

  Posun pole s předpovězenými bity, vyřazení již neplatných vektorů, předčasné vyřazení vektorů, přepočítání kritéria ohodnocení atd.

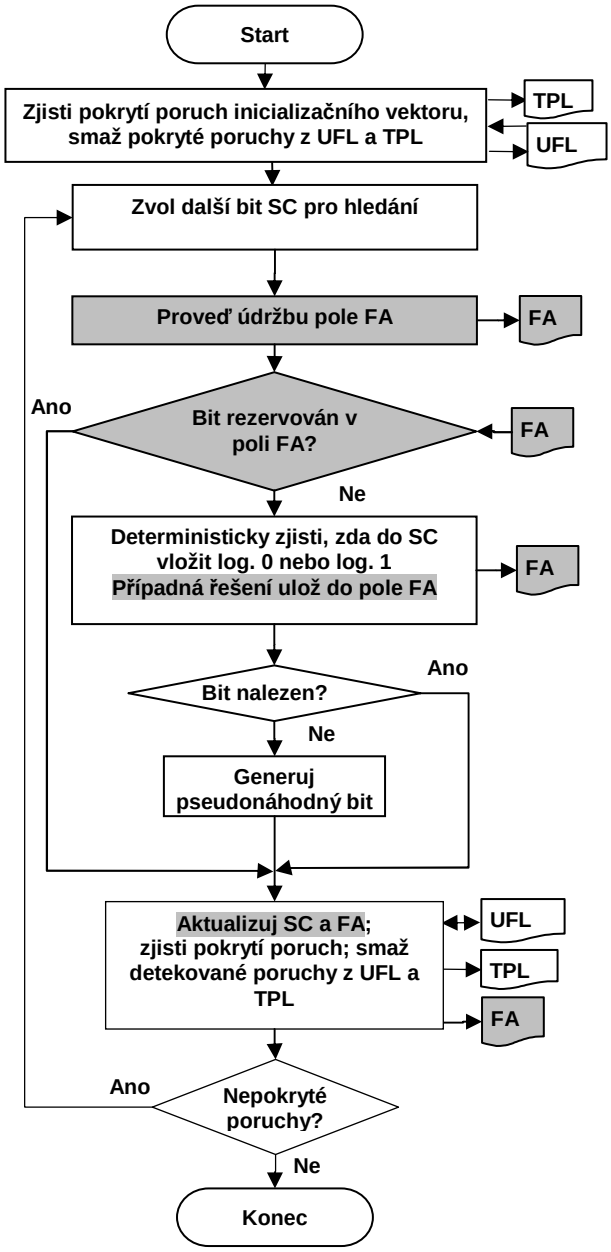
- *Výběr předpovězeného řešení*

  Je-li v aktuální pozici pole s budoucími bity specifikovaný bit, jsou všechny následující fáze vyhledávání přeskočeny, a bit je rovnou použitý jako výsledné řešení aktuálního kroku.

- *Doplnění zásobníku s budoucími bity*

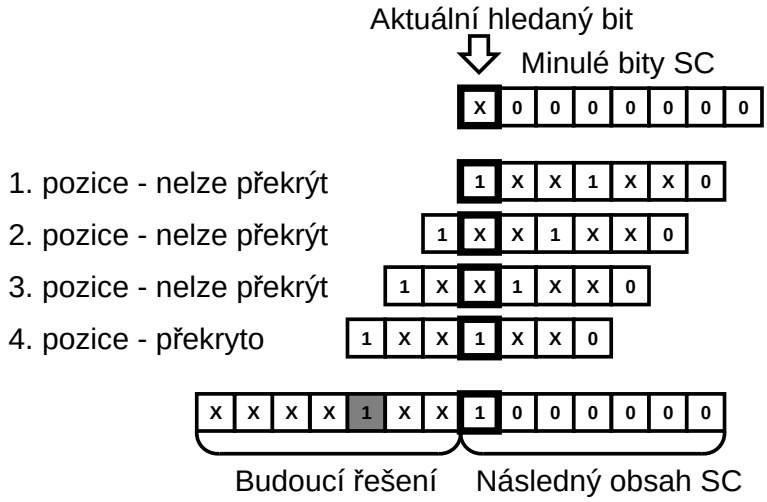
  Pokud není přítomen žádný předpovězený bit, dojde k ohodnocení všech zbývajících vektorů, a k jejich postupnému zařazení do zásobníku. Přitom je nutné ověřovat, zda nekolidují s již uloženými vektory. Pokud nastane kolize, je uložen vektor s lepším hodnocením.

Jednotlivé kroky budou později popsány v podkapitolách.



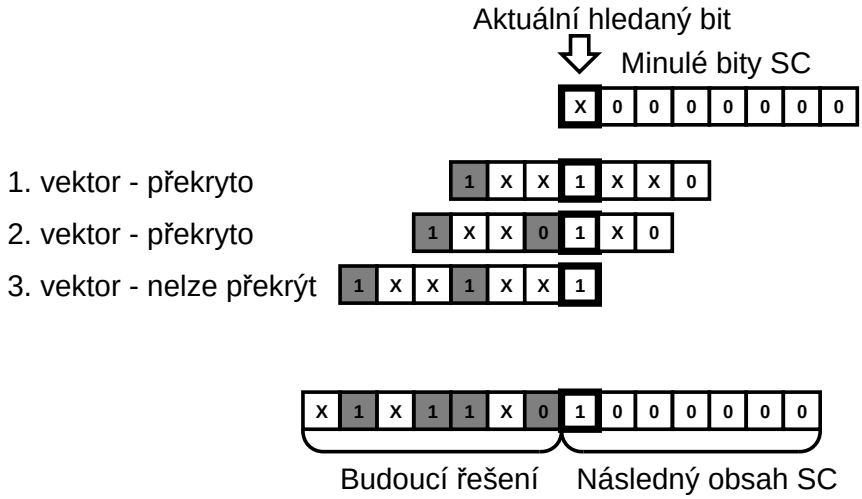
Obrázek 20: Kompresní algoritmus rozšířený o předpovídání bitů

Princip předpovídání bitů naznačuje Obrázek 21: Nalezení jednoho budoucího bitu. Na něm je v prvním kroku algoritmu (hodnoty skenovacího řetězce jsou vynulované) jeden z vektorů postupně překrýván s aktuálním stavem skenovacího řetězce tak dlouho, než je nalezeno platné překrytí bez kolize na pozici, která zároveň v aktuálním kroku nabízí specifikovaný bit. Následně jsou všechny budoucí bity zapsány do pole budoucích řešení.

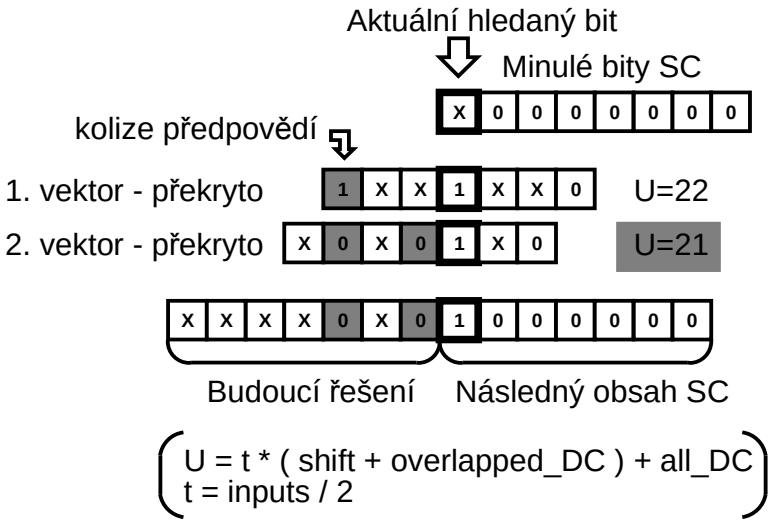


Obrázek 21: Nalezení jednoho budoucího bitu

Stejně se postupuje v případě, že je k dispozici více vektorů, všechny budoucí bity jsou postupně zapisovány do pole budoucích řešení (viz Obrázek 22: Nalezení budoucích bitů ze tří vektorů bez kolize).

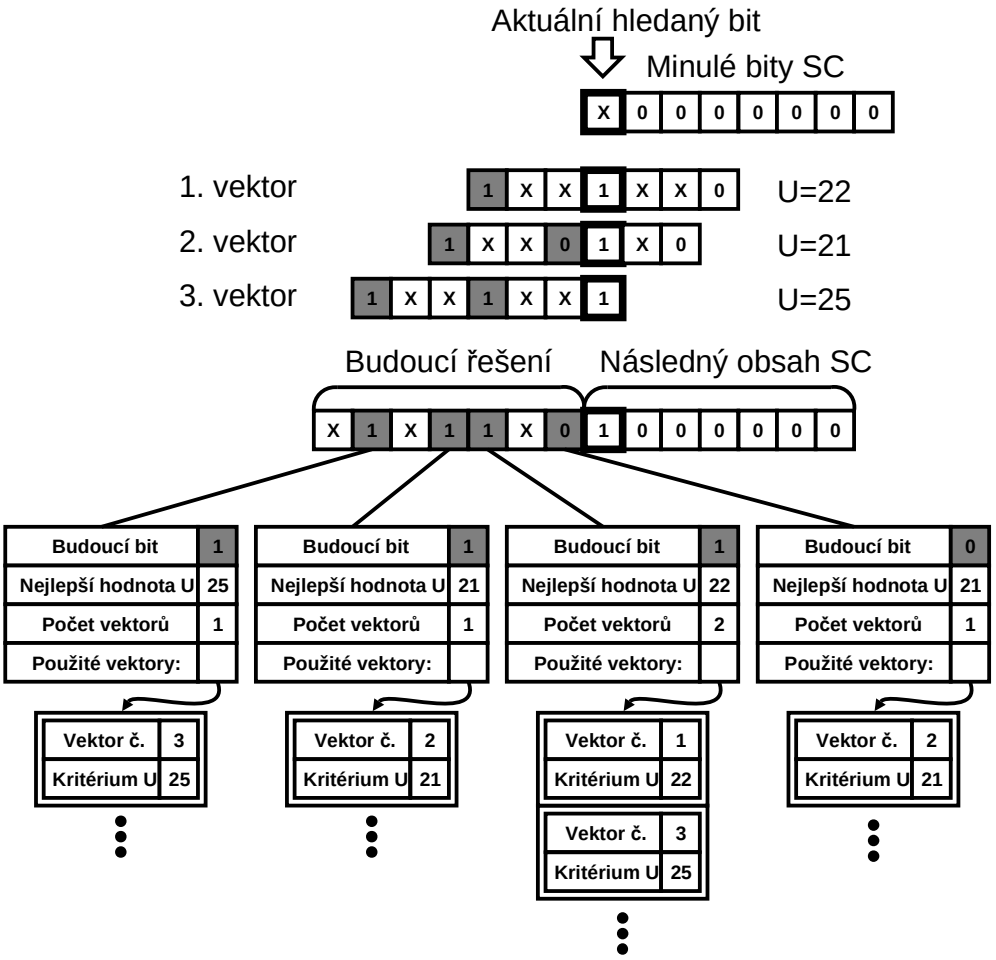


Obrázek 22: Nalezení budoucích bitů ze tří vektorů bez kolize



Obrázek 23: Kolize předpovídaných bitů

Pokud ale do pole budoucích řešení zařazujeme postupně více vektorů, může nastat kolize budoucích řešení, viz Obrázek 23: Kolize předpovídaných bitů. Potom je nutné podle kritéria ohodnocení zvolit výhodnější vektor, tedy vektor s menší hodnotou U, a přepsat a případně vymazat bity, které patří méně výhodnému vektoru. K tomu je nutné pamatovat si, které budoucí bity patří kterým vektorům, a jaké mají příslušné vektory aktuální ohodnocení. Kvůli rychlosti je také ukládáno nejlepší dosažené ohodnocení. Aby bylo možné rychle vyhledat, který vektor je výhodnější, jsou v paměti vektory seřazené vzestupně podle hodnoty kritéria. Datové struktury ilustruje Obrázek 24: Datové struktury pro předpovídání bitů.



Obrázek 24: Datové struktury pro předpovídání bitů

5.3.1. Údržba datových struktur

V každém kroku komprese je nejprve provedena údržba datových struktur souvisejících s předpovídáním bitů posloupnosti.

- Všechny vektory, které přispěly k řešení v aktuálním kroku, jsou otestovány, není-li zvolen jejich poslední bit. Pokud vektor přispěl svým posledním specifikovaným bitem, je zřejmé, že všechny jeho specifikované bity jsou již nasunuty ve skenovacím řetězci. Vektor tedy bude jistě v následujících krocích algoritmu ve skenovacím řetězci obsažen. Není tak potřeba čekat na simulátor poruch a potvrzení, že vektoru příslušná porucha bude detekována a je možné ho ihned odstranit.

Tato metoda tedy urychluje běh programu i v případě, že vůbec není použit simulátor poruch.

- Pole se celé posune o jeden bit, včetně všech navázaných struktur.
- V budoucnosti nejvzdálenější pozice, která se uvolnila posunem o jednu pozici, je nastavena na počáteční hodnoty, tj. nejhorší možné kritérium ohodnocení, neznámá hodnota bitu a nulový počet přispívajících vektorů.
- Úplně všem vektorům, nejen těm, co přispívají k budoucím řešením, je aktualizovaná hodnota kritéria. Protože je kritérium jednoduché, jedná se jen o případné odečtení konstanty. Je výhodnější aktualizovat všechny vektory a některým přitom odečíst z hodnocení, než aktualizaci kritéria odkládat, a v budoucích krocích algoritmu pak zdlouhavě vyhodnocovat překrývání vektorů.
- Pročištění datových struktur, kdy jsou odstraněny všechny vektory příslušné poruchám detekovaným simulátorem poruch. Vektory jsou následně opět seřazeny podle hodnoty kritéria.

5.3.2. Výběr řešení

Je-li v aktuální pozici pole s budoucími bity specifikovaný bit, je tento ihned zvolen jako platné řešení. Protože byl bit zařazen v poli budoucích řešení, je jisté, že nebude kolidovat s ostatními vektory. Je také jisté, že zvolit tento bit je nejvýhodnější, protože bity jsou ukládány podle kritéria ohodnocení.

Tím, že je možné okamžitě zvolit řešení, jsou kompenzovány časové ztráty související s údržbou datových struktur. Klasické překrývání všech vektorů s aktuálním skenovacím řetězcem je totiž časově mnohem náročnější.

5.3.3. Doplňování zásobníku s budoucími bity

Pokud není přítomen žádný předpovězený bit, dojde k ohodnocení všech zbývajících testovacích vektorů pomocí kritéria. Všechny vektory jsou tedy co nejvíce překryty s aktuálním stavem skenovacího řetězce a následně je vypočítáno

jejich kritérium užitečnosti. Zároveň je pro všechny vektory uchována hodnota posunutí oproti skenovacímu řetězci. Ohodnocené vektory jsou následně seřazeny vzestupně podle hodnoty kritéria, tedy od nejlepšího po nejhorší.

Seznam seřazených vektorů je pak předán k zařazení do pole případných budoucích řešení. Vektory jsou zařazovány postupně od nejlepšího po nejhorší, přičemž je kontrolována jeho použitelnost a případné kolize s jinými vektory. Pro každý vektor jsou provedeny následující činnosti:

- Je provedena kontrola, zda již vektor není zařazen v poli budoucích řešení. Pokud ano, dále se nepokračuje, protože není možné dosáhnout lepšího umístění. Vícenásobně zařazené vektory navíc mohou kolidovat samy se sebou a tím prodlužovat dobu výpočtu.

- Je zkontrolováno, jestli všechny specifikované bity právě zařazovaného vektoru nekolidují s již zařazenými bity. Pokud nenastane žádná kolize, jsou všechny specifikované bity vektoru zařazeny do pole budoucích řešení. Zároveň je uloženo i jejich kritérium ohodnocení. Pro urychlení pozdějších porovnání je seznam vektorů seřazen podle hodnoty kritéria.

- Pokud kolize nastane, jsou možné tři případy:

    - Bit je z vektoru s horším ohodnocením, než už zařazený bit. Vektor by tedy nepřinesl kvalitnější řešení, a proto není zařazen.

    - Bit je z vektoru se stejným ohodnocením, než už zařazený bit. Ani v tomto případě by vektor nepřinesl lepší řešení, navíc by narušil již započatou posloupnost do této chvíle nejlepšího vektoru. Vektor proto není zařazen.

    - Bit je z vektoru s lepším ohodnocením. Tento případ je nejsložitější. Protože všechny ostatní vektory na aktuální pozici mají horší hodnocení, je nutné je odstranit. Navíc je nutné projít celé pole, a odstranit všechny výskyty těchto vektorů, ne jen na aktuální pozici, protože zbylé bity odstraňovaných vektorů by

nevhodně ovlivňovaly řešení. Následně je hodnota bitu v aktuální pozici změněna, a jako jediný prvek seznamu je vložen aktuálně zařazovaný vektor.

Výpočetně nejnáročnější částí při zařazování budoucích řešení je kolize bitů, kdy je nový vektor výhodnější než už uložené řešení. Aby se tomuto případu maximálně předcházelo, jsou vektory zařazovány od nejlépe hodnoceného k nejhoršímu. Tím je zaručeno, že nejlepší vektory hned na začátku obsadí svoje pozice budoucích řešení, a další vektory už je nemohou přepsat, nanejvýš mohou doplňovat dosud neobsazené pozice.

5.3.4. Velikost zásobníku

Na kvalitu komprese má vliv velikost zásobníku budoucích řešení. Při příliš malé velikosti se zásobník rychle vyčerpá a nebude tedy schopen nabízet vhodná řešení. To vede jednak zhoršení kompresního poměru, tak k prodloužení doby komprese. Příliš velké hodnoty zásobníku jsou sice paměťově náročnější, ale na vlastní kvalitu komprese nemají negativní vliv. Zásobník nanejvýš nebude plně využitý. Množství vektorů, které lze bez kolize zařadit je omezené, a proto je zbytečné volit příliš velkou hodnotu zásobníku. Od určité hodnoty již do zásobníku není možné zařadit další vektory kvůli kolizím, jeho zvětšování tedy nezlepšuje výsledky.

Tabulka 2: Počet budoucích řešení, které se nevešly do zásobníku

Obvod	Velikost zásobníku								
	1	2	5	10	20	50	100	200	500
s15850	342	338	211	75	5	0	0	0	0
s38417	18	19	16	2	0	0	0	0	0
s35932	991	784	551	375	226	18	0	0	0
s38584	827	1204	1061	660	285	3	0	0	0

Jako vhodná velikost zásobníku byla zvolena hodnota 500 vektorů, která již neomezuje ani největší obvody. Při běhu programu je možné nechat vypsat dodatečné ladicí informace, které ukazují obsazenost zásobníku, četnost kolizí

apod. V případě nutnosti je tedy možné velikost zásobníku vyladit přímo pro daný obvod.

5.4. Náhodné bity

Při hledání dalšího komprimovaného bitu může nastat případ, že řešení nebude nalezeno v poli budoucích řešení. Pokud není možné získat řešení ani pomocí překrývání všech vektorů, je nutné zvolit bit náhodně, protože do skenovacího řetězce nesmí vstoupit nespecifikovaná hodnota.

Zcela náhodná volba bitu ale není vhodná, protože pro každý běh programu mohou vyjít různé výsledky. Nelze pak rozumně porovnávat kvalitu komprese ani její dobu. To lze vyřešit např. tak, že se za náhodnou hodnotu bude dosazovat vždy log. 0 nebo log. 1, případně poslední použitá hodnota. Ani jedna z těchto metod ale nebere přímo ohled na komprimovaná data.

Proto byla pro výběr pseudonáhodného bitu navržena metoda, která spočítá poměr všech log. 0 vůči všem log. 1 ve zbývajících, dosud nezkomprimovaných datech. Pokud je ve zbývajících datech více log. 1 a přijde požadavek na pseudonáhodnou hodnotu, je vrácena log. 1 (a naopak). Protože jsou tak do skenovacího řetězce umisťovány hodnoty, u kterých je větší pravděpodobnost výskytu, teoreticky může dojít ke zlepšení komprese. Poměr zbývajících jedniček a nul je v průběhu programu přepočítáván tak, aby odpovídal aktuálním datům.

Tabulka 3: Závislost délky komprimované posloupnosti na volbě náhodných bitů

Obvod	Volba pseudonáhodného bitu			
	Vždy 0	Vždy 1	Zopakuj poslední	Poměr zbývajících 1 a 0
s15850	7903	8941	7403	7909
s38417	2117	2117	2117	2117
s35932	25850	21668	23107	28496
s38584	12058	13711	11213	11873

Jako metoda volby pseudonáhodného bitu byla použita metoda počítající poměr specifikovaných bitů v zbývajících nekomprimovaných datech, přestože

nedosahuje nejlepších výsledků. Jako jediná totiž bere přímý ohled na zbývající nezkomprimovaná data. Všechny navržené metody odstraňují náhodnost volby bitu, komprimovaná sekvence je tedy vždy závislá nanejvýš na vstupních datech. Měření je tak při stejných vstupních datech opakovatelné se zachováním výsledků na různých operačních systémech, kompilátorech apod.

5.5. Interní simulátor

Výpočet komprimované testovací posloupnosti fungoval, ale trval velmi dlouho. Velkou nevýhodou byla zdlouhavá příprava dat pro externí simulátor poruch, jeho spuštění pro simulaci jediného testovacího vektoru, a následná analýza výstupních dat, jak bylo popsáno výše. Tento nedostatek je obtížně odstranitelný, neboť kvalitní simulátor poruch je velice obtížné získat se zdrojovými kódy a souhlasem autora k dalšímu použití. Vestavěním simulátoru poruch do systému COMPAS na úrovni zdrojových kódů je však možné dosáhnout značné časové úspory.

Původně byl simulátor spouštěn velmi neefektivním způsobem: v každém jednotlivém kroku byl vytvořen dávkový soubor s parametry pro simulátor, druhý soubor obsahoval seznam dosud nedetekovaných poruch, v třetím byl jediný testovací vektor. Dávkový soubor byl pak pomocí funkce *system()* spuštěn, následně se spustil simulátor poruch HOPE a teprve ten vykonával užitečnou činnost. Tato verze programu komprimovala např. data obvodu s15850 přes 76 hodin, což je nepřijatelné.

Jako počáteční optimalizace bylo zvoleno občasné vynechávání spouštění simulátoru (v případech, kdy bylo zřejmé, že jeho spuštění nepovede k výraznému zhoršení kompresního poměru). Tato metoda sice program výrazně zrychluje, ale většinou také mírně prodlužuje výslednou testovací posloupnost, protože některé z vektorů, které by již jinak byly detekovány simulátorem poruch, stále zbytečně ovlivňují výsledné řešení. Tabulka 4 v druhém sloupci obsahuje výslednou délku komprimované testovací posloupnosti a ve čtvrtém sloupci čas potřebný pro kompresi, pokud použijeme spouštění externího simulátoru poruch HOPE s občasným vynecháním. Třetí sloupec ukazuje čas verze bez vynechávání.

Výsledky měřeny na Intel P4 na 3 GHz. Pro původní verzi program doběhnul pouze v jednom případě, v ostatních byl po cca sto hodinách běhu zastaven, protože je zřejmá jeho nepoužitelnost pro takto rozsáhlý obvod.

Tabulka 4: Výsledky verze s občasným vynecháváním simulace

Obvod	Občasné vynechávání simulace [bitů]	Doba běhu bez vynechávání simulace	Doba běhu s vynecháváním simulace
s15850	7543	cca 76h	2m 32s
s38417	21664	více než 100h	32m 12s
s35932	1861	více než 100h	9m 1s
s38584	7165	více než 100h	13m 56s

Jako vestavěný simulátor poruch byl nejprve zvolen simulátor FSIM [11] z balíku ATALANTA (trvalé poruchy, paralelní simulace testovacích vektorů), spouštěný po vygenerování každého jednotlivého bitu komprimované testovací posloupnosti. Simulátor je plně začleněn do systému COMPAS, je tedy součástí stejného spustitelného souboru.

Vestavěním simulátoru poruch byla zbytečná režie v podobě opakovaného zápisu, čtení a spouštění dalšího procesu odstraněna, navíc simulace probíhá v každém kroku, nedochází proto k degradaci kompresního poměru, protože jsou poruchy detekovány bez zpoždění několika kroků. Jako simulátor poruch byl vybrán simulátor FSIM z balíku ATALANTA. Spojení COMPASu s FSIMem není vyřešeno jako simulovaný zápis do souboru a jeho následné jakoby čtení, spojení je skutečně na úrovni vnitřních datových struktur a funkcí, algoritmy navzájem využívají přímo interní datové struktury druhé části, obě části jsou v jediném spustitelném souboru.

Tím, že se kompletně odstranil jakýkoli zápis do souborů, neustálé opakované načítání popisu obvodu, připravování struktury obvodu a vytváření a ukončování procesu simulátoru, bylo dosaženo značného urychlení v souvislosti se simulací poruch.

Vestavění simulátoru FSIM ale přináší i potenciální nevýhody. Jednou je nesnadná případná záměna simulátoru poruch za jiný. Pokud je simulátor spouštěn externě a zamění se za jiný, stačí upravit kód ukládání a načítání zpravidla textového vstupního a výstupního souboru. Je-li ovšem integrován, je nutné znovu analyzovat kód jak simulátoru, tak COMPASu a najít místa, kde by oba programy bylo možné spojit. Pro vestavění simulátoru je také nutné mít k dispozici jeho zdrojové kódy, to je vzhledem k licenčním požadavkům obtížné.

Tabulka 5: Výsledky verze s integrovaným simulátorem oproti občasnému vynechávání

Obvod	Bitů	Čas [s]	Zrychlení
s15850	6797	123	1,2x
s38417	20037	1137	1,7x
s35932	1861	148	3,7x
s38584	6671	426	2,0x

Uvedená optimalizace byla úspěšně implementována a experimentálně ověřena. Čas nutný pro kompresi se podařilo výrazně zkrátit, přičemž zůstal zachován výborný kompresní poměr (i v porovnání s ostatními metodami komprese testovacích vektorů).

5.6. Běh bez simulátoru poruch

Běh programu bez simulátoru poruch může být vynucen z příkazové řádky. Tato vlastnost je užitečná pro kompresi libovolných obecných vektorů, vektory mohou být vytvořeny i pro testování jiných než trvalých poruch. Simulátor poruch také není možné použít, pokud není k dispozici popis obvodu. Algoritmus se následně snaží maximálně komprimovat posloupnost prostým vzájemným překrýváním vektorů. Protože ale není možné využít úzké spolupráce se simulátorem poruch, je dosaženo horší komprese. Komprese bez použití simulátoru poruch má ale i výhodu: každý nekomprimovaný vektor je zcela jistě přítomen ve výsledném komprimovaném řetězci. Pokud není použit simulátor poruch, kompresní algoritmus je bezeztrátový jak z hlediska detekovaných poruch, tak z pohledu testovacích vektorů obsažených v komprimované sekvenci..

Tabulka 6: Příklad komprese dat pro IDDQ test ukazuje výsledky komprese dat pro IDDQ test. Data byla vygenerována pomocí ATPG MILEF. Protože COMPAS nemá k dispozici simulátor pro IDDQ model poruch, byl spuštěn bez použití simulátoru, a bylo tak využito prosté překrývání nekomprimovaných testovacích vektorů. Přestože není simulace poruch použita, dosahuje komprese velmi dobrých výsledků.

Tabulka 6: Příklad komprese dat pro IDDQ test

Obvod	Množství spec. bitů	Komprimovaná délka [b]	Čas komprese [s]
s15850	0,81%	2392	1,3
s38417	0,25%	3873	13,4
s35932	0,20%	1831	6,9
s38584	0,27%	7175	18,8

Tabulka 7: Příklad komprese dat bez použití simulátoru poruch srovnává spuštění COMPASu se simulací poruch a bez simulace. Komprimována data jsou vygenerovaná pomocí ATPG ATALANTA a slouží pro test trvalých poruch. Z výsledků je zřejmé, že použití simulátoru poruch značně zlepšuje výsledky komprese. Je možné předpokládat, že ke stejně výraznému zlepšení by došlo, pokud by byl použit vhodný simulátor poruch při kompresi dat pro IDDQ test.

Tabulka 7: Příklad komprese dat bez použití simulátoru poruch

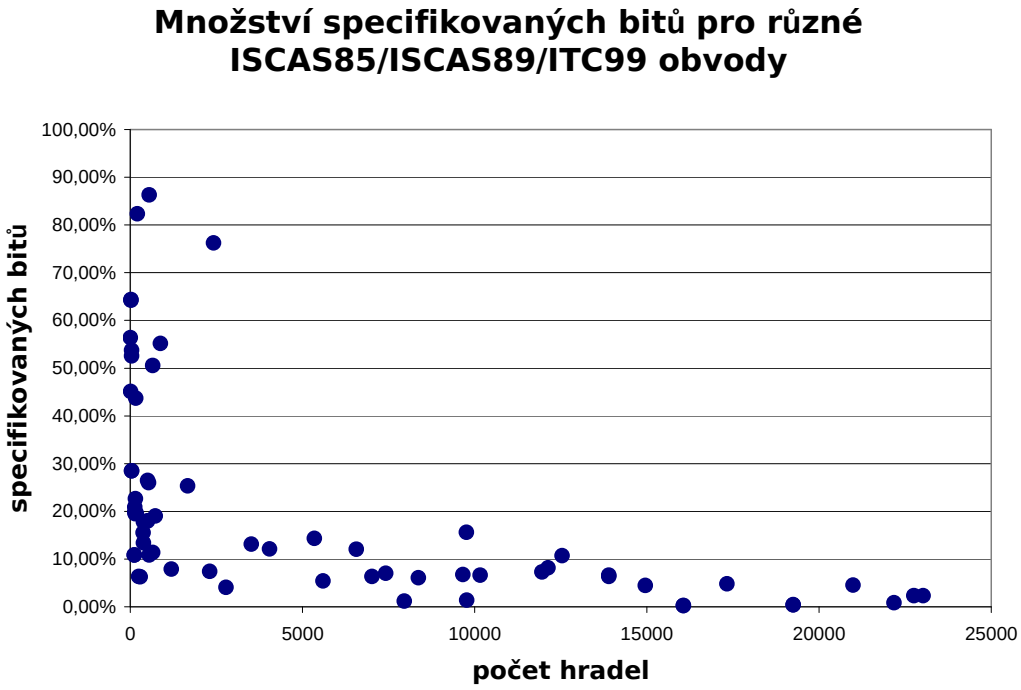
Obvod	Množství spec. bitů	Komprimovaná délka se simulací [b]	Komprimovaná délka bez simulace [b]
s15850	1,38%	1872	13223
s38417	0,84%	21557	88378
s35932	0,26%	1962	2092
s38584	0,39%	11685	14754

5.7. Množství dat u velkých obvodů

Překonat původní omezení programu ATALANTA, který umožňoval generovat testovací vektory pouze pro malé obvody (do 30 tisíc hradel) se podařilo použitím programu ATALANTA-M [32] odvozeného z původního ATALANTA Ing.

Petrem Fišerem. Tento program jsem pro svoje účely musel dále upravit. Bylo potřeba změnit formát výstupu, a především vyřešit nadměrné paměťové nároky. V současné době je tak možné generovat testy i pro obvody se stovkami tisíc hradel (větší nemáme v současné době k dispozici).

Do operační paměti byla ukládána nekomprimovaná data. To se ukázalo jako problém, protože např. pro největší obvod z testovací sady ITC99 je vygenerováno přes 6 GB testovacích dat. V současné době je v paměti uložen pro jednu poruchu jen jeden testovací vektor. Pokud by jich bylo více, bude mít algoritmus teoreticky větší šanci najít vhodné uspořádání vektorů a tak dosáhnout lepších kompresních poměrů.



Graf 3: Množství specifikovaných bitů pro různé obvody ISCAS85, ISCAS89 a ITC99

5.7.1. Motivace

V práci [34] bylo zjištěno, že využití více testovacích vektorů významně zlepšuje kompresi, stejný předpoklad je možné učinit i u COMPASu. Odhad objemu nekomprimovaných testovacích dat ovšem vychází pro větší obvody (přes sto tisíc

hradel) a při použití třeba jen deseti různých vektorů na jednu poruchu v desítkách gigabytů. Je proto nutné použít lepší metodu ukládání testovacích dat v paměti počítače, aby bylo vůbec možné více vektorů využít. S tím souvisí i časová náročnost algoritmu. Je nutné program dostatečně urychlit, aby se doba výpočtu se zvýšeným množstvím vektorů neúnosně neprodloužila.

Pro optimální rozhodování kompresního algoritmu je nutné načíst všechna data najednou do operační paměti počítače, proto je nutné pro ukládání dat do paměti vyvinout novou metodu. Přímé nahrání datového souboru není kvůli jeho velikosti pro velké obvody vhodné.

5.7.2. Metody kódování

Místo jednoduchého načtení dat musí být proveden ještě jedna fáze komprese. První fáze zkomprimuje surová data načtená ze souboru a uloží je do operační paměti. Druhá fáze následně využívá už částečně komprimovaná data z paměti ke kompresi pomocí překrývání testovacích vzorků.

V programu jsou použity tři rozdílné metody ukládání testovacích dat. Data vygenerovaná pomocí ATPG jsou na disku uložena jako čistý text tvořený znaky ‘0’, ‘1’ a ‘X’. Každý znak je na disku uložen jako osmibitový typ *char* (pokud použijeme popis v jazyce C). Přímé načtení těchto dat do paměti je první možností uložení dat v paměti. Další dvě možnosti využívají zakódování dat do komprimovaného tvaru.

První způsob zakódování je lepší zakódování ternární abecedy tvořené písmeny ‘01X’ tak, že každý znak není uložen v osmi bitech ale jen ve dvou, tedy přímý převod osmibitového znaku na dva bity. Dva bity umožňují zakódovat čtyři různé kombinace, tři jsou využity pro zakódování znaků ‘01X’, čtvrtá možnost je nevyužitá. Tímto kódováním, nazývejme ho ternární kódování, je každý znak, který původně zabíral 8 bitů převeden na 2 bity, je tedy ušetřeno 6 bitů (tj. 75% paměti) bez ohledu na vstupní data.

Druhý způsob kódování vytváří řídké vektory, což znamená, že do paměti jsou ukládány pouze specifikované bity a jejich pozice. Jejich použití je možné vzhledem k množství nespecifikovaných bitů v testovacích datech. Graf 3: Množství specifikovaných bitů pro různé obvody ISCAS85, ISCAS89 a ITC99 zobrazuje závislost množství specifikovaných bitů na velikosti obvodu. Je vidět, že nespecifikovaných bitů je s výjimkou malých obvodů velké množství, které se navíc ještě zvyšuje s rostoucí velikostí obvodu.

Nespecifikované bity nejsou při využití řídkých vektorů vůbec do paměti ukládány. Jako základní datový typ prvku řídkého vektoru je možné použít libovolný celočíselný datový typ. Z tohoto základního typu je pak vyhrazen jeden bit pro uchování skutečné hodnoty bitu a zbytek je použit pro uložení pozice bitu v původním nekomprimovaném testovacím vektoru.

Tabulka 8: Rozsahy datových typů

Datový typ	*Velikost datového typu*	*Rozsah*	*Max. délka vektoru*
unsigned char	1 Byte (8bitů)	0-255	127
unsigned short int	2 Byty (16 bitů)	0-65'535	32'767
unsigned int	4 Byty (32bitů)	0-4'294'967'295	2'147'483'647

5.7.3. Volba datového typu

Při volbě základního datového typu je třeba vzít v úvahu jeho rozsah, aby bylo možné zpracovávat i větší obvody s délkou testovacích vektorů v tisících bitů. Tabulka 8 uvádí některé základní celočíselné bezznaménkové datové typy, jejich velikost v paměti, úplný rozsah a maximální možná délka testovacího vektoru, kterou je možné pomocí nich zakódovat. Volba základního datového typu je důležitá kvůli výsledné efektivitě řídkého vektoru.

Zakódovat vektor do řídkého tvaru se vyplatí pouze pokud obsahuje dostatečně malé množství specifikovaných bitů, přesněji pokud je splněna nerovnost:

$$len * 8 > (cares + 1) * type\_bits$$

Kde značí:

len délka nekomprimovaného testovacího vektoru

cares počet specifikovaných bitů v testovacím vektoru

type_bits počet bitů pro uložení s použitím zvoleného základního datového typu

Přičtení jedničky v pravé části nerovnosti je nutné kvůli uložení celkového počtu specifikovaných bitů v daném vektoru. Pro tuto hodnotu je použitý stejný datový typ jako pro jednotlivé prvky řídkého vektoru.

Jako základní datový typ byl s ohledem na rozsah (viz Tabulka 8) zvolen *short int*, protože poskytuje nejmenší rozsah dostatečný i pro větší obvody. Nepředpokládá se tedy obvod se skenovacím řetězcem delším než 32767 buněk. Pokud by takový případ nastal, je možné základní datový typ jednoduše změnit na *int*, který umožňuje skenovací řetězec o délce přes dvě miliardy buněk.

5.7.4. Volba kódování

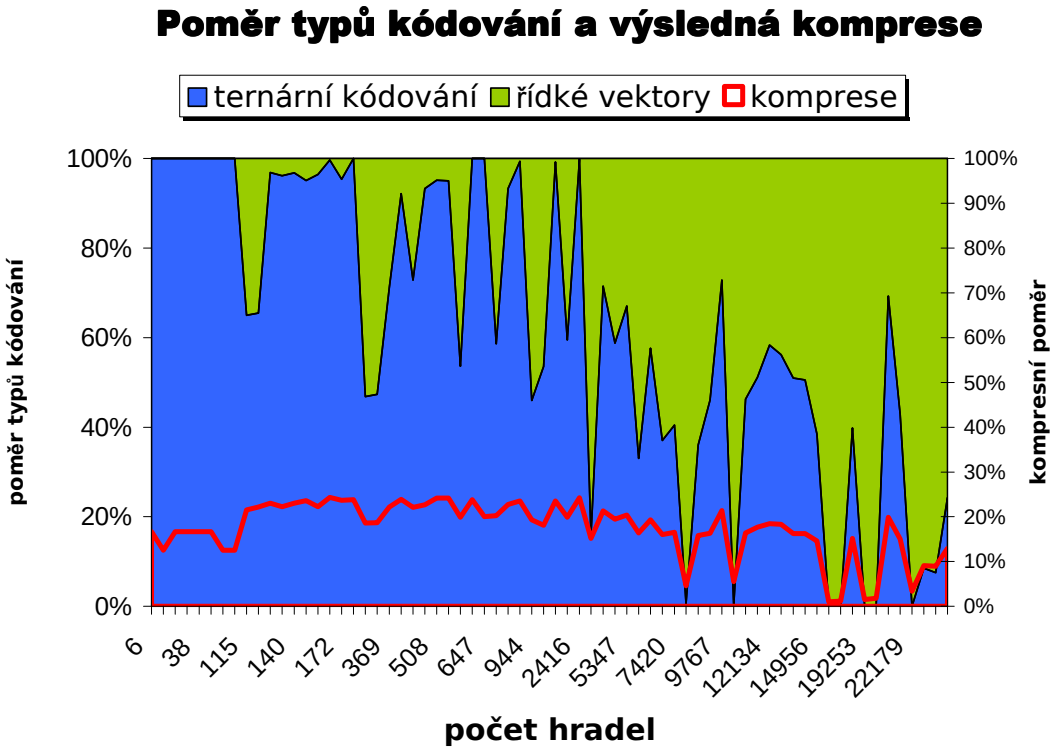
Jelikož jsou v řídkém vektoru pro zakódování jednoho specifikovaného bitu využívány dva byty namísto jednoho, je tato kompresní metoda ve srovnání s přímým použitím nekomprimovaných dat vhodná jen pokud je množství specifikovaných bitů menší než 50%, jinak zakódovaný vektor zabere v operační paměti více místa než nezakódovaný. Dále je nutné vzít v úvahu, že ternární kódování s jistotou snižuje množství dat na 25%. Pokud tedy má být výhodnější vektor uložit v řídké formě než v ternárním kódování, musí mít méně než 12,5% specifikovaných bitů. Nerovnost je tedy možné upravit na:

$$len * 2 > (cares + 1) * type\_bits$$

Graf 3: Množství specifikovaných bitů pro různé obvody ISCAS85, ISCAS89 a ITC99 ukazuje, že větší obvody tento požadavek s rezervou splňují.

Protože efektivita komprese u řídkých vektorů je závislá na načtených datech, je způsob kódování dynamicky zvolen tak, aby bylo množství spotřebované paměti minimální. Po analýze dat je možné zvolit kódování jednotně buď pro celou

testovací sadu vektorů, nebo i pro každý vektor zvlášť. Zvolené metody kódování garantují minimálně 75% úsporu paměti. Pro větší obvody, které mají v testovacích datech více nespecifikovaných bitů, úspora dosahuje běžně přes 95% (viz Graf 4: Poměr počtu vektorů kódovaných různými metodami a výsledná komprese).



Graf 4: Poměr počtu vektorů kódovaných různými metodami a výsledná komprese

Kódování bitů s sebou sice nese jistou výpočetní režii, ta je ale zanedbatelná a nepodařilo se ji změřit, protože se pohybuje pod hranicí statistické chyby.

5.8. Urychlení vynecháváním výpočtů

Rychlost výpočtu je možné významně urychlit vynecháním některých výpočetních kroků, je-li zřejmé, že jejich vykonání není přínosné.

5.8.1. Překrývání vektorů

Tabulka 9 zobrazuje možný stav algoritmu při kompresi, pro jednoduchost je uvažována základní verze algoritmu bez použití predikce budoucích bitů, protože princip optimalizace zůstává stejný.

Při hledání dalšího bitu komprimované testovací posloupnosti (vyznačen otazníkem) jsou postupně ověřovány všechny zbylé nekomprimované testovací vektory na maximální překrytí s aktuálním stavem.

V tomto okamžiku byly nalezeny tři vektory, které se překrývají s doposud vygenerovanou posloupností, jeden vektor (D) se překrýt nepodařilo. Protože v komprimované posloupnosti není možné uložit nespecifikovaný bit, připadají při volbě dalšího bitu v úvahu pouze vektory B, C a D, které shodně vyžadují použít hodnotu „1“.

Tabulka 9: Příklad platného překrytí vektorů

	aktuální bit											
	budoucí bity						vygenerované bity					
takt	5	4	3	2	1	0						
obsah SC						?	0	0	1	1	1	0
vektor A			1	X	X	X	0	0				
vektor B			1	X	X	1	0	0				
vektor C					X	1	X	0	1	1		
vektor D	0	X	1	1	1	1						

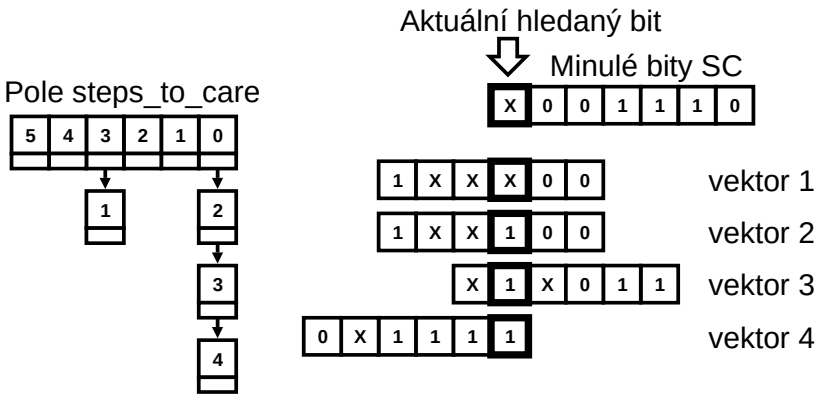
Tento bit je tedy zvolen jako výsledné řešení, uložen do komprimované testovací posloupnosti. Následně je spuštěn simulátor poruch pro aktuální stav skenovacího řetězce a detekované poruchy a jim odpovídající testovací vektory jsou odstraněny z paměti. Následně se algoritmus vrací na počátek, kde by měl opět pro každý zbývající vektor zjistit jeho aktuální překrytí.

5.8.2. Motivace

Důležité je uvědomit si, že vektor A nabízí jako svoje řešení v aktuálním taktu 0 a dvou následujících (1. a 2.) jen nepoužitelné nespecifikované bity. Tyto nejen že nemůžou přispět k řešení, zároveň je také jisté, že nebudou kolidovat s žádným bitem, který bude vypočítán. Je tedy možné vektor třikrát (v taktech 1, 2 i 3) úplně vynechat z výpočtu překrytí. V taktech 1 a 2 je známa jeho nevyužitelnost, a v taktu 3 je jisté, že bude bez problému překryt až k poslednímu bitu. Vektory B a C sice mají v následujících taktech nespecifikované bity, ale není jisté, jestli bude v aktuální pozici zvoleno řešení, které prosazují, tedy hodnota „1“.

5.8.3. Vynechání výpočtu překrytí

Pokud má tedy vektor na aktuální pozici nespecifikovaný bit, je potřeba zjistit jen kolik následujících taktů je ve vektoru nespecifikovaný bit, neboli po kolika taktech se na aktuální pozici objeví další specifikovaný bit. Tento výpočet proběhne pro každý nově zjištěný výskyt série nespecifikovaných bitů ve vektoru jen jednou, zároveň je rychlejší, než zjištění překrytí s už zkomprimovanou posloupností. Při každém vyhodnocení překrytí je tedy zároveň zjištěno, zda následuje série nespecifikovaných bitů, a jak je dlouhá. Pokud následuje alespoň jeden nespecifikovaný bit, je vyhodnocení nasunutí vektoru v dalším běhu programu o patřičný počet kroků odloženo.



Obrázek 25: Dynamická struktura pro přeskakování výpočtu překrytí

Vektory jsou podle počtu zjištěných DC bitů ukládány do dynamické struktury (viz Obrázek 25). Podle počtu taktů nutných k dosažení dalšího specifikovaného bitu je do struktury uložený odkaz na každý vektor.

V aktuálním taktu je vyhodnoceno překrytí pouze vektorů v pozici „0“, ostatní nemohou do řešení zasáhnout. Je zřejmé, že je nutné ukládat jen pozici k následujícímu specifikovanému bitu, tedy že každý vektor bude ve struktuře uložen pouze jednou. Po volbě vhodného bitu podle vektorů v čelní pozici jsou tyto zařazeny na odpovídající místo ve struktuře, s ohledem na to jestli byl vybrán jejich bit nebo ne.

Na závěr každého kroku algoritmu je seznam příslušející taktu 0 prázdný, došlo už k přeřazení všech vektorů do odpovídajících seznamů. Je proto možné celé pole posunout o jeden krok vpravo, takže na pozici 0 se dostane seznam z pozice 1, na 1 z 2 atd. Tím je celá struktura připravena na další cyklus algoritmu.

5.8.4. Režie metody

Nároky na uložení dynamické struktury v paměti jsou velmi nízké. Pole „steps to care“ má délku rovnou délce testovacího vektoru (mezní případ pro vektor obsahující dva specifikované bity na opačných koncích vektoru), v každém prvku pole je pouze ukazatel na začátek zřetězeného seznamu. Ve zřetězeném seznamu se ke každému vektoru ukládá pouze ukazatel na nekomprimovaný testovací vektor a ukazatel na další prvek seznamu, tedy dva ukazatele.

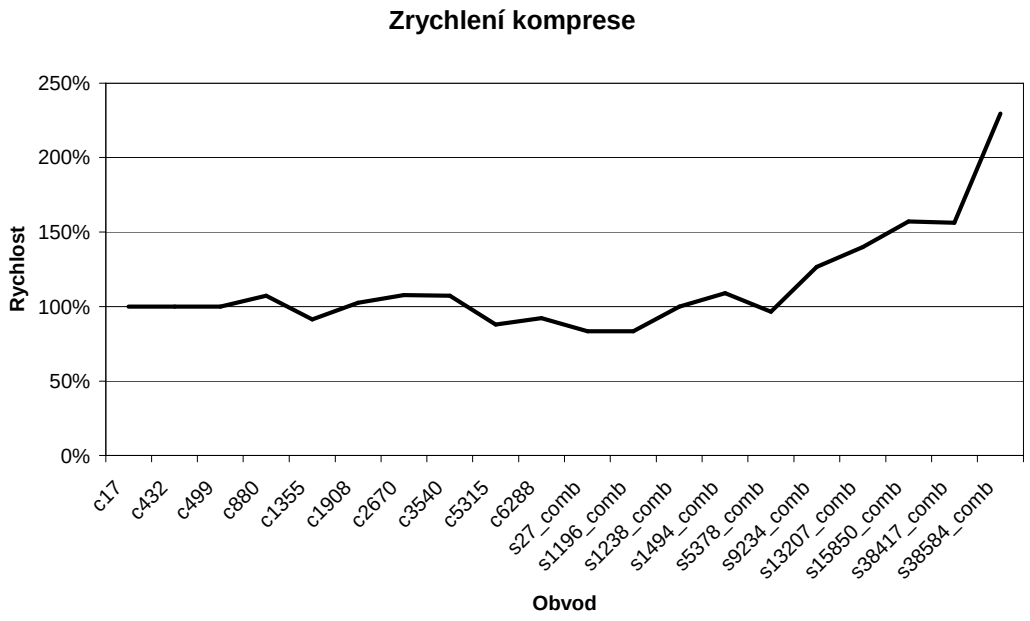
Při detekování poruchy odpovídající určitému testovacímu vektoru je prvek z paměti odstraněn, paměťové nároky tedy v průběhu komprese klesají.

Maximum obsazené paměti touto strukturou lze vypočítat pro 32bitovou architekturu jako součet dvou částí:

- fixní část: pole délky shodné s délkou testovacího vektoru, ve kterém každá položka vyžaduje 4 byty

- proměnná část: maximální velikost je rovna počtu nekomprimovaných testovacích vektorů, kde každý vektor vyžaduje dva ukazatele po 4 bytech

Pro obvod s milionem poruch a délkou testovacího vektoru 10000 je tedy pomocná struktura velká jen necelých 8 MB.



Graf 5: Zrychlení komprimační části algoritmu

5.8.5. Výsledky

Graf 3: Množství specifikovaných bitů pro různé obvody ISCAS85, ISCAS89 a ITC99 ukazuje, že nekomprimovaná testovací data obsahují značné množství nespecifikovaných bitů, navíc se poměr zlepšuje pro velké obvody. Vynecháním ohodnocení vektoru při výskytu nespecifikovaného bitu je tedy možné ušetřit velké množství výpočtů.

Tabulka 10: Časová náročnost jednotlivých částí algoritmu

	Původní verze			Nová verze			Výsledná rychlost [%]		
Obvod	Komprese [s]	FSIM [s]	Celkem [s]	Komprese [s]	FSIM [s]	Celkem [s]	Komprese	FSIM	Celkem
c17	0,01	0,01	0,02	0,01	0,01	0,02	100%	100%	100%
c432	0,03	0,06	0,1	0,03	0,07	0,1	100%	86%	100%
c499	0,07	0,15	0,3	0,07	0,18	0,3	100%	83%	100%
c880	0,44	3,6	4,3	0,41	2,99	3,9	107%	120%	110%
c1355	0,64	1,47	2,2	0,7	1,47	2,3	91%	100%	96%
c1908	0,42	1,86	2,4	0,41	1,81	2,4	102%	103%	100%
c2670	16,7	20,06	37,5	15,5	19,58	36,4	108%	102%	103%
c3540	0,44	3,6	4,3	0,41	2,99	3,9	107%	120%	110%
c5315	3,18	6,31	9,7	3,61	8,45	12,3	88%	75%	79%
c6288	0,12	0,8	1	0,13	0,79	1	92%	101%	100%
s27_comb	0,15	0,83	1	0,18	0,81	1,1	83%	102%	91%
s1196_comb	0,15	0,83	1	0,18	0,81	1,1	83%	102%	91%
s1238_comb	0,18	0,92	1,2	0,18	0,9	1,2	100%	102%	100%
s1494_comb	0,12	0,67	0,8	0,11	0,67	0,8	109%	100%	100%
s5378_comb	4,09	12,43	17	4,24	12,66	17,3	96%	98%	98%
s9234_comb	40,7	111,16	153,4	32,14	111,49	145,2	127%	100%	106%
s13207_comb	43,32	79,29	123,6	30,95	82,95	115,2	140%	96%	107%
s15850_comb	62,35	163,88	227,7	39,64	147,74	188,9	157%	111%	121%
s38417_comb	1006,65	1397,48	2413,9	644,25	1148,21	1798,7	156%	122%	134%
s38584_comb	217,79	397,99	617,9	94,89	471,58	569	230%	84%	109%

Tabulka 10 rozebírá časový přínos algoritmu a jeho vliv na celkovou rychlost komprese. (Změřeno na procesoru Intel Core2Duo 2,4 GHz.) Je zřejmé, že algoritmus podává lepší výsledky pro větší obvody, což odpovídá poměrnému výskytu nespecifikovaných bitů. Údržba dynamické struktury také vyžaduje systémový čas, proto se může stát, že sice budou opakovaně vynechány výpočty překrytí vektoru, ale režie sama spotřebuje tolik, že celá komprimace spotřebuje víc času. Algoritmus v každém kroku spouští simulaci poruch, proto je celkový čas složen ze dvou částí, času komprese (tedy vlastního hledání bitů komprimované testovací posloupnosti) a z času poruchové simulace. Protože simulace poruch probíhá v obou verzích algoritmu stejně, je i čas simulace přibližně stejný.

Zrychlení se tedy odehrává v komprimační části, jak ukazuje i Graf 5. Protože je ale v současnosti doba komprese menší než čas spotřebovaný simulátorem, v celkovém času se zrychlení projeví poměrně méně. Na situaci je možné zhruba aplikovat Amdahlův zákon zrychlení [23], v tomto případě by paralelní část byla komprese a sériová simulace poruch.

Rychlost se podařilo zvýšit vynecháním některých výpočetních kroků, je-li zřejmé, že jejich vykonání není přínosné. Nekomprimovaná testovací data

obsahují značné množství nespecifikovaných bitů, navíc se poměr zlepšuje pro velké obvody. Vynecháním výpočtu vektoru při výskytu nespecifikovaného bitu je tedy možné ušetřit velké množství výpočtů.

5.9. Závislost na použitém generátoru poruch

5.9.1. Struktura dat

Do operační paměti jsou nekomprimovaná data ukládána v podobě struktury, kde ke každé poruše náleží jeden testovací vektor. Porucha přiřazená testovacímu vektoru je ta, pro kterou ATPG vygeneroval daný testovací vektor. Jedná se ve skutečnosti o redukovanou tabulku poruch, přesněji její diagonálu. Použití kompletní tabulky poruch by sice poskytlo kompresnímu algoritmu dodatečné informace, ale kvůli její velikosti (počet testovacích vektorů násoben počtem všech testovatelných poruch v obvodu) je to pro větší obvody nereálné.

Díky zvolené struktuře vstupních dat je také možné generovat testovací data pro obvod paralelně spuštěním několika ATPG. Limitním případem tak je spuštění tolika ATPG kolik je poruch.

5.9.2. Motivace

Systém COMPAS byl vyvíjen pro použití s programovým balíkem ATALANTA. To vedlo původně k jistým omezením (problémy s většími obvody, vývoj ATALANTy byl ukončen). Zároveň je kompresní algoritmus opakovaně laděn na stejných vstupních datech, vzniká tedy otázka na jeho chování v případě použití jiného ATPG.

Hypotéza: Při dané struktuře vstupních dat pro COMPAS je možné použít libovolný ATPG, aniž by byl výsledný kompresní poměr výrazně ovlivněn.

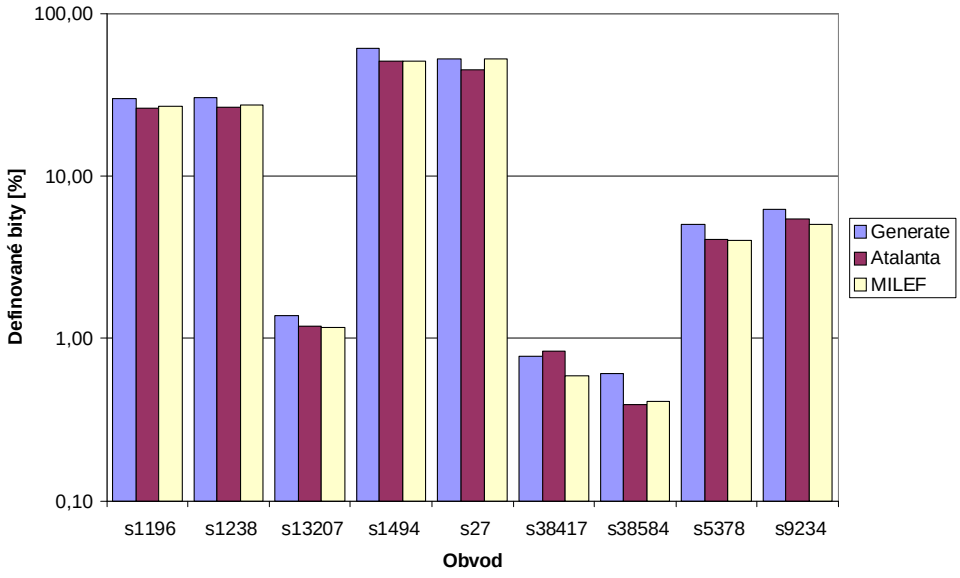
5.9.3. Rozšíření COMPASu

K ověření vyslovené hypotézy byl systém COMPAS plnohodnotně rozšířen o podporu nástrojů z balíku TurboTester [40] vyvíjeného na Technické univerzitě

v Tallinnu, Estonsko. Jedná se o komplexní sadu programů zaměřených na testování obvodů pomocí SSBDD (strukturně syntetizované binární rozhodovací diagramy) [39]. Pro generování deterministických testů slouží ATPG Generate, používá popis obvodu pomocí SSBDD, generuje testovací vektory pro testování trvalých poruch a používá algoritmus PODEM. Jako simulátor poruch byl vybrán SPSimul, neboli „Single Pattern Simulator“.

Pro ověření hypotézy byl také použit upravený ATPG MILEF [41], kterým používá Mixed Level FAN algoritmus, byly generovány testy pro kombinační obvody s trvalými poruchami. Simulátor poruch z programu MILEF nebyl do COMPASu začleněn, namísto toho byl používán už vestavěný simulátor FSIM.

Pro začlenění dalšího simulátoru poruch však bylo nutné zdvojit většinu datových struktur, funkcí svázaných se strukturou obvodu a použitým simulátorem poruch, funkcí pro načítání a zápis vstupních a výstupních formátů, především ale integrovat na úrovni zdrojového kódu simulátor poruch pracující s SSBDD.

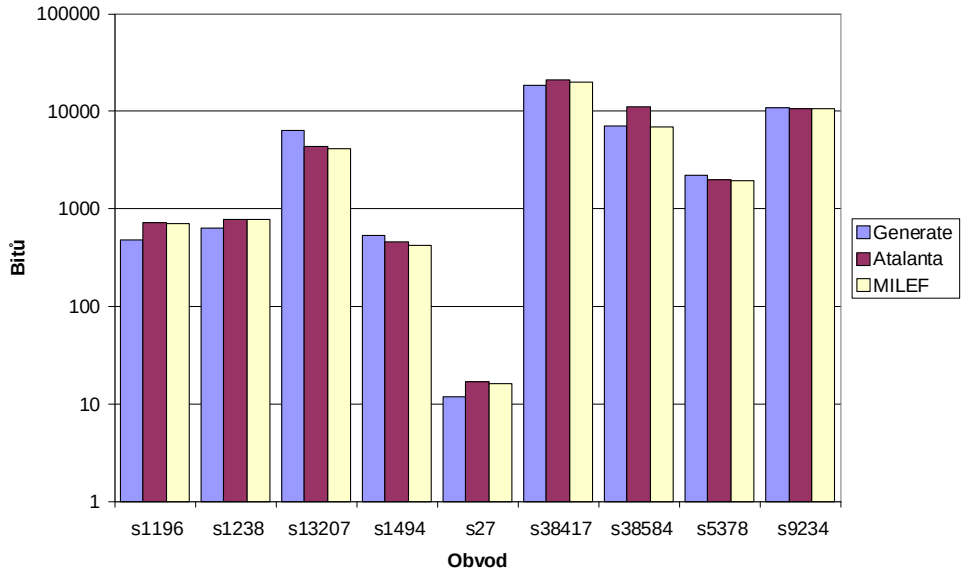


Graf 6: Počet specifikovaných bitů v nekomprimovaných testovacích datech generovaných různými ATPG

Jako základ pro druhý integrovaný simulátor poruch byl použit SPSimul. Tento simulátor pracuje s jednotlivými testovacími vektory, neumožňuje použití nespecifikovaných bitů ve vstupním vektoru, používá kompaktní seznamy poruch a fault dropping.

Zdánlivá omezení simulátoru na jeden testovací vektor v jednom simulačním cyklu a nemožnost použití nespecifikovaných bitů ve skutečnosti umožňují urychlit simulační algoritmus. Z popisu kompresního algoritmu je totiž zřejmé, že vždy bude simulován pouze aktuální stav skenovacího řetězce. Ten ale nespecifikované bity nemůže obsahovat. Simulace je spuštěna po nalezení každého jednotlivého bitu komprimované testovací posloupnosti, je tedy nutné simulovat jen jediný testovací vektor. Zároveň jsou jednou detekované poruchy odstraňovány z paměti. Tyto dodatečné informace mohou být využity při návrhu a úpravách simulátoru, což potenciálně vede k jednoduššímu a rychlejšímu kódu.

Upravený simulátor byl plně včleněn do COMPASu, a je ho tak možné použít pro simulaci poruch při kompresi s použitím SSBDD modelu obvodu. V současné době tedy COMPAS obsahuje dva interní simulátory poruch, FSIM pro BENCH modely obvodů a SPSimul pro SSBDD modely.



Graf 7: Délky komprimovaných posloupností

5.9.4. Testovací data

Jádro kompresního algoritmu, tedy hledání bitů komprimované posloupnosti, je stejné při použití obou možností modelu obvodu – BENCH i SSBDD, liší se pouze použitým simulátorem poruch. Pokud ponecháme shodnou i konstantu *t* ve funkci ohodnocení testovacího vektoru a použijeme stejný testovací obvod, je možné přímo srovnat výsledky komprese s použitím obou modelů. Tím je možné prokázat, že algoritmus není vyladěný specificky na data produkovaná ATALANTou.

Protože všechny tři použité ATPG (ATALANTA, GENERATE i MILEF) používají jiný algoritmus, testovací vektory pro shodné poruchy mohou být rozdílné. Možnost komprese je teoreticky značně ovlivněna množstvím nespecifikovaných bitů v testovacích datech, protože nespecifikované bity se nejsnáze překrývají.

Všemi ATPG byla vygenerována testovací data pro některé obvody z testovací sady ISCAS89, počet care bitů v datech zobrazuje Graf 6: Počet specifikovaných bitů v nekomprimovaných testovacích datech generovaných různými ATPG. Je vidět, že GENERATE většinou generuje nejméně DC bitů. Zároveň je vidět, že všechny ATPG podávají podobné výsledky. Komprimovaná posloupnost by tedy měla být pro vstupní data z GENERATE delší.

5.9.5. Výsledky

Výsledky komprese ale předchozí tvrzení potvrzují jen částečně. I když má algoritmus v datech k dispozici více nespecifikovaných bitů pro potenciální překrytí, výsledná komprimovaná posloupnost není vždy kratší, viz Graf 7: Délky komprimovaných posloupností. Vztah „více nespecifikovaných bitů, lepší výsledek“ tak platí jen přibližně v polovině případů. Tento výsledek pravděpodobně souvisí s rozložením specifikovaných bitů v jednotlivých vektorech. Je třeba si uvědomit, že rozdíly délek komprimovaných posloupností pro různé druhy vstupních dat jsou nanejvýš několikanásobek délky testovacího vektoru, jak uvádí Tabulka 11. Stačí tedy například jeden testovací vektor s tak

obtížným rozložením vygenerovaných specifikovaných bitů, že se ho nepodaří příliš překrýt, a výsledek je značně ovlivněn k horšímu.

Tabulka 11: Rozdíly délek po kompresi vůči délkám testovacích vektorů

Rozdíl délek komprimovaných posloupností [b]			Délka test. vektoru [b]	Rozdíl délek v násobcích délky vektoru		
ATALANTA - GENERATE	MILEF - GENERATE	ATALANTA - MILEF		ATALANTA - GENERATE	MILEF - GENERATE	ATALANTA - MILEF
232	221	11	32	7,25	6,91	0,34
152	150	2	32	4,75	4,69	0,06
1930	2199	269	700	2,76	3,14	0,38
76	107	31	14	5,43	7,64	2,21
5	4	1	7	0,71	0,57	0,14
2531	1361	1170	1664	1,52	0,82	0,70
4160	208	4368	1464	2,84	0,14	2,98
191	247	56	214	0,89	1,15	0,26
299	385	86	247	1,21	1,56	0,35

COMPAS byl tedy úspěšně rozšířen o podporu softwarového balíku TurboTester, jako součást COMPASu byl implementován další simulátor poruch SPSimul. Účinnost algoritmu byla ověřena použitím třech různých ATPG. Přestože kompresní algoritmus nebyl pro data generovaná ATPG Generate nebo MILEF nijak upravován, ani volitelný parametr *t* nebyl nijak měněn, vykazuje COMPAS velmi dobrou účinnost, srovnatelnou s předchozími výsledky získanými kompresí dat z ATPG ATALANTA. Všechny tři ATPG přitom používají jiný algoritmus pro generování testovacích vektorů. Je tedy možné prohlásit, že kompresní algoritmus je možné úspěšně použít s libovolným ATPG schopným generovat vektory s nespecifikovanými bity. Hypotéza ze začátku kapitoly je tedy platná.

Pro ověření hypotézy byly prozatím použity pouze ATPG z akademické sféry, dalším krokem tedy bude využití komerčních generátorů testů. Tím bude tento směr výzkumu pravděpodobně uzavřen.

5.10. Obtížně testovatelné poruchy

5.10.1. Motivace

Dalšího zlepšení kompresního poměru je možné dosáhnout změnou pořadí překrývaných vzorků, jejich lepším řazením. Je výhodné na začátku zakódovat testovací vzorky, které detekují obtížně detekovatelné poruchy (poruchy odolné proti náhodnému testu), protože poruchy lehce testovatelné náhodnými vzorky budou pravděpodobně detekovány v mezilehlých krocích algoritmu.

5.10.2. Nalezení obtížně detekovatelných poruch

Dva hlavní problémy jsou jak určit, která porucha je obtížně detekovatelná, a jak tuto informaci efektivně využít pro kompresní algoritmus.

Pro rozlišení, zda je porucha obtížně detekovatelná nebo ne, existuje několik různých metod. Tyto metody jsou většinou založeny na analýze vnitřní struktury obvodu a následném výpočtu řiditelnosti a pozorovatelnosti a jsou používány pro snížení náročnosti výpočtu při deterministickém výpočtu testovacích vektorů. Známým příkladem je Sandia Controllability/Observability Analysis Program (SCOAP) [42]. Tyto metody ale poskytují informace o poruchách, a protože COMPAS zpracovává již deterministicky vygenerované testovací vektory, jednoduchá informace, že vybraná porucha je nebo není obtížně detekovatelná, není dostačující.

Pro použití s COMPASem musí být proto použité jiné metody jak zjistit, je-li porucha obtížně detekovatelná. Prvním způsobem je provést poruchovou simulaci s velkým množstvím náhodných vektorů a spočítat, kolikrát byla každá porucha detekovaná. Při použití této metody nemusí být některé poruchy vůbec detekovány.

Další metodou je použít při simulaci již vygenerovaných deterministických vzorků. Jelikož tyto vzorky obsahují nespecifikované bity, je tyto nutné vyplnit zvolenou hodnotou, nebo použít simulátor poruch, který podporuje nespecifikované vstupní hodnoty. Pro případné vyplňování je možné použít

několik metod, např. vyplnění nulou, vyplnění jedničkou nebo vyplnění náhodnou hodnotou. Tato metoda garantuje, že každá porucha bude detekovaná přinejmenším jednou.

Obě předchozí metody nevyžadují přímou analýzu vnitřní struktury obvodu, pracují pouze se simulátorem poruch. Výsledkem obou metod je zjištění frekvence detekce pro všechny poruchy, neboli kolikrát byla která porucha detekovaná, kde nižší číslo značí obtížněji detekovatelnou poruchu.

5.10.3. Obtížnost vektoru

Výše zmíněné metody poskytují informaci o obtížnosti testovatelnosti poruchy náhodným vektorem v lépe použitelné formě než metoda založena na pozorovatelnosti a řiditelnosti. Protože COMPAS komprimuje testovací vektory, je nutné informaci o poruchách nějakým způsobem převést na informaci o testovacích vektorech. To je provedeno tak, že všechny testovací vektory poskytnuté pomocí ATPG jsou každý zvlášť simulovány a je zjišťováno kolik poruch vektor detekoval a jak moc jsou detekované poruchy odolné vůči náhodnému testu. Výsledné zhodnocení testovacího vektoru je poté vypočteno jako součet frekvencí detekce všech poruch detekovaných daným testovacím vektorem. Nazývejme tuto hodnotu *obtížnost vektoru*, kde vektor s nižší hodnotou obtížnosti detekuje méně lehce detekovatelných poruch a více obtížně detekovatelných. Vektor s nízkou hodnotou obtížnosti by měl být umístěn v komprimované posloupnosti co nejdříve.

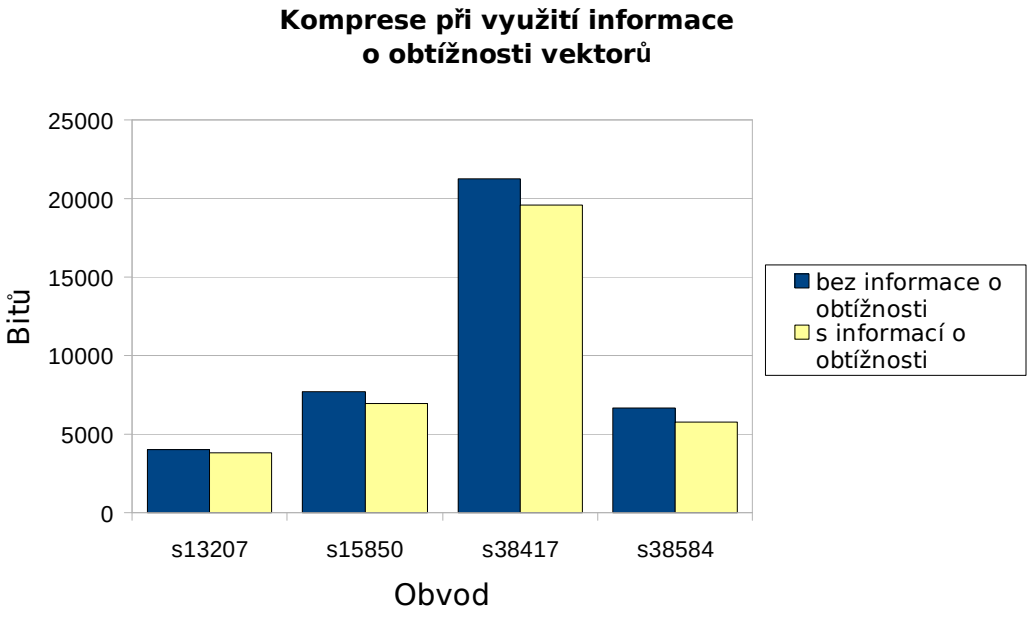
5.10.4. Nové kritérium

Použitím hodnoty obtížnosti vektoru můžeme definovat nové kritérium ohodnocení užitečnosti vektoru:

$$U = t * (overlapped\_cares + shift) + all\_cares + hard\_patt$$

Kde *hard_patt* značí hodnotu obtížnosti vektoru a ostatní parametry mají stejný význam jako v původním kritériu. Když algoritmus využije tento nový vzorec, všechny testovací vektory budou ohodnoceny jinak než v předchozí verzi. Odlišné

hodnoty užitečnosti pak vedou k jinému pořadí výběru vektorů a ke zlepšení komprese.



Graf 8: Zlepšení komprese při využití informace o obtížnosti vektorů

5.10.5. Výsledky

Graf 8: Zlepšení komprese při využití informace o obtížnosti vektorů ukazuje zlepšení kompresního poměru při použití informace o poruchách detekovaných testovacím vektorem a jejich obtížnosti oproti verzi bez tohoto ohodnocení. Pro zjištění obtížnosti vektoru byla zvolená metoda simulace všech deterministických nekomprimovaných testovacích vektorů a jejich následná resimulace s výpočtem obtížnosti vektoru s použitím simulátoru poruch vestavěného v programu COMPAS. Nespecifikované bity byly vyplněny náhodnou hodnotou. Průměrné zlepšení kompresního poměru je přibližně deset procent, viz též Tabulka 12: Zlepšení komprese při využití informace o obtížnosti vektorů.

Tabulka 12: Zlepšení komprese při využití informace o obtížnosti vektorů

Obvod	Bez ohodnocení těžkých vektorů [b]	S ohodnocením těžkých vektorů [b]	Zkrácení na
s13207	4,024	3,819	94,91%
s15850	7,737	6,930	89,57%
s38417	21,280	19,597	92,09%
s38584	6,675	5,778	86,56%

5.10.6. Režie metody

Ta část algoritmu, kde je zjišťována obtížnost vektoru, je spouštěna pouze jednou. Výpočetní čas potřebný pro zjištění hodnoty obtížnosti je závislý jen na rychlosti použitého simulátoru poruch. Výsledek je ukládán jako čistě textový soubor pro pozdější opakované použití. Čas potřebný pro samotnou kompresi se díky využití přepočítaných hodnot nemění (jeden parametr přidaný do kritéria užitečnosti nemá na čas výpočtu měřitelný vliv).

5.11. Optimalizace pro víceprocesorové systémy

Jako řešení pro paralelizaci algoritmu bylo vybráno rozhraní OpenMP [35], protože poskytuje dostatečnou úroveň abstrakce, je platformě nezávislé a je zdarma.

Běh programu byl analyzován a vybrané úseky vhodné pro paralelní spouštění byly označeny pomocí direktiv OpenMP. Kompilátor, který podporuje standard OpenMP, pak při překladu zajistí vytvoření souběžných programových vláken.

Jako úseky vhodné pro spuštění ve více vláknech byly vybrány především časově nejnáročnější části kódu, tedy překrývání vektorů, zařazování vektorů do seznamu budoucích řešení a řadicí algoritmus, který je často používán v různých částech algoritmu.

Překrývání vektorů je přirozeným kandidátem na paralelizaci. K zjištění překrytí jednoho každého vektoru se skenovacím řetězcem dochází zcela nezávisle na

ostatních vektorech. Nezávisle je vypočteno i ohodnocení vektoru. To umožňuje rozdělení výpočtu na několik vláken. Při použití dvou procesorů jsou tedy vektory rozděleny na dvě skupiny, kde každá je nezávisle ohodnocena.

Po skončení překrývání a ohodnocení jsou vektory seřazeny podle hodnoty kritéria. Jako řadicí algoritmus byl pomocí OpenMP implementován paralelní mergesort, protože se dobře paralelizuje, je to přirozený a stabilní řadicí algoritmus a má nízkou složitost $O(N \log N)$.

Nakonec jsou seřazené vektory zařazovány do pole budoucích řešení, což je časově poměrně náročné. Protože je to ale poslední krok před konečným výběrem aktuálního komprimovaného bity a simulací poruch, mohou už být prováděny některé údržbové práce na pomocných datových strukturách. V tomto případě jsou tedy v každém programovém vlákně prováděny rozdílné činnosti.

Paralelní a sériová verze algoritmu byla porovnána na počítači s dvoujádrovým procesorem Core2Duo na 2,4 GHz. Paralelizace časově náročných částí algoritmu bohužel nepřinesla zrychlení programu (výsledky byly přibližně shodné), přestože vytížení procesorů v systému vzrostlo z 50% (jedno jádro plně zatíženo sériovým algoritmem, druhé nevyužito) na přibližně 80% (obě jádra částečně využita). Důvodem může být příliš málo paralelní práce a příliš velká režie vícevláknového běhu. Množství paralelní práce (tedy množství vektorů, které je nutné překrýt, následně setřídit a zařadit do pole FA) je totiž účinně snižováno pomocí metody z kapitoly 5.8 Urychlení vynecháváním výpočtů. Velkou část doby komprese navíc tvoří simulace poruch (viz Tabulka 10: Časová náročnost jednotlivých částí algoritmu), proto se zrychlení vyhledávání komprimovaných bitů projeví poměrně málo.

5.12. Paralelní skenovací řetězce

V kapitole 2.3 jsou zmíněny výhody použití více skenovacích řetězců. Pokud je použito více skenovacích řetězců, je vhodné použít jako dekompresní hardware RESPIN architekturu [26]. Použití této architektury vyžaduje dodržení několika podmínek. Jejich rozbor a zdůvodnění je možné nalézt např. v [52].

Pro kompresi je důležité, jak návrhář zapojil skenovací řetězce, které skenovací buňky jsou v jakém řetězci, kolik řetězců je použito, jaké je propojení s ETC jádrem apod. Všechny tyto informace ovlivňují pořadí bitů v testovacích vektorech. Pokud jsou ale dodrženy všechny podmínky použití RESPIN architektury, projeví se všechny varianty zapojení jen jako změna pořadí vstupů obvodu.

Je důležité upozornit, že tato práce se nezbývá vlivem zapojení testovací architektury na pořadí vstupů obvodu. Testovací hardware navrhuje podle různých potřeb návrhář, je tedy jeho úkolem dodat popis pořadí vstupů, jaké si zvolil.

Popis pořadí vstupů je COMPASu předán jako jednoduchý textový soubor s řadou čísel, která postupně určuje pořadí jednotlivých vstupů obvodu. Soubor je čistě textový, uživatel ho tedy může přímo číst i upravovat. COMPAS soubor načte a při postupném načítání nekomprimovaných testovacích dat ihned provede změnu pořadí bitů.

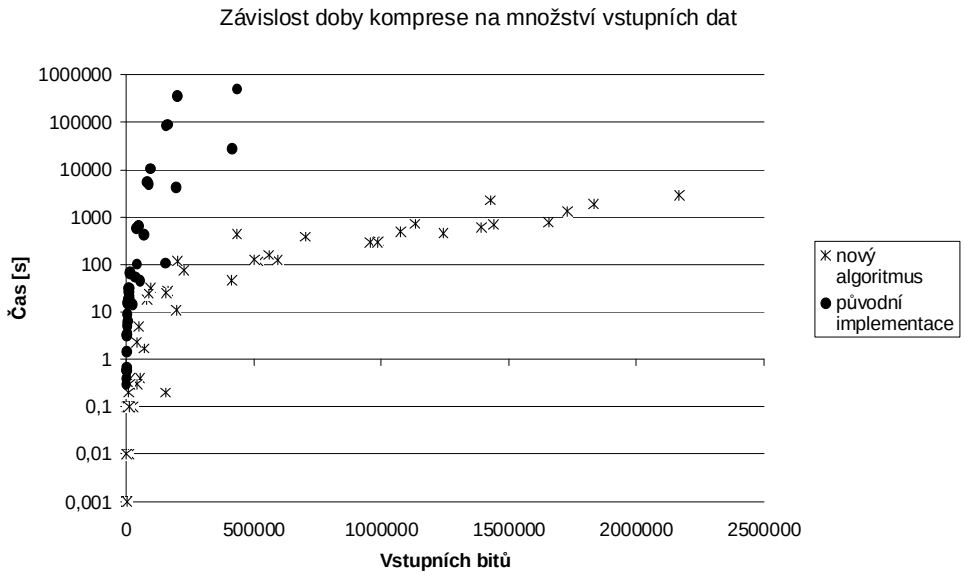
Komprese dat s prohozeným pořadím vstupů pak probíhá až na simulaci poruch stejně jako komprese bez prohození. V průběhu komprese je vždy po vygenerování dalšího komprimovaného bitu spuštěna simulace poruch, kdy jako testovací vektor slouží aktuální stav skenovacího řetězce. Simulátor poruch ale načítá jen strukturu obvodu, nemá k dispozici popis zapojení testovacího hardwaru. Před spuštěním simulace je tedy nutné vždy změnit pořadí bitů ve skenovacím řetězci tak, aby odpovídalo pořadí vyžadovanému návrhářem obvodu. Po dokončení simulace poruch je obsah skenovacího řetězce obnoven do původního stavu a komprese pokračuje běžným způsobem.

6. Shrnutí a závěr

6.1. Srovnání s původní verzí algoritmu

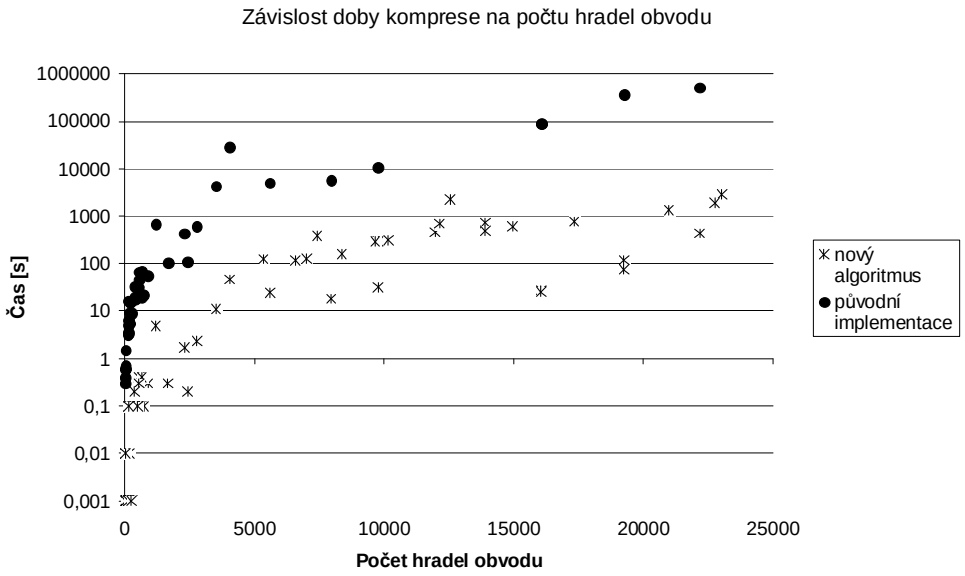
Ve srovnání s původní verzí algoritmu byly výrazně sníženy paměťové nároky programu (viz kapitola 5.7). Zvolená metoda garantuje minimálně 75% úsporu paměti. Pro větší obvody, které mají v testovacích datech více nespecifikovaných bitů, úspora dosahuje běžně přes 95%. Pomocí využití informací o obtížně testovatelných poruchách byla zlepšena komprese o průměrně 10%.

Nový algoritmus má lepší škálovatelnost (viz Graf 9: Závislost doby komprese na množství vstupních dat).



Graf 9: Závislost doby komprese na množství vstupních dat

Je také výrazně rychlejší (viz Graf 10: Závislost doby komprese na počtu hradel obvodu), v některých případech až o čtyři řády.



Graf 10: Závislost doby komprese na počtu hradel obvodu

6.2. Srovnání s metodami pro kompresi testovacích dat

Tabulka 13: Porovnání výsledků komprese dat různých metod

	MinTest	Stat. Coding	LFSR Reseeding	Illinois Scan	FDR Codes	EDT	RESPIN++	COMPAS	
1	2	3	4	5	6	7	8	9	10
Obvod	bitů	bitů	bitů	bitů	bitů	bitů	bitů	bitů	čas [s]
s13207	163,100	52,741	11,285	109,772	30,880	10,585	26,004	3,819	8
s15850	58,656	49,163	12,438	32,758	26,000	9,805	32,226	6,930	11
s38417	113,152	172,216	34,767	96,269	93,466	31,458	89,132	19,597	177
s38584	161,040	128,046	29,397	96,056	77,812	18,568	63,232	5,778	57

Tabulka 13 obsahuje výsledky některých dalších metod komprese testovacích vektorů ve srovnání s výsledky COMPASu, pokud použijeme všechna dostupná vylepšení. (Měřeno na procesoru Intel Core2Duo 2,4 GHz.) Výsledky jsou zobrazené jen pro čtyři obvody, protože pouze tyto byly zkoumány i ostatními metodami. V druhém sloupci je zobrazeno množství testovacích dat pro ATPG vektory, které prošly pouze kompakcí [44]. Sloupec 3 ukazuje počet uložených bitů pro statistické kódování testovacích vzorků z předchozího sloupce [45]. Sloupec 4 obsahuje výsledky statistického kódování společně s znovunastavením

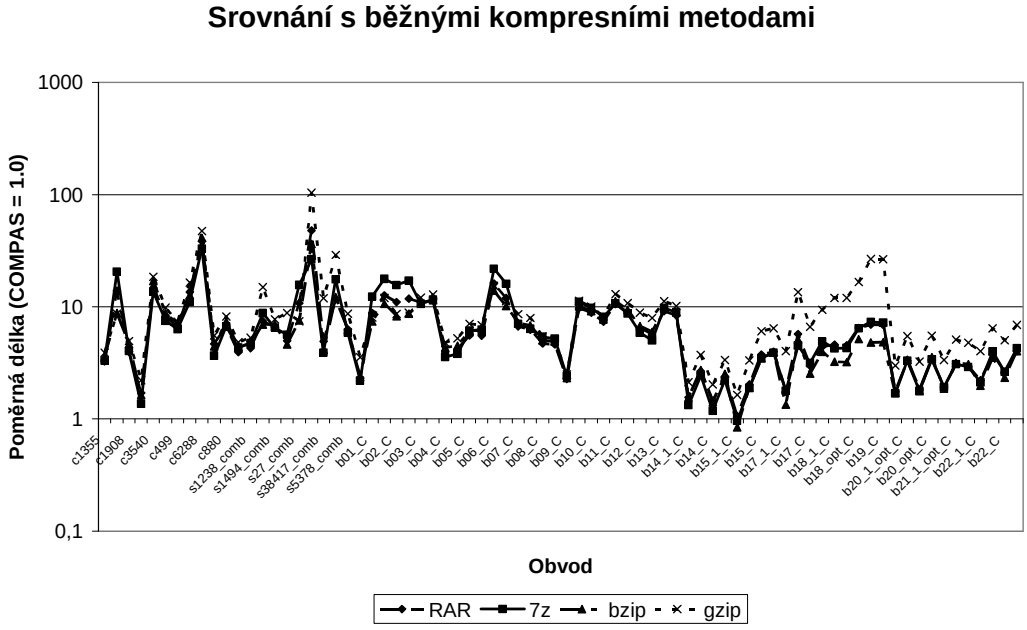
LFSR [29]. Sloupec 5 ukazuje výsledky komprese s paralelními/sériovými skenovacími řetězci [46], sloupec 6 výsledky frekvenčně řízeného kódování [28]. Výsledky metody Embedded Deterministic Test [47] jsou prezentovány ve sloupci 7. Sloupec 8 ukazuje množství bitů uložených v ATE pro architekturu RESPIN++ prezentovanou v [31]. Je zřejmé, že počet v ATE uložených bitů je pro metodu COMPAS (sloupec 9) výrazně nižší, než pro ostatní kompresní metody.

Je nutné poznamenat, že většina zmíněných metod komprese testovacích dat nepoužívá simulaci poruch při kódovaní jednotlivých testovacích vektorů (s výjimkou metody RESPIN++). Tyto metody ale používají kompaktní sadu testovacích vektorů, pokrytí poruch bylo proto simulátorem poruch zjišťováno v průběhu generování testu a kompakce pomocí ATPG. Počet simulací poruch je v těchto případech roven celkovému počtu nekompaktních testovacích vektorů. V případě RESPIN++ architektury a COMPASu jsou ATPG vektory generovány bez simulace poruch, protože jsou poruchy simulovány v průběhu komprese po každém vygenerování bitu komprimované testovací posloupnosti. Celkový počet spuštění simulátoru poruch je tak rovný výsledné délce komprimované testovací posloupnosti v bitech. Přestože je počet simulací poruch velký, celkový výpočetní čas algoritmu zůstává poměrně nízký. Obtížnost vzájemného srovnání výsledků komprese také způsobuje využití pseudonáhodných vektorů pro některé kompresní metody.

6.3. Srovnání s běžnými kompresními metodami

COMPAS dosahuje výrazně lepšího kompresního poměru než běžné kompresní programy jako gzip, bzip2, RAR nebo 7Z, viz Graf 11: Srovnání délky komprimované posloupnosti oproti běžným kompresním metodám. Vynesen je poměr dosažených délek komprimovaných posloupností jednotlivých programů oproti COMPASu, který je brán jako základ, tedy 1,0. Je vidět, že až na horší gzip dosahují ostatní kompresní programy podobných výsledků. COMPAS je výrazně lepší pro drtivou většinu z 77 testovaných obvodů (v případě srovnání s gzip

programem a obvodem s35932 dokonce více než 100 krát). Jedinou výjimkou je varianta obvodu b15 z testovací sady ITC99, kdy je COMPAS překonán programy bzip2 a 7z.



Graf 11: Srovnání délky komprimované posloupnosti oproti běžným kompresním metodám

Běžné algoritmy většinou používají pro redukci objemu dat slovník, který obsahuje už použité úseky dat. Je tedy výhodné mít slovník co největší, ideálně aby se do něj vešla všechna data. Datový úsek, které se vyskytuje v nekomprimovaných datech opakovaně, je s pomocí slovníku uložen jen jednou a každý další výskyt je nahrazen jen odkazem na předchozí výskyt. Takto redukovaný datový proud je pak obvykle dále komprimován některou statistickou metodou, například Huffmanovým kódem.

Program gzip používá Lempel-Zivův algoritmus s 256 KB velkým slovníkem. Program bzip2 používá Burrows-Wheelerův blokový třídící algoritmus s 900 KB slovníkem a podle potřeby přepíná ještě mezi Lempel-Zivovým algoritmem s 256 KB slovníkem. Program RAR je komerční a používá nedokumentovaný kompresní algoritmus s několika možnými velikostmi slovníku. Pro kompresi byl vybrán největší možný 4 MB slovník. Program 7zip používá Lempel-Ziv-

Markovův řetězový algoritmus, který umožňuje různou, po krocích nastavitelnou, velikost slovníku. Kvůli omezení velikosti operační paměti byl slovník nastaven na velikost 2725 MB.

Překrývání testovacích vektorů při kompresi COMPASem může být považováno za jistý druh slovníkové metody, neboť se opakovaně využívá dat již uložených ve skenovacím řetězci. Pokud se části dvou testovacích vektorů překrývají, je tato část uložena jen jednou a použita pro oba vektory. Protože se většinou podaří stejnou část dat překrýt pro několik vektorů, je tato komprese účinná. Velikost slovnímu je potom shodná s velikostí skenovacího řetězce.

Velkou výhodou algoritmu COMPAS oproti běžným metodám je použití simulátoru poruch při kompresi. Tím, že je každý krok algoritmu a tím i každý vygenerovaný vektor simulován, dochází k rychlé detekci poruch testovatelných náhodným testem. Jakmile je libovolná porucha detekována, je jí odpovídající vektor odstraněn z komprimovaných dat. Přitom nezáleží, jaké je rozložení bitů ve vektoru, vektor může být ze statistického hlediska velmi odlišný oproti aktuálnímu stavu komprimovaného řetězce, ale protože je schopen detekovat stejnou poruchu, je redundantní a jako takový je odstraněn. Algoritmus tedy šetří místo nejen statistickou metodou, kdy vyhledává největší podobnost mezi jednotlivými testovacími vektory tím, že je vzájemně překrývá, ale i využitím informací o vnitřní struktuře obvodu.

Výsledný komprimovaný řetězec spočítaný COMPASem by mohl být dále komprimován nějakou statistickou metodou, např. Huffmanovým kódem nebo aritmetickým kódem. To by ale vynucovalo použití značně komplikovaného a rozsáhlého dekompresního hardware uvnitř testovaného obvodu. Proto není žádné takové kódování použito.

6.4. Závěr

Systém pro kompresi testovacích dat COMPAS ukazuje, že metodu založenou na překrývání testovacích vektorů je možné úspěšně použít i na relativně velké obvody a že výsledný objem testovacích dat je velmi nízký. Největší

z používaných obvodů jsou varianty obvodu b19 z testovací sady ITC99, každý s více než 200 tisíci hradly. Dosažené výsledky ukazují, že algoritmus použitý v COMPASu může být dále vylepšen zjištěním dodatečných informací o vstupních datech.

Jako vstupní data používá COMPAS nekomprimované testovací vektory s nespecifikovanými bity. Vstupní data mohou být generována libovolným ATPG, který je schopen vytvořit dvojici porucha a vektor s nespecifikovanými bity. Protože algoritmus pracuje s páry tvořenými poruchou a jí příslušným vektorem, je COMPAS schopný komprimovat data vytvořená několika současně běžícími ATPG procesy, což může významně urychlit celý proces přípravy testu. Data generovaná ATPG jsou předzpracována a je určena možnost jednotlivých testovacích vektorů detekovat obtížně detekovatelné poruchy.

Na úrovni zdrojových kódů byly integrovány dva různé simulátory poruch, tato úprava vedla k více než stonásobnému urychlení. Urychlením komprese a efektivnějším kódováním nekomprimovaných testovacích dat v operační paměti počítače byly výrazně sníženy nároky systému COMPAS a otevřela se tak cesta pro další vylepšování algoritmu z hlediska lepšího kompresního poměru, především pomocí využití více testovacích vektorů pro jednu poruchu. Běh programu byl výrazně urychlen vynecháním opakovaných výpočtů a byla zlepšena škálovatelnost kompresního algoritmu. Byla ověřena nezávislost algoritmu COMPAS na použitém generátoru testovacích vektorů a použitém poruch. Byly úspěšně provedeny základní experimenty s paralelním zpracováním dat.

Všechny komprimované testovací vektory jsou v průběhu komprese simulovány simulátorem poruch zda nepokrývají nějaké dodatečné poruchy. Tento mechanismus redukuje počet výsledných vektorů, protože mezilehlé stavy vzniklé při posouvání dat ve skenovacím řetězci jsou používány pro detekci náhodně testovatelných poruch. Tyto poruchy jsou v jiných metodách obvykle detekovány náhodným testem a vedou na mixed-mode test, v COMPASu použitá metoda tento obvykle časově i energeticky náročný krok eliminuje. Přestože algoritmus

při kompresi používá simulaci mezilehlých stavů, které přímo nepřísluší žádnému nekomprimovanému vektoru (a šly by chápat jako pseudonáhodné), nelze ho přímo zařadit mezi metody smíšeného testu, protože deterministická a pseudonáhodná část se překrývají.

Zvolenou metodu komprese je možné velmi dobře použít v kombinaci s metodou Boundary Scan, protože pak pro dekompresi testovacích dat není vyžadován žádný dodatečný hardware. Může být použitá i pro sekvenční obvody s jedním či více skenovacími řetězci, pak je nutné použít dekompresního hardwaru, např. RESPIN architekturu. Testovací sekvence je pro použití s RESPIN architekturou a více skenovacími řetězci patřičně přeřazena tak, aby byla správně dekomprimována. Při dodržení IEEE 1500 standardu není nutné přidávat žádný dodatečný hardware s výjimkou jednoho multiplexoru a jedné zpětné vazby pro každé jádro. Testovací sekvence generovaná COMPASem může být použita pro méně časově náročný test sekvenčních obvodů než pro test dosažitelný s mixed-mode testovacími metodami.

Systém COMPAS vytváří vysoce komprimované testovací vektory, dle dostupných materiálů je kompresní poměr lepší než pro ostatní srovnatelné metody a pro test je tak potřebná menší paměť, než pro srovnatelné metody.

Některé kompresní metody sice mohou dosáhnout ještě vyšší komprese, potřebují ale velké množství (obvykle 10000) pseudonáhodných testovacích vektorů. To výrazně zvyšuje čas komprese a energetickou náročnost testu. Použití algoritmu COMPAS vyžaduje pro vykonání testu nižší počet hodinových cyklů, test obvodu je proto rychlejší a levnější než při použití jiných kompresních metod.

Hardwarové nároky jsou tvořeny dekompresním hardwarem a pamětí nutnou pro uložení komprimované posloupnosti. Při použití architektury RESPIN není potřeba žádný speciální hardware, dekomprese testovacích vzorků probíhá ve skenovacích řetězcích jiných jader. Hardwarové nároky tedy tvoří jen nutná paměť.

Shrnutím výsledků je, že navržená kompresní a testovací metoda dosahuje při kompletním hodnocení (náročnost návrhu, velikost nutného hardware, rychlost a energetická náročnost testu) výrazně lepších výsledků než ostatní kompresní metody.

Shrnutí přínosů k rozvoji vědního oboru

V práci je popsán nový algoritmus komprese testovacích dat, který využívá metody překrývání testovacích vektorů. Algoritmus je podrobně popsán a je uveden přínos jednotlivých částí z hlediska vylepšení kompresního poměru, snížení doby komprese a snížení paměťových nároků.

Výsledky kompresní metody ukazují, že pro dosažení výborných výsledků není nutné kompletní prohledávání stavového prostoru, ale postačuje hladový algoritmus. Je prokázáno, že pro dosažení dobrého kompresního poměru je vhodné v průběhu komprese využívat simulátor poruch a brát v úvahu obtížně detekovatelná poruchy.

Shrnutí přínosů pro praxi

Všechny navržené metody byly otestovány na sadách ISCAS85, ISCAS89 a ITC99 s výbornými výsledky. Testované obvody tvoří dostatečně široké spektrum, aby bylo možné prohlásit, že COMPAS je dobře použitelný v praxi. Návrhářům obvodů je tak k dispozici rychlá, kvalitní a univerzálně použitelná kompresní metoda.

6.5. Budoucí práce a experimenty

Zlepšením algoritmu je možné uvažovat o použití více testovacích vektorů pro jednu poruchu, což pravděpodobně povede k zlepšení komprese. Budoucí úlohou je také použití dalších ATPG, zvláště komerčních.

Algoritmus již částečně využívá paralelní zpracování dat, dalším úkolem bude dopracovat a vyladit program pro efektivnější využití více procesorů.

Seznam literatury

Vlastní publikace

[1] Jeníček, J.:Optimalizace systému pro kompresi testovacích vektorů COMPAS, PAD2005, ISBN 80-01-03298-1, pp.73-76

[2] Novák, O., Zahrádka, J., Holubec, M., Jeníček, J.:Test Set Compaction and Compression for circuits with Scan, Informal Digest of Papers of the IEEE European Test Symposium, Ajacio Corsica, France, 2004, pp.13-14

[3] Novák, O.; Plíva, Z.; Jeníček, J.; Mader, Z.; Jarkovský, M.:Self Testing SoC with Reduced Memory Requirements and Minimized Hardware Overhead, The 21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'06), October 2006, Washington DC, USA

[4] Jeníček, J.: Optimalizace kompresního systému COMPAS. Proc. of PAD 2006, Papradno, SK, pp. 77-82, ISBN 80-969202-2-7

[5] Jeníček, J.: Použití algoritmu COMPAS s různými ATPG, Proc. of PAD 2007, pp. 27-32, ISBN 978-80-7043-605-9

[6] Jeníček J., Novák O.:A Test Pattern Compression Based on Pattern Overlapping, Proc. of DDECS 2007, Apr. 2007, Krakow, Poland, pp.29 - 34, ISBSN: 1-4244-1161-0

[7] Novák, O., Plíva, Z., Jeníček, J., Mader, Z., Jarkovský, M.: Self-Testing SoC with Reduced Memory Requirements and Minimized Hardware Overhead. Acta Electrotechnica et Informatica, Vol. 8, No. 1, pp. 22-32, 2008, ISSN 1335-8243

[8] Jeníček, J.: Efficient Test Pattern Compression Method Using Hard Fault Preferring. Proc. of DSD 2008, September 2008, Parma, Italy

[9] Jeníček, J.: Nový kompresní algoritmus pro systém COMPAS. Proc. of PAD 2008, September 2008, Hejnice, Czech, pp. 81-88, ISBN 978-80-7372-378-1

[10] Novák, O., Jeníček, J.: Test Pattern Overlapping - a Promising Compression Method for Narrow Test Access Mechanism SOC Circuits, Radioelectronics & Informatics, No. 1, pp. 26-33, 2008, ISSN 1563-0064

Použitá literatura

[11] ATALANTA, FSIM a HOPE http://www.ee.vt.edu./ha/cadtools

[12] ISCAS 85 Benchmark http://www.cbl.ncsu.edu/www/CBL_Docs/iscas85.html

[13] ISCAS 89 Benchmark http://www.cbl.ncsu.edu/www/CBL_Docs/iscas89.html

[14] ITC 99 Benchmark http://www.cerc.utexas.edu/itc-99benchmarks/bench.html

[15] Jaderný, J: Evoluční a genetické algoritmy, ročníkový projekt, TU Liberec 2004

[16] Novák, O., Nosek, J.: Test Pattern Decompression Using a Scan Chain, proc. of IEEE International Symposium on Defects and Fault Tolerance in VLSI Systems 2001

[17] Zahrádka, J., Holubec, M., Novák, O.: COMPAS – System for Finding of a Compress Test Pattern Sequence, proc. of ECMS Liberec 2003

[18] Novák, O., Zahrádka, J., Plíva, Z.: COMPAS - Compressed Test Pattern Sequencer for Scan Based Circuits. EDCC2005, Lecture Notes in Computer Science 3463, pp. 403-414, Springer-Verlag 2005, ISSN 0302-9743

[19] Plíva, Z., Novák, O., Siekierska, K., Grodner, M.: Test Access Circuit for Education. proceedings of DDECS2005, Sopron, Hungary, April 2005, pp. 27-32, ISBN 963-9364-48-7

[20] Jarkovský, M., Plíva, Z., Novák, O.: Software for Test_Access Circuit for Education. Proceedings ECMS 2005, Electronique, Cotrôle, Modélisation, Mesure et Signal, 17-20 May 2005. Toulouse: Université Paul Sabatier

[21] Jarkovský, M.: Uživatelské rozhranní a řídicí programové vybavení pro diagnostiku SoC obvodů s využitím RESPIN architektury. Počítačové architektury & diagnostika 2005 (PAD 2005), str.61-65, 21.- 23.9.2005, Lázně Sedmihorky. ISBN 80-01-03298-1

[22] Mader, Z.: Diagnostika SoC obvodů s využitím RESPIN architektury. In: Seminář PAD, Moravany nad Váhom, Slovak Republic, Sept. 2004, ISBN 80-969202-0-0, s. 66-71

[23] Tvrdík, P.: Paralelní systémy a algoritmy, Vydavatelství ČVUT, 2002, ISBN 80-01-02267-6, 226 s.

[24] Podrazký, M: Analýza možností výstavby clusterů v operačním prostředí Linux, diplomová práce, Vojenská akademie v Brně, 2003

[25] Geist, A. et al.: PVM: Parallel Virtual Machine – A Users' Guide and Tutorial for

Networked Parallel Computing, The MIT Press, 1994, 279 s.

[26] Dorsch, R., Wunderlich, H.-J.: Reusing Scan Chains for Test Pattern Decompression, Proc. of the IEEE European Test Workshop 2001, pp. 24 – 32

[27] Chandra, A., Chakrabarty, K.: Test Data Compression for System-on-a-Chip Using Golomb Codes, Proc. of VTS Symp. 2000

[28] C Chandra, A., Chakrabarty, K.:Frequency/Directed Run Length (FDR) Codes with Application to System/on/Chip Test Sata Compression, Proc. of VLSI Test Symp., 2001, pp. 42-47

[29] Krishna, C.V., Touba, N.A.: Reducing Test Data Volume Using LFSR Reseeding with Seed Compression, Proc. of ITC 2002, pp. 321-330

[30] Hellebrand, S., Liang, H. G., Wunderlich, H. J.: A mixed mode BIST scheme based on reseeding of folding counters, Proc. of IEEE ITC, 2000

[31] Schafer, Dorsch, R., Wunderlich, H.J.: RESPIN++ - Deterministic Embedded Test, Proc. European Test Workshop, 2002, pp. 37-42

[32] ATALANTA-M http://service.felk.cvut.cz/vlsi/prj/Atalanta-M

[33] Holubec, M.: Komprese testovacích vektorů pro obvody se sériovým diagnostickým přístupem, diplomová práce, TU Liberec, 2004

[34] Fišer, P., Kubátová, H.: Multiple-Vector Column-Matching BIST Design Method, Proc. 9th IEEE Design and Diagnostics of Electronic Circuits and Systems 2006 (DDECS'06), Prague, CZ, 18.-21.4.2006, pp. 268-273

[35] OpenMP http://www.openmp.org/

[36] GCC - GNU Compiler Collection http://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc.pdf, 478 stran

[37] Fišer, P.: Mixed-Mode BIST Based on Column Matching, Počítačové Architektury & Diagnostika 2005, Lázně Sedmihorky, ČR, 21. – 23. 9. 2005, pp. 45-50

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

[39] A. Jutman, J. Raik, R. Ubar: SSBDDs: Advantageous Model and Efficient Algorithms for Digital Circuit Modeling, Simulation & Test, proc. of IWSBP'02, Freiberg, Germany, Sept. 19-20, 2002, pp. 157-166

[40] TurboTester http://www.pld.ttu.ee/tt/

[41] Gläser, U., Vierhaus, H. T. 1992. MILEF: an efficient approach to mixed level automatic test pattern generation. In Proc. of the Conference on European Design Automation, Los Alamitos, CA, pp. 318-321.

[42] L.H. Goldstein: Controllability/Observability Analysis of Digital Circuits, IEEE Trans. On Circuits and Systems, Vol. CAS-26, No. 9, pp. 685-693, September, 1979

[43] Marinissen, E. J., Zorian, Y., Kapur, R. Taylor T., and Whetsel. L.:Towards a Standard for Embedded Core Test: An Example. Proc. of ITC, pp. 616–627. IEEE, 1999.

[44] Bernhart et al.: OPMISR: the foundation for compressed ATPG vectors. Proc. ITC, 2001, pp. 748-757

[45] Abhijit Jas, Jayabrata Ghos-Dastir, and Nur A. Touba: Scan Vector Compression/Decompression Using Statistical Coding. Proc. VTS 1999

[46] Pandey, A. R., Patel, H. J.: Reconfiguration Technique for Reducing Test Time and Test Data Volume in Illinois Scan Architecture Based Designs. Proc. IEEE VLSI Test Symp, 2002, pp. 9-15

[47] Rajski, J. et al.: Embedded Deterministic Test. IEEE Trans. on CAD, vol. 23, No. 5, May 2004, pp. 776-792

[48] Mitra, S., Kim, K.S.: X-Compact: An Efficient Response Compaction for Test Cost Reduction, proc. of ITC 2002

[49] Jaderný, J.: Optimalizace parametru systému komprese diagnostických dat COMPAS, diplomová práce, TU Liberec 2005

[50] Jutman, A., Tsertov, A., Tsepurov, A., Aleksejev, I., Ubar, R., Wuttke, H.-D.: BIST analyzer: A training platform for SoC testing, proc. of FIE' 07, pp. S3H-8-S3H-13

[51] Vierhaus, H. T.: Introduction to IC test technology, Summer Academy „Design and Test Technology for Dependable Hardware / Software Systems", Cottbus, 2008

[52] Mader, Z.: Diagnostický systém jádrově založených SoC obvodů s nízkými nároky na paměť, disertační práce, TU Liberec, 2006

[53] Crouch, L. A.: Design for Test, Prentice Hall, 2000, ISBN 0-13-084827-1

Dodatek A – Přehled nastavení příkazové řádky

Formát příkazového řádku:

compas [volitelné nastavení] –a|–b <soubor>

Povinné nastavení:

Je nutné vybrat právě jednu z následujících možností.

-a <soubor> Načte soubor jménem „soubor.agm“, ve kterém je obsažen popis obvodu ve formátu programového balíku TurboTester. Jméno souboru může být zadáno s příponou i bez přípony.

-b <soubor> Načte soubor jménem „soubor.bench“, ve kterém je obsažen popis obvodu ve formátu programového balíku ATALANTA. Jméno souboru může být zadáno s příponou i bez přípony.

Volitelné nastavení ovlivňující načítání dat a kompresi:

-c n Nastavení konstanty kritéria. Hodnota „n“ musí být celé nezáporné číslo. Pokud není konstanta nastavena z příkazové řádky, program její hodnotu automaticky nastaví na polovinu počtu vstupů obvodu (počet vstupů obvodu je zjištěn při načítání popisu obvodu).

-f Použij predikci bitů. Doporučeno zapnout, umožní rychlejší a lepší kompresi.

-i <soubor> Načte počáteční stav skenovacího řetězce ze souboru. Základní nastavení je úplně vynulovaný skenovaní řetězec.

-p <soubor> Načte soubor určující pořadí vstupů obvodu. Základní nastavení je pořadí vstupů načtené z popisu obvodu.

-S x Podle „x“ vyber simulátor poruch:

Pro všechny typy popisu obvodů:

0 Spustí kompresi bez použití simulátoru poruch.

Pro BENCH popis obvodu:

f Použije vestavěný FSIM simulátor. Toto je základní automatická volba pro BENCH popis obvodu.

F Použije vnější simulátor FSIM. Toto nastavení vyžaduje přítomnost upravené verze simulátoru FSIM ve stejném adresáři, kde je umístěn COMPAS.

H Použije vnější simulátor HOPE. Toto nastavení vyžaduje přítomnost upravené verze simulátoru HOPE ve stejném adresáři, kde je umístěn COMPAS.

Pro AGM popis obvodu:

t Použije vestavěný simulátor SPSimul. Toto je základní automatická volba pro AGM popis obvodu.

T Použije vnější simulátor SPSimul. Toto nastavení vyžaduje přítomnost upravené verze simulátoru SPSimul ve stejném adresáři, kde je umístěn COMPAS.

-u Neukládat seznam dosud nedetekovaných poruch pro vnější simulátory poruch. Tuto volbu je možné nastavit pouze v kombinaci s některým z vnějších simulátorů poruch. Volba výrazně zpomaluje běh programu a slouží především pro ladicí účely.

Volitelné nastavení ovlivňující výstup programu:

-l Vygeneruj soubor se statistikou o běhu programu. V případě již existujícího souboru je stávající soubor bez dotazu přepsán.

-n n Po uplynutí každých „n“ sekund uloží aktuální délku komprimované posloupnosti do souboru, který má shodný název jako zpracovávaný obvod a příponu „patt“. Po dokončení běhu programu je v souboru zapsána výsledná délka komprimované posloupnosti. V případě již existujícího souboru je stávající soubor bez dotazu opakovaně přepisován.

-r Uloží výslednou komprimovanou testovací posloupnost ve formě testovacích vektorů. Soubor má formát podle programového balíku ATALANTA. V případě již existujícího souboru je stávající soubor bez dotazu přepsán.

-s Uloží výslednou komprimovanou testovací posloupnost jako proud jedniček a nul. V případě již existujícího souboru je stávající soubor bez dotazu přepsán.

-t V průběhu komprese ukládá detailní časovou statistiku běhu jednotlivých částí algoritmu. Soubor má shodný název jako zpracovávaný obvod a příponu „csv“. V případě již existujícího souboru je stávající soubor bez dotazu přepsán.

-T Uloží výslednou komprimovanou testovací posloupnost ve formě testovacích vektorů. Soubor má formát podle programového balíku TurboTester. V případě již existujícího souboru je stávající soubor bez dotazu přepsán.

-v n Nastavuje detailnost výpisů v průběhu komprese podle hodnoty „n“. Program vypisuje všechna hlášení až do hodnoty „n“.

0 Bez přístupu na konzoli. Program nemá možnost vypsat žádné, ani chybové, hlášení.

1 Vypisuje pouze kritické chyby.

2 Výpis varovných hlášení.

3 Vypíše krátkou statistiku běhu programu. Toto je standardní úroveň výpisů.

4 Vypisuje dodatečné informace pro každý krok komprese, ve kterém se podaří snížit počet zbývajících nekomprimovaných dat.

5 Vypisuje dodatečné informace pro každý krok komprese.

6 Přidává výpis informačních hlášení jednotlivých rutin programu. Tato volba slouží převážně k ladění programu.

7 Vypisuje všechny ladicí informace. Tato volba vypisuje extrémní množství informací a slouží jen pro ladění programu.

s Způsobí vypsání statistických informací o načtených datech a obvodu a ihned ukončí program. Komprese dat tedy není prováděna! (Ve skutečnosti je v tomto případě nastavena úroveň výpisů 5, a program je ukončen po načtení vstupních dat.)

-W Potlačí výpis varovných hlášení

Pokud je některá vlastnost nastavena opakovaně (např. „–v 3 –v 5“), v úvahu je brána poslední volba. Pro daný příklad je tedy nastavena úroveň výpisů 5.

Příklad:

compas -b s27 -s -f

Zkomprimuj testovací data pro obvod ze souboru „s27“, popis obvodu je ve formátu BENCH. Výsledek ulož jako proud jedniček a nul. Použij předpovídání bitů.