VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

METODIKA APLIKACE TESTU OBVODU ZALOŽENÁ NA IDENTIFIKACI TESTOVATELNÝCH BLOKŮ

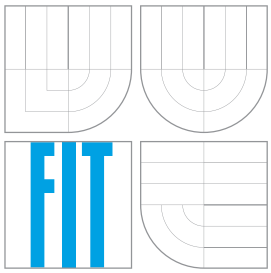
DIZERTAČNÍ PRÁCE
PHD THESIS

AUTOR PRÁCE
AUTHOR

Ing. TOMÁŠ HERRMAN

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

METODIKA APLIKACE TESTU OBVODU ZALOŽENÁ NA IDENTIFIKACI TESTOVATELNÝCH BLOKŮ

TEST APPLICATION METHODOLOGY BASED ON THE IDENTIFICATION OF TESTABLE BLOCKS

DIZERTAČNÍ PRÁCE
PHD THESIS

AUTOR PRÁCE
AUTHOR

Ing. TOMÁŠ HERRMAN

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. ZDENĚK KOTÁSEK, CSc.

BRNO 2010

Abstrakt

Dizertační práce se zabývá analýzou číslicových obvodů popsaných na úrovni meziregistrových přenosů. Je v ní zahrnuta pouze problematika související s testovatelností obvodových datových cest, řadičem ovládajícím tok dat těmito cestami se nezabývá. Stěžejní částí práce je návrh konceptu testovatelného bloku (TB), pomocí něhož se obvod rozdělí na části, jež jsou plně testovatelné přes jejich vstupy a výstupy, přes takzvané hraniční registry bloku nebo primární vstupy/výstupy. Přínosem nové metodiky je také redukce počtu registrů v řetězci scan, do něhož jsou zařazeny pouze hraniční registry. Segmentací obvodu dosáhneme také zjednodušení generování testu rozdělením tohoto problému na více menších částí. Navržená metodika pro identifikaci TB v číslicovém obvodu využívá dvou vybraných evolučních algoritmů operujících na formálním modelu obvodu na úrovni RT.

Klíčová slova

číslicový obvod, testovatelný blok, řetězec scan, optimalizace, genetický algoritmus, simulované žíhání, rozdělení obvodu

Abstract

The PhD thesis deals with the analysis of digital systems described on RT level. The methodology of data paths analysis is decribed, the data path controller analysis is not solved in the thesis. The methodology is built on the concept of Testable Block (TB) which allows to divide digital component to such segments which can be tested through their inputs/outputs, border registers and primary inputs/outputs are used for this purpose. As a result, lower number of registers is needed to be included into scan chain - border registers are the only ones which are scanned. The segmentation allows also to reduce the volume of test vectors, tests are generated for segments, not for the complete component. To identify TBs, two evolutionary algorithms are used, they operate on TB formal model which is also defined in the thesis.

Keywords

digital circuit, testable block, scan chain, optimalization, genetic algorithm, simulated annealing, circuit partitioning

Citace

Tomáš Herrman: Metodika aplikace testu obvodu založená na identifikaci testovatelných bloků, dizertační práce, Brno, FIT VUT v Brně, 2010

Prohlášení

Prohlašuji, že jsem tuto dizertační práci vypracoval samostatně pod vedením pana Doc. Ing. Zdeněka Kotáska, CSc. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Herrman
14. října 2010

Poděkování

Děkuji svému školiteli docentu Zdeňku Kotáskovi za skvělé vedení během mého studia, za cenné rady, připomínky a komentáře k mé práci. Dále děkuji svým rodičům a prarodičům za finanční i morální podporu během mého studia, bez níž by tato práce nemohla vzniknout. Tato práce byla podporována grantovou agenturou GAČR v rámci grantu GD102/05/H050 – Integrovaný přístup k výchově studentů DSP v oblasti paralelních a distribuovaných systémů a GA102/04/0737 - Moderní metody syntézy číslicových systémů. Dále MŠMT v rámci grantu MSM0021630528 - Výzkum informačních technologií z hlediska bezpečnosti.

Obsah

Kapitola 1

Úvod

Už při výrobě elektronické komponenty mohou vzniknout závady, které je nutné co nejdříve odhalit. Proto je nutné vybavit obvody diagnostickými prvky, se kterými se musí počítat již při návrhu integrovaného obvodu. Tyto prvky pak umožní testování obvodu jak při výrobě, tak při ověřování funkčnosti obvodu v praktickém použití.

Menší obvody se dříve mohly testovat přímo při výrobě a nebylo nutné při návrhu počítat s nutností zavedení podpůrných prostředků pro test, což bylo možné díky jednoduchosti jejich struktury. Obvod většinou představoval jeden logický celek se vstupy a výstupy, bylo tak možné vkládat testovací vektory přímo na vstup a analyzovat výstup.

Dnes dosahují integrované obvody takových složitostí, že je nutné zařadit do etapy návrhu i úvahy o tom, jak bude navrhovaný obvod testován. Složitost obvodu tím samozřejmě narůstá, proto úvahy o způsobu implementace těchto principů do výsledné obvodové struktury jsou důležitým krokem a mohou (i výrazným způsobem) ovlivnit cenu. K tomuto účelu se používají různé metody, které se pak využívají při syntéze obvodu. Syntéza obvodu je proces, při němž je obvod navrhován pomocí CAD nástrojů tak, aby plnil požadovanou funkci a přitom vyhovoval různým kriteriím. Z hlediska diagnostiky je důležitým kriteriem testovatelnost obvodu.

Nefunkčností obvodu způsobenou chybným návrhem se zabývá samostatná disciplína zvaná verifikace návrhu. Toto pojednání se však zabývá principy aplikace testu, konkrétně způsobem aplikace testu na úrovni RT (Register Transfer - meziregistrové přenosy). Pro zvýšení testovatelnosti jsou jednotlivé registry zřetězeny do registru scan, což je v podstatě propojení registrů do posuvného registru, který má jeden sériový datový vstup a jeden sériový datový výstup. Jedna z možností spočívá v použití metody úplný scan, při níž jsou při aplikaci testu všechny registry (klopné obvody) propojeny do posuvného registru, přes který se vkládají na vstupy testovaných vnitřních prvků testovací vektory, resp. přes který se snímají odezvy na tyto vektory. Metoda úplný scan má dvě zásadní nevýhody: doba potřebná pro aplikaci testu je dlouhá (veškeré přesuny diagnostických dat se realizují sériově) a náklady na realizaci obvodu jsou vysoké (konstrukce klopného obvodu je složitější než u běžného typu klopného obvodu). Tomu se chceme vyhnout a redukovat počet registrů zapojených do posuvného registru. Touto technikou se zabývá např. [7, 12, 20, 37, 39, 41, 42, 56]. Snahou této práce je využití formálního přístupu k řešení zmíněného problému.

V práci bude navrhnut koncept testovatelného bloku (TB), pomocí něhož se obvod rozdělí na části, jež jsou plně testovatelné přes jejich vstupy a výstupy, přes takzvané hraniční registry bloku nebo primární vstupy/výstupy. Přínosem nové metodiky je také redukce počtu registrů v řetězci scan, do něhož jsou zařazeny pouze hraniční registry. Segmentací obvodu dosáhneme také zjednodušení generování testu rozdělením tohoto problému

na více menších částí. V práci je prezentována metodika pro identifikaci TB v číslicovém obvodu, jež využívá evoluční algoritmus operující na formálním modelu obvodu na úrovni RT.

1.1 Struktura dizertační práce

Struktura dizertační práce je volena tak, aby reflektovala současný stav vědění v tématu dizertační práce a obsahovala všechny důležité informace popisující výzkumnou činnost realizovanou v rámci dizertační práce a její výsledky.

Po úvodu následuje druhá kapitola, ve které jsou uvedeny základní pojmy, jež bylo nutné nastudovat. Stručně jsou zmíněny možné úrovně popisu číslicového obvodu, výrazně větší část je věnována popisu na úrovni meziregistrových přenosů, protože s tímto typem obvodu se v práci pracuje - pro tuto úroveň je v práci vytvořen formální model analyzovaného obvodu. Další část je proto věnována právě této úrovni popisu. Následuje popis základních pojmů z oblasti diagnostiky číslicových obvodů. V poslední části kapitoly je uveden základní princip genetického algoritmu a simulovaného žíhání.

Třetí kapitola se věnuje současnému stavu řešené problematiky a technikami blízce souvisejícími s problematikou řešenou v této práci. Jsou v ní uvedeny základní principy návrhu obvodů pro snadnou testovatelnost a existující metodiky, jež jsou blízké analyzovanému problému.

Ve čtvrté kapitole jsou popsány cíle práce spolu s motivací pro takto definované cíle.

Pátá kapitola je věnována základnímu formálnímu modelu založeném na koncepci i−cest, na němž je řešení vystavěno. Model je konstruován pomocí definic a vět a doprovázen příklady, jež je blíže osvětlují. Další část je věnována přenosu diagnostických informací obvodem s využitím transparentních módů prvků. V poslední části je navrženo rozšíření modelu o podporu invertovaných i-cest.

V šesté kapitole je představen formální model TB, s nímž zde prezentovaná metodika pracuje a příklady vysvětlující dané definice. Poté následuje popis a definice metodiky identifikace TB, jejímž cílem je uplatnění tohoto pojmu při analýze číslicového obvodu. Jsou zde uvedeny stěžejní algoritmy a postup řešení při rozdělení obvodu na TB. Následně je prezentován problém smyček v obvodu a navrhnut postup pro identifikaci smyček a jejich následnou eliminaci. Následuje část zabývající se implementací popsané metodiky. Zde je zdůvodněna volba genetického algoritmu a simulovaného žíhání, popsán a vývojovým diagramem znázorněn způsob výpočtu fitness funkce pro genetický algoritmus a simulované žíhání. Dále je popsána implementace metodiky detekce zpětnovazebních smyček a jejich přerušení. V poslední části této kapitoly je shrnuta implementace a postup zavedené metodiky.

Sedmá kapitola shrnuje experimentální výsledky a porovnání s profesionálně používaným návrhovým systémem.

V závěrečné kapitole jsou shrnuty výsledky a dosažené cíle dizertační práce.

Kapitola 2

Základní pojmy

V této kapitole budou uvedeny základní informace, na nichž je metodika vystavěna nebo je používá. Bude objasněn pojem logický obvod. Dále budou uvedeny úrovně popisu logického obvodu, pozornost bude zaměřena na obvody na úrovni meziregistrových přenosů (RT). V následující sekci bude diskutován pojem diagnostika, následují pak obvody se snadnou testovatelností a generátory testu. V poslední části budou představeny dva algoritmy: genetický algoritmus a simulované žíhání, které jsou využity v metodice, jež rozděluje obvod na TB.

2.1 Logické obvody

V [10] je *signál* definován jako booleovská proměnná, která může mít pouze dvě hodnoty reprezentované symboly 0 a 1. Dále definuje *hradlo*, což je jednoduchý elektrický prvek, který provede logickou operaci s jedním nebo více vstupními signály a vytvoří výstupní signál. Typická hradla používaná v logických obvodech jsou AND (konjunkce), OR (disjunkce), NAND (negovaná konjunkce), NOR (negovaná disjunkce) a NOT (negace vstupu). Další používaná hradla jsou XOR (exclusive-OR) a XNOR (exclusive-NOR).
Základní *logický obvod* je pak modelován jako propojení několika hradel. V pokročilejších logických obvodech pak vystupují složitější komponenty[1] jako jsou: klopný obvod, multiplexor, registr, čítač, dekodér, kodér, ap.

2.2 Úrovně popisu obvodu

Za současného stavu technologie není možné při návrhu složitých obvodů postupovat ručně. Proto vzniká řada automatizovaných návrhových systémů a nástrojů podporujících návrh číslicových obvodů. Návrhář pak není nucen zabývat se detailně strukturou jednotlivých komponent obvodu, soustředí se na korektní a přesnou specifikaci funkce obvodu a rutinní činnosti přenechá návrhovému systému. Proces, kterým systém tvoří obvod ze specifikace návrháře, se nazývá *syntéza*.

Návrhář popíše chování navrhovaného obvodu (behaviorální popis), návrhový systém pak vygeneruje strukturu obvodu. Forma popisu obvodu bývá zpravidla textová, pak lze hovořit o jazyce pro popis obvodů. Typickými představiteli těchto jazyků jsou např. VHDL, Verilog a Abel. V menší míře se užívá grafická forma popisu chování, např. stavový diagram konečného automatu. Od specifikace obvodu např. schématem se dnes ustupuje zejména

[1] lze je modelovat převedením na základní hradla, např. RS klopný obvod pomocí dvou hradel NAND

proto, že taková forma jednak nutí návrháře zamýšlet se nad strukturou (což při komplexnosti dnešních obvodů představuje značný problém), hlavně však komplikuje automatizované zpracování, verifikaci návrhu a jiné činnosti související s návrhem.

Další rozdělení můžeme odvodit od úrovně abstrakce popisu systému, pak můžeme rozlišit několik úrovní popisu systému - počet a pojmenování těchto úrovní není jednotné a často se v literatuře liší. V [13] jsou rozlišeny následující úrovně popisu číslicových systémů:

1. *Systémová úroveň* (System Level - SYL) - na této úrovni je systém modelován pomocí tzv. zdrojů (sběrnice, procesory, paměti, ...) např. za účelem analýzy efektivnosti systému s ohledem na využití zdrojů při různých úlohách nebo definování komunikačního protokolu mezi zdroji a algoritmem činností jednotlivých zdrojů.

2. *Úroveň instrukční sady* (Instruction Set Level - ISL) - tato úroveň popisu je vhodná zejména k definici a simulaci instrukcí procesoru. Instrukce jsou v simulaci vybírány a dekódovány s ohledem na jejich formát. K simulaci provádění instrukcí je obvykle použito aritmetických a logických operací.

3. *Strukturální úroveň* (Structural Level - STL) - tato úroveň popisuje systém pomocí vzájemně propojených komponent. Každá komponenta může být tvořena několika menšími, vzájemně propojenými podkomponentami, ty zase podrobnějšími podkomponentami.

4. *Úroveň meziregistrových přenosů* (Register-Transfer Level - RTL) - pro systém na této úrovni je typické, že je složen ze dvou částí a to datových cest a obvodového řadiče ovládajícího tok dat těmito cestami. Datová část pak obvykle sestává z funkčních bloků oddělených registry nebo paměťmi a z propojovacích prvků - obvykle multiplexorů a sběrnic. Obvodový řadič bývá realizován stavovým automatem. Velmi často je tento typ popisu používán jako vstup logické syntézy [50]. Protože způsob propojení funkčních bloků, paměťových prvků a propojovacích prvků je znám, lze popis na této úrovni chápat jako strukturální popis.

5. *Úroveň hradel* (Gate Level - GAL) - jedná se o další speciální případ strukturálního popisu. Návrh je tvořen vzájemným propojením prvků nízké úrovně - obvykle logických hradel nebo klopných obvodů. Tento popis bývá výstupem logické syntézy nebo nástroje provádějícího rozmístění a je použit buď jako doprovodný popis k seznamu spojů nebo jako model struktury návrhu pro účely simulace.

6. *Úroveň tranzistorů* (Transistor Level - TRL) - opět se jedná o speciální případ strukturálního popisu - návrh je tentokrát tvořen vzájemným propojením tranzistorů (např. unipolární technologie CMOS). Na této úrovni popisu nebývají obvody současných složitostí a rozměrů navrhovány přímo návrhářem - tento popis je obvykle součástí automatizované implementace systému, jejímž hlavním cílem je příprava masky k výrobě čipu. I na této velmi nízké úrovni popisu je však někdy prováděna (implementačně velmi přesná, avšak také výpočetně náročná) simulace a i pro tuto úroveň existují modely poruch.

2.3 Obvod na úrovni RT

Model na úrovni RT (meziregistrových přenosů) se získává při syntéze obvodu na vyšší úrovni popisu. K syntéze se používají automatizované prostředky. Zde už má smysl uplatňovat

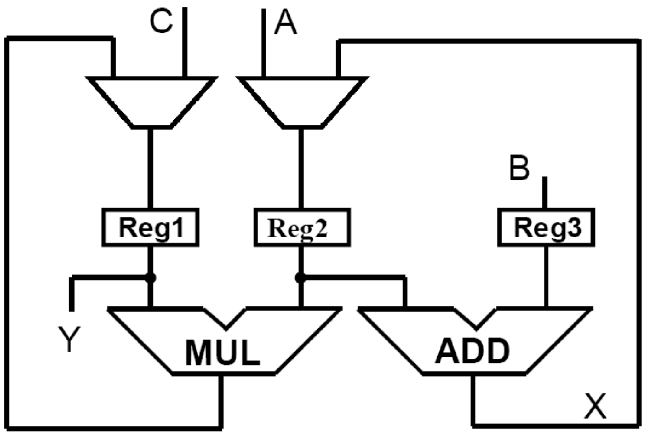
úvahy o diagnostice navrhovaného obvodu, respektive o testovatelnosti vznikající struktury. Je třeba podotknout, že výsledkem syntézy obvodu na vyšší úrovni popisu (High-Level Synthesis - HLS) je struktura datových cest obvodu a dále pak řídicí jednotka (řadič - obvykle reprezentován jako stavový automat), která řídí výpočet – tok dat datovými cestami. Vzhledem k tomu, že se práce zabývá testovatelností datových cest, bude nadále zmiňována pouze datová část. V návrhovém systému pak pokračuje syntéza obvodu přes úroveň hradel, případně úroveň tranzistorů až do fáze generování rozvržení čipu.

Jedná se o abstraktní úroveň modelování integrovaného číslicového obvodu, která nese informaci o struktuře obvodu a z níž je možné zjišťovat souběžné cesty, jejichž existence může ovlivnit testovatelnost obvodu. Obvod na úrovni RT se skládá z funkčních jednotek, paměťových prvků (jsou to registry nebo paměťové moduly), propojovacích prvků (sběrnice, multiplexory) a vícebitových spojů [47]. Registry nesou stavovou informaci obvodu a jsou důležitým prvkem při testování. Dále jsou ještě na úrovni RT zahrnuty řídicí vodiče (cesty), které slouží k adresování paměťových prvků, přepínání multiplexorů nebo budičů sběrnice a přivádí operační kódy k aritmeticko-logickým jednotkám. Obvod na úrovni RT pracuje v diskrétním čase (nezávisle na implementaci hodin).

Pro propojování se nejvíce používají dvě strategie: multiplexové datové cesty a obousměrné sběrnice [55].

Pro multiplexové datové cesty je typické (viz. obrázek 2.1):

- aritmetickologické operace jsou realizovány čistě kombinačními obvody,
- pro uložení dat se používají registry,
- přepínání spojů je realizováno multiplexory (vytvoření datové cesty),
- všechny registry jsou řízeny jedním společným synchronizačním signálem.

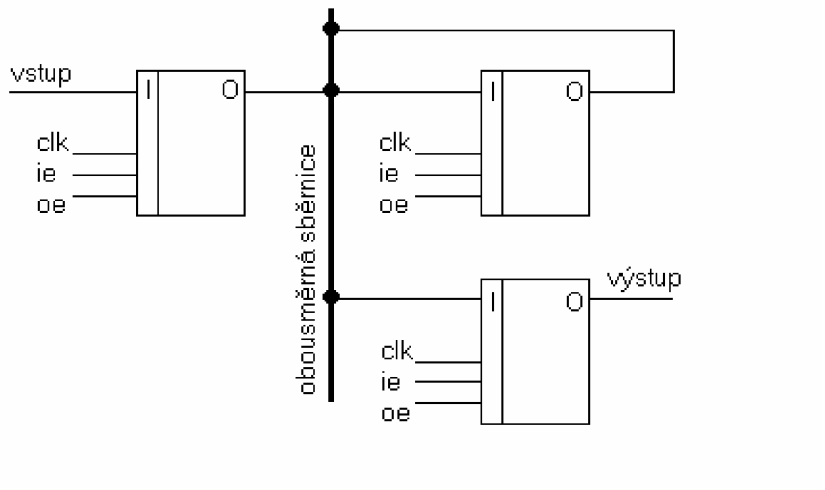


Obrázek 2.1: Příklad obvodu na úrovni RT s multiplexovými datovými cestami

Pro strategii s obousměrnými sběrnicemi jsou typické tyto skutečnosti (viz. obrázek 2.2):

- funkční jednotky mají paměťové schopnosti, na jejich vstupech a výstupech jsou záchytné registry,

- většinou se užívají registrová pole s jedním výstupem a vstupem pro všechny registry,
- je možné použít dvoufázové hodiny.



Obrázek 2.2: Příklad obvodu na úrovni RT s obousměrnými sběrnicemi

2.4 Diagnostika číslicových obvodů

Diagnostika se zabývá analýzou technického stavu číslicového obvodu či systému (systémem může být např. deska), jejím cílem je zjistit na základě odezev systému na vstupní stimuly (tzv. testovací vektory), zda je, či není v systému porucha, popř. i ve které části systému tato porucha nastala. V této formulaci jsou obsaženy dvě základní úlohy diagnostiky, a to detekce poruchy a její lokalizace. Původní snahou bylo používat postupů, při nichž není nutná demontáž součástek a vystačit si pouze s přístupem k vývodům čipu, tzv. primárním vstupům a primárním výstupům. Brzy se však ukázalo, že s tímto omezením není možné vždy vystačit[2] a že je nutné si tuto úlohu usnadnit přístupem k některým vnitřním bodům obvodu, k tzv. testovacím bodům. Tento přístup je obvykle podmíněn zásahem do původního návrhu. Z tohoto pohledu pak už nehovoříme pouze o návrhu, ale o návrhu pro snadnou testovatelnost[3] (viz. kapitola 2.6), ten je vykoupen změnou parametrů obvodu, jako jsou: dynamické parametry, nárůst plochy či počtu vývodů obvodu. Další formou aktivního diagnostického přístupu je navrhování bezpečných obvodů, na které navazuje navrhování systémů odolných proti poruchám.

Při testování číslicového obvodu je nutné vytvořit (vygenerovat) posloupnost testovacích vektorů a zároveň je nutné definovat způsob aplikace testu. Testovací vektory a odezvy na ně budeme označovat jako *diagnostická data*. Manuální přístup se dnes již téměř nepoužívá, místo toho se používají automatizované prostředky, tzv. *generátory testu* (GT). Tyto se dělí na *generátory testu pro kombinační obvody* (GTKO) a *generátory testu pro sekvenční obvody* (GTSO). My budeme pracovat s generováním deterministického tzv. hierarchického testu

[2] skupinou obvodů, jejichž testování silně závisí na nastavení a zjištění jejich vnitřního stavu, jsou sekvenční obvody

[3] anglicky Design for Testability DfT

pro obvod na úrovni RT založeném na generování dílčích lokálních testů pro komponenty obvodu a jejich transformaci na globální test, jak je uvedeno v [2, 45, 51, 53, 54].

Principy diagnostiky se klasifikují podle několika hledisek. První klasifikace se odvozuje od způsobu získávání testovacích vektorů a druhá od způsobu vyhodnocování správnosti odezev. Generování vstupní testovací posloupnosti probíhá buď předem (off-line) nebo v průběhu testu (on-line), a to buď programovými, nebo technickými prostředky. Vyhodnocování správnosti odezev probíhá buď srovnáváním se správnými odezvami, čtenými z paměti, nebo srovnáváním s kombinacemi získanými z etalonu, který se testuje současně. Tento způsob je označován jako tzv. komparační (srovnávací) test. Nejčastější klasifikace forem diagnostiky vychází z časového uspořádání testování, kdy rozlišujeme diagnostiku periodickou a průběžnou. Dále pak rozeznáváme diagnostiku vnější a vnitřní.

2.5 Detekce a lokalizace poruch

Diagnostické testy provádějí buď detekci poruchy nebo její lokalizaci. Tyto testy se podle toho nazývají *detekční* nebo *lokalizační testy*. Snahou je nalézt co nejpřesnější *model poruch*, který bude ovšem stále svým rozsahem použitelný pro další práci. Mezi přesností a velikostí modelu je nutné nalézt vhodný kompromis.

Snažíme se nalézt test, který je *úplný*, což znamená že pokrývá 100% poruch[4] a také se snažíme nalézt test *minimální*, jež by byl tvořen minimálním počtem testovacích vektorů. Opět je zde nutné najít kompromis, který bere v potaz i cenu takového testu.

2.6 Obvody se snadnou testovatelností a jejich návrh

S rostoucí složitostí navrhovaných obvodů a se zvyšujícím se tlakem na zkrácení procesu návrhu a přípravy sériové výroby vystupuje do popředí požadavek, aby navrhované obvody bylo možné co nejsnáze testovat. Dále tu jsou důvody ekonomické: v procesu návrhu i výroby je žádoucí detekovat každou případnou poruchu co nejdříve. Později nalezená porucha znamená značně zvýšené náklady na její odstranění. Cílem návrhu a hlavně syntézy je, aby jejím výstupem byl obvod plnící požadovanou funkci, který by zároveň vyhovoval různým návrhovým kritériím. Mezi ně patří např. cena obvodu, počet vývodů, dynamické parametry a plocha na křemíkovém plátku.

Z hlediska uplatnění požadavku diagnostiky je dalším kritériem testovatelnost obvodu. Tu je možno chápat především jako snahu o zvýšení řiditelnosti/pozorovatelnosti vnitřních uzlů obvodu. Bez využití metod návrhu pro snadnou testovatelnost [3, 6, 14, 46] si žádný složitější návrh nelze představit. Tato disciplína prolíná všemi etapami návrhu počínaje systémovým návrhem až po testování vyrobených produktů. Testovatelností na úrovni RT se zabývá např. [29, 40, 53, 54].

2.7 Nástroje pro automatické generování testu (GT)

Pod pojmem GT budeme v tomto textu rozumět programový balík, v němž je implementována konkrétní metodika, nebo více metodik pro generování testovacích vektorů. Jeho úkolem je nalézt vstupní (testovací) posloupnost tak, aby při aplikaci této sekvence na obvod bylo možné testerem rozhodnout, je-li tento obvod v pořádku nebo vykazuje poruchy. Vygenerované testovací vektory jsou využívány již při výrobě polovodičových čipů, pro

[4] pokrytí poruch vyjadřuje v procentech, jak velká část poruch ze všech možných je testována

jejich otestování před expedicí. Efektivnost GT je měřena pokrytím poruch na počet generovaných testovacích vektorů, efektivnější GT se snaží generovat co nejméně testovacích vektorů a pokrýt co nejvíce poruch. Tato metrika pak dává větší číslo při vyšším pokrytí poruch a nižším počtu testovacích vektorů. Doba potřebná pro ruční generování testovacích vektorů bývá dlouhá, [15] uvádí půl roku až rok a půl. Použití GT zkrátí tuto dobu na dny až týdny.

2.7.1 GT pro kombinační obvody (GTKO)

Standartně sestává proces generování testovacích vektorů pomocí GTKO[5] z následujících tří činností. První činnost je výběr poruchy z předem daného seznamu poruch, jež je nutné detekovat. Pro tuto poruchu budeme vytvářet testovací vektor. Tato volba má veliký vliv na dobu potřebnou pro generování testu, je proto nutné výběr provádět ve vhodném pořadí. Vliv tohoto výběru je značný a může představovat až padesátiprocentní pokles doby nutné pro generování testu [15]. Při výběru poruch ze seznamu je také vhodné počítat s tím, že nastavení detekční cesty představuje mnohem náročnější problém než nastavení cesty, jež aktivuje poruchu. Doporučuje se proto nejprve detekovat poruchy co nejblíže primárním výstupům obvodu.

Druhým krokem je aktivace poruchy. Aktivací poruchy rozumíme nastavení signálu v bodě s poruchou na opačnou hodnotu než představuje porucha, kterou chceme detekovat. Pokud například chceme aktivovat poruchu trvalá 0 v bodě a, znamená to nastavit hodnotu logická 1 v bodě a. Nastavení logické hodnoty v daném místě představuje procházení struktury obvodu a vyhledání vhodné kombinace vstupů, kterou dosáhneme, nastavení požadované hodnoty signálu v daném místě.

Posledním krokem je sestavení detekční cesty, přes kterou je možné detekovat poruchu na primárním výstupu obvodu. Pro sestavení této cesty je nutné uvést prvky po cestě z bodu s aktivovanou poruchou k primárnímu výstupu do transparentního režimu.

Tímto způsobem lze detekovat všechny poruchy typu trvalá 0, trvalá 1. Postup detekce je považován za vyřešený [30, 59, 61].

2.7.2 GT pro sekvenční obvody (GTSO)

Generování testovacích vektorů pro sekvenční obvody je mnohem obtížnější úloha, protože výstup sekvenčního obvodu je dán nejen kombinací vstupních hodnot, ale i aktuálním stavem obvodu, jež je uchováván v jednotlivých paměťových prvcích obvodu. Úloha vygenerování testu pro sekvenční obvody představuje NP-úplný problém, což bylo dokázáno v [17].

Většina sekvenčních generátorů pracuje na úrovni hradel. I když máme dnes k dispozici obrovskou výpočetní sílu, vytváření testu pro složité obvody na úrovni hradel je stále nevyřešenou otázkou. Jednou z možností řešení tohoto problému je použít některou z technik pro snadnou testovatelnost například techniku úplný scan, viz. kapitola 3.2. Obecně užívaná zkratka pro generování/generátor testu pro sekvenční číslicové obvody je GTSO[6]. GTSO pro složitější obvody využívající techniky scan je popsán např. v [63].

[5] anglicky ATPG („Automatic Test Pattern Generation“ nebo „Automatic Test Pattern Generator“ - automatické generování testovacích vektorů nebo automatický generátor testovacích vektorů)

[6] anglicky SATPG („Sequential Automatic Test Pattern Generation/Generator“)

2.8 Genetické algoritmy (GA)

V metodice, která je v této práci prezentována, je využit genetický algoritmus, proto je zde uvedena stručná informace o této technice.

Evoluční algoritmy [9, 18, 43, 16] jsou založeny na formalizaci Darwinovy evoluční teorie, která byla převzata do informatiky z biologie. Genetický algoritmus je jeden druh evolučního algoritmu.

Evoluční algoritmy byly představeny v roce 1960 I. Rechenbergem a publikovány v jeho knize nazvané „Evoluční strategie“ (v originále Evolutionsstrategie) v roce 1973 [57]. Jeho myšlenka byla později rozvíjena dalšími výzkumníky. Genetický algoritmus byl vytvořen Johnem Hollandem a implementován společně s jeho studenty a kolegy. Publikován pak byl v knize „Adaption in Natural and Artificial Systems“ v roce 1975 [28].

Genetické algoritmy představují univerzální výpočetní prostředek, který je velmi robustní a dokáže zpracovat velké množství parametrů, vztahů a druhů závislostí. Rozsah řešitelnosti problémů pomocí genetických algoritmů je oproti klasickým metodám mnohem větší. Důvodem takové výkonnosti je fakt, že řešení systému se postupně vyvíjí a výpočet se díky zpětné vazbě získané z velikosti hodnoty fitness funkce jednotlivých jedinců dokáže přizpůsobit. Genetické algoritmy se používají k nalézání průchodných řešení různě složitých systémů, nebo k hledání suboptimálních a optimálních řešení. Stávají se tak mocným nástrojem v oblasti, která je velmi populární a zároveň problematická.

V nauce o GA se vyskytují následující pojmy:

Chromozom (kolekce genů v genotypu) je řetězec pevné délky a představuje kandidátní řešení problému. *Chromozomy* jsou uspořádány v *populaci*. Darwinova evoluční teorie se zakládá na tezi přirozeného výběru, podle které přežívají nejpřizpůsobivější jedinci populace. Reprodukcí (jinak zvané křížení, rekombinace) dvou jedinců s vysokým *fitness* dostáváme potomky, kteří budou s vysokou pravděpodobností dobře přizpůsobeni k přežití. Samotná reprodukce však není dostatečně efektivní pro vznik dobře přizpůsobených jedinců, a proto je nutné zapojit mutace, které náhodně ovlivňují genetický materiál populace. V biologii je *fitness* definována jako relativní schopnost přežití a reprodukce genotypu v daném prostředí. V evolučních algoritmech *fitness funkce* reprezentuje prostředí a přiřazuje *chromozomu* (obvykle kladné) reálné číslo, které vyjadřuje jeho kvalitu (čím vyšší číslo, tím je jedinec lepší). Evoluční algoritmus produkuje posloupnost populací (každou jako sadu jedinců) a končí nalezením dostatečně kvalitního řešení (chromozomu) nebo vyčerpáním dovoleného počtu populací. V případě genetického algoritmu se ještě před výpočtem *fitness* obvykle aplikuje zobrazení, které vytváří z genotypu (zakódovaného řešení) fenotyp (vlastní řešení).

Algoritmus 2.8.1. Genetický algoritmus

1. [Start] Generuj náhodnou populaci n chromozomů.
2. [Fitness] Vypočítej fitness $f(x)$ každého chromozomu x v populaci.
3. [Nová populace] Vytvoř novou populaci opakováním následujících kroků, dokud není nová populace kompletní.
   (a) [Selekce] Vyber dva rodičovské chromozomy z populace vzhledem k jejich hodnotě fitness (chromozomy s vyšší hodnotou mají větší šanci výběru).

(b) [Křížení] S určitou pravděpodobností křížení proveď křížení vybraných rodičů, tím vzniknou dva potomci. Pokud křížení neproběhne, jsou potomci kopií rodičů.

(c) [Mutace] S určitou pravděpodobností pro mutaci zmutuj oba potomky.

(d) [Přijetí] Vlož nové potomky do populace.

4. [Nahrazení] Použij novou populaci k dalšímu běhu programu.

5. [Test] Pokud je splněna podmínka ukončení, zastav a vrať nejlepší řešení z populace.

6. [Smyčka] Vrať se do bodu 2.

Někdy je při nahrazování použit takzvaný *elitizmus*, kdy jsou z nové populace do populace vkládána pouze lepší řešení.

Proces selekce lze provést takzvanou *ruletou*, kdy je každému chromozomu přidělena určitá výseč v kruhu. Velikost výseče závisí na velikosti *fitness*. Nebo lze použít *výběr podle pořadí*, kdy je velikost výseče závislá na pořadí chromozomu při třídění podle *fitness*.

Další základní informace jsou též v [52].

2.9 Simulované žíhání

Simulované žíhání je jednou z heuristických metod používaných k nalezení globálně optimálního, případně suboptimálního řešení složitých kombinatorických úloh. Tyto úlohy jsou většinou NP-úplné. To znamená, že mají velmi rozsáhlý prostor přípustných řešení a není možné v rozumném čase všechna řešení otestovat a najít nejlepší řešení (globální optimum). Metoda simulovaného žíhání má fyzikální základ a inspiruje se tzv. horolezeckým algoritmem, který systematicky prohledává stavový prostor všech řešení a snaží se najít řešení nejoptimálnější. Jde vlastně o variantu gradientové metody bez gradientu, kdy se směr nejprudšího spádu určí prohledáním okolí.

Aby bylo možné posoudit, jak je určité řešení kvalitní, musí existovat funkce, která je schopna ohodnotit jakékoliv řešení patřící do prostoru všech řešení. Tato funkce se nazývá *účelová funkce*, nebo též *cenová funkce* (v případě GA *fitness funkce*). Tato funkce je zobrazením z množiny všech řešení X do množiny reálných čísel.

Definice 2.9.1. Nechť X je množina všech řešení, pak
$f(x) : X \longrightarrow \mathbf{R}$ je účelová funkce.
■

Základní myšlenka horolezeckého algoritmu je v tom, že k určitému řešení sestrojíme daný počet nových řešení tak, že na zvolené řešení použijeme konečný počet transformací. V případě bitového vektoru ve zvoleném řešení náhodně změníme některé bity. Říkáme, že zvolené řešení je středem oblasti z něho generovaných řešení. Z takové oblasti vybereme nejlepší řešení, to znamená řešení s minimální funkční hodnotou a toto řešení použijeme v následujícím interakčním kroku jako střed nové oblasti. Tento algoritmus opakujeme předem zvoleným počtem iterací. Uchováváme si nejlepší řešení, které se v průběhu výpočtu našlo.

Nevýhodou tohoto přístupu je, že během celého výpočtu se akceptují buď stejně dobrá nebo lepší řešení, než bylo řešení výchozí. Díky tomu se může stát, že se metoda dostane k nevýraznému lokálnímu minimu blízko od počátečního náhodně vygenerovaného řešení a již nikdy nedosáhne optimálního řešení. Tento nedostatek je možné odstranit tak, že se algoritmus opakovaně spouští a tím se náhodně volí počáteční řešení úlohy. Za výsledné řešení se

přijme nejlepší nalezený výsledek. Stochastičnost této metody spočívá pouze v náhodném výběru počátečního řešení, neboť následně použitý optimalizační algoritmus postupuje systematicky bez jakékoliv náhodnosti.

Uváznutí v lokálním minimu úspěšně řeší metoda simulovaného žíhání, která využívá stochastické operátory a na rozdíl od horolezeckého algoritmu přijímá s jistou pravděpodobností (danou Metropolisovým kritériem) i řešení horší, než bylo výchozí řešení x. Tuto metodu již lze právem nazývat stochastickou, neboť vzorkuje po celém prostoru řešení. Dalším rozdílem je, že se negeneruje okolí a nevybírá se nejlepší řešení, ale určitým operátorem se stochasticky transformuje výchozí řešení x na nové x'. Tím se zajistí rozptýlení po celém prostoru a nikoliv pouze v jeho malé části.

Metoda simulovaného žíhání patří mezi stochastické optimalizační algoritmy, jež mají svůj základ ve fyzice. Tento algoritmus je založený na analogii mezi žíháním tuhých těles a optimalizačním problémem. Žíháním se ve fyzice označuje takový proces, ve kterém je těleso umístěné do pece, zahřáto na vysokou teplotu a postupným pomalým snižováním teploty se odstraňují vnitřní defekty tělesa. Na myšlenku, že by se jevu probíhajícího při žíhání tuhého tělesa mohlo využít k hledání globálního minima, přišli začátkem osmdesátých let Kirkpatrick, Gelatt a Vecchi z výzkumného centra IBM (Watson Research Center of the IBM, USA).

Nejdříve bylo zapotřebí nahradit fyzikální realizaci žíhání numerickou simulací. Inspirace byla nalezena v algoritmu z padesátých let, kdy použil Metropolis a jeho spolupracovníci numerickou simulační metodu Monte Carlo pro výpočet termodynamických konstant plynu. Simulovali vývoj fyzikálního systému směrem k tepelné rovnováze pro určitou konstantní teplotu T. Metodu Monte Carlo, která simuluje evoluci, lze popsat následujícím způsobem: nechť je dán aktuální stav systému, potom se malá náhodná porucha generuje tak, že jsou částice mírně posunuté. Porucha musí být symetrická, tj. pravděpodobnost toho, že malou poruchou se stav A změní na stav B, musí být stejná jako při změně stavu B na stav A. Pokud je rozdíl energie mezi porušeným stavem a aktuálním stavem negativní ($E' < E$), potom proces pokračuje s novým porušeným stavem. V opačném případě pravděpodobnost přijetí porušeného stavu určuje exponenciála.

Definice 2.9.2. Nechť f je účelová funkce, x je aktuální stav systému a x' je nový stav pak

$$P(x \to x') = \begin{cases} 1 & \text{pro } f(x') \leq f(x) \\ e^{\frac{f(x')-f(x)}{T}} & \text{pro } f(x') > f(x) \end{cases}$$ je pravděpodobnost přijetí nového stavu.

■

Parametr T je teplota systému. V případě, že nový stav x' má menší nebo stejnou funkční hodnotu jako původní stav x, potom se stav x změní na x'. V opačném případě je nový stav akceptován s pravděpodobností v rozmezí $0 < P(x \to x') \leq 1$.

Toto pravidlo přijetí porušeného stavu se nazývá Metropolisovo kritérium. Podle tohoto kriteria aplikováním velkého počtu poruch a jejich přijetím do dalšího procesu s pravděpodobností P dostáváme systém v tepelné rovnováze. Distribuce pravděpodobnosti rozložení stavů se asymptoticky blíží k Boltzmannové distribuci. Tento tvar metody Monte Carlo se ve statistické fyzice nazývá Metropolisův algoritmus.

Metropolisův algoritmus je možné použít pro počítačovou simulaci žíhání. V tomto přístupu se simulované žíhání chápe jako posloupnost Metropolisových algoritmů realizovaných pro posloupnost vhodně se snižujících teplot, přičemž výstupní stav z poslední aplikace Metropolisova algoritmu slouží jako vstupní stav pro následující běh Metropolisova algoritmu. Teplota se inicializuje na maximální a algoritmus se aplikuje tak dlouho, až

dokud se nedosáhne tepelné rovnováhy. Potom se teplota sníží (nejčastěji vynásobením teploty konstantou menší než 1) a algoritmus se znovu aplikuje. Celý proces končí při dosažení minimální hodnoty teploty. Tento postup je demonstrován algoritmem 2.9.1, kde na řádku 6 je ve vzorci zaměněno x a x'. Tato úprava zajistí hledání maxima účelové funkce, místo minima.

Algoritmus 2.9.1. Simulované žíhání (upravené pro hledání maxima)

```
1  T=Tmax;
2  while (T>Tmin) {
3    // METROPOLIS begin
4    for(k=0; k<kmax; k++) {
5      x‘=mutation(x);
6      Pr=exp(-(f(x)-f(x‘))/T);
7      if(Pr>1) Pr=1;
8      if (mtrand1()<Pr) {
9        x=x‘;
10     }
11   }
12   // METROPOLIS end
13   T=ALPHA*T;
14 }
```

Simulované žíhání a genetický algoritmus jsou v této práci použité pro hledání globálního maxima fitness funkce, jež je součástí metodiky pro rozdělení obvodu na TB.

Kapitola 3

Současný stav řešeného problému

V této kapitole budou zmíněny některé existující metodiky a techniky návrhu obvodů pro snadnou testovatelnost. Výčet není určitě úplný, jsou zde uvedeny takové příklady technik, s nimiž zde prezentovaná metodika souvisí (např. metody pro úplný a částečný scan).

3.1 Techniky návrhu obvodů pro snadnou testovatelnost

Cílem algoritmu pro generování testu je vygenerovat pro daný číslicový obvod test vyhovující konkrétním kritériím, např. co nejkratší sekvenci testovacích vzorků s co největším pokrytím poruch v daném obvodu. Kromě pokrytí poruch, které je významným ukazatelem kvality testu, používají se i ukazatele nákladů spojených s testem - mezi ně patří zejména náklady na generování testovacích vzorků, náklady na simulaci poruch a generování informace o lokalizaci poruch, náklady na vybavení nutné pro test a náklady na samotný proces testování, jehož ukazatelem je např. doba aplikace testu. Protože náklady spojené s testováním prvků mohou být vysoké tak, že mohou přesáhnout i nejvyšší přípustné náklady na návrh a výrobu obvodu, je důležité, abychom byli schopni uchovat je v rozumných mezích. Tímto problémem se zabývají techniky návrhu pro snadnou testovatelnost.

Techniky snadno testovatelného návrhu mají kromě pozitivního dopadu (zvýšení řiditelnosti a pozorovatelnosti) také negativní důsledky. Mezi ně patří zvětšení plochy na čipu, zvýšení počtu vývodů obvodu a zhoršení dynamických vlastností obvodu. Při každém návrhu je proto nutné volit jistý kompromis mezi těmito aspekty. Jde vlastně o kompromis mezi požadavky návrháře a diagnostika.

Zvětšení potřebné plochy čipu má tyto negativní dopady:

- zvýšený příkon a tím pádem větší zahřívání obvodu a zvýšení požadavků na odvod tepla,
- snížení výtěžnosti,
- snížení spolehlivosti.

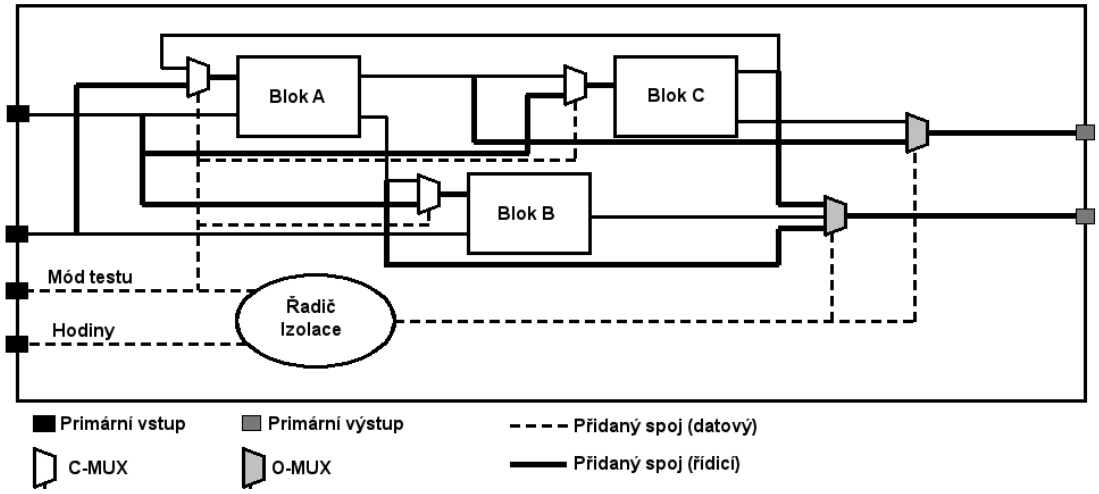
Snížení výtěžnosti souvisí bezprostředně s nárůstem obvodu jako výsledek doplnění obvodu o diagnostické prvky, je proto nutné právě zde usilovat o účelný kompromis. Použitím principů snadno testovatelného návrhu můžeme dosáhnout:

- snížení doby potřebné pro generování testu,
- zvýšení kvality testu (zvýšením pokrytí poruch),

- zkrácení testu,
- zkrácení doby aplikace testu.

3.1.1 Metoda snadno testovatelného návrhu využívající rozdělení obvodu a využití multiplexorů

Tato metoda je popsána v [29]. Obvod je rozdělen na bloky, kde každý blok je izolovaný pomocí primárních vstupů a výstupů a přidaných multiplexorů. Tak je možné generovat test pro každý blok zvlášť. Testovací vektory jsou pak přiváděny přes multiplexory, jež jsou řízeny vnitřním řadičem testu.



Obrázek 3.1: Izolace bloků použitím multiplexorů

Na obrázku 3.1 vidíme obvod rozdělený na tři bloky A, B, C. Po rozdělení obvodu jsou přidány dvouvstupové multiplexory (C-MUX) mezi primární vstupy a vstupy bloků, což umožní nezávislé řízení bloku. Další multiplexory (O-MUX) jsou přidány mezi primární výstupy a výstupy bloků, tyto multiplexory dovolují pozorovat výstupy bloků. Vstup pro přepnutí do módu testu pak uvede konečný automat do výchozího stavu a ten pak s hodinovým taktem nastavuje multiplexory tak, že postupně připojuje každý blok zvlášť na primární vstup a výstup.

Pro rozdělení obvodu je využito metody PINS[1]. Obvod reprezentovaný klasickými prvky obvodu na úrovni RT je převeden na orientovaný graf. Uzly reprezentují prvky obvodu, hrany reprezentují datové spoje mezi prvky obvodu. Metodika pracuje s pojmem „seskupení“, což jsou takové části analyzovaného obvodu, které jsou samostatně testovatelné. Metoda PINS nejprve vybere základní uzel[2] seskupení. Preferovány jsou uzly reprezentující primární vstupy/výstupy, prvky s mnoha vstupy/výstupy a prvky modelující konstantu. Dále pak seskupení roste přibíráním okolních připojených uzlů s jednou podmínkou, že seskupení má lineární strukturu (příklad lineární struktury je na obrázku 3.2) a počet vstupů a výstupů je menší než kolik má celý obvod. Seskupení přestane růst, pokud už nelze přidat další uzel. Pak se znovu vybere základní uzel a tvoří se nové seskupení. Proces končí, až jsou všechny uzly zařazeny do seskupení, které reprezentují jednotlivé bloky.

[1] DFT using Partitioning and Isolation with Non Scan design
[2] uzel v této metodě představuje prvky obvodu a primární vstupy a výstupy

Existuje také metoda PIPS[3]. Metoda PINS rozděluje obvod na příliš mnoho bloků, pokud obvod obsahuje mnoho vlastních smyček (obrázek 3.2), jež jsou netestovatelné, protože obvodový prvek i registr musí být každý v jiném bloku. Metoda PIPS tedy kombinuje metodu PINS s metodou částečný scan. Registry ve vlastní smyčce a registry konečného stavového řízení (FSM) jsou zařazeny do scan řetězce před rozdělením. Zbytek rozdělení je prováděn stejně jako u PINS. Registry zařazené do scanu pak nejsou v metodě PINS považovány za registry. Struktura řetězce scan pak v metodě PIPS má tu vlastnost, že scan řetězec může být rozdělen bez použití dalších vstupů a výstupů pro každý blok, což vidíme na obrázku 3.3. Počet testovacích vektorů je pak redukován díky rozdělení řetězců scan.

3.2 Metoda úplný a částečný scan

Jak již bylo zmíněno v kapitole 2.7.2, vytvoření testovací posloupnosti pro sekvenční obvody je obecně složitější problém než pro kombinační obvody. Existence vnitřního stavu určujícího hodnotu výstupu v následujícím taktu komplikuje detekci i lokalizaci poruchy, protože její projev na výstupu se může zpozdit o několik taktů. Proces vytvoření testu je také časově náročnější, což vyžaduje delší strojové časy a tím pádem představuje zvýšení ceny navrhovaných obvodů.

Jako jedno z řešení testování sekvenčních obvodů se nabízí použití metod typu úplný scan obvodu. V režimu testu jsou všechny klopné obvody propojeny do posuvného registru, takže jsou snadno řiditelné/pozorovatelné. Při aplikaci testu pak můžeme na takový obvod pohlížet jako na obvod sestávající z posuvného registru a kombinační logiky. Pro vygenerování testu takového obvodu je možné využít GTKO pro kombinační obvody. Nevýhodou této metody je ovšem:

- prodloužení doby potřebné pro aplikaci testu,
- zvětšení plochy čipu.

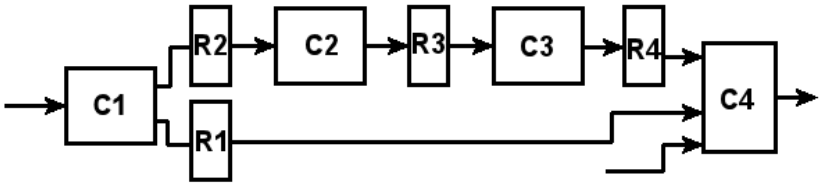
Zvýšení doby potřebné pro aplikaci testu je způsobeno tím, že veškeré diagnostické údaje jsou v testovaném obvodě přenášeny sériově. Toto je možné demonstrovat na příkladě obvodu, který obsahuje 64 klopných obvodů. Znamená to, že pro vložení jednoho testovacího vektoru do posuvného registru musí být nejprve generováno 64 synchronizačních impulsů a pak jeden puls, jímž je tento testovací vektor aplikován. Jestliže test tvoří 200 testovacích vektorů, pak pro jeho aplikaci musí být generováno $200 * 65 = 13000$ pulsů, pro přenos odezvy na poslední vektor na primární výstup pak dalších 64 impulsů.

Počet **s**ynchronizačních **p**ulsů, které je potřeba generovat, pokud je použita metoda úplný scan, je možné určit podle vztahu:
$sp_{úplný\ scan} = v * (a + 1) + a,$
kde v je počet testovacích vektorů, a je počet klopných obvodů zařazených do posuvného registru.

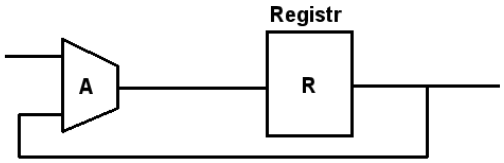
Dobu aplikace testu je možné snížit změnou zapojení posuvného registru. Spočívá v rozdělení registru na více segmentů, které jsou pak samostatně ovládány [37]. Toto řešení představuje nárůst logiky řízení aplikace testu. Naopak zvětšení plochy na čipu souvisí s konstrukcí klopného obvodu, který je u těchto metod používán. Jeho struktura je obecně složitější, než je struktura běžného klopného obvodu, např. typu D.

Řešením zmíněných negativních dopadů uplatnění metody úplný scan je využití metod označovaných jako částečný scan, kdy nejsou do posuvného registru zařazeny všechny klopné

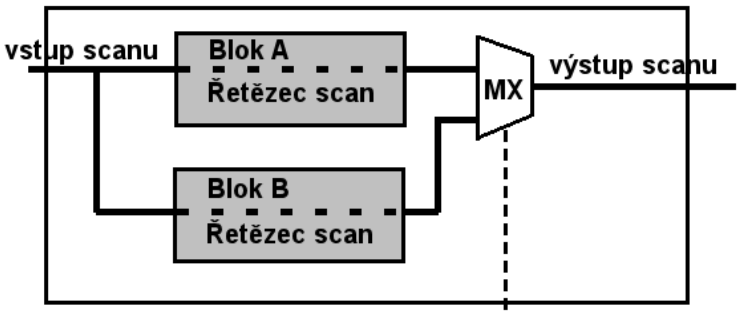
[3]DFT using Partitioning and Isolation with Partial Scan design



Obrázek 3.2: Příklad lineární struktury obvodu



Obrázek 3.3: Příklad vlastní smyčky



Obrázek 3.4: Struktura řetězce scan u metody PIPS

obvody, obvod má však požadované vlastnosti z hlediska jeho testovatelnosti. Testovatelnost je přitom vyjádřena např. pomocí pokrytí poruch, jehož se použitím konkrétní metody dosáhne nebo řiditelností/pozorovatelností prvků v obvodě.

Výběr klopných obvodů je možné realizovat v obvodě na úrovni hradel nebo na úrovni RT. Pro výběr prvků do registru jsou používána různá kriteria. Podle tohoto kriteria je možné je dělit následujícím způsobem:

- metody založené na analýze testovatelnosti (řiditelnosti/pozorovatelnosti),
- metody využívající generátor testovacích vektorů,
- metody využívající výsledky analýzy struktury obvodu.

První skupina metod je založena na analytickém přístupu k řešenému problému. Je stanoven způsob (funkce), jímž se vypočte globální testovatelnost analyzovaného obvodu. Hodnota této funkce je ovlivněna řiditelností/pozorovatelností jednotlivých prvků obvodu. V obvodu jsou pak prováděny změny (např. zařazení klopného obvodu do registru scan) a vyhodnocuje se vliv těchto změn na globální testovatelnost obvodu. Výsledkem uplatnění takové metodiky je doporučení, které klopné obvody mají být zařazeny do posuvného registru. [11, 44]

Druhou skupinou jsou metody využívající generátor testovacích vektorů, které pracují se strukturou obvodu na úrovni hradel. [4, 5] Jeden z možných přístupů spočívá ve výpočtu pokrytí poruch pro případ, kdy jsou do posuvného registru zařazeny všechny klopné obvody. Toto pokrytí poruch je pak považováno za maximální dosažitelné pokrytí. Pak se počet klopných obvodů sníží (podle konkrétního kritéria) a znovu se vypočte pokrytí poruch. Tento proces se pak opakuje tak dlouho, dokud je pokrytí poruch vyšší nebo shodné s požadovaným. Nevýhodou tohoto přístupu je doba potřebná pro tento proces, způsobená tím, že se opakovaně pomocí GTSO počítá pokrytí poruch. V závislosti na složitosti obvodu a na použitém sekvenčním generátoru to může být proces značně časově náročný. Do této skupiny patří například jedna z metod programu FastScan firmy Mentor Graphics®.

Metody založené na analýze struktury jsou dvojí:

- metody založené na identifikaci zpětných vazeb,
- metody založené na identifikaci paralelních i-cest[4].

Je uznávanou skutečností, že existence zpětnovazebních smyček výrazně ovlivňuje výsledek použití GTSO a potřebný strojový čas pro vygenerování testu [12]. Naopak u metod založených na identifikaci i-cest je cílem nalezení alternativy k metodám strukturovaného návrhu. Je založena na analýze struktury navrhovaného obvodu a hledání cest, po nichž by bylo možné přenášet diagnostická data, tzn. testovací vektory z primárních vstupů na vstupy testovaných prvků a odezvy jejich výstupu na primární výstupy. Výsledky takové analýzy jsou pak předmětem dalších úvah o možnosti uplatnění metody částečný scan.

Je také možné, aby se při analýze testovatelnosti zohledňovala libovolná kombinace těchto hledisek.

Optimalizovat je také možné vkládání testovacích vektorů přes scan registry. Tímto se zabývá [62]. Technika se snaží nahradit malé množství (max 2%) skenovatelných registů rozšířenými skenovatelnými registry, jež jsou schopny uchovávat dva bity. Tyto registry jsou

[4] i-cesta viz. kapitola 3.2.2

řízeny signály generovanými uvnitř obvodu. Tyto rozšířené registry pak zvyšují podíl bitů v testovacím vektoru, které nejsou významové[5]. Tyto bity pak zvyšují účinnost kompresních algoritmů a snižují dobu testu. V článku je navržena metodika pro výběr těchto registrů a využívá ohodnocení řiditelnosti a snížení množství testovacích vektorů v důsledku záměny.

3.2.1 Metoda částečný scan Indického institutu technologie

Metoda demonstrovaná v [7] kombinuje 3 různé metody do jedné a využívá genetický algoritmus při optimalizaci.

První část metody se zabývá analýzou testovatelnosti a je založená na metodě SCOAP [19]. Tato metoda funguje tak, že jsou všechny registry seřazeny podle míry testovatelnosti. Ty s největší mírou testovatelnosti jsou vybrány jako kandidáti do řetězce scan. Tento způsob však neuvažuje s přímo sousedícími registry. Vliv sousedících registrů je v této metodě zohledněn takto: Ze sekvenčního obvodu je vytvořen s-graf [12]. Mějme dva uzly n_1 a n_2 a nechť $výstupní\_stupeň(n_1) > výstupní\_stupeň(n_2)$, tzn., že n_1 je připojen k více registrům než n_2. Pak pokud je testovatelnost n_2 větší než n_1, může být n_2 vybrán do řetězce scan na úkor n_1 i přesto, že n_1 může ovlivnit větší část obvodu. Podobně je tomu i u vstupních stupňů. I když je vstupní stupeň n_1 větší než n_2, n_1 může být také vhodným kandidátem na zařazení do řetězce scan.

Druhá část metody zkoumá strukturu obvodu. Je zkoumán počet zpětných vazeb procházejících daným registrem. Každému registru je pak přiřazeno číslo (CN) reprezentující počet zpětných vazeb. Čím větší je toto číslo, tím větší je pravděpodobnost, že registr bude vybrán do řetězce scan.

Poslední část zohledňuje spotřebu energie. Jako míra spotřeby energie se zde využívá počet překlopení na signálových vodičích obvodu při aplikaci testu.
$power = 0.5 * V_{dd}^2 * f_{clock} * \Sigma C_f D(f)$
C_f je kapacita brány/uzlu f, $D(f)$ je počet překlopení. V_{dd} a f_{clock} jsou konstantní. Je třeba optimalizovat $\Sigma C_f D(f)$, což je obecně označováno jako počet překlopení (NTC - Node Transition Count).

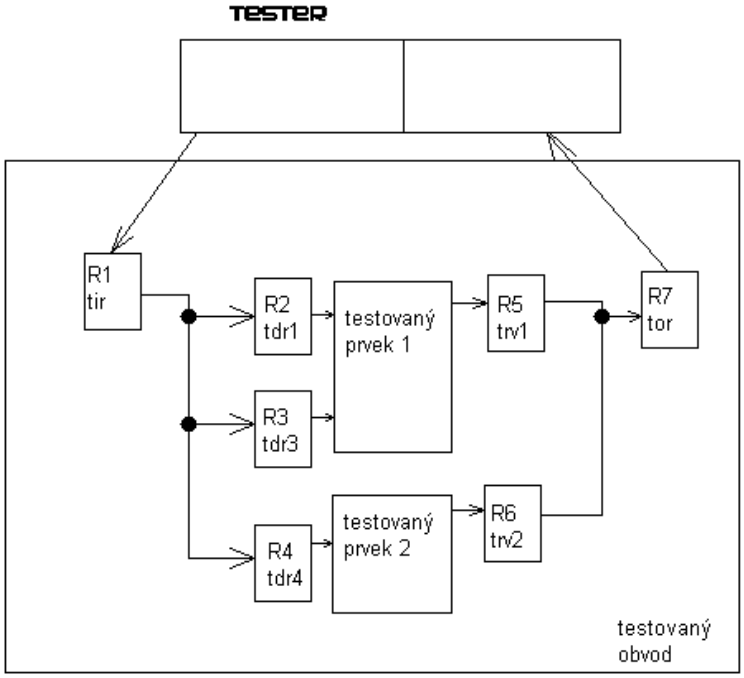
Pro optimalizaci je použit genetický algoritmus. Jednotlivé bity chromozomu určují, zda jde o klasický registr nebo o registr scan. Fitness funkce je vypočtena podle tří výše uvedených hledisek. Dále pak je použito jednobodové křížení, mutace s pravděpodobností 0.05. Počet iterací algoritmu je rovný $N * N$, kde N je počet registrů, nejméně však 500.

3.2.2 Metodika pro výběr registrů do řetězce scan založená na analýze i-cest

Tato práce využívá pojmů: „režim identity dat“ (režim i) a „identická přenosová cesta“ (i-cesta). Tyto pojmy vyjadřují schopnost prvku přenést data z jeho vstupu na výstup příp. z výstupu jednoho prvku na vstup jiného prvku tak, že tato data nejsou modifikována, viz definice 5.2.1. Pojem i-cesta byl zaveden v [1], vlastní metodika postavená na tomto pojmu byla vyvinuta na Ústavu počítačových systémů, Fakulty informačních technologií, VUT v Brně [38] a dále rozvíjena v [58]. Její základní myšlenkou je v co největší míře využít možností, které nabízí navržený obvod, pro aplikaci testu. Snahou je tedy nalézt co nejvíce i-cest, po kterých bude možné řídit vstupní registry testu (vrt[38], tir[58]) z primárních vstupů a co nejvíce i-cest, které by zaručily pozorovatelnost výstupních registrů testu ($vyrt$[38], tor[58]) na primárních výstupech.

[5] tzv. don't care bity

V praxi není možné vždy nalézt v obvodu i-cestu, po níž lze z primárních vstupů přenést testovací vektory na vstup každého registru *tir* a i-cestu z každého výstupu registru *tor* na primární výstupy obvodu. Pro ty případy, kdy to možné není, se začne uvažovat o modifikaci registru tak, aby bylo možné jej z vnějšku řídit, či pozorovat, např. úpravě registru na skenovatelný registr. Tato koncepce navíc umožňuje sdílení vstupního či výstupního registru testu pro více prvků. Pro zajištění plné testovatelnosti obvodu je třeba zajistit řiditelnost všech vstupních bran každého prvku a pozorovatelnost všech výstupních bran každého prvku. Z toho plyne, že je třeba nalézt alespoň jeden registr pro každou vstupní bránu, z něhož je možné přivést testovací vektory na tuto bránu, tzv. vysílač testovacích vektorů (*vtv*[38], *tdr*[58]). Obdobně je třeba nalézt alespoň jeden registr pro každou výstupní bránu, z nějž je možné pozorovat výstupy z této brány tzv. přijímač odezev na testovací vektory (*potv*[38], *trv*[58]).



Obrázek 3.5: Vysvětlení pojmu registru *tir*/*tor*

Dále se budeme držet značení zavedeného v [58]. Na obrázku 3.5 vidíme příklad obvodu, v němž je zobrazen registr *tir* ($R1$), který je spojený nějakou i-cestou s více registry *tdr* ($R2, R3, R4$) a podobně více registrů *trv* ($R5, R6$) je propojeno do jednoho registru *tor* ($R7$). Sdílení jediného registru *tir* více registry *tdr* má výhodu v tom, že stačí zajistit řiditelnost pouze registru *tir*. Obdobně je tomu na výstupu, kdy stačí zajistit pozorovatelnost registru *tor*.

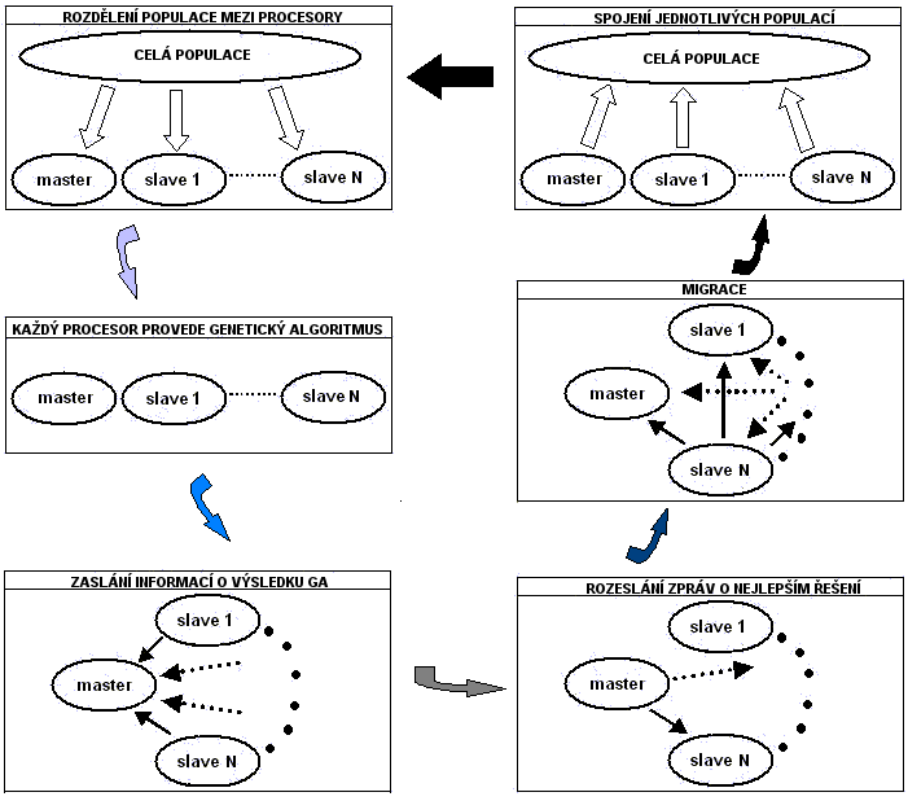
Pro každý registr *tdr*, je třeba nalézt vhodný registr *tir*. Stejně tak pro každý registr *trv* se hledá vhodný registr *tor*. Poté je třeba určit množinu takových registrů, které mohou hrát roli vstupního registru testu, ale nejsou řiditelné z žádného primárního vstupu (pomocí i-cest), ty budou kandidáty pro zařazení do registru scan. Dále je třeba určit systém množin $\Im$, jež obsahuje všechny třídy TIR_{tdr} pro všechny registry *tdr*. Každá jednotlivá množina TIR_{tdr} je množina všech vstupních registrů *tir*, použitelných jako vstupní registr pro *tdr*. Obdobně systém množin $\aleph$ obsahuje všechny třídy TOR_{trv}, kde každá jednotlivá množina

TOR_{trv} je množina všech výstupních registrů tor, použitelných jako výstupní registr pro trv. Ze systému množin $\Im$ se určí systém $\Im_{NC}$, jež je systémem množin registrů tir pro takové registry tdr, které nemají zajištěnou řiditelnost žádným řiditelným[6] registrem tir. Pokud je množina $\Im_{NC}$ neprázdná, bude třeba některé registry upravit tak, aby byly řiditelné (zařadit do řetězce scan). Je třeba zajistit řiditelnost alespoň jednoho registru z každé třídy TIR_{tdr}. Protože však třídy z $\Im_{NC}$ nemusí být disjunktní, lze vybrat registry právě z průniku dvou tříd a snížit tak počet registrů k úpravě (zařazení do scanu). Obdobně postupujeme u výstupu, tzn. určíme systém množin $\aleph_{NO}$, jež je systémem množin registrů tor, pro takové registry trv, které nemají zajištěnou pozorovatelnost žádným pozorovatelným[7] registrem tor. Pokud je systém množin $\aleph_{NO}$ neprázdný, bude třeba některé registry upravit tak, aby byly pozorovatelné (zařadit do řetězce scan). Je třeba zajistit pozorovatelnost alespoň jednoho registru z každé třídy TOR_{trv}, protože však třídy z $\aleph_{NO}$ nemusí být disjunktní, lze vybrat registry k úpravě právě z průniku těchto tříd a snížit tak počet registrů k úpravě (k zařazení do scanu).

3.3 Další metodiky pro rozdělení obvodu

V [8] je navržen paralelní evoluční algoritmus pro rozdělení obvodu na části. Toto rozdělení se využívá pro snadnější zpracování obvodu v návrhových systémech na paralelních počítačích. Metodika prezentovaná v článku si klade za cíl rozdělit obvod na nepřekrývající se bloky obsahující minimálně jedno hradlo. Algoritmus se snaží minimalizovat cenu komunikace mezi procesory tak, že bude přerušen co nejmenší počet spojů. A dále se snaží rovnoměrně rozdělit práci mezi procesory tím, že jednotlivé části budou přibližně stejně velké. Pro výpočet využívá orientovaného acyklického grafu. Řešení je reprezentováno řetězcem čísel, které přiřazují každému hradlu označení bloku, do kterého patří. Křížení pak spočívá v přesunu hradel mezi sousedními bloky. Mutace je založena na přesunu hraničních hradel do přilehlých bloků. Výchozí rozdělení je náhodné. Paralelizace algoritmu je pak provedena rozdělením populace na jednotlivé procesory. Každý procesor provede sekvenční genetický algoritmus a informace o fitness je shromažďována v hlavním procesoru. Hlavní procesor provádí distribuci nejlepších mezivýsledků do jednotlivých procesorů. Mezi jednotlivými procesory probíhá řízená migrace řešení, jejímiž parametry jsou topologie, počet migrujících individuí a interval mezi migracemi (obr. 3.6). Nakonec se jednotlivé populace spojí a proces se opakuje.

[6] řiditelným z primárních vstupů přímo nebo přes i-cesty
[7] pozorovatelným na primárních výstupech přímo nebo přes i-cesty



Obrázek 3.6: Průběh paralelního algoritmu

Kapitola 4

Motivace a cíle práce

Na Ústavu počítačových systémů, Fakulty informačních technologií, VUT v Brně byl vyvinut formální model obvodu na úrovni RT s cílem ověřit možnost využití matematického aparátu pro diagnostické účely. Existence tohoto modelu byla impulzem pro další využití a rozšiřování modelu s cílem ukázat, že je možné využít tento formální aparát k definování a implementaci metodik analýzy testovatelnosti.

V profesionálních nástrojích pro generování testovacích sekvencí číslicového obvodu jsou využívány tyto přístupy:

1. Generátor testu pro sekvenční obvody předpokládá, že test bude aplikován přes jeho primární vstupy/výstupy. Znamená to, že na jeho primární vstupy jsou vkládány testovací vektory, odezvy na ně jsou vyhodnocovány na jeho primárních výstupech. Proces vygenerování takového testu je časově náročný a objem testovacích vektorů je značný.

2. Generátor testu předpokládá, že číslicový obvod je navržen s využitím metody úplný scan. Celý obvod je tak rozdělen na řadu dílčích kombinačních obvodů, pro než vygenerování testu je proces jednodušší než v předcházejícím případě.

3. Generátor testu předpokládá, že číslicový obvod je navržen s využitím metody částečný scan. Celý obvod je opět rozdělen na řadu dílčích kombinačních obvodů, pro něž vygenerování testu je proces jednodušší než v případě 1. Navíc počet registrů využitých pro aplikaci testu je nižší než v případě 2.

Všechny výše uvedené principy jsou využity např. v nástroji FlexTest fy Mentor Graphics. V návaznosti na výše popsané principy využívané v profesionálních nástrojích pro generování testu a jeho aplikaci byly cíle práce stanoveny takto:

1. Definovat cíle metodiky pro rozdělení obvodu na testovatelné bloky (TB). Tyto principy definovat tak, aby výsledkem implementace bylo rozdělení analyzovaného obvodu na menší celky a možnost aplikovat test kombinováním výše uvedených přístupů. Kriteriem rozdělení budou konkrétní strukturální vlastnosti, nikoliv vlastnosti funkční.

2. Definovat strukturální vlastnosti, které budou identifikovány při analýze číslicové komponenty s cílem rozdělit ji na jistý počet TB.

3. Definovat strukturální vlastnosti, které musí splňovat TB tak, aby bylo možné aplikovat test přes rozhraní TB.

4. Vytvořit formální model analyzovaného obvodu tak, aby umožňoval realizaci bodů 2. a 3.

5. Implementovat principy definované v bodech 2. - 4. Demonstrovat možnost využití formálního matematického modelu pro diagnostické účely.

6. Navrženou a implementovanou metodiku vyhodnotit. Vytvořit a ověřit experimenty, výsledky využít pro srovnání metodiky s existujícím metodikami používanými v diagnostických systémech. (počet registrů vybraných do částečného scanu, počet testovacích vektorů, pokrytí poruch, objem přídavné elektroniky a doba generování testu).

7. Získané výsledky shrnout.

Kapitola 5

Formální model obvodu

Metodika prezentovaná v této práci pracuje s obvody na úrovni RT. Na této úrovni existují různé formální modely, jeden z nich byl vyvinut na Ústavu počítačových systémů, Fakulty informačních technologií, VUT v Brně [38, 58]. Prezentovaná metodika předpokládá, že pro přenos diagnostických dat je využita strategie multiplexovaných datových cest (obvod neobsahuje obousměrné sběrnice). Výše zmíněný formální model bylo nutno pro účely zde prezentované metodiky dále rozšířit.

Zvolený formální model obvodu chápeme jako pětici množin, které reprezentují statickou strukturu obvodu (viz. definice 5.1.1). Tato základní definice je pak doplněna o další definice, které přiřazují prvkům modely chování a další vztahy. Cílem činností realizovaných při tvorbě modelu bylo co nejlépe vystihnout strukturu skutečných obvodů spolu s vlastnostmi důležitými pro vytvořenou metodiku.

5.1 Existující model

Model obvodu zmíněný v úvodu kapitoly popisuje číslicový obvod na úrovni RT, který může vzniknout například syntézou z vyšší úrovně popisu nebo z popisu chování. Model obvodu je zaměřen na popis obvodu s využitím strategie multiplexovaných datových cest. Model celého obvodu je tvořen uspořádanou pěticí množin a dále umožňuje hierarchický popis. Je zavedena základní klasifikace obvodových prvků s ohledem na jejich chování a roli v obvodě a také s ohledem na skutečnosti, které plynou z použití strategie syntézy s využitím multiplexovaných datových cest. Jedna z množin základní pětice modelující obvod je vlastně relace. Je to relace spojů mezi branami obvodových prvků.

Nyní budou uvedeny definice, které jsou dále v práci použity a jsou převzaty z [58].

Definice 5.1.1. Nechť E je množina obvodových prvků,
P je množina jejich bran (vstupů a výstupů),
C je množina spojů mezi branami prvků obvodu,
PI je množina primárních vstupů obvodu a
PO je množina primárních výstupů obvodu,
pak $UUA = (E, P, C, PI, PO)$ je uspořádaná pětice reflektující model struktury číslicového obvodu na úrovni RT.
■

Model obvodu, popsaný v definici 5.1.1, vychází z tradičního pohledu na strukturu číslicového obvodu na úrovni RT. Celý testovaný obvod UUA lze také chápat jako část

nějakého většího celku, o které je známo její chování vyjádřené v tomto případě chováním jeho prvků z množiny E propojených prostřednictvím svých bran P, PI, PO spoji z množiny C. Lze na něj tedy nahlížet jako na prvek se strukturou (E, P, C) a branami (PI, PO). Stejně tak každý prvek E obvodu UUA může být pro další analýzu dekomponován na svou strukturu a na své (při pohledu na strukturu prvku) primární brány. Podobně hierarchický model je též v [49].

Množinu obvodových prvků E lze rozdělit podle následující definice:

Definice 5.1.2. Nechť R je konečná množina registrů,
FU je konečná množina funkčních jednotek,
MX je konečná množina multiplexorů,
pak $E = (R \cup MX \cup FU)$ je množina obvodových prvků.
■

Definice 5.1.2 pokrývá všechny druhy prvků, které se mohou vyskytnout ve struktuře obvodu na úrovni RT. Jsou to registry, funkční jednotky a multiplexory.

Toto rozdělení je důležité pro účely analýzy testovatelnosti. Množina registrů R představuje všechny prvky s paměťovým charakterem, jež jsou nositelem sekvenčního chování obvodu. Prvky množiny MX řídí datový tok a prvky množiny FU mají vlastnosti kombinačních sítí.

Věta 5.1.1. Podmnožiny R, FU a MX tvoří rozklad množiny E podle relace ekvivalence „být prvkem stejného druhu“, jsou tedy navzájem disjunktní a jejich sjednocení dává právě množinu E.

Důkaz věty je jednoduchý. Relace „být prvkem stejného druhu“ je relací vylučující, aby jeden prvek byl více než jednoho druhu, proto je relací ekvivalence.

Pro účely strukturní analýzy je také důležité definovat množinu všech bran (portů), jež představují body, přes něž bude přenášena informace. Každý prvek má brány, přes něž je propojen s branami dalších prvků nebo s primárními vstupy či výstupy.

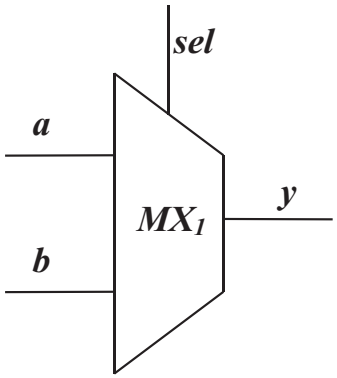
Definice 5.1.3. Nechť IN je konečná množina vstupních bran obvodových prvků,
OUT je konečná množina výstupních bran obvodových prvků,
CI je konečná množina řídicích a synchronizačních vstupů obvodových prvků,
pak $P = (IN \cup OUT \cup CI)$ je množina bran obvodových prvků.
■

Příklad 5.1.1. Užití předcházejících definic bude nyní demonstrováno na multiplexoru se dvěma vstupy a jedním výstupem (obrázek 5.1):

Pro multiplexor $\{MX_1\} \subseteq E$, se dvěma datovými vstupy a a b, jedním výstupem y a jedním bitem pro výběr adresy sel pak platí $\{a, b, y, sel\} \subseteq P$, $\{a, b\} \subseteq IN$, $\{y\} \subseteq OUT$ a $\{sel\} \subseteq CI$.

Definice 5.1.4. Nechť existuje funkce: $\psi : E \rightarrow 2^P$, která přiřazuje množinu bran obvodovému prvku (E je množina obvodových prvků a P je množina bran viz. definice 5.1.1), pak musí platit:

1. $\psi(e) = \{p | p \in P \wedge p$ je brána prvku $e\}$.
2. Funkce ψ je definována pro všechny prvky z množiny E.



Obrázek 5.1: Dvouvstupový multiplexor

3. Musí platit: $e_1 \neq e_2 \Leftrightarrow \psi(e_1) \cap \psi(e_2) = \emptyset$

■

Funkce ψ tvoří vazbu mezi množinou prvků a množinou bran. Tato funkce určuje, která brána z množiny bran přísluší konkrétnímu prvku. Z fyzikálního významu funkce ψ plyne nutnost podmínky 2 a podmínky 3. Podmínka 2 zajišťuje, že v obvodě se nevyskytne prvek, u kterého by nebylo možné identifikovat jeho brány. Podmínka 3 říká, že každý prvek z množiny P je funkcí ψ vázán k jedinému (obvodovému) prvku $e \in E$.

Příklad 5.1.2. Pro multiplexor z příkladu 5.1.1 a obrázku 5.1 bude platit: $\psi(MX_1) = \{a, b, y, sel\}$.

Prvky jsou mezi sebou propojeny pomocí spojů. Koncept propojení je definován takto:

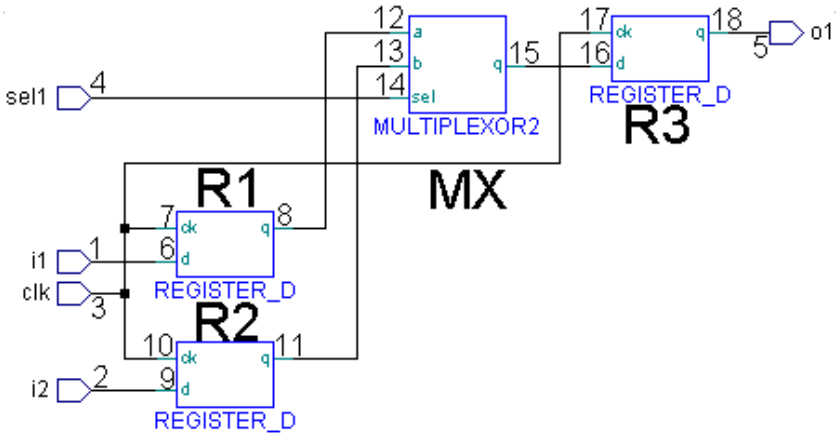
Definice 5.1.5. Nechť P je množina jejich bran (vstupů a výstupů),
PI je množina primárních vstupů obvodu,
PO je množina primárních výstupů obvodu,
pak $C \subset (PO \cup PI \cup P) \times (PO \cup PI \cup P)$ je množina spojů mezi branami prvků obvodu.
■

Jde vlastně o binární relaci v množině bran P a též mezi primárními vstupy a branami prvků a branami prvků a primárními výstupy. Obsahuje uspořádané dvojice bran, mezi kterými existuje v obvodě spoj (i-cesta délky 0 - viz kapitola 5.2)

C je relace, která je reflexivní, symetrická a tranzitivní, tudíž je relací ekvivalence. [58]. Prvky množiny C modelují galvanické spoje, které samy o sobě nemohou mít orientaci. To, co jim později dodává orientaci, je směr toku dat branami, které spojují.

Může nastat situace, kdy jsou mezi sebou navzájem propojeny více než dva body v obvodě. Pak je logické, že všechny tyto body jsou spolu (vždy po dvojicích) navzájem v relaci. Lze též říci, že v každém okamžiku mají všechny tyto body (a též spoj) stejnou hodnotu. Množina bodů obvodu, které jsou navzájem galvanicky propojeny spojem (jsou v relaci C), se bude dále nazývat uzel, viz definice 5.1.6.

Definice 5.1.6. Nechť $\forall p_1, p_2 \in V : (p_1, p_2) \in C$, pak množinu $V \subset (P \cup PI \cup PO)$ označujeme jako uzel.
■



Obrázek 5.2: Příklad obvodu

Následující dvě definice se týkají hodnot, kterých mohou brány nabývat a jsou základem pro definici 5.2.1:

Definice 5.1.7. Nechť číslo m udává šířku datových cest obvodu a $\{0,1\}^m$ je množina všech m-bitových binárních slov, pak $D = \{0, 1, \uparrow, \downarrow\} \cup \{0,1\}^m$ je množina hodnot, které se na bráně mohou vyskytovat. Symboly $\uparrow$ a $\downarrow$ mají význam nástupné resp. sestupné hrany.
■

Definice 5.1.8. Nechť existuje zobrazení $\nu : P \rightarrow D$, pak zápis $\nu(p) = w$ znamená, že na bráně $p \in P$ se vyskytuje hodnota $w \in D$.
■

Na obrázku 5.2 je příklad obvodu, jehož strukturu je možné reprezentovat podle výše uvedených definic takto:
$E = \{R1, R2, MX, R3\}$, $R = \{R1, R2, R3\}$, $MX = \{MX\}$, $FU = \{\}$, $PI = \{1, 2, 3, 4\}$, $PO = \{5\}$, $IN = \{6, 9, 12, 13, 16\}$, $OUT = \{8, 11, 15, 18\}$, $CI = \{7, 10, 14, 17\}$, $C = \{(1,1), (1,6), (2,2), (2,9), (3,3), (3,7), (3,10), (3,17), (4,4), (4,14), (5,5), (5,18), (6,1), (6,6), (7,3), (7,7), (7,10), (7,17), (8,8), (8,12), (9,2), (9,9), (10,3), (10,7), (10,10), (10,17), (11,11), (11,13), (12,8), (12,12), (13,11), (13,13), (14,4), (14,14), (15,15), (15,16), (16,15), (16,16), (17,3), (17,7), (17,10), (17,17), (18,5), (18,18)\}$

5.2 Přenos diagnostických informací obvodem s využitím transparentních módů prvků

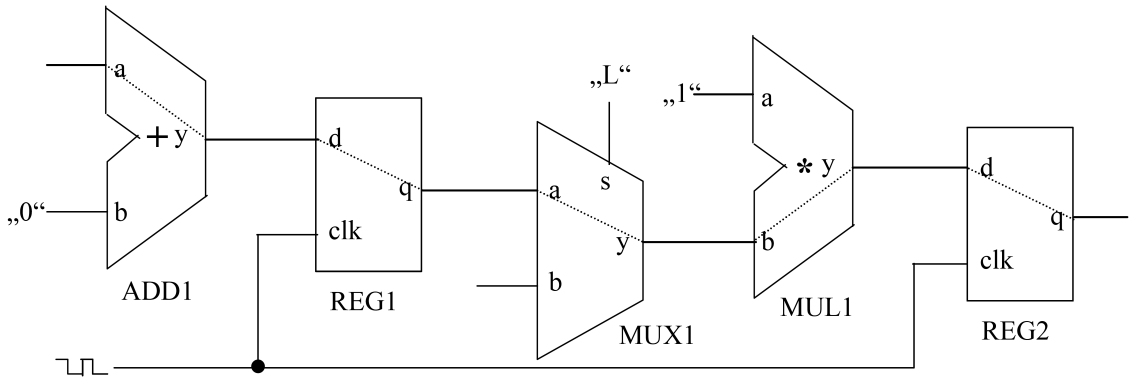
Aplikace testu na obvod spočívá v přivedení množiny testovacích vektorů na vstupy testované komponenty, získání odezev komponenty a jejich vyhodnocení. Právě řešení problému, jak přivést testovací vektory až do bodu, kde je připojen testovaný prvek (který je často někde hluboko ve struktuře obvodu, běžně zvnějšku nedostupný) a vyvedení odezvy do bodu, kam lze připojit analyzátor odezev, vede často k dodatečným zásahům do struktury obvodu a její větší složitosti. Ve struktuře obvodu jsou takové části, které je možné využít i v režimu testu obvodu k přenosu diagnostické informace. Podobnými vlastnostmi se zabýval tým z výzkumných laboratoří firmy Philips v Eindhovenu [48].

Těchto vlastností využijeme, proto zde uvádíme následující definice, jež popisují transparentní vlastnosti prvků a jejich použití. Následující definice jsou převzaty z [58] a jsou

představeny pouze ty, které budou dále využívány. Transparentností se myslí taková schopnost přenosu dat mezi dvěma body obvodu, kdy data vložená do jednoho bodu jsou přenesena beze změny do jiného bodu.

Definice 5.2.1. Nechť je možné za určitých podmínek (nastavením transparentních režimů prvků po cestě) z bodu p_1 do bodu p_2, kde $p_1 \in P, p_2 \in P$, v UUA přenést data beze změny, pak říkáme, že mezi body p_1 a p_2 existuje i-cesta.
Množina $I = \{(p_1, p_2) | p_1 \in P \wedge p_2 \in P \wedge \forall d \in \{0,1\}^m : \nu(p_1) = d \wedge \nu(p_2) = d\}$ je množinou všech možných i-cest v analyzovaném obvodě.
■

Pokud se tyto dva body nachází na jediném spoji (jsou součástí jednoho uzlu), není většinou s transparentností problém. Problém je ovšem s i-cestami, jejichž počáteční a koncový bod neleží ve stejném uzlu. Pak se na i-cestě jistě objeví i obvodové prvky (obrázek 5.3). U těch transparentnost není z principu zajištěna, neboť se předpokládá, že obvodové prvky data určitým způsobem transformují, provádějí s nimi potřebné operace. U některých prvků však lze nalézt režim činnosti, kdy je prvek pro data transparentní, data přivedená na jeho vstup se objeví na jeho výstupu v nezměněné podobě. Tento režim činnosti bude dále (podle [1] a též podle [41]) nazýván i-režim (i jako identita). V tomto režimu je pak prvek využíván jen jako transportér dat. Toho lze s výhodou využít při aplikaci testu na obvod, kdy by bylo jinak nutno netransparentní prvek obejít například úpravou obvodu.



Obrázek 5.3: Příklad i-cesty při využití vlastností obvodových prvků

Poslední využívanou definicí je definice relace ρ:

Definice 5.2.2. Nechť je dána relace:
$\rho : I \times \Pi(IN \cup OUT \cup PI \cup PO)$ takto:
$(a,b)\rho(a,p_1,p_2,...,b)$, kde $\forall i \in \{1,...,k\} : p_i \in (IN \cup OUT)$,
pak v relaci ρ s i-cestou (specifikovanou počátečním a koncovým bodem) je vždy posloupnost (k-tice) bran (včetně vstupního a koncového bodu), přes které i-cesta vede.
■

Symbol Π v definici 5.2.2 znamená n-násobný kartézský součin. Pro zápis relace $(a,b)\rho(a,p1,p2,...,b)$ bude dále používána notace $\rho(a,b) = (a,p1,p2,...,b)$ a zápis $\rho(a,b)$ bude znamenat některou z posloupností $(a,p1,p2,...,b)$, která je s i-cestou (a,b) v relaci ρ, i když v případě relace ρ nejde o zobrazení. Pro jedinou i-cestu (a,b) totiž může obecně existovat více posloupností $(a,p1,p2,...,b)$, tedy mezi dvěma body v obvodě může existovat i více než jedna jediná varianta posloupnosti bran, přes které i-cesta prochází.

Příklad 5.2.1. Pro příklad obvodu na obrázku 5.2 podle definic 5.2.1 a 5.2.2 existují tyto množiny (z důvodu velikosti je uvedena pouze jejich část):
$I = \{(1,1),(1,5),(1,6),(1,8),(1,12),(1,15),(1,16),(1,18),(2,2),(2,5),(2,9),(2,11),$
$(2,13),(2,15),(2,16),(2,18),(3,3),(3,7),(3,10),(3,17),(4,4),\dots\}$
$\rho = \{((1,1),(1,1)),((1,5),(1,6,8,12,15,16,18,5)),((1,6),(1,6)),((1,8),(1,6,8)),$
$((1,12),(1,6,8,12)),((1,15),(1,6,8,12,15)),\dots\}$

5.3 Rozšíření modelu

Pro účely metodiky, která je prezentována v této dizertační práci, byl tento model dále rozšířen. Pokud je v datové cestě invertor, za nímž jsou data invertována, pak to nemusí znamenat, že takovou cestu nelze pro přenos diagnostických dat použít. Při generování testovací posloupnosti pro daný prvek je třeba zohlednit, že data vložená na začátek cesty budou přivedena na testovaný prvek v invertované podobě. Takovou cestu nazveme ii-cesta (invertující i-cesta). Dále je pak možné složením dvou ii-cest vytvořit neinvertující i-cestu. Tohoto rozšíření využívá metodika pro identifikaci TB. Další pomocná rozšíření jsou uvedena v kapitole 6.2 a 6.3.

Pro definování množiny ii-cest definice 5.3.3 je nutné zavést negaci bitové hodnoty na bráně. Negaci hodnoty na bráně zavádíme definicí 5.3.2, jež využívá definice 5.3.1, která zavádí jednobitovou negaci.

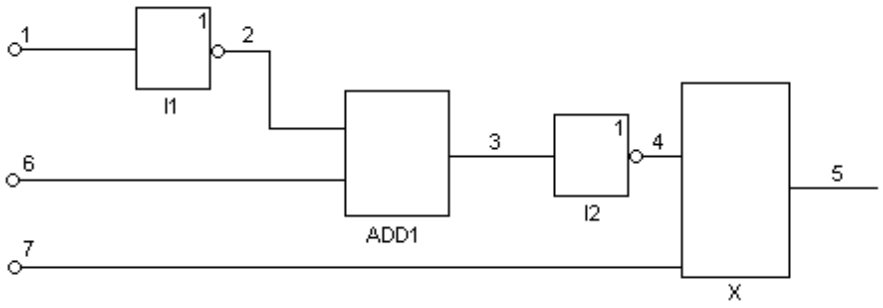
Definice 5.3.1. Nechť $x \in \{0,1,\uparrow,\downarrow\}$, pak $\neg x = \begin{cases} 0, & x = 1 \\ 1, & x = 0 \\ \uparrow, & x = \downarrow \\ \downarrow, & x = \uparrow \end{cases}$

■

Definice 5.3.2. Nechť m je šířka datové cesty, posloupnost $w = (d_1, ...d_m) \in D$ je hodnota, jež se vyskytuje na bráně, pak $\neg w = (\neg d_1, ..., \neg d_m)$ je negace hodnoty na bráně.
■

Definice 5.3.3. Nechť je možné za určitých podmínek (nastavení transparentních režimů prvků po cestě a průchodem přes inventory) z brány p_1 do brány p_2, kde $p_1 \in P, p_2 \in P$, v UUA přenést data invertovaně, pak říkáme, že mezi body p_1 a p_2 existuje ii-cesta.
Množina $I = \{(p_1,p_2) | p_1 \in P \wedge p_2 \in P \wedge \forall d \in \{0,1\}^m : \nu(p_1) = d \wedge \nu(p_2) = \neg d\}$ je množinou všech možných ii-cest v obvodě UUA.
■

Na obrázku 5.4 existuje i-cesta z uzlu 1 do uzlu 4. Je tedy možné přivést beze změny testovací data ze vstupu 1 k testovanému prvku X. Pokud chceme testovat prvek ADD1 je patrné, že na jeho vstup, který je připojený do uzlu 2, neexistuje žádná transparentní cesta (i-cesta) pro přivedení testovacích dat. Pokud ale při generování testu budeme počítat s tím, že testovací vektory budou do uzlu 2 přivedeny ze vstupu invertovaně, můžeme test vygenerovat. Říkáme tedy, že mezi uzly 1 a 2 existuje invertující i-cesta, tedy ii-cesta. Další ii-cesta je na obrázku vidět mezi uzly 6 a 4.



Obrázek 5.4: Příklad obvodu pro demonstraci rozšíření modelu

Kapitola 6

Testovatelný blok

Metodika prezentovaná v této práci je založena na pojmu TB. Metodika je vystavěna na formálním modelu popsaném v předcházející kapitole. Pro účely zde prezentované metodiky je formální model v této kapitole dále rozšířen. Cílem metodiky založené na identifikaci TB v analyzovaném obvodě je zjednodušení aplikace testu celého obvodu na problém aplikace testu menších komponent, na něž je analyzovaný obvod s využitím principů definovaných v této metodice rozdělen. Pro takto identifikované bloky je pak možné vygenerovat testovací posloupnosti sestávající z menšího počtu testovacích vektorů než by tomu bylo pro celý analyzovaný obvod. Dalším problémem souvisejícím s aplikací testu je rozhodnutí o tom, jak budou testovací vektory a odezvy na ně přenášeny analyzovaným obvodem. V metodice zde prezentované je využita technika registrů scan. Do registru scan nejsou zařazeny všechny registry analyzovaného obvodu, proto zde použitou metodu řadíme mezi tzv. metody částečný scan. Jiné metody scan jsou např. v [7, 12, 37, 41, 42].

6.1 Formální model TB

V této části bude definován formální model, který byl využit pro definování principů metodiky. Tento model musí reflektovat diagnostické a strukturální vlastnosti analyzovaného obvodu. Model pak bude využit pro implementaci metodiky.

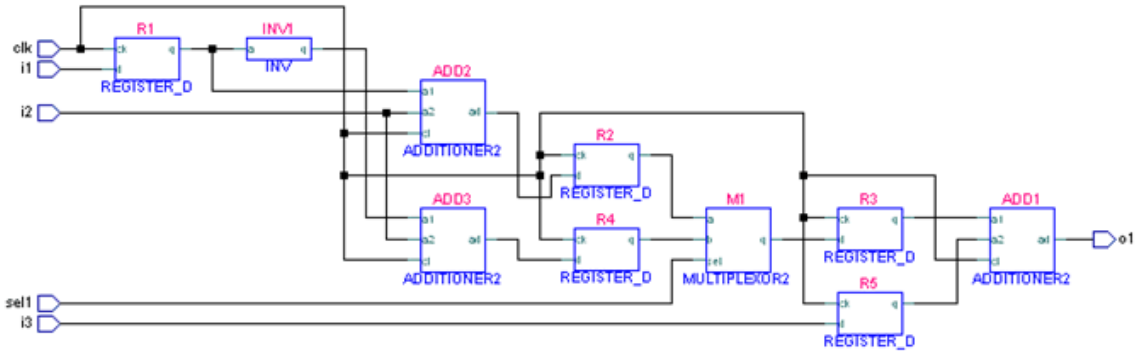
6.1.1 Vymezení obvodových prvků TB

Každý prvek z množiny obvodových prvků E vyskytující se v analyzovaném obvodě se podílí na chování celého obvodu jako celku. Totéž platí, pokud rozdělíme obvod na TB. Důležitou charakteristikou je také vzájemné propojení prvků analyzovaného obvodu, resp. TB (struktura obvodu).

Platí, že TB je částí analyzovaného obvodu. Tato skutečnost je reflektována v následující definici 6.1.1. Je postavena na faktu, že množina prvků TB je podmnožinou prvků analyzovaného obvodu.

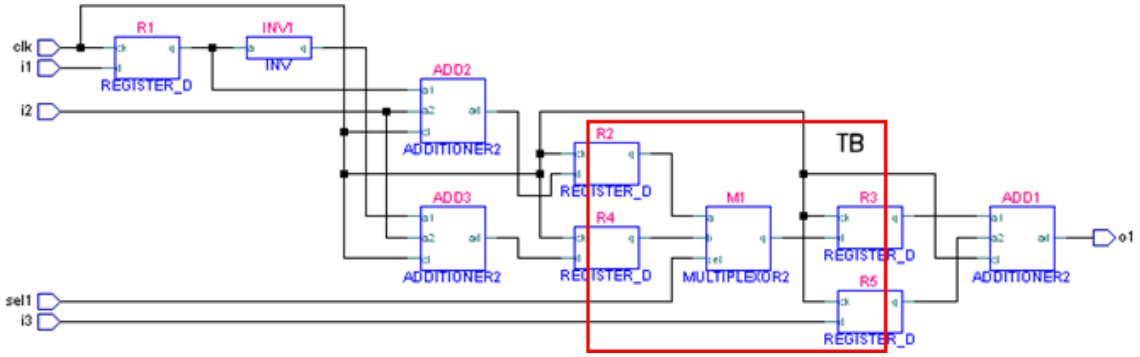
Definice 6.1.1. Nechť E je množina obvodových prvků, pak E_{TB} je množina obvodových prvků, jež patří do TB, pro niž platí $E_{TB} \subseteq E$.
■

Příklad 6.1.1. Pro případ obvodu na obrázku 6.1 platí:
$E = \{R1, INV1, ADD2, ADD3, R2, R4, M1, R3, R5, ADD1\}$



Obrázek 6.1: Příklad obvodu

Při vymezení TB podle obrázku 6.2 platí:
$E_{TB} = \{R2, R4, M1, R3, R5\}$



Obrázek 6.2: Příklad obvodu s vyznačením TB

6.1.2 Rozhraní obvodových prvků v TB

Každý obvodový prvek má jednoznačně definované rozhraní. Zde prezentovaná metodika předpokládá, že toto rozhraní neslouží pouze k realizaci patřičné funkce, ale také k aplikaci testu prvku. Obecně platí, že pro účely testování může být toto rozhraní vybaveno dodatečnými vstupy/výstupy, které jsou využívány pouze v režimu aplikace testu. Metodika založená na využití pojmu TB se však zaměřuje na analýzu struktury obvodu s cílem nezasahovat do vybavení analyzovaného obvodu a využívat pro účely testování rozhraní vzniklé v etapě návrhu obvodu. Je tedy zřejmé, že budou využívány brány prvků, přes něž jsou realizovány jejich požadované funkce. Brány mohou být vstupní, výstupní nebo řídicí.

V definici 6.1.2 je definována množina všech bran, které patří prvkům zařazeným do TB. Je zřejmé, že pro účely této metodiky je nutné tuto množinu jednoznačně identifikovat, protože role těchto bran je v metodice zásadní a nezastupitelná. Množina je definována jako sjednocení množin bran všech prvků z množiny E_{TB}. Je zřejmé, že takto identifikovaná množina je podmnožinou množiny všech bran obvodu.

Definice 6.1.2. Nechť E_{TB} je množina obvodových prvků, jež patří do TB, P je množina bran obvodových prvků a $\psi(e)$ je funkce, která přiřazuje množinu bran obvodovému prvku

e, pak $P_{TB} = \bigcup_{e \in E_{TB}} \psi(e) \subseteq P$ je množina bran obvodových prvků patřících do TB.
■

Příklad 6.1.2. Pro obvod na obrázku 6.1 platí:
$P = \{R1.ck, R1.d, R1.q, R2.ck, R2.d, R2.q, R3.ck, R3.d, R3.q, R4.ck, R4.d, R4.q, R5.ck, R5.d, R5.q, INV1.a, INV1.q, ADD1.a1, ADD1.a2, ADD1.cl, ADD1.o1, ADD2.a1, ADD2.a2, ADD2.cl, ADD2.o1, ADD3.a1, ADD3.a2, ADD3.cl, ADD3.o1, M1.a, M1.b, M1.sel, M1.q\}$
Pro případ obvodu na obrázku 6.2 s vymezením TB platí:
$P_{TB} = \{R2.ck, R2.d, R2.q, R3.ck, R3.d, R3.q, R4.ck, R4.d, R4.q, R5.ck, R5.d, R5.q, M1.a, M1.b, M1.sel, M1.q\}$

6.1.3 Propojení TB s primárními vstupy/výstupy

Metodika předpokládá, že test konkrétního TB bude realizován přes primární vstupy a primární výstupy analyzovaného obvodu a přes hraniční registry TB. Mohou nastat situace, kdy bude využita pouze jedna ze zmíněných alternativ, či obě. Předpokládá se, že hraniční registry TB budou při aplikaci testu propojeny do posuvného registru scan (viz. definice 6.1.8). Pro účely metodiky je tedy nutné rozpoznat, které primární vstupy a výstupy analyzovaného obvodu jsou propojeny se vstupy a výstupy jednotlivých TB. Tato informace je obsažena v množinách PO_{TB} a PI_{TB}. Role primárních vstupů resp. primárních výstupů analyzovaného obvodu je v této definici chápána takto: přes primární vstupy jsou vkládány testovací vektory, na primárních výstupech je naopak možné snímat odezvy na tyto testovací vektory. Důležitým principem zde prezentované metodiky je to, že tok diagnostických dat vnitřní strukturou TB je realizován paralelně přes i-cesty.

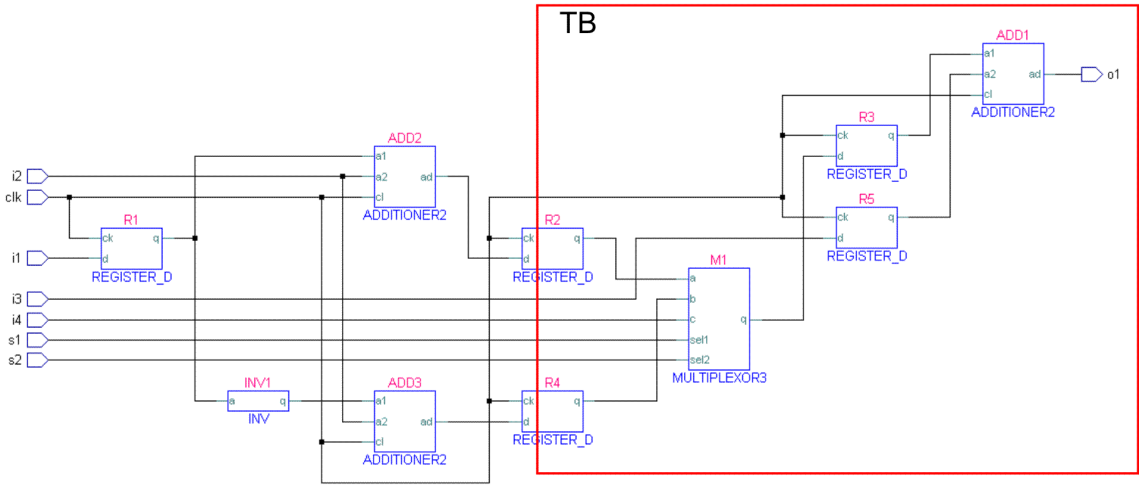
Definice 6.1.3. Nechť PO je množina primárních výstupů analyzovaného obvodu, P je množina bran obvodových prvků a C je množina spojů obvodu, pak $PO_{TB} = \{po | po \in PO \wedge (\exists p \in P_{TB} : ((p, po) \in C))\} \subseteq PO$ je podmnožina primárních výstupů obvodu, jež jsou přímo vyvedeny z TB.
■

Definice 6.1.3 říká, že PO_{TB} je podmnožina primárních výstupů obvodu, pro které platí, že existuje brána obvodového prvku patřícího do TB, která je s primárním výstupem propojena.

Definice 6.1.4. Nechť PI je množina primárních vstupů obvodu, P je množina bran obvodových prvků a C je množina spojů obvodu, pak $PI_{TB} = \{pi | pi \in PI \wedge (\exists p \in P_{TB} : ((p, pi) \in C))\} \subseteq PI$
je podmnožina primárních vstupů obvodu, jež jsou přímo přivedeny do TB.
■

Definice 6.1.4 říká, že PI_{TB} je podmnožina primárních vstupů obvodu, pro které platí, že existuje brána obvodového prvku z TB, která je s primárním vstupem spojena.

Příklad 6.1.3. Pro případ obvodu na obrázku 6.3 s vymezením TB platí:
$PI = \{i1, i2, i3, i4\}, PO = \{o1\}$
$PI_{TB} = \{i3, i4\}, PO_{TB} = \{o1\}$



Obrázek 6.3: Další příklad obvodu s vyznačením TB

6.1.4 Sekvenční obvodové prvky v TB

V sekvenčním číslicovém obvodě plní registry roli paměťových obvodových prvků. Existence paměťových prvků v konkrétní struktuře odlišuje sekvenční číslicové obvody od kombinačních. Role registrů při aplikaci testu je důležitá, přes ně jsou testovací vektory vkládány na vstupy testovaných prvků. Stejně tak odezvy na testovací vektory jsou z výstupů testovaných prvků vkládány do registrů, ze kterých jsou snímány. Pro rozlišení registrů od ostatních kombinačních prvků v TB je zavedena množina R_{TB}.

Definice 6.1.5. Nechť E_{TB} je množina obvodových prvků, jež patří do TB a R je množina registrů obvodu, pak $R_{TB} = \{r | r \in R \wedge r \in E_{TB}\}$ je množina registrů TB.
■

Definicí 6.1.5 je vymezena množina registrů patřících do TB. Tato množina je dále použita v definici hraničních registrů.

6.1.5 Rozhranní TB

Dalšími významnými prvky, které jsou důležité pro proces aplikace testu a musí být proto jednoznačně v TB identifikovány, jsou hraniční registry. Tyto představují jednu ze dvou[1] možností pro přivedení testovacích vektorů a snímání odezev na testovací vektory. Hraniční registry tvoří také rozhraní, přes něž jsou jednotlivé TB propojeny do jednoho celku. Rozlišujeme dva typy hraničních registrů: výstupní hraniční registry a vstupní hraniční registry.

Definice 6.1.6. Nechť R_{TB} je množina registrů TB, C je množina spojů obvodu, P je množina bran obvodových prvků, $\psi(e)$ je funkce, která přiřazuje množinu bran obvodovému prvku e a P_{TB} je množina bran obvodových prvků uvnitř TB, pak $BRO_{TB} = \{r | r \in R_{TB} \wedge \exists(p_1, p_2) \in C : (p_1 \in \psi(r) \wedge p_2 \in (P \backslash P_{TB}))\}$ je množina výstupních hraničních registrů TB.
■

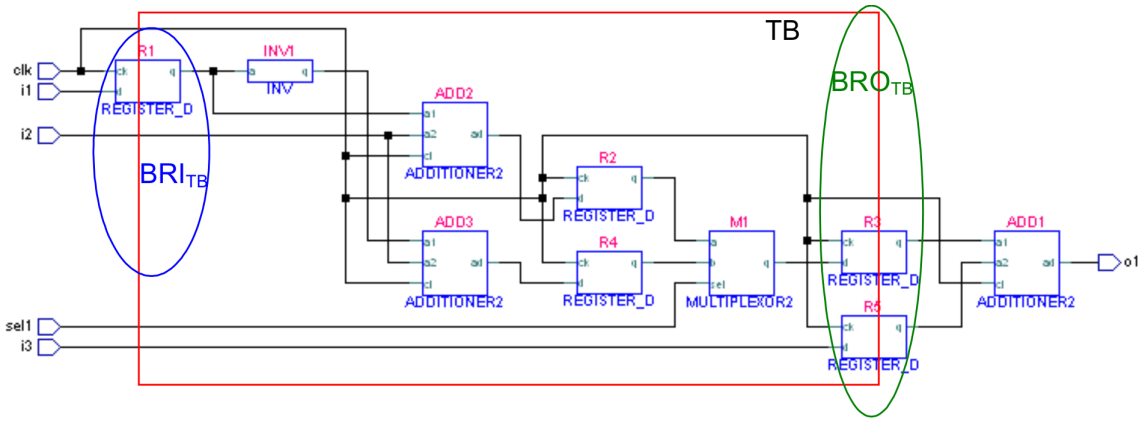
[1] Druhou možností je existence přímého propojení prvku uvnitř TB na PI/PO. (Kapitola 6.1.3)

BRO_{TB} je podle definice 6.1.6 podmnožina registrů TB pro něž platí, že existují spoje, zařazené do množiny spojů obvodu, které začínají v bráně těchto registrů a končí v bráně prvku, který nepatří do TB (příklad 6.1.4).

Definice 6.1.7. Nechť R_{TB} je množina registrů TB, C je množina spojů analyzovaného obvodu, P je množina bran obvodových prvků, $\psi(e)$ je funkce, která přiřazuje množinu bran obvodovému prvku e a P_{TB} je množina bran obvodových prvků uvnitř TB, pak $BRI_{TB} = \{r | r \in R_{TB} \wedge \exists (p_1, p_2) \in C : (p_1 \in (P \backslash P_{TB})) \wedge p_2 \in \psi(r)\}$ je množina vstupních hraničních registrů TB.
■

BRI_{TB} je podle definice 6.1.7 podmnožina registrů TB pro něž platí, že existují spoje, zařazené množiny spojů obvodu, které začínají v bráně prvku, který nepatří do TB a končí v bráně těchto registrů (příklad 6.1.4).

Definice 6.1.8. Nechť BRO_{TB} je množina výstupních hraničních registrů TB a BRI_{TB} je množina vstupních hraničních registrů TB, pak $BR_{TB} = BRO_{TB} \cup BRI_{TB}$ je množina hraničních registrů TB.
■



Obrázek 6.4: Příklad obvodu s vyznačením hraničních registrů TB

Příklad 6.1.4. Pro obvod na obrázku 6.4 s vymezením TB platí:
$R_{TB} = \{R1, R2, R3, R4, R5\}$, $BRI_{TB} = \{R1\}$, $BRO_{TB} = \{R3, R5\}$, $BR_{TB} = \{R1, R3, R5\}$

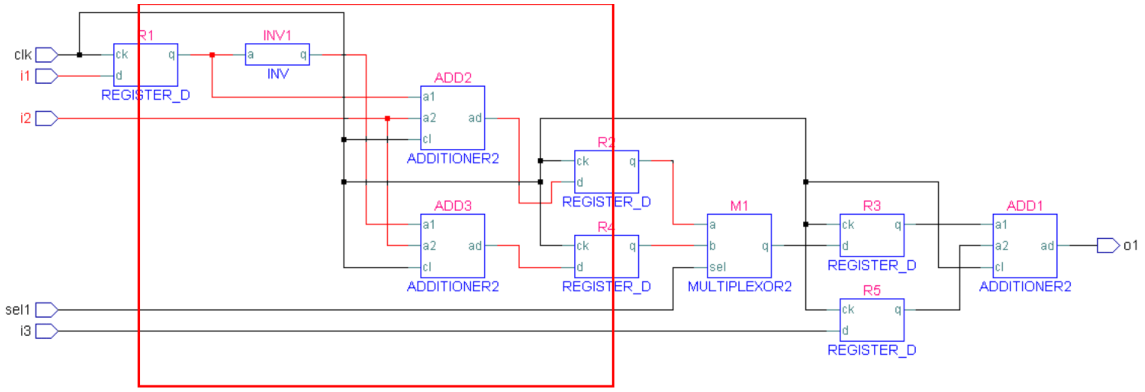
6.1.6 Struktura TB

Výsledkem návrhu obvodu, který je následně předmětem naší analýzy, jejímž cílem je jeho rozdělení na TB, je kromě jiného jeho přesně definovaná struktura. Pro účely této analýzy je nutné toto propojení rozpoznat a patřičně je reprezentovat ve formálním modelu. Pro tuto reprezentaci byla zvolena reprezentace formou množiny uspořádaných dvojic - tuto množinu označujeme termínem množina spojů. Přes spoje jsou vedeny jak testovací tak provozní data a mohou být jednobitové nebo vícebitové.

Definice 6.1.9. Nechť C je množina spojů obvodu, P_{TB} je množina bran obvodových prvků uvnitř TB, PO_{TB} je podmnožina primárních výstupů obvodu, jež jsou připojeny do TB, P je množina všech bran obvodových prvků analyzovaného obvodu, $\psi(e)$ je funkce, která

přiřazuje množinu bran obvodovému prvku e, BRO_{TB} je množina výstupních hraničních registrů TB a BRI_{TB} je množina vstupních hraničních registrů TB, pak $C_{TB} = C \cap [(P_{TB} \times (PO_{TB} \cup P_{TB})) \cup ((PO_{TB} \cup P_{TB}) \times P_{TB}) \cup ((\bigcup_{e \in BRO_{TB}} \psi(e)) \times P) \cup (P \times (\bigcup_{e \in BRI_{TB}} \psi(e)))]$ je množina spojů TB.
■

Definice 6.1.9 říká, že množina spojů TB je podmnožinou množiny spojů celého obvodu a zároveň do této množiny patří pouze spoje uvnitř TB, spoje z PI do TB, spoje z TB do PO a spoje z/do hraničních registrů TB.



Obrázek 6.5: Příklad obvodu s vyznačením spojů TB

Příklad 6.1.5. Pro obvod na obrázku 6.5 s vymezením TB platí:
$C_{TB} = \{(i1, R1.d), (R1.d, INV1.a), \ldots\}$ (červené spoje)

6.1.7 Formální model TB

Konečná formální reprezentace TB je tedy shrnuta do následující definice 6.1.10.

Definice 6.1.10. Nechť E_{TB} je množina obvodových prvků, jež patří do TB, P_{TB} je množina bran obvodových prvků uvnitř TB, PO_{TB} je podmnožina primárních výstupů obvodu, jež jsou připojeny do TB, C_{TB} je množina spojů TB, PI_{TB} je podmnožina primárních vstupů obvodu, jež jsou připojeny do TB a nechť platí $\forall e \in E_{TB}, \forall p \in \psi(e), \exists (p_1, p_2) \in I : (((p_1 \in PI_{TB}) \vee (p_1 \in BR_{TB})) \wedge p_2 = p) \vee (p_1 = p \wedge (p_2 \in PO_{TB}) \vee (p_2 \in BR_{TB}))$, pak $TB = (E_{TB}, P_{TB}, C_{TB}, PI_{TB}, PO_{TB})$ představuje formální model TB.
■

Reprezentace vytvořená podle definice 6.1.10 reprezentuje formální model TB. Část definice $[\forall e \in E_{TB}, \forall p \in \psi(e), \exists (p_1, p_2) \in I : (((p_1 \in PI_{TB}) \vee (p_1 \in BR_{TB})) \wedge p_2 = p) \vee (p_1 = p \wedge (p_2 \in PO_{TB}) \vee (p_2 \in BR_{TB}))]$ říká, že všechny obvodové prvky, z nichž je TB složen, musí být přístupné přes i-cesty. Všechny i-cesty určené pro přenos testovacích vektorů, musí začínat ve vstupním hraničním registru TB nebo na primárním vstupu analyzovaného obvodu, zatímco i-cesta, přes niž jsou přenášeny odezvy, musí končit ve výstupním hraničním registru TB nebo na primárním výstupu analyzovaného obvodu.
Tyto skutečnosti je možné vyjádřit také těmito dvěma konstatováními:

1. vstupní brány vnitřních prvků TB musí být řiditelné z hraničních registrů TB nebo primárních vstupů analyzovaného obvodu,

2. výstupní brány vnitřních prvků TB musí být pozorovatelné v hraničních registrech TB nebo na primárních výstupech analyzovaného obvodu.

6.1.8 Nutné podmínky pro existenci TB

Dále musí být splněny následující tvrzení, jež představují nedílnou součást definice TB:

Tvrzení 6.1.1. Žádné dva TB se nesmí překrývat:
Pro každé dva různé testovatelné bloky TB_x a TB_y musí platit:
$\forall e \in (E_{TB_x} \backslash BR_{TB_x}) : e \notin E_{TB_y}$
Překrývání množin P_{TB} a C_{TB} pak není možné díky platnosti předchozích definic.

Tvrzení 6.1.2. Pouze hraniční registry mohou být zapojeny do řetězce scan:
$\forall r \in (R_{TB} \backslash BR_{TB}) : r \notin SCAN$

Tvrzení 6.1.1 říká, že všechny obvodové prvky testovatelného bloku TB_x, mimo hraničních registrů BR_{TB_x}, nesmí být součástí jiného testovatelného bloku TB_y. Jinak řečeno, množiny, které reprezentují strukturu TB, musí být disjunktní.

Tvrzení 6.1.2 říká, že žádný registr, který patří do množiny registrů TB a nepatří do množiny hraničních registrů TB, nebude zařazen do množiny registrů scan. Je to důsledek již zmíněného principu - **diagnostická data jsou strukturou TB přenášena paralelně, díky čemuž není nutné jejich vnitřní registry vkládat do registru scan.**

6.2 Metodika identifikace TB

Metodika, jejímž cílem je rozdělení analyzovaného obvodu na TB, je implementována na výše definované reprezentaci. Na této reprezentaci jsou identifikovány všechny prvky a spoje, příp. další vlastnosti analyzovaného obvodu. Vytvořené algoritmy nepracují tudíž s reprezentací např. na úrovni jazyka VHDL, ale s formální reprezentací definovanou v této práci. Základní princip spočívá v rozdělení analyzovaného obvodu na jistý počet TB, které jsou navzájem odděleny registry[2]. Počet TB může ovlivnit např. počet testovacích vektorů a dobu trvání aplikace testu celého obvodu. Jednotlivé TB jsou pak plně testovatelné přes tyto registry a primární vstupy a výstupy analyzovaného obvodu.

Jádrem této části je popis metodiky pro rozdělení obvodu na TB na základě identifikování některých registrů jako hraničních. Další částí metodiky pak jsou: postup pro určení počtu synchronizačních pulsů nutných k průchodu testovacích dat přes TB a detekce smyček. Všechny zmíněné techniky jsou implementovány na formálním modelu definovaném v této práci. Součástí implementace je pak optimalizační algoritmus hledající nejvhodnější rozdělení obvodu[3], které co nejvíce splňuje požadovaná kriteria.

[2] hraničními registry
[3] výběr hraničních registrů

6.2.1 Metodika pro rozdělení obvodu na TB

V této části bude popsán postup rozdělení analyzovaného obvodu na TB. S konkrétním rozdělením velmi úzce souvisí identifikace registrů, které budou při aplikaci testu plnit funkci registrů hraničních. Vstupem tohoto algoritmu bude formální model obvodu vytvořený podle zde definovaných definic a principů. Vývojový diagram algoritmu bude uveden a popsán v kapitole 6.3 o implementaci, neboť s ní bezprostředně souvisí.

Vstupem formálního algoritmu je formální model analyzovaného obvodu podle definice v kapitole 5 a konečný bitový vektor (viz. definice 6.2.1) v, jež má délku stejnou, jako je počet registrů. Jednotlivé bity vektoru pak určují, zda je registr hraniční (ve vektoru označen 1) nebo není hraniční (ve vektoru označen 0).

Definice 6.2.1. Nechť existuje kladné přirozené číslo n a nechť existuje funkce, jejímž definičním oborem D jsou přirozená čísla 1 až n ($D = \{1, 2, 3, 4, ...n\}$) a oborem hodnot je dvojice číslic 0 a 1 ($H = \{0, 1\}$). Pak se tato funkce nazývá *konečný bitový vektor*[4] a číslo n se nazývá *délka vektoru*.
■

Značení: Je zvykem prvky z definičního oboru vektoru označovat n (případně $k, l, i, ...$) a funkční hodnoty vektoru označovat a_n (případně b_n, a_k, ...) a nazývat je členy vektoru. Funkční hodnoty vektoru (členy vektoru) v je také možné označovat $v[n]$, $v[n] = a_n$. My se budeme držet tohoto posledního zmíněného značení.

Výstupem formálního algoritmu je dvojice množin E_TB a R_TB, které určují zařazení jednotlivých obvodových prvků do TB.

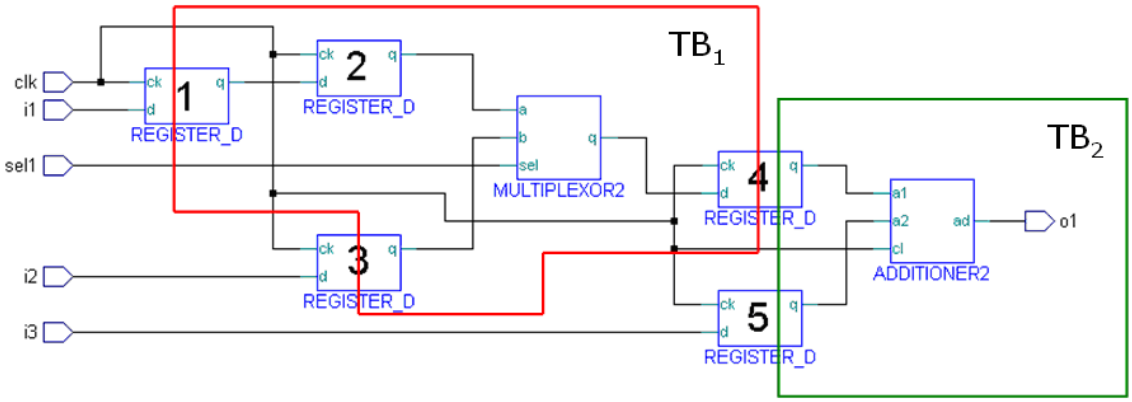
Věta 6.2.1. Množina uspořádaných dvojic, která každému obvodovému prvku mimo registrů přiřazuje právě jedno celé kladné číslo, nebo jedno číslo z množiny {-1, 0} bude označena:
$E\_TB = \{(e, t) : e \in (E/R), t \in (N \cup \{-1, 0\})\}$.
Význam hodnot t je pak následující:
Je-li $t > 0$, pak t označuje TB, do kterého patří.
Je-li $t = 0$, pak prvek nepatří do žádného TB.
Je-li $t = -1$, pak je prvek připojen přímo na primární vstup nebo primární výstup.

Věta 6.2.2. Množina uspořádaných dvojic, která každému registru přiřazuje právě jednu dvojici přirozených čísel rozšířených o 0 a -1 bude označena:
$R\_TB = \{(r, (ti, to)) : r \in R, ti \in (N \cup \{-1, 0\}), to \in (N \cup \{-1, 0\})\}$.
Význam hodnot ti a to je pak následující:
Je-li $ti > 0$, pak to označuje TB, do kterého patří vstup registru.
Je-li $ti = 0$, pak vstup registru nepatří do žádného TB.
Je-li $ti = -1$, pak je vstup registru připojen přímo na primární vstup.
Je-li $to > 0$, pak to označuje TB, do kterého patří výstup registru.
Je-li $to = 0$, pak výstup registru nepatří do žádného TB.
Je-li $to = -1$, pak je výstup registru připojen přímo na primární výstup.

Užití předcházejících množin bude nyní demonstrováno na obvodu z obrázku 6.6.

Příklad 6.2.1. Obrázek 6.6 ukazuje příklad pro chromozom: 10111
R_TB={[R1,(-1,1)], [R2,(1,1)], [R3,(-1,1)], [R4,(1,2)], [R5,(-1,2)]}, E_TB={(MX1,1), (ADD1,1)}

[4] případně konečná bitová posloupnost



Obrázek 6.6: Příklad rozdělení na TB

V následujícím algoritmu je uveden formální postup určení R_TB, E_TB z bitového vektoru v:

Algoritmus 6.2.1.

1. $E\_TB = \{(e,0)|e \in (E/R)\}$
2. $R\_TB = \{(r,(0,0))|r \in R\}$
3. $BR = \{r|v[r] = 1\}$
4. $Tbnum = 1$
5. $\forall r \in BR$:
   (a) $(\forall p \in \psi(r) : ($
      i. $(p \in IN) \vee (\exists x : ((r,(0,x)) \in R\_TB)) \vee (\exists (p,b) \in C : b \in PI) : E\_TB = E\_TB/\{(r,(0,x))\} \cup \{(r,(-1,x))\}$
      ii. $(p \in OUT) \vee (\exists x : ((r,(x,0)) \in R\_TB)) \vee (\exists (b,p) \in C : b \in PO) : E\_TB = E\_TB/\{(r,(x,0))\} \cup \{(r,(x,-1))\}$
      iii. $(p \in IN) \vee (\exists x : ((r,(0,x)) \in R\_TB)) \vee \neg(\exists (p,b) \in C : b \in PI) : ($
         - $(\forall x : (x,p) \in I)$:
           - $(\forall a_j \in \rho(x,p))$:
             A. $a_j \notin \bigcup_{r_i \in R_{TB}} \psi(r_i) \vee (\exists e : a_j \in \psi(e)) \vee (e,0) \in E\_TB : E\_TB = E\_TB/(e,0) \cup (e,Tbnum)$
             B. $a_j \in \bigcup_{r_i \in R_{TB}} \psi(r_i) \vee r_i \notin BR \vee (r_i(0,0)) \in R\_TB \vee (\exists e : a_j \in \psi(e)) : R\_TB = R\_TB/(e,(0,0)) \cup (e,(Tbnum,Tbnum))$
             C. $a_j \in \bigcup_{r_i \in R_{TB}} \psi(r_i) \vee r_i \in BR \vee \exists y : (r_i,(y,0)) \in R\_TB \vee (\exists e : a_j \in \psi(e)) : R\_TB = R\_TB/(r_i,(y,0)) \cup (r_i,(y,Tbnum))$
)
      iv. $(p \in OUT) \vee (\exists x : ((r,(x,0)) \in R\_TB)) \vee \neg(\exists (b,p) \in C : b \in PI) : ($

- $(\forall x : (p, x) \in I)$:
  - $(\forall a_j \in \rho(p, x))$:
    A. $a_j \notin \bigcup_{r_o \in R_{TB}} \psi(r_o) \vee (\exists e : a_j \in \psi(e)) \vee (e, 0) \in E\_TB : E\_TB = E\_TB/(e, 0) \cup (e, Tbnum)$
    B. $a_j \in \bigcup_{r_o \in R_{TB}} \psi(r_o) \vee r_o \notin BR \vee (r_o, (0, 0)) \in R\_TB \vee (\exists e : a_j \in \psi(e)) : R\_TB = R\_TB/(e, (0, 0)) \cup (e, (Tbnum, Tbnum))$
    C. $a_j \in \bigcup_{r_o \in R_{TB}} \psi(r_o) \vee r_o \in BR \vee \exists y : (r_o, (0, y)) \in R\_TB \vee (\exists e : a_j \in \psi(e)) : R\_TB = R\_TB/(r_o, (0, y)) \cup (r_i, (Tbnum, y))$
)
)

Vstupem algorimu 6.2.1 je bitový vektor v a formální model obvodu. Výstupem jsou množiny R_TB a E_TB. Uvedený algoritmus lze slovně vyjádřit takto (číslování odpovídá algoritmu):

1. Množina E_TB je naplněna prvky určujícími, že všem obvodovým prvkům je přiřazena hodnota $t = 0$. To značí, že dosud nebylo rozhodnuto o přiřazení obvodového prvku do některého TB.

2. Množina R_TB je naplněna prvky určujícími, že všem registrům je přiřazena dvojice hodnot $ti = to = 0$. To značí, že dosud nebylo rozhodnuto o přiřazení vstupu a výstupu registru do některého TB.

3. Je vytvořena množina hraničních registrů BR podle vektoru v tak, že jsou do množiny BR vloženy všechny registry, u nichž odpovídající člen vektoru v je roven 1.

4. Počítadlo je uvedeno do výchozí hodnoty 1.

5. Dále se provádí pro všechny hraniční registry r následující postup:

   (a) pro všechny brány[5] p každého hraničního registru r se dále zkoumá:

      i. Je-li brána p vstupní, vstup registru r, jemuž brána patří, není zařazen do žádného TB a tato brána je propojena na primární vstup. Vstup registru r je označen „-1“ (což znamená: připojen přímo na PI).

      ii. Je-li brána p výstupní, výstup registru r, jemuž brána patří, není zařazen do žádného TB a tato brána je propojena na primární výstup. Výstup registru r je označen „-1“ (což znamená: připojen přímo na PO).

      iii. Je-li brána p vstupní, vstup registru r, jemuž brána patří, není zařazen do žádného TB a tato brána není propojena na primární vstup. Pokračuje se následujícím postupem:

         - U všech i-cest vedoucích do zkoumané brány p, jejichž výchozí bod je označen x, se prochází všechny brány na i-cestě a pro každou bránu se zjišťuje toto:

[5] množina bran je dána funkcí $\psi(r)$, kde r je aktuálně zpracovávaný registr a $\psi(r)$ představuje množinu jeho bran

A. Nepatří-li brána registru, ale nějakému prvku e, který není přiřazen do žádného TB, je prvku e přiřazeno[6] číslo TB.

B. Patří-li brána registru, který není hraniční a kterému není přiřazeno číslo TB, je vstupu i výstupu tohoto registru přiřazeno[7] stejné číslo TB.

C. Patří-li brána registru, který je hraniční a kterému není přiřazeno číslo TB, je výstupu tohoto registru přiřazeno[8] číslo TB a pokračuje se další i-cestou.

iv. Je-li brána p výstupní, výstup registru r, jemuž brána patří, není zařazen do žádného TB a tato brána není propojena na primární výstup. Pokračuje se následujícím postupem:

- U všech i-cest vedoucích ze zkoumané brány p, jejichž koncový bod je označen x, se prochází všechny brány na i-cestě a pro každou bránu se zjišťuje toto:

A. Nepatří-li brána registru, ale nějakému prvku e, který není přiřazen do žádného TB, je prvku e přiřazeno číslo TB.

B. Patří-li brána registru, který není hraniční a kterému není přiřazeno číslo TB, je vstupu i výstupu tohoto registru přiřazeno číslo TB.

C. Patří-li brána registru, který je hraniční a kterému není přiřazeno číslo TB, je vstupu tohoto registru přiřazeno číslo TB a pokračuje se další i-cestou.

6.2.2 Počet pulsů potřebných pro průchod dat i-cestou

Jedním z kritérií, jež se používají při výpočtu kvality rozdělení obvodu na TB, je stejný počet synchronizačních pulzů jednotlivých TB potřebných k jejich otestování. Pro tyto účely je v definici 6.2.2 definován počet pulzů potřebných k průchodu diagnostických dat přes testovatelný blok (s_{TB}).

Definice 6.2.2. Nechť je dána funkce $\sigma : E \to \mathbf{N}$, která každému prvku z množiny prvků E přiřazuje celé nezáporné číslo určující počet synchronizačních pulsů potřebných pro průchod dat přes tento prvek. Prvkům, které nejsou synchronizovány (jsou prvky podmnožin MX a FU) je přiřazena 0. Dále nechť existuje posloupnost (cesta ze vstupu TB na výstup TB) v TB: $(a, x_1, x_2, x_3, \ldots, x_n, b)$, pro niž platí:

- $\forall i \in \{1..n\} : x_i \in P_{TB} \wedge ((x_i \in OUT_{TB} \wedge (i \text{ je liché})) \vee ((x_i \in IN_{TB} \wedge (i \text{ je sudé}))$
- $a \in PI_{TB} \vee (a \in OUT_{TB} \wedge \psi^{-1}(a \in BR_{TB}))$
- $b \in PO_{TB} \vee (b \in OUT_{TB} \wedge \psi^{-1}(a \in BR_{TB}))$

pak počet synchronizačních pulsů potřebných pro průchod dat touto cestou s je roven: $s = \sum_{i=1,3,5,\ldots n-1} \sigma(\psi^{-1}(x_i))$. Maximální hodnota s je pak rovna s_{TB}.

■

[6] zápisem do množiny E_TB
[7] zápisem do množiny R_TB
[8] zápisem do množiny R_TB

6.2.3 Vliv smyček v obvodu rozděleného na TB

Při testování metodiky se ukázalo, že některé obvody jsou rozdělovány velmi nerovnoměrně[9]. Touto skutečností bylo nutno se zabývat. Bylo zjištěno, že se v obvodech vyskytují různé typy propojení obvodových prvků, které znemožňují použití metody TB. Hlavním problémem jsou různé typy smyček. Tyto části obvodu působí problémy při použití navržené metodiky, proto je nutné je detekovat. K tomuto účelu byla vyvinuta metodika, jež je schopna tyto konstrukce odhalit a vložit do obvodu podpůrný testovací prvek. V této kapitole bude popsána metoda, pomocí niž jsou smyčky detekovány a je realizováno základní rozdělení smyček na přímé a nepřímé. Dále bude naznačen způsob přerušení smyček vložením testovacího prvku. Výsledky této úpravy jsou pak diskutovány v kapitole 7.

V tabulce 6.1 je uveden příklad nerovnoměrného rozdělení obvodu. TB1 je neúměrně rozsáhlý. Je to způsobeno tím, že mnoho registrů nelze použít jako hraniční. Tento rys je demonstrován na obrázku 6.7, kde jsou elipsou označeny dva registry a tučnou čarou spoje, jež vytváří přímou smyčku. **Přímou smyčkou** je tedy nazývána datová cesta, jež končí a začíná ve stejném obvodovém prvku. U levého registru jsou dvě smyčky. Další část obvodu se smyčkami je na obrázku 6.8 a část řadiče ISA sběrnice je na obrázku 6.9. Tato skutečnost znemožňuje použití označených registrů jako hraničních.

Označení	Hraničních reg.	Vnitřních reg.	Funkčních jedn.	Uzlů	Test. vektorů
TB1	28	86	222	241	139
TB2	3	3	8	7	22
TB3	3	3	8	7	22
TB4	3	3	8	7	22
TB5	5	1	6	3	9
TB6	5	1	6	3	9

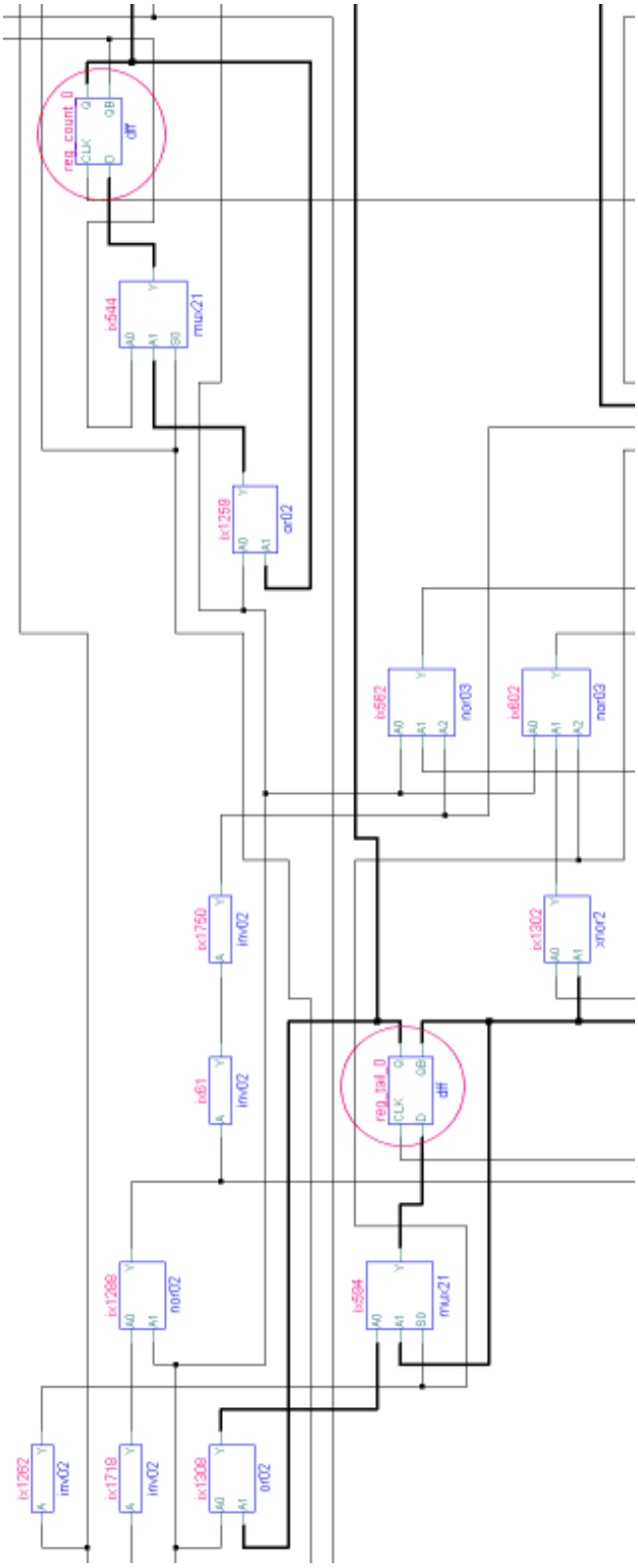
Tabulka 6.1: Příklad rozdělení obvodu FIFO2 na TB

Další obvodovou strukturou, která komplikuje aplikaci testu, je existence **nepřímých smyček**. Takto je označena smyčka, jež se uzavírá i mezi dvěma vstupy nebo výstupy jednoho prvku (obrázek 6.10). Smyčka probíhá registrem 2, pokračuje na vstup a multiplexoru a vrací se ze vstupu b na vstup registru d. Protože toto zapojení odporuje definici 6.1.9, není možné registr použít jako hraniční. Reálný příklad nepřímé smyčky vidíme na obrázku 6.11.

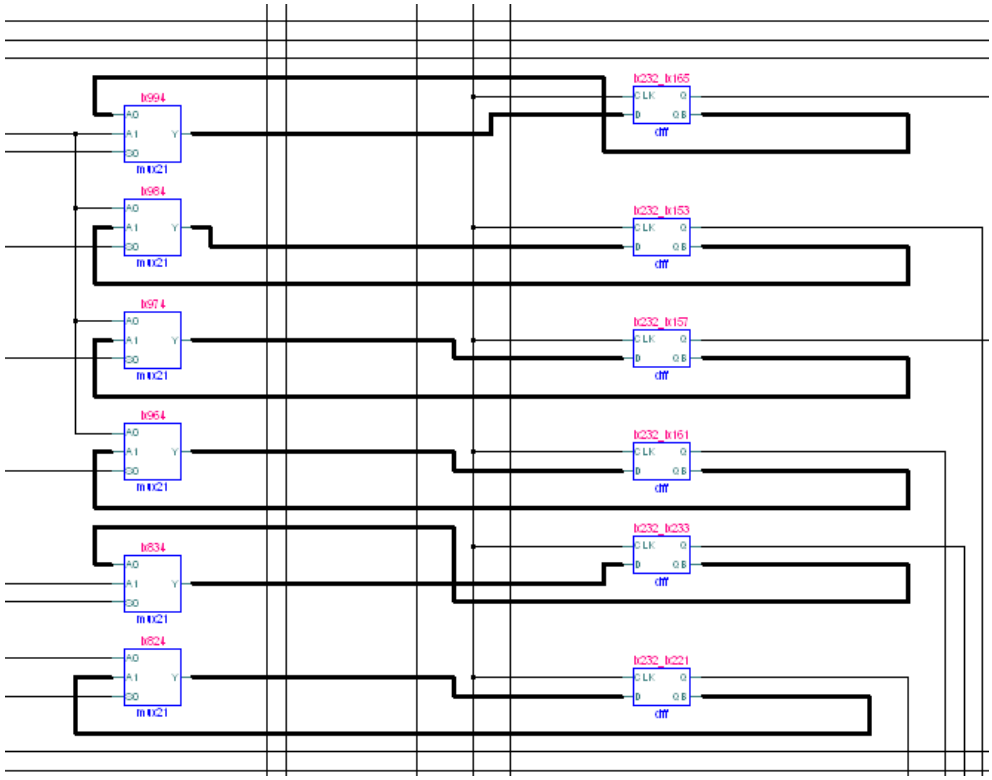
Řešením problému by mohlo být přerušení smyček „diagnostickými registry“ (DR). DR je speciální prvek, jež je transparentní ve funkčním režimu, v režimu test je zařazen do řetězce scan. V odborné literatuře bývá tento problém označován jako „eliminace zpětnovazebních smyček zařazením registru scan“. Kvalita řešení tohoto problému se odvíjí od rozhodnutí, kam v obvodové struktuře DR umístíme. Byly ověřeny tyto dvě možnosti: a/ náhodné umístění DR, b/ umístění tak, aby bylo přerušeno co nejvíce smyček. Druhá alternativa vyžaduje detailní analýzu obvodu, což může být časově náročné.

Smyčky jsou vyhledávány ve dvou fázích. V první fázi jsou vyhledávány nepřerušitelné smyčky (což jsou smyčky, které neobsahují registr, kterým by se dala smyčka přerušit). Zkoumají se datové cesty v obvodu, jež končí i začínají v témže registru, obsahují-li další registr. Pokud neobsahují takový registr, je cesta uložena k pozdějšímu doplnění DR. Vyhledávání nepřímých smyček je mnohem složitější. Je třeba procházet obvod s ohledem na možnost uzavření smyčky mezi jednotlivými vstupy nebo výstupy téhož prvku. Je třeba

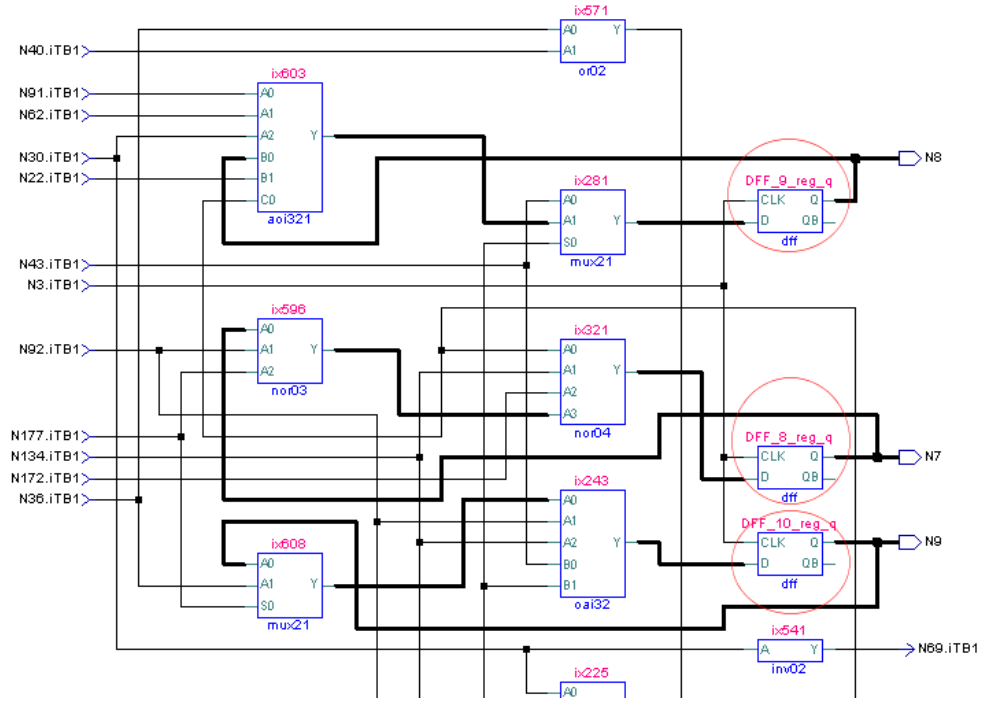
[9]Některý TB byl neúměrně velký proti ostatním. Nebo byly některé bloky příliš malé.



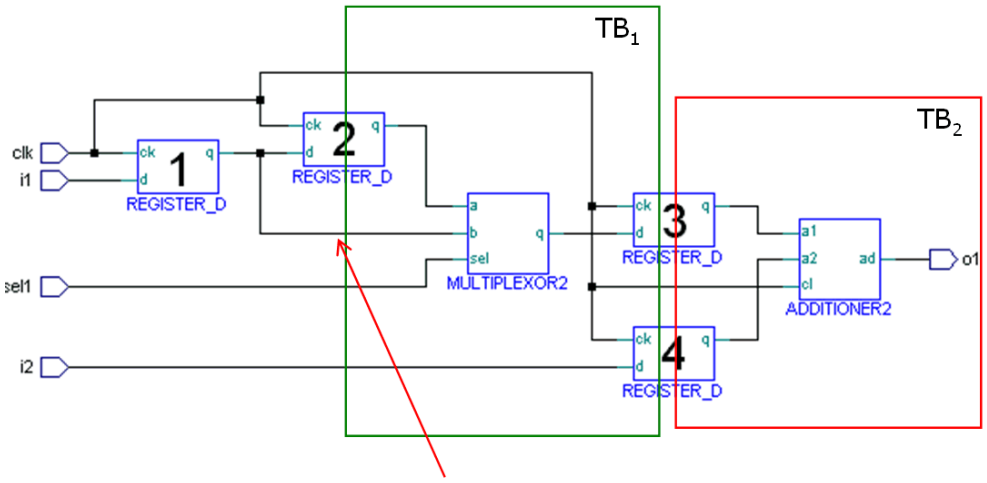
Obrázek 6.7: Příklad obvodu s vyznačením registrů a smyček



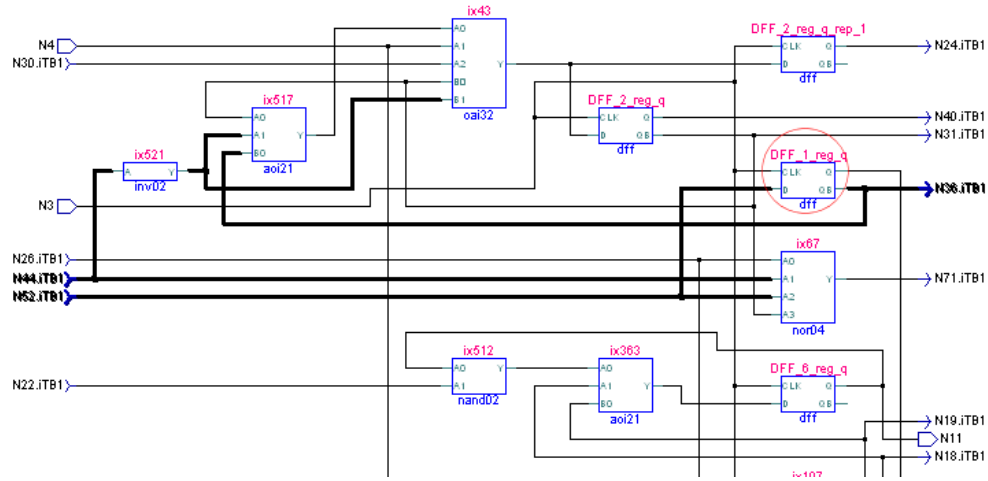
Obrázek 6.8: Příklad obvodu s vyznačením smyček



Obrázek 6.9: Příklad obvodu řadiče sběrnice ISA s vyznačením registrů a smyček

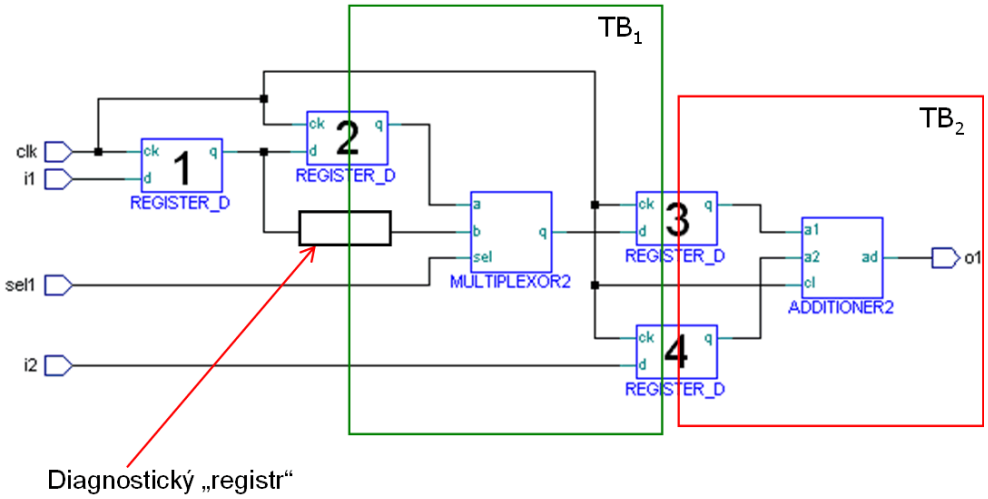


Obrázek 6.10: TB_1 není TB. Nesplňuje podmínku oddělení registry.



Obrázek 6.11: Příklad reálného obvodu s vyznačením nepřímé smyčky a registru

rozvětvovat cestu na každém procházeném prvku a pokud je nalezena smyčka, je nutné zjistit, zda se v ní vyskytuje registr. Nakonec jsou označené smyčky přerušeny registrem DR, který se vloží do smyčky. Je nutné aktualizovat všechny množiny modelu. Výsledný model (s DR) se rozdělí na TB. Po rozdělení obvodu je nutné odstranit DR, jež nebudou využity jako hraniční, což se provádí automaticky v modulu pro rozdělení na TB.



Obrázek 6.12: Příklad umístění „diagnostického registru“

Podrobnější funkční popis je uveden v kapitole 6.3.3 Implementace.

6.3 Implementace

6.3.1 Volba algoritmu

Protože velikost prohledávaného prostoru exponenciálně roste v závislosti na počtu registrů v obvodu, byl pro řešení problému zvolen **genetický algoritmus** a **simulované žíhání**.

Genetické algoritmy náleží do kategorie alternativních metod, kam patří např. metoda Monte Carlo, fuzzy logika, neuronové sítě, metoda simulovaného žíhání apod. Důvodem vzniku a rozšíření těchto metod byla stále narůstající složitost řešených úloh. V praktických problémech se objevila nelinearita, náhodnost, velké množství vzájemných vztahů a začaly chybět metody, které by dokázaly tyto problémy vyřešit. Klasické matematické metody, jako jsou např. lineární programování, statistika nebo regresní a korelační analýza, jsou omezené různými požadavky, jako linearita, počet vzájemných vztahů, rozsah hodnot apod. Navíc existují problémy, kde vůbec není možné tyto metody použít, protože je nelze popsat tradičními způsoby. Existence zpětných vazeb, skokové závislosti, náhodné rozhodování, to vše jsou aspekty, které naplňují praktické problémy stále více, a které způsobují selhání klasických metod.

Jelikož v C zabudovaný kongruentní generátor náhodných hodnot `rand()` je pro generování reálných čísel nepoužitelný, byl použit generátor náhodných čísel „Mersenne Twister“ publikovaný v [60] , který je cca 4x rychlejší a poskytuje kvalitnější posloupnost.

6.3.2 Fitness funkce

Důležitou součástí genetického algoritmu a simulovaného žíhání je ohodnocení každého jedince. Tento problém je řešen s využitím fitness funkce. Jejím úkolem je rozdělit obvod podle daného chromozomu a určit kvalitu rozdělení na TB. Princip rozdělení analyzovaného obvodu na TB využívající fitness funkce má tuto podobu:

Jsou identifikovány množiny R_TB **a** E_TB z binárního vektoru (chromozomu). Jednička na dané pozici ve vektoru určuje, že bude registr brán jako hraniční.
Postup výpočtu je následující:
V množinách R_TB a E_TB se pro všechny prvky položí $t = 0$, což určuje, že dosud nebylo rozhodnuto o přiřazení k některému TB. Dále se pro všechny registry provádí následující postup:
Pokud nemá být registr hraničním (ve vektoru označen 0), pokračuje se dalším registrem. Pokud ano, začne se procházet množina $\psi(r)$, kde r je aktuálně zkoumaný registr a $\psi(r)$ představuje množinu jeho bran. Výpočet se dále větví podle toho, zda je brána vstupní či výstupní.

Je-li brána **vstupní**:
Zjistí se, zda je již přiřazena vstupní část[10] registru k nějakému TB. Pokud ano, pokračuje se další branou. Pokud ne, zjišťuje se dále, zda je brána připojena na primární vstup (PI). Pokud ano, je označen speciální značkou a pokračuje se další branou. Pokud není připojen na PI, začne se procházet množina i-cest ρ, u kterých je analyzovaná brána koncem i-cesty. I-cesta se prochází od konce (od analyzované brány) a pro každou bránu na této i-cestě se provádí tato analýza:

1. Nepatří-li brána registru zjišťujeme, zda už nemá přiřazeno číslo TB, do kterého patří. Pokud nemá, přiřadí se mu nové číslo právě vytvářeného TB. Pokud již má číslo TB, dosavadní tvorba rozdělení na TB se zastaví, zruší se nové značení a začíná znovu s číslem, jež měl obvodový prvek.

2. Patří-li brána registru, dále rozlišujeme, zda je tento registr označen jako hraniční nebo ne.

   (a) Není-li označen jako hraniční, je s ním postupováno stejně jako v bodu 1. S tím, že se zkoumá vstup i výstup.

   (b) Je-li označen jako hraniční, zkoumá se pouze výstup.

      i. Patří-li k nějakému jinému TB, dosavadní přiřazování prvků do nově vznikajícího TB se zastaví, značení se zruší a začíná se znovu s číslem, jež měla výstupní část registru.

      ii. Nepatří-li k nějakému TB, přiřadí se mu číslo právě vytvářeného TB a průchod i-cesty se ukončí (pokračuje se další i-cestou).

Je-li brána **výstupní**:
Zjistí se, zda je již přiřazena výstupní část registru k nějakému TB. Pokud ano, pokračuje se další branou. Pokud ne, zjišťuje se dále, zda je brána připojena na primární výstup (PO).

[10] jak již bylo zmíněno dříve, vstup registru může patřit do jiného TB než výstup v případě hraničního registru, proto se vstupní a výstupní část každého registru analyzují odděleně

Pokud ano, je označen speciální značkou a pokračuje se další bránou. Pokud není připojen na PO, začne se procházet množina i-cest ρ, kde analyzovaná brána je začátkem i-cesty. I-cesta se prochází od začátku (od analyzované brány) a pro každou bránu na této i-cestě se provádí tato analýza:

1. Nepatří-li brána registru, zjišťujeme, zda už nemá přiřazeno číslo TB, do kterého patří. Pokud nemá, přiřadí se mu nové číslo právě vytvářeného TB. Pokud již má číslo jiného TB, dosavadní tvorba nového TB se zastaví, zruší se značení a začíná se znovu s číslem, jež měl obvodový prvek.

2. Patří-li brána registru, dále rozlišujeme, zda je tento registr označen jako hraniční nebo ne.

   (a) Není-li označen jako hraniční, je s ním postupováno stejně jako v bodu 1. S tím, že se zkoumá vstup i výstup.

   (b) Je-li označen jako hraniční, zkoumá se pouze vstup.

      i. Patří-li k nějakému TB, dosavadní tvorba TB se zastaví, zruší a začíná znovu s číslem, jež měla vstupní část registru.

      ii. Nepatří-li k nějakému TB, přiřadí se mu číslo právě vytvářeného TB a průchod i-cesty se ukončí (pokračuje se další i-cestou).

Výpočet hodnoty fitness funkce:
Hodnota fitness funkce je složena z těchto částí:
1. část: určuje, kolik procent registrů je součástí nebo hraničním registrem některého TB.
2. část: stanovuje, kolik procent obvodových prvků a vnitřních registrů je součástí některého TB. Hodnota menší jak 100 procent indikuje, že některé prvky zůstaly mimo TB a nelze je do žádného TB zařadit.
3. část: udává, kolik procent registrů není zařazeno do řetězce scan.
4. část: určuje, kolik procent TB obsahuje alespoň jeden vnitřní prvek.
5. část: zohledňuje střední odchylku synchronizační vzdálenosti[11] jednotlivých TB.
6. část: je střední odchylka velikosti TB. Hodnota je největší když je velikost jednotlivých TB stejná.
Výsledná fitness funkce je pak rovna váženému součtu jednotlivých částí.

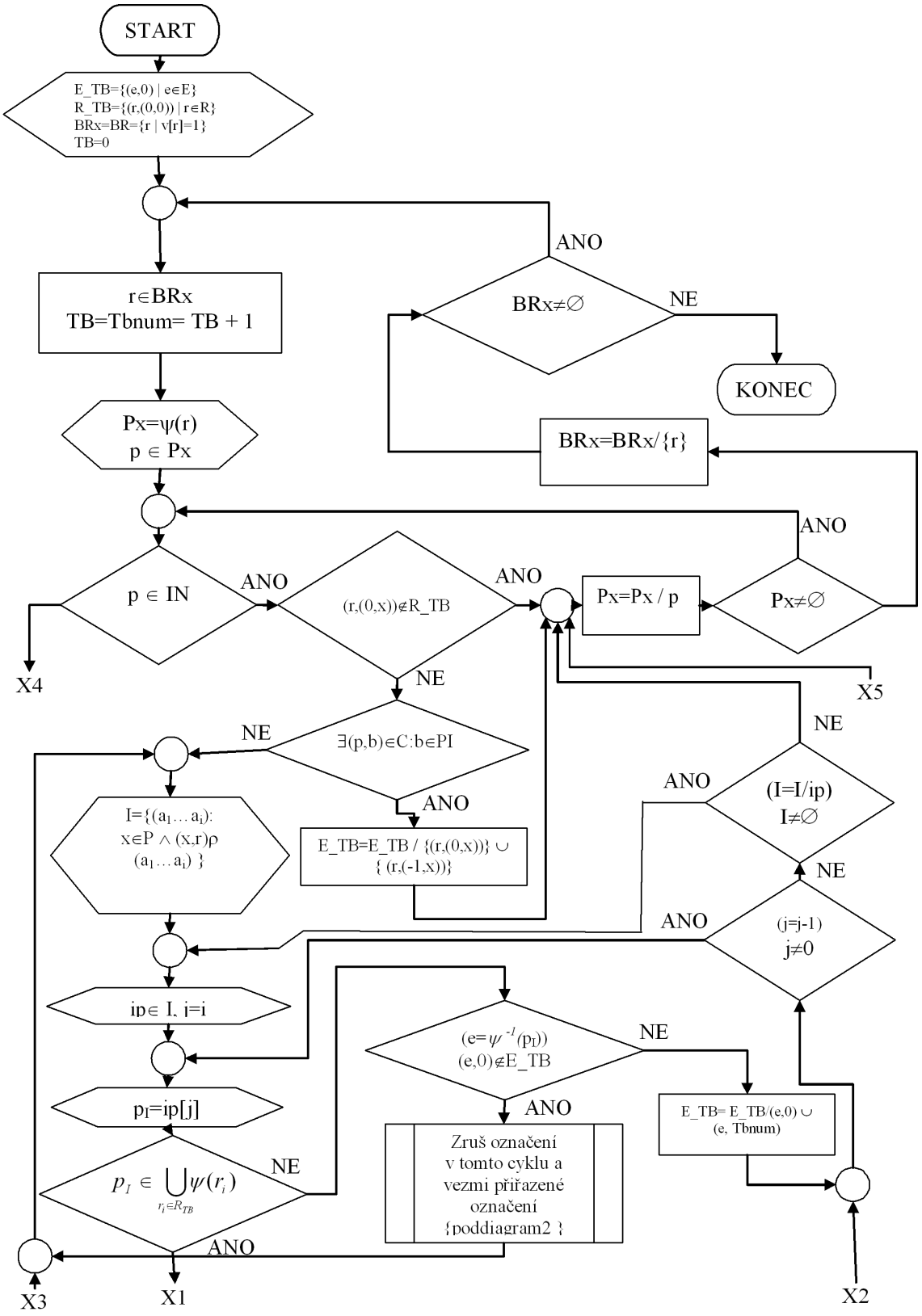
6.3.3 Implementace metody pro detekci smyček

V kapitole 6.2.3 byla zavedena metoda, jež umožňuje detekci a přerušování přímých a nepřímých smyček. Nyní bude popsána její implementace.

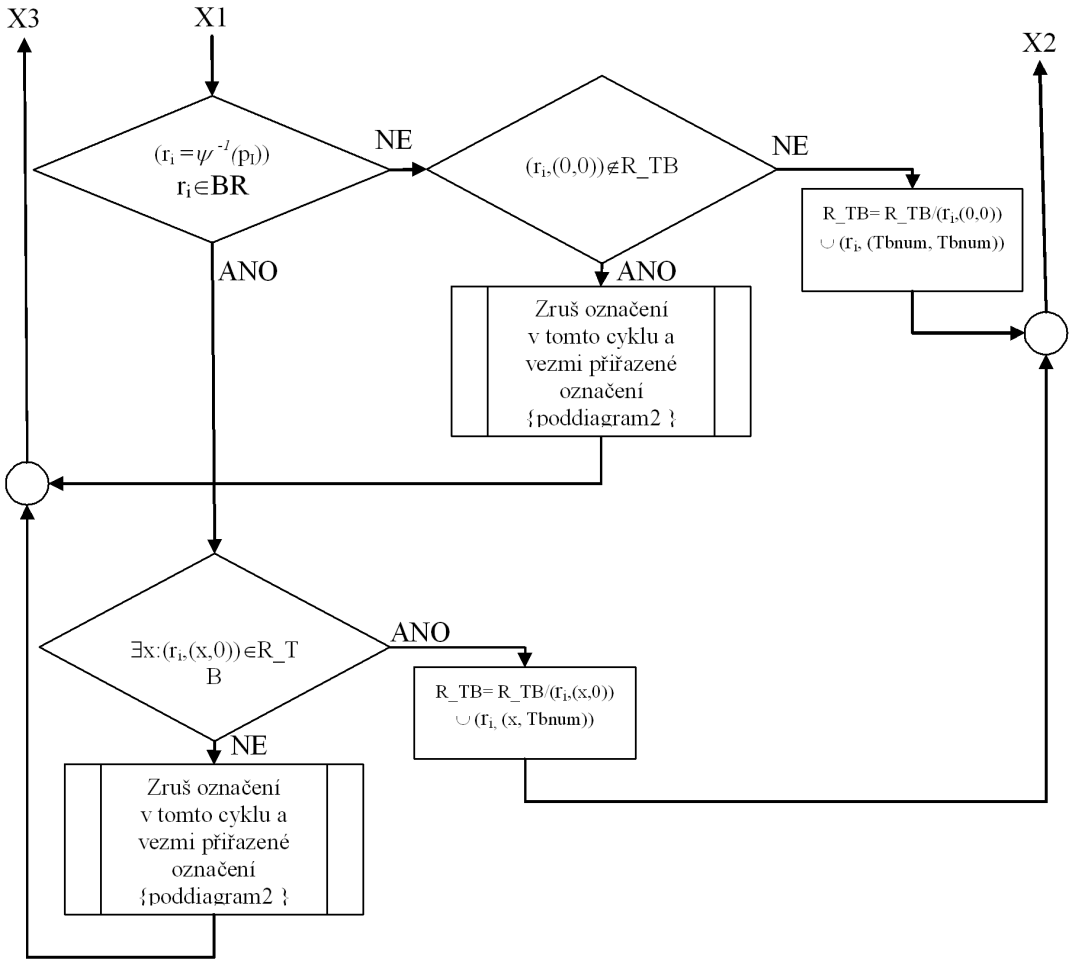
Detekce odstraňování smyček je realizována na formálním modelu. V první části programu jsou vyhledávány přímé smyčky. Procházejí se všechny registry v obvodu a pro každou bránu jednotlivých registrů se v množině ρ hledá i-cesta, jež v dané bráně začíná a končí ve stejném obvodovém prvku. Tato i-cesta se poté prochází, aby se zjistilo, zda obsahuje registr, jímž by tato smyčka mohla být přerušena. Pokud takový registr i-cesta neobsahuje, je tato i-cesta uložena pro pozdější doplnění DR[12].

Další částí je vyhledání nepřímých smyček. Z každého registru se zkoumají i-cesty, které v tomto registru začínají a zároveň nekončí ve stejné bráně, ze které vycházejí a nejsou

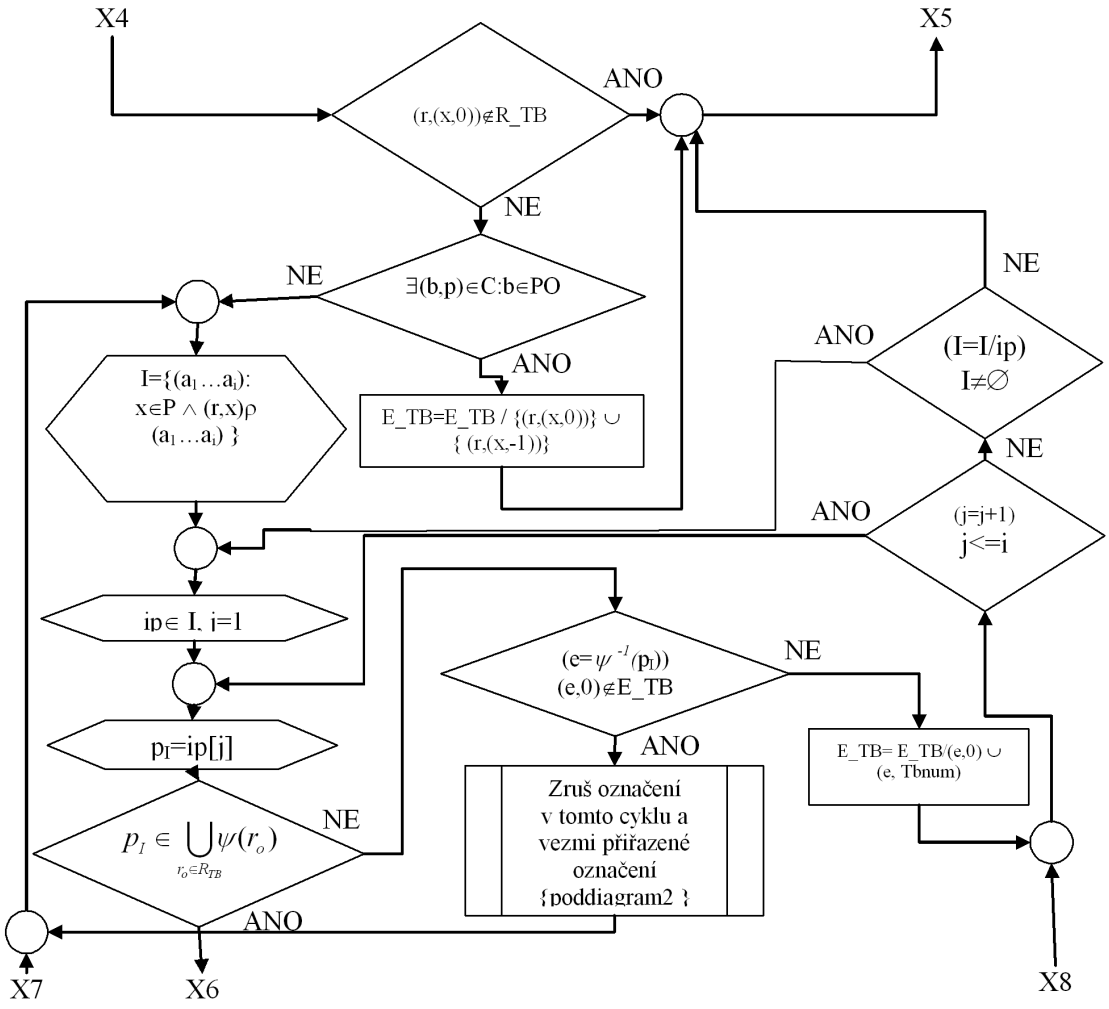
[11] synchronizační vzdálenost je počet hodinových pulzů nutných k průchodu testovacích vektorů TB
[12] Pojem Diagnostický Registr byl zaveden v kapitole 6.2.3.



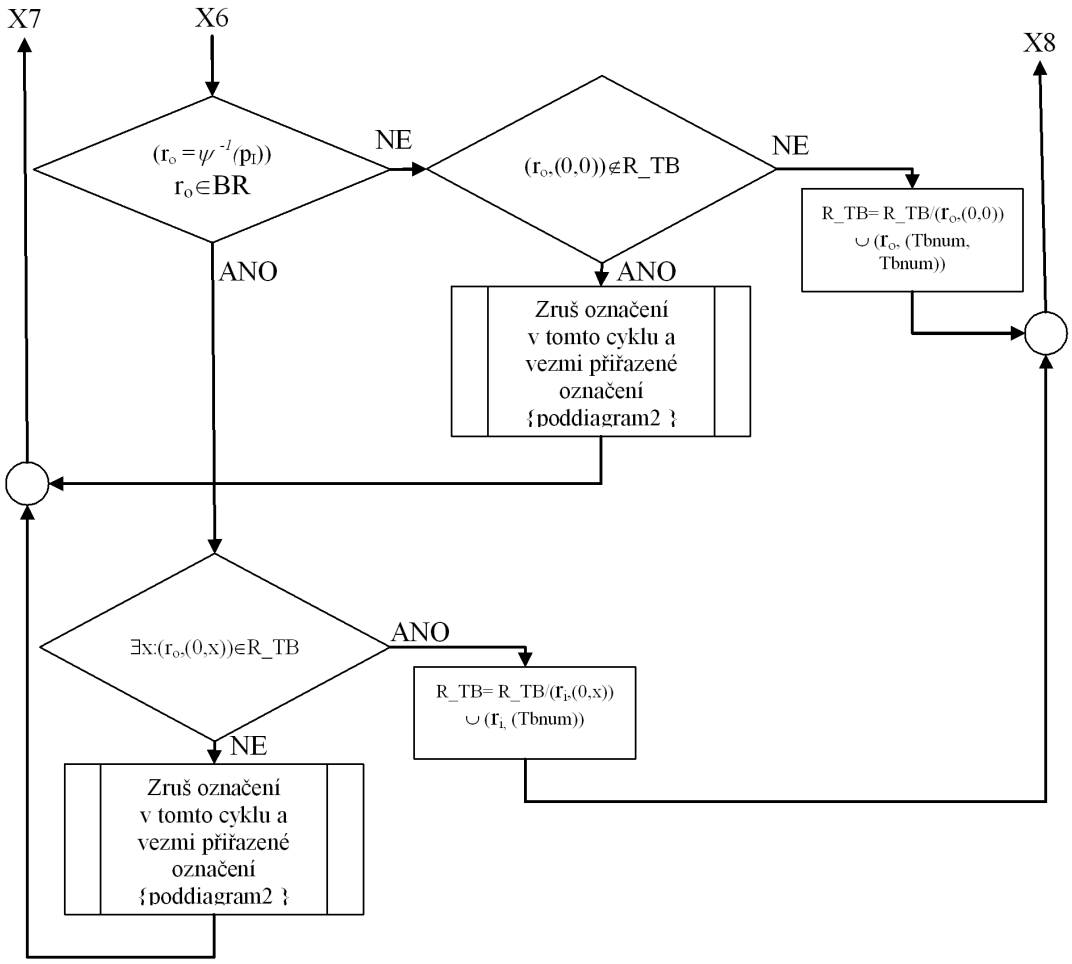
Obrázek 6.13: Vývojový diagram výpočtu R_TB, E_TB z bitového vektoru v



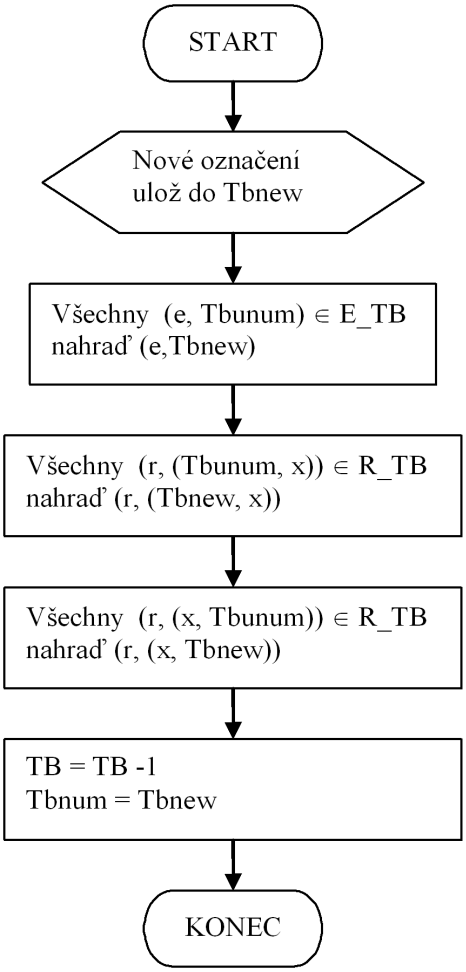
Obrázek 6.14: Vývojový diagram výpočtu R_TB, E_TB z bitového vektoru v pokračování



Obrázek 6.15: Vývojový diagram výpočtu R_TB, E_TB z bitového vektoru v pokračování



Obrázek 6.16: Vývojový diagram výpočtu R_TB, E_TB z bitového vektoru v pokračování



Obrázek 6.17: Poddiagram 2

to smyčky, které spojují přímo výstup se vstupem. I-cesta se prochází a pokud v ní není registr, zkoumáme, zdali končí v registru, který je právě zpracováván. Pokud ne, je i-cesta k prvku, ve kterém končí, uložena a prvek je uložen i s i-cestou ze zkoumaného registru k němu pro další rozšiřování i-cesty. Pokud končí v registru, který je právě zpracováván, ale v jiné bráně než začíná, je nalezena smyčka. Tato smyčka se uloží pro pozdější přerušení „diagnostickým registrem“. Podobný postup se provádí pro i-cesty, jež končí ve zkoumaném registru.

Z každého prvku obvodu, do kterého se lze dostat nějakou i-cestou ze zkoumaného registru, jež neobsahuje registry, se prochází všechny i-cesty a ty jsou zkoumány stejným způsobem, jak je uvedeno v předchozím odstavci.

Tím je identifikována množina všech smyček v obvodu včetně i-cesty, kterou smyčky procházejí. Každou z těchto smyček je třeba přerušit registrem DR. Je vybráno náhodné místo ve smyčce, kde bude umístěn DR. Je třeba aktualizovat všech 19 množin, což sestává z činností:

1. vytvoření nového obvodového prvku,
2. přerušení vybraného místa smyčky,
3. připojení brány nového prvku DR,
4. nastavení správné šířky spojů,
5. prodloužení i-cesty,
6. aktualizace množiny uzlů a bran.

Výsledný modifikovaný formální model se uloží na disk a je možné jej zpracovat programem pro rozdělení obvodů na TB. **Tento program má v sobě zabudován algoritmus, jež DR, které nevyužil jako hraniční, z obvodu reverzním postupem odstraní.**

6.3.4 Implementace metodiky pro rozdělení obvodu na TB

1. krok

Prvním krokem je analýza VHDL (případně verilogu) kódu (reprezentujícího analyzovaný obvod) a jeho převedení do formálního modelu, jež je popsán v kapitole 5. K tomuto účelu je využita aplikace, jež vznikla v rámci diplomové práce [31]. Vstupem je strukturální[13] VHDL kód popisující analyzovaný obvod na úrovni RT. Dále je třeba připojit soubor *.VHC, který slouží pro popis i-režimů prvků, které se nacházejí v obvodu. Ke každému prvku je zde určeno, jakým způsobem je možné jej uvést do transparentního režimu a mezi kterými branami je to možné. Nakonec je třeba specifikovat, které brány jsou řídicí, seznam těchto bran je uveden v souboru *.EXC.

Výstupem aplikace je soubor množin formálního modelu popsaného v kapitole 5. Součástí výstupu je i seznam všech i-cest a ii-cest[14] v obvodu, jež je vypočten pomocí modifikovaného Dijkstrova algoritmu.

Protože je aplikace časově náročná, bylo při implementaci přistoupeno k paralelizaci Dijkstrova algoritmu, jež je v ní implementován, aby bylo možné ji provozovat na víceprocesorových výpočetních strojích. K tomuto účelu bylo využito rozšíření jazyka `c++` s názvem `OpenMP`.

[13] Behaviorální VHDL je třeba předsyntetizovat pomocí některého nástroje a namapovat například na prvky z knihovny AMI.

[14] Rozšíření modelu

2. krok

Dalším krokem je rozdělení obvodu na TB. V tomto kroku je použit genetický algoritmus nebo simulované žíhání. Chromozom je tvořen posloupností 0 a 1 a jeho délka je rovna počtu registrů v obvodu. Každá pozice v chromozomu odpovídá jednomu registru. Hodnota „0“ znamená, že registr není hraniční (bude tedy vnitřním registrem TB a nebude zařazen do registru scan), hodnota „1“ znamená, že registr je hraničním registrem TB (bude zařazen do registru scan). V případě genetického algoritmu se pro výběr rodičů ke křížení používá rulety, jedinci s vyšší hodnotou fitness funkce zaujímají větší část rulety a tím je i vyšší pravděpodobnost jejich výběru. Po výběru prvního rodiče se tento z rulety vyřadí a vybere se druhý. S danou pravděpodobností se pak provádí jednobodové křížení vybraných rodičů. Dále se s jistou pravděpodobností provádí mutace určeného počtu bitů. Takto vzniklí jedinci jsou zařazeni do populace. Dva jedinci s nejnižší hodnotou fitness funkce jsou z populace vyřazeni. V případě simulovaného žíhání se osvědčila jednobitová mutace.

Nejvíce časově náročný je výpočet fitness funkce každého jedince populace. Součástí výpočtu je rozdělení obvodu na TB podle chromozomů s využitím formálního modelu TB. K tomu je použit algoritmus detailně popsaný v části zabývající se fitness funkcí. Tento algoritmus prochází obvod z každého registru označeného jako hraniční a označuje prvky po i-cestě číslem TB tak dlouho, až se dostane k dalšímu registru označenému jako hraniční. Po té pokračuje procházením obvodu a označuje prvky dalším číslem TB. Označení jednotlivých prvků obvodu se ukládá do množin E_TB a R_TB. Definice těchto množin je uvedena v kapitole 6.2.1.

3. krok

V této fázi je obvod rozdělen na TB a je možné vyčíslit hodnotu fitness funkce. Tento výpočet se provádí způsobem popsaným na konci kapitoly 6.3.2.

Genetický algoritmus tedy optimalizuje rozdělení na TB tak, aby byla hodnota fitness funkce co největší. Po ukončení genetického algoritmu je vybrán jedinec (tzn. rozdělení obvodu na TB) s největší fitness funkcí. V případě simulovaného žíhání je tento jedinec přímo výsledkem výpočtu. Jsou uloženy množiny E_TB a R_TB a přichází na řadu algoritmus, jež uloží rozdělený obvod tak, že každý TB je uložen v samostatném souboru. Tyto soubory jsou pak spojeny v celý obvod v hlavním souboru. Výstupním formátem je Verilog.

Algoritmus 6.3.1. Uložení rozděleného obvodu

- Vygeneruje se soubor, jež obsahuje seznam základních prvků obvodu.
- Pro každý TB se provádí tento postup:
  - Projdou se všechny uzly obvodu, zjistí se, zda je uzel v TB vůči zbytku obvodu vstupní, výstupní, vnitřní, vnější (postup dále).
  - Generuje se rozhraní, jež se určí na základě uzlů označených jako externí.
  - Určí a označí se signály jako vstupní a výstupní.
  - Pro vnitřní uzly se definují vodiče.
  - Naleznou se vnitřní registry TB a zapíše se zapojení jejich rozhraní.
  - Naleznou se vnitřní prvky TB a zapíše se zapojení jejich rozhraní.
- Vygeneruje se rozhraní (PO, PI) hlavního obvodu spojujícího jednotlivé TB do jednoho celku.

- Určí se vstupní a výstupní signály rozhraní a zapíší se do hlavního souboru.
- Vloží se jednotlivé TB a zapojí se jejich rozhraní (pomocné vodiče, které budou sloužit k propojení, se ukládají do pomocné množiny).
- Vkládají se prvky TB, které nejsou v žádném TB a zapíše se zapojení jejich rozhraní (pomocné vodiče se opět ukládají do pomocné množiny).
- Zapíší se hraniční a nezařazené prvky a uloží se zapojení jejich rozhraní (pomocné vodiče se ukládají do pomocné množiny).
- Na závěr se zapíše seznam pomocných vodičů, jež jsou uloženy v pomocné množině.

Výstupem metodiky je tedy obvod rozdělený na TB, popsaný ve Verilogu a množiny E_TB a R_TB, jež přiřazují jednotlivým prvkům obvodu čísla TB, do kterých patří.

V dalším kroku je možné jednoduše vygenerovat generátorem testu soubor testovacích vektorů pro každý TB zvlášť, protože každý TB je popsán v samostatném souboru.

Kapitola 7

Experimentální výsledky

V této kapitole je uvedena analýza experimentálních výsledků získaných použitím algoritmů prezentovaných v kapitole 6. Nejdříve jsou uvedeny výsledky testů, prováděných na testovací sadě obvodů vyvinuté na Ústavu počítačových systémů, Fakulty informačních technologií, VUT v Brně s použitím genetického algoritmu a simulovaného žíhání. Důvodem pro použití této sady byla neexistence vhodné testovací sady, jež by obsahovala obvody s větším počtem registrů. Dále je uvedeno porovnání těchto výsledků. Následuje popis experimentů získaných na dalších obvodech při aplikaci algoritmu pro odstranění smyček. V závěru jsou shrnuty výsledky experimentů.

7.1 Experimenty na testovacích obvodech

V tabulkách 7.1 až 7.12 jsou základní informace o každém konkrétním obvodě, doba potřebná pro převod do formálního modelu a doba potřebná k rozdělení na TB pomocí tří různých metod. Řádek 1 informuje o počtu obvodových prvků v obvodu. V tomto počtu jsou zahrnuty kombinační prvky, registry i multiplexory (tento údaj odpovídá velikosti množiny E). Údaj na řádku 2 udává celkový počet spojů v obvodu (množina C). Spoj je vždy veden pouze mezi dvěmi branami. Bránou se rozumí vstup/výstup obvodového prvku, řídicí vstup/výstup obvodového prvku nebo primární vstup/výstup obvodu. Počet registrů na řádku 3 odpovídá velikosti množiny R. Řádky 4 až 8 udávají velikost množin PI, PO, IN, OUT, CI. Na řádku 9 je uveden celkový počet bran, jež je tvořen součtem řádků 4 až 8. Pokrytí poruch uvedené na řádku 10 bylo určeno generátorem testu Flextest, generátor byl aplikován v základním nastavení na obvod bez použití řetězce scan. Řádek 11 udává celkový počet i-cest nalezených v obvodu, tzn. dvojic bran, mezi kterými existuje i-cesta. Na řádku 12 je velikost množiny ρ, jež přiřazuje každé dvojici bran, mezi kterými existuje i-cesta, posloupnost bran, jimiž tato i-cesta prochází. Toto číslo je větší než počet i-cest, protože mezi dvěmi branami může existovat více různých i-cest. Všechny časy uvedené na řádcích 13, 15, 16, 20, 21 a 25 jsou průměrné hodnoty z 10 opakovaných měření. Řádek 18, 23 a 27 udává počet registrů, jež byly označeny jako hraniční. Některé registry jsou však dosažitelné přímo z PO/PI a není třeba je zařazovat do registru scan. Počet skutečně zařazených registrů do řetězce scan je pak uveden na řádku 17, 22 a 26. Počet TB, na něž byl obvod rozdělen, je uveden na řádku 19, 24 a 28. Údaj udávající počet registrů zařazených do řetězce scan programem DFTAdvizor firmy Mentor Graphics® (řádek 14) je rozdělen na dvě části. První číslo je výsledkem metody „ATPG-Based Partial Scan[1]“, druhé číslo je výsledek metody „AUTOMATIC-Based

[1]Tato technika využívá sekvenční generátor testu pro navržení registrů do řetězce scan.

Partial Scan[2]“. Všechny testovací obvody uvedené v tabulkách 7.1 až 7.12 byly rozděleny úplně (tzn., že každý obvodový prvek patří do nějakého TB), předmětem dalších experimentů jsou obvody, u nichž tohoto výsledku nebylo dosaženo. Číslo zapsané za názvem jednotlivých metod udává maximální nalezenou hodnotu fitness funkce.

Metoda **přímého rozdělení** nevyužívá evoluce, jedná se pouze o jednu aplikaci funkce pro rozdělení obvodu na TB se vstupním vektorem, ve kterém jsou všechny registry označeny jako hraniční. Tato metoda označí všechny registry, jež lze použít jako hraniční a rozdělí tak obvod na maximální možný počet TB.

Základní údaje o obvodu:

1	počet obvodových prvků	146	8	počet řídicích bran	75
2	počet spojů	6131	9	počet všech bran	496
3	počet registrů	75	10	pokrytí poruch (bez scanu)	66%
4	počet primárních vstupů	36	11	počet i-cest	7688
5	počet primárních výstupů	10	12	velikost množiny ρ	8452
6	počet vstupních bran	225	13	doba převodu do modelu	6m 27s
7	počet výstupních bran	150	14	počet reg., jež navrhuje Mentor	44; 9

Genetický algoritmus: 29.4715

15	prům. doba výp. rozdělení na TB	13,2s
16	doba výpočtu jedné generace	36ms
17	počet registrů ve scanu	30
18	počet hraničních registrů	68
19	počet TB	9

Simulované žíhání: 29.447

20	prům. doba výp. rozdělení na TB	15,2s
21	doba výpočtu jedné generace	35ms
22	počet registrů ve scanu	8
23	počet hraničních registrů	33
24	počet TB	3

Metoda přímého rozdělení: 29.4639

25	prům. doba výp. rozdělení na TB	2,4s
26	počet registrů ve scanu	30
27	počet hraničních registrů	70
28	počet TB	10

Tabulka 7.1: Výsledky experimentů s obvodem „m10“

7.1.1 Ověření škálovatelnosti problému

Moderní výpočetní systémy (a dnes i domácí počítače) mají několik jader/procesorů, je tedy výhodné využít pro výpočet celý výpočetní výkon počítače a urychlit tak celý výpočet algoritmu u rozsáhlejších obvodů. Rozhodli jsme se ověřit škálovatelnost algoritmu pro převod obvodu do formálního modelu.

[2]Tato metoda automaticky vybere nejvhodnější metodu z ATPG-Based Partial Scan (založená na generátoru testu), SCOAP-based [19] a STRUCTURE-based (založená na analýze struktury obvodu)

Doba převodu obvodu z VHDL kódu do formálního modelu byla měřena na osmiprocesorovém počítači s procesorem Intel® Xeon® 2,66GHz. U obvodu „m10“ je doba potřebná k převodu obvodu do množinového modelu a k nalezení všech i-cest 6 minut a 27 sekund, tohoto času bylo dosaženo paralelním zpracováním algoritmu[3] na více procesorech současně. Pokud použijeme neparalelní verzi algoritmu, trvá převod na počítači s procesorem Intel® Pentium® 4 3,4GHz 51 minut a 17 sekund. Rozdělením úlohy na 8 procesorů se tak dosáhlo 7,9 násobné zrychlení výpočtu[4], režie paralelního výpočtu tedy tvoří pouze 1,25%[5]. To svědčí o dobré škálovatelnosti tohoto algoritmu.

7.1.2 Experimentální výsledky

Obvod, jež je analyzován v tabulce 7.1, byl pomocí GA rozdělen na 9 TB, z celkového počtu 75 registrů bylo 68 registrů označeno jako hraniční. Přístupných přes primární vstupy/výstupy je 38 registrů, proto je třeba do řetězce scan zařadit pouze 30[6] registrů. Komerční návrhový systém doporučuje 44 resp. 9 registrů dle typu analýzy. Pokrytí poruch určené generátorem testu Flextest bez použití navržené metodiky je 66%. Použití navržené metodiky pak zvýší toto pokrytí až na 100%.

[3] viz. kapitola 6.3.4 – 1. krok

[4] čas při 8 procesorech / čas na jednom procesoru = $51,283min/6,45min = 7,9$

[5] (1 - (zrychlení / počet CPU)) * 100 = $(1 - (7,9/8)) * 100 = 1,25\%$

[6] 68 hraničních registrů - 38 registrů přístupných přes PI/PO = 30 registrů

Základní údaje o obvodu:

1	počet obvodových prvků	156	8	počet řídicích bran	85
2	počet spojů	6311	9	počet všech bran	591
3	počet registrů	85	10	pokrytí poruch (bez scanu)	0%
4	počet primárních vstupů	61	11	počet i-cest	9801
5	počet primárních výstupů	35	12	velikost množiny ρ	10671
6	počet vstupních bran	240	13	doba převodu do modelu	13m 25s
7	počet výstupních bran	170	14	počet reg., jež navrhuje Mentor	15; 9

Genetický algoritmus: 29.5156

15	prům. doba výp. rozdělení na TB	30,6s
16	doba výpočtu jedné generace	485ms
17	počet registrů ve scanu	0
18	počet hraničních registrů	55
19	počet TB	10

Simulované žíhání: 29.505

20	prům. doba výp. rozdělení na TB	29,375s
21	doba výpočtu jedné generace	67ms
22	počet registrů ve scanu	0
23	počet hraničních registrů	48
24	počet TB	10

Metoda přímého rozdělení: 20.4382

25	prům. doba výp. rozdělení na TB	3,703s
26	počet registrů ve scanu	15
27	počet hraničních registrů	85
28	počet TB	25

Tabulka 7.2: Výsledky experimentů s obvodem „m10b“

Základní údaje o obvodu:

1	počet obvodových prvků	123	8	počet řídicích bran	52
2	počet spojů	2450	9	počet všech bran	426
3	počet registrů	52	10	pokrytí poruch (bez scanu)	0%
4	počet primárních vstupů	28	11	počet i-cest	25705
5	počet primárních výstupů	2	12	velikost množiny ρ	68541
6	počet vstupních bran	207	13	doba převodu do modelu	4m 27s
7	počet výstupních bran	137	14	počet reg., jež navrhuje Mentor	36; 6

Genetický algoritmus: 29.1832

15	prům. doba výp. rozdělení na TB	4m 49,5s
16	doba výpočtu jedné generace	4,45s
17	počet registrů ve scanu	6
18	počet hraničních registrů	7
19	počet TB	4

Simulované žíhání: 29.5122

20	prům. doba výp. rozdělení na TB	4m 11s
21	doba výpočtu jedné generace	567ms
22	počet registrů ve scanu	8
23	počet hraničních registrů	9
24	počet TB	5

Metoda přímého rozdělení: 20.4214

25	prům. doba výp. rozdělení na TB	8,531s
26	počet registrů ve scanu	48
27	počet hraničních registrů	52
28	počet TB	25

Tabulka 7.3: Výsledky experimentů s obvodem „m10c“

Základní údaje o obvodu:

1	počet obvodových prvků	281	8	počet řídicích bran	140
2	počet spojů	19261	9	počet všech bran	941
3	počet registrů	140	10	pokrytí poruch (bez scanu)	68.88%
4	počet primárních vstupů	61	11	počet i-cest	32423
5	počet primárních výstupů	10	12	velikost množiny ρ	41681
6	počet vstupních bran	440	13	doba převodu do modelu	87m 34s
7	počet výstupních bran	290	14	počet reg., jež navrhuje Mentor	109; 18

Genetický algoritmus: 29.485

15	prům. doba výp. rozdělení na TB	3m 6,5s
16	doba výpočtu jedné generace	8,1s
17	počet registrů ve scanu	70
18	počet hraničních registrů	129
19	počet TB	20

Simulované žíhání: 29.3607

20	prům. doba výp. rozdělení na TB	4m 55s
21	doba výpočtu jedné generace	677ms
22	počet registrů ve scanu	36
23	počet hraničních registrů	68
24	počet TB	12

Metoda přímého rozdělení: 29.4821

25	prům. doba výp. rozdělení na TB	11,141s
26	počet registrů ve scanu	70
27	počet hraničních registrů	130
28	počet TB	20

Tabulka 7.4: Výsledky experimentů s obvodem „m20“

Základní údaje o obvodu:

1	počet obvodových prvků	426	8	počet řídicích bran	215
2	počet spojů	43591	9	počet všech bran	1436
3	počet registrů	215	10	pokrytí poruch (bez scanu)	74%
4	počet primárních vstupů	96	11	počet i-cest	63899
5	počet primárních výstupů	20	12	velikost množiny ρ	74575
6	počet vstupních bran	665	13	doba převodu do modelu	469m 5,48s
7	počet výstupních bran	450	14	počet reg., jež navrhuje Mentor	164; 30

Genetický algoritmus: 29,4916

15	prům. doba výp. rozdělení na TB	9m 28s
16	doba výpočtu jedné generace	19,58s
17	počet registrů ve scanu	100
18	počet hraničních registrů	197
19	počet TB	30

Simulované žíhání: 29,3961

20	prům. doba výp. rozdělení na TB	12m 22s
21	doba výpočtu jedné generace	1,7s
22	počet registrů ve scanu	46
23	počet hraničních registrů	102
24	počet TB	23

Metoda přímého rozdělení: 29,4885

25	prům. doba výp. rozdělení na TB	26,828s
26	počet registrů ve scanu	100
27	počet hraničních registrů	200
28	počet TB	30

Tabulka 7.5: Výsledky experimentů s obvodem „m30“

Základní údaje o obvodu:

1	počet obvodových prvků	59	8	počet řídicích bran	25
2	počet spojů	1106	9	počet všech bran	192
3	počet registrů	25	10	pokrytí poruch (bez scanu)	84%
4	počet primárních vstupů	12	11	počet i-cest	4687
5	počet primárních výstupů	4	12	velikost množiny ρ	7350
6	počet vstupních bran	93	13	doba převodu do modelu	3,88s
7	počet výstupních bran	58	14	počet reg., jež navrhuje Mentor	14; 1

Genetický algoritmus: 29,56

15	prům. doba výp. rozdělení na TB	5m 312s
16	doba výpočtu jedné generace	98ms
17	počet registrů ve scanu	2
18	počet hraničních registrů	10
19	počet TB	2

Simulované žíhání: 29,556

20	prům. doba výp. rozdělení na TB	5,3s
21	doba výpočtu jedné generace	12ms
22	počet registrů ve scanu	2
23	počet hraničních registrů	10
24	počet TB	2

Metoda přímého rozdělení: 29,0433

25	prům. doba výp. rozdělení na TB	672ms
26	počet registrů ve scanu	5
27	počet hraničních registrů	20
28	počet TB	4

Tabulka 7.6: Výsledky experimentů s obvodem „kom5“

Základní údaje o obvodu:

1	počet obvodových prvků	114	8	počet řídicích bran	47
2	počet spojů	3140	9	počet všech bran	368
3	počet registrů	47	10	pokrytí poruch (bez scanu)	84,24%
4	počet primárních vstupů	20	11	počet i-cest	12008
5	počet primárních výstupů	5	12	velikost množiny ρ	21664
6	počet vstupních bran	183	13	doba převodu do modelu	1m 54s
7	počet výstupních bran	117	14	počet reg., jež navrhuje Mentor	28; 2

Genetický algoritmus: 29,2086

15	prům. doba výp. rozdělení na TB	24,3s
16	doba výpočtu jedné generace	1,21s
17	počet registrů ve scanu	13
18	počet hraničních registrů	29
19	počet TB	7

Simulované žíhání: 29,527

20	prům. doba výp. rozdělení na TB	46,2s
21	doba výpočtu jedné generace	106ms
22	počet registrů ve scanu	4
23	počet hraničních registrů	18
24	počet TB	2

Metoda přímého rozdělení: 29,1954

25	prům. doba výp. rozdělení na TB	2,5s
26	počet registrů ve scanu	13
27	počet hraničních registrů	37
28	počet TB	7

Tabulka 7.7: Výsledky experimentů s obvodem „kom10“

Základní údaje o obvodu:

1	počet obvodových prvků	114	8	počet řídicích bran	47
2	počet spojů	3140	9	počet všech bran	368
3	počet registrů	47	10	pokrytí poruch (bez scanu)	84,24%
4	počet primárních vstupů	20	11	počet i-cest	12008
5	počet primárních výstupů	5	12	velikost množiny ρ	21664
6	počet vstupních bran	183	13	doba převodu do modelu	1m 54s
7	počet výstupních bran	117	14	počet reg., jež navrhuje Mentor	28; 2

Genetický algoritmus: 29,4289

15	prům. doba výp. rozdělení na TB	5m 8s
16	doba výpočtu jedné generace	6,55s
17	počet registrů ve scanu	31
18	počet hraničních registrů	68
19	počet TB	16

Simulované žíhání: 29,4463

20	prům. doba výp. rozdělení na TB	4m 17s
21	doba výpočtu jedné generace	590ms
22	počet registrů ve scanu	22
23	počet hraničních registrů	42
24	počet TB	11

Metoda přímého rozdělení: 29,4182

25	prům. doba výp. rozdělení na TB	8,609s
26	počet registrů ve scanu	31
27	počet hraničních registrů	79
28	počet TB	16

Tabulka 7.8: Výsledky experimentů s obvodem „kom20“

Základní údaje o obvodu:

1	počet obvodových prvků	108	8	počet řídicích bran	65
2	počet spojů	3062	9	počet všech bran	411
3	počet registrů	56	10	pokrytí poruch (bez scanu)	74,63%
4	počet primárních vstupů	33	11	počet i-cest	6403
5	počet primárních výstupů	10	12	velikost množiny ρ	8090
6	počet vstupních bran	176	13	doba převodu do modelu	2m 57s
7	počet výstupních bran	127	14	počet reg., jež navrhuje Mentor	33; 8

Genetický algoritmus: 29,4479

15	prům. doba výp. rozdělení na TB	13,6s
16	doba výpočtu jedné generace	297ms
17	počet registrů ve scanu	17
18	počet hraničních registrů	30
19	počet TB	5

Simulované žíhání: 29,4818

20	prům. doba výp. rozdělení na TB	14s
21	doba výpočtu jedné generace	33ms
22	počet registrů ve scanu	17
23	počet hraničních registrů	29
24	počet TB	5

Metoda přímého rozdělení: 25,1141

25	prům. doba výp. rozdělení na TB	1,937s
26	počet registrů ve scanu	29
27	počet hraničních registrů	49
28	počet TB	14

Tabulka 7.9: Výsledky experimentů s obvodem „3m10“

Základní údaje o obvodu:

1	počet obvodových prvků	103	8	počet řídicích bran	60
2	počet spojů	2587	9	počet všech bran	386
3	počet registrů	51	10	pokrytí poruch (bez scanu)	0%
4	počet primárních vstupů	28	11	počet i-cest	5290
5	počet primárních výstupů	5	12	velikost množiny ρ	6886
6	počet vstupních bran	171	13	doba převodu do modelu	2m 13s
7	počet výstupních bran	122	14	počet reg., jež navrhuje Mentor	39; 11

Genetický algoritmus: 29,4157

15	prům. doba výp. rozdělení na TB	6,2s
16	doba výpočtu jedné generace	220ms
17	počet registrů ve scanu	3
18	počet hraničních registrů	8
19	počet TB	2

Simulované žíhání: 29,5569

20	prům. doba výp. rozdělení na TB	9,7s
21	doba výpočtu jedné generace	22ms
22	počet registrů ve scanu	3
23	počet hraničních registrů	12
24	počet TB	2

Metoda přímého rozdělení: 25,1084

25	prům. doba výp. rozdělení na TB	1,516s
26	počet registrů ve scanu	34
27	počet hraničních registrů	44
28	počet TB	14

Tabulka 7.10: Výsledky experimentů s obvodem „3m10b“

Základní údaje o obvodu:

1	počet obvodových prvků	213	8	počet řídicích bran	126
2	počet spojů	9277	9	počet všech bran	808
3	počet registrů	105	10	pokrytí poruch (bez scanu)	76,8%
4	počet primárních vstupů	62	11	počet i-cest	13294
5	počet primárních výstupů	15	12	velikost množiny ρ	16141
6	počet vstupních bran	352	13	doba převodu do modelu	45m 29s
7	počet výstupních bran	353	14	počet reg., jež navrhuje Mentor	74; 17

Genetický algoritmus: 29,4367

15	prům. doba výp. rozdělení na TB	1m 37,5s
16	doba výpočtu jedné generace	220ms
17	počet registrů ve scanu	45
18	počet hraničních registrů	56
19	počet TB	20

Simulované žíhání: 29,4253

20	prům. doba výp. rozdělení na TB	46,25s
21	doba výpočtu jedné generace	826ms
22	počet registrů ve scanu	34
23	počet hraničních registrů	42
24	počet TB	20

Metoda přímého rozdělení: 25,5619

25	prům. doba výp. rozdělení na TB	6,828s
26	počet registrů ve scanu	64
27	počet hraničních registrů	92
28	počet TB	27

Tabulka 7.11: Výsledky experimentů s obvodem „3m20“

Základní údaje o obvodu:

1	počet obvodových prvků	7829	8	počet řídicích bran	183
2	počet spojů	18140	9	počet všech bran	1176
3	počet registrů	153	10	pokrytí poruch (bez scanu)	54,8%
4	počet primárních vstupů	84	11	počet i-cest	24895
5	počet primárních výstupů	13	12	velikost množiny ρ	29969
6	počet vstupních bran	523	13	doba převodu do modelu	205m 7s
7	počet výstupních bran	373	14	počet reg., jež navrhuje Mentor	120; 25

Genetický algoritmus: 29,4383

15	prům. doba výp. rozdělení na TB	2m 43s
16	doba výpočtu jedné generace	5,6s
17	počet registrů ve scanu	62
18	počet hraničních registrů	83
19	počet TB	29

Simulované žíhání: 29,41

20	prům. doba výp. rozdělení na TB	3m 49s
21	doba výpočtu jedné generace	524ms
22	počet registrů ve scanu	27
23	počet hraničních registrů	45
24	počet TB	27

Metoda přímého rozdělení: 25,7175

25	prům. doba výp. rozdělení na TB	15,47s
26	počet registrů ve scanu	107
27	počet hraničních registrů	133
28	počet TB	40

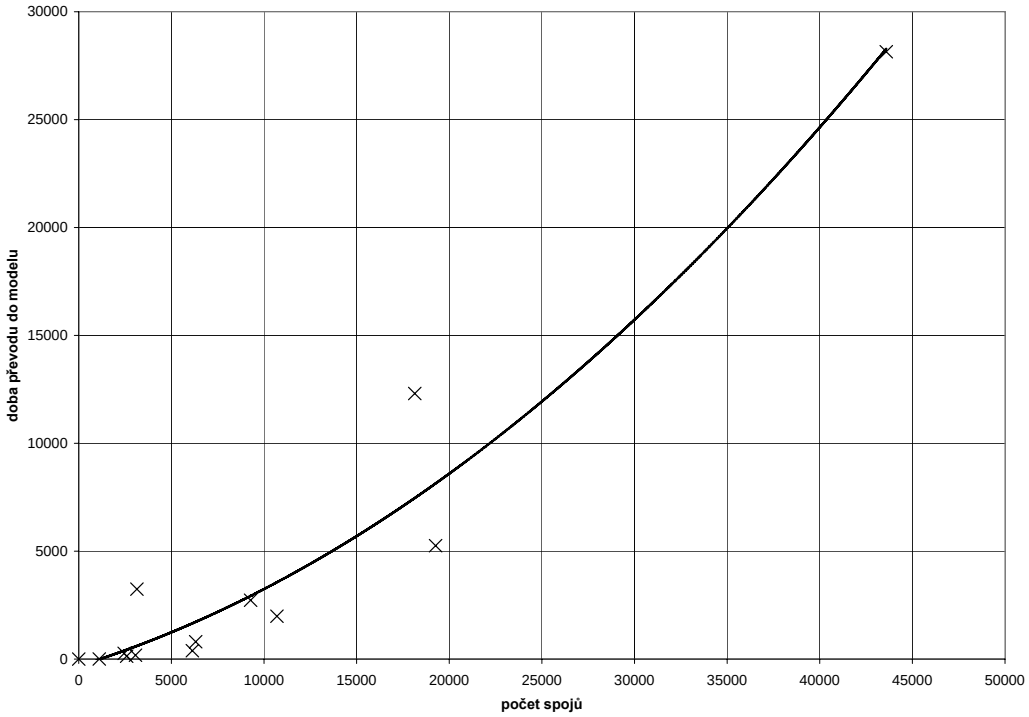
Tabulka 7.12: Výsledky experimentů s obvodem „3m30“

U obvodů zkoumaných v tabulkách 7.2 až 7.12 je vždy počet registrů, jež musí být zařazeny do řetězce scan, nižší než navrhuje komerční systém při analýze „ATPG-based“ a větší než v případě automatické volby analýzy systémem. Větší počet registrů, jež byl určen pomocí navržené metodiky, je způsoben nutností oddělit každý TB od dalších TB hraničními registry. Výše představené obvody jsou vhodné pro použití metodiky pro rozdělení obvodu na TB, protože mají velký počet registrů a pravidelnou strukturu s malým počtem zpětných vazeb, které by mohly způsobovat problémy popsané v kapitole 6.2.3. U každého jednotlivého TB nalezeného v obvodech 7.1 až 7.12 se dosahuje pokrytí poruch až 100%.

V grafu na obrázku 7.1 je zřejmá polynomiální závislost doby potřebné pro určení obsahu množin modelu ze zdrojového VHDL souboru. Tento algoritmus představuje nejpomalejší část celého procesu. Doba výpočtu rychle narůstá se vzrůstající velikostí obvodu (počtu spojů). Vzhledem k časové náročnosti je pak přijatelná velikost obvodu maximálně v desítkách tisíc spojů i přes paralelizaci algoritmu (viz kapitola 6.3.4). Jako další směr výzkumu by bylo vhodné nalézt rychlejší algoritmus.

Pro úplnost je na obrázku 7.2 uvedena doba převodu do formální reprezentace v závislosti na počtu i-cest. V tomto případě nelze určit typ závislosti, protože počet i-cest je silně závislý na struktuře obvodu.

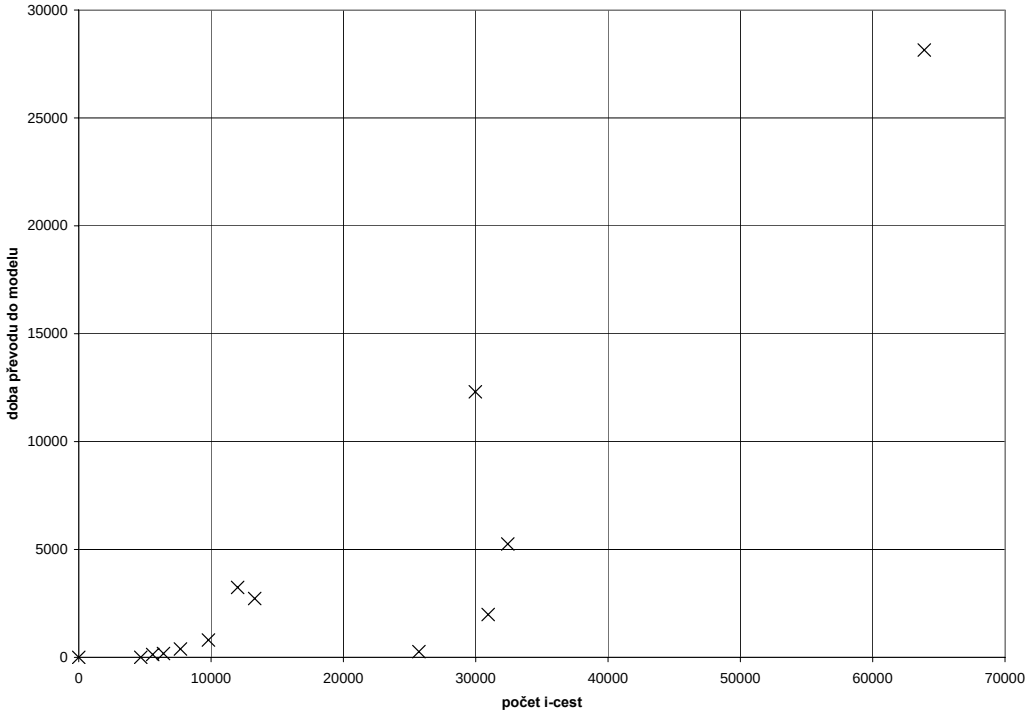
Čas potřebný pro výpočet fitness funkce jedné generace je polynomiálně závislý na počtu



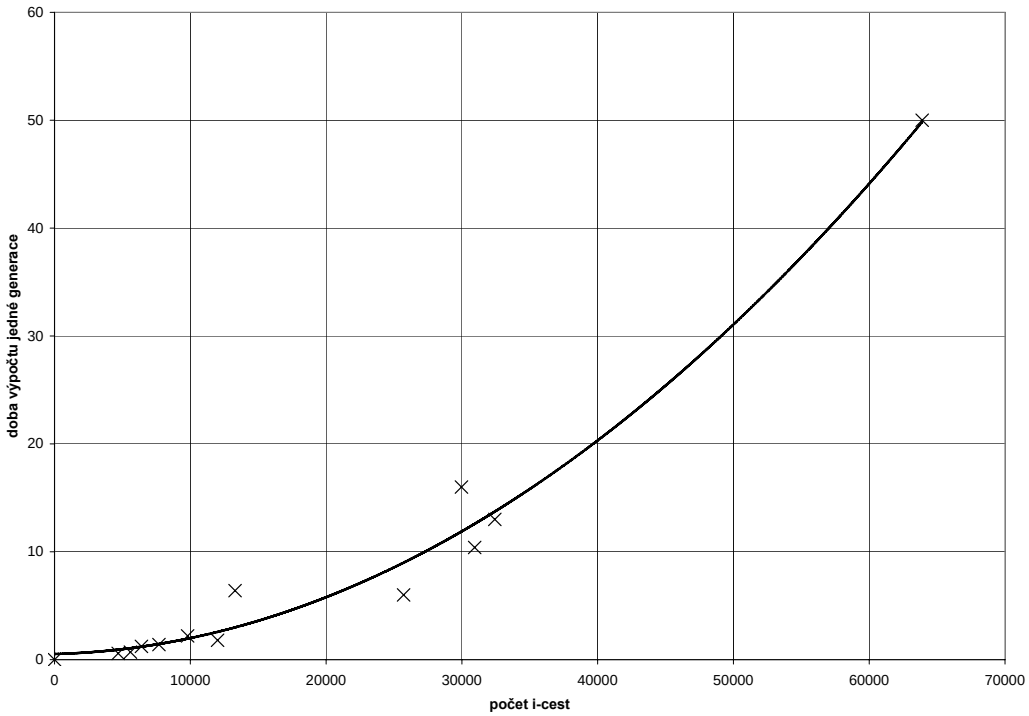
Obrázek 7.1: Čas potřebný pro převod do modelu [s] v závislosti na počtu spojů

i-cest, což je zřejmé z grafu na obrázku 7.3. Tato závislost je způsobena tím, že je algoritmus založen na procházení i-cest ze všech registrů. Stejnou závislost má potom i celková doba výpočtu rozdělení na TB, což je patrné na obrázku. 7.4.

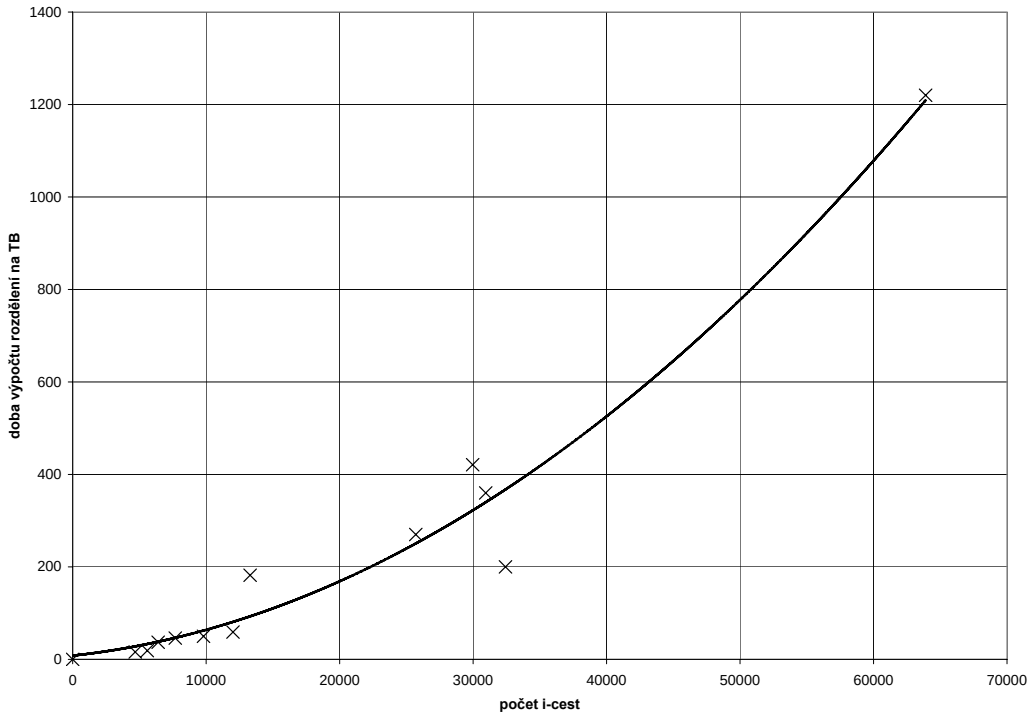
Pro úplnost jsou uvedeny i závislosti doby výpočtu jedné generace a doby výpočtu rozdělení na TB v závislosti na počtu registrů, viz. obrázek 7.5 a 7.6. Do grafů 7.1 až 7.6 byly vyneseny hodnoty z tabulek 7.1 až 7.12.



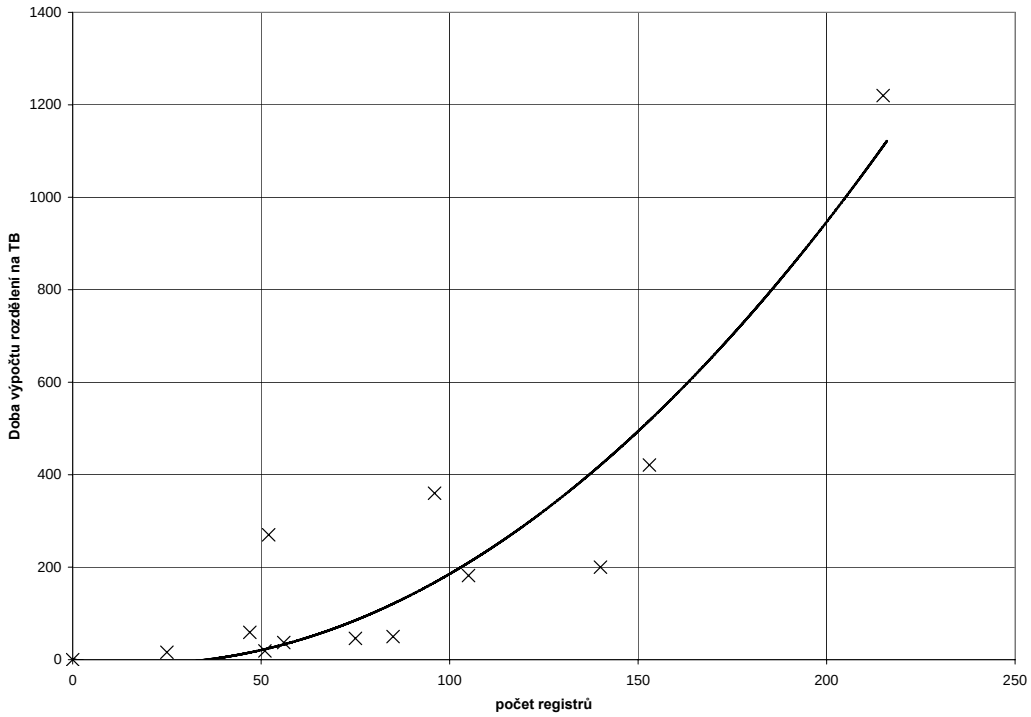
Obrázek 7.2: Čas potřebný pro převod do modelu [s] v závislosti na počtu i-cest



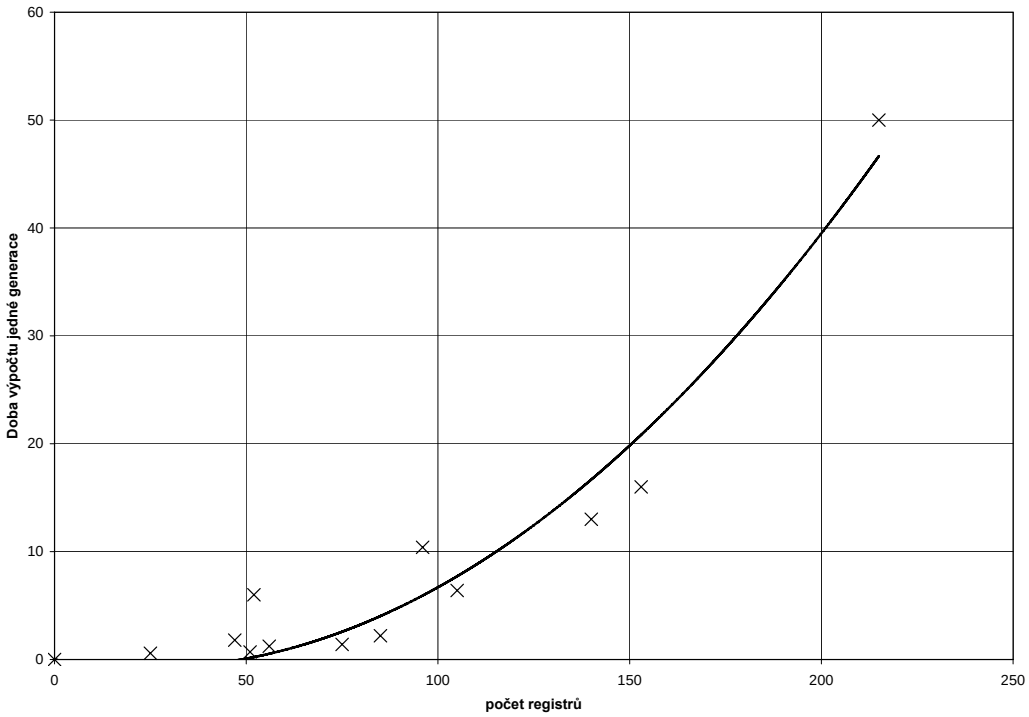
Obrázek 7.3: Čas potřebný pro výpočet jedné generace [s] v závislosti na počtu i-cest



Obrázek 7.4: Čas potřebný pro výpočet rozdělení na TB [s] v závislosti na počtu i-cest



Obrázek 7.5: Čas potřebný pro výpočet rozdělení na TB [s] v závislosti na počtu registrů



Obrázek 7.6: Čas potřebný pro výpočet jedné generace [s] v závislosti na počtu registrů

Označení	Počet funkčních jednotek [1]	Počet registrů [1]	Počet i-cest [1]	Počet spojů [1]	Doba potř. pro identifikaci i-cest [s]	Doba rozdělení na TB [s]	Počet TB [1]	Prvky nezařazené do TB [%]
COM	45	21	1874	1217	242	3.2	5	17
ISA	75	29	8831	2988	2375	15.2	2	4
DIFFEQ	11	6	2041	213	2.2	0.81	1	0
DEC	29	7	1283	529	22.8	4.3	2	20.6
FIFO2	226	144	33339	11297	41200	233	6	14
S298	47	19	10204	1367	349	14.7	2	3

Tabulka 7.13: Výsledky experimentů s dalšími obvody

Označení. [-]	Počet hraničních registrů [1]	Počet registrů uvnitř TB [1]	Počet funkčních jednotek [1]	Počet uzlů [1]
TB1	2	4	12	15
TB2	4	1	6	3
TB3	2	4	12	15
TB4	2	4	12	15
TB5	2	4	12	15

Tabulka 7.14: Detailní rozbor jednotlivých identifikovaných TB v obvodu COM

Označení. [-]	Počet hraničních registrů [1]	Počet registrů uvnitř TB [1]	Počet funkčních jednotek [1]	Počet uzlů [1]
TB1	28	86	222	241
TB2	3	3	8	7
TB3	3	3	8	7
TB4	3	3	8	7
TB5	5	1	6	3
TB6	5	1	6	3

Tabulka 7.15: Detailní rozbor jednotlivých identifikovaných TB v obvodu FIFO2

V tabulce 7.13 vidíme parametry dalších vybraných testovaných obvodů, jež nelze úplně rozdělit na TB. V prvních pěti sloupcích jsou uvedeny základní údaje o obvodu. V šestém sloupci je uvedena doba potřebná k identifikaci i-cest a v sedmém sloupci doba potřebná k rozdělení obvodu na TB pomocí genetického algoritmu. V dalších sloupcích je uveden počet TB, na něž byl obvod rozdělen, v posledním sloupci je část obvodu, kterou nelze rozdělit na TB. Je patrné, že obvod DIFFEQ, který je plně testovatelný, byl označen za jeden TB.

Některé obvody, jež mají pravidelnou strukturu, mohou být rozděleny tak, že některé TB jsou svým zapojením totožné. To vypovídá o tom, že pravidla pro rozdělení analyzovaného obvodu na TB byla definována jednoznačně a byla správně implementována. V tabulce 7.14 je vidět, že TB3, TB4 a TB5 mají i stejné parametry. V tabulce 7.15 mají stejné parametry TB2, TB3 a TB4.

7.2 Úplné prohledání prostoru

Nyní budou prohledávací algoritmy porovnány s úplným prohledáním stavového prostoru. Z důvodu velké časové náročnosti tohoto prohledávání byly pro srovnání zvoleny menší obvody. Čas potřebný k plnému prohledání exponenciálně roste s počtem registrů.

V tabulkách 7.16 až 7.19 jsou porovnány časy výpočtu a maximální nalezení hodnoty fitness funkce různých metod s metodou úplného prohledání stavového prostoru. Z tohoto porovnání vychází genetický algoritmus jako nejvhodnější metoda, protože ve všech případech nalezla nejlepší rozdělení v dobrém čase.

Metoda	**Doba výpočtu**	**Počet TB**	**Max. fitness**	**Reg. ve scanu**
Úplné prohledání	13m 2,5s	2 TB	29.4786	1
Metoda přímého rozdělení	0,2s	4 TB	21.5	10
Genetický algoritmus	1s	2 TB	29.4786	1
Simulované žíhání	1,2s	2 TB	29.3744	2

Tabulka 7.16: Porovnání úplného prohledávání s ostatními metodami - testovací obvod (18 registrů)

Metoda	**Doba výpočtu**	**Počet TB**	**Max. fitness**	**Reg. ve scanu**
Úplné prohledání	0,16s	1 TB	26.08	0
Metoda přímého rozdělení	0,1s	1 TB	26	0
Genetický algoritmus	0.14s	1 TB	26.08	0
Simulované žíhání	0,23s	1 TB	26.08	0

Tabulka 7.17: Porovnání úplného prohledávání s ostatními metodami - tseng (5 registrů)

Metoda	**Doba výpočtu**	**Počet TB**	**Max. fitness**	**Reg. ve scanu**
Úplné prohledání	0,19s	2 TB	26.4071	3
Metoda přímého rozdělení	0,15s	2 TB	26.3571	3
Genetický algoritmus	0.23s	2 TB	26.4071	3
Simulované žíhání	0,407s	2 TB	26.4071	3

Tabulka 7.18: Porovnání úplného prohledávání s ostatními metodami - diffeq (6 registrů)

7.3 Detekce a eliminace smyček

V této části je experimentálně ověřen pozitivní vliv detekce a přerušení smyček na schopnost metodiky rozdělit obvod na TB.

V tabulce 7.20 jsou uvedeny výsledky aplikace algoritmu pro detekci smyček na stejné obvody jako v tabulce 7.13. V prvním sloupci je uvedena doba nutná k detekci smyček a k přidání pomocných registrů. V dalším sloupci je uvedena doba rozdělení na TB, je patrné, že se tato doba oproti tabulce 7.13 zvýšila, což je způsobeno větším počtem registrů. Ve všech případech došlo ke snížení počtu obvodových prvků, jež nelze zařadit do žádného TB. Struktura některých obvodů neumožňuje zařadit všechny prvky do nějakého TB. To může být způsobeno např. přímým napojením částí obvodu na primární vstup nebo výstup bez oddělení registrem.

Metoda	Doba výpočtu	Počet TB	Max. fitness	Reg. ve scanu
Úplné prohledání	40,3s	2 TB	26.5571	3
Metoda přímého rozdělení	0,55s	5 TB	26	4
Genetický algoritmus	1,5s	2 TB	26.5571:	3
Simulované žíhání	1,7s	2 TB	26.55	3

Tabulka 7.19: Porovnání úplného prohledávání s ostatními metodami - COM s eliminací smyček (14 registrů)

Ozn. [-]	Hledání smyček [s]	Rozdělení na TB [s]	Počet TB [1]	Počet přidaných registrů [1]	Počet použitých registrů [1]	Prvky nezařazené do TB [%]
COM	4	13	6	17	2	6,7
ISA	59,7	201	2	33	1	2,5
DEC	1,2	2.3	2	7	1	13
FIFO2	2758	1740	18	58	50	10
S298	61	210	1	41	1	0

Tabulka 7.20: Výsledky metody pro detekci smyček

7.4 Generování testu

V této kapitole je porovnána náročnost generování testu pro obvody rozdělené na TB a pro obvody, u nichž byl navrhnut řetězec scan pomocí komerčního systému.

Tabulka 7.21 slouží k porovnání doby potřebné k vygenerování testu pro obvody rozdělené na TB a obvody, pro něž vygeneroval řetězec scan návrhový systém. Dále je zde uvedeno pokrytí poruch, jež dokáže odhalit vygenerovaný test. Pokrytí poruch je, až na jeden případ, větší u obvodů rozdělených na TB než u obvodů, pro něž vygeneroval řetězec scan návrhový systém. Přičemž čas na vygenerování testu je přibližně stejný. Výjimku tvoří obvod „m30“, u něhož je patrné, že generování testu pro obvod rozdělený na TB je méně časově náročné.

Význam jednotlivých sloupců v tabulce 7.21 je tento:

1. název obvodu,
2. doba potřebná k vygenerování testovacích vektorů pro obvod,
3. počet testovacích vektorů přivedených přes primární vstupy,
4. počet testovacích vektorů přivedených přes registr scan,
5. pokrytí poruch.

Alternativa s příponou TB uvádí výsledky pro obvod rozdělený na TB. Naopak tam, kde přípona chybí, byl pro definování částečného scanu použit nástroj DFTAdvizor firmy Mentor Graphics®. Pokrytí poruch se liší od pokrytí uvedeného v tabulkách 7.1 až 7.11, protože v tomto případě je generován test, který se snaží vygenerovat co nejméně testovacích vektorů a dosáhnout co největší pokrytí. Tento test je zvolen, aby vynikly rozdíly mezi návrhovým systémem a rozdělením na TB.

Počet sekvenčních testovacích vektorů vygenerovaných generátorem testu je v případě použití metodiky pro rozdělení obvodu na TB menší než je počet vektorů nutných pro otestování obvodu bez rozdělení. Metodika pro rozdělení obvodu na TB umožňuje vkládat testovací vektory i paralelně na primární vstupy, aplikace takovýchto vektorů je rychlejší než v případě sekvenčního načítání do registru scan.

Ozn. obvodu	generování t.v. [ms]	t. vektorů	sekv. vektorů	pokrytí poruch [%]
m10	20	0	9	27,9
m10-TB	10	9	6	68,3
m10b	20	0	13	65,7
m10b-TB	10	6	8	76,9
m10c	10	nelze	nelze	0,0
m10c-TB	< 10	0	2	63,5
m20	70	0	11	17,6
m20-TB	35	3	11	39,0
m30	320	0	10	18,9
m30-TB	20	13	7	78,9
kom5	20	0	14	95,4
kom5-TB	20	1	17	98,0
kom10	50	0	26	98,4
kom10-TB	10	3	20	97,0
kom20	25	0	35	99,0
kom20-TB	10	7	18	99,5
3m10	< 10	0	2	9,2
3m10-TB	< 10	5	0	32,6
3m20	10	2	4	12,3
3m20-TB	< 10	5	2	37,2
3m30	10	2	4	8,79
3m30-TB	10	1	4	24,3

Tabulka 7.21: Doba potřebná k vygenerování testovacích vektorů a pokrytí poruch

7.5 Příkon obvodu při diagnostickém testu

Snahou návrhářů diagnostických testů je vytvořit test, který bude dosahovat maximální pokrytí chyb a zároveň doba potřebná k jeho provedení bude co nejkratší. Mezi těmito parametry je nutné hledat kompromis, protože je není možné splnit současně. Dnes je však dalším důležitým parametrem příkon. Je nutné najít takový test, který při jeho aplikaci nepřekročí mezní hranici příkonu čipu (P_{max}). Z tohoto důvodu je nutné zahrnout příkon jako třetí rozměr kompromisního řešení. Jak již bylo řečeno dříve, je možné testovat více TB současně. Pokud však začneme počítat s příkonem čipu při aplikaci testu, je možné při plánování testu využít rozdělení obvodu na TB a vybrat pro souběžný test jen určitou skupinu TB tak, aby nebyla překročena hranice P_{max}.

Kapitola 8

Závěr

8.1 Výsledky práce a zhodnocení

Tato práce se zabývá analýzou číslicových obvodů popsaných na úrovni meziregistrových přenosů. V práci je zahrnuta pouze problematika související s testovatelností obvodových datových cest, řadičem ovládajícím tok dat těmito cestami se nezabývá. V úvodu jsou uvedeny úrovně popisu obvodu, ty jsou rozděleny na 6 úrovní. Práce se týká obvodů na úrovni RT, proto následuje detailnější popis této úrovně. Dále jsou shrnuty základní pojmy z diagnostiky číslicových obvodů a detekce a lokalizace poruch. Poté následuje kapitola zabývající se nástroji pro automatické generování testu pro sekvenční a kombinační číslicové obvody. Poslední část této kapitoly je věnována genetickému algoritmu a simulovanému žíhání.

V kapitole 3 je shrnut současný přístup k návrhu obvodů pro snadnou testovatelnost a popsány metody, jež se zabývají problematikou podobnou této práci. Techniky návrhu snadno testovatelného obvodu mají pozitivní i negativní dopad na analyzovaný obvod. Jasným přínosem je zvýšení řiditelnosti a pozorovatelnosti testovaného obvodu, čímž se dosáhne snížení doby potřebné pro generování testu, zvýšení pokrytí poruch a zkrácení testu z hlediska délky i času. Negativní dopad je však spatřován v nárůstu plochy čipu, zvýšení počtu vývodů, zvýšení příkonu, zhoršení dynamických vlastností obvodu či spolehlivosti. Mezi kladnými a zápornými aspekty je nutné nalézt vhodný kompromis, což je hlavním cílem navržených technik. Prezentované techniky se snaží rozdělit obvod za účelem zjednodušení generování testu sekvenčního číslicového obvodu. Jedna z metod využívá přidaných multiplexorů, další využívají řetězce scan. Multiplexory je možné použít pro přepnutí do režimu testu. V režimu testu se připojí primární vstupy přímo k testované části obvodu a odpojí se od původního spoje, v normálním režimu je přes multiplexor připojena původní datová cesta. Nevýhodou tohoto řešení je velký nárůst velikosti obvodu a nemožnost detekovat poruchu při chybném přepnutí multiplexoru do režimu funkce obvodu.

Dalším přístupem k testování obvodu jsou metody scan. První možností této metody je zaměnit všechny registry za registry scan a propojit je do řetězce. Sériově je tak možné transportovat diagnostická data do/z každého registru. Tento přístup však neúměrně zvětšuje plochu čipu a také se projevuje časová režie sériového přenosu dat. Aby se snížil negativní dopad této techniky, jsou zařazovány do řetězce scan jen některé registry. Vhodným výběrem těchto registrů se zabývají různé metody založené na různých přístupech. Jsou to například analýza míry testovatelnosti, analýza struktury obvodu, analýza spotřeby energie, analýza transparentních cest a další.

V další kapitole je pak zaveden formální model datové části číslicových obvodů na úrovni RT a některých jeho vlastností, význačných z pohledu analýzy testovatelnosti [58]. Model celého obvodu je tvořen uspořádanou pětící množin. Lze říci, že model má charakter grafu. Model umožňuje dále hierarchický popis. Je provedena základní klasifikace obvodových prvků s ohledem na jejich chování a roli v obvodě a též s ohledem na skutečnosti, které plynou z použití strategie syntézy s využitím multiplexovaných datových cest. Jedna z množin základní pětice modelující obvod je relace. Je to relace spojů mezi branami obvodových prvků. Zavedení formálního modelu umožnilo převést problémy související s analýzou testovatelnosti obvodu na problémy diskrétní matematiky, matematické logiky a teoretické informatiky. Dále je zaveden množinový model umožňující popsat transparentní vlastnosti prvků a také celého analyzovaného obvodu, čehož metodika pro rozdělení obvodu na TB využívá.

Formální model, který je zaveden v předchozí kapitole, je využit formálním modelem TB, který je v dizertační práci navržen. Formální model je rozšířen o další množiny a pomocí definic je vymezen a formálně definován TB. Na TB můžeme nahlížet jako na část obvodu, která je plně testovatelná přes svoje vstupy a výstupy - hraniční registry nebo primární vstupy a výstupy. Pouze hraniční registry je možné do řetězce scan zařadit. V modelu jsou vymezeny obvodové prvky. Obvodový prvek může mít kombinační charakter (multiplexory, logické prvky) nebo sekvenční charakter (registry). Rozhraní prvků tvoří brány, které jsou přiřazeny k jednotlivým prvkům pomocí funkce ψ. Mezi těmito branami je pak realizováno propojení. Propojení je dále možné s primárními vstupy a výstupy, jež jsou definovány jako samostatné množiny. Dále je definováno rozhraní TB, což je vlastně množina hraničních registrů, přes něž je TB diagnostikován. Struktura TB je vymezena množinou spojů, která určuje propojení bran obvodových prvků mezi sebou a připojení k primárním vstupům a výstupům. Výsledkem je pětice reprezentující TB. Nakonec jsou vymezeny nutné podmínky pro existenci TB, jež zamezují překrývání TB a určují zařazení hraničních registrů do řetězce scan. Zavedené pojmy jsou demonstrovány na vhodných příkladech a doplněny obrázky.

Další kapitola se zabývá metodikou pro rozdělení obvodu na TB, jež je na formálním modelu založena. Práce si kladla za cíl navrhnout metodiku, jejíž podstatou bude zjednodušení generování testu pro sekvenční obvody. Hlavní částí sekvenčních číslicových obvodů jsou paměťové prvky (registry) a právě tyto prvky jsou důležité i pro navrženou metodiku. Základní postup analýzy obvodu spočívá v převedení struktury obvodu zapsaného ve VHDL/Verilogu do formálního modelu a rozdělení pomocí metodiky na TB. Hraniční registry jsou pak zařazeny do registru scan. Použití metodiky rozdělení na TB je vhodné zejména pro obvody obsahující větší množství registrů a menší množství zpětných vazeb. Testovatelnost obvodu rozděleného na TB je dále možné porovnávat s profesionálními nástroji. Metodika je uvedena ve formě algoritmů, pracujících na matematických pojmech. Algoritmy jsou slovně vysvětleny a doplněny vývojovými diagramy. Základem algoritmu je procházení transparentních cest a přiřazení registrů a obvodových prvků do TB. Výběr hraničních registrů je proveden genetickým algoritmem a simulovaným žíháním. Dále je v práci diskutován vliv smyček v obvodu na možnosti rozdělení obvodu na TB. Je konstatováno, že tento vliv je značný a značně omezuje metodiku při detekci TB. Proto je navržena a implementována metodika pro detekci a přerušení smyček diagnostickými registry.

Následuje popis implementace algoritmu. Je zvolen genetický algoritmus a simulované žíhání. Poté je určena podoba fitness funkce. Fitness funkce se skládá z 6 částí, mezi nimiž je proveden vážený součet. Genetický algoritmus a simulované žíhání pak optimalizuje výběr hraničních registrů tak, aby bylo dosaženo co největší hodnoty fitness funkce. Algoritmus

rozdělení obvodu na TB na základě vybraných hraničních registrů je popsán slovně a doplněn vývojovým diagramem.

Z experimentálních výsledků uvedených v kapitole 7 vyplývá, že počet registrů zařazených do řetězce scan je pro dané obvody srovnatelný s profesionálním návrhovým systém. Nevýhodou této metodiky je čas potřebný k analýze a rozdělení obvodu. U syntetických testovacích obvodů je vždy počet registrů, jež musí být zařazeny do řetězce scan, nižší než navrhuje komerční systém při analýze DFT a větší než v případě automatické volby analýzy systémem. Doba výpočtu je polynomiálně závislá na počtu i-cest. Metodika pro detekci a přerušení smyček vede ke snížení počtu obvodových prvků, jež nelze zařadit do žádného TB. Pokrytí poruch je většinou větší u obvodů rozdělených na TB než u obvodů, pro něž vygeneroval řetězec scan návrhový systém.

8.2 Přínos

Za zásadní přínos této práce lze považovat vytvoření nové metodiky pro zajištění testovatelnosti číslicového obvodu na úrovni RT. Metodika je založena na identifikaci testovatelných bloků podle definovaných kritérií. Přínos práce je následující:

1. Jsou definovány vlastnosti testovatelného bloku (ty jsou modifikovatelné pro jiné typy zadání).
2. Je zaveden formální popis číslicového obvodu na úrovni RT (do jisté míry převzat z předcházejících prací), je doplněn o formální popis testovatelného bloku a jeho vlastností.
3. Na formálním modelu jsou definovány procedury identifikace testovatelných bloků.
4. Na formálním modelu jsou tyto procedury implementovány.
5. Pro ověření metodiky jsou použity dva optimalizační postupy - genetický algoritmus a horolezecký algoritmus, zkušenosti s jejich využitím jsou v práci diskutovány.
6. Na několika číslicových obvodech je metodika identifikace testovatelných bloků ověřena, experimentální výsledky jsou v práci diskutovány.
7. Bylo prokázáno, že pro účely diagnostiky a testování číslicových systémů je možné využít matematický aparát, především diskrétní matematiku, teorii množin a teorii grafů.

8.3 Publikované práce

Články na konferencích: [20, 21, 24, 23, 22, 33, 32, 26, 25, 34, 35, 27].
Článek v časopise: [36].

Literatura

[1] Abadir, M. S.; Breuer, M. A.: A Knowledge Based System for Designing Testable VLSI Chips. *IEEE Design & Test of Computers*, srpen 1985, s. 56–68.

[2] Abraham, J. A.; Vishakantaiah, P.; Saab, D. G.: Composition of Hierarchical Sequential Tests Using ATKET. *Proceedings of International Test Conference*, 1993, s. 606–615.

[3] Abramovici, M.; Breuer, M. A.; Friedman, A. D.: *Digital Systems Testing and Testable Design*. IEEE Press and Piscataway, 1990, 670 s.

[4] Agrawal, V. D.; Cheng, K.; Johnson, D. D.; aj.: A Complete Solution to the Partial Scan Problem. *International Test Conference*, 1987, s. 44 – 51.

[5] Agrawal, V. D.; Cheng, K.; Johnson, D. D.; aj.: Designing Circuits with Partial Scan. *IEEE Design&Test of Computers*, April 1988, s. 8 – 15.

[6] Aitken, R. C.: An Overwiev of Test Synthesis Tools. ročník 1, č. 1, 1995, s. 8–15.

[7] Arora, V.; Sengupta, I.: A Unified Approach to Partial Scan Design using Genetic Algorithm. *Proceedings of the 14th Asian Test Symposium (ATS '05)*, 2005, s. 414–421.

[8] Banos, R.; Gil, C.; Montoya, M.; aj.: A Parallel Evolutionary Algorithm for Circuit Partitioning. *Eleventh Euromicro Conference on Parallel, Distributed and Network-Based Processing*, 2003, s. 365–371.

[9] Bäck, T.: *Evolutionary Algorithms in Theory and Practice*. New York, Oxford: Oxford University Press, 1996, 314 s.

[10] Chakradhar, S. T.; Agrawal, V. D.; Bushnell, M. L.: *Neural Models and Algorithms for Digital Testing*. Kluwer Academic Publishers, 1991, ISBN 0-7923-9165-9, 184 s.

[11] Chakradhar, S. T.; Balakrishnan, A.; Agrawal, V. D.: An exact algorithm for selecting partial scan flip-flops. *J. Electron. Test.*, ročník 7, č. 1-2, 1995, s. 83–93, ISSN 0923-8174, doi:http://dx.doi.org/10.1007/BF00993316.

[12] Cheng, K.; Agrawal, V.: A partial scan method for sequential circuits with feedback. *IEEE Transactions on Computers*, ročník 39, April 1990, s. 544–548.

[13] Cohen, B.: *VHDL Coding Styles and Methodologies*. Dodrecht, Netherlands: Kluwer Academic Publishers, 2001, ISBN 0-7923-8474-1, 460 s.

[14] Cox, H.: Synthesizing Circuits with Implicit Testability Constraints. *IEEE Design & Test of Computers*, ročník 1, 1995, s. 16–22.

[15] Crouch, A. L.: *Design for Test: For Digital Integrated Circuits*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999, ISBN 0-1308-4827-1, 347 s.

[16] Drechsler, R.; Drechsler, N.: *Evolutionary Algorithms for Embedded System Design*. Kluwer Academic Publishers, 2002, ISBN 1-4020-7276-7, 177 s.

[17] Fujiwara, H.: *Logic Testing and Design for Testability*. Cambridge, MA, USA: MIT press, 1985, ISBN 0-262-06096-5, 304 s.

[18] Goldberg, D.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addision-Wesley Publishing Company, 1989, 432 s.

[19] Goldstein, L. H.; Thigpen, E. L.: SCOAP: Sandia controllability/observability analysis program. *Proceedings of the 17th conference on Design automation*, Minneapolis, Minnesota, United States, June 1980, s. 190–196.

[20] Herrman, T.: Využití optimalizačních technik pro výběr registrů do řetězce SCAN. *Proceedings of 10th Conference and Competition Student EEICT 2004, Volume 1*, Faculty of Electrical Engineering and Communication BUT, 2004, ISBN 80-214-2634-9, s. 263–265.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=7570

[21] Herrman, T.: Metody aplikace testu založené na testovatelných jádrech. *Počítačové architektury & diagnostika 2005*, Czech Technical University, 2005, ISBN 80-01-03298-1, s. 51–54.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=7893

[22] Herrman, T.: Formal Model of Testable Block. *Proceedings of 12th Conference Student EEICT 2006, Volume 4*, Faculty of Electrical Engineering and Communication BUT, 2006, ISBN 80-214-3163-6, s. 451–455.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8051

[23] Herrman, T.: Metodika aplikace testu obvodu založená na identifikaci Testovatelných bloků. *Počítačové architektúry a diagnostika - zborník príspovkov*, Institute of Informatics, Slovak Academy of Sciences, 2006, ISBN 80-969202-2-7, s. 131–136.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8175

[24] Herrman, T.: Testability Analysis Based on Formal Model. *Proceedings of the Sevnth International Scientific Conference ECI 2006*, Faculty of Electrical Engineering and Informatics, University of Technology Košice, 2006, ISBN 80-8073-598-0, s. 243–248.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8176

[25] Herrman, T.: Metodika identifikace testovatelných bloků v obvodu na úrovni RT. *Počítačové architektury a diagnostika 2007*, University of West Bohemia in Pilsen, 2007, ISBN 978-80-7043-605-9, s. 67–76.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8453

[26] Herrman, T.: Testability Analysis Based on the Identification of Testable Blocks with Predefined Properties. *MEMICS proceedings 2007*, Ing. Zdeněk Novotný, CSc., 2007,

ISBN 978-80-7355-077-6, s. 269–269.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8500

[27] Herrman, T.: Identifikace testovatelných bloků v obvodu na úrovni RT. *Počítačové architektury a diagnostika 2008*, Liberec University of Technology, 2008, ISBN 978-80-7372-378-1, s. 25–35.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8812

[28] Holland, J. H.: Adaption in Natural and Artificial Systems. *SIGART Newsletter*, ročník 1975, č. 53, 1975, s. 15–15.

[29] Hosokawa, T.; Kawaguchi, K.; Ohta, M.; aj.: A Design for Testability Method Using RTL Partitioning. *Proceedings of the 5th Asian Test Symposium (ATS '96)*, 1996, s. 88–93.

[30] Jervan, G.; Markus, A.; Raik, J.: Hierarchical Test Generation with Multi-Level Decision Diagram Models. *Proceedings of the 7th IEEE North Atlantic Test Workshop*, West Greenwich, RI, USA, 1998, s. 26–33.

[31] Škarvada, J.: *Verifikace testovatelnosti návrhu číslicového obvodu*. Diplomová práce, VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ, Fakulta informačních technologií, 2004.

[32] Škarvada, J.; Herrman, T.; Kotásek, Z.: RTL Testability Analysis Based on Circuit Partitioning and Its Link with Professional Tool. *IEEE 8th Workshop on RTL and High Level Testing*, Institute of Computing Technology, Chinese Academy of Sciences, 2007, s. 175–181.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8487

[33] Škarvada, J.; Herrman, T.; Kotásek, Z.: Testability Analysis Based on the Identification of Testable Blocks with Predefined Properties. *10th EUROMICRO CONFERENCE ON DIGITAL SYSTEM DESIGN Architectures, Methods and Tools (DSD 2007)*, IEEE Computer Society, 2007, ISBN 0-7695-2978-X, s. 611–618.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8414

[34] Škarvada, J.; Kotásek, Z.; Herrman, T.: Power Conscious RTL Test Scheduling. *4th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, Masaryk University, 2008, ISBN 978-80-7355-082-0, s. 265–265.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8795

[35] Škarvada, J.; Kotásek, Z.; Herrman, T.: Power Conscious RTL Test Scheduling. *Proceedings of 11th Euromicro Conference on Digital Systems Design Architectures, Methods and Tools*, IEEE Computer Society, 2008, ISBN 978-0-7695-3277-6, s. 721–728.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8700

[36] Škarvada, J.; Kotásek, Z.; Herrman, T.: Testability Analysis Based on the Identification of Testable Blocks with Predefined Properties. *Microprocessors and Microsystems*, ročník 32, č. 5, 2008, s. 296–302, ISSN 0141-9331.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8699

[37] Kiefer, K.; Wunderlich, H.: Deterministic BIST with Multiple Scan Chains. *sborník konference IEEE ETW (European Test Workshop)*, 1998, s. 39–43.

[38] Kotásek, Z.: Uplatnění principů řiditelnosti/pozorovatelnosti při návrhu číslicových obvodů, Habilitační práce. Brno University of Technology, 2000, str. 80.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=6042

[39] Kotásek, Z.; Pečenka, T.; Strnadel, J.: Improving Testability Parameters of Pipelined Circuits Through the Identification of Testable Cores. *Proc. of the 7th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, Slovak Academy of Science, 2004, ISBN 80-969117-9-1, s. 99–104.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=7507

[40] Kotásek, Z.; Růžička, R.: Behavioral Analysis for Testability on VHDL Source File. *Proceedings of Design and Diagnostics of Electronic Circuits and Systems Workshop sborník konference IEEE DDECS*, Slovak Academy of Science, 2000, ISBN 80-968320-3, s. 209–212.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=6043

[41] Kotásek, Z.; Růžička, R.; Zbořil, F.: Partial Scan Methodology in VHDL Environment. *CEI'99*, 1999, ISBN 80-88922-05-4, s. 146–151.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=6608

[42] Kotásek, Z.; Strnadel, J.; Pečenka, T.: Methodology of Selecting Scan-Based Testability Improving Technique. *Proc. of 8th IEEE Design and Diagnostic of Electronic Circuits and Systems Workshop*, University of West Hungary, 2005, ISBN 963-9364-48-7, s. 186–189.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=7747

[43] Kvasnička, V.; Pospíchal, J.; Tiňo, P.: *Evolučné algoritmy*. STU Bratislava, 2000, ISBN 80-227-1377-5, 215 s.

[44] Lee, D.; Reddy, S.: On determining scan flip-flops in partial-scan designs. *Computer-Aided Design, 1990. ICCAD-90. Digest of Technical Papers.*, Santa Clara, CA, USA, November 1990, s. 322–325.

[45] Lee, J.; Patel, J. H.: Hierarchical Test Generation Under Architectural Level Functional Constraints. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems.*, ročník 15, 1997, s. 1144–1151.

[46] Lee, M. T.: *High-level Test Synthesis of Digital VLSI Circuits*. Artech House, 1997, 220 s.

[47] Maerz, S.: High-Level Synthesis. *The Synthesis Approach to Digital System Design*, Kluwer Academic Publishers, 1992, s. 115–220.

[48] Marinissen, E. J.: Philips' Approach to Core-Based System Chip Testing. *4th IEEE DDECS 2001*, 2001, ISBN 963-7175-16-4, s. 15–24.

[49] Marinissen, E. J.: The Role of Test Protocols in Automated Test Generation for Embedded-Core-Based System ICs. *Journal Of Electronic Testing: Theory and Applications (JETTA)*, ročník svazek 18, srpen 2002, str. 10.

[50] März, S.: *High-Level Synthesis*, ročník 170 of The Kluwer International Series in Engineering and Computer Science. Boston: Kluwer Academic Publishers, 1992, ISBN 0-7923-9199-3, 432 s.

[51] Murray, B. T.; Hayes, J. P.: Hierarchical Test Generation Using Precomputed Tests for Modules. *IEEE Transactions on Computer Aided Design*, ročník 9, 1990, s. 594–603.

[52] Obitko, M.: Introduction to genetic algorithms with Java applets. http://www.obitko.com/tutorials/genetic-algorithms, 1998. URL http://www.obitko.com/tutorials/genetic-algorithms

[53] Orailoglu, A.; Makris, Y.: Property-Based Testability Analysis for Hierarchical RTL Designs. *Proceedings of the International Conference on Electronics Circuits and Systems (ICECS)*, 1999, s. 1089–1092.

[54] Orailoglu, A.; Makris, Y.; Collins, J.; aj.: A System for RTL Testability Analysis, DFT Guidance and Hierarchical Test Generation. *Proceedings of Custom Integrated Circuits Conference.*, 1999, s. 159–162.

[55] Paulin, P. G.; Knight, J. P.; Girczyc, E. F.: A multi–paradigm approach to automatic data path synthesis. *Sborník 23rd ACM/IEEE Design Automation Conference (DAC)*, Association for Computing Machinery, Inc., 1986, s. 263–270.

[56] Pečenka, T.: *Metodika analýzy testovatelnosti obvodu na úrovni RT*. Diplomová práce, FIT VUT v Brně, 2003.

[57] Rechenberg, I.: *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Fromman-Holzboog Verlag, 1973, 170 s.

[58] Růžička, R.: *Formální přístup k analýze testovatelnosti číslicových obvodů na úrovni RT*. Dizertační práce, Fakulta informačních technologií VUT v Brně, 2002. URL http://www.fit.vutbr.cz/research/view_pub.php?id=7031

[59] Schulz, M. H.; Auth, E.: Improved Deterministic Test Pattern Generation with Applications to Redundancy Identification. *IEEE Transactions on CAD*, ročník 8, č. 7, 1989, s. 811–816, ISSN 0278-0070.

[60] Wagner, R.: Mersenne Twister Random Number Generator. URL http://www-personal.umich.edu/~wagnerr/MersenneTwister.html

[61] Waicukauski, J. A.; Shupe, P. A.; Giramma, D. J.: ATPG for Ultra-Large Structured Designs. *Proceedings of the International Test Conference*, 1990, ISBN 0-8186-9064-X, s. 44–51.

[62] Wang, S.; Wei, W.: Low Overhead Partial Enhanced Scan Technique for Compact and High Fault Coverage Transition Delay Test Patterns. *13th European Test Symposium*, 2008, s. 125–130.

[63] Wu, S.; Wang, L.-T.; Jiang, Z.; aj.: On Optimizing Fault Coverage, Pattern Count, and ATPG Run Time Using a Hybrid Single-Capture Scheme for Testing Scan Designs. *IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, 2008, s. 143–151.

Seznam symbolů a zkratek

UUA	analyzovaný obvod - unit under analysis
E	množina obvodových prvků - elements
P	množina bran obvodových prvků - ports
C	množina spojů - connections
PI	množina primárních vstupů obvodu - primary inputs
PO	množina primárních výstupů obvodu - primary outputs
R	množina registrů
MX	množina multiplexorů
FU	množina funkčních jednotek - function units
IN	množina vstupních bran obvodových prvků
OUT	množina výstupních bran obvodových prvků
CI	množina řídících a synchronizačních bran obvodových prvků
$\psi(e)$	funkce, která přiřazuje množinu bran obvodovému prvku $e \in E$
$\square$	konec části důkazu
$\blacksquare$	konec důkazu
V	uzel (množina propojených bran)
$\uparrow$	náběžná hrana
$\downarrow$	sestupná hrana
$\rightarrow$	implikace
D	množina hodnot, kterých může uzel (brána) v obvodě nabývat
$\nu(p)$	zobrazení, přiřazující hodnotu z množiny D bráně $p \in P$ v daném okamžiku
I	Množina všech i-cest v obvodě, relace mezi dvojicemi bran v obvodě
$\rho(p_1, p_2)$	Relace přiřazující i-cestě mezi dvěma branami p_1, p_2 posloupnost bran, přes něž i-cesta vede
E_{TB}	množina obvodových prvků jež patří do TB
P_{TB}	množina bran obvodových prvků uvnitř TB
PO_{TB}	podmnožina primárních výstupů obvodu, jež jsou připojeny do TB
PI_{TB}	podmnožina primárních vstupů obvodu, jež jsou připojeny do TB
R_{TB}	množina registrů TB
BRO_{TB}	množina výstupních hraničních registrů TB
BRI_{TB}	množina vstupních hraničních registrů TB
BR_{TB}	množina hraničních registrů TB
C_{TB}	množina spojů TB
TB	formální model Testovatelného bloku
$SCAN$	množina registrů přes něž je aplikován test
$\sigma(e)$	funkce, která prvku $e \in E$ přiřazuje celé nezáporné číslo určující počet synchronizačních pulsů potřebných pro průchod dat přes tento prvek
s_{TB}	počet synchronizačních pulzů potřebných pro průchod diagnostických dat přes TB
E_TB	množina uspořádaných dvojic, která každému obvodovému prvku přiřazuje právě jedno celé číslo označující TB, do kterého patří
R_TB	množina uspořádaných dvojic, která každému registru přiřazuje právě jednu dvojici přirozených čísel označujících, do kterého TB patří vstup a do kterého výstup registru

II	množina všech ii-cest v obvodě, relace mezi dvojicemi bran v obvodě
$\neg$	bitová negace hodnoty na bráně
tir	vstupní registr testu
tor	výstupní registr testu
vrt	vstupní registr testu
$vyrt$	výstupní registr testu
tdr	vysílač testovacích vektorů (registr)
trv	přijímač odezev na testovací vektory (registr)
TIR_{tdr}	množina vstupních registrů testu k danému vysílači testovacích vektorů
TOR_{trv}	množina výstupních registrů testu k danému přijímači odezev na testovací vektory
$\Im$	množina množin (tříd) registrů tir
$\aleph$	množina množin (tříd) registrů tor