VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PROSTŘEDKY A METODY PRO AUTOMATICKÉ GENEROVÁNÍ TESTOVACÍCH OBVODŮ

TOOLS AND METHODS FOR AUTOMATED GENERATING OF BENCHMARK CIRCUITS

DISERTAČNÍ PRÁCE
DOCTORAL THESIS

AUTOR PRÁCE Ing. Tomáš Pečenka
AUTHOR

VEDOUCÍ PRÁCE Doc. Ing. Zdeněk Kotásek, CSc.
SUPERVISOR

BRNO 2007

Prostředky a metody pro automatické generování testovacích obvodů

Prohlášení

Prohlašuji, že jsem tuto disertační práci vypracoval samostatně pod vedením Doc. Ing. Zdeňka Kotáska, CSc. Další informace mi poskytli Doc. Ing. Lukáš Sekanina, Ph.D. a Ing. Josef Strnadel, Ph.D..
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Pečenka
15.6.2007

Citace

Pečenka Tomáš: Prostředky a metody pro automatické generování testovacích obvodů. Brno, 2007, disertační práce, FIT VUT v Brně.

Poděkování

Na tomto místě bych rád poděkoval všem, kteří přispěli k vytvoření této disertační práce. Zejména bych rád poděkoval svému školiteli Doc. Ing. Zdeňku Kotáskovi, CSc. za vedení v oblasti mého doktorského studia a za podporu při vytváření této práce. Rád bych také poděkoval Doc. Ing. Lukáši Sekaninovi, PhD. za cenné připomínky, nápady a motivaci při publikování dílčích výsledků této práce. Dále bych rád poděkoval Ing. Josefu Strnadelovi, PhD. za poskytnutí metod pro ohodnocení testovatelnosti obvodů a Ing. Janu Kořenkovi a Ing. Tomáši Martínkovi za cenné rady z oblasti návrhu číslicových obvodů.

Za finanční podporu mého výzkumu pak děkuji zejména grantové agentuře GAČR, která jej v letech 2004–2006 podporovala v rámci grantu GA102/04/0737 „Moderní metody syntézy číslicových systémů“ a v letech 2005–2006 v rámci grantu GD102/05/H050 „Integrovaný přístup k výchově studentů DSP v oblasti paralelních a distribuovaných systémů“. Za finanční podporu bych také rád poděkoval agentuře Fondu rozvoje vysokých škol, který tuto práci podporoval v roce 2005 v rámci grantu FR3041/2005/G1 „Evoluční návrh testovacích obvodů“. Za ocenění mého výzkumu Cenou Siemens 2004 bych rád poděkoval společnosti Siemens, Českému vysokému učení technickému v Praze a Fóru průmyslu a vysokých škol.

Také bych rád poděkoval všem mým blízkým za trpělivost a podporu při vytváření této práce.

Tato práce vznikala v letech 2003-2007 na Ústavu počítačových systémů Fakulty informačních technologií Vysokého učení technického v Brně. Děkuji tedy i tomuto ústavu a fakultě za pracovní podmínky, které vedly k dokončení této práce.

Tomáš Pečenka
Rožnov pod Radhoštěm, 14.června 2007

Abstrakt

Vzhledem k rostoucí složitosti číslicových systémů se jejich návrh realizuje stále častěji na vyšších úrovních popisu obvodu. Tento trend kopírují také návrháři diagnostických metod a nástrojů. Vznikají tak nové metody a nástroje pracující na vyšších úrovních popisu. Pro účely ověřování správné funkce těchto nástrojů potřebujeme mít k dispozici prostředky, prostřednictvím kterých bychom měli možnost ověřit, že navržené metody pracují podle teoretických předpokladů, popř. bychom měli možnost porovnat dosažené výsledky s výsledky jiných autorů. Pro tyto účely se již mnoho let používají sady testovacích obvodů (angl. benchmark sets). V současné době však bohužel neexistuje sada testovacích obvodů vhodná pro ověřování diagnostických metod pracujících na vyšší úrovni popisu – konkrétně na úrovni meziregistrových přenosů, která by obsahovala obvody odpovídající složitosti dnešních číslicových obvodů. Tato práce se zabývá způsobem, jak takové testovací obvody získat.

Metoda navržená v této práci představuje nový přístup k vytváření syntetických testovacích obvodů. Navržený přístup je založen na využití metody evolučního návrhu pro vytváření testovacích obvodů s požadovanou složitostí a diagnostickými vlastnostmi. Základním problémem všech dosud existujících přístupů využívajících pro návrh číslicových obvodů evoluční návrh byl problém škálovatelnosti. Existující metody byly schopny vytvářet pouze jednoduché číslicové obvody o složitosti v řádu maximálně jednotek tisíc hradel. Naproti tomu metoda navržená v této práci umožňuje navrhovat obvody o složitosti až stovek tisíc hradel. Této složitosti vytvářených obvodů bylo dosaženo díky aplikaci evolučního návrhu na problémy, u nichž jsme schopni dosáhnout ohodnocení kandidátního řešení v polynomiálním čase.

Příkladem právě takového problému je návrh syntetických testovacích obvodů s požadovanou složitostí a diagnostickými vlastnostmi. Návrh testovacích obvodů je v této práci realizován jako optimalizační proces, kde optimalizačním kritériem jsou vlastnosti obvodu z hlediska struktury spojů a diagnostických vlastností analyzovaného obvodu. Tyto vlastnosti jsou ohodnoceny prostřednictvím navržených algoritmů, které pracují nad formálním modelem struktury číslicového obvodu zavedeného v této práci. Stejný formální model je pak také použit pro popis samotné návrhové metody. Výhodou formálního přístupu je zejména jednoznačnost zápisu a možnost transformovat problémy návrhu na známé a řešené problémy diskrétní matematiky a teoretické informatiky.

Součástí práce je také poměrně rozsáhlé experimentální ověření návrhové metody, které bylo realizováno pomocí profesionálních nástrojů firmy Mentor Graphics. Výsledkem provedených experimentů je potvrzení, že navržená metoda je schopna navrhovat obvody s požadovanou strukturou a diagnostickými vlastnostmi. V současné době neexistuje v oblasti evolučního návrhu obvodů metoda, která by umožňovala realizovat návrh obdobně složitých obvodů s požadovanými diagnostickými vlastnostmi.

Klíčová slova

Sady testovacích obvodů, syntetické testovací obvody, evoluční návrh, analýza testovatelnosti, evoluční programování, FITTest_BENCH06, úroveň meziregistrových přenosů, diagnostika číslicových obvodů, model obvodu, hierarchický test.

Abstract

Due to the growing complexity of digital circuits the design activities are moving towards higher abstraction levels together with the design and diagnostic methods and tools for this level. New methods and tools which operate on higher description levels are coming up. It is necessary to develop resources which can be used to verify correct operation of the newly created methodologies. It must be also verified that theoretical assumptions were fulfilled and experimental results of the methodology compared with those of other authors. Benchmark circuits are used for these purposes for a long time. Unfortunately, no benchmark set exists containing circuits of the required complexity which can be used for the verification of high level diagnostic methods and tools – especially on the register transfer level. The goal of this work is to provide of such benchmark circuits.

The method presented in the thesis represents a new approach to the design of synthetic benchmark circuits. The limitation of all existing approaches which utilize evolutionary design to develop digital circuits with required properties is that the evolutionary approach is not ”scalable” (i.e. it is not able to generate circuits of an arbitrarily increasing complexity). Existing methods are able to design only simple digital circuits with the maximum complexity up to thousands of gates. In the developed approach, an evolutionary design is utilized to design benchmark circuits with required complexity and diagnostic properties. The method is able to design circuits with complexity of up to hundreds thousands of gates. This complexity was achieved because of applying evolutionary design to problems where it is possible to evaluate the candidate solutions in polynomial time complexity.

The design of synthetic benchmark circuits with required complexity and diagnostic properties is an example of such problem. In the thesis, the design of benchmark circuits is performed through optimization process with structural and diagnostic properties of the analyzed circuit being the optimization criteria. Structural and diagnostic properties are evaluated as a part of the developed algorithms which operate on the formal representation of the structure of digital circuit introduced in the thesis. The formal representation is also used to define the principles of the developed design method. The advantages of the utilized formal approach are especially the exactness of the notation and the possibility to transform the design problems to the known and already solved problems of discrete mathematics and theoretical informatics.

The complex experimental verification which was performed with the utilization of professional tools from Mentor Graphics, is also the part of the thesis. As the result of the experimental verification it was confirmed that the developed method is able to design benchmark circuits with required complexity and diagnostic properties. Nowadays, no other method exists in the evolutionary design which is able to design digital circuits with such complexity and required diagnostic properties.

Keywords

Benchmark sets, synthetic benchmark circuits, evolutionary design, testability analysis, evolutionary programming, FITTest_BENCH06, register-transfer level, diagnosis of digital circuits, circuit model, hierarchical test.

Obsah

Použité zkratky

Zkratka	Význam
ATPG	automatický generátor testovacích vektorů (z angl. Automatic Test Pattern Generator)
CAD	počítačem podporovaný návrh (z angl. Computer Aided Design)
D-KO	klopný obvod typu D
DFT	návrh s ohledem na testovatelnost (z angl. Design for Testability)
EA	evoluční algoritmus (z angl. Evolutionary Algorithm)
EDIF	univerzální formát pro popis elektronických obvodů (z angl. Electronic Design Interchange Format)
EP	evoluční programování (z angl. Evolutionary Programming)
FPGA	programovatelné hradlové pole (z angl. Field Programmable Gate Array)
HDL	jazyk pro popis hardwaru (z angl. Hardware Description Language)
ISCAS	International Symposium on Circuits and Systems – konference zaměřená na návrh a implementaci obvodů
ISCAS'85	sada testovacích obvodů představená na konferenci ISCAS v roce 1985
ISCAS'89	sada testovacích obvodů představená na konferenci ISCAS v roce 1989
ITC	International Test Conference – konference zaměřená na testování elektronických obvodů a systémů
ITC'99	sada testovacích obvodů představená na konferenci ITC v roce 1999
ITC'02	sada testovacích obvodů představená na konferenci ITC v roce 2002
LUT	základní stavební prvek architektury FPGA (z angl. Look-Up Table)
SoC	systém na čipu (z angl. System on Chip)
RT	úroveň meziregistrových přenosů (z angl. Register Transfer level)
VHDL	jazyk pro popis hardwaru (z angl. Very High-Speed Integrated Circuit Hardware Description Language)

Použité symboly a značky

Symbol/Značka	**Význam**
$\lvert M \rvert$	počet prvků množiny M
$\mathbb{N}$	množina přirozených čísel (včetně nuly)
$\mathbb{R}$	množina reálných čísel
$\mathbb{R}_{\langle 0,1 \rangle}$	množina reálných čísel ležících v intervalu $\langle 0, 1 \rangle$
$\emptyset$	prázdná množina
$\square$	konec zvláštního textu (definice, věty, důkazu)
$A \setminus B$	rozdíl množin A a B
C	konečná množina spojů obvodu
E	konečná množina prvků obvodu
I	rozhraní obvodu – čtveřice (DI, DO, CI, CO)
I_{ALL}	množina všech bran rozhraní
I_{CI}	množina vstupních řídicích bran rozhraní
I_{CO}	množina výstupních řídicích bran rozhraní
I_{DI}	množina vstupních datových bran rozhraní
I_{DO}	množina výstupních datových bran rozhraní
I_{IN}	množina vstupních bran rozhraní
I_{OUT}	množina výstupních bran rozhraní
$\mathcal{I}_{modes}$	množina všech *i-režimů* prvku
$\mathcal{I}_{paths}$	množina všech *i-cest* v obvodu
Φ	zobrazení přiřazující bitovou šířku branám obvodu
P_{ALL}	množina všech bran rozhraní obvodu
P_{CI}	množina vstupních řídicích bran obvodu
P_{CO}	množina výstupních řídicích bran obvodu
P_{DI}	množina vstupních datových bran obvodu
P_{DO}	množina výstupních datových bran obvodu
P_{IN}	množina vstupních bran obvodu
P_{OUT}	množina výstupních bran obvodu

Kapitola 1

Úvod

Se zvyšující se složitostí elektronických zařízení se zvyšují také požadavky na funkce realizované číslicovými obvody. Tato rostoucí složitost klade na návrháře nové požadavky v podobě hledání nových způsobů návrhu číslicových obvodů a jejich realizace. Nedílnou součástí úvah o nových metodách návrhu přitom musí být řešení problému testovatelnosti navrženého obvodu. V této souvislosti hovoříme o návrhu snadno testovatelných obvodů, v anglické terminologii označovaném jako *Design For Testability (DFT)*.

Pokud se zabýváme vývojem nových metod a nástrojů, potřebujeme mít k dispozici množinu testovacích obvodů (v angl. benchmark circuits), na níž bychom měli možnost ověřit, že námi navržené metody pracují podle teoretických předpokladů, popř. bychom měli možnost porovnat námi dosažené výsledky s výsledky jiných autorů. Pro tyto účely jsou k dispozici sady testovacích obvodů, které umožňují „objektivní“ srovnání různých metod a nástrojů (viz např. [9, 10, 19, 101–103]). Bohužel složitost testovacích obvodů v současné době dostupných testovacích sadách již nereprezentuje složitost dnešních číslicových obvodů – např. nejsložitější obvod z dnes používaných testovacích sad určených pro ověřování diagnostických metod a nástrojů (sady ISCAS’85, ISCAS’89 a ITC’99) se skládá pouze z 231 320 hradel a 6 642 klopných obvodů. Naproti tomu složitost dnešních číslicových systémů se pohybuje v řádech stovek miliónů až jednotek miliard tranzistorů [105].

S rostoucí složitostí číslicových obvodů se mění také způsob jejich návrhu. Návrh již není obvykle realizován na nižších úrovních, ale přesouvá se na vyšší úrovně popisu (např. úroveň meziregistrových přenosů). Tento přesun sebou přináší také potřebu nových CAD (z angl. Computer Aided Design) nástrojů, které musí umožňovat nejen samotný návrh na vyšší úrovni popisu, ale také se na této úrovni zabývat například diagnostickými vlastnostmi navrhovaných obvodů – realizovat analýzu testovatelnosti nebo generování testu.

Bohužel v současné době nemáme k dispozici sadu testovacích obvodů, která by umožňovala efektivní ověření nově vznikajících metod a nástrojů pracujících na vyšší úrovni popisu – k dispozici je pouze 31 relativně jednoduchých testovacích obvodů ze sady ITC’99 (revize této sady proběhla v roce 2002). Jak problém nedostatku testovacích obvodů řešit?

V některých oblastech se jako možná alternativa standardních testovacím obvodů začínají prosazovat tzv. *syntetické testovací obvody* [18, 48, 49, 51, 52, 60, 77, 87, 94, 95]. Syntetické testovací obvody jsou testovací obvody vytvořené pomocí automatizovaného procesu, který zajišťuje vytváření obvodů s požadovanými parametry. S použitím těchto testovacích obvodů se můžeme setkat například při ověřování algoritmů pro rozmístění prvků v FPGA [18] (angl. Place and Route algorithms), algoritmů pro automatické rozdělení velkých obvodů [77] (angl.

Partitioning algorithms), testování logických optimalizátorů [52], apod. Hlavní výhoda použití syntetických testovacích obvodů spočívá v možnosti mít plnou kontrolu nad důležitými charakteristickými vlastnostmi testovacího obvodu, jako je např. velikost obvodu, struktura spojů, popř. funkce realizovaná obvodem. Bohužel existující metody návrhu syntetických testovacích obvod jsou velmi jednoduché a umožňují pouze měnit existující vzorový obvod na základě sledování určitých jeho charakteristických vlastností. V současné době neexistuje metoda vytváření syntetických testovacích obvodů, které by bylo možné použít pro ověření diagnostických metod a nástrojů.

1.1 Cíle práce

Cílem této práce je návrh metody umožňující vytvářet syntetické testovací obvody s požadovanou složitostí a diagnostickými vlastnostmi, které by byly vhodné pro ověřování diagnostických metod a nástrojů pracujících na úrovni meziregistrových přenosů. Splnění cíle této práce pak předpokládá splnění následujících dílčích cílů:

- analýza existujících metod návrhu syntetických testovacích obvodů,
- návrh formálního modelu obvodu na úrovni meziregistrových přenosů vhodného pro formální popis metody návrhu syntetických testovacích obvodů,
- návrh a formální popis metody návrhu testovacích obvodů s požadovanými vlastnostmi,
- návrh a formální popis metody analýzy struktury a testovatelnosti obvodu vhodné pro návrh syntetických testovacích obvodů,
- implementace navržené metody návrhu,
- experimentální ověření navržené metody z hlediska splnění požadovaných vlastností vytvářených obvodů,
- vytvoření sady syntetických testovacích obvodů vhodné pro ověřování diagnostických metod a nástrojů.

Splněním cílů této práce získáme možnost vytvářet testovací obvody s požadovanými vlastnostmi, které budou použity pro efektivní ověření nových diagnostických metod a nástrojů pracujících na úrovni meziregistrových přenosů (angl. Register Transfer level – úrovni RT). Je možné předpokládat, že vznik uznávané sady testovacích obvodů by mohl představovat určitý impuls k rozvoji diagnostických metod na úrovni RT podobně, jako se tomu stalo u metod generování testu pro kombinační obvody po představení sady ISCAS'85 nebo u metod generování testu pro sekvenční obvody po představení sady ISCAS'89 [44].

1.2 Struktura práce

Druhá kapitola této práce je věnována zavedení základních pojmů z oblastí úzce souvisejících s tématem této práce, které jsou potřeba v další části práce. Nejprve jsou představeny základní úrovně modelování číslicových obvodů. Podrobněji je popsána zejména úroveň meziregistrových přenosů, která je použita pro modelování číslicových obvodů v této práci. Dále je

představena problematika diagnostiky číslicových obvodů, modelování poruch typu trvalá *0/1*, jsou popsány existující přístupy k vytváření testu číslicového obvodu a principy automatických generátorů testovacích vektorů. V závěru kapitoly jsou představeny principy optimalizačních metod evolučního programování a simulovaného žíhání, které jsou použity dále v této práci.

Cílem třetí kapitoly je seznámit čtenáře s vývojem a aktuálním stavem v oblastech, které úzce souvisí s řešenou problematikou. Nejprve jsou krátce popsány existující sady testovacích obvodů používané pro ověřování diagnostických metod a nástrojů. Dále je popsán aktuální stav v oblasti metod vytváření syntetických testovacích obvodů a představen princip vybraných metod návrhu. Podrobněji jsou popsány existující metody analýzy testovatelnosti obvodu na úrovni RT a práce z oblasti evolučního návrhu.

Ve čtvrté kapitole je zaveden formální model číslicového obvodu na úrovni RT, který je použit dále v této práci při formálním zápisu návrhové metody. Navržený formální model umožňuje popsat jak základní prvek úrovně RT tak hierarchicky popsaný obvod na úrovni RT pomocí trojice reprezentující rozhraní, vnitřní strukturu a propojení vnitřních bran obvodu. Na začátku kapitoly je nejprve formálně definována trojice reprezentující obvod a její jednotlivé členy. Dále jsou zavedeny některá pomocná zobrazení zjednodušující práci se zavedeným formálním modelem.

Pátá kapitola je věnována popisu samotné návrhové metody. Jsou specifikovány požadavky na návrhovou metodu a na základě specifikovaných požadavků je navržena metoda návrhu syntetických testovacích obvodů. Součástí kapitoly je také podrobný popis navržené metody, použitého optimalizačního algoritmu, zakódování problému, realizace operátoru mutace a procesu ohodnocení kvality kandidátních řešení. Pro popis celé návrhové metody je použit formální model zavedený ve čtvrté kapitole.

Samostatná kapitola je věnována metodě ohodnocení testovatelnosti kandidátního obvodu. Vzhledem k časové složitosti představuje ohodnocení testovatelnosti časově nejnáročnější část návrhového procesu. Vlastnosti použité metody analýzy testovatelnosti podstatným způsobem ovlivňují vlastnosti návrhové metody jako celku. Použitá metoda analýzy testovatelnosti je nejprve obecně popsána. Dále pak následuje podrobnější popis tzv. knihovny prvků, která obsahuje modely prvků, které jsou využity při analýze testovatelnosti obvodu. Jádro této kapitoly pak tvoří algoritmus ohodnocení řiditelnosti a pozorovatelnosti bran obvodu a algoritmus ohodnocení výsledné testovatelnosti obvodu.

Sedmou kapitolu tvoří výsledky získané při experimentálním ověření navržené metody. Cílem experimentálního ověření metody je v první části kapitoly zejména základní ověření a nalezení vhodných parametrů použitého optimalizačního algoritmu. Další část experimentů je pak zaměřena na experimentální ověření možností návrhové metody z hlediska rozsahu, na němž je schopna splnit uživatelem specifikované požadavky. Část experimentálního ověření je realizována s využitím profesionálních nástrojů od firmy Mentor Graphics, s jejichž pomocí je ověřena realizovatelnost a diagnostické vlastnosti navržených obvodů.

Osmá kapitola je věnována představení sady syntetických testovacích obvodů vytvořených pomocí metody představené v této práci a v závěrečné kapitole je diskutován přínos této práce a navrženy další možné směry navazujícího výzkumu.

Kapitola 2

Prerekvizity

Vzhledem k tomu, že tato práce předpokládá podrobnější znalosti z oblasti modelování číslicových obvodů, diagnostiky a také znalost optimalizačních technik, bude tato kapitola zaměřena na seznámení čtenáře s uvedenou problematikou. Na začátku kapitoly jsou popsány úrovně modelování číslicových obvodů. Podrobněji je představena zejména úroveň meziregistrových přenosů, která je v této práci použita pro modelování číslicového obvodu. Krátce je také popsána problematika testování číslicových obvodů, modelování poruch a automatického vytváření testu číslicového obvodu. Znalost pojmů z těchto oblastí je potřebná, abychom se mohli zabývat návrhem číslicových obvodů s požadovanými diagnostickými vlastnostmi a experimentálním ověřením navržené metody. V závěru kapitoly bude krátce popsán princip optimalizačních metod použitých v této práci.

2.1 Úrovně modelování číslicového obvodu

Číslicové obvody procházejí během posledních 40 let prudkým vývojem. Počet tranzistorů na čipu od uvedení prvního integrovaného obvodu v roce 1959 exponenciálně roste. V současné době dosahuje řádu stovek miliónů až jednotek miliard (viz procesor Intel Dual-Core Itanium 2 [105]). S rostoucí složitostí číslicových systémů se stává složitější také otázka jejich návrhu a případně testování.

Jedním ze způsobů, jak se vypořádat s rostoucí složitostí číslicových obvodů je realizovat jejich návrh na vyšší úrovni popisu a využít existující softwarové nástroje pro realizaci navržené struktury na nižších úrovních. Tento přístup návrháře odstiňuje od rutinních činností souvisejících s návrhem na nižších úrovních popisu obvodu a umožňuje mu koncentrovat se na vlastní funkci obvodu. Je však potřeba mít stále na paměti, že přestože softwarové nástroje dokáží některé části návrhu automatizovat, realizují obvykle pouze omezené typy transformací. Softwarové nástroje nemohou převést špatně vytvořený návrh na kvalitní implementaci řešení daného problému.

Podle úrovně abstrakce, se kterou při popisu číslicového obvodu pracujeme, můžeme rozlišit několik úrovní popisu. Obvykle se můžeme v literatuře setkat s následujícími úrovněmi popisu číslicového obvodu [13, 20, 96]:

systémová úroveň – obvod je modelován pomocí tzv. zdrojů (např. sběrnice, procesory, paměti). Tato úroveň popisu může být použita například pro analýzu efektivnosti systému

s ohledem na využití zdrojů, definici komunikačního protokolu mezi zdroji nebo algoritmu činnosti jednotlivých zdrojů.

úroveň popisu chování – obvod je modelován popisem chování s využitím některého z jazyků pro popis hardware (angl. Hardware Description Languages – HDL). Požadované chování výsledného obvodu je specifikováno pomocí konstrukcí vybraného jazyka HDL (např. VHDL, Verilog, Handel-C, . . .). Je důležité si uvědomit, že na této úrovni popisu ještě není známa struktura obvodu. Struktura obvodu se vytváří během procesu syntézy na základě parametrů syntézy a požadavků návrháře.

úroveň meziregistrových přenosů (úroveň RT) – obvod na této úrovni se skládá z datové a řídicí části. Datovou část tvoří funkční bloky oddělené registry či pamětmi a propojovací prvky (multiplexory a sběrnice). Řídicí část tvoří obvodový řadič, který bývá realizován stavovým automatem a řídí tok datovými cestami. Úroveň meziregistrových přenosů je nejabstraktnější úroveň modelování číslicového obvodu, na níž je již známa struktura obvodu. Velmi často je proto tento typ popisu obvodu návrháři používán pro kontrolu struktury obvodu. Myšlenka návrháře je na této úrovni stále zřejmá a nezaniká v záplavě automaticky generovaných struktur.

úroveň hradel – obvod je tvořen vzájemným propojením nízkoúrovňových prvků – obvykle logických hradel či klopných obvodů. Obvod na této úrovni je často výsledkem syntézy obvodu popsaného na vyšších úrovních popisu a je vstupem automatizovaného nástroje pro překlad do prvků cílové technologie.

úroveň tranzistorů – obvod je tvořen vzájemným propojením tranzistorů (např. unipolární technologie CMOS). Na této úrovni popisu nebývají obvody současné složitosti navrhovány přímo, ale tento popis je výsledkem automatizované implementace systému, jejímž hlavním cílem je příprava masky k výrobě čipu.

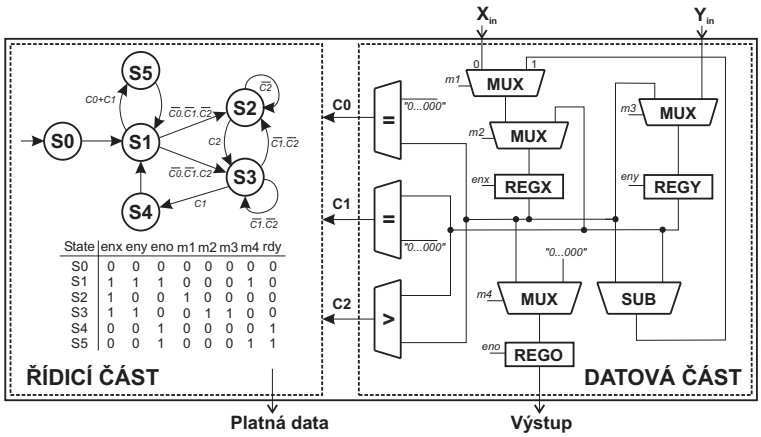
V této práci budeme dále pracovat s modelem číslicového obvodu na úrovni RT. Zabývat se návrhem, popř. diagnostikou právě na této úrovni je vhodné zejména proto, že se jedná o nejabstraktnější úroveň modelování číslicového obvodu, která již nese informaci o jeho struktuře. Tato informace již není k dispozici o úroveň výše (úrovni popisu chování), protože zobrazení *popis chování* → *úroveň RT* není injektivní. Například obvod realizující nějaký algoritmus popsaný chováním může mít mnoho podob od sekvenčního (sériového) po čistě paralelní řešení. Volba použité architektury bude záviset na podmínkách syntézy a požadavcích návrháře – požadavky na rychlý výpočet povedou spíše na paralelní řešení, kdežto prostorová omezení zapříčiní spíše sériové řešení [81].

Výhodou úrovně RT je také to, že je na této úrovni stále ještě zřejmá informace o funkci obvodu a tato informace ještě úplně nezaniká v záplavě automatizovaně generovaných opakujících se a navzájem si podobných struktur, jako je tomu na úrovni hradel. Této vlastnosti využívá také většina dostupných syntézních nástrojů (např. Leonardo Spectrum, Precision, . . .), které umožňují v průběhu vytváření struktury obvodu z vyšších úrovní popisu zobrazit schéma vytvořeného obvodu na úrovni RT. Uživatel má tak možnost ověřit strukturu obvodu po syntéze z popisu chování před započetím další fáze syntézy do nižších úrovní popisu, kde je již kontrola struktury (vzhledem k množství automaticky generovaných konstrukcí a provedených optimalizací) velmi obtížná. Podrobněji se obvodem na úrovni RT zabývá následující podkapitola.

2.1.1 Číslicový obvod na úrovni RT

Číslicový obvod na úrovni RT můžeme chápat jako systém, který je rozdělen do menších modulů, z nichž každý realizuje nějakou dílčí úlohu [66]. Moduly mohou představovat složitější obvodové konstrukce nebo základní prvky úrovně RT (registry, dekodéry, multiplexery, aritmeticko-logické prvky, řídicí logiku, . . .). Na obrázku 2.1 je příklad číslicového obvodu na úrovni meziregistrových přenosů, který realizuje výpočet největšího společného dělitele. Je patrné rozdělení obvodu na datovou a řídicí část. Řídicí část je tvořena stavovým automatem, který řídí tok dat datovou částí obvodu, která je tvořena registry, kombinačními prvky a propojením těchto prvků.

Z hlediska diagnostiky je zajímavá zejména datová část obvodu a to ze dvou důvodů. Prvním důvodem je to, že problém testování řídicí části obvodu je problém, který je považován za již vyřešený – stavový automat je možné modifikovat tak, aby umožňoval sám sebe otestovat [45]. Druhým důvodem proč je z hlediska testovatelnosti zajímavější datová část je ten, že tuto část obvykle tvoří více logických členů a vyšší je tedy i pravděpodobnost výskytu chyby v této části obvodu.



State	enx	eny	eno	m1	m2	m3	m4	rdy
S0	0	0	0	0	0	0	0	0
S1	1	1	1	0	0	0	1	0
S2	1	0	0	1	0	0	0	0
S3	1	1	0	0	1	1	0	0
S4	0	0	1	0	0	0	0	1
S5	0	0	1	0	0	0	1	1

Obr. 2.1: Příklad číslicového obvodu na úrovni meziregistrových přenosů.

Úmluva 1: Pokud budeme dále v této práci mluvit o *obvodu* na úrovni RT, bude implicitně myšlena jeho *datová část*. Moduly, popř. základní prvky úrovně RT, které tvoří datovou část obvodu, budeme v této práci souhrnně nazývat *prvky obvodu*. Pokud bude potřeba rozlišit, že se jedná o *modul*, popř *základní prvek* úrovně RT, tak to bude výslovně uvedeno. Rozhraní jednotlivých prvků obvodu tvoří tzv. *brány*. Podle orientace toku dat těmito branami můžeme rozlišit *vstupní*, *výstupní* a *vstupně/výstupní* brány. Pokud bude potřeba zdůraznit, že se jedná o bránu reprezentující rozhraní obvodu, bude použit výraz *primární brána*, popř. *primární vstup/výstup*. Dalšími možnými kritérii pro rozdělení bran pak bude informace o typu brány (datová/řídicí) nebo informace o její bitové šířce.
□

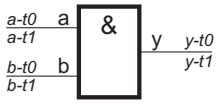
2.2 Diagnostika číslicových obvodů

Nefunkčnost obvodu může být způsobena chybou při jeho návrhu nebo závadou vzniklou při výrobě. Problematikou chyb vzniklých při návrhu se zabývá samostatná disciplína zvaná *verifikace návrhu*. Na chyby vzniklé při výrobě se pak zaměřuje disciplína všeobecně známá jako *diagnostika*. I když bude při verifikaci obvodu zjištěno, že navržený obvod plní správně požadovanou funkci, není tím zajištěno, že obvod vyrobený podle verifikovaného návrhu bude pracovat bezchybně. Během procesu výroby mohou být do obvodu zaneseny poruchy dané technologickým postupem výroby. Projevem poruchy pak obvykle je chybná (jiná než žádaná) funkce obvodu. Navíc se taková chybná funkce obvodu v praxi nemusí projevit ihned, pokud je zasažena část obvodu, která není aktivně využívána. Úkolem diagnostiky je vytvořit prostředky a metody, které umožní odhalit chybné chování vyrobeného obvodu a případně umožní lokalizovat jeho příčinu.

Cílem testování obvodu je obvykle pouze detekce *poruchy*, která je definována jako jev spočívající v ukončení schopnosti výrobku plnit požadovanou funkci podle technických podmínek [46]. Porucha je obvykle detekována na základě *chyby* – definované jako neshoda mezi správnou a skutečnou hodnotou proměnné zjištěné na výstupu. Cílem testování obvykle není zjištění o jakou poruchu se jedná, ani to, kolik poruch se v diagnostickém objektu vyskytuje. Výsledkem testu je pouze jednobitová hodnota informující o tom, zda byla nebo nebyla detekována porucha. Poznamenejme ještě, že každá porucha nemusí vést ke vzniku chyby. Potom hovoříme o tzv. *latentní poruše*.

2.2.1 Modelování poruch

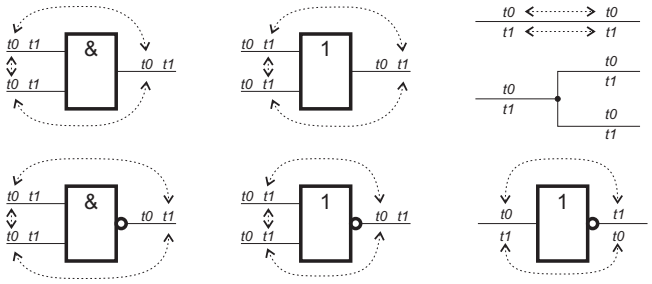
Modelování poruch spočívá v modelování fyzických poruch prostřednictvím matematických konstrukcí, které mohou být algoritmicky zpracovány (např. simulátorem poruch). Nejznámějším modelem poruchy je porucha typu *trvalá 0 (t0)* a *trvalá 1 (t1)* (angl. single-stuck-at fault). Základním předpokladem tohoto modelu je, že v obvodu existuje pouze jedna porucha. Modelování poruch typu *t0/t1* se obvykle používá pro modelování poruch na úrovni hradel. Poruchy typu *t0/t1* jsou na úrovni hradel modelovány připojením pinu hradla na vysokou úroveň (typ *t1*) nebo nízkou úroveň (typ *t0*). Na obrázku 2.2 je ukázka modelování šesti možných poruch typu *t0/t1* (*a-t0*, *a-t1*, *b-t0*, *b-t1*, *y-t0* a *y-t1*) dvouvstupého hradla AND.



Obr. 2.2: Modelování poruch dvouvstupého hradla AND pomocí modelu poruchy typu *t0/t1*.

Množinu všech poruch prvku/obvodu můžeme redukovat odstraněním tzv. ekvivalentních poruch. Ekvivalentními poruchami se rozumí poruchy, jež jsou vzájemně nerozlišitelné prostřednictvím primárních výstupů obvodu. Příkladem ekvivalentních poruch jsou u hradla AND na obrázku 2.2 poruchy *a-t0*, *b-t0* a *y-t0*. Tyto poruchy není možné na výstupu prvku vzájemně rozlišit. Výskyt libovolné z těchto poruch se na výstupu hradla AND projeví jako porucha *t0*. Rozklad množiny všech poruch do ekvivalentních tříd by pro hradlo AND na obrázku 2.2 vypadal následovně $\{\{a\text{-}t0, b\text{-}t0, y\text{-}t0\}, \{a\text{-}t1\}, \{b\text{-}t1\}, \{y\text{-}t1\}\}$. Podobně u hradla OR nemůžeme rozlišit mezi poruchou typu *t1* na vstupech a výstupu hradla. Množiny ekvivalentních poruch

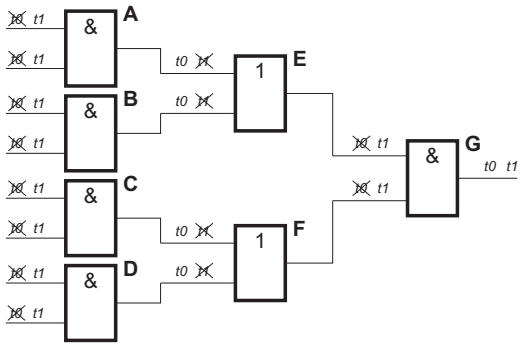
pro spoje a základní typy hradel jsou graficky naznačeny na obrázku 2.3.



Obr. 2.3: Ekvivalentní poruchy typu trvalá 0/1.

V praxi je tak potřeba vědět, zda udávaný počet poruch obvodu vyjadřuje počet všech neredukovaných poruch (parametr označovaný anglicky výrazem „uncollapsed faults“) nebo vyjadřuje redukovaný počet poruch (označovaný anglicky výrazem „collapsed faults“). Z hlediska vytváření testu je zajímavější počet redukovaných poruch, protože představuje počet poruch, pro něž je potřeba generovat testovací vektory.

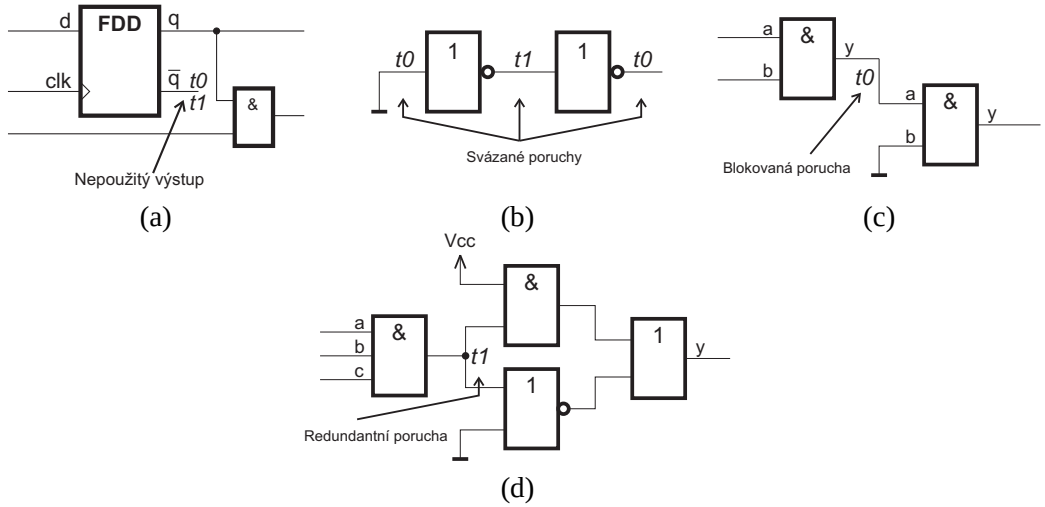
Princip nalezení redukované množiny poruch si můžeme ukázat na obvodu na obrázku 2.4. Nejprve vytvoříme seznam všech poruch v obvodu. Tento seznam je dán poruchami jednotlivých hradel z nichž se obvod skládá. Algoritmus odstranění ekvivalentních poruch začíná z primárních vstupů obvodu. Jednotlivá hradla jsou zpracována ve chvíli, kdy jsou zpracovány všechny vstupy tohoto hradla. Zpracování hradla spočívá v analýze vstupních a výstupních poruch. Výsledkem je redukce všech poruch, které jsou ekvivalentní výstupním poruchám daného hradla. Pokud předpokládáme obvod na obrázku 2.4, tak analýza začíná na primárních vstupech obvodu. Nejprve jsou odstraněny poruchy typu *t0* na vstupech hradel *A*, *B*, *C* a *D*, protože jsou ekvivalentní s poruchou *t0* na výstupu hradel. V druhém kroku jsou odstraněny poruchy typu *t1* na vstupech hradel *E* a *F*, protože jsou ekvivalentní s poruchou *t1* na výstupu těchto hradel. V posledním kroku jsou odstraněny poruchy typu *t0* na vstupech hradla *G*, protože jsou ekvivalentní s poruchou *t0* na výstupu hradla. V uvedeném obvodu se tak počet poruch redukoval z celkového počtu 30 poruch na 16 poruch, pro něž je potřeba vygenerovat testovací vektory.



Obr. 2.4: Ukázka obvodu s redukovaným seznamem poruch.

V obvodu však může existovat porucha, která se nemusí projevit chybou (jedná se o tzv. latentní poruchy). Na obrázku 2.5 jsou ukázky některých latentních poruch [71]. Příkladem latentní poruchy je například porucha na obrázku 2.5a). Výstup $\overline{q}$ není použit, porucha na

tomto výstupu se tedy neprojeví chybou. Podobně se neprojeví další poruchy zobrazené na obrázku 2.5.



Obr. 2.5: Příklady latentních poruch.

2.2.2 Pokrytí poruch

Základní mírou používanou pro stanovení kvality číslicových systémů z hlediska diagnostiky je informace o *pokrytí poruch* (angl. fault coverage) vztažená k dané množině testovacích vektorů. Parametr pokrytí poruch udává, jakou část poruch je daná množina testovacích vektorů schopna detekovat:

$$\textit{Pokrytí poruch} = \frac{\textit{Počet detekovaných poruch}}{\textit{Celkový počet poruch}} \cdot 100\,[\%]\,, \tag{2.1}$$

kde *Počet detekovaných poruch* představuje počet poruch, které je daná množina testovacích vektorů schopna detekovat a *Celkový počet poruch* udává celkový počet možných poruch obvodu.

Pokud tedy máme například obvod s celkovým počtem 10 000 poruch a navržený test detekuje 9 951 poruch, získáváme pokrytí poruch ve výši 99,51%. Obvyklé hodnoty pokrytí poruch jsou podle [17] od 95% do 99,9%. Bohužel neexistuje jednotnost mezi výrobci ATPG nástrojů z hlediska výpočtu pokrytí poruch. Různé nástroje mohou pro stejný obvod a stejnou testovací posloupnost úvádět různé hodnoty pokrytí poruch v závislosti na tom, jestli za celkový počet poruch považují počet redukovaných nebo neredukovaných poruch, popř. zda do celkového počtu poruch započítávají také např. latentní poruchy.

2.3 Testování číslicových obvodů

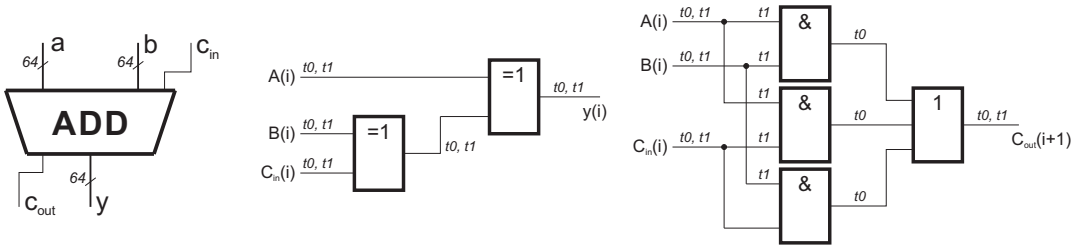
Test číslicového obvodu můžeme definovat jako „aplikaci požadovaných testovacích vektorů na vstupy obvodu s požadovaným vnitřním stavem a porovnání odezvy na testovací vektory s očekávanou odezvou [17]“. Test číslicového obvodu může být realizován na nejrůznějších

úrovních - od úrovně tranzistorů, hradel, čipů, desek až po úroveň systémů. Každý tento typ testování předpokládá aplikaci testovacích vzorů na vstupní piny testovaného objektu. U číslicových obvodů na úrovni hradel předpokládáme testovací vektory ve formě řetězce nad binární abecedou.

Aplikace testovacích vektorů na vstup testovaného objektu může být problematická zejména pokud je testovaný objekt součástí nějakého většího systému (např. testování čipů osazených na deskách). V takovém případě může být potřeba modifikovat testovací vektory tak, aby mohly být aplikovány prostřednictvím dalších čipů na desce, popř. prostřednictvím speciálních testovacích sběrnic. Nutnou podmínkou aplikace testovacích vektorů na daný prvek také je, aby byl testovaný prvek v požadovaném stavu. Nastavení do požadovaného stavu může být realizováno například aplikací dané sekvence hodnot nebo nějakou formou nulování jednotky. Poslední podmínkou je možnost sledování odezvy na testovací vektor. Přičemž požadovanou odezvu na testovací vektor získáme na základě předepsaného chování obvodu nebo simulací modelu obvodu. Odezvy na testovací vektory mohou být snímány buď přímo z primárních výstupů testovaného prvku, přeneseny přes strukturu obvodu nebo může být využita přídavná testovací struktura (např. příznakový analyzátor).

2.3.1 Funkční a strukturní testy

Generování testu číslicového obvodu je v dnešní době stále nejčastěji realizováno na úrovni hradel. Na této úrovni popisu se můžeme setkat s dvěma hlavními typy testů – funkčními a strukturními testy. Funkční testy jsou používány pro verifikaci požadovaného chování. Pokud budeme předpokládat, že testovaná jednotka představuje např. sčítačku, tak cílem funkčního testu bude ověření, že daný prvek skutečně realizuje sčítání. Pokud se jedná o obecnou sčítačku, tak bude otestována na všechny funkční kombinace – pro *n* bitovou sčítačku to máme $2^{(2 \cdot n)}$ testovacích vzorů. Naproti tomu strukturní test vytváří testovací vektory s cílem detekovat možné poruchy v obvodu. Výsledkem strukturního testu je ověření spojů obvodu a ověření funkce jednotlivých prvků obvodu. Při jeho sestavování vycházíme z použitého modelu poruchy – nejčastěji modelu poruchy typu *t0*/*t1*. Vytváření testovacích vektorů probíhá v několika krocích. Prvním krokem je aktivace poruchy – nastavení místa poruchy na opačnou hodnotu než kterou představuje porucha (např. poruchu *t0* aktivujeme nastavením místa poruchy na hodnotu log. 1). Druhým krokem je propagace hodnoty z místa poruchy na primární výstupy obvodu, kde může být porucha detekována na základě porovnání získané a očekávané hodnoty odezvy.



(a) (b) (c)

Obr. 2.6: Úplná sčítačka s postupným přenosem (a), modelování poruch typu trvalá 0/1 pro část realizující bitový součet (b) a přenos (c).

Princip funkčního a strukturního testu můžeme demonstrovat na obvodu 64 bitové sčítačky s postupným přenosem (viz obrázek 2.6). Z hlediska funkčního testu se sčítačka jeví jako modul se 129 bitovým vstupem a 65 bitovým výstupem. Pro úplné ověření funkce je tedy potřeba 2^{129} vstupních kombinací, které produkují 2^{65} výstupních kombinací. Testování takového prvku na testeru, generujícího testovací vzorky v taktu 1GHz by trvalo přibližně 10^{22} let [17].

Naproti tomu strukturní testy jsou zaměřeny na detekci dané množiny poruch. Na obrázku 2.6(b,c) je struktura pro výpočet jednoho bitu sčítání a přenosu. V uvedené struktuře může nastat celkem 36 poruch typu *t0/t1*. Odstraněním ekvivalentních poruch získáme množinu 27 poruch, pro jejichž detekci budeme potřebovat maximálně 27 testovacích vzorů. Test celé 64 bitové sčítačky tak může být proveden v $64 \cdot 27 = 1728$ krocích.

Je tedy zřejmé, že úplné funkční testy přicházejí v úvahu pouze pro jednoduché číslicové obvody. Pro složitější obvody je potřeba použít strukturní testy. V praxi se můžeme setkat s tím, že je použita určitá podmnožina funkčních testovacích vektorů, která je pro dosažení požadovaného pokrytí doplněna testovacími vektory vygenerovanými pomocí generátoru strukturního testu.

2.3.2 Hierarchický test

Další možností je zabývat se vytvářením testu na vyšší úrovni popisu obvodu. Modelováním obvodu na vyšší úrovni popisu se redukuje počet prvků v testovaném obvodu a celý problém je snadněji řešitelný. Problémem přístupu založeného na vytváření testu na vyšší úrovni popisu je však nižší pokrytí poruch dané tím, že nemáme pro generování testu k dispozici informace o vnitřní struktuře jednotlivých prvků.

Kombinaci generování testu na více úrovních popisu obvodu představuje tzv. hierarchický test [53, 64, 65, 86]. Princip hierarchického testu spočívá v tom, že pracuje s obvodem na více úrovních abstrakce. Hierarchické vytváření testu je možné vyjádřit jako posloupnost následujících činností [86]:

1) Rozčlenění obvodu na moduly – rozčlenění obvodu na moduly bývá z velké části určeno již samotným stylem návrhu obvodu a/nebo, zejména v případě návrhu obvodu popisem jeho chování, bývá provedeno v rámci procesu syntézy obvodu. Jednotlivé moduly mohou představovat například základní prvky dané úrovně nebo určitou část obvodu, která představuje vhodně propojené prvky obvodu. Složitost jednotlivých modulů se obvykle volí tak, aby bylo možné vygenerovat test daného modulu na nižší úrovni popisu.

2) Vytvoření testu pro jednotlivé moduly – generování lokálního testu pro jednotlivé moduly obvykle probíhá na úrovni hradel s využitím existujících nástrojů pro generování testu sekvenčního obvodu. Výsledkem je sada lokálních testovacích vektorů a odezev na tyto testovací vektory, které je potřeba transformovat do globálního testu obvodu.

3) Identifikace transparentních cest – pro každý modul je potřeba identifikovat transparentní cesty, které je možné použít pro přenos lokálních testovacích vektorů z primárních vstupů obvodu na vstupy testovaného modulu a přenosu odezev z výstupu testovaného modulu na primární výstupy obvodu. Umožňuje-li to struktura daného obvodu, pak je pro každý modul po skončení identifikace transparentních cest známo, jakou posloupností hodnot generovanou na primárních vstupech obvodu lze zajistit výskyt požadovaných dat (lokálních testovacích vektorů) na vstupech testovaného modulu resp. jakou posloupností lze zajistit pozorování dat (odezev) z výstupu modulu.

4) Vytvoření globálního testu obvodu – posledním krokem je generování globálních testovacích vektorů zajišťujících test všech modulů obvodu. Při jejich vytváření je potřeba zohlednit možné omezující podmínky např. ve formě maximální doby trvání testu nebo maximálního odběru testovaného systému. Na základě těchto omezujících podmínek pak probíhá vytváření globálních testovacích vektorů obvykle s ohledem na maximální využití paralelního testování jednotlivých modulů.

Realizace hierarchického generování testu je však prakticky vždy spojena s jistou (oproti nižší úrovni popisu ještě vyšší) abstrakcí od cílové obvodové struktury; proto současné metody pro hierarchické generování testu kvalitativně nedosahují výsledku nízkoúrovňových metod. Příčinou těchto nedostatků a omezení je zejména tzv. koncepce transparentnosti, z níž tyto metody obvykle vycházejí a která znamená vnesení dalších abstrakcí.

2.4 Automatické generování testu – nástroje ATPG

Složitost číslicových obvodů neustále roste. Kromě samotného počtu tranzistorů na čipu roste také hodinový kmitočet, na kterém čipy pracují, roste počet vývodů pouzder a roste také složitost struktur obsažených v čipu (paměti, rozhraní sběrnic, . . .). Rostou také požadavky na kvalitu a spolehlivost, zkracuje se doba potřebná pro uvedení čipu na trh (angl. time-to-market) a samozřejmě se požaduje nízká cena výsledného čipu. Náklady na testování tvoří nezanedbatelnou část nákladů na vyrobení číslicového obvodu. U vestavěných systémů mohou podle [21] tyto náklady představovat až 50% celkových výrobních nákladů. Vzhledem ke snaze snížit náklady na výrobu na minimum, je snaha snížit také náklady na testování na co nejnižší možnou míru.

Dříve bylo potřeba několik měsíců (např. [17] uvádí 8 až 18 měsíců) pro ruční generování testovacích vektorů tak, aby bylo dosaženo potřebných hodnot pokrytí poruch. Použitím automatizovaných nástrojů pro generování testu jsme schopni tuto dobu zkrátit do řádu dnů až týdnů. Dalším důvodem, proč je v dnešní době generování testovacích vektorů realizováno automatizovanými nástroji, je efektivita vytvořených testů. Automatizovaně vytvořené testy dosahují lepších parametrů co se týká pokrytí poruch a pokrytí poruch za hodinový takt [11]. Pro dosažení stejného pokrytí poruch tak potřebujeme méně testovacích vektorů a získáváme kratší dobu potřebnou pro provedení celého testu. Nevýhodou použití ATPG nástrojů je, že je potřeba spravovat samotný ATPG nástroj a knihovnu prvků, se kterými tento nástroj pracuje. Do jisté míry se při použití ATPG nástroje také stáváme závislí na dodavateli ATPG nástroje a jeho podpoře.

2.4.1 ATPG pro kombinační obvody

Proces generování testovacích vektorů se skládá z několika dílčích kroků. Předpokládejme, že máme k dispozici seznam poruch, které má vytvořený test detekovat. Prvním krokem vytváření testu je *výběr poruchy*, pro jejíž detekci budeme vytvářet testovací vektor. Volba vhodného pořadí, ve kterém budeme jednotlivé poruchy vybírat, má poměrně velký vliv na dobu potřebnou pro generování testu. Podle [17] může vést vhodné pořadí výběru poruch až na poloviční dobu potřebnou pro generování testu. Při procesu výběru poruch je potřeba zohlednit, že nastavení detekční cesty představuje náročnější úlohu než nastavení cesty, která poruchu

aktivuje. Doporučená strategie je tedy volit nejprve poruchy co nejblíže k primárním výstupům obvodu.

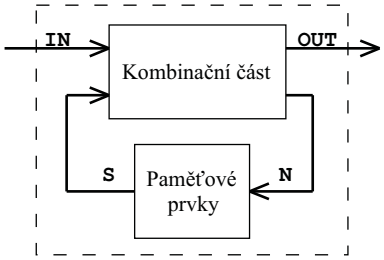
Dalším krokem je *aktivace poruchy*. Aktivace poruchy znamená nastavení místa s poruchou na opačnou hodnotu než představuje porucha, kterou chceme detekovat. Pokud například chceme aktivovat poruchu *t0* v místě *a*, znamená to nastavit hodnotu v místě *a* na log. 1. Nastavení této hodnoty v daném místě obvodu znamená zpětné procházení struktury obvodu a hledání kombinace vstupních hodnot, které požadovanou poruchu aktivují.

Třetím krokem je *sestavení detekční cesty*, která umožňuje detekovat poruchu na primárních výstupech obvodu. Představuje to nastavení prvků na cestě mezi místem s aktivovanou poruchou a vybraným primárním výstupem do transparentního režimu. Obecně může být potřeba řešit také konflikty mezi cestou aktivující poruchu a cestou, která ji detekuje.

Existuje řada algoritmů pro generování kombinačních testovacích vektorů pro poruchy typu *t0/t1*. Základními algoritmy jsou například D-algoritmus a PODEM algoritmus. Vzhledem k tomu, že tato práce není zaměřena na generování testovacích vektorů, nebudou jednotlivé algoritmy popisovány. Podrobné informace o algoritmech pro generování testu kombinačních obvodů je možné nalézt např. v [11]. Obecně je možné říci, že problém automatického generování testovacích vektorů pro kombinační obvody je při použití modelu poruch typu *t0/t1* považován od konce osmdesátých let za vyřešený [53, 82, 97].

2.4.2 ATPG pro sekvenční obvody

Obtížnější úlohu představuje generování testovacích vektorů pro sekvenční obvody. Je to dáno tím, že výstup sekvenčního obvodu je dán nejen kombinací vstupních hodnot, ale také aktuálním stavem obvodu, který je definován obsahem jednotlivých paměťových prvků obvodu (viz model sekvenčního obvodu na obrázku 2.7).



Obr. 2.7: Model sekvenčního obvodu.

Bylo dokázáno, že vytváření testu pro sekvenční obvody představuje NP-úplný problém [29]. Velké výzkumné úsilí bylo věnováno otázce vytváření testu sekvenčních obvodů na úrovni hradel. Vytváření testu na této úrovni vedlo ke kvalitním testům, ale výpočetní složitost vytváření těchto testů byla velmi vysoká a díky tomu pro obvody v řádu statisíců hradel nepoužitelná. I přes rostoucí výpočetní sílu současných počítačů je vytváření testu složitých sekvenčních obvodů na úrovni hradel stále nevyřešenou otázkou. Metody generování testu sekvenčních obvodů jsou v praxi limitovány pouze na „jednoduché“ sekvenční obvody.

Jedním z řešení tohoto problému je použití některé z technik pro snadnou testovatelnost (angl. Design for Testability – DFT) – např. techniky úplný scan, která je založena na modifikaci paměťových prvků obvodu tak, aby bylo možné řídit a pozorovat hodnoty uložené v těchto prvcích prostřednictvím primárních vstupů a výstupů obvodu. Problematika generování testu

sekvenčního obvodu se nám tak při použití metody úplný scan redukuje na problém generování testu kombinačního obvodu. Nevýhodou techniky úplný scan je však nárůst složitosti obvodu a změna dynamických parametrů obvodu. Možným řešením je použití techniky částečný scan, kdy je modifikována pouze část paměťových buněk.

2.5 Analýza testovatelnosti číslicových obvodů

Pro návrháře číslicového obvodu je velmi důležité mít možnost zjistit jak obtížné bude vytvořit test pro jím navržený obvod, popřípadě přímo identifikovat obtížně testovatelné části obvodu. Pro tyto účely se používají *metody analýzy testovatelnosti*. Jejich výhodou je nižší časová složitost oproti metodám generování testu (obvykle lineární vzhledem k počtu prvků a spojů obvodu). Metody analýzy testovatelnosti tak umožňují návrháři poměrně rychle identifikovat problematické části obvodu a případně také nabídnout možné řešení pro zlepšení testovatelnosti dané části obvodu.

Obecně přijímaná definice testovatelnosti bohužel v dnešní době neexistuje. Existuje však řada dílčích, konkrétně aplikačně orientovaných definic a z nich vycházejících metod, z nichž každá je obvykle konstruována pro jistou konkrétní úroveň popisu obvodu a pro zohlednění vybrané podmnožiny nákladů spojených s jeho testováním [86]. Nejednotnost a mnohdy značná rozdílnost přístupů zabývajících se problematikou testovatelnosti byla podnětem pro standardizaci pojmů z této oblasti. V roce 1998 byla sestavena standardizační skupina zabývající se problematikou diagnostiky elektronických systémů pracující na standardu IEEE P1522 [1, 85]. Ten se týká standardizace pojmů, zejména měr a vlastností, z oblasti testovatelnosti a diagnostiky elektronických systémů. Standardizační proces byl ukončen v roce 2004. Bohužel zatím nenašel širší uplatnění.

Přístup použitý v této práci pro ohodnocení testovatelnosti obvodu je založen na ohodnocení parametrů *řiditelnosti* a *pozorovatelnosti* obvodu. Řiditelnost resp. pozorovatelnost je chápána jako schopnost ovlivnit resp. změřit hodnotu signálu v daném místě v obvodu. Je snahou řiditelnost resp. pozorovatelnost číselně ohodnotit. Hodnota řiditelnosti resp. pozorovatelnosti pak obvykle vyjadřuje míru snadnosti nastavení resp. zjištění hodnoty signálu v daném místě obvodu [3]. Podrobněji bude metoda analýzy testovatelnosti použitá v této práci představena v kapitole 6.

2.6 Optimalizační metody

Optimalizační metody umožňují řešit tzv. optimalizační problém, který můžeme formulovat jako problém minimalizace účelové funkce $f_0(x)$ při splnění podmínek $f_i(x) \leq b_i$, $i = 1, \ldots, m$ [7]. V této formulaci je vektor $x = (x_1, \ldots, x_n)$ optimalizovaná proměnná, $f_0 : \mathbb{R}^n \to \mathbb{R}$ představuje účelovou funkci, $f_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m$ jsou omezující funkce a $b_1, \ldots, b_m$ omezující konstanty (hranice). Vektor x^* se nazývá optimální nebo také řešení problému, pokud splňuje omezující podmínky a pokud hodnota účelové funkce je pro tento vektor minimální – tedy pro libovolné $z \in \mathbb{R}^n$ splňující omezující podmínky $f_1(z) \leq b_1, \ldots, f_m(z) \leq b_m$ platí, že $f_0(z) \geq f_0(x^*)$.

Pro účely této práce jsou zajímavé zejména metody kombinatorické optimalizace. Jedná se o metody určené pro řešení problémů, u nichž je množina potenciálních řešení diskrétní nebo může být na takovou množinu redukována. Příkladem typického kombinatorického problému

je například problém obchodního cestujícího, hledání minimální kostry grafu, úloha osmi dam, apod. Pro řešení těchto problémů se obvykle používají heuristické prohledávací metody. Například horolezecký algoritmus, simulované žíhání, evoluční algoritmy, zakázané prohledávání, mravenčí kolonie, atd. V následující části bude podrobněji popsán pouze princip využití evolučních algoritmů a simulovaného žíhání, které jsou v této práci dále použity. Podrobnější informace o dalších optimalizačních metodách je možné nalézt například v [61].

2.6.1 Evoluční algoritmy

Evoluční algoritmy jsou stochastické iterační algoritmy, které využívají modely základních mechanismů evoluce živé hmoty (reprodukce, mutace a křížení) pro účely optimalizace [4, 61]. Evoluční algoritmy jsou založeny na formalizaci Darwinovy evoluční teorie. Základem Darwinovy evoluční teorie je teorie přirozeného výběru. Silnější jedinci (lépe přizpůsobení prostředí) mají v přírodě v další generaci více potomků než slabší jedinci a reprodukcí dvou jedinců s kvalitní genetickou výbavou získáme s velkou pravděpodobností opět jedince, kteří budou dobře přizpůsobeni danému prostředí.

U evolučních algoritmů je každý jedinec zakódován ve formě chromozomů – lineárních řetězců symbolů (nejčastěji binárních). Chromozomy tvoří populaci kandidátních řešení. Kvalita jedince je ohodnocena pomocí účelové funkce (v některých zdrojích se můžeme setkat s pojmy hodnotící funkce, popř. fitness funknce). Výsledkem ohodnocení jedince je obvykle reálné číslo z množiny $\mathbb{R}_{\langle 0,1 \rangle}$, kde nižší ohodnocení (předpokládáme-li minimalizační problém) znamená kvalitnějšího jedince. Výběr jedinců, kteří se účastní reprodukce, je realizován pseudonáhodně tak, že kvalitnější jedinci mají větší naději účastnit se reprodukce, během níž si vymění část své genetické informace. Samotná reprodukce však není dostatečně efektivní pro vznik dobře přizpůsobených jedinců, a proto je nutné zapojit mutace, které náhodně (s určitou pravděpodobností) ovlivňují genetický materiál populace. Evoluční proces produkuje posloupnost populací a probíhá tak dlouho, dokud není splněna ukončující podmínka, která je obvykle vyjádřena počtem generací, popř. požadavky na výsledné řešení.

Existuje mnoho variant evolučních algoritmů, které se liší např. přístupem k zakódování kandidátního řešení, implementací procesu reprodukce, nahrazení rodičovské populace, apod. Různé varianty evolučních algoritmů jsou popsány např. v [6]. V této práci se zaměříme na speciální případ evolučního algoritmu – *evoluční programování*.

2.6.2 Evoluční programování

Evoluční programování (angl. Evolutionary Programming – EP) [4,6,26,28] navrhl Lawrence Fogel v roce 1960. Specifické pro evoluční programování je, že kandidátní řešení (jedinec) není zakódováno ve formě binárního vektoru (jako např. u genetického algoritmu), ale evoluce probíhá přímo nad reprezentací daného problému. Počáteční populace je obvykle vytvářena náhodně. Rodiče, kteří se účastní reprodukce, jsou vybráni pomocí tzv. turnajového výběru. Princip turnajového výběru spočívá v tom, že je z populace náhodně vybráno n jedinců, kde n je typicky 2 nebo 3. Vybraní jedinci spolu po dvojicích soupeří a vítězí jedinec, který má nejvyšší fitness. Další typickou vlastností EP je absence operátoru křížení. Potomci jsou vytvářeni pouze aplikací operátoru mutace na vybraného rodiče. Během každé generace jsou vytvářeni noví jedinci pomocí operátoru mutace a tito jsou přidáni do populace rodičů. Dosažením dvojnásobné velikosti populace se aktivuje proces nahrazení, který je realizován odstraněním

poloviny nejslabších jedinců. Proces evoluce se pak znovu opakuje, dokud není dosažen zadaný počet generací nebo není nalezeno požadované řešení. Kromě této původní varianty evolučního programování existuje také tzv. *průběžná varianta* [26], kde nový jedinec jednoduše nahrazuje nejslabšího jedince původní populace. Základní varianta EP je popsána na obrázku 2.8.

```
Vytvoř populaci P o N náhodně vytvořených jedincích
Ohodnoť kvalitu jedinců v populaci P
while (nebylo nalezeno akceptovatelné řešení or
       nebylo dosaženo maximálního počtu generací) do {
  while (|P| < 2*N) do {
    x := vyber rodiče za pomoci turnaje z populace P
    x':= vytvoř kopii rodiče x
    y := aplikuj operátor mutace na x'
    Ohodnoť kvalitu jedince y
    Přidej y do populace P
  }
  Odstraň N nejslabších jedinců z populace P
}
```

Obr. 2.8: Základní varianta evolučního programování.

2.6.3 Simulované žíhání

Simulované žíhání (angl. Simulated Annealing - SA) představuje stochastický optimalizační algoritmus, který je založen na analogii mezi žíháním tuhých těles a problémů optimalizace. Tento algoritmus navrhli Kirkpatrick, Gelatt a Vecchi v roce 1983 [57] a nezávisle na nich pak Černý v roce 1985 [23].

Název a inspirace pochází z procesu žíhání tuhého tělesa, techniky spočívající v zahřátí a kontrolovaném ochlazování tuhého tělesa, při němž dochází k odstranění vnitřních defektů a pnutí. Ohřátí tělesa na vysokou teplotu umožní jeho atomům překonat lokální energetické bariéry a tím se dostávají do rovnovážných poloh. Postupné snižování teploty tělesa má za důsledek, že se rovnovážné polohy atomů fixují. Při konečné teplotě žíhaní (podstatně nižší než byla počáteční teplota) jsou atomy v rovnovážných polohách a těleso neobsahuje vnitřní defekty a pnutí. Podrobnější informace o nahrazení fyzikální realizace simulovaného žíhání jeho numerickou simulací je možné nalézt například v práci [61]. Pro naše účely bude postačovat představit si pouze základní princip metody simulovaného žíhání.

Základem simulovaného žíhání je Metropolisův algoritmus. Metropolisův algoritmus sestává z iteračního procesu, který funguje následovně: nechť x je aktuální stav systému a tento stav se poruší na nový stav y. Akceptace nového stavu y se řeší pomocí tzv. Metropolisova kritéria (viz vztah 2.2), které určuje pravděpodobnost nahrazení starého stavu novým

$$P(x, y, T) = \begin{cases} 1 & \text{pokud} f(y) \leq f(x), \\ e^{-\frac{f(y)-f(x)}{T}} & \text{jinak.} \end{cases} , \tag{2.2}$$

kde T je parametr interpretovaný jako teplota, $f(*)$ je hodnotící funkce a hledáme stav s minimálním ohodnocením.

V případě, že nový stav y má menší nebo stejnou funkční hodnotu jako původní stav x (y představuje lepší nebo stejně kvalitní řešení), potom vykonáme nahrazení stavu x stavem y. V opačném případě je nový stav y akceptovaný s pravděpodobností $0 < P(x, y, T) < 1$. Hodnota parametru T (teploty) podstatně ovlivňuje pravděpodobnost $P(x, y, T)$ pro případ, že $f(y) > f(x)$. Pro velké hodnoty T je tato pravděpodobnost blízká jedné (tzn. akceptují se téměř všechny nové stavy). Pokud se ale T snižuje, tak se také snižuje pravděpodobnost akceptování horšího stavu. Příklad implementace Metropolisova algoritmu je na obrázku 2.9.

```
procedure Metropolisův_algoritmus(var x; T, kmax) {
  k:=0;
  while (k < kmax) do {
    y:=mutace(x);
    P:=min(1, exp(-(f(y)-f(x))/T));
    if (random < P) then x:=y;
    k:=k+1;
  }
}
```

Obr. 2.9: Implementace Metropolisova algoritmu.

Vlastní simulované žíhaní je pouze opakované použití Metropolisova algoritmu pro posloupnost klesajících teplot, přičemž konečný stav x Metropolisova algoritmu pro teplotu T je použit jako počáteční stav pro Metropolisův algoritmus na nové teplotě, která je nižší než teplota předcházející. Formálně $T = T \cdot \alpha$, kde $0 << \alpha < 1$. Příklad implementace metody simulovaného žíhání je na obrázku 2.10.

```
procedure Simulované_žíhání(var x; Tmin, Tmax, kmax, alfa) {
  x:=náhodně vygenerovaný stav;
  T:=Tmax;
  while (T > Tmin) do {
    Metropolisův_algoritmus(x, T, kmax);
    T:=alfa*T;
  }
}
```

Obr. 2.10: Základní varianta algoritmu simulovaného žíhání.

Simulovaného žíhání pak můžeme pro účely optimalizace použít následovně. Mějme stavový prostor možných řešení a hodnotící funkci $f(x)$ reprezentující kvalitu řešení x. Z fyzikálního hlediska se jedná o množinu možných vnitřních stavů tělesa a obdobu vnitřní energie tělesa. Předpokládejme, že nižší hodnota $f(x)$ značí lepší řešení (těleso s nižší vnitřní energií). Výsledkem fyzikální realizace simulovaného žíhání je stav tělesa s nejnižší energií. Pokud provedeme numerickou simulací tohoto procesu získáváme řešení našeho optimalizačního problému.

2.7 Shrnutí

Cílem této kapitoly bylo seznámit čtenáře se základními pojmy a metodami z oblasti modelování číslicových obvodů, diagnostiky a optimalizačních technik. Na začátku této kapitoly byly krátce popsány úrovně modelování číslicových obvodů. Z hlediska dalších částí této práce je zajímavá zejména úroveň meziregistrových přenosů, která je použita pro modelování číslicového obvodu v této práci. Představeny byly také základní pojmy z oblasti diagnostiky číslicových systémů – modelování poruch, vytváření funkčních a strukturních testů a automatického generování testu. Tyto informace budou využity v další části této práce při návrhu metody analýzy testovatelnosti číslicového obvodu a také při experimentálním ověření navržené metody. Vzhledem k tomu, že jsou pro návrh syntetických testovacích obvodů v této práci použity optimalizační algoritmy, byl závěr kapitoly věnován krátkému popisu principu optimalizačních metod evolučního programování a simulovaného žíhání.

Kapitola 3

Aktuální stav řešené oblasti

Cílem této kapitoly je představit čtenáři aktuální stav v oblastech souvisejících s tématem této práce. Jedná se zejména o oblasti týkající se testovacích obvodů určených pro ověřování diagnostických metod a nástrojů, metod syntetického vytváření testovacích obvodů, metod analýzy testovatelnosti číslicových obvodů na úrovni RT a evolučního návrhu číslicových obvodů.

3.1 Existující sady testovacích obvodů

Sady testovacích obvodů (angl. benchmark sets) jsou používány již mnoho let pro ověření a „objektivní“ srovnání metod a nástrojů. Typicky se testovací sada skládá z množiny obvodů, které byly vybrány tak, aby reprezentovaly určitou cílovou doménu testovaných metod a nástrojů. V principu platí, že pokud budou všichni využívat stejnou sadu testovacích obvodů, tak bude snadné jednotlivé nástroje vzájemně porovnat. V praxi to však tak jednoduché není [44]. Typicky se například můžeme setkat s tím, že jsou výsledky publikovány pouze pro část testovacích obvodů z dané sady. Důvodem může být například to, že výsledky pro jiné obvody z této sady nejsou pro autora příliš příznivé. Dalším problémem při srovnání výsledků jsou různé podmínky, za kterých byly dané výsledky dosaženy (použité výpočetní prostředky, aplikované předpoklady/omezení, . . .). Je potřeba si také uvědomit, že testovací obvody nemusí odpovídat reálným obvodům, protože reálné obvody mohou obsahovat konstrukce, které testovací obvody neobsahují. Dalším problémem při porovnání metod je škálovatelnost. Výsledek dosažený na obvodu, který má 1 000 prvků neříká nic o tom, jaké výsledky poskytne nástroj pro obvod, který má 100 000 prvků.

Existující testovací obvody můžeme rozdělit do různých kategorií například podle účelu, pro který jsou určeny nebo úrovně popisu. Jiné testovací obvody jsou potřeba pro ověřování syntézních algoritmů[1], které vyžadují obvody popsané chováním, zatímco propojovací algoritmy (angl. routing algorithms) mohou být testovány pouze na testovacích obvodech popsaných na nižších úrovních popisu. Vzhledem k tomu, že tato práce se zabývá testovacími obvody pro ověřování metod a nástrojů z oblasti diagnostiky, budou krátce představeny sady testovacích obvodů určené pro ověření a srovnání právě těchto metod.

[1] Syntézní algoritmy (angl. synthesis algorithms) umožňují překlad obvodu popsaného na vyšší úrovni popisu (např. úrovni chování) na obvod popsaný na nižší úrovni popisu (obvykle úroveň hradel nebo technologických primitiv dané technologické platformy).

V současné době existuje relativně velké množství testovacích obvodů určených pro ověřování diagnostických metod a nástrojů [9, 10, 16, 19, 101–103]. Nejčastěji používanými [44] jsou obvody z testovacích sad ISCAS, ITC'99 a ITC'02, které budou podrobněji popsány v následující části.

3.1.1 Testovací sady ISCAS'85 a ISCAS'89

Sady testovacích obvodů ISCAS'85 [10] a ISCAS'89 [9] patří mezi nejznámnější sady testovacích obvodů. I přesto, že se jedná o jedny z nejstarších sad, patří stále mezi nejpoužívanější. Jejich charakter tyto obvody předurčuje pro porovnání metod a nástrojů z oblasti testování a diagnostiky číslicových systémů [44].

Sada ISCAS'85 byla představena v roce 1985 na konferenci International Symposium on Circuits and Systems (ISCAS) ve speciální sekci zaměřené na oblast generování testu a simulace poruch. Sadu tvoří 10 kombinačních obvodů, jejichž složitost se pohybuje od 160 do 3 512 hradel (viz příloha A, tabulka A.1). Obvody této složitosti již zdaleka nereprezentují dnešní číslicové obvody. Jejich existence však ve své době umožnila srovnání kombinačních ATPG algoritmů a podnítila další výzkum v této oblasti [30]. Pro dokreslení „aktuálnosti“ této testovací sady poznamenejme, že její testovací obvody byly distribuovány účastníkům konference ISCAS'85 na magnetofonových páskách. Nyní jsou k dispozici prostřednictvím internetu (např. [101])

Na základě úspěchu sady ISCAS'89 byla v roce 1989 představena navazující sada 31 sekvenčních číslicových obvodů (viz příloha A, tabulka A.2), jejichž složitost se pohybuje v řádu od 10 hradel a 3 klopných obvodů typu D po 22 179 hradel a 1 636 klopných obvodů typu D [9]. Existence těchto testovacích obvodů pak ve své době přinesla možnost vzájemného porovnání nových sekvenčních ATPG algoritmů a vedla tak opět k rozvoji v této oblasti.

V roce 1993 byla sada ISCAS'89 doplněna o dalších 14 sekvenčních obvodů označovaných jako Addendum'93 (viz příloha A, tabulka A.3) [35, 36] určených pro ověřování sekvenčních generátorů testu. Složitost těchto obvodů se pohybuje v řádu od 160 hradel a 15 klopných obvodů typu D po 3 080 hradel a 239 klopných obvodů typu D.

Hlavní nevýhodou ISCAS testovacích obvodů je v současné době jejich zastaralost. Největší obvod z této trojice testovacích sad se skládá pouze z 22 179 hradel a 1 636 klopných obvodů typu D. Dalším omezením těchto testovacích obvodů je, že jsou popsány pouze na úrovni hradel. Návrh číslicových systémů je však v dnešní době již obvykle realizován na vyšších úrovních popisu. Vznikají také metody analýzy testovatelnosti a generování testu, které pracují na vyšších úrovních popisu. Bohužel pro ověření těchto metod obvody ze sady ISCAS není možné použít.

Aby bylo možné ISCAS testovací obvody alespoň částečně použít také pro ověřování obvodů na vyšší úrovni popisu, byly v roce 1999 pomocí reverzního inženýrství vytvořeny modely vybraných ISCAS testovacích obvodů na úrovni RT [43].

3.1.2 Testovací sada ITC'99

Na konferenci ITC (International Test Conference) v roce 1999 byla představena nová sada testovacích obvodů ITC'99. Hlavním cílem autorů této sady bylo překonat nedostatky ISCAS testovacích obvodů – poskytnout obvody vyšší složitosti popsané na různých úrovních abstrakce.

Původní verzi této testovací sady tvořilo 21 obvodů, které byly popsány na úrovni RT prostřednictvím syntetizovatelného jazyka VHDL a na úrovni hradel ve formě formátu EDIF a ISCAS'89 *BENCH* formátu. Jejich složitost se pohybuje od 28 hradel a 4 klopných obvodů po 68 752 hradel a 3 320 klopných obvodů (viz příloha A, tabulka A.4). Všechny obvody této sady jsou plně synchronní, mají pouze jednu hodinovou doménu, nemají vnitřní paměti, nepoužívají třístavové vodiče a nepoužívají logiku wire-OR/AND.

Původní sada ITC'99 byla revidována v roce 2002. Do sady byly přidány nové obvody a byly vytvořeny optimalizované varianty obvodů na úrovni hradel. Revidované vydání sady ITC'99 tvoří celkem 31 obvodů: 3 reálné obvody, 2 obvody z akademické sféry, jeden kombinační obvod, 22 obvodů původní sady ITC'99 popsaných na úrovni RT a 3 obvody ze sady ISCAS'89 (viz příloha A, tabulka A.5). Část z těchto obvodů je dostupná volně, pro získání všech testovacích obvodů je potřeba podepsání tzv. Community Source License [19]. Bohužel ale ani obvody revidované sady ITC'99 stále nereprezentují obvody dnešní složitosti. Největší obvod se skládá z 231 320 hradel a 6 642 klopných obvodů typu D.

3.1.3 Testovací sada ITC'02 a další

Další sadou testovacích obvodů, která byla představena v roce 2002 na konferenci International Test Conference v roce 2002, je sada pojmenovaná ITC'02 (SoC benchmarks) [67]. Tato sada se skládá z 12ti obvodů popsaných na úrovni bloků, které jsou určeny pro ověření a porovnání metod a nástrojů modulárního testování (angl. modular testing) SoC obvodů. Sada je dostupná na adrese [103]. Tyto obvody mohou být například použity pro návrh struktur potřebných pro plánování testu jednotlivých bloků.

Kromě výše uvedených sad existuje také řada menších testovacích sad (např. Low Power Group benchmarks [104]) popř. samostatných testovacích obvodů (např. Diffeq, Tseng, Paulin, Bert, . . .). Problémem menších testovacích sad je, že výsledky získané na testovacích obvodech z těchto sad je obvykle obtížné použít po porovnání výsledků s jinými autory. Samostatné obvody pak zase dostatečně nepokrývají cílovou doménu, na které budou ověřované nástroje použity.

3.1.4 Shrnutí

Vzhledem k rostoucí složitosti číslicových obvodů je jejich návrh stále častěji realizován na vyšší úrovni popisu. Tento trend musí kopírovat také návrháři diagnostických metod, protože pro návrháře je důležité si již během návrhu na vyšší úrovni ověřit diagnostické vlastnosti jím navrženého obvodu. Vznikají tedy nové metody, které jsou schopny analyzovat diagnostické vlastnosti na vyšších úrovních popisu a zároveň vzniká také potřeba testovacích obvodů, které by bylo možné použít pro ověření nových metod. Vzhledem k tomu, že tato práce je zaměřena na číslicové obvody na úrovni RT, budeme dále předpokládat, že je návrh realizován právě na této úrovni popisu.

V současné době ale máme pro ověřování diagnostických metod a nástrojů na úrovni RT k dispozici pouze 31 relativně jednoduchých sekvenčních testovacích obvodů z druhého vydání sady ITC'99. Největší obvod dostupný v této sadě se skládá z 231 320 hradel a 6 642 klopných obvodů. Bohužel tato složitost již ale neodpovídá složitosti dnešních číslicových obvodů, která se pohybuje v řádu stovek miliónů hradel. V současné době tak **chybí testovací obvody na úrovni RT, které by svou složitostí odpovídaly dnešním číslicovým obvodům**.

Jak vyřešit nedostatek testovacích obvodů na úrovni RT? Jednou z možností je použití existujících obvodů. Tady však narážíme na problém jak získat zdrojový kód těchto obvodů, který je obvykle chráněn licenční politikou dané instituce a která obvykle nedovolí volné šíření zdrojového kódu obvodu. Další možností je využití tzv. open-source obvodů. Tyto obvody však většinou nedosahují potřebné složitosti, popřípadě nejsou vhodné pro porovnání diagnostických metod a nástrojů, protože nedosahují potřebného rozsahu parametrů testovatelnosti (pro účely ověřování metod testovatelnosti je například vhodné mít obvody s různou mírou testovatelnosti).

Podobný problém s nedostatkem testovacích obvodů bylo potřeba řešit například také při ověřování metod a nástrojů z oblasti automatického rozmístění prvků v FPGA [18] nebo testování optimalizátorů logických výrazů [52]. Kvůli nedostatku testovacích obvodů potřebné složitosti nebylo možné ověřovat nově navržené metody z těchto oblastí. Řešením uvedených problémů bylo použití tzv. syntetických testovacích obvodů. Princip syntetických testovacích obvodů bude představen v následující podkapitole.

3.2 Syntetické testovací obvody

Syntetické testovací obvody jsou automatizovaným procesem vytvořené testovací obvody s požadovanými vlastnostmi, které jsou vhodné pro ověření konkrétní metody nebo algoritmu. Výhodou syntetických testovacích obvodů je, že uživatel má plnou kontrolu nad důležitými charakteristickými vlastnostmi vytvářených obvodů, jako je jejich velikost, struktura spojů, popř. funkce. V ideálním případě je možné jednotlivé parametry volit nezávisle na sobě a získáváme tak plnou kontrolu granularity generovaných obvodů. Hlavní nevýhodu syntetických testovacích obvodů je pak problematické prokázání jejich relace vůči reálným obvodům. Je to dáno způsobem návrhu, který je obvykle založen na splnění vlastností obvodu z hlediska jeho struktury a nikoliv funkce, jak je to typické pro běžný návrh.

Vzhledem k tomu, že cílem této práce je návrh metody pro vytváření syntetických testovacích obvodů s požadovanými diagnostickými vlastnostmi, bude v této podkapitole podrobněji shrnut aktuální stav této oblasti.

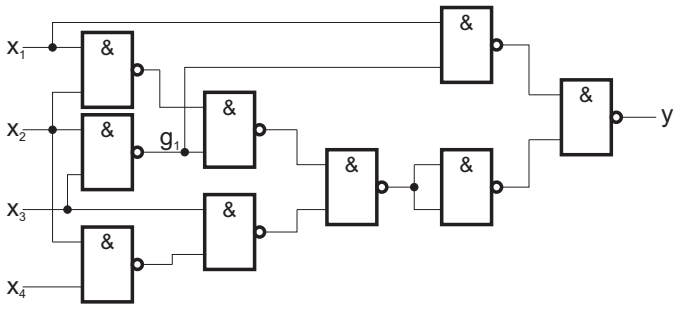
3.2.1 Metoda náhodných transformací

Jednou z prvních prací z oblasti syntetických testovacích obvodů byla práce Iwamy a Hina [51] zabývající se vytvářením syntetických testovacích obvodů pro testování logických optimalizátorů. Vstupem jimi navržené metody je libovolný kombinační obvod sestávající se z hradel typu NAND. Samotná metoda je založena na opakované aplikaci transformací, které nemění logickou funkci realizovanou obvodem, ale mění jeho strukturu. Množina transformací je navíc zvolena tak, že nekonečná posloupnost aplikací těchto transformací umožňuje prozkoumat celý stavový prostor obvodů realizujících ekvivalentní logickou funkci jako původní vstupní obvod. Navržená metoda tedy umožňuje vytvářet různě složité obvody realizují stejnou logickou funkci, které mohou být použity pro testování logických optimalizátorů.

Nevýhodou původní metody představené v práci [51] bylo, že v důsledku obecně zvolených transformačních pravidel nebyl omezen maximální počet vstupů hradel. To však neodpovídá praxi, kdy je počet vstupů hradel obvykle omezen. Další nevýhodou pak bylo, že bylo potřeba mít sadu vstupních obvodů, které tvořily základ generovaných testovacích obvodů.

Původní metodu návrhu rozšířil Iwama v práci [52]. Rozšíření spočívalo ve vytvoření nových parametrizovaných transformačních pravidel, pomocí nichž bylo možné omezit maximální

počet vstupů použitých hradel. Nová verze transformačních pravidel umožňuje transformovat vstupní obvod *C1* pomocí sekvence transformačních pravidel na výstupní obvod *C2* tak, že během této transformace meziprodukt vždy splňuje požadovaná kritéria z hlediska maximálního počtu vstupů jednotlivých hradel.



$$y = ((x_1, g_1), ((((x_1, x_2), g_1), (x_3, (x_4, x_2)))))$$
$$g_1 = (x_2, x_3)$$

Obr. 3.1: Ukázka obvodu a jeho reprezentace [52].

Struktura obvodu je v práci reprezentována výrazem, který tvoří *k*-tice reprezentující *k*-vstupová hradla NAND (viz obrázek 3.1). Na tyto k-tice pak můžeme aplikovat transformační pravidla. Příklad těchto pravidel pro $k = 2$ (2-vstupová hradla NAND) je na obrázku 3.2.

1) $(1) \Leftrightarrow 0$
2) $(0) \Leftrightarrow 1$
3) $(S_x, S_x) \Leftrightarrow (S_x)$
4) $(S_x, (S_x)) \Leftrightarrow 1$
5) $S_x, ((S_y, S_z)) \Leftrightarrow ((S_x, S_y)), S_z$
6) $S_x, S_y \Leftrightarrow S_y, S_x$
7) $(S_x, 1) \Leftrightarrow (S_x)$
8) $((S_x)) \Leftrightarrow S_x$
9) $(S_x, (S_y, S_z)) \Leftrightarrow (((S_x, (S_y)), (S_x, (S_z))))$
10) Pokud $g_i = f$ je definice podobvodu g_i potom $g_i \Leftrightarrow f$.
11) Pokud je g_i použitý symbol a nevyskytuje se na pravé straně definice žádného podobvodu ani obvodu, pak odstraň g_i.
12) Nechť g_i je dosud nepoužitý symbol a C je libovolný obvod skládající se z 2-vstupových hradel NAND.

Obr. 3.2: Sada transformačních pravidel pro hradla NAND a $k = 2$ [52].

Princip metody náhodných transformací spočívá v tom, že návrhová metoda na začátku automaticky navrhne obvod, který realizuje náhodně vygenerovanou logickou funkci. Na takto navržený obvod je následně aplikována sekvence transformací, které nemění logickou funkci realizovanou obvodem, ale mění složitost výsledné logické funkce. Autor ve své práci prezentuje vytvořené obvody o složitosti v řádu tisíců hradel. Omezením této metody je, že výsledný obvod je tvořen pouze jedním typem hradel a také není žádným způsobem omezena struktura obvodu (pomineme-li požadavek na *k*-vstupová hradla).

Zajímavé jsou experimentální výsledky získané při využití syntetických testovacích obvodů pro ověření dostupných optimalizátorů. Porovnávané optimalizátory, které dosahovaly na standardních testovacích obvodech srovnatelné výsledky, vykazují při aplikaci na syntetické

testovací obvody výsledky, které se liší až o 400% z hlediska počtu hradel ve výsledných obvodech. Autor také uvádí, že asi u 10% obvodů nebyly některé optimalizátory vůbec schopny vstupní výraz optimalizovat.

3.2.2 Metoda *RMC* (Random Mapped Circuits)

Jednou z dalších oblastí, která se potýkala s nedostatkem testovacích obvodů potřebné složitosti, byl návrh algoritmů pro rozmístění a propojení prvků (angl. Place and Route Algorithms) v FPGA (Field Programmable Gate Array). Problém nedostatku testovacích obvodů se pokusil vyřešit Darnauer a Dai pomocí metody *RMC* (z angl. Random Mapped Circuits) [18]. Jimi navržená metoda vytváří obvod s požadovanými strukturními vlastnostmi popsaný na úrovni základních prvků technologie Xilinx XC4000 (tzv. LUTů[2]). Vstupem metody jsou informace o požadovaném počtu logických členů (g), počtu vstupů všech logických členů (p), průměrném počtu vstupů jednotlivých LUTů (f), počtu primárních vstupů (i), počtu primárních výstupů (o) a hodnota Rentova kritéria (r)[3]. V této práci je Rentovo pravidlo chápáno jako vztah $(i + o) \approx (f + 1)g^r$.

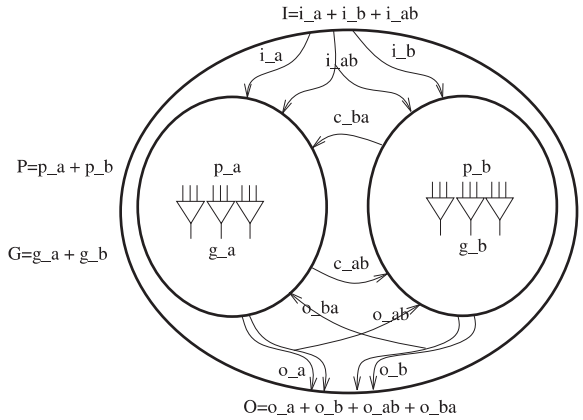
Pro návrh struktury obvodu je využita rekurzivní procedura, která na základě vstupních parametrů generuje požadovaný testovací obvod. Nejprve je ověřena realizovatelnost obvodu se zadanými parametry a dále následuje rekurzivní proces generování testovacího obvodu. Pro generování struktury obvodu máme k dispozici informaci o zadaném počtu logických členů, celkovém počtu vstupů, primárních vstupů a primárních výstupů. Uvedené prostředky jsou náhodně (existují jisté omezující pravidla) rozděleny na dvě části – nazvěme tyto části moduly A a B (viz obrázek 3.3). Každý modul se také skládá z daného počtu hradel (g_a, g_b) a těmto hradlům náleží daný počet vstupů (p_a, p_b). Každý modul má také své vstupy a výstupy (označme je i_a, i_b, i_{ab}, o_a, o_b, c_{ab}, c_{ba}, o_{ab}, o_{ba}), kde význam jednotlivých označení je následující (viz obrázek 3.3):

i_a, i_b	primární vstupy obvodu přivedené na vstupy modulů A a B,
i_{ab}	primární vstupy obvodu přivedené na vstupy obou modulů A a B,
o_a, o_b	výstupy modulů A a B přivedené na primární výstupy,
c_{ab}, c_{ba}	výstupy modulu $A(B)$ přivedené na vstupy modulu $B(A)$,
o_{ab}, o_{ba}	výstupy modulu $A(B)$ přivedené na vstupy modulu $B(A)$ a zároveň primární výstupy obvodu.

Samotný návrh obvodu s požadovanými parametry je realizován ve dvou krocích. V prvním kroku je navržena základní struktura obvodu a v druhém kroku je pak obvod modifikován tak, aby bylo splněno zadané Rentovo kritérium. Návrh struktury obvodu je realizován jako řešení následující soustavy 8 rovnic a 10 nerovnic o 13 neznámých (g_a, g_b, p_a, p_b, i_a, i_b, i_{ab}, o_a, o_b, o_{ab}, o_{ba}, c_{ab} a c_{ba}) [18]:

[2] LUT (Look-Up Table) základní stavební prvek architektury FPGA. LUT o *n* vstupech je paměť, která umožňuje realizovat libovolnou booleovu funkci *n* proměnných.

[3] Rentovo pravidlo $N_p = K_p \cdot {N_g}^{\beta}$ – empiricky získaný vztah vyjadřující vztah mezi počtem primárních vstupů/výstupů obvodu (N_p) a počtem logických členů (N_g). β představuje Rentovu konstantu a K_p poměrnou konstantu. Hodnota konstant byla pro vybrané typické systémy zjištěna experimentálně (viz [62]).



Obr. 3.3: Ukázka rozdělení obvodu na moduly, typy spojů mezi jednotlivými moduly [18].

$$
\begin{aligned}
i_a + i_{ab} + c_{ba} + o_{ba} &= I_a \\
i_b + i_{ba} + c_{ab} + o_{ab} &= I_b \\
o_a + o_{ab} + c_{ab} &= O_a \\
o_b + o_{ba} + c_{ba} &= O_b \\
g_a + g_b &= G \\
p_a + p_b &= P \\
i_a + i_b + i_{ab} &= I \\
o_a + o_b + o_{ab} + o_{ba} &= O
\end{aligned}
\qquad
\begin{aligned}
p_a \geq g_a,\ \ p_b \geq g_b \\
i_a + i_{ab} + c_{ba} + o_{ba} > 0 \\
i_b + i_{ab} + c_{ab} + o_{ab} > 0 \\
o_a + o_{ab} + c_{ab} > 0 \\
o_b + o_{ba} + c_{ba} > 0 \\
I_a + G_a \leq O_a + P_a \\
I_b + G_b \leq O_b + P_b \\
G_a(G_a - 1) + G_a O_a \geq O_a + P_a \\
G_b(G_b - 1) + G_b O_b \geq O_b + P_b
\end{aligned}
$$

Sada prvních osmi rovnic a šesti nerovnic zajišťuje, že každý modul má alespoň jeden vstup a výstup a tvoří jej alespoň jedno hradlo. Dále, že každá LUT použitá v modulu má alespoň jeden vstup a že každé hradlo, všechny dostupné vstupy prvků LUT a každý primární vstup a výstup modulu budou zapojeny. Sada posledních čtyřech nerovnic pak zajišťuje, že na každý spoj v modulu bude připojen právě jeden výstup a nejméně jeden vstup a že jedna LUT nebude využívat jeden spoj více než jednou (zamezení zpětné vazby v rámci jedné LUT). Druhým krokem je splnění Rentova kritéria r_a a r_b pro moduly A a B. Toho je dosaženo změnou počtu vzájemných propojení mezi moduly A a B (např. převedením části vstupů/výstupů z jednoho modulu na druhý).

Návrh obvodu je realizován jako rekurzivní proces. Pro každý modul následuje rekurzívní volání stejné procedury, která navrhuje strukturu daného modulu. Autor prezentuje použití metody pro návrh obvodů o velikosti stovek prvků. Nevýhodou metody *RMC* je, že navrhuje obvody s pravidelnou strukturou. Autor se také bohužel vůbec nezabývá relací s reálnými obvody. Z hlediska reálného použití navržených obvodů není například provedena kontrola maximálního počtu vstupů připojených na jeden výstup. Uvedená metoda je ale zajímavá z hlediska navrženého rekurzivního způsobu návrhu. Na tuto práci navazují další práce, které uvedené nedostatky odstraňují.

3.2.3 Metoda *GN1*

Stroobandt aj [87] představili metodu vytváření testovacích obvodů podobnou metodě *RMC*. Podobně jako u metody *RMC* má uživatel možnost specifikovat požadavky na vytvářený obvod ve formě počtu prvků, rozhraní prvků a požadovaného Rentova kritéria. Návrh obvodů je však u této metody realizován na principu zdola nahoru. Na základě zadaného Rentova kritéria jsou nejprve vytvářeny jednoduché moduly, s pomocí kterých se později vytvářejí složitější moduly, které tvoří samotný obvod.

Tuto původní myšlenku Stroobandt dále rozšířil v navazující práci [88], kde byla dříve navržená metoda rozšířena o možnost specifikovat konkrétní prvky, z nichž se má obvod skládat. Zajímavým způsobem byla také navržena eliminace zpětnovazebních smyček v rámci kombinační logiky. Pro každý vstup prvku je vytvořen seznam výstupů, které jsou ovlivněny prostřednictvím daného vstupu. Tento seznam se při vytváření spojů obvodu aktualizuje a zajišťuje tak, že není vytvořena nežádoucí zpětnovazební smyčka.

Stroobandtem navržená metoda umožňuje navrhovat obvody o složitosti až 10 000 prvků. Autor prokazuje kvalitu vytvořených obvodů na základě analýzy Rentova kritéria a distribuce větvení spojů obvodu, jejichž hodnoty korelují s hodnotami získanými pro reálné obvody. Problematická je ale redundance prvků obvodu, která se pohybuje pro některé obvody až na úrovni 99%.

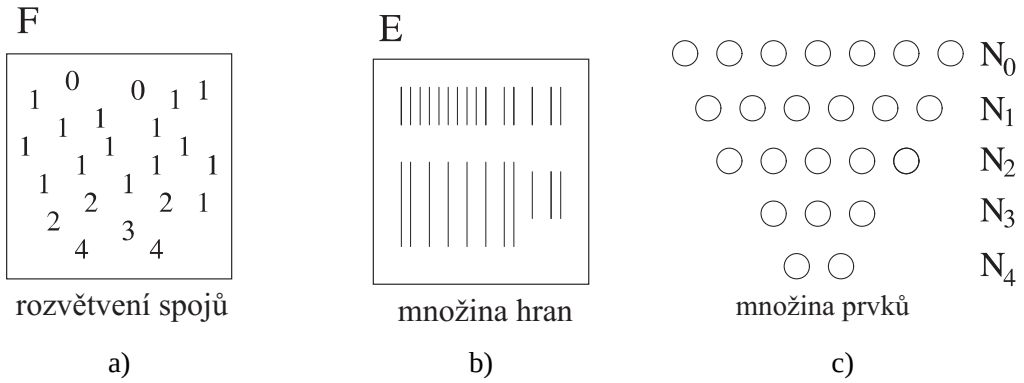
3.2.4 Metoda klonování existujících obvodů (*Circ&Gen*)

Hutton se ve své práci [48] zabýval návrhem metody, která by umožňovala navrhovat syntetické testovací obvody odpovídající reálným obvodům. Jimi navržená metoda je založena na „klonování“ existujících kombinačních obvodů. Vstupem metody je obvod, nad nímž je spuštěn charakterizační proces *Circ*. Výsledkem charakterizačního procesu je sada parametrů reprezentujících strukturu analyzovaného obvodu. Výsledek charakterizačního procesu pak využívá samotná návrhová metoda – generátor obvodů *Gen*. Cílem charakterizačního procesu je analyzovat následující parametry:

- **počet prvků obvodu (n), počet primárních vstupů (pi) a výstupů (po) obvodu**,
- **kombinační zpoždění obvodu** – kombinační zpoždění na úrovni hradla x je definováno délkou nejdelší orientované hrany z primárních vstupů obvodu na vstup hradla x. Kombinační zpoždění obvodu pak představuje nejdelší kombinační zpoždění v obvodu,
- **struktura obvodu** – počet hradel na jednotlivých úrovních kombinačního zpoždění,
- **histogram délek spojů v obvodu** – histogram popisující výskyt spojů v obvodu podle jejich délek, kde délka hrany je definována jako rozdíl kombinačních zpoždění na úrovni hradel, jež daný spoj propojuje,
- **histogram rozvětvení obvodu** – histogram počtu vstupů, které jsou připojeny k jednomu výstupu hradla.

Vstupem generátoru syntetických testovacích obvodů jsou výše uvedené informace, na základě kterých je vytvářena struktura testovacího obvodu. Problém generování testovacích obvodů si můžeme představit tak, že máme k dispozici určité stavební bloky (viz obrázek 3.4)

a s využitím těchto stavebních bloků vytváříme testovací obvody se stejnými charakteristickými vlastnostmi jako měl vzorový obvod. Analýzou struktury vzorového obvodu jsme získali informace o rozvětvení spojů obvodu (obrázek 3.4a), máme k dispozici množinu spojů různých délek (obrázek 3.4b) a množinu prvků, které jsou uspořádány podle kombinační hloubky (obrázek 3.4c).



Obr. 3.4: Znázornění zdrojů, které budou použity pro vytvoření testovacího obvodu.

Celý proces generování testovacích obvodů můžeme převést na problém přiřazení spojů z množiny E prvkům z množiny $N = N_1 \cup N_2 \cup \ldots$ (N_i je počet prvků na úrovni odpovídající kombinačnímu zpoždění i), při uplatnění informace o větvení spojů z množiny F. Hledáme takové přiřazení spojů $e \in E$ prvkům $n_1, n_2 \in N$, aby byly splněny následující podmínky:

1. počet výstupních hran prvku $x \in N$ je roven větvení $f_x \in F$,
2. každý prvek $x \in N_i$ má nejméně jeden spoj z úrovně N_{i-1} ($i > 0$),
3. počet vstupů prvku $x \leq k$ pro všechny $x \in N$,
4. výstup libovolného prvku $x \in N$ je připojen k nejvýše jednomu vstupu libovolného prvku $y \in N$.

Vzhledem k NP-časové složitosti analytického algoritmu návrhu takových testovacích obvodů je návrh realizován pomocí heuristických metod. Návrh obvodu skládajícího se z 30 000 prvků trvá 30 sekund [48]. Metoda umožňuje navrhovat kombinační testovací obvody o velikosti až 200 000 prvků.

Původní metodu *Circ&Gen* generující pouze kombinační obvody rozšířil Hutton v práci [49] o možnost generování sekvenčních testovacích obvodů. Přibyly nové charakterizační parametry, jako je sekvenční zpoždění prvku, rozložení sekvenčních prvků na jednotlivých úrovních, sekvenční hloubka obvodu, apod. Proces návrhu sekvenčních obvodů je realizován tak, že se nejprve vygenerují kombinační části, které jsou potom spojeny sekvenčními prvky tak, aby byly splněny charakteristické parametry vzorového obvodu. Složitější metoda návrhu bohužel omezuje složitost vytvářených obvodů – metoda je schopna navrhovat testovací obvody do velikosti 100 000 prvků.

3.2.5 Metoda *PartGen*

Další práci z oblasti syntetických testovacích obvodů představuje metoda *PartGen* [77] umožňující navrhovat obvody o stotisících prvcích v řádu jednotek minut vhodné pro testování tzv. „partitioning“ algoritmů – algoritmů, jejichž úkolem je rozdělit velký obvod na menší části, které by byly realizovatelné na dostupných hardwarových prostředcích. Návrhová metoda vychází z analýzy existujících reálných a testovacích obvodů. Na základě této analýzy konstatuje, že obvody tvoří z hlediska problematiky „partitioning“ algoritmů pět různých typů obvodových částí:

1. **kombinační logika s pravidelnou strukturou** (např. sčítačky, čítače, násobičky, . . .),
2. **kombinační logika s nepravidelnou strukturou**, která obvykle tvoří propojení jednotlivých bloků a realizuje např. generování jednoduchých řídicích signálů,
3. **kombinační a sekvenční logika** – pro tyto části obvodu je typický vysoký poměr primárních vstupů/výstupů obvodu k počtu spojů obvodu (např. řadič),
4. **paměťové bloky** (např. paměti RAM, datové cache) a
5. **propojení prvků**.

Navržený generátor *PartGen* se skládá z generátorů umožňujících generovat jednotlivé typy těchto bloků. Kombinační logika s pravidelnou strukturou je reprezentována násobičkou s požadovanou datovou šířkou. Kombinační logika s nepravidelnou strukturou je vytvářena pomocí nástroje *Circ&Gen* [48]. Bloky s kombinační a sekvenční logikou jsou generovány na podobném principu, který byl prezentován u metody *RMC* [18]. Paměťové bloky jsou realizovány parametrizovanými 32 bitovými pamětmi. Pro propojení jednotlivých prvků obvodu je využit podobný princip, který byl prezentován u metody *Circ&Gen* [48]. Samotná metoda *PartGen* pracuje tak, že na začátku výpočtem určí minimální a maximální počet primárních vstupů a výstupů. Skutečný počet je pak zvolen náhodně. Potom metoda hledá nejvhodnější propojení jednotlivých bloků a primárních bran obvodu.

Validace kvality vytvářených obvodů probíhá nepřímo pomocí tzv. „partitioning“ algoritmů, pro jejichž testování jsou vytvořené testovací obvody určeny. Obvod je považován za reálný, pokud je dosaženo aplikací různých typů partitioning algoritmů na odpovídající reálné a syntetické obvody podobných hodnot z hlediska zaplnění čipu[4] a využití pinů čipu[5].

3.2.6 Shrnutí

Syntetické testovací obvody jsou v některých oblastech úspěšně používány jako náhrada chybějících standardních testovacích obvodů. Možnost využití syntetických testovacích obvodů pro ověřování diagnostických metod a nástrojů se jeví jako zajímavá, ale bohužel z hlediska existujících metod jako nerealizovatelná z toho důvodu, že existující metody návrhu syntetických testovacích obvodů jsou obvykle založeny pouze na řízené modifikaci vybraného vzorového

[4] **Zaplnění čipu** – nechť n je počet FPGA potřebných pro implementaci obvodu o $|E|$ prvcích a k je počet prvků v FPGA. Potom $EU = |E|/(n \cdot k)$ vyjadřuje průměrné zaplnění FPGA.

[5] **Průměrné využití pinů čipu** – nechť n je počet FPGA potřebných pro implementaci obvodu, který má $|I|$ primárních vstupů a výstupů a c je počet pinů dostupných v FPGA. Potom $IU = |I|/(n \cdot c)$ vyjadřuje průměrné využití pinů FPGA.

obvodu, při které jsou sledovány určité charakteristické vlastnosti vytvářeného syntetického obvodu, případně jsou založeny na vytváření struktury obvodu při splnění pouze jednoduchých strukturálních vlastností. Dosud však neexistuje metoda návrhu, která by umožňovala navrhovat syntetické testovací obvody s tak komplexní charakteristickou vlastností jakou je například testovatelnost.

Na Ústavu počítačových systémů, Fakulty informačních technologií VUT v Brně se dlouhodobě zabýváme možnostmi využití evolučních algoritmů v oblasti diagnostiky. Jednou z možností použití evolučních algoritmů je automatický návrh číslicových obvodů – tzv. evoluční návrh. Použití evolučního návrhu v diagnostice není žádnou novinkou (viz např. [14, 31, 63, 68, 84, 92]). Dosud však neexistuje přístup, který by umožňoval navrhovat obvody s požadovanými diagnostickými vlastnostmi a složitostí odpovídající dnešním číslicovým obvodům. Cílem této práce je využít evoluční návrh pro vytváření syntetických testovacích obvodů s požadovanými diagnostickými vlastnostmi. Možnosti existujících metod evolučního návrhu jsou představeny v následující podkapitole.

3.3 Evoluční návrh číslicových obvodů a diagnostika

Evoluční algoritmy se v současné době začínají prosazovat i do oblasti návrhu číslicových obvodů – hovoříme o tzv. evolučním návrhu číslicových systémů. Princip evolučního návrhu obvodů je založen na použití evolučního algoritmu pro iteračně založené hledání takového obvodu, který nejlépe odpovídá požadavkům návrháře specifikovaných ve formě hodnotící (tzv. fitness) funkce. Důvodů použití evolučního návrhu pro návrh číslicových obvodů je hned několik. Jedna z výhod tohoto přístupu spočívá v tom, že evoluce umožňuje navrhovat obvody, které jsou mimo možnosti konvenčního návrhu. To je umožněno odstraněním omezení, která jsou součástí tradičního návrhu, tj. nepoužitím klasických návrhových technik založených na dekompozici a minimalizaci a odstraněním předsudků, které vnáší sám návrhář [83]. Další výhodou evolučního návrhu je, že uživatel nemusí specifikovat, jakým způsobem se má daný obvod vytvořit, ale pouze specifikuje jak se má navržený obvod chovat, popř. jaké má mít vlastnosti.

Vzhledem k zaměření této práce a rozsahu, který problematika evolučního návrhu představuje, bude v následující části této práce představen aktuální stav v oblasti evolučního návrhu pro aplikace v oblasti diagnostiky. Podrobnější informace o evolučním návrhu číslicových obvodů je možné nalézt např. v [70, 72, 92].

3.3.1 Evoluční návrh obvodů odolných proti poruchám

První práce využívající evoluční techniky pro návrh obvodů s určitými diagnostickými vlastnostmi byly práce Adriana Thompsona [89–92], který se zabýval využitím evolučního návrhu pro vytváření obvodů odolných proti poruchám. Při konvenčním způsobu návrhu obvodu odolného proti poruchám je standardním přístupem použití *redundance*. Thompson ve své práci [89] ukázal, že evoluční návrh dokáže, při splnění určitých podmínek, vytvářet systémy odolné proti poruchám automaticky bez použití redundance.

Thompson toto zjištění opírá o jev, který popsal již dříve např. Eigen [22] nebo Huynen [50] při studiu molekulární evoluce. Tento jev spočívá v tom, že proces evoluce směřuje k produkci jedinců, kteří se vyznačují nejen vysokým fitness, ale také strukturou, která způsobuje, že aplikací mutace na jedince reprezentujícího jisté lokální maximum získáváme jedince, který

také s velkou pravděpodobností představuje jedince s vyšším fitness. Popisovaný jev je obvykle vysvětlován tak, že pokud máme jedince v populaci, tak šíření jeho genetické informace napříč populací je dáno nejen tím kolik potomků vyprodukuje, ale také tím, kolik potomků vyprodukují tito potomci. Jedinec s vysokým fitness, který je dostatečně odolný proti mutaci, bude mít potomky opět s vysokým fitness. Genetická informace tohoto potomka se bude populací šířit rychleji, než genetická informace potomka s vysokým fitness, který je více citlivý na mutaci genetické informace a je tedy předpoklad, že výsledkem evoluce bude právě jedinec odolnější proti mutaci, protože jeho genetická informace se bude vyskytovat v populaci jedinců častěji.

Obdobný princip demonstroval Thompson u evolučního návrhu obvodů odolných proti poruchám. Pokud předpokládáme v obvodu poruchy typu trvalá 0/1, tak tyto poruchy odpovídají aplikaci genetického operátoru mutace na binární chromozom. Evolucí vytvářené obvody jsou tak díky způsobu návrhu implicitně odolné proti tomuto typu poruch, který je nejčastěji používán pro modelování poruch. Thompson tento princip prezentoval na řadiči robota implicitně odolného proti poruchám.

3.3.2 Využití evolučního návrhu pro automatické zotavení z poruchy

Další zajímavou vlastností evolučního návrhu je možnost automatického zotavení obvodu z poruchy. Pokud evoluční návrh probíhá přímo v místě nasazení (např. přímo v programovatelném hradlovém poli) a nastane porucha některého prvku, evoluce umožňuje automaticky změnit konfiguraci tak, aby se porucha kompenzovala. Evoluce v tomto případě navíc dokáže nejen tolerovat chybný prvek, ale dokáže případně využít chybné chování prvku pro realizaci požadované funkce. Toho je možné využít v aplikacích, kde je možné, aby evoluce běžela permanentně na pozadí.

Možností automatického zotavení z poruchy se zabýval např. Lohn [63], který ve své práci zejména oceňuje, že evoluční návrh umožňuje využít také část obvodu s poruchou. Další výhodou automatického zotavení také je, že není potřeba přesně diagnostikovat, kde nastala chyba – evoluční návrh provede automatické obnovení funkce. Autor navrhuje možnost využití této vlastnosti na systému založeném na zálohování. Pokud jedno zařízení přestane fungovat, je využito záložní zařízení. První zařízení se rekonfiguruje a po zotavení je připraveno opět plnit svou funkci.

Zotavení z poruchy demonstruje Lohn na reálném příkladu, který reprezentuje detektor směru otáčení. Pro návrh detektoru otáčení je použit evoluční návrh, který jej realizuje pomocí stavového automatu o čtyřech stavech. V navrženém obvodu je pak dále simulována porucha a pomocí evolučního návrhu je provedeno zotavení z této poruchy. Výsledkem zotavení z poruchy je nalezení konfigurace programovatelného zařízení, která nejen chybný prvek toleruje, ale dokonce jej pro svou činnost využívá.

3.3.3 Návrh obvodů explicitně odolných proti poruchám

Thompson se kromě implicitní odolnosti evolucí navržených obvodů [89] zabýval také možností vytvářet obvody odolné proti poruchám tak, že se tato vlastnost explicitně zakomponuje přímo do hodnotící funkce [90, 91]. Odolnost proti poruchám je potom explicitní částí požadovaného chování takto evolucí navržených obvodů. Jedním z kroků ohodnocení kvality kandidátního řešení je v tomto případě softwarová simulace vlivu možných poruch obvodu na požadovanou funkci. Pokud probíhá evoluční návrh takového obvodu přímo v programovatelném poli,

můžeme poruchy emulovat přímo změnou konfigurace daného zařízení.

Podobný princip využil ve své práci také Sekanina [84], který prezentoval použití evolučního návrhu pro vytváření snadno testovatelných obrazových filtrů. Součástí ohodnocení kvality kandidátního řešení bylo, kromě ohodnocení kvality navržených filtrů, také ohodnocení testovatelnosti těchto filtrů. Výsledkem evolučního návrhu pak byly automatický navržené snadno testovatelné obrazové filtry.

3.3.4 Návrh obvodů s vestavěným testem

Další možnosti využití evolučního návrhu v diagnostice představuje práce [32], ve které Garvie společně s Thompsonem navrhli metodu umožňující evoluční návrh obvodů s vestavěným testem. Při vytváření obvodů s vestavěným testem vycházejí z toho, že se v hodnotící funkci kromě správné funkce obvodu ohodnotí také správná funkce výstupu E, který indikuje chybný výstup obvodu.

Garvie a Thompson ve své práci demonstrují fungování metody prostřednictvím evolučního návrhu dvou úplných sčítaček a násobičky s vestavěným testem. První navržená sčítačka obsahuje on-line vestavěný test a poskytuje 100% pokrytím poruch (při uváděném nárůstu složitosti sčítačky z 5 na 8 hradel). Druhá navržená sčítačka obsahuje off-line vestavěný test s pokrytím poruch 82% (při nárůstu složitosti sčítačky o 2 hradla). Výsledkem evolučního návrhu dvoubitové násobičky s vestavěným testem je on-line vestavěný test se 100% pokrytím poruch (při nárůstu složitosti ze 7 na 10 hradel).

3.3.5 Evoluční návrh obvodů s požadovanou funkcí

Metoda *EGG (angl. Evolutionary Graph Generation)* [47] představuje první přístup k evolučnímu návrhu obvodů s požadovanou funkcí. Návrh obvodu je realizován pomocí evolučního algoritmu. Míra do jaké navržený obvod plní požadovanou funkci je ohodnocena pomocí hodnotící funkce. Celý proces návrhu je realizován nad strukturou obvodu, která je reprezentována šesticí $G = (N, T_O, T_I, \nu_O, \nu_I, \epsilon)$, kde [47]:

- N je množina prvků obvodu,
- T_O je množina výstupních bran,
- T_I je množina vstupních bran,
- ν_O zobrazení z T_O na N: $n = \nu_O(u)$ vyjadřuje, že výstupní brána $u \in T_O$ patří prvku $n \in N$,
- ν_I zobrazení z T_I na N: $n = \nu_I(v)$ vyjadřuje, že vstupní brána $v \in T_I$ patří prvku $n \in N$,
- ϵ bijektivní zobrazení z T_O na T_I: $v = \epsilon(u)$ vyjadřuje, že výstupní brána $u \in T_O$ je připojena ke vstupní bráně $v \in T_I$.

Proces návrhu je realizován jednoduchým evolučním algoritmem, který pracuje nad výše uvedenou reprezentací obvodu. Na začátku je vytvořena počáteční populace skládající se z p náhodně vytvořených obvodů a je ohodnocena kvalita těchto obvodů pomocí hodnotící funkce. Ohodnocení kandidátního řešení o n prvcích probíhá v polynomiálním čase $O(n^3)$ a jejím cílem je ohodnotit do jaké míry obvod realizuje požadovanou funkci. Dále následuje vlastní evoluční proces. Z populace jedinců jsou pseudonáhodně vybráni (kvalitnější jedinci mají vyšší pravděpodobnost, že budou vybráni) dva jedinci p_1 a p_2, kteří se účastní reprodukce. Vybraní jedinci si vymění část své genetické informace. Na výsledné obvody p_1' a p_2' je s určitou pravděpodobností aplikován operátor mutace, který umožňuje přidat, popř. odstranit prvek ze

struktury obvodu. Noví jedinci jsou následně přidáni do populace potomků M. Po dosažení požadované velikosti populace potomků je vytvořena nová populace jedinců P_{t+1} výběrem nejlepších jedinců z populací P a M. Funkčnost navržené metody byla demonstrována na návrhu násobičky s jedním konstantním operandem.

Právě práce Hommy byla prvotním impulsem pro vznik této práce. Výsledkem jím navržené metody je návrh jednoduchých obvodů s požadovanou funkcí. Problémem metod založených na evolučním návrhu je však obvykle škálovatelnost vytvářených obvodů. Současné metody evolučního návrhu jsou omezeny na návrh obvodů o složitosti stovek až tisíců hradel. Problém návrhu obvodů vyšší složitosti spočívá v tom, že ohodnocení složitějšího obvodu trvá delší dobu než ohodnocení jednoduchého obvodu. V typickém evolučním algoritmu je dán počet potřebných ohodnocení násobkem počtu generací a velikosti populace. Je zřejmé, že více ohodnocení představuje prozkoumání větší části stavového prostoru a tím zvyšuje také pravděpodobnost nalezené optimálního řešení.

Pokud se zabýváme návrhem číslicového obvodu a zvýšíme počet vstupů o jeden, čas potřebný pro ohodnocení kvality tohoto obvodu, pokud předpokládáme, že ohodnocení spočívá v aplikaci všech možných binárních kombinací na vstupy obvodu, vzroste na dvojnásobek (roste exponenciálně). Jako rozumná strategie se jeví možnost aplikovat pouze podmnožinu možných vstupních kombinací. Bohužel ale například Miller ukazuje [73], že takto evolucí navržené obvody obvykle nepracují pro zbývající vstupní testovací vektory korektně.

Možným řešením problému škálovatelnosti je omezit využití evolučního návrhu pouze na problémy, u nichž doba potřebná pro ohodnocení jejich kvality neroste exponenciálně, ale polynomiálně. Příkladem takového problému je analýza testovatelnosti.

3.4 Analýza testovatelnosti obvodu na úrovni RT

Metody analýzy testovatelnosti tvoří poměrně rozsáhlou oblast, která je již řadu let předmětem aktivního výzkumu. Vzhledem k rozsahu této problematiky budou v této podkapitole pouze krátce představeny principy existujících metod analýzy testovatelnosti pracující na úrovni RT, na kterou je tato práce zejména zaměřena. Podrobnější informace o metodách analýzy testovatelnosti pracujících na nižší úrovni popisu je možné nalézt např. v [86].

Původně pracovaly metody analýzy testovatelnosti téměř výhradně na úrovni hradel. Z metod, které našly podle [78] nejširšího uplatnění, můžeme jmenovat např. metodu SCOAP [38], CAMELOT [5] nebo VICTOR [79]. Vzhledem k rostoucí složitosti číslicových systémů a tomu, že návrh číslicových obvodů je stále častěji realizován na vyšších úrovních popisu, začaly vznikat také metody analýzy testovatelnosti pracující na vyšších úrovních popisu. Výhodou těchto metod je, že není potřeba z důvodu analýzy testovatelnosti vytvářet reprezentaci na úrovni hradel a teprve na této úrovni realizovat samotnou analýzu testovatelnosti (popř. generování testu). Pokud máme k dispozici metody, které pracují na stejné úrovni jako probíhá samotný návrh, tak získáváme rychlejší zpětnou vazbu, která nás dokáže informovat o možných problémech aktuálního návrhu. Další výhodou je také to, že návrhář získává informace o problematických částech obvodu ve formě, která je mu srozumitelná a která odpovídá úrovni na níž je realizován samotný návrh. Pokud se zabýváme analýzou testovatelnosti na nižších úrovních, tak je obvykle obtížnější zpětně identifikovat problematické konstrukce, protože nižší úrovně jsou obvykle výsledkem automatizovaného procesu, během něhož dochází k optimalizacím, které mohou vést například k přejmenování či odstranění některých částí obvodu.

Pro metody analýzy testovatelnosti na úrovni RT je typické, že nějakým způsobem hodnotí řiditelnost a pozorovatelnost jednotlivých bran obvodu. Na základě ohodnocení těchto parametrů pak identifikují problematická místa obvodu a hodnotí celkovou testovatelnost obvodu. Během vývoje metod analýzy testovatelnosti docházelo k postupnému vývoji ohodnocení řiditelnosti/pozorovatelnosti jednotlivých uzlů od dvouhodnotového ohodnocení řiditelnosti/pozorovatelnosti bran obvodu až po dnešní ohodnocení zohledňující hned několik parametrů ovlivňující řiditelnost/pozorovatelnost dané brány. Vzhledem k rozsahu, který představují existující metody analýzy testovatelnosti, budou v další části této práce představeny pouze vybrané metody, které jsou nějakým způsobem zajímavé z hlediska tématu této práce. Informace o dalších metodách analýzy testovatelnosti je možné nalézt např. v [15,33,34,54,78,80,93,99].

3.4.1 Metoda TMEAS

Jednou z nejstarších metod analýzy testovatelnosti na úrovni RT je metoda *TMEAS* (z angl. Testability Measurement) představená Grasonem v roce 1976 [40], popřípadě její upravená verze představená v práci [39]. Metoda *TMEAS* umožňuje ohodnotit testovatelnost strukturou popsaného obvodu na úrovni RT prostřednictvím ohodnocení řiditelnosti a pozorovatelnosti bran obvodu. Vlastní ohodnocení testovatelnosti probíhá ve dvou fázích. V první fázi probíhá ohodnocení řiditelnosti bran obvodu $CY(s)$ jako ohodnocení obtížnosti nastavení brány s na požadovanou hodnotu prostřednictvím primárních vstupů obvodu. V druhé fázi pak probíhá ohodnocení pozorovatelnosti $OY(s)$ bran obvodu, které je realizováno jako ohodnocení obtížnosti detekce chybné odezvy na testovací vektor prostřednictvím primárních výstupů obvodu. Během procesu ohodnocení řiditelnosti a pozorovatelnosti bran obvodu jsou jednotlivým branám obvodu přiřazeny hodnoty z množiny $RealX$, kde vyšší hodnota představuje výskyt dané vlastnosti v lepší formě.

Speciálním případem jsou primární vstupy $i_1, i_2, \ldots, i_{n_i}$ a primární výstupy $o_1, o_2, \ldots, o_{n_o}$ obvodu, pro které platí, že $CY(i_k) = 1$ pro $k = 1, 2, \ldots n_i$ a $OY(o_k) = 1$ pro $k = 1, 2, \ldots n_o$. Vlastní ohodnocení řiditelnosti a pozorovatelnosti bran obvodu probíhá postupně směrem od primárních bran obvodu prostřednictvím níže uvedených vztahů.

Analýza řiditelnosti

Předpokládejme, že máme prvek N se vstupními branami $x_1, x_2, \ldots, x_n$ a výstupními branami $z_1, z_2, \ldots, z_m$. Hodnota řiditelnosti $CY(z_j)$ pro každý výstup z_j prvku N je dána vztahem

$$CY(z_j) = CTF \cdot \frac{1}{n} \sum_{i=1}^{n} CY(x_i), \tag{3.1}$$

kde CTF představuje činitel přenosu řiditelnosti (angl. Controllability Transfer Factor), který vyjadřuje jak obtížné bude řídit hodnotu na výstupu prvku prostřednictvím jeho vstupů. Hodnota tohoto parametru je vyjádřena vztahem

$$CTF = \frac{1}{m} \sum_{j=1}^{m} \left(1 - \frac{|N_j(0) - N_j(1)|}{2}\right), \tag{3.2}$$

kde $N_j(0)$ (resp. $N_j(1)$) je počet vstupních kombinací pro něž má z_j hodnotu 0 (resp. 1).

Analýza pozorovatelnosti

Podobně, pozorovatelnost každého vstupu x_i je vypočtena na základě vztahu

$$OY(x_i) = OTF \cdot \frac{1}{m} \sum_{j=1}^{m} OY(z_j), \tag{3.3}$$

kde OTF je činitel přenosu pozorovatelnosti (angl. Observability Transfer Factor), který vyjadřuje pravděpodobnost, že chybná hodnota na některém ze vstupů prvku bude propagována na výstup prvku. Činitel přenosu pozorovatelnosti je vyjádřen vztahem

$$OTF = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{NS_i}{2^n} \right), \tag{3.4}$$

kde NS_i je počet vstupních kombinací pro něž změna vstupu x_i způsobí změnu výstupu.

Zvláštní případ nastává, pokud se spoj s větví do k větví $b_1, b_2, \ldots, b_k$. Pak řiditelnost a pozorovatelnost je vyjádřena vztahy

$$CY(b_x) = \frac{CY(s)}{1 + \log k}, \qquad OY(s) = 1 - \prod_{i=1}^{k} (1 - OY(b_i)). \tag{3.5}$$

Nevýhodou metody *TMEAS* je poměrně jednoduchý model transparentnosti jednotlivých prvků. Transparentnost prvků je modelována pouze pomocí dvojice reálných čísel *CTF* a *OTF*, které vyjadřují jak obtížné bude řídit hodnotu na výstupu prvku, případně jak obtížné bude propagovat chybnou hodnotu na některém ze vstupů prvku na jeho výstup.

3.4.2 Metoda CAMELOT

Z metody *TMEAS* představené v předchozí části principiálně vychází metoda *CAMELOT* (Computer-Aided MEasure for LOgic Testability) [5, 69], která umožňuje lépe modelovat transparentnost jednotlivých prvků a tím dosahuje přesnějších výsledků ohodnocení testovatelnosti.
Princip metody CAMELOT můžeme shrnout do následujících bodů:

1. Načtení dat obvodu a inicializace hodnot řiditelnosti a pozorovatelnosti pro primární vstupy a výstupy obvodu.
2. Výpočet parametrů testovatelnosti pro všechny brány obvodu:
   (a) výpočet hodnoty řiditelnosti CY (postup směrem od primárních vstupů na primární výstupy),
   (b) výpočet hodnoty pozorovatelnosti OY (postup směrem od primárních výstupů na primární vstupy),
   (c) výpočet testovatelnosti $TY = CY \cdot OY$.
3. Výpočet celkové hodnoty testovatelnosti TY pro zadaný obvod jako průměrné hodnoty testovatelnosti jednotlivých bran obvodu.

Analýza řiditelnosti

Předpokládejme opět, že máme prvek N se vstupními branami $x_1, x_2, \ldots, x_n$ a výstupními branami $z_1, z_2, \ldots, z_m$. Hodnota řiditelnosti $CY(z_j)$ výstupu z_j prvku N je dána vztahem

$$CY(z_j) = CTF(z_j) \cdot f(CY(x_1), CY(x_2), \cdots, CY(x_n)) \tag{3.6}$$

kde $CTF(z_j)$ představuje činitel přenosu řiditelnosti pro výstup z_j a f zpracovává řiditelnost všech vstupů prvnu N, které ovlivňují hodnotu výstupu z_j. Do řiditelnosti výstupu z_j se tak započítá pouze řiditelnost těch vstupních bran, které mají vliv na hodnotu řiditelnosti výstupu z_j. Hodnotu činitele přenosu řiditelnosti vyjadřuje vztah

$$CTF(z_j) = 1 - \left| \frac{N(0)_{z_j} - N(1)_{z_j}}{N(0)_{z_j} + N(1)_{z_j}} \right|, \tag{3.7}$$

kde $N(0)_{z_j}$ resp. $N(1)_{z_j}$ jsou počty vstupních kombinací pro které výstup z_j nabývá hodnoty 0 resp. 1.

Analýza pozorovatelnosti

Pozorovatelnost vstupu x_i je vypočtena na základě hodnoty pozorovatelnosti daného výstupu z_j a hodnot řiditelnosti CY vstupů $(x_1, \ldots, x_k)$ nezbytných pro zajištění pozorovatelnosti z x_i na z_j a hodnoty činitele přenosu pozorovatelnosti $OTF(x_i - z_j)$ podle vztahu

$$OY(x_i) = OTF(x_i - z_j) \cdot OY(z_j) \cdot g(x_1, x_2, \ldots, x_k), \tag{3.8}$$

kde OTF je činitel přenosu pozorovatelnosti, $OY(z_j)$ je pozorovatelnost výstupní brány z_j a $g(x_1, x_2, \ldots, x_k)$ zohledňuje řiditelnost vstupních bran potřebných pro propagaci pozorovatelnosti z brány x_i na bránu z_j. Pro vyjádření činitele OTF je potřeba zavést pomocné funkce $N(SP : x_i - z_j)$ resp. $N(IP : x_i - z_j)$, vyjadřující počet citlivých cest pro šíření poruchy ze vstupu x_i na výstup z_j resp. počet způsobů, jimiž je možné šíření poruchy ze vstupu x_i na výstup z_j blokovat. Pro daný vstup x_i a výstup z_j je možné vyjádřit $OTF(x_i - z_j)$ vztahem

$$OTF(x_i - z_j) = \frac{N(SP : x_i - z_j)}{N(SP : x_i - z_j) + N(IP : x_i - z_j)} \tag{3.9}$$

Testovatelnost $TY(x)$ vnitřního bodu obvodu x je určena součinem hodnot řiditelnosti a pozorovatelnosti bodu x: $TY(x) = CY(x) \cdot OY(x)$. Celková testovatelnost obvodu je dána aritmetickým průměrem testovatelnosti TY všech vnitřních bodů analyzovaného obvodu.

3.4.3 Práce inspirované metodou SCOAP

Samostatnou kapitolu metod ohodnocení testovatelnosti na úrovni RT tvoří práce inspirované metodou *SCOAP* (Sandia Controllability Observability Analysis Program) [37, 38]. Metoda *SCOAP* je pravděpodobně nejznámější metodou pro analýzu testovatelnosti používanou na

úrovni logických hradel. Tato metoda je založena na ohodnocení obtížnosti ovládání resp. pozorování konkrétních jednobitových hodnot na vnitřních branách obvodu, přičemž vyšší hodnoty řiditelnosti resp. pozorovatelnosti indikují větší obtížnost této činnosti. Časová složitost analýzy testovatelnosti pomocí SCOAP je lineární vzhledem k počtu hradel obvodu, což z metody SCOAP činí atraktivní nástroj pro posuzování testovatelnosti na úrovni logických členů.

Metodou *SCOAP* se ve své práci inspirovali např. Chen a Menon [12], kteří představili na konferenci ITC'89 metodu analýzy testovatelnosti pracující na úrovni RT, která používá podobný princip jako metoda SCOAP. Ohodnocení řiditelnosti a pozorovatelnosti jednotlivých částí obvodu je založeno na ohodnocení 4 parametrů, které vyjadřují obtížnost nastavení a pozorování hodnoty brány v daném místě obvodu: kombinační řiditelnosti (CC), sekvenční řiditelnosti (SC), kombinační pozorovatelnosti (CO) a sekvenční pozorovatelnosti (SO). Hodnoty parametrů řiditelnosti a pozorovatelnosti jsou pak ještě dále rozděleny tak, aby umožňovaly vyjádřit obtížnost nastavení daného místa na log. 1 popř. 0. Ve skutečnosti tak máme pro každý spoj k dispozici informaci o šesti parametrech: CC_0, SC_0, CC_1, SC_1, CO a SO, kde parametr CC_0 resp. CC_1 reprezentuje pravděpodobnost, že daný vnitřní signál je nastaven na hodnotu 0, resp. 1. Parametr SC_0 (resp. SC_1) představuje odhad sekvenční délky potřebné pro nastavení dané brány na požadovanou hodnotu. Parametr CO vyjadřuje pravděpodobnost, že změna na vstupu se projeví jako změna na výstupu prvku a poslední parametr SO vyjadřuje odhad počtu časových rámců, které jsou potřeba pro propagaci změny na vstupech prvku na primární výstupy. Kombinací těchto šesti parametrů získáváme informaci o testovatelnosti analyzovaného obvodu.

Podobný přístup, založený na metodě *SCOAP*, pak ve své práci prezentovali například také Kuchcinski, Gu a Peng [41, 42, 59]. Vzájemné porovnání těchto metod je však vzhledem k neexistenci testovacích obvodů, které by mohly být použity pro vzájemné porovnání metod, poměrně problematické.

3.4.4 Pravděpodobnostní metody analýzy testovatelnosti

Další třídou metod analýzy testovatelnosti na úrovni RT jsou pravděpodobnostní metody analýzy testovatelnosti. Příkladem tohoto typu metod je například metoda analýzy testovatelnosti, kterou navrhl Flottes [25]. Řiditelnost n bitové brány N je v této práci určena pravděpodobností, že daná brána může být prostřednictvím primárních vstupů obvodu nastavena na požadovanou hodnotu. Pro výpočet řiditelnosti C brány N platí, že $C(N) = x/2^n$, kde x je počet různých hodnot, na které je možné prostřednictvím primárních vstupů obvodu nastavit bránu N. Podobně pozorovatelnost n bitové brány N obvodu je vyjádřena vztahem $O(N) = x/(2^{n-1} \cdot (2^n - 1))$, kde x je počet různých dvojic n bitových vzorů, které mohou být propagovány a rozlišeny na primárních výstupech obvodu. Pozorovatelnost brány obvodu tak vyjadřuje, do jaké míry jsme schopni prostřednictvím primárních výstupů rozlišit chybnou hodnotu na této bráně.

Výhodou tohoto přístupu je, že nevyužívá žádný abstraktní model, který by modeloval transparentní vlastnosti prvků v závislosti na přivedení neutrální hodnoty na jejich jiný vstup. Je možné jej tedy použít pro libovolný prvek pro nějž známe jeho chování. Nevýhodou je však časová složitost této metody, kdy pro jednotlivé prvky musíme simulovat jejich funkci a zjišťovat možnost přenosu diagnostických dat strukturou obvodu. Metoda také nezohledňuje, zda je možné na danou bránu přivést potřebné testovací vektory.

Zajímavé na této práci je to, že se jedná o jeden z prvních přístupů, kdy jsou výsledky

získané pomocí navržené metody analýzy testovatelnosti použity pro řízení procesu syntézy tak, aby bylo dosaženo zlepšení testovatelnosti výsledného obvodu.

3.4.5 Numerické metody analýzy testovatelnosti

Nový přístup k analýze testovatelnosti představil Fernandes ve své práci [24]. Jím představená metoda určená pro ohodnocení testovatelnosti sekvenčního obvodu na úrovni RT je založená na modelování synchronního sekvenčního obvodu jako markovského řetězce. Vychází z toho, že následující stav synchronního sekvenčního obvodu je dán jeho aktuálním stavem a hodnotou na vstupu. Chování synchronního sekvenční obvodu je v této práci modelováno pomocí grafu přechodů vyjádřeného pomocí *n*-tice $(\Sigma, \Delta, Q, q_0, \delta, \lambda)$, kde Σ je neprázdná konečná množina vstupních symbolů reprezentujících hodnotu vstupních bran obvodu, Δ je konečná množina výstupních symbolů reprezentující výstup obvodu, Q je neprázdná konečná množina stavů, q_0 je počáteční stav obvodu po aplikaci signálu reset, $\delta : (Q \times \Sigma) \rightarrow Q$ je přechodová funkce a λ:$(Q \times \Sigma) \rightarrow \Delta$ je výstupní funkce udávající hodnotu výstupních bran. Pro každý stav obvodu můžeme určit pravděpodobnost přechodu z předcházejícího stavu. Řešením soustavy Chapman-Kolmogorových rovnic popisujících ustálený stav obvodu pak získáme informace o řiditelnosti jednotlivých vnitřních částí obvodu. Otázkou u této metody však zůstává její použitelnost pro větší obvody. Autoři prezentují výsledky metody pouze na části obvodů ze sady ITC'99.

3.4.6 Metody založené na využití transparentních cest

Další zajímavý typ metod analýzy testovatelnosti představují metody založené na využití transparentních cest a hierarchického testu. Příkladem těchto metod jsou např. metody [64,81,86]. Tyto metody analýzy testovatelnosti předpokládají, že jsou k dispozici testovací vektory umožňující otestovat jednotlivé moduly obvodu. V rámci analýzy testovatelnosti jsou pak především analyzovány transparentní cesty v obvodu z hlediska možnosti přivést testovací vektory z primárních vstupů obvodu na vstupy testovaného modulu a sledovat odezvy na testovací vektory prostřednictvím primárních výstupů obvodu.

V [58] je prezentována možnost uplatnění principů řiditelnosti/pozorovatelnosti při návrhu číslicových systémů. Je využit pojem transparentnosti prvků, které se mohou v obvodě na úrovni RT vyskytnout. Tento pojem je pak využit v metodice klasifikace registrů z hlediska jejich role při aplikaci testu, pro tyto účely je použit formální model založený na pojmech diskrétní matematiky. Součástí práce je také jednoduchá metodika pro výběr registrů do řetězce scan využívající zavedenou klasifikaci registrů.

Principy uplatnění formálního postupu při analýze testovatelnosti jsou dále rozvíjeny v disertační práci R. Růžičky [81]. Růžičkou navržená metoda pracuje nad datovou částí strukturou popsaného obvodu na úrovni RT. Ohodnocení testovatelnosti je u této metody realizováno jako ohodnocení řiditelnosti a pozorovatelnosti spojů analyzovaného obvodu. Chování prvků obvodu z hlediska průchodu diagnostických dat strukturou obvodu je modelováno pomocí tří typů prvků – multiplexerů, registrů a funkčních jednotek. Jednotlivé spoje obvodu jsou pak ohodnoceny buď jako řiditelné, pokud existuje *i-cesta* z primárních vstupů obvodu, prostřednictvím které je možné nastavit daný spoj na požadovanou hodnotu, popř. pozorovatelné pokud existuje *i-cesta* prostřednictvím níž je možné na primárních výstupech pozorovat hodnotu na daném spoji. Autor ve své práci kromě samotné metody analýzy testovatelnosti představuje také

algoritmus výběru registrů do řetězce *scan* a také model založený na Petriho sítích umožňující analyzovat konflikty při průchodu diagnostických dat strukturou obvodu.

Přínosem Růžičkovy práce [81] je také formální přístup, který je použit pro účely analýzy testovatelnosti. V práci je zaveden formální model struktury číslicového obvodu na úrovni RT a transparentních cest obvodu. Veškeré objekty, které jsou předmětem analýzy (obvodové prvky, spoje atd.), jsou podle svých vlastností sdruženy do množin; další vlastnosti a vztahy mezi těmito objekty jsou vyjádřeny relacemi a pro popis je volen jazyk predikátové logiky. Výhodami formálního přístupu jsou zejména jednoznačnost zápisu a možnost transformovat problémy analýzy testovatelnosti na známé a řešené problémy diskrétní matematiky a teoretické informatiky a využít známých, efektivních a ověřených postupů a algoritmů.

Na práci Růžičky navázal Strnadel, který se ve své disertační práci [86] více zaměřil na problematiku analýzy testovatelnosti obvodu na úrovni RT. Řiditelnost a pozorovatelnost spojů obvodu je v jeho práci ohodnocena reálným číslem, které vyjadřuje obtížnost nastavení hodnoty v daném místě obvodu, popř. obtížnost pozorování hodnoty v tomto místě prostřednictvím primárních výstupů obvodu. V práci je použit formální model číslicového obvodu zavedený Růžičkou, který je dále rozšířen např. o modelování šířky spojů a tzv. virtuální porty, které představují určitou abstrakci nad rozhraním jednotlivých prvků v obvodu a umožňují lépe modelovat transparentní cesty strukturou obvodu. Zcela nový je pak model transparentních cest v obvodu, který využívá právě abstrakce zavedené pomocí virtuálních portů a umožňuje modelovat transparentní cesty, které nebylo možné modelovat s využitím původního Růžičkova modelu.

Strnadel ve své práci také představuje formální přístup k analýze testovatelnosti strukturou popsaného obvodu na úrovni RT. Z hlediska principu navržené metody je možné říci, že vychází z metody SCOAP [37,38] a dalších prací navazující na tuto metodu [12,41,42,59]. Ohodnocení testovatelnosti je realizováno nad formálním modelem toku diagnostických dat obvodem, který je modelován pomocí dvojice digrafů G_S a G_I, kde G_S modeluje tok testovacích vektorů strukturou obvodu a G_I modeluje tok odezev na testovací vektory. Uzly těchto grafů představují brány prvků obvodu a hrany představují metalické spoje mezi jednotlivými branami, popř. existenci transparentní cesty mezi těmito branami. Pro každou transparentní cestu pak dále existuje relace popisující podmínky potřebné ke vzniku dané cesty.

Analýza testovatelnosti je realizována ve dvou krocích jako ohodnocení uzlů grafu G_S a G_I. V prvním kroku algoritmus ohodnotí dostupnost jednotlivých uzlů grafu G_S prostřednictvím primárních vstupů obvodu (tento krok odpovídá analýze řiditelnosti) a v druhém kroku pak dostupnost primárních výstupů obvodu z jednotlivých uzlů grafu G_I (tento krok odpovídá analýze pozorovatelnosti). Řiditelnost a pozorovatelnost jednotlivých bran obvodu je ohodnocena reálným číslem z intervalu $\langle 0; 1\rangle$, kde 0 představuje absenci daného atributu a 1 značí existenci daného atributu v jeho nejlepší možné formě. Celková testovatelnost obvodu je pak vyjádřena jako funkce řiditelnosti a pozorovatelnosti jednotlivých uzlů grafu G_S a G_I.

Výhodou tohoto přístupu oproti například pravděpodobnostním metodám analýzy testovatelnosti je jeho nižší časová náročnost. Nižší časová náročnost je dána zejména odstraněním nutnosti analyzovat funkci jednotlivých prvků obvodu při procesu generování globálních testovacích vektorů. Metoda totiž předpokládá, že je pro každý modul k dispozici model umožňující modelovat transparentní chování modulu. Na kvalitě použitého modelu transparentního chování prvku pak závisí také výsledná kvalita metody analýzy testovatelnosti.

3.5 Shrnutí

V této kapitole byl představen aktuální stav v oblastech úzce souvisejících s tématem řešeným v rámci této disertační práce. Na základě analýzy aktuálního stavu existujících sad testovacích obvodů určených pro ověřování diagnostických metod můžeme konstatovat, že v současné době chybí sada testovacích obvodů potřebné složitosti popsaná na některé z vyšších úrovní popisu, kterou by bylo možné použít pro efektivní ověřování nových metod a nástrojů. Druhá podkapitola tak byla věnována metodám vytváření syntetických testovacích obvodů, které se v některých oblastech (viz podkapitola 3.2) prosazují jako možná alternativa standardních testovacích obvodů. Bohužel zatím neexistuje přístup, který by umožňoval navrhovat syntetické testovací obvody s tak komplexním parametrem jaký představuje testovatelnost číslicového obvodu. Možným řešením problému návrhu syntetických testovacích obvodů s požadovanými diagnostickými vlastnostmi by mohla být kombinace efektivní metody analýzy testovatelnosti a evolučního návrhu. Současné metody evolučního návrhu obvykle narážejí na problém škálovatelnosti, protože doba potřebná pro ohodnocení kandidátního řešení roste s rostoucí složitostí ohodnocovaných obvodů exponenciálně. Myšlenkou této práce je využít metody evolučního návrhu pro návrh obvodů s požadovanými diagnostickými vlastnostmi, u nichž doba potřebná pro ohodnocení kandidátního řešení roste polynomiálně. V závěru této kapitoly tak byly analyzovány metody, které by bylo možné použít pro analýzu testovatelnosti kandidátního obvodu na úrovni RT.

Kapitola 4

Model obvodu na úrovni RT

Ještě než přistoupíme k samotnému návrhu metody pro automatické vytváření syntetických testovacích obvodů, zavedeme nejprve model číslicového obvodu na úrovni RT, který nám umožní přesný, přehledný a nesporný popis navržené metody. Vzhledem k tomu, že tato práce určitým způsobem navazuje na disertační práci Richarda Růžičky [81] a Josefa Strnadela [86], budeme v této práci vycházet z formálních modelů zavedených těmito autory.

Růžičkou zavedený formální model vyjadřuje strukturu číslicového obvodu UUA (z angl. Unit Under Analysis) jako pětici konečných množin $UUA = (E, P, C, PI, PO)$, kde E je množina obvodových prvků, P množina bran těchto prvků, C množina spojů obvodu, PI množina primárních vstupů obvodu a PO množina primárních výstupů obvodu. Nevýhodou uvedeného modelu je, že pro exaktní definici struktury obvodu jsou potřeba další pomocná zobrazení, které například přiřazují brány k jednotlivým prvkům, modelují šířku spojů obvodu nebo umožňují rozlišit řídicí a datové brány. Uvedený model také neumožňuje hierarchický popis struktury obvodu, který je často využit zejména u složitějších obvodů, kdy se za pomoci základních prvků úrovně RT sestavují jednoduché moduly (např. FIFO, čítače, sady multiplexerů, . . .) a teprve s použitím těchto modulů vznikají moduly složitější, které na nejvyšší úrovni tvoří obvod realizující požadovanou funkci.

Vzhledem k tomu, že uvedené nedostatky dostatečně neřešil ani formální model zavedený v disertační práci Josefa Strnadela [86], která dále rozšířila Růžičkou definovaný formální model např. o přesnější modelování transparentních cest, vznikl v rámci této práce nový formální model, který řeší všechny výše uvedené nedostatky.

Navržený formální model byl sestaven tak, aby umožňoval reprezentovat jak základní prvky úrovně RT tak složitější moduly a obvody na této úrovni. Model byl záměrně navržen tak, že nepodporuje modelování třístavových sběrnice ve struktuře obvodu. Rozšíření o možnost modelování třístavových sběrnic nepředstavuje složitý problém, ale vzhledem k tomu, že návrh obvodů s třístavovými sběrnicemi není cílem této práce, znamenal by takový model zbytečně složitější a méně přehledný zápis navrhované metody.

Vytvořený model se skládá ze tří hlavních částí: modelu struktury, modelu rozhraní a modelu spojů. Samostatnou součástí modelu je pak také model transparentních cest. Jednotlivé části modelu společně s modelem transparentních cest budou představeny v následujících podkapitolách.

4.1 Model struktury obvodu na úrovni RT

Pro přesný a přehledný zápis metody návrhu syntetických testovacích obvodů potřebujeme mít k dispozici model obvodu, který nám umožní modelovat strukturu hierarchicky popsaného číslicového obvodu na úrovni RT a bude vhodný pro formální zápis navržené metody generování syntetických testovacích obvodů. Vzhledem k tomu, že dostupné formální modely tento požadavek nesplnily, bylo potřeba takový model navrhnout.

Při vytváření formálního modelu se tato práce inspirovala strukturním popisem obvodu v jazyce VHDL. Strukturní popis obvodu v jazyce VHDL tvoří dvě hlavní části:

- **popis rozhraní** – obsahuje informace rozhraní obvodu (sekce *entity* ve zdrojovém VHDL kódu). Pro každou primární bránu obvodu je v této sekci k dispozici informace o názvu brány, její bitové šířce a orientaci,
- **popis struktury** – obsahuje informace o použitých prvcích a jejich vzájemném propojení (sekce *architecture* ve zdrojovém VHDL kódu).

Podobně reprezentuje strukturu obvodu také formální model navržený v této práci. Struktura číslicového obvodu UUA je reprezentována jako uspořádaná trojice $UUA = (I, E, C)$, kde I reprezentuje rozhraní obvodu UUA, E je množina prvků, z nichž se obvod UUA skládá a C je množina spojů vyjadřující vzájemné propojení prvků obvodu UUA. Formálně je model obvodu zaveden v definici 4.1.

Definice 4.1: Mějme strukturou popsaný obvod na úrovni RT. Potom nechť uspořádaná trojice $UUA = (I, E, C)$ vyjadřuje model struktury tohoto číslicového obvodu, kde význam jednotlivých členů této trojice je následující: I je čtveřice (DI, DO, CI, CO) navzájem disjunktních konečných množin popisujících rozhraní obvodu UUA, E je konečná množina trojic $\{e_1, \ldots, e_n\} = \{(I_1, E_1, C_1), \ldots, (I_n, E_n, C_n)\}$ reprezentujících prvky (moduly) obvodu UUA a C je konečná množina spojů obvodu UUA.
□

První člen trojice UUA, čtveřice $I = (DI, DO, CI, CO)$ vzájemně disjunktních konečných množin, reprezentuje rozhraní obvodu UUA. Jednotlivé množiny DI, DO, CI a CO reprezentují různý typ (datová/řídicí) a orientaci (vstupní/výstupní) bran rozhraní UUA. Každá brána rozhraní je pak v jedné z uvedených čtyřech množin reprezentována dvojicí $(id, width)$, kde id je jednoznačná identifikace brány rozhraní v rámci obvodu (vyjádřená např. ve formě *název_prvku.název_brány*) a $width$ vyjadřuje bitovou šířku brány. Druhý člen trojice UUA – konečná množina $E = \{e_1, \ldots, e_n\} = \{(I_1, E_1, C_1), \ldots, (I_n, E_n, C_n)\}$ reprezentuje množinu prvků, z nichž se obvod skládá. Pro modelování rozhraní (případně struktury) těchto prvků (modulů) je opět rekurzivně použit tento model. Třetí člen trojice UUA – konečná množina C dvojic bran (in, out) reprezentuje metalické propojení jednotlivých bran prvků a rozhraní obvodu UUA.

Ještě než přistoupíme k formální definici jednotlivých členů trojice $UUA = (I, E, C)$ (viz podkapitola 4.2 a 4.3), bude nejprve podrobněji neformálně představen význam jednotlivých členů a bude také demonstrováno využití tohoto modelu pro popis základního prvku a jednoduchého obvodu na úrovni RT.

Příklad 4.1: Mějme dvouvstupový 8 bitový multiplexor (viz multiplexor MUX na obrázku 4.1). Pro popis struktury tohoto prvku použijme formální model zavedený v definici 4.1.

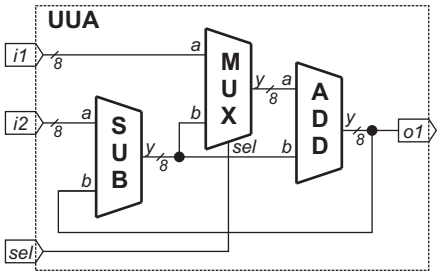
Multiplexor představuje základní prvek úrovně RT – jako základní prvek je tedy reprezentován pouze svým rozhraním a jeho vnitřní struktura není na úrovni RT dále modelována – množiny E a C jsou prázdné. Rozhraní prvku tvoří dvě vstupní 8 bitové datové brány ($mux.a$, $mux.b$), jedna 8 bitová výstupní datová brána ($mux.y$) a jedna vstupní řídicí brána ($mux.sel$).

$I = (\{(mux.a, 8), (mux.b, 8)\}, \{(mux.y, 8)\}, \{(mux.sel, 1)\}, \emptyset)$
$E = \emptyset$
$C = \emptyset$

Celkový model prvku MUX je dán trojicí (I, E, C):

$$\begin{aligned} MUX =& (I, E, C) = ((DI, DO, CI, CO), E, C) = \\ =& ((\{(mux.a, 8), (mux.b, 8)\}, \{(mux.y, 8)\}, \{(mux.sel, 1)\}, \emptyset), \emptyset, \emptyset) \end{aligned}$$

Stejný formální model, který byl použit pro modelování základního prvku úrovně RT můžeme použít také pro modelování číslicového obvodu. Možnosti modelování struktury obvodu pomocí zavedeného modelu si můžeme demonstrovat na obvodu na obrázku 4.1.



Obr. 4.1: Ukázka obvodu na úrovni RT.

Příklad 4.2: Mějme obvod UUA na obrázku 4.1, použijme formální model zavedený v definici 4.1 pro popis struktury tohoto obvodu.

Rozhraní obvodu UUA tvoří dvě vstupní 8 bitové datové brány $uua.i1$ a $uua.i2$, výstupní 8 bitová datová brána $uua.o1$ a řídící brána $uua.sel$. Rozhraní obvodu UUA můžeme formálně popsat pomocí čtveřice I.

$I = (\{(uua.i1, 8), (uua.i2, 8)\}, \{(uua.o1, 8)\}, \{(uua.sel, 1)\}, \emptyset)$

Strukturu obvodu tvoří tři prvky – SUB, MUX a ADD. Tyto prvky můžeme popsat podobně jako multiplexer v příkladu 4.1.

$$\begin{aligned} SUB =& \ ((\{(sub.a, 8), (sub.b, 8)\}, \{(sub.y, 8)\}, \emptyset, \emptyset), \quad \emptyset, \quad \emptyset) \\ MUX =& ((\{(mux.a, 8), (mux.b, 8)\}, \{(mux.y, 8)\}, \{(mux.sel, 1)\}, \emptyset), \quad \emptyset, \quad \emptyset) \\ ADD =& \ ((\{(add.a, 8), (add.b, 8)\}, \{(add.y, 8)\}, \emptyset, \emptyset), \quad \emptyset, \quad \emptyset)\} \\ E =& \{SUB, MUX, ADD\} \end{aligned}$$

Vzájemné propojení prvků obvodu a primárních vstupů a výstupů můžeme zapsat ve formě dvojic (x, y), kde x představuje bránu obvodu se vstupní orientací a y bránu s výstupní orientací. Pro obvod na obrázku 4.1 můžeme propojení prvků obvodu zapsat pomocí následující relace C.

$$C = \{ (mux.a, uua.i1), (sub.a, uua.i2), (mux.b, sub.y), (add.b, sub.y), \\ (add.a, mux.y), (uua.o1, add.y), (sub.b, add.y), (mux.sel, uua.sel)\}$$

Samotný obvod UUA je pak definován trojicí (I, E, C), kde jednotlivé členy této trojice byly definovány výše.

$UUA = (I, E, C)$

V předchozí části této kapitoly byl zaveden formální model strukturou popsaného obvodu na úrovni RT a použití tohoto modelu bylo demonstrováno na příkladu základního prvku a jednoduchého obvodu na úrovni RT. Dosud však nebyly formálně definovány jednotlivé členy trojice $UUA = (I, E, C)$. Než však přistoupíme k formální definici jednotlivých členů trojice (I, E, C) zaveďme nejprve úmluvu, která nám zjednoduší zápis operací s jednotlivými členy trojice a zároveň umožní přehlednější práci s vytvořeným modelem.

Úmluva 2: Mějme trojici $UUA = (I, E, C)$, která reprezentuje strukturu obvodu UUA. Potom nechť zápis $I(UUA)$ reprezentuje rozhraní I obvodu UUA a podobně $E(UUA)$ a $C(UUA)$ reprezentuje konečnou množinu prvků E a spojů C obvodu UUA.
□

4.2 Model rozhraní

Model rozhraní obvodu (prvku) vychází z definice rozhraní typického pro většinu HDL jazyků. Rozhraní je v těchto jazycích obvykle definováno jako seznam bran jednoznačně identifikovaných svým názvem, kde je pro každou bránu k dispozici informace o její orientaci a bitové šířce. V této práci je pro účely analýzy testovatelnosti obvodu model rozhraní navíc dále rozšířen o modelování datových a řídicích bran.

Navržený model reprezentuje rozhraní obvodu I jako čtveřici konečných množin (DI,DO,CI,CO), kde jednotlivé členy této čtveřice modelují typ brány (datová/řídicí) a její orientaci (vstupní/výstupní) – pro zjednodušení zápisu návrhové metody předpokládáme, že v obvodu nejsou použity třístavové sběrnice. Samotné brány jsou pak modelovány ve formě dvojic $(gate\_id, gate\_width)$, kde $gate\_id$ je jednoznačná identifikace brány a $gate\_width$ je její bitová šířka.

Definice 4.2: Nechť $gate\_id$ je jednoznačná identifikace brány na dané obvodové úrovni a $gate\_width$ je bitová šířka této brány. Potom brána obvodu g je definována uspořádanou dvojicí $(gate\_id, gate\_width)$.
□

Definice 4.3: Nechť rozhraní I obvodu $UUA = (I, E, C)$ je definováno uspořádanou čtveřicí konečných množin $I = (DI, DO, CI, CO)$, kde DI reprezentuje množinu vstupních datových bran, DO množinu výstupních datových bran, CI množinu vstupních řídicích bran a CO množinu výstupních řídicích bran rozhraní UUA.
□

Výše zavedený model rozhraní obvodu tedy dokáže pro všechny brány rozhraní obvodu modelovat název (identifikátor) brány, její datovou šířku, orientaci a typ brány. Vzhledem k tomu, že v další části této práce bude potřeba přistupovat a pracovat s jednotlivými množinami bran odpovídajícími členům čtveřice (DI, DO, CI, CO), budou nyní zavedeny pomocné množiny I_{DI}, I_{DO}, I_{CI} a I_{CO}, jež ale na rozdíl od bran DI, DO, CI a CO obsahují pouze informace o názvu brány patřící do odpovídající množiny bran (neobsahují informaci o bitové šířce bran). Budou také zavedeny pomocné množiny I_{IN}, I_{OUT} a I_{ALL} sdružující brány rozhraní.

Definice 4.4: Nechť $I(UUA)$ reprezentuje rozhraní (DI, DO, CI, CO) obvodu $UUA = (I, E, C)$. Pak množiny I_{DI}, I_{DO}, I_{CI}, I_{CO}, I_{IN}, I_{OUT} a I_{ALL} jsou definovány takto:

$I_{DI}(UUA) = \{g_{id} | (g_{id}, g_{width}) \in DI\}$,
kde I_{DI} identifikuje množinu vstupních datových bran UUA,
$I_{DO}(UUA) = \{g_{id} | (g_{id}, g_{width}) \in DO\}$,
kde I_{DO} identifikuje množinu výstupních datových bran UUA,
$I_{CI}(UUA) = \{g_{id} | (g_{id}, g_{width}) \in CI\}$,
kde I_{CI} identifikuje množinu vstupních řídicích bran UUA,
$I_{CO}(UUA) = \{g_{id} | (g_{id}, g_{width}) \in CO\}$,
kde I_{CO} identifikuje množinu výstupních řídicích bran UUA,
$I_{IN}(UUA) = I_{DI}(UUA) \cup I_{CI}(UUA)$,
kde I_{IN} identifikuje množinu vstupních bran UUA,
$I_{OUT}(UUA) = I_{DO}(UUA) \cup I_{CO}(UUA)$,
kde I_{OUT} identifikuje množinu výstupních bran UUA,
$I_{ALL}(UUA) = I_{IN}(UUA) \cup I_{OUT}(UUA)$,
kde I_{ALL} identifikuje množinu všech bran UUA ($I_{DI} \cup I_{DO} \cup I_{CI} \cup I_{CO}$).
□

Pomocí zavedených množin získáváme možnost pracovat se skupinou bran, které jsou pro nás v daném kontextu zajímavé. Použitím zápisu I_{DI} pro množinu vstupních datových bran $DI = \{(add1.a, 8), (add1.b, 8)\}$ například získáváme množinu $I_{DI} = \{add1.a, add1.b\}$ identifikátorů vstupních datových bran. Odstranění informace o šířce bran nám umožní v některých případech výrazně zjednodušit zápis operací s jednotlivými branami. Abychom mohli jednoduše pracovat také s bitovou šířkou bran, zavedeme dále zobrazení Φ, které každé bráně rozhraní $g \in I_{ALL}(UUA)$ obvodu (prvku) UUA přiřadí informaci o její bitové šířce.

Definice 4.5: Mějme obvod $UUA = (I, E, C)$. K němu definujme zobrazení $\Phi \subseteq (I_{ALL}(UUA) \times (\mathbb{N} \setminus \{0\}))$, které každé bráně rozhraní obvodu $g_{id} \in I_{ALL}(UUA)$ přiřazuje její bitovou šířku $g_{width} \in (\mathbb{N} \setminus \{0\})$: $(g_{id}, g_{width}) \in \Phi(UUA) = DI \cup DO \cup CI \cup CO$.
□

4.3 Model spojů obvodu

Model spojů obvodu $UUA = (I, E, C)$ modeluje vzájemné propojení prvků obvodu jako relaci C, která bráně x se vstupní orientací přiřazuje bránu y s výstupní orientací, se kterou je brána x propojena metalickým spojem. Abychom mohli model spojů C formálně popsat, je potřeba zavést některé pomocné množiny reprezentující brány dostupné ve struktuře obvodu (brány rozhraní obvodu UUA a brány rozhraní jednotlivých prvků z množiny E).

Brány obvodu (včetně bran rozhraní) můžeme podobně jako brány rozhraní rozdělit do čtyř vzájemně disjuktních množin P_{DI}, P_{DO}, P_{CI} a P_{CO} podle typu brány (datová/řídicí) a její orientace (vstupní/výstupní). Jednotlivé množiny bran pak můžeme sdružit např. podle jejich orientace do množin představující brány se vstupní orientací (P_{IN}) a výstupní orientací (P_{OUT}), popř. všechny brány obvodu (P_{ALL}). Zavedení těchto množin nám později umožní jednodušší zápis algoritmů pracujících se zavedeným formálním modelem.

4.3.1 Modelování chování primárních bran vůči vnitřní struktuře obvodu

Při sestavování jednotlivých množin bran P_{DI}, P_{DO}, P_{CI} a P_{CO} je potřeba si uvědomit, že primární vstupy obvodu představují z hlediska vnitřní struktury obvodu brány s výstupní orientací. Podobně primární výstupy obvodu představují z hlediska vnitřní struktury obvodu brány se vstupní orientací. Nejlépe bude demonstrovat si tuto vlastnost na příkladu. Jako příklad můžeme opět použít obvod na obrázku 4.1 (strana 50). V uvedeném obvodu reprezentuje brána $uua.i1$ primární vstup obvodu UUA. Z hlediska vnitřní struktury obvodu (vzhledem ke vstupu $mux.a$ multiplexeru MUX) se ale tato brána chová jako brána výstupní – přivádí data na bránu $mux.a$. Podobně brána $uua.o1$ reprezentuje primární výstup obvodu. Z hlediska vnitřní struktury obvodu se ale chová jako vstupní brána – na tuto bránu jsou přiváděna data z výstupní brány $add.y$. Abychom mohli jednoduše vyjádřit chování bran rozhraní obvodu vzhledem k vnitřní struktuře obvodu, zavedeme nyní operátor „negace“, jehož aplikací na libovolný obvod získáme obvod s prázdnou vnitřní architekturou a rozhraním, které reprezentuje rozhraní původního obvodu vzhledem k vnitřní struktuře obvodu.

Definice 4.6: Mějme libovolný obvod $UUA = (I, E, C) = ((DI, DO, CI, CO), E, C)$, potom operátor „negace“ je definován takto: $\overline{UUA} = ((DO, DI, CO, CI), \emptyset, \emptyset)$.
□

Příklad 4.3: Mějme obvod UUA na obrázku 4.1, kde $UUA = (I, E, C)$ a $I = (\{(uua.i1, 8), (uua.i2, 8)\}, \{(uua.o1, 8)\}, \{(uua.sel, 1)\}, \emptyset)$.

Potom $\overline{UUA} = (\overline{I}, \emptyset, \emptyset)$, kde

$\overline{I} = (\{(uua.o1, 8)\}, \{(uua.i1, 8), (uua.i2, 8)\}, \emptyset, \{(uua.sel, 1)\})$.

Rozhraní obvodu $\overline{UUA}$ reprezentuje orientaci bran obvodu UUA vzhledem k vnitřní struktuře obvodu – brána $uua.o1$ se chová z hlediska vnitřní struktury obvodu UUA jako brána se vstupní orientací a brány $uua.i1$, $uua.i2$ a $uua.sel$ se chovají jako brány s výstupní orientací.

4.3.2 Rozdělení bran obvodu

Abychom mohli definovat množinu spojů obvodu, potřebujeme nejprve definovat množiny bran, které jsou spoji propojeny. Brány obvodu (včetně bran rozhraní) jsou podobně jako brány rozhraní v předchozí podkapitole zařazeny do odpovídajících množin podle své orientace (vstupní/výstupní) a typu brány (datová/řídicí). Při definici těchto množin využijeme operátor „negace“, který byl zaveden výše, abychom zařadili do správných množin také brány rozhraní obvodu.

Definice 4.7: Mějme obvod $UUA = (I, E, C)$. Pak množiny P_{DI}, P_{DO}, P_{CI}, P_{CO}, P_{IN}, P_{OUT} a P_{ALL} jsou definovány takto:

$P_{DI}(UUA) = \bigcup_{\forall e \in (E(UUA) \cup \{\overline{UUA}\})} I_{DI}(e)$,
kde P_{DI} je množina datových bran obvodu UUA se vstupní orientací,

$P_{DO}(UUA) = \bigcup_{\forall e \in (E(UUA) \cup \{\overline{UUA}\})} I_{DO}(e)$,
kde P_{DO} je množina datových bran obvodu UUA s výstupní orientací,

$P_{CI}(UUA) = \bigcup_{\forall e \in (E(UUA) \cup \{\overline{UUA}\})} I_{CI}(e)$,
kde P_{CI} je množina řídicích bran obvodu UUA se vstupní orientací,

$P_{CO}(UUA) = \bigcup_{\forall e \in (E(UUA) \cup \{\overline{UUA}\})} I_{CO}(e)$,
kde P_{CO} je množina řídicích bran obvodu UUA s výstupní orientací,

$P_{IN}(UUA) = P_{DI}(UUA) \cup P_{CI}(UUA)$,
kde P_{IN} je množina bran obvodu UUA se vstupní orientací,

$P_{OUT}(UUA) = P_{DO}(UUA) \cup P_{CO}(UUA)$,
kde P_{OUT} je množina bran obvodu UUA s výstupní orientací,

$P_{ALL}(UUA) = P_{IN}(UUA) \cup P_{OUT}(UUA)$,
kde P_{ALL} je množina všech bran obvodu UUA.

□

Nejlépe bude demonstrovat si opět vytvoření výše uvedených množin na příkladu.

Příklad 4.4: Mějme obvod UUA na obrázku 4.1 (strana 50). Vytvořme pro tento obvod množiny bran podle definice 4.7.

$$
\begin{aligned}
&P_{DI}(UUA) = \\
&\quad = I_{DI}(SUB) \cup I_{DI}(MUX) \cup I_{DI}(ADD) \cup I_{DI}(\overline{UUA}) = \\
&\quad = \{sub.a, sub.b\} \cup \{mux.a, mux.b\} \cup \{add.a, add.b\} \cup \{uua.o1\} = \\
&\quad = \{sub.a, sub.b, mux.a, mux.b, add.a, add.b, uua.o1\}, \\
&P_{DO}(UUA) = \\
&\quad = I_{DO}(SUB) \cup I_{DO}(MUX) \cup I_{DO}(ADD) \cup I_{DO}(\overline{UUA}) = \\
&\quad = \{sub.y\} \cup \{mux.y\} \cup \{add.y\} \cup \{uua.i1, uua.i2\} = \\
&\quad = \{sub.y, mux.y, add.y, uua.i1, uua.i2\}, \\
&P_{CI}(UUA) = \\
&\quad = I_{CI}(SUB) \cup I_{CI}(MUX) \cup I_{CI}(ADD) \cup I_{CI}(\overline{UUA}) = \\
&\quad = \emptyset \cup \{mux.sel\} \cup \emptyset \cup \emptyset = \{mux.sel\}, \\
&P_{CO}(UUA) =
\end{aligned}
$$

$$= I_{CO}(SUB) \cup I_{CO}(MUX) \cup I_{CO}(ADD) \cup I_{CO}(\overline{UUA}) =$$
$$= \emptyset \cup \emptyset \cup \emptyset \cup \{uua.sel\} = \{uua.sel\},$$
$$P_{IN}(UUA) = P_{DI}(UUA) \cup P_{CI}(UUA) =$$
$$= \{sub.a, sub.b, mux.a, mux.b, add.a, add.b, uua.o1, mux.sel\},$$
$$P_{OUT}(UUA) = P_{DO}(UUA) \cup P_{CO}(UUA) =$$
$$= \{sub.y, mux.y, add.y, uua.i1, uua.i2, uua.sel\},$$
$$P_{ALL}(UUA) = P_{IN}(UUA) \cup P_{OUT}(UUA) = \{sub.a, sub.b, mux.a, mux.b, add.a, add.b, uua.o1, mux.sel, sub.y, mux.y, add.y, uua.i1, uua.i2, uua.sel\}.$$

4.3.3 Reprezentace spojů obvodu

Nyní můžeme konečně přistoupit k formální definici relace C vyjadřující metalické spojení bran obvodu. Při její definici využijeme zavedených množin P_{IN} a P_{OUT}, které reprezentují množiny bran obvodu se vstupní a výstupní orientací. Konečná množina spojů C reprezentuje relaci $C \subseteq P_{IN} \times P_{OUT}$ takovou, že $x \in P_{IN}$ je v relaci s $y \in P_{OUT}$ když a jen když existuje metalický spoj mezi bránou x a y.

Definice 4.8: Mějme obvod $UUA = (I, E, C)$, kde relace $C = \{(x, y) | (x \in P_{IN}(UUA)) \wedge (y \in P_{OUT}(UUA)) \wedge$ *existuje metalický spoj mezi branami* x *a* $y\}$ reprezentuje spoje obvodu UUA.
□

Věta 4.1: Pokud je zapojení obvodu $UUA = (I, E, C)$ elektricky korektní, jsou zapojeny všechny brány obvodu a v obvodu nejsou použity třístavové sběrnice, představuje C surjektivní zobrazení z množiny $P_{IN}(UUA)$ na množinu $P_{OUT}(UUA)$.
□

Důkaz: Podle definice 4.8 je $C \subseteq (P_{IN} \times P_{OUT})$ relace. Pokud v obvodu nejsou použity třístavové sběrnice a obvod je elektricky korektní, tak musí platit, že libovolná vstupní brána může být připojena k maximálně jedné výstupní bráně. Pokud tedy pro libovolnou vstupní bránu $x \in P_{IN}$ platí, že $y_1, y_2 \in P_{OUT} \wedge (x, y_1) \in C \wedge (x, y_2) \in C$, pak z předpokladu, že v obvodu nejsou použity třístavové sběrnice, vyplývá, že $y_1 = y_2$ a relace C tak představuje zobrazení. Z předpokladu, že jsou zapojeny všechny brány obvodu pak vyplývá, že C je surjektivní zobrazení.
□

Model struktury číslicového obvodu na úrovni RT tak máme nyní kompletní. Pro účely analýzy testovatelnosti bude v další části této kapitoly ještě zavedený model rozšířen o možnost modelování transparentních cest obvodu.

4.4 Model transparentních cest

Existuje několik přístupů k modelování transparentních cest v obvodu. Nejrozšířenější je koncept tzv. *i-režimu* a *i-cest* zavedený Abadirem [2] a dále rozšířený např. v [58, 81, 86]). Kromě uvedeného modelu však existuje také řada dalších modelů (např. *T-cesta*, *S-cesta*, ...) [2, 8] a práce, které na tyto modely navazují. Rozdíl mezi jednotlivými modely spočívá ve způsobu

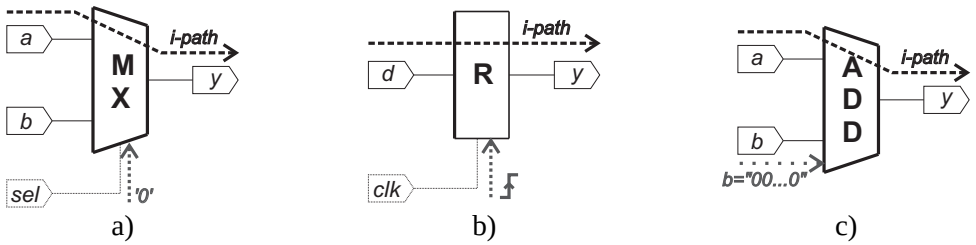
modelování transparentních cest. Obecně platí, že rozšíření konceptu transparentnosti směrem k realističtějšímu modelování transparentních cest sebou obvykle přináší vyšší časovou složitost práce s tímto modelem. Je tak potřeba najít vhodný kompromis mezi přesností použitého modelu a časovou složitostí práce s tímto modelem.

Vzhledem k tomu, že tato práce jistým způsobem navazuje na práce [58, 81, 86], bude v této práci použit model transparentních cest založený na tzv. *i-režimu* a *i-cestě*, který podle výsledků prezentovaných v [58, 81] poskytuje pro účely této práce vhodný kompromis mezi přesností a časovou náročností práce s tímto modelem. Pro definici pojmů souvisejících s modelováním transparentních cest v obvodu bude použit formální model obvodu zavedený v této kapitole.

Definice 4.9: Mějme prvek $UUA = (I, E, C)$. V prvku UUA se vstupní bránou $x \in I_{IN}(UUA)$ a výstupní bránou $y \in I_{OUT}(UUA)$ existuje *i-režim* mezi branami x a y, pokud existuje režim činnosti prvku UUA, při kterém je možné přenést libovolná data z brány x na bránu y, aniž by tato data byla změněna.
□

Podle typu podmínky, která je potřeba pro nastavení *i-režimu* můžeme prvky rozdělit do tří kategorií [58, 81, 86]:

1. prvky s *i-režimem* závislým na hladinových řídicích vstupech (viz obrázek 4.2(a)),
2. prvky s *i-režimem* závislým na hranových řídicích (resp. synchronizačních) vstupech (viz obrázek 4.2(b)),
3. prvky s *i-režimem* závislým na datových vstupech (viz obrázek 4.2(c)).



Obr. 4.2: Příklady různých typů *i-režimů*.

Abychom mohli dále pracovat s *i-režimy* prvků, zavedeme zápis umožňující formálně reprezentovat *i-režim* prvku.

Definice 4.10: Nechť $x \in I_{IN}(UUA)$ je vstupní brána prvku UUA, $y \in I_{OUT}(UUA)$ je výstupní brána prvku UUA, $conds \in 2^{I_{IN}(UUA)}$ je množina identifikující vstupní brány potřebné pro nastavení *i-režimu* a $depth \in \mathbb{N}$ je počet taktů hodin potřebných pro přenos dat ze vstupu x na výstup y (sekvenční hloubka prvku). Pak *i-režim* i_1 prvku $UUA = (I, E, C)$ je popsán čtveřicí $i_1 = (x, y, conds, depth)$.
□

Definice 4.11: Množinu $\mathcal{I}_{modes}(UUA)$ budeme označovat jako množinu všech *i-režimů* prvku UUA.
□

Příklad 4.5: Mějme prvky na obrázku 4.2. Použijme aparát zavedený v definici 4.10 a 4.11 pro popis *i-režimů* těchto prvků.

a) dvouvstupý 8 bitový multiplexor MX –
$MX = (I, E, C) = ((\{(mx.a, 8), (mx.b, 8)\}, \{(mx.y, 8)\}, \{(mx.sel, 1)\}, \emptyset), \emptyset, \emptyset)$,
$i_1(MX) = (mx.a, mx.y, \{mx.sel\}, 0)$,
$i_2(MX) = (mx.b, mx.y, \{mx.sel\}, 0)$,
$\mathcal{I}_{modes}(MX) = \{(mx.a, mx.y, \{mx.sel\}, 0), (mx.b, mx.y, \{mx.sel\}, 0)\}$

b) 8 bitový registr R –
$R = (I, E, C) = ((\{(r.d, 8)\}, \{(r.y, 8)\}, \{(r.clk, 1)\}, \emptyset), \emptyset, \emptyset)$,
$i_1(R) = (r.d, r.y, \{r.clk\}, 1)$,
$\mathcal{I}_{modes}(R) = \{(r.d, r.y, \{r.clk\}, 1)\}$

c) 8 bitová sčítačka ADD –
$ADD = (I, E, C) = ((\{(add.a, 8), (add.b, 8)\}, \{(add.y, 8)\}, \emptyset, \emptyset), \emptyset, \emptyset)$,
$i_1(ADD) = (add.a, add.y, \{add.b\}, 0)$,
$i_2(ADD) = (add.b, add.y, \{add.a\}, 0)$,
$\mathcal{I}_{modes}(ADD) = \{(add.a, add.y, \{add.b\}, 0), (add.b, add.y, \{add.a\}, 0)\}$

Nyní máme k dispozici aparát, který nám umožňuje formálně popsat transparentní cesty ve struktuře prvku. Můžeme tak postoupit o krok dále a to definici pojmu *i-cesty*, která je vlastně zobecněním *i-režimu* prvku na strukturu obvodu. *I-cestu* si je možné představit jako řetězec prvků s aktivovaným *i-režimem*. Ke každé *i-cestě* náleží tzv. aktivační plán, který obsahuje informace o hodnotách a časování řídicích a datových signálů, které jsou potřeba pro nastavení *i-cesty*.

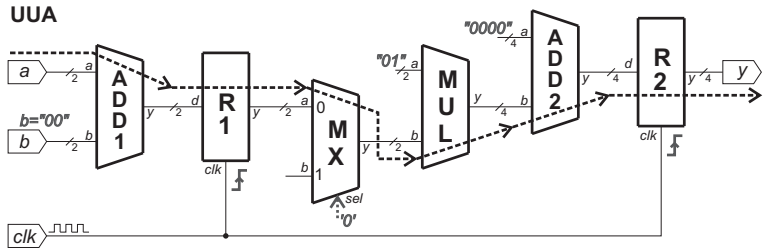
Definice 4.12: V obvodu $UUA = (I, E, C)$ existuje *i-cesta* (z angl. *identity-transfer path – i-path*) z brány $x \in P_{ALL}(UUA)$ na bránu $y \in P_{ALL}(UUA)$, pokud je možné přenést libovolná data z brány x na bránu y, aniž by tato data byla změněna.
□

Z hlediska modelování průchodu diagnostických dat obvodem pomocí konceptu *i-cest* je pro nás důležitá informace o počáteční bráně *i-cesty* $g_1 \in P_{ALL}$, cílové bráně *i-cesty* $g_2 \in P_{ALL}$, informace o branách, které je potřeba řídit pro nastavení *i-cesty* a informace o počtu taktů, které jsou potřeba pro přenos dat z brány g_1 na bránu g_2.

Definice 4.13: Nechť g_1 je počáteční brána *i-cesty* ($g_1 \in P_{ALL}$), g_2 je cílová brána *i-cesty* ($g_2 \in P_{ALL}$), $cond$ je množina, která identifikuje vstupní brány, které je nutné řídit pro nastavení *i-cesty* z brány g_1 na bránu g_2 ($cond \in 2^{P_{IN}}$) a seq je sekvenční hloubka *i-cesty*. Pak *i-cesta* v obvodu $UUA = (I, E, C)$ je definována jako čtveřice $(g_1, g_2, cond, seq)$.
□

Definice 4.14: Množinu $\mathcal{I}_{paths}(UUA)$ budeme označovat jako množinu všech *i-cest* v obvodu UUA.
□

Příklad *i-cesty* v obvodu na úrovni RT je na obrázku 4.3. Uvedená *i-cesta* využívá *i-režimu* celkem šesti prvků. Existence této *i-cesty* je tak podmíněna nastavením *i-režimu* u všech prvků *i-cesty*.



Obr. 4.3: Příklad *i-cesty* v obvodu na úrovni RT.

Příklad 4.6: Mějme *i-cestu* v obvodu na obrázku 4.3. Použijme formální aparát zavedený výše pro popis této *i-cesty*.

***i-režimy* jednotlivých prvků –**

$\mathcal{I}_{modes}(ADD1) = \{(add1.a, add1.y, \{add1.b\}, 0), (add1.b, add1.y, \{add1.a\}, 0)\}$,
$\mathcal{I}_{modes}(R1) = \{(r1.d, r1.y, \{r1.clk\}, 1)\}$,
$\mathcal{I}_{modes}(MX) = \{(mx.a, mx.y, \{mx.sel\}, 0), (mx.b, mx.y, \{mx.sel\}, 0)\}$,
$\mathcal{I}_{modes}(MUL) = \{(mul.a, mul.y, \{mul.b\}, 0), (mul.b, mul.y, \{mul.a\}, 0)\}$,
$\mathcal{I}_{modes}(ADD2) = \{(add2.a, add2.y, \{add2.b\}, 0), (add2.b, add2.y, \{add2.a\}, 0)\}$,
$\mathcal{I}_{modes}(R2) = \{(r2.d, r2.y, \{r2.clk\}, 1)\}$

***i-cesta* z brány *uua.a* na bránu *uua.y* –**

$I_{path} = (uua.a, uua.y, \{add1.b, r1.clk, mx.sel, mul.a, add2.a, r2.clk\}, 2)$

4.5 Shrnutí

V této kapitole byl představen formální model umožňující popsat strukturu obvodu na úrovni RT a modelovat transparentní vlastnosti prvků na této úrovni. Formální model představený v této kapitole navazuje na dříve publikovaný model zavedený v práci [81] a později rozšířený v práci [86]. Nevyužití již existujících modelů nebylo samoúčelné, ale bylo vyvoláno potřebou modelovat další skutečnosti týkající se struktury obvodu a jeho vlastností. Za nejdůležitější rozšíření navrženého modelu oproti původním modelům je možné považovat následující skutečnosti:

- **hierarchický popis struktury obvodu** – navržený model je možné použít pro hierarchický popis struktury obvodu. Model je možné použít pro modelování základního prvku úrovně RT, modulu skládajícího se z dalších prvků, ale také pro popis samotného obvodu na úrovni RT,
- **kompaktní popis struktury obvodu** – struktura obvodu je vyjádřena trojicí reprezentující informaci o rozhraní obvodu, prvcích, z nichž se obvod skládá a informaci o vzájemném propojení těchto prvků. Pro jednoznačný popis struktury obvodu není potřeba zavádět žádné další pomocné zobrazení, jak tomu bylo např. v [81, 86],
- **informace o bitové šířce bran obvodu** – pro každou bránu obvodu je modelována její bitová šířka,
- **rozlišení řídicích a datových bran** – model umožňuje rozlišit řídicí a datové brány, což je možné využít při analýze testovatelnosti obvodu.

Kapitola 5

Metoda návrhu testovacích obvodů

V předchozí části této práce byl představen aktuální stav a zavedeny základní pojmy z oblastí úzce souvisejících s tématem této práce. Výsledkem analýzy aktuálního stavu v oblasti existujících sad testovacích obvodů bylo zjištění, že v současné době nejsou k dispozici testovací obvody potřebné složitosti, které by bylo možné použít pro ověřování diagnostických metod a nástrojů. Možným řešení tohoto stavu je využít pro ověřování diagnostických metod a nástrojů tzv. syntetické testovací obvody. Aktuálně však v oblasti syntetických testovacích obvodů neexistuje metoda, která by návrh vhodných testovacích obvodů umožňovala. Dostupné metody návrhu syntetických testovacích obvodů jsou obvykle založeny pouze na řízené modifikaci struktury již existujících obvodů nebo jsou schopny navrhovat obvody při splnění pouze jednoduchých strukturních vlastností. Návrh syntetických testovacích obvodů s požadovanými diagnostickými vlastnostmi, které by byly vhodné pro ověřování diagnostických metod a nástrojů, však znamená mnohem komplexnější problém. Abychom se mohli zabývat návrhem takového typu testovacích obvodů, bylo potřeba vytvořit nový přístup k jejich vytváření.

5.1 Specifikace požadavků

Cílem této práce je navrhnout, formálně popsat, implementovat a experimentálně ověřit metodu návrhu syntetických testovacích obvodů, která bude umožňovat automatické vytváření testovacích obvodů s požadovanou strukturou a diagnostickými vlastnostmi. Vytvářené obvody budou reprezentovat datovou část obvodu na úrovni RT. Pokud bychom měli formulovat obecné požadavky na testovací obvody, můžeme vycházet z požadavků specifikovaných pro testovací obvody ze sady ITC'99. Testovací obvody vytvářené navrženou metodou by měly splňovat následující požadavky [102]:

- obvody na úrovni RT popsané v jazyce VHDL,
- syntetizovatelné bez použití specifických direktiv překladu,
- využívají pouze základní knihovny standardu IEEE (*std_logic* a *std_arith*),
- musí představovat plně synchronní obvody s jednou hodinovou doménou,
- nesmí používat 3-stavové sběrnice,
- nevyužívají tzv. drátovou logiku (tzv. logika wire-AND, wire-OR).

Pokud budeme vycházet z požadavků návrháře, tak základními vstupními parametry návrhové metody bude informace o složitosti a požadované struktuře testovacích obvodů, která bude vyjádřena ve formě počtu primárních vstupů a výstupů obvodu a dále počtu a typu prvků z nichž se má obvod skládat. Přičemž prvky obvodu mohou být reprezentovány základními prvky úrovně RT (registry, sčítačky, násobičky, dekodéry, . . .) nebo mohou reprezentovat složitější obvodové konstrukce. Dalším důležitým požadavkem návrháře je specifikace diagnostických vlastností vytvářených obvodů. Vzhledem k tomu, že zatím stále neexistuje všeobecně uznávaná míra vyjadřující diagnostické vlastnosti obvodu, je otázkou, jak požadované diagnostické vlastnosti specifikovat. Pro účely této práce byla zvolena specifikace ve formě parametrů řiditelnosti a pozorovatelnosti bran obvodu (viz podkapitola 3.4, strana 50).

Pro specifikaci uživatelem definovaných požadavků je použit formát jazyka XML. Příklad specifikace požadavků uživatele je na obrázku 5.1. Uživatel ve vstupním souboru specifikuje počet primárních vstupů a výstupů obvodu a také prvky, z nichž se má obvod skládat. Kromě samotné informace o struktuře obvodu má uživatel možnost specifikovat také parametry, které jsou předány použitému optimalizačnímu algoritmu.

```
<circuit>
  <testability controllability="0.5" observability="0.5"/>
  <evolution population="30" replacement="0.95"
             mutation="0.02" steps="200"/>
  <primary inputs="5" outputs="5"/>
  <comp name="SUB" width="8" quantity="10"/>
  <comp name="ADD" width="16" quantity="10"/>
  <comp name="MUL" width="8,16" quantity="10"/>
  <comp name="MUX2" width="8" quantity="5"/>
  <comp name="MUX2" width="16" quantity="5"/>
</circuit>
```

Obr. 5.1: Příklad specifikace požadavků uživatele.

Vzhledem k tomu, že rozhraní a prvky, z nichž se obvod skládá, jsou dány požadavky uživatele, můžeme problém návrhu syntetických obvodů formulovat jako problém nalezení nejvhodnějšího propojení uživatelem zadaných prvků tak, aby výsledný obvod splňoval požadované diagnostické vlastnosti.

5.2 Metoda návrhu

Existující metody návrhu syntetických testovacích obvodů většinou vycházejí z určitého vzorového obvodu, analyzují jeho charakteristické vlastnosti (kombinační zpoždění, větvení obvodu, apod.) a na základě této analýzy řízeně modifikují strukturu obvodu tak, aby vznikl obvod s novou strukturou, ale zůstaly zachovány charakteristické vlastnosti původního obvodu. Testovatelnost obvodu (ať již je měřena průměrnou řiditelností a pozorovatelností bran obvodu, pokrytím poruch nebo jinou mírou) ale bohužel představuje poměrně komplexní parametr, u kterého obvykle není možné předem přesně určit, jak bude tento parametr ovlivněn určitou změnou v obvodové struktuře. Použití některé z existujících analytických metod návrhu syntetických testovacích obvodů je tak bohužel obtížně realizovatelné.

Určité výzkumné úsilí bylo v rámci této práce věnováno možnosti využít některou z existu-

jících metod návrhu syntetických testovacích obvodů a realizovat řízenou modifikaci struktury vzorového obvodu na základě výsledků inkrementální analýzy testovatelnosti, která dokáže na základě aktuálního ohodnocení testovatelnosti obvodu a provedené změny struktury obvodu ohodnotit testovatelnost výsledného obvodu. Bohužel analýza dopadů provedených změn na testovatelnost jednotlivých uzlů obvodu byla pro metody inkrementální analýzy testovatelnosti navržené v rámci uvedeného výzkumu ve většině případů časově náročnější, než samotná analýza testovatelnosti. Existující metody návrhu syntetických testovacích obvodů tak nebylo možné použít.

5.3 Evoluční návrh testovacích obvodů

V rámci této práce byla vytvořena nová metoda vytváření syntetických testovacích obvodů, která je založena na tzv. evolučním návrhu (viz podkapitola 3.3). Navržená metoda realizuje vytváření struktury obvodu jako optimalizační proces, který hledá nejvhodnější propojení zadaných prvků tak, aby byly splněny uživatelem specifikované požadavky na testovatelnost. Jako optimalizační algoritmus je v této metodě použit algoritmus evolučního programování (viz podkapitola 2.6.2). Dílčí experimenty ale byly realizovány také s optimalizačním algoritmem simulovaného žíhání (viz podkapitola 7.6). Přístup založený na evolučním algoritmu byl zvolen, protože evoluce představuje obecný nástroj pro řešení problémů [27]. Je pravda, že evoluční algoritmus je jedna z mnoha optimalizačních technik a obvykle není možné obecně říct, která technika je nejvhodnější pro daný typ nebo třídu problémů. Podle [6] je však možné identifikovat techniky, které pro širokou škálu různých problémů vedou k lepšímu řešení než techniky ostatní. Právě evoluční techniky, jejichž úspěšné použití bylo demonstrováno na stovkách různých typů problémů, do této kategorie patří. V některých oblastech byly úspěšně použity také další algoritmy (např. horolezecký algoritmus, simulované žíhání, atd.), pouze u evolučních algoritmů je však možné říci, že byly obecně úspěšné [6]. Tato teorie se nakonec potvrdila i v této práci při experimentálním srovnání optimalizační metody simulovaného žíhání a evolučního programování (viz podkapitola 7.6).

Optimalizační algoritmus evolučního programování (EP), který je použit pro návrh struktury testovacích obvodů, je speciálním případem evolučního algoritmu. Pro evoluční programování jsou typické následující skutečnosti (podrobnosti viz podkapitola 2.6):

- **zakódování problému** – EP obvykle pracuje přímo nad reálnou reprezentací problému (nevyužívá např. zakódování problému do binární podoby). V našem případě pracuje evoluční programování přímo se spoji obvodu,
- **výběr jedinců** – výběr jedinců, kteří se zúčastní reprodukce, je realizován turnajem (jsou náhodně vybráni 2 jedinci, z nichž ten s vyšší hodnotou fitness se účastní reprodukce),
- **reprodukční operátor** – jako reprodukční operátor je u EP použit pouze operátor mutace.

Princip algoritmu, který je použit pro návrh obvodů v této práci, je naznačen na obrázku 5.2. Nejprve je vytvořena počáteční populace P tvořená obvody, které vznikly náhodným propojením uživatelem specifikovaných prvků a primárních vstupů a výstupů obvodu. Dále je ohodnocena kvalita každého obvodu (kandidátního řešení) pomocí tzv. *fitness funkce* (hodnotící funkce). Fitness funkce v našem případě ohodnotí kandidátní řešení reálným číslem v rozsahu $\langle 0, 1 \rangle$ podle toho, jak kvalitní řešení daný jedinec představuje.

```
Vytvoření počáteční populace P skládající se z N jedinců
Ohodnocení kvality jedinců v populaci P
t := 0
while (není splněna podmínka ukončení) do {
  // Vytvoření populace potomků
  P' := ∅
  while (|P'| < R*N) do {
    i := turnajový výběr jedince z populace P
    i':= mutace M% spojů jedince i
    Ohodnocení kvality jedince i'
    Přidání i' do populace P'
  }
  // Proces nahrazení rodičovské populace
  for (∀i1 ∈ P') do {
    i2 = náhodně vybraný jedinec z populace P
    i = turnajový výběr mezi i1 a i2
    Nahrazení i2 jedincem i v populaci P
  }
  t := t + 1
}
Ulož obvod reprezentovaný nejlepším jedincem v jazyce VHDL
```

Obr. 5.2: Princip algoritmu návrhu syntetických testovacích obvodů.

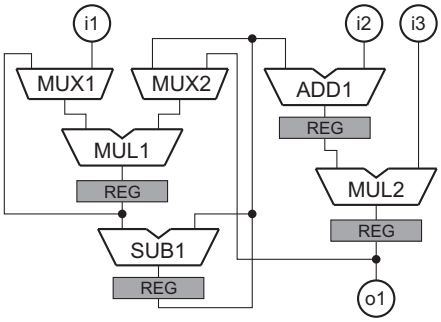
Pak je zahájen vlastní evoluční proces, který probíhá tak dlouho, dokud není buď nalezeno požadované řešení nebo není dosaženo zadaného počtu generací. V každém kroku (generaci) vzniká na základě aplikace operátorů selekce a mutace na aktuální rodičovské populaci P tzv. populace potomků P'. Ve chvíli, kdy dosáhne populace potomků požadované velikosti, nastává tzv. proces nahrazení. V našem případě je proces nahrazení implementován tak, že jsou postupně vybíráni jedinci z populace potomků a tito jedinci pak dále soupeří s náhodně vybraným jedincem z populace rodičů. Pokud představuje jedinec z populace potomků lepší řešení, nahradí jedince z populace rodičů.

V současné době uživatel nemá možnost specifikovat počet registrů, které budou v obvodu použity. Registry jsou umístěny automaticky návrhovou metodou na výstupy všech kombinačních prvků (kromě multiplexerů, které jsou považovány za propojovací prvky obvodu) před provedením analýzy testovatelnosti. Pokud existuje více možností jak registry umístit, je vybrána varianta s nejnižším počtem registrů. Příklad automatického umístění registrů v obvodu je na obrázku 5.3.

Na závěr je obvod reprezentovaný nejlepším jedincem uložen ve formě strukturou popsaného VHDL kódu. Jednotlivé části návrhového procesu budou podrobněji popsány v následujících podkapitolách.

5.4 Reprodukční operátory

Základem metody návrhu je reprodukční operátor mutace, který realizuje změny v propojení jednotlivých prvků obvodu a vytváří tak nová kandidátní řešení. Navržená metoda využívá dva typy operátoru mutace. Výběr operátoru, který bude aplikován na daný obvod, je dán náhodným výběrem. Počet mutovaných spojů je dán parametrem M, který vyjadřuje jaká procentuální část



Obr. 5.3: Příklad automatického rozmístění registrů v obvodu.

spojů má být v každém kroku změněna. Při aplikaci operátorů je vždy zohledněna datová šířka bran obvodu a struktura spojů, na něž je operátor mutace aplikován. Operátory byly navrženy tak, aby aplikací jejich nekonečné posloupnosti bylo zajištěno, že se realizují všechny možné kombinace spojů obvodu. Při popisu operátorů mutace je využit formální model zavedený v předchozí kapitole.

5.4.1 Mutace I. – přepojení jedné brány

První typ operátoru mutace je aplikován na jednu bránu obvodu a způsobí přepojení této brány na jiný spoj. Operátor funguje tak, že nejprve náhodně vybere bránu obvodu se vstupní orientací $x_1 \in P_{DI}$. V dalším kroku ověří, zda je možné na vybranou bránu tento typ mutace aplikovat. Podmínkou aplikovatelnosti je, že výstupní brána $y_1 \in P_{DO}$, s níž je vybraná brána x_1 spojena, je připojena k alespoň jedné další vstupní bráně a přepojením brány x_1 na jiný spoj tak nenastane stav, kdy by brána y_1 zůstala nezapojena. Po ověření aplikovatelnosti operátoru již následuje výběr brány $y_2 \in P_{DO}$ takové, že $(x_1, y_2) \notin C$, se stejnou bitovou šířkou jakou má brána y_1. Pokud existuje brána y_2 splňující zadané požadavky, je provedeno vlastní přepojení brány x_1 z brány y_1 na bránu y_2. Algoritmus mutace je popsán v algoritmu 1. Časová složitost algoritmu je lineární vzhledem k počtu spojů obvodu $O(|C|)$.

Algoritmus 1:

Procedura *Operátor_mutace_1(***var:** UUA*)* {

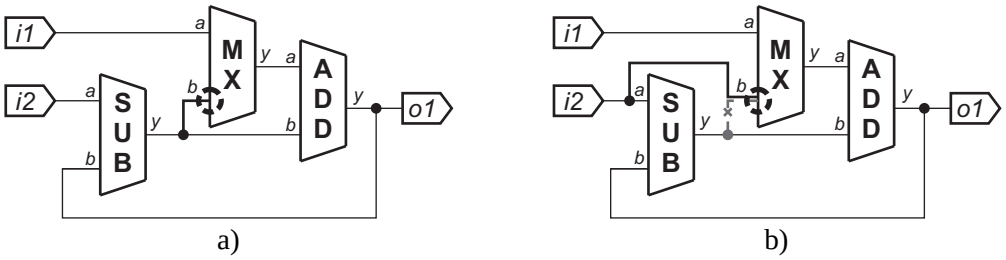
1. [*Inicializace operátoru mutace*]
   Náhodný výběr brány $x_1 \in P_{DI}$
   Vyber $y_1 \in P_{DO}$ takové, že $(x_1, y_1) \in C$

2. [*Ověření aplikovatelnosti operátoru mutace*]
   Pokud $|\{x_2|(x_2, y_1) \in C\}| < 2$ **potom** KONEC

3. [*Nalezení brány y_2 na kterou bude brána x_1 přepojena*]
   $Y = \{y_2|(x_1, y_2) \notin C \wedge \Phi(x_1) = \Phi(y_2)\}$
   Pokud $|Y| = 0$ **potom** KONEC
   Náhodný výběr brány $y_2 \in Y$

4. [*Aplikace operátoru mutace*]
$C \leftarrow (C \setminus \{(x_1, y_1)\}) \cup \{(x_1, y_2)\}$

}



Obr. 5.4: Ukázka fungování operátoru mutace – přepojení jedné brány.

Použití prvního typu mutace si ukážeme na příkladu obvodu na obrázku 5.4. Předpokládejme, že všechny datové cesty obvodu mají stejnou šířku. Aplikace operátoru mutace probíhá v těchto krocích: (1) náhodný výběr brány se vstupní orientací – např. $mx.b$, (2) kontrola aplikovatelnosti operátoru mutace – operátor je možné na tuto bránu aplikovat, protože kromě brány $mx.b$, je na stejný spoj (výstupní bránu $sub.y$) připojena také brána $add.b$. Spoj tedy zůstane zachován i po přepojení brány $mx.b$ na jinou výstupní bránu. (3) Náhodný výběr brány obvodu s výstupní orientací a stejnou datovou šířkou jakou má brána $mx.b$. V našem případě mají všechny brány stejnou datovou šířku. Platí tedy, že $Y = \{i1, i2, mx.y, add.y\}$. Zvolme náhodně například bránu $i2$. (4) Brána $mx.b$ je přepojena z brány $sub.y$ na bránu $i2$.

5.4.2 Mutace II. – přepojení dvojice bran

Druhý typ mutace je na rozdíl od prvního typu aplikován na dvojici bran a zajišťuje vzájemnou záměnu spojů přivedených k těmto branám. Operátor pracuje tak, že v obvodě náhodně vybere dvě brány se vstupní orientací (např. $x_1, x_2 \in P_{DI}$) a stejnou datovou šířkou $\Phi(x_1) = \Phi(x_2)$ a zamění spoje, které jsou k těmto branám přivedeny. Fungování operátoru můžeme zapsat algoritmem 2. Algoritmus má lineární časovou složitost vzhledem k počtu spojů obvodu $O(|C|)$.

Algoritmus 2:

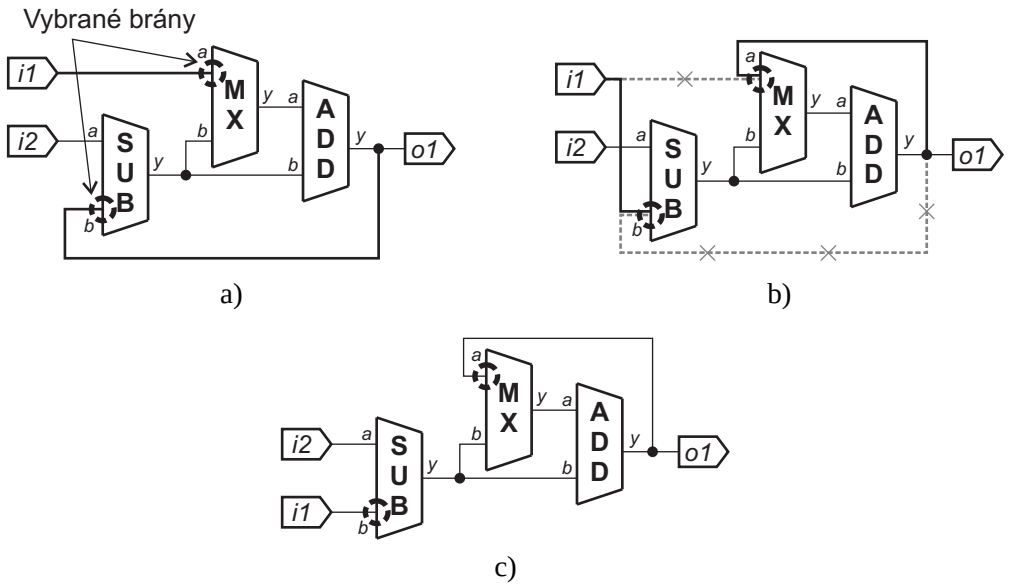
Procedura *Operátor_mutace_2(***var:** UUA*)* {

1. [*Výběr dvojice bran na které bude operátor aplikován*]
Vyber $x_1, x_2 \in P_{DI}$ **takové, že**
$(x_1, y_1) \in C \wedge (x_2, y_2) \in C \wedge y_1 \neq y_2 \wedge \Phi(x_1) = \Phi(x_2)$

2. [*Aplikace operátoru mutace*]
$C \leftarrow (C \setminus \{(x_1, y_1), (x_2, y_2)\}) \cup \{(x_1, y_2), (x_2, y_1)\}$

}

Aplikace druhého typu operátoru mutace je demonstrována na obvodu na obrázku 5.5. Opět předpokládejme, že všechny datové cesty v obvodu mají stejnou šířku. V prvním kroku jsou náhodně vybrány brány $sub.b$ a $mx.a$ splňující požadavky operátoru mutace (viz obrázek 5.5(a)). V druhém kroku je brána $sub.b$ přepojena na výstup $i1$ místo brány $mx.a$ a brána



Obr. 5.5: Ukázka fungování operátoru mutace – přepojení dvojice bran.

$mx.a$ je přepojena na výstup $add.y$ místo brány $sub.b$. Přepojení těchto bran je znázorněno na obrázku 5.5(b). Výsledný obvod je překreslen na obrázku 5.5(c).

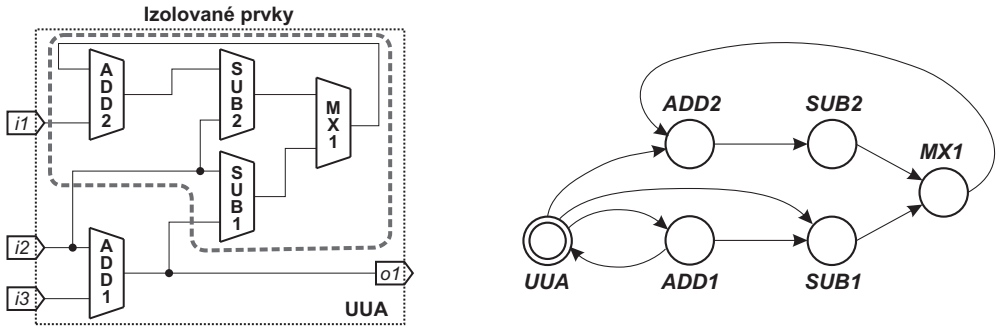
5.5 Ohodnocení kvality kandidátních řešení

Ohodnocení kvality kandidátních řešení je nejdůležitější částí navržené metody. Použití vhodné metody ohodnocení kandidátních obvodů má významný vliv na kvalitu vytvářených obvodů, protože ovlivňuje pravděpodobnost, že se daný jedinec zúčastní reprodukce a jeho genetická informace se tak bude dále šířit populací. Ohodnocení jedince je realizováno pomocí tzv. fitness funkce a probíhá postupně ve třech krocích: (1) ohodnocení struktury obvodu, (2) ohodnocení propojení prvků obvodu a (3) ohodnocení testovatelnosti obvodu. První dva kroky hodnotí kvalitu kandidátního řešení z hlediska struktury obvodu – hodnotí možnosti realizace obvodu, způsob propojení prvků a konstrukce použité ve struktuře obvodu. Třetím krokem je ohodnocení, do jaké míry se diagnostické vlastnosti vytvářených obvodů liší od požadavků specifikovaných uživatelem. Prvním dvěma krokům se budou věnovat následující podkapitoly. Ohodnocení testovatelnosti bude věnovaná samostatná kapitola.

5.5.1 Analýza struktury obvodu

Cílem **analýzy struktury obvodu** je identifikace tzv. *izolovaných prvků* ve struktuře obvodu. Izolovaným prvkem se v tomto kontextu rozumí prvek, jež není přímo ani nepřímo (prostřednictvím dalších prvků) spojen s výstupy obvodu a neovlivňuje tak výstupní funkci realizovanou obvodem. Příklad obvodu s izolovanými prvky je na obrázku 5.6(a). Izolovanými prvky v tomto obvodu jsou prvky $ADD2$, $SUB2$, $SUB1$ a $MX1$.

Cílem této práce je navrhovat obvody, které neobsahují izolované prvky, protože tyto prvky jsou v průběhu syntézy do nižších úrovní popisu obvodu návrhovými nástroji automaticky odstraněny. Existuje několik možností, jak izolované prvky v obvodu identifikovat. Algoritmus



a) b)

Obr. 5.6: Příklad obvodu s izolovanými prvky (a) a grafová reprezentace obvodu (b).

navržený v této práci využívá toho, že strukturu obvodu je možné chápat jako orientovaný graf, kde prvky obvodu reprezentují uzly tohoto grafu a orientované hrany grafu reprezentují datové spoje. Problém identifikace izolovaných prvků pak můžeme převést na problém analýzy souvislosti grafu.

Algoritmus identifikace izolovaných prvků v obvodu použitý v této práci je založen na analýze orientovaných cest v grafu reprezentujícím strukturu obvodu. Navržený algoritmus se skládá ze tří částí. V první části je sestaven orientovaný graf $IC = (V, H)$. Uzly grafu IC představují prvky E obvodu a prvek $\overline{UUA}$ reprezentující rozhraní samotného obvodu – formálně zapsáno $V = E \cup \{\overline{UUA}\}$. Orientované hrany grafu IC představují datové spoje obvodu. Mezi dvěma uzly grafu $e_1, e_2 \in V$ existuje orientovaná hrana, pokud existuje datová cesta z některé výstupní brány e_1, která vede na některou vstupní bránu prvku e_2. V druhém kroku probíhá sestavení tranzitivního uzávěru IC' grafu IC pomocí některého ze známých algoritmů (viz např. [74]) a na konec jsou identifikovány izolované prvky obvodu jako uzly grafu IC', pro něž neexistuje orientovaná hrana do uzlu reprezentujícího prvek $\overline{UUA}$.

Algoritmus 3:

Procedura *Identifikace_izolovaných_prvků(UUA;* **var:** $E_{isolated}$*)* {

1. [*Sestavení orientovaného grafu IC=(V,H)*]
   $V = (E \cup \{\overline{UUA}\})$,
   $H = \{(x, y) | x, y \in V \wedge (u, v) \in C \wedge u \in I_{DI}(y) \wedge v \in I_{DO}(x)\}$,
   $IC = (V, H)$.

2. [*Vytvoření tranzitivního uzávěru IC' grafu IC*]
   $V' = V$
   $H' = \{(u, v) |$ existuje orientovaná cesta z u do v v grafu $IC\}$.
   $IC' = (V', H')$

3. [*Vytvoření množiny izolovaných prvků v obvodu*]
   $E_{isolated} = \{e | (e \in (V' \setminus \{\overline{UUA}\})) \wedge ((e, \overline{UUA}) \notin H')\}$

}

Analyzujme nyní časovou složitost algoritmu. První krok algoritmu má časovou složitost $O(|E| + |C|)$, protože pro vytvoření grafu IC je potřeba projít celou množinu prvků a spojů

obvodu. Druhým krokem je vytvoření tranzitivního uzávěru prostřednictvím Warshallova algoritmu, který pracuje s časovou složitostí $O(|E|^3)$. Posledním krokem je detekce izolovaných prvků, která je realizována průchodem hran grafu IC', který může mít maximálně $|E|^2$ hran – časová složitost $O(|E|^2)$. Výsledná celková časová složitost algoritmu je $O(|E|^3 + |C|)$.

Analýza struktury obvodu – ukázka

Princip algoritmu identifikace izolovaných prvků bude demonstrován na obvodu na obrázku 5.6(a). Na strukturu obvodu $UUA = (I, E, C)$ na obrázku 5.6(a) můžeme nahlížet jako na orientovaný graf $IC = (V, H)$, kde uzly grafu reprezentují prvky obvodu ($V = E \cup \{\overline{UUA}\}$) a hrany grafu reprezentují spoje v obvodu (viz obrázek 5.6(b)). Samotný obvod UUA je reprezentován v grafu uzlem ($\overline{UUA} \in V$). Tím je umožněno modelování spojů mezi primárními branami obvodu a prvky obvodu. Mezi dvěma uzly e_1 a $e2$ existuje orientovaná hrana (e_1, e_2), pokud existuje v obvodu UUA spoj z výstupu prvku odpovídající uzlu e_1 na vstup prvku odpovídajícímu uzlu e_2. Datovou část obvodu UUA můžeme reprezentovat grafem $IC = (V, H)$, kde $V = \{ADD1,\ ADD2,\ SUB1,\ SUB2,\ MX1,\ \overline{UUA}\}$ a $V = \{(\overline{UUA}, ADD1), (\overline{UUA}, ADD2), (\overline{UUA}, SUB1), (ADD1, \overline{UUA}), (ADD1, SUB1), (SUB1, MX1), (ADD2, SUB2), (SUB2, MX1), (MX1, ADD2)\}$ (viz obrázek 5.6(b)).

Problém identifikace izolovaných prvků obvodu nyní může být formulován jako problém dosažitelnosti uzlu $\overline{UUA}$ z jednotlivých uzlů grafu IC. Pokud uzel $\overline{UUA}$ není dosažitelný z uzlu $e \in V$, reprezentuje uzel e izolovaný prvek. Úlohu dosažitelnosti uzlu grafu budeme řešit tak, že nejprve sestavíme tranzitivní uzávěr orientovaného grafu IC. Tranzitivní uzávěr grafu $IC = (V, H)$ je orientovaný graf $IC' = (V, H')$ s původními vrcholy a množinou hran takovou, že v grafu IC' existuje orientovaná hrana z uzlu e_1 do uzlu e_2 právě tehdy, pokud v původním orientovaném grafu IC existuje orientovaná cesta (posloupnost orientovaných hran) z uzlu e_1 do uzlu e_2.

Pro konstrukci tranzitivního uzávěru můžeme použít například Warshallův algoritmus [98], popřípadě jiné algoritmy pro konstrukci tranzitivního uzávěru [74]. Warshallův algoritmus použitý v této práci pracuje nad binární incidenční maticí M o rozměru $N \times N$, kde N je počet uzlů grafu. Na pozici $[e_1, e_2]$ v incidenční matici M je hodnota 1, pokud existuje hrana z uzlu e_1 do uzlu e_2, popř. 0 pokud tato hrana neexistuje. Příklad incidenční matice reprezentující hrany grafu na obrázku 5.6(b) je na obrázku 5.7(a). Výsledkem aplikace Warshallova algoritmu na tuto incidenční matici je matice na obrázku 5.7(b), která reprezentuje incidenční matici tranzitivního uzávěru – graf IC'.

Uzel grafu $e \in V$ reprezentuje izolovaný prvek obvodu, pokud v grafu IC' neexistuje hrana $(e, \overline{UUA}) \in H'$. Z incidenční matice můžeme přímo odečíst, že vrcholy $ADD2$, $SUB1$, $SUB2$ a $MX1$ reprezentují izolované prvky obvodu. Výsledkem analýzy struktury obvodu je množina $E_{isolated}$ reprezentující izolované prvky. Pokud jsou v obvodu nalezeny izolované prvky, tak se již neprovádí analýza spojů a analýza testovatelnosti obvodu. Je to dáno tím, že takové řešení nepředstavuje akceptovatelné řešení a je zbytečné pro takové obvody provádět například časově náročnou analýzu testovatelnosti. Pokud ve struktuře obvodu nejsou identifikovány izolované prvky, tak po analýze struktury následuje analýza spojů.

	UUA	ADD1	ADD2	SUB1	SUB2	MX1
UUA	0	1	1	1	0	0
ADD1	1	0	0	1	0	0
ADD2	0	0	0	0	1	0
SUB1	0	0	0	0	0	1
SUB2	0	0	0	0	0	1
MX1	0	0	1	0	0	0

a)

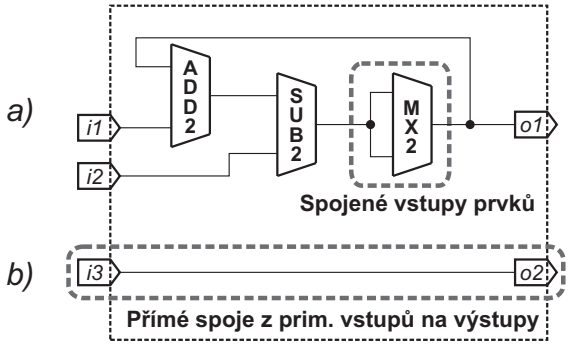
	UUA	ADD1	ADD2	SUB1	SUB2	MX1
UUA	**1**	1	1	1	1	1
ADD1	**1**	1	1	1	1	1
ADD2	**0**	0	1	0	1	1
SUB1	**0**	0	1	0	1	1
SUB2	**0**	0	1	0	1	1
MX1	**0**	0	1	0	1	1

b)

Obr. 5.7: Incidenční matice grafu reprezentujícího obvod UUA před aplikací (a) a po aplikaci tranzitivního uzávěru (b).

5.5.2 Analýza spojů obvodu

Cílem analýzy spojů je ohodnocení spojů obvodu z hlediska existence konstrukcí, které by se ve struktuře spojů neměly vyskytovat. V případě této práce se jedná o identifikaci vzájemně spojených vstupů jednoho prvku (viz obrázek 5.8(a)) a přímých spojů z primárních vstupů na primární výstupy obvodu (viz obrázek 5.8(b)).



Obr. 5.8: Příklad obvodu se spojenými datovými vstupy prvků (a) a přímými spoji z primárních vstupů na primární výstupy obvodu (b).

Identifikace spojených datových vstupů prvků

Identifikace spojených datových vstupů prvků je realizována přímo nad formálním modelem obvodu $UUA = (I, E, C)$. Pro každý prvek obvodu a také prvek reprezentovaný obvodem UUA je provedena analýza vzájemného propojení vstupních bran jednotlivých prvků. Výsledkem identifikace vstupních bran prvků je množina P_{shorts}, která obsahuje množiny spojených vstupů.

Algoritmus identifikace spojených vstupů pracuje tak, že pro jednotlivé prvky obvodu a jejich vstupní datové brány vytvoří relaci ekvivalence R takovou, že dvě vstupní datové brány x_1, x_2 jsou v této relaci, pokud jsou připojeny ke stejné výstupní bráně. Vytvořená relace R definuje pro každý prvek rozklad vstupních datových bran prvku podle výstupní brány, k níž

jsou vstupní datové brány připojeny. Výsledná množina spojených vstupních datových bran obvodu UUA je definována jako sjednocení všech množin spojených vstupních datových bran jednotlivých prvků s kardinalitou větší než jedna.

Algoritmus 4:

Procedura *Identifikace_spojených_vstupů_prvků(UUA;* **var:** P_{shorts}*)* {

1. [*Vytvoření relace ekvivalence $R(e)$ "brány prvku připojeny ke stejnému spoji"*]
   Pro ($\forall e \in (E \cup \{\overline{UUA}\}$) **vytvoř**
   $$R(e) = \{(i_1, i_2) \mid i_1 \in I_{DI}(e) \wedge i_2 \in I_{DI}(e) \wedge o \in P_{DO}(UUA) \wedge (i_1, o) \in C(UUA) \wedge (i_2, o) \in C(UUA)\}$$

2. [*Vytvoření rozkladu množiny vstupních bran prvku e podle relace ekvivalence $R(e)$*]
   Pro ($\forall e \in (E \cup \{\overline{UUA}\})$) **vytvoř**
   Rozklad $EC(e)$ $(= I_{DI}(e)/R(e))$ množiny $I_{DI}(e)$ podle relace ekvivalence $R(e)$

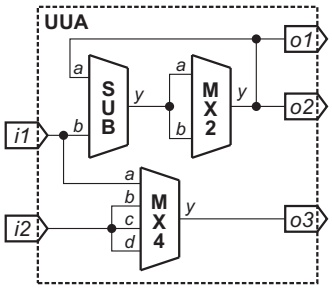
3. [*Vytvoření množiny P_{shorts} obsahující množiny spojených vstupních bran*]
   $$P_{shorts} = \bigcup_{\forall e \in (E \cup \{\overline{UUA}\})} \bigcup_{\forall x \in EC(e)} \begin{cases} \emptyset & \text{pokud } |x| = 1, \\ x & \text{jinak.} \end{cases}$$

}

Analyzujme nyní časovou složitost tohoto algoritmu. Pro zjednodušení předpokládejme, že obvod obsahuje pouze jeden typ prvku, který má *k* vstupů. V prvním kroku se pro každý prvek $e \in E \cup \{\overline{UUA}\}$ a každou vstupní bránu $i_1 \in I_{DI}(e)$ prochází množina C a hledají se další brány $i_2 \in I_{DI}(e)$ připojené ke stejnému spoji – časová složitost $O(|E| \cdot |C| \cdot k^2)$. V druhém kroku je vytvořen rozklad vstupních bran jednotlivých prvků $O(|E| \cdot k^2)$ a v závěrečném kroku je sestavena množina P_{shorts} – časová složitost $O(|E| \cdot k)$. Celkovou časovou složitost algoritmu můžeme vyjádřit vztahem $O(|E| \cdot |C| \cdot k^2)$.

Vlastní algoritmus pracuje tak, že v prvním kroku vytvoří pro každý prvek $e \in (E \cup \{\overline{UUA}\})$ relaci $R(e) \subseteq 2^{I_{DI}(e)}$ takovou, že dvě vstupní brány $i_1, i_2 \in I_{DI}(e)$ prvku e jsou v relaci $R(e)$, pokud jsou připojeny ke stejné výstupní bráně $o \in P_{DO}(UUA)$. Vzhledem k obecným předpokladům, že v obvodu nejsou použity 3-stavové sběrnice a všechny brány obvodu jsou zapojeny, můžeme konstatovat, že binární relace $R(e)$ je relací ekvivalence, protože splňuje podmínky reflexivity, tranzitivity a symetričnosti. Reflexivita relace vyplývá z toho, že pokud máme brány $i_1, i_2 \in I_{DI}(e) \wedge i_1 = i_2 = i$, pak musí existovat výstupní brána $o \in P_{DO}(UUA)$ taková, že $(i, o) \in C(UUA)$. Z definice relace $R(e)$ pak vyplývá, že $(i, i) \in R(e)$ $\Rightarrow$ relace je reflexivní. Podobně, pokud platí, že $(i_1, i_2) \in R(e)$ a $(i_2, i_3) \in R(e)$, potom musí existovat brány $o_1, o_2 \in P_{DO}(UUA)$ takové, že $(i_1, o_1), (i_2, o_1), (i_2, o_2), (i_3, o_2) \in C(UUA)$. Z předpokladu, že každá vstupní brána je připojena k právě jedné výstupní bráně plyne, že $o_1 = o_2$ a platí tedy také, že $(i_1, i_3) \in R(e) \Rightarrow$ relace je tranzitivní. Symetricita relace je pak zřejmá přímo z definice relace $R(e)$.

Relace ekvivalence $R(e)$ definuje rozklad $EC(e) = I_{DI}(e)/R(e)$ množiny vstupních bran $IC(e)$ podle výstupních bran, k nimž jsou vstupní brány připojeny. Posledním krokem identifikace spojených vstupních bran je sestavení množiny P_{shorts}, která obsahuje množiny vstupních bran, které jsou v rámci jednoho prvku vzájemně spojeny. Množinu P_{shorts} získáme tak, že pro $\forall e \in (E \cup \{\overline{UUA}\})$ provedeme sjednocení množin rozkladů $EC(e)$, ze kterých jsou odstraněny množiny s kardinalitou 1.



Obr. 5.9: Ukázka datové části obvodu se spojenými vstupy prvků.

Příklad 5.1: Mějme obvod na obrázku 5.9. Identifikujme všechny spojené vstupní brány jednotlivých prvků.

$$
\begin{aligned}
&UUA = (I, E, C)\\
&I = \ (\{(uua.i1, 8), (uua.i2, 8)\}, \{(uua.o1, 8), (uua.o2, 8), (uua.o3, 8)\}, \emptyset, \emptyset)\\
&E = \{SUB, MX2, MX4\}\\
&C = \{(mx4.a, uua.i1), (mx4.b, uua.i2), (mx4.c, uua.i2), (mx4.d, uua.i2),\\
&\qquad (mx2.a, sub.y), (mx2.b, sub.y), (sub.a, mx2.y), (sub.b, uua.i1),\\
&\qquad (uua.o1, mx2.y), (uua.o2, mx2.y), (uua.o3, mx4.y)\}
\end{aligned}
$$

$$
\begin{aligned}
&SUB = (\ (\{(sub.a, 8), (sub.b, 8)\}, \{(sub.y, 8)\}, \emptyset, \emptyset), \emptyset, \emptyset)\\
&MX2 = ((\{(mx2.a, 8), (mx2.b, 8)\}, \{(mx2.y, 8)\}, \emptyset, \emptyset), \emptyset, \emptyset)\\
&MX4 = ((\{(mx4.a, 8), (mx4.b, 8), (mx4.c, 8), (mx4.d, 8)\}, \{(mx4.y, 8)\}, \emptyset, \emptyset), \emptyset, \emptyset)
\end{aligned}
$$

Nyní sestavíme pro množinu I_{DI} vstupních bran každého prvku $e \in (E \cup \{\overline{UUA}\})$ relaci $R(e)$ ”být připojen ke stejné výstupní bráně”.

$$
\begin{aligned}
&R(SUB) = \{\ (sub.a, sub.a), (sub.b, sub.b)\},\\
&R(MX2) = \{(mx2.a, mx2.a), (mx2.a, mx2.b), (mx2.b, mx2.a), (mx2.b, mx2.b)\},\\
&R(MX4) = \{(mx4.a, mx4.a), (mx4.b, mx4.b), (mx4.b, mx4.c), (mx4.b, mx4.d),\\
&\qquad (mx4.c, mx4.b), (mx4.c, mx4.c), (mx4.c, mx4.d), (mx4.d, mx4.b),\\
&\qquad (mx4.d, mx4.c), (mx4.d, mx4.d)\},\\
&R(\overline{UUA}) = \{\ (uua.o1, uua.o1), (uua.o1, uua.o2), (uua.o2, uua.o1), (uua.o2, uua.o2),\\
&\qquad (uua.o3, uua.o3)\}.
\end{aligned}
$$

Dále vytvoříme rozklad množiny I_{DI} vstupních bran každého prvku $e \in (E \cup \{\overline{UUA}\})$ podle relace $R(e)$.

$$
\begin{aligned}
&EC(SUB) = \{\ \{sub.a\}, \{sub.b\}\},\\
&EC(MX2) = \{\{mx2.a, mx2.b\},\\
&EC(MX4) = \{\{mx4.a\}, \{mx4.b, mx4.c, mx4.d\}\},\\
&EC(\overline{UUA}) = \{\ \{uua.o1, uua.o2\}, \{uua.o3\}\},
\end{aligned}
$$

Sestavíme množinu P_{shorts} množin reprezentujících spojené vstupní brány jednotlivých prvků. Z této množiny je zřejmé, které brány jsou vzájemně spojeny. U analyzovaného obvodu jsou vzájemně spojeny vstupní brány $\{mx2.a, mx2.b\}$, $\{mx4.b, mx4.c, mx4.d\}$ a $\{uua.o1, uua.o2\}$.

$$P_{shorts} = \{\ \{mx2.a, mx2.b\}, \{mx4.b, mx4.c, mx4.d\}, \{uua.o1, uua.o2\}\}$$

Identifikace přímých spojů z primárních vstupů na primární výstupy

Druhým krokem analýzy spojů obvodu je identifikace přímých spojů z primárních vstupů na primární výstupy obvodu. Tento problém je jednoduše řešitelný průchodem konečné množiny spojů obvodu, při kterém hledáme množinu spojů $C_{direct} \subseteq C$ takovou, že

$$C_{direct} = \{(u, v) | (u, v) \in C \wedge u\ \in I_{DI}(\overline{UUA}) \wedge v\ \in I_{DO}(\overline{UUA})\} \tag{5.1}$$

Celkové ohodnocení spojů obvodu

Celkové ohodnocení spojů obvodu je dáno následujícím vztahem

$$connects = 1 - \frac{\left(\sum\limits_{\forall p \in P_{shorts}} |p|\right) + |C_{direct}|}{|P_{DI}(UUA)| + |I_{DO}(UUA)|}, \tag{5.2}$$

kde $\sum\limits_{\forall p \in P_{shorts}} |p|$ vyjadřuje počet vzájemně spojených vstupních bran jednotlivých prvků, $|C_{direct}|$ je počet přímých spojů z primárních vstupů obvodu na primární výstupy, $|P_{DI}(UUA)|$ je počet bran obvodu se vstupní orientací a $|I_{DO}(UUA)|$ je počet primárních výstupů obvodu. Výsledkem ohodnocení obvodu je reálné číslo v intervalu $\langle 0, 1\rangle$, kde vyšší ohodnocení znamená kvalitnější obvod.

5.5.3 Analýza testovatelnosti obvodu

Po analýze struktury spojů následuje ohodnocení testovatelnosti obvodu. Testovatelnost obvodu je v této práci vyjádřena ve formě parametrů řiditelnosti a pozorovatelnosti. Parametr řiditelnosti vyjadřuje průměrnou obtížnost nastavení hodnoty v daném místě obvodu prostřednictvím primárních vstupů obvodu. Parametr pozorovatelnosti vyjadřuje průměrnou obtížnost pozorování hodnoty v daném místě obvodu prostřednictvím primárních výstupů obvodu. Pro analýzu testovatelnosti obvodu je použita metoda analýzy testovatelnosti, která je podrobně popsána v následující kapitole. Navržená metoda analýzy testovatelnosti umožňuje ohodnocení průměrné řiditelnosti ($avg\_cont.$) a pozorovatelnosti ($avg\_obs.$) bran kandidátního obvodu. Hodnoty získané na základě analýzy testovatelnosti obvodu jsou následně porovnány s hodnotami řiditelnosti ($req\_cont.$) a pozorovatelnosti ($req\_obs.$), které byly požadovány uživatelem. Hodnota parametru $testability$ je reálné číslo z intervalu $\langle 0; 1\rangle$, které vyjadřuje do jaké míry jsou splněny uživatelem specifikované požadavky.

$$testability = 1 - \frac{(req\_cont. - avg\_cont)^2 + (req\_obs. - avg\_obs.)^2}{2} \tag{5.3}$$

5.5.4 Celkové ohodnocení fitness

Výsledná hodnota fitness je dána kombinací výsledků analýzy struktury obvodu (identifikace izolovaných prvků), analýzy spojů obvodu (identifikace přímých spojů a spojených vstupů prvků) a analýzy testovatelnosti. Váhy jednotlivých parametrů byly zvoleny experimentálně a vyjadřují jakou vahou se má daný parametr projevit na celkovém ohodnocení kvality obvodu. V návrhové metodě představené v této kapitole je kladen zejména důraz na splnění uživatelem specifikovaných diagnostických vlastností vytvářených obvodů – ohodnocení testovatelnosti se na celkovém ohodnocení kvality obvodu podílí z 50%.

$$fitness = 0,25 \cdot \frac{|E_{isolated}|}{|E|} + 0,25 \cdot connects + 0,5 \cdot testability \tag{5.4}$$

Výsledkem ohodnocení kandidátního řešení je reálné číslo v intervalu $\langle 0; 1 \rangle$, které vyjadřuje kvalitu analyzovaného obvodu. Vyšší ohodnocení reprezentuje „kvalitnější“ obvod z hlediska požadavků uživatele.

5.6 Shrnutí

V této kapitole byla představena metoda návrhu syntetických testovacích obvodů vhodných pro ověřování diagnostických metod a nástrojů. Navržená metoda využívá nový přístup k vytváření syntetických testovacích obvodů, který je založen na využití evolučního návrhu. Uživatel má možnost specifikovat požadavky na vlastnosti vytvářených obvodů ve formě počtu bran rozhraní, počtu a typu prvků, z nichž se má obvod skládat a požadovaných diagnostických vlastností obvodu. Návrh testovacích obvodů je realizován jako optimalizační proces, kdy se za pomocí algoritmu evolučního programování hledá nejvhodnější propojení uživatelem specifikovaných prvků tak, aby byly splněny požadované diagnostické vlastnosti.

Pro jednoznačnost zápisu navržené metody je použit formální model obvodu zavedený v předcházející kapitole. S pomocí zavedeného modelu jsou definovány operátory mutace použité pro změnu struktury spojů obvodu a je definována hodnotící funkce umožňující ohodnotit kvalitu vytvářených obvodů z hlediska struktury obvodu a splnění diagnostických vlastností obvodu (podrobněji se bude otázkou ohodnocení diagnostických vlastností kandidátního obvodu zabývat následující kapitola). Výhodou použití formálního přístupu pro popis návrhové metody je zejména jednoznačnost zápisu a možnost transformovat problémy návrhu na známé a řešené problémy diskrétní matematiky a teoretické informatiky.

Kapitola 6

Analýza testovatelnosti

Analýza testovatelnosti kandidátního obvodu představuje časově nejnáročnější část procesu evolučního návrhu syntetických testovacích obvodů. Volba vhodné metody analýzy testovatelnosti významným způsobem ovlivňuje možnosti výsledné návrhové metody. Metoda použitá pro ohodnocení testovatelnosti kandidátních obvodů v této práci vychází z metody *ADFT* představené v disertační práci Josefa Strnadela [86]. Strnadelem navržená metoda je v této práci optimalizována a dále rozšířena tak, aby byla vhodná pro evoluční návrh testovacích obvodů.

6.1 Metoda *ADFT*

Metoda *ADFT* je založena na ohodnocení řiditelnosti a pozorovatelnosti vnitřních bran obvodu. Ohodnocení je realizováno nad formálním modelem toku diagnostických dat obvodem, který je modelován pomocí dvojice orientovaných grafů G_S a G_I, kde G_S modeluje tok testovacích vektorů strukturou obvodu a G_I modeluje tok odezev na testovací vektory. Uzly grafů G_S a G_I představují brány obvodu a orientované hrany grafu představují metalické spoje, popř. existenci transparentní cesty vnitřní strukturou prvku. Základem metody je knihovna prvků, která obsahuje informace o transparentních vlastnostech jednotlivých prvků, které jsou modelovány pomocí tzv. *i-režimů* (viz podkapitola 4.4). Právě na kvalitě knihovny prvků a použitém způsobu modelování jejich transparentních režimů závisí, jak přesných výsledků bude navržená metoda analýzy testovatelnosti schopna dosáhnout.

Analýza řiditelnosti probíhá nad grafem G_S, který modeluje tok testovacích vektorů. Analýza pozorovatelnosti probíhá nad grafem G_I a při ohodnocení pozorovatelnosti se částečně využívá výsledků analýzy řiditelnosti. Při analýze řiditelnosti (pozorovatelnosti) uzlu x grafu G_S (G_I) je každému uzlu přiřazena informace (1) o sekvenční hloubce brány, která je daným uzlem grafu reprezentována, (2) počtu bran a (3) součinu řiditelností bran, které je potřeba řídit pro nastavení/pozorování požadované brány obvodu. Na základě tohoto ohodnocení je pak vypočtena řiditelnost (pozorovatelnost) daného uzlu. Ohodnocení řiditelnosti (pozorovatelnosti) je realizováno následujícím vztahem [86]:

$$C/O(x) = (1 - \frac{seq(x)}{seq(UUA)+1}) \times (1 - \frac{sti(x)}{sti(UUA)+1}) \times (\prod_{y \in conds(x)} C(y)) \quad , \tag{6.1}$$

kde $seq(x)$ je odhad sekvenční hloubky brány x,

$seq(UUA)$ je maximální sekvenční hloubka obvodu UUA, který je předmětem analýzy testovatelnosti,
$sti(x)$ je odhad maximálního počtu bran, které je potřeba řídit pro zajištění řiditelnosti/pozorovatelnosti libovolné brány obvodu,
$sti(UUA)$ počet vstupních bran v obvodu UUA,
$conds(x)$ množina bran, které je potřeba řídit pro zajištění řiditelnosti/pozorovatelnosti brány x,
$C/O(x)$ je řiditelnost/pozorovatelnost brány x.

Výsledkem analýzy testovatelnosti provedené pomocí metody *ADFT* je informace o průměrné řiditelnosti a pozorovatelnosti bran obvodu, která je vypočtena jako průměrná hodnota řiditelnosti a pozorovatelnosti bran obvodu.

6.2 Navržená metoda analýzy testovatelnosti

Přístup použitý pro analýzu testovatelnosti v této práci vychází z výše uvedené metody vytvořené Josefem Strnadelem a dále jím navrženou metodu rozšiřuje a optimalizuje tak, aby byla vhodná pro evoluční návrh testovacích obvodů. Pokud bychom měli shrnout hlavní provedené změny, pak jsou to zejména tyto:

- **možnost vkládat nové prvky do knihovny prvků** – metoda navržená v této práci není omezena pouze na obvody skládající se ze základních prvků úrovně RT (sčítačky, násobičky, . . .), ale umožňuje uživateli knihovnu prvků dále rozšiřovat např. o složitější moduly (ALU, FIFO, . . .).

- **analýza testovatelnosti hierarchicky popsaného obvodu** – navržená metoda umožňuje realizovat analýzu testovatelnosti hierarchicky popsaného obvodu na úrovni RT.

- **zohlednění obtížnosti testování jednotlivých prvků** – ohodnocení průměrné řiditelnosti a pozorovatelnosti bran obvodu je realizováno jako vážený průměr, který zohledňuje bitovou šířku jednotlivých bran a také počet testovacích vektorů a odezev na tyto vektory, které jsou potřeba pro otestování jednotlivých prvků. Pokud tedy některý prvek například potřebuje pro své otestování více testovacích vektorů, projeví se také více jeho řiditelnost na celkové průměrné řiditelnosti bran obvodu.

- **odstranění tzv. virtuálních portů** – Strnadel ve své práci používá abstrakci nad rozhraním jednotlivých prvků v podobě tzv. virtuálních portů, které umožňují modelovat transparentní cesty vnitřní strukturou prvku nejen na úrovni bran, ale také na úrovni jednotlivých bitů těchto bran. Použití této abstrakce však vede na vyšší časovou složitost výsledného algoritmu. Navržený algoritmus pracuje s transparentními cestami pouze na úrovni bran.

- **modelování transparentních cest** – modelování transparentních cest není realizováno pomocí dvojice orientovaných grafů jako u metody *ADFT*, ale probíhá přímo nad strukturou obvodu popsanou pomocí formálního modelu zavedeného v této práci.

Z původní metody ADFT je v této práci převzat způsob výpočtu řiditelnosti a pozorovatelnosti bran obvodu (viz vztah 6.1). Nový je ale způsob, jakým je ohodnocení řiditel-

nosti a pozorovatelnosti bran obvodu realizováno. Navržená metoda realizuje analýzu testovatelnosti přímo nad strukturou obvodu popsanou formálním modelem zavedeným v kapitole 4. Pro účely analýzy testovatelnosti byl model struktury obvodu rozšířen o zobrazení $TA \subseteq P_{ALL} \times (\mathbb{R}_{\langle 0,1 \rangle} \times \mathbb{R}_{\langle 0,1 \rangle} \times \mathbb{N} \times \mathbb{N})$, které každé bráně obvodu $g \in P_{ALL}(UUA)$ přiřazuje čtveřici $(g_{co}, g_o, g_{seq}, g_{sti})$, kde jednotlivé členy čtveřice mají následující význam: g_{co} vyjadřuje řiditelnost brány, g_o pozorovatelnost brány, g_{seq} sekvenční hloubku brány a g_{sti} počet bran, které je potřeba řídit pro nastavení/pozorování hodnoty brány.

Ohodnocení testovatelnosti obvodu probíhá ve dvou krocích. V prvním kroku je realizována analýza řiditelnosti bran obvodu a v druhém kroku pak následuje analýza pozorovatelnosti, která využívá výsledků přecházejícího kroku. Výsledná průměrná řiditelnost a pozorovatelnost je dána jako vážený průměr řiditelnosti a pozorovatelnosti bran obvodu, kde váha jednotlivých bran je dána bitovou šířkou a počtem testovacích vektorů, které jsou potřeba pro otestování prvku, k němuž daná brána náleží.

Jednotlivé fáze ohodnocení řiditelnosti a pozorovatelnosti můžeme popsat následujícím způsobem. Algoritmus ohodnocení řiditelnosti bran vychází z primárních vstupů obvodu, které mají maximální hodnotu řiditelnosti ($C = 1,00$). Dále pokračuje směrem k primárním výstupům obvodu, přičemž hodnota řiditelnosti předcházející brány je vždy propagována nejprve prostřednictvím metalických spojů obvodu a pokud to není možné, tak prostřednictvím transparentních cest prvků. Propagace hodnoty řiditelnosti probíhá tak, že při propagaci prostřednictvím metalických spojů zůstává hodnota řiditelnosti zachována a při propagaci prostřednictvím transparentních cest je řiditelnost následující brány vždy snížena s ohledem na to, jak problematické je nastavení daného prvku do transparentního režimu. Algoritmus končí ve chvíli, kdy jsou ohodnoceny všechny brány obvodu nebo neexistuje žádný další metalický spoj (popř. transparentní cesta), kterou by bylo možné použít pro propagaci řiditelnosti strukturou obvodu.

Algoritmus ohodnocení pozorovatelnosti navazuje na ohodnocení řiditelnosti a pro svou činnost využívá informace získané při analýze řiditelnosti. Ohodnocení pozorovatelnost bran obvodu začíná z primárních výstupů obvodu, které mají maximální hodnotu pozorovatelnosti ($O = 1,00$). Pozorovatelnost ohodnocených bran obvodu je propagována prostřednictvím metalických spojů a transparentních cest prvky obvodu směrem k primárním vstupům obvodu. Vlastní propagace hodnot pozorovatelnosti probíhá podobně jako propagace řiditelnosti. Při propagaci prostřednictvím metalických spojů je hodnota pozorovatelnosti výstupní brány dána nejvyšší hodnotou pozorovatelnosti brány, která je k dané výstupní bráně připojena. Při propagaci prostřednictvím transparentních cest je pozorovatelnost snížena s ohledem na vlastnosti použité transparentní cesty a řiditelnosti bran, které jsou potřeba pro nastavení této cesty.

Výsledná testovatelnost obvodu je dána dvojicí reprezentující průměrnou řiditelnost a pozorovatelnost bran obvodu. Výpočet je realizován jako vážený průměr řiditelnosti a pozorovatelnosti jednotlivých bran obvodu, kde váhy jednotlivých bran jsou dány bitovou šířkou bran a počtem testovacích vektorů a odezev na tyto vektory, které jsou potřeba pro otestování prvku patřícího k této bráně.

Z principu algoritmu pro ohodnocení testovatelnosti popsaného výše je zřejmé, že důležitou informací při ohodnocení řiditelnosti a pozorovatelnosti obvodu je informace o transparentních vlastnostech jednotlivých prvků. Pro tyto účely existuje knihovna prvků, která tyto informace obsahuje. Problémem existujících metod založených na modelování transparentních režimů prvků je obvykle problematické rozšíření knihovny prvků o další prvky. Jedním z cílů, který byl sledován při návrhu metody analýzy testovatelnosti představené v této kapitole, bylo také umožnění uživateli vkládat nové prvky do knihovny prvků. Než přistoupíme k vlastnímu popisu

algoritmů pro ohodnocení řiditelnosti a pozorovatelnosti obvodu, bude nejprve představena knihovna prvků a také algoritmy, které umožňují rozšíření navržené knihovny prvků.

6.3 Knihovna prvků

Navržená metoda analýzy testovatelnosti předpokládá existenci knihovny prvků, která dokáže modelovat vybrané vlastnosti prvků z hlediska analýzy testovatelnosti a pro účely této práce také určité vlastnosti z hlediska vytváření testovacích obvodů. Knihovna prvků obsahuje pro každý prvek informace o jeho rozhraní a struktuře, dále informaci o transparentních vlastnostech prvku a informaci o počtu testovacích vektorů a odezev na tyto vektory, které jsou potřeba pro testování prvku. Pro popis jednotlivých vlastností je opět využit formální model, který byl zaveden v kapitole 4. Rozhraní a struktura prvku je modelována pomocí trojice $UUA = (I, E, C)$ (viz definice 4.1, strana 49), transparentní vlastnosti prvku jsou modelovány pomocí tzv. *i-režimů* prvku $\mathcal{I}_{modes}$ (viz definice 4.11, strana 56) a obtížnost testování jednotlivých prvků je modelována jako dvojice $Test(UUA) = (n, m)$ (viz definice 6.1).

Definice 6.1: Nechť n je počet testovacích vektorů, které je potřeba při testování prvku UUA přivést na vstupy prvku UUA a m je počet odezev na testovací vektory, které je potřeba analyzovat prostřednictvím výstupů prvku UUA. Pak obtížnost testování prvku UUA je vyjádřena pomocí dvojice $Test(UUA) = (n, m) \in (\mathbb{N} \times \mathbb{N})$.
□

Samotná knihovna prvků je pak definovaná následovně.

Definice 6.2: Nechť $UUA = (I, E, C)$ modeluje rozhraní a vnitřní strukturu prvku UUA, $\mathcal{I}_{modes}(UUA)$ je množina všech *i-režimů* prvku UUA a $Test(UUA)$ vyjadřuje obtížnost testování prvku UUA. Pak knihovna Lib je konečná množina prvků, které jsou reprezentovány trojicí $(UUA, \mathcal{I}_{modes}(UUA), Test(UUA))$.
□

Součástí navržené metody je knihovna prvků, která obsahuje modely základních prvků úrovně RT. Uživatel má možnost knihovnu prvků dále rozšiřovat o další – strukturou popsané prvky v jazyce VHDL.

6.3.1 Přidání nového prvku

Předpokládejme, že máme strukturou popsaný prvek v jazyce VHDL, který chceme přidat do knihovny prvků. Vlastnímu přidání nového prvku předchází proces tzv. charakterizace prvku, při kterém jsou analyzovány vlastnosti prvku důležité z hlediska analýzy testovatelnosti. Postup charakterizace prvku se skládá z několika dílčích kroků:

1. **analýza rozhraní a struktury prvku** – analýza zdrojového VHDL kódu, prostřednictvím které získáme informace o rozhraní prvku a jeho vnitřní struktuře. Na základě výsledků analýzy VHDL kódu můžeme sestavit model rozhraní a struktury obvodu pomocí formálního modelu zavedeného v kapitole 4.

2. **analýza *i-režimů* prvku** – probíhá nad formálním modelem prvku vytvořeným v předcházejícím kroku. Cílem analýzy *i-režimů* je nalezení všech transparentních režimů vkládaného prvku, které by bylo možné použít pro přenos diagnostických dat strukturou prvku.

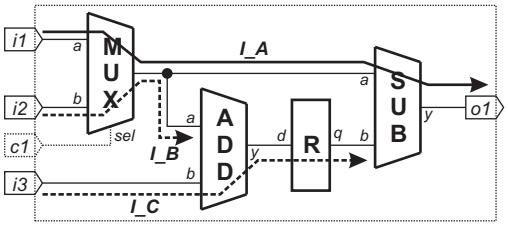
Jedná se o nalezení všech *i-cest* z primárních vstupů prvku na jeho primární výstupy, kdy je potřeba zohlednit možnost existence konfliktů mezi jednotlivými *i-cestami*. Analýze *i-režimů* prvku se podrobněji věnuje následující podkapitola.

3. **vytvoření testu prvku** – pro vkládaný prvek je pomocí generátoru testu vytvořen test a do knihovny prvků je vložena informace o počtu testovacích vektorů, které je potřeba přivést na vstupy testovaného prvku a počtu odezev, které je potřeba sledovat na výstupech testovaného prvku pro jeho otestování.

6.3.2 Analýza transparentních vlastností prvku knihovny

Při vkládání nového prvku do knihovny prvků je potřeba nejprve analyzovat *i-režimy* vkládaného prvku. V této práci budeme předpokládat, že se jedná o strukturou popsaný prvek na úrovni RT skládající se z prvků, které jsou k dispozici v knihovně prvků. Analýza *i-režimů* takového prvku obvykle představuje poměrně obtížný problém, protože je potřeba při analýze *i-režimů* prvku řešit problematiku plánování *i-cest* vnitřní strukturou prvku.

Plánování *i-cest* (v angl. *i-path* scheduling) je poměrně náročný proces. Během procesu plánování *i-cesty* je potřeba pro každý prvek použitý pro transport diagnostických dat zajistit jeho nastavení do *i-režimu*. Problém v tomto případě představují zejména datově závislé *i-režimy* (viz podkapitola 4.4), u nichž je *i-režim* podmíněn nastavením dalších datových vstupů daného prvku na požadovanou hodnotu. Je tedy obvykle potřeba plánovat další *i-cesty*, které umožní nastavení požadovaných vstupů. Při plánování těchto *i-cest* je pak potřeba řešit konflikty, které mezi těmito *i-cestami* vznikají. Příklad možného konfliktu mezi *i-cestami* je na obrázku 6.1. Pro nastavení *i-cesty I_A* z primárního vstupu *i1* na primární výstup *o1* je potřeba nastavit datově závislý *i-režim* prvku SUB (*i-cesta I_C*). Pro nastavení *i-cesty I_C* je pak potřeba nastavit *i-režim* prvku ADD (*i-cesta I_B*). Na obrázku 6.1 je vidět konflikt mezi *i-cestami I_A* a *I_B*, který vzniká při plánování *i-cest* prvkem MUX, kdy je tento prvek použit jak *i-cestou I_A* tak *i-cestou I_B*.



Obr. 6.1: Příklad konfliktu mezi *i-cestami*

Existuje několik přístupů, jak zabránit konfliktům mezi *i-cestami*. Nejjednodušší přístup zakazuje použití jednoho prvku více *i-cestami*. Pokud by tento přístup byl použit pro plánování *i-cest* v obvodu na obrázku 6.1, tak by výsledkem bylo, že v uvedeném obvodu neexistuje žádná *i-cesta* z primárních vstupů obvodu na jeho primární výstupy.

Další možný přístup představuje práce Růžičky [81], která byla později rozvedena Škarvadou v [56]. Plánování *i-cest* je u obou prací realizováno ve dvou fázích. V první fázi se plánují *i-cesty* v obvodu bez ohledu na jejich možné konflikty a v druhé fázi je pak použita Petriho síť, která modeluje průchod diagnostických dat obvodem a umožňuje detekovat konflikty mezi *i-cestami*. Problémem tohoto přístupu však je, že umožňuje pouze detekovat problémy mezi

i-cestami, ale neumožňuje automaticky navrhnout řešení – je nutná interakce s uživatelem, který musí navrhnout řešení konfliktu. Vzhledem k tomu, že cílem této práce je automatizovat proces vkládání nového prvku, bylo potřeba vytvořit metodu plánování *i-cest*, která by umožňovala automaticky řešit konflikty mezi jednotlivými *i-cestami* v obvodu.

6.3.3 Algoritmus plánování *i-cest* v obvodu

Byla navržena metoda automatického plánování bezkonfliktních *i-cest* v obvodu [75], která je založena na novém přístupu, kdy je plánování *i-cest* a využití jednotlivých prvků realizováno nejen v rámci struktury obvodu, ale také v rámci jednotlivých časových rámců (taktů hodin). Metoda byla navržena s cílem identifikovat všechny existující bezkonfliktní *i-cesty* v obvodu. Pro plánování *i-cest* je využit formální model zavedený v této práci. Uvedený model byl pro účely analýzy testovatelnosti rozšířen o modelování využití jednotlivých prvků v různých časových úsecích odpovídajících hodinovým taktům. Díky tomu může být jeden prvek využit opakovaně v různých taktech hodin. S využitím tohoto modelu máme možnost plánovat *i-cesty* nejen v rámci struktury obvodu, jak k plánování přistupují existující metody [56, 81]), ale také v rámci jednotlivých časových úseků. Získáváme tak další stupeň volnosti, který nám umožňuje vyhnout se potenciálním konfliktům mezi jednotlivými *i-cestami*.

Navržený algoritmus umožňuje nalézt všechny bezkonfliktní *i-cesty* v obvodu UUA vedoucí z primárních vstupů na primární výstupy obvodu UUA. Pokud obvod UUA reprezentuje strukturou popsaný prvek, pak nalezené *i-cesty* reprezentují všechny *i-režimy* prvku UUA. Vstupem algoritmu je informace o struktuře obvodu a knihovna prvků, která obsahuje informace o *i-režimech* použitých prvků. Vlastní algoritmus se skládá ze dvou částí. Hlavní část algoritmu (viz algoritmus 5) zajišťuje inicializaci a volání výkonné rekurzivní části (viz algoritmus 6), která využívá tzv. *backtracking* pro nalezení všech bezkonfliktních *i-cest* v obvodu. Algoritmus začíná s plánováním *i-cest* z primárních výstupů obvodu a postupně sestavuje všechny možné *i-cesty* směrem k primárním vstupům obvodu. Při sestavování *i-cest* postupuje tak, že se nejprve snaží využít metalické spoje obvodu. Pokud již není možné metalické spoje obvodu využít, jsou použity transparentní *i-režimy* prvků. Pokud má daný prvek k dispozici více *i-režimů* činnosti, jsou postupně ověřeny všechny možné *i-režimy* prvku. Ve chvíli, kdy není možné využít žádný další *i-režim* a *i-cestu* tak není možné dále propagovat, nastává tzv. návrat zpět (angl. backtrack), kdy se algoritmus vrací k nejbližšímu místu, kde je možné pokračovat jinou cestou. Tím je zaručeno, že jsou postupně analyzovány všechny možné *i-cesty*. Jednotlivé části algoritmu realizující identifikaci bezkonfliktních *i-cest* v obvodu budou popsány v následující části.

Hlavní část algoritmu

Hlavní část algoritmu zajišťuje inicializaci proměnných a realizuje volání vlastní výkonné rekurzivní procedury pro identifikaci *i-cest* v obvodu. Inicializace spočívá v zavedení proměnných a jejich inicializaci na počáteční hodnotu. V algoritmu 5 se jedná o proměnné I_{mode}, I_{modes} a $used\_elems$, kde I_{mode} reprezentuje aktuální rozpracovaný *i-režim* prvku UUA, I_{modes} je množina nalezených *i-režimů* a $used\_elems$ je množina dvojic modelující využití prvků v jednotlivých očíslovaných časových intervalech. V druhé části algoritmu je postupně na všechny primární výstupy volána výkonná rekurzivní část algoritmu zajišťující identifikaci *i-režimů* prvku UUA.

Algoritmus 5:

Procedura *Identifikace_i-režimů_prvku(*UUA, $Library$*;* **var:** I_{modes}*)* {

1. [*Zavedení a inicializace proměnných*]
   Zaveď $UUA = (I, E, C)$
   $I_{mode} \in ((I_{IN}(UUA) \cup \{\#\}) \times I_{OUT}(UUA) \times 2^{I_{IN}(UUA)} \times \mathbb{N})$
   $I_{modes} \subseteq (I_{IN}(UUA) \times I_{OUT}(UUA) \times 2^{I_{IN}(UUA)} \times \mathbb{N})$
   $used\_elems \subseteq (E \times \mathbb{N})$
   Přiřaď $I_{modes} = \emptyset$
2. [*Identifikace i-režimů prvku UUA – volání rekurzivní procedury*]
   Pro $\forall po \in I_{OUT}(UUA)$ **dělej**
   Přiřaď $I_{mode} = (\#, po, \emptyset, 0)$
   $used\_elems = \emptyset$
   Identifikace_i-režimu(UUA, $((po, 0), \#)$, $used\_elems$, I_{mode}, $\&I_{modes}$*);*

}

Výkonná část algoritmu

Algoritmus 6 reprezentuje výkonnou část algoritmu pro identifikaci *i-režimů* prvku UUA. Vstupními parametry algoritmu je seznam bran $todo$, pro něž je potřeba nalézt *i-cestu* z primárních vstupů UUA, množina $used\_elems$ modelující využití prvků obvodu UUA v jednotlivých taktech hodin a informace o aktuálním rozpracovaném *i-režimu* I_{mode}. Identifikace *i-režimů* je realizována v pěti krocích, které budou podrobněji popsány v následujících odstavcích.

Algoritmus 6:

Procedura *Identifikace_i-režimu(*UUA, $todo$, $used\_elems$, I_{mode}*;* **var:** I_{modes}*)* {

1. [*Nalezen i-režim prvku (seznam bran pro zpracování je prázdný)?*]
   Pokud ($todo = \#$) **potom**
   1.1 [*Ulož informace o i-režimu*]
   $I_{modes} = I_{modes} \cup \{I_{mode}\}$;
   1.2 [*Pokračuj v hledání dalších i-režimů*]
   return;

2. [*Proměnnou* $todo$ *můžeme reprezentovat jako n-tici*]
   $((gate, depth), todo\_tail) = todo$

3. [*Kontrola dosažení maximální povolené hloubky nebo primárních bran*]
   3.1. [*Dosažena maximální hloubka?*]
   Pokud ($depth > MAX\_DEPTH$) **potom**
   return;
   3.2. [*Dosaženy primární vstupní brány obvodu?*]
   Pokud ($gate \in I_{IN}(UUA)$) **potom**
   $(I_{src}, I_{dst}, I_{conds}, I_{depth}) = I_{mode}$
   # Aktualizujeme sekvenční hloubku i-režimu
   Pokud ($depth > I_{depth}$) **potom**

$I_{depth} = depth$
 # Přiřazena I_{src} brána?
Pokud ($I_{src} = \#$) **potom**
 # Brána $gate$ je počáteční brána i-cesty.
$I_{src} = gate$
jinak
 # Brána $gate$ nastavuje i-režim některého prvku na i-cestě.
$I_{conds} = I_{conds} \cup \{gate\}$
 # Aktualizujeme informace o aktualním i-režimu
$I_{mode} = (I_{src}, I_{dst}, I_{conds}, I_{depth})$;
 # Pokračujeme se zpracováním dalších položek seznamu todo
Identifikace_i-režimu(UUA, $todo\_tail$, $used\_elems$, I_{mode}, I_{modes});
 # Pokračujeme v hledání dalších i-režimů – backtrack
return;

4. [*Plánování i-cesty přes metalické spoje*]
Pokud ($gate \in P_{IN}(UUA)$) **potom**
Vyber $src \in P_{OUT}(UUA)$ takové, že $(gate, src) \in C(UUA)$
$todo\_new = ((src, depth), todo\_tail)$;
Identifikace_i-režimů(UUA, $todo\_new$, $used\_elems$, I_{mode}, I_{modes});
return;

5. [*Plánování i-cesty přes vnitřní strukturu prvku*]
Pokud ($gate \in P_{OUT}(UUA)$) **potom**
5.1 [*Vybereme prvek s výstupní bránou $gate$*]
Vyber $e \in E$ takové, že $gate \in I_{OUT}(e)$
5.2 [*Použijeme prvek e pokud není v aktuálním časovém úseku používán*]
Pokud ($(e, depth) \in used\_elems$) **potom return**;
5.3. [*Ověřme možnost nastavit jednotlivé i-režimy prvku e*]
Pro ($\forall (I_{src}, gate, I_{conds}, I_{depth}) \in \mathcal{I}_{modes}(e)$) **dělej**
 # Přidej brány jimiž je i-režim podmíněn do seznamu bran ke zpracování
$todo\_new = todo\_tail$
Pro ($\forall g \in I_{conds}$) **dělej**
$todo\_new = ((g, I_{depth} + depth), todo\_new)$;
 # Přidej počáteční bránu i-režimu
$todo\_new = ((I_{src}, I_{depth} + depth), todo\_new)$;
 # Přidej prvek e do seznamu používaných prvků
$used\_elems\_new = used\_elems \cup \{(e, depth)\}$
 # Pokračuj v identifikaci i-režimů
Identifikace_i-režimů($todo\_new$, $used\_elems\_new$, I_{mode}, I_{modes});
5.4. [*Všechny i-režimy prvku e byly ověřeny*]
return;
}

V první části algoritmu je nejprve ošetřena situace, kdy je seznam bran ke zpracování prázdný, tzn. byl nalezen *i-režim* prvku UUA. Informace o nalezeném *i-režimu* jsou uloženy do množiny *i-režimů* I_{modes} a algoritmus pokračuje hledáním dalších *i-režimů*. V druhém kroku al-

goritmu je pouze obsah proměnné $todo$ vyjádřen ve formě seznamu $((gate, depth), todo\_tail)$. Samotný algoritmus identifikace *i-režimů* prvku začíná až ve třetí části, kde je nejprve ověřeno, že nebyla dosažena maximální sekvenční hloubka obvodu – tato kontrola zajišťuje prevenci zacyklení algoritmu v případě existence zpětnovazební smyčky v obvodu. Pokud je právě zpracovávaná brána primární vstup obvodu, tak je aktualizována informace o aktuálním *i-režimu* a algoritmus pokračuje hledáním *i-cest* pro zbývající brány v množině $todo$, jinak se se ve čtvrtém kroku pokračuje plánováním *i-cesty* prostřednictvím metalických spojů (pokud má aktuálně zpravávaná brána vstupní orientaci) nebo v pátém kroku plánováním *i-cesty* prostřednictvím transparentních cest strukturou prvku (pokud má aktuálně zpravávaná brána výstupní orientaci). Při plánování *i-cesty* strukturou prvku je vždy nejprve ověřeno, zda již není prvek v daném časovém úseku používán. Pokud není, tak jsou postupně analyzovány všechny *i-režimy* prvku tak, aby byly identifikovány všechny *i-režimy* UUA. Informace o identifikovaných *i-režimech* prvku UUA jsou po skončení algoritmu k dispozici v proměnné I_{modes}.

6.4 Algoritmus ohodnocení testovatelnosti

Nyní se již dostáváme k samotnému popisu algoritmu analýzy testovatelnosti použitého pro ohodnocení testovatelnosti kandidátních obvodů. Princip algoritmu byl neformálně popsán v podkapitole 6.2. V této části práce bude algoritmus popsán s využitím formálního modelu zavedeného v kapitole 4.

Pro ohodnocení testovatelnosti obvodu je použit algoritmus 7. Vstupem algoritmu je informace o struktuře obvodu UUA a knihovna prvků Lib s informacemi o rozhraní, struktuře, transparentních vlastnostech a obtížnosti testování jednotlivých prvků. Výstupem algoritmu je informace o průměrné řiditelnosti a pozorovatelnosti bran obvodu, která vyjadřuje testovatelnost daného obvodu.

Algoritmus 7:

Procedura *Analýza_testovatelnosti(*UUA*,* Lib*;* **var:** *řiditelnost, pozorovatelnost)* {

1. [*Zavedení a inicializace proměnných*]
   Zaveď $UUA = (I, E, C)$
   $UUA_{seq} \in \mathbb{N}$, $UUA_{sti} \in \mathbb{N}$ – globální proměnné
   $TA \subseteq (P_{ALL} \times (\mathbb{R}_{\langle 0,1 \rangle} \times \mathbb{R}_{\langle 0,1 \rangle} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R}))$
   Přiřaď $TA = \{(g, 0.0, 0.0, 0, 0) |\ g \in P_{ALL}(UUA)\}$

2. [*Výpočet parametrů obvodu*]
   2.1 [*Výpočet maximální sekvenční hloubky obvodu*]
   $UUA_{seq} = 0$;
   Pro každé $(e \in E)$ **dělej**
   $I_{seqs}(e) = \{I_{seq} | (I_{src}, I_{dst}, I_{conds}, I_{seq}) \in \mathcal{I}_{modes}(e) \wedge$
   $(e, \mathcal{I}_{modes}(e), Test(e)) \in Lib\}$
   $UUA_{seq} = UUA_{seq} + max_{\forall x \in (I_{seqs}(e) \cup \{0\})} x$
   2.2 [*Výpočet maximálního počtu bran potřebného pro zajištění průchodu diag. dat*]
   $UUA_{sti} = |P_{IN}(UUA)| - 1$

3. [*Ohodnocení řiditelnosti*]
   Ohodnocení_řiditelnosti(UUA*,* Lib*, &*TA*)*;

4. [*Ohodnocení pozorovatelnosti*]
   Ohodnocení_pozorovatelnosti(UUA*,* Lib*, &*TA*)*;

5. [*Ohodnocení testovatelnosti*]
   Ohodnocení_testovatelnosti(UUA*,* Lib*,* TA*, &řiditelnost, &pozorovatelnost)*;

}

Ohodnocení testovatelnosti probíhá v pěti krocích. V prvním kroku algoritmu jsou zavedeny proměnné, je vytvořena a inicializována struktura TA, která každé bráně obvodu $g \in P_{ALL}(UUA)$ přiřazuje čtveřici $(g_{co}, g_{ob}, g_{seq}, g_{sti}) \in (\mathbb{R}_{\langle 0,1 \rangle} \times \mathbb{R}_{\langle 0,1 \rangle} \times \mathbb{N} \times \mathbb{N})$, pro uložení mezivýsledků a výsledků analýzy řiditelnosti a pozorovatelnosti bran obvodu.

V druhém kroku algoritmu je proveden odhad maximální sekvenční hloubky obvodu a maximálního počtu bran, které jsou potřeba pro zajištění průchodu diagnostických dat obvodem. Maximální sekvenční hloubka obvodu UUA_{seq} je odhadnuta jako suma maximálních sekvenčních hloubek *i-režimů* jednotlivých prvků. Maximální počet bran UUA_{sti}, které jsou potřeba pro zajištění průchodu diagnostických dat obvodem, je odhadnut jako počet vstupních datových a řídicích bran obvodu UUA snížený o bránu, která je použita jako začátek *i-cesty*. Obě tyto hodnoty představují nejhorší možný případ, který může teoreticky nastat.

V třetím kroku je provedena analýza řiditelnosti (viz podkapitola 6.5) a ve čtvrtém kroku pozorovatelnosti bran obvodu (viz podkapitola 6.6). Posledním krokem je výpočet průměrné řiditelnosti a pozorovatelnosti bran obvodu, která je v této práci použita jako měřítko testovatelnosti obvodu. Výpočet průměrné řiditelnosti a pozorovatelnosti bran obvodu realizuje algoritmus 8. Průměrná řiditelnost a pozorovatelnost je vypočtena jako vážený průměr řiditelnosti a pozorovatelnosti jednotlivých bran obvodu, kde váha jednotlivých bran je dána bitovou šířkou brány a obtížností testování prvku, ke kterému brána patří.

Algoritmus 8:

Procedura *Ohodnocení_testovatelnosti(*UUA*,* Lib*,* TA*;* **var:** *řiditelnost, pozorovatelnost)* {

1. [*Výpočet vah jednotlivých bran*]

$$co_{weight} = \{(g, co_w \cdot \Phi(g)) |\ g \in P_{ALL}(UUA) \wedge g \in I_{ALL}(e) \wedge (e, \mathcal{I}_{modes}, (co_w, o_w)) \in Lib\}$$

$$o_{weight} = \{(g, o_w \cdot \Phi(g)) |\ g \in P_{ALL}(UUA) \wedge g \in I_{ALL}(e) \wedge (e, \mathcal{I}_{modes}, (co_w, o_w)) \in Lib\}$$

2. [*Výpočet průměrné řiditelnosti a pozorovatelnosti UUA*]

$$\textit{řiditelnost} = \frac{\sum\limits_{g \in P_{ALL}(UUA)} co_{weight}(g) \cdot g_{co}}{\sum\limits_{g \in P_{ALL}(UUA)} co_{weight}(g)}, \text{ kde } (g, (g_{co}, g_o, g_{seq}, g_{sti})) \in TA$$

$$\textit{pozorovatelnost} = \frac{\sum\limits_{g \in P_{ALL}(UUA)} o_{weight}(g) \cdot g_o}{\sum\limits_{g \in P_{ALL}(UUA)} o_{weight}(g)}, \text{ kde } (g, (g_{co}, g_o, g_{seq}, g_{sti})) \in TA$$

}

6.5 Algoritmus ohodnocení řiditelnosti

Pro ohodnocení řiditelnosti bran obvodu je použit algoritmus 9. Vstupem algoritmu je informace o struktuře obvodu UUA, knihovna prvků Lib a struktura TA, která bude využita pro uložení mezivýsledků a výsledků analýzy řiditelnosti.

Algoritmus 9:

Procedura *Ohodnocení_řiditelnosti(*UUA, Lib*;* **var:** TA*)* {

1. [*Inicializace vstupních datových a řídicích primárních bran*]
   $G_{ToDo} = I_{IN}(UUA)$;
   $TA = (TA \setminus \{(g, ta) | (g, ta) \in TA \wedge g \in I_{IN}(UUA)\}) \cup$
   $\{(g, (1.0, 0.0, 0, 0)) | g \in I_{IN}(UUA)\}$

2. [*Smyčka pro šíření hodnot řiditelnosti*]
   Dokud $(G_{ToDo} \neq \emptyset)$ **dělej**
   $G'_{ToDo} = \emptyset$;
   Pro každé $g \in G_{ToDo}$ **proveď**
   2.1 [*Pokud je g výstupní brána* $\Rightarrow$ *přenos přes metalické spoje*]
   Pokud $g \in P_{OUT}(UUA)$ **potom**
   Řiditelnost_Přenos_přes_spoje(g, UUA, $\&TA$, $\&G'_{ToDo}$*)*;
   2.2 [*Pokud je g vstupní brána* $\Rightarrow$ *přenos přes strukturu prvku*]
   Pokud $g \in P_{IN}(UUA)$ **potom**
   Řiditelnost_Přenos_přes_prvek(g, $\mathcal{I}_{modes}$, $\&TA$, $\&G'_{ToDo}$*)*, kde
   $(e, \mathcal{I}_{modes}, Test) \in Lib \wedge g \in I_{ALL}(e)$
   $G_{ToDo} = G'_{ToDo}$;

}

Na začátku algoritmu je provedena inicializace, při níž se nastaví parametry vstupních bran obvodu – tyto brány mají maximální řiditelnost, sekvenční hloubka je nulová a nulový je také počet bran, které jsou potřeba pro řízení hodnot na těchto branách. Všechny primární vstupní brány jsou také přidány do množiny G_{ToDo}, která obsahuje brány, jejichž řiditelnost bude v dalším kroku propagována prostřednictvím metalických spojů a transparentních režimů prvků.

V druhé části algoritmu probíhá šíření hodnot řiditelnosti pro všechny brány $g \in G_{ToDo}$. Pokud je g brána s výstupní orientací, probíhá šíření řiditelnosti po spojích (viz algoritmus 10). Pokud brána g představuje bránu se vstupní orientací, probíhá šíření řiditelnosti prostřednictvím struktury prvku, k němuž brána g patří (viz algoritmus 11).

6.5.1 Šíření hodnot řiditelnosti prostřednictvím metalických spojů

Šíření hodnot řiditelnosti prostřednictvím metalických spojů je realizováno algoritmem 10. Nejprve je vytvořena množina bran $G_{connect}$, které jsou připojeny k výstupní bráně x. Ohodnocení řiditelnosti brány x je pak propagováno na všechny brány z množiny $G_{connect}$.

Algoritmus 10:

Procedura *Řiditelnost_Přenos_přes_spoje(*x, UUA*;* **var:** TA, G'_{ToDo}*)* {

1. [*Vytvoř množinu bran, které jsou metalicky spojeny s bránou* x]
   $G_{connect} = \{y \mid (y, x) \in C(UUA)\}$

2. [*Přenos řiditelnosti na brány z množiny* $G_{connect}$]
   $TA = (TA \setminus \{(g, TA(g)) \mid g \in G_{connect}\}) \cup \{(g, TA(x)) \mid g \in G_{connect}\}$

3. [*Přidej ohodnocené brány do* G'_{ToDo}]
   $G'_{ToDo} = G'_{ToDo} \cup G_{connect}$

}

6.5.2 Šíření hodnot řiditelnosti přes prvky

Obtížnější úlohu představuje šíření hodnot řiditelnosti prostřednictvím transparentních cest strukturou prvku (viz algoritmus 11). Existence transparentní cesty je totiž obvykle podmíněna nastavením dalších bran prvku. Vstupem algoritmu zajišťujícího šíření hodnot řiditelnosti prostřednictvím transparentních cest prvky je vstupní brána x a informace o transparentních režimech prvku, kterému brána x přísluší. Výstupem algoritmu je propagace hodnot řiditelnosti přes strukturu prvku, aktualizace struktury TA a doplnění změněných bran do množiny G'_{ToDo}.

Algoritmus 11:

Procedura *Řiditelnost_Přenos_přes_prvek(*x, $\mathcal{I}_{modes}$*;* **var:** TA, G'_{ToDo}*)* {

1. [*Zavedení a inicializace proměnných*]
   $y_{seq}, y_{sti} \in \mathbb{N}$; $y_{co}, y_o \in \mathbb{R}_{\langle 0,1 \rangle}$ a $y_{mul} \in \mathbb{R}$

2. [*Vytvoření množiny i-režimů, které jsou ovlivněny bránou* x]
   $I_{modes} = \{(I_{src}, I_{dst}, I_{conds}, I_{seq}) | (I_{src}, I_{dst}, I_{conds}, I_{seq}) \in \mathcal{I}_{modes} \wedge$
   $(x = I_{src} \vee x \in I_{conds})\}$

3. [*Přenos ohodnocení řiditelnosti přes prvky*]
   Pro každé $(src, dst, conds, seq) \in I_{modes}$ **proveď**
   2.1 [*Použij aktuální i-cestu pro průchod prvkem*]
   Mějme $(src_{co}, src_o, src_{seq}, src_{sti}) = TA(src)$ a
   $(dst_{co}, dst_o, dst_{seq}, dst_{sti}) = TA(dst)$
   $y_{seq} = src_{seq} + seq$;
   $y_{sti} = src_{sti} + |conds|$;
   $y_{mul} = src_{co} \cdot \prod_{\forall g \in conds} g_{co}$, kde $(g_{co}, g_o, g_{seq}, g_{sti}) = TA(g)$;
   $y_{co} = (1 - \frac{y_{seq}}{UUA_{seq}}) \cdot (1 - \frac{y_{sti}}{UUA_{sti}}) \cdot y_{mul}$
   2.2 [*Aktualizuj řiditelnost* dst *pokud došlo ke zlepšení řiditelnosti*]
   Pokud $(y_{co} > dst_{co})$ **potom**
   $TA = (TA \setminus \{(dst, TA(dst))\}) \cup \{(dst, (y_{co}, 0.00, y_{seq}, y_{sti}))\}$;
   $G'_{ToDo} = G'_{ToDo} \cup \{dst\}$;

}

Samotný algoritmus pro propagaci řiditelnosti strukturou prvku pracuje tak, že nejprve sestaví množinu I_{modes} *i-režimů*, které jsou ovlivněny zadanou branou x. Pro každý *i-režim* $(src, dst, conds, seq) \in I_{modes}$ z této množiny algoritmus provede šíření hodnot řiditelnosti. Sekvenční hloubka cílové brány dst je dána sekvenční hloubkou zdrojové brány src a sekvenční hloubkou použitého *i-režimu* ($dst_{seq} = src_{seq} + seq$). Počet bran, které je potřeba řídit pro nastavení brány dst, je dán počtem bran, které je potřeba řídit pro nastavení zdrojové brány src zvětšený o počet bran, jimiž je podmíněn použitý *i-režim* ($dst_{sti} = src_{sti} + |conds|$). Řiditelnost brány dst je pak vypočtena na základě vztahu 6.2, který byl převzat z [86].

$$dst_{co} = (1 - \frac{dst_{seq}}{(UUA_{seq} + 1)}) \cdot (1 - \frac{dst_{sti}}{(UUA_{sti} + 1)}) \cdot \prod_{g \in (cond \cup \{src_{co}\})} g_{co}, \tag{6.2}$$

kde UUA_{seq} je maximální sekvenční hloubka UUA, UUA_{sti} je maximální počet bran, jimiž může být podmíněna existence *i-cesty* v UUA a g_{co} představuje řiditelnost vybrané brány g. První člen výše uvedeného vztahu zohledňuje sekvenční hloubku cílové brány, druhý člen zohledňuje počet a třetí reprezentuje součin řiditelností těchto bran, které jsou potřeba pro řízení *i-režimu*.

6.6 Algoritmus ohodnocení pozorovatelnosti

Pro ohodnocení pozorovatelnosti bran obvodu je použit algoritmus 12, který následuje po ohodnocení řiditelnosti a využívá výsledky získané analýzou řiditelnosti. Vstupem algoritmu je informace o struktuře obvodu UUA, knihovna prvků Lib a struktura TA obsahující výsledky analýzy testovatelnosti, která bude zároveň využita pro uložení mezivýsledků a výsledků analýzy pozorovatelnosti.

Algoritmus 12:

Procedura *Ohodnocení_pozorovatelnosti(*UUA, $Library$*;* **var:** TA*)* {

1. [*Inicializace primárních výstupních bran*]
   $G_{ToDo} = I_{OUT}(UUA)$;
   $TA = (TA \setminus \{(g, TA(g)) \mid g \in I_{OUT}(UUA)\}) \cup$
   $\{(g, (g_{co}, 1.00, 0, 0)) \mid g \in I_{OUT}(UUA) \wedge (g_{co}, g_o, g_{seq}, g_{sti}) = TA(g)\}$

2. [*Smyčka pro šíření hodnot pozorovatelnosti*]
   Dokud $(G_{ToDo} \neq \emptyset)$ **dělej**
   $G'_{ToDo} = \emptyset$;
   Pro každé $g \in G_{ToDo}$ **proveď**
   2.1 [*Pokud je g vstupní brána* $\Rightarrow$ *přenos přes metalické spoje*]
   Pokud $g \in P_{IN}(UUA)$ **potom**
   Pozorovatelnost_Přenos_přes_spoje(g, UUA, $\&TA$, $\&G'_{ToDo}$*)*;
   2.2 [*Pokud je g výstupní brána* $\Rightarrow$ *přenos přes strukturu prvku*]
   Pokud $g \in P_{OUT}(UUA)$ **potom**
   Pozorovatelnost_Přenos_přes_prvek(g, $\mathcal{I}_{modes}$, $\&TA$, $\&G'_{ToDo}$*)*, kde
   $(e, \mathcal{I}_{modes}, Test) \in Lib \wedge g \in I_{ALL}(e)$

$G_{ToDo} = G'_{ToDo}$;

3. [*Korekce hodnot pozorovatelnosti*]
   Pro každé $y \in P_{OUT}(UUA)$ **proveď**
   3.1 [*Vytvoř množinu bran, které jsou spojeny s výstupní bránou* y]
   $G_{in} = \{x|(x,y) \in C(UUA)\}$;
   3.2 [*Proveď korekci pozorovatelnosti vstupních bran*]
   $TA = (TA \setminus \{(x,(x_{co},x_o,x_{seq},x_{sti})) \mid x \in G_{in}\}) \cup$
   $\{(x,(x_{co},y_o,y_{seq},y_{sti})) \mid x \in G_{in} \wedge (y_{co},y_o,y_{seq},y_{sti}) = TA(y)\}$;

}

Algoritmus analýzy pozorovatelnosti pracuje podobně jako algoritmus analýzy řiditelnosti. Na začátku algoritmu je provedena inicializace, při níž se nastaví parametry výstupních bran obvodu – tyto brány mají maximální řiditelnost, jejich sekvenční hloubka je nulová a nulový je také počet bran, které jsou potřeba pro propagaci hodnot z těchto bran na primární výstupy obvodu. Všechny primární výstupní brány jsou také přidány do množiny G_{ToDo}, která obsahuje brány, jejichž pozorovatelnost bude v dalším kroku propagována prostřednictvím metalických spojů nebo transparentních režimů prvků.

Hlavní část algoritmu je tvořena smyčkou, která zajišťuje šíření hodnot pozorovatelnosti prostřednictvím spojů (viz algoritmus 13) a strukturou prvků (viz algoritmus 14) obvodu. Vzhledem k tomu, že šíření pozorovatelnosti probíhá z primárních výstupů obvodu, může při ohodnocení pozorovatelnosti nastat situace, kdy dvě vstupní brány $g_1, g_2 \in P_{IN}(UUA)$, které jsou připojeny na stejnou výstupní bránu, mají rozdílnou pozorovatelnost. Tento problém řeší třetí krok algoritmu, který zajistí, že všechny brány, které jsou připojeny ke stejnému spoji, mají také stejnou pozorovatelnost.

6.6.1 Šíření hodnot pozorovatelnosti po spojích

Šíření hodnot pozorovatelnosti prostřednictvím metalických spojů je realizováno algoritmem 13. Vstupem algoritmu je informace o bráně x, z níž se má hodnota pozorovatelnosti šířit a informace o struktuře obvodu UUA. Šíření pozorovatelnosti po spojích je realizováno tak, že je nejprve nalezena brána y, ke které je brána x připojena. Pokud má brána x lepší pozorovatelnost než brána y, je pozorovatelnost této brány aktualizována na základě pozorovatelnosti brány x. Výsledkem algoritmu jsou aktualizované hodnoty pozorovatelnosti ve struktuře TA a aktualizovaná množina bran G'_{ToDo}.

Algoritmus 13:

Procedura *Pozorovatelnost_Přenos_přes_spoje(*x, UUA; **var:** TA, G'_{ToDo}, *)* {

1. [*Najdi výstupní bránu* y *ke které je brána* x *připojena*]
   Najdi $y \in P_{OUT}(UUA)$ takové, že $(x,y) \in C(UUA)$
2. [*Propaguj pozorovatelnost brány* x]
   Mějme $(x_{co},x_o,x_{seq},x_{sti}) = TA(x)$ a
   $(y_{co},y_o,y_{seq},y_{sti}) = TA(y)$;
   [*Pokud brána* x *nabízí lepší pozorovatelnosti* $\Rightarrow$ *propaguj*]
   Pokud $(y_o < x_o)$ **potom**
   $TA = (TA \setminus \{(y,y_{co},y_o,y_{seq},y_{sti})\} \cup \{(y,y_{co},x_o,x_{seq},x_{sti})\}$;

[*Přidej změněnou bránu do* G'_{ToDo}]
$G'_{ToDo} = G'_{ToDo} \cup \{y\}G$;
}

6.6.2 Šíření hodnot pozorovatelnosti přes prvky

Šíření hodnot pozorovatelnosti prostřednictvím struktury prvků probíhá podobně jako při šíření řiditelnosti. Vstupem algoritmu je brána y s výstupní orientací, informace o transparentních režimech prvku, kterému brána y přísluší a informace o řiditelnosti a pozorovatelnosti dalších bran obvodu. Výstupem algoritmu je propagace pozorovatelnosti přes strukturu prvku, aktualizace struktury TA a doplnění změněných bran do množiny G'_{ToDo}.

Algoritmus 14:

Procedura *Pozorovatelnost_Přenos_přes_prvek(*y, $\mathcal{I}_{modes}$*;* **var:** TA, G'_{ToDo}*)* {

1. [*Zavedení a inicializace proměnných*]
   $x_{seq}, x_{sti} \in \mathbb{N}$; $x_{co}, x_o \in \mathbb{R}_{\langle 0,1 \rangle}$ a $x_{mul} \in \mathbb{R}$

2. [*Vytvoř množinu i-režimů, ve kterých se uplatňuje brána* y]
   $I_{modes} = \{(src, y, conds, seq) | (src, y, conds, seq) \in \mathcal{I}_{modes}\}$

3. [*Přenos ohodnocení pozorovatelnosti přes prvek*]
   Pro každé $((src, dst, conds, seq) \in I_{modes})$ **proveď**
   2.1 [*Použij aktuální i-cestu pro průchod prvkem*]
   Mějme $(src_{co}, src_o, src_{seq}, src_{sti}) = TA(src)$ a
   $(dst_{co}, dst_o, dst_{seq}, dst_{sti}) = TA(dst)$
   $x_{seq} = dst_{seq} + seq$;
   $x_{sti} = dst_{sti} + |conds|$;
   $x_{mul} = dst_o \cdot \prod_{\forall g \in conds} g_{co}$, kde $(g_{co}, g_o, g_{seq}, g_{sti}) = TA(g)$;
   $x_o = (1 - \frac{x_{seq}}{UUA_{seq}+1}) \cdot (1 - \frac{x_{sti}}{UUA_{sti}+1}) \cdot x_{mul}$
   2.2 [*Aktualizuj* src *pokud došlo ke zlepšení řiditelnosti*]
   Pokud $(x_o > src_o)$ **potom**
   $TA = (TA \setminus \{(src, TA(src))\}) \cup \{(src, (src_{co}, x_o, x_{seq}, x_{sti}))\}$;
   $G'_{ToDo} = G'_{ToDo} \cup \{src\}$;
}

Algoritmus nejprve sestaví množinu I_{modes} *i-režimů*, ve kterých se uplatňuje brána y a v dalším kroku pak tyto *i-režimy* využívá pro šíření hodnot pozorovatelnosti. Pro každý *i-režim* $(src, y, conds, seq) \in I_{modes}$ algoritmus provede šíření pozorovatelnosti (viz vztah 6.3 [86]) a výsledek porovná s aktuální hodnotou pozorovatelnosti dané brány. Pokud vybraný *i-režim* nabízí lepší pozorovatelnost, tak aktualizujeme pozorovatelnost brány src.

$$src_o = (1 - \frac{src_{seq}}{(UUA_{seq} + 1)}) \cdot (1 - \frac{src_{sti}}{(UUA_{sti} + 1)}) \cdot dst_o \cdot \prod_{g \in conds} g_{co} \tag{6.3}$$

6.7 Časová složitost ohodnocení testovatelnosti

Časová složitost navržené metody analýzy testovatelnosti do značné míry závisí na vlastnostech použitých prvků a požadavcích specifikovaných uživatelem. Pro účely analýzy časové složitosti dále předpokládejme, že obvod se skládá z prvků jednoho typu s počtem vstupů R_{IN}, počtem výstupů R_{OUT} a počtem *i-režimů* prvku $R_{\mathcal{I}}$. Nyní můžeme přejít k vlastní analýze složitosti jednotlivých kroků analýzy testovatelnosti. V prvním kroku bude analyzována časová složitost algoritmu ohodnocení řiditelnosti, v druhém kroku časová složitost ohodnocení pozorovatelnosti a v závěrečném kroku bude analyzována celková časová složitost algoritmu ohodnocení testovatelnosti.

Ohodnocení řiditelnosti může být realizováno v maximálně v $|E|$ krocích, kde jednotlivé kroky znamenají přenos řiditelnosti přes spoje a strukturu prvku. Pro každý krok může být maximálně analyzováno $|P_{IN}|$ vstupních bran a $|P_{OUT}|$ výstupních bran. Přenos řiditelnosti přes spoje je realizován s konstantní časovou složitostí. Přenos řiditelnosti přes prvek je realizován s lineární časovou složitostí vzhledem k počtu *i-režimů* prvku $O(R_{\mathcal{I}})$. Ohodnocení celkové řiditelnosti obvodu je realizováno s časovou složitostí $O(|E| \cdot (|P_{IN}| \cdot R_{\mathcal{I}} + |P_{OUT}|))$.

Podobně ohodnocení pozorovatelnosti může být realizováno v maximálně $|E|$ krocích, kde jednotlivé kroky znamenají přenos pozorovatelnosti přes spoje a strukturu prvku. Pro každý krok může být maximálně analyzováno $|P_{IN}|$ vstupních bran a $|P_{OUT}|$ výstupních bran. Přenos pozorovatelnosti přes spoje je realizován s konstantní časovou složitostí. Přenos pozorovatelnosti přes prvek je realizován s lineární časovou složitostí vzhledem k počtu *i-režimů* prvku $O(R_{\mathcal{I}})$. Časová složitost korekce hodnot řiditelnosti je dána počtem spojů obvodu $|C| = |P_{IN}|$. Pozorovatelnost obvodu je ohodnocena s časovou složitostí $O(|E| \cdot (|P_{IN}| + |P_{OUT}| \cdot R_{\mathcal{I}}))$.

Algoritmus ohodnocení testovatelnosti pracuje v pěti krocích. V prvním kroku jsou inicializovány hodnoty testovatelnosti všech bran obvodu $O(|P_{ALL}|)$. V druhém kroku je analyzována maximální sekvenční hloubka obvodu $O(|E| \cdot R_{\mathcal{I}})$. Třetí krok reprezentuje analýzu řiditelnosti $O(|E| \cdot (|P_{IN}| \cdot R_{\mathcal{I}} + |P_{OUT}|))$ a čvrtý analýzu pozorovatelnosti $O(|E| \cdot (|P_{IN}| + |P_{OUT}| \cdot R_{\mathcal{I}}))$. V pátém kroku je vypočtena průměrná testovatelnost bran obvodu $O(|P_{ALL}|)$. Celková časová složitost ohodnocení testovatelnosti kandidátního obvodu je tak $O(|E| \cdot (|P_{IN}| + |P_{OUT}|) \cdot R_{\mathcal{I}})$. Pokud budeme předpokládat na začátku uvedené omezení z hlediska počtu vstupů a výstupů získáváme následující vztah $O(|E|^2 \cdot (R_{IN} + R_{OUT}) \cdot R_{\mathcal{I}})$. Můžeme tedy konstatovat, že algoritmus ohodnocení testovatelnosti pracuje s polynomiální časovou složitostí vzhledem k počtu prvků obvodu.

6.8 Shrnutí

Analýza testovatelnosti kandidátního řešení představuje časově nejnáročnější část procesu evolučního návrhu syntetických testovacích obvodů. Pro nalezení metody analýzy testovatelnosti vhodné pro evoluční návrh bylo nutné najít kompromis mezi časovou složitostí a přesností použité metody. Metoda navržená v této práci tento požadavek podle dosažených experimentálních výsledků (viz kapitola 7) splňuje. Navržená metoda vychází z metody *ADFT* představené v rámci disertační práce Josefa Strnadela [86]. Původní Strnadelem navržená metoda je v této práci upravena tak, aby byla vhodná pro evoluční návrh testovacích obvodů. Pokud bychom měli krátce popsat jednotlivé provedené změny, tak se jedná zejména o možnost vkládat nové

prvky do knihovny prvků a s ní související možnost analyzovat testovatelnost hierarchicky popsaného obvodu – navržená metoda tak není omezena pouze na obvody skládající se ze základních prvků úrovně RT. Při výpočtu průměrné řiditelnosti a pozorovatelnosti obvodu je zohledněna bitová šířka jednotlivých bran a také počet testovacích vektorů a odezev na tyto vektory, které jsou potřeba pro otestování jednotlivých prvků. Pokud tedy některý prvek například potřebuje pro své otestování více testovacích vektorů, projeví se také více jeho řiditelnost na celkové průměrné řiditelnosti bran obvodu. Výhodou navržené metody je také to, že není potřeba vytvářet pomocné grafy modelující tok diagnostických dat obvodem, protože je analýza testovatelnosti realizována přímo nad strukturou obvodu.

Kapitola 7

Experimentální výsledky

Důležitou částí této práce je také experimentální ověření navržené metody. V rámci experimentálního ověření byla sestavena množina experimentů s cílem, jednak ověřit návrhovou metodu z hlediska použitého optimalizačního algoritmu a také z hlediska plnění uživatelem specifikovaných kritérií. Pro ověření plnění uživatelem specifikovaných požadavků byly v této práci použity profesionální nástroje firmy Mentor Graphics. Realizovatelnost navržených obvodů je ověřena syntézou do technologie TSMC $0,35\mu$ nástrojem Leonardo Spectrum. Diagnostické vlastnosti vytvářených obvodů jsou ověřeny pomocí ATPG nástroje FlexTest. Výsledky vybraných experimentů jsou představeny v následujících podkapitolách.

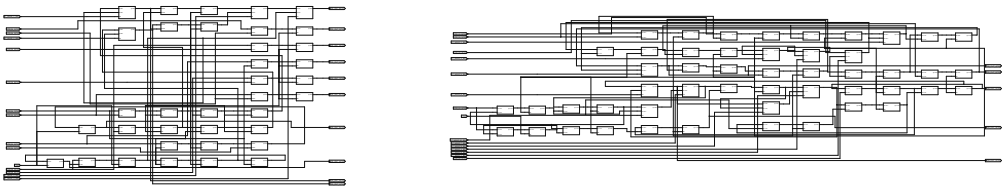
7.1 Základní ověření metody

První provedené experimenty byly zaměřeny na základní ověření funkčnosti navržené metody. Metoda byla nejprve použita pro návrh dvou obvodů s různým počtem prvků a různými požadavky na testovatelnost. Vytvořené obvody pak byly analyzovány z hlediska struktury (zda je tvoří uživatelem specifikované prvky a mají zadaný počet primárních vstupů a výstupů), možnosti realizace a testovatelnosti vytvářených obvodů.

Navržená metoda byla nejprve použita pro návrh snadno testovatelného obvodu s 80% řiditelností a pozorovatelností, který tvoří 20 prvků: 8x ADD(8bitů), 8xSUB(8bitů) a 4xMUX2(8bitů) a 10 primárních datových vstupů a výstupů. Evoluční algoritmus byl použit s následujícími parametry: velikost populace $N = 30$, počet generací $G = 200$, procento mutovaných spojů $M = 2\%$ a nahrazovaná část populace $R = 95\%$ (popis jednotlivých parametrů viz 5.3). Na obrázku 7.1(a) je ukázka testovacího obvodu na úrovni RT vytvořeného navrženou metodou na základě výše uvedených vstupních parametrů. Navržený obvod se skládá z 20 prvků a 16 automaticky vložených registrů a pro jeho realizaci je zapotřebí 2 645 hradel cílové technologie TSMC 0,35μ. Pro tento obvod byla požadována 80% řiditelnost a pozorovatelnost; pro výsledný testovací obvod byla naměřena řiditelnost 74,5% a pozorovatelnost 80,6%.

V rámci druhého experimentu byl navržen obvod s 20% řiditelností a 33% pozorovatelností, který tvoří 30 prvků: 10xADD(8bitů), 10xSUB(8bitů) a 10xMUX2(8bitů) a 5 primárních datových vstupů a výstupů. Parametry evolučního algoritmu zůstaly stejné jako u předchozího experimentu. Výsledný obvod na obrázku 7.1(b) tvoří 20 prvků a 16 automaticky vložených registrů. Pro realizaci navrženého obvodu je zapotřebí 3 605 hradel cílové technologie TSMC

0,35μ. Navržený obvod má řiditelnost 20,4% a pozorovatelnost 36,7%.



(a) (b)

Obr. 7.1: Příklady evolucí nalezených obvodů: 20 (a) a 30 (b) prvkový obvod.

Na obrázku 7.1 je patrné, že se oba obvody z hlediska své struktury výrazně liší. Obvod na obrázku 7.1(a) – požadovaná řiditelnost/pozorovatelnost 80% – obsahuje pouze několik krátkých zpětnovazebních smyček. Naproti tomu obvod na obrázku 7.1(b) – požadovaná řiditelnost=33% a pozorovatelnost=20% – obsahuje mnoho dlouhých zpětnovazebních smyček. Toto zjištění odpovídá známému faktu, že obvody obsahující dlouhé zpětnovazební smyčky jsou obvykle obtížně testovatelné.

Výsledkem tohoto experimentu je zjištění, že navržená metoda umožňuje vytvářet syntetické testovací obvody s požadovanou strukturou a testovatelností. Další experimenty již budou zaměřeny na ověření konkrétních vlastností návrhové metody (zjištění limitů, nalezení vhodných parametrů použitého optimalizačního algoritmu, ...).

7.2 Hledání vhodných parametrů použitého EA

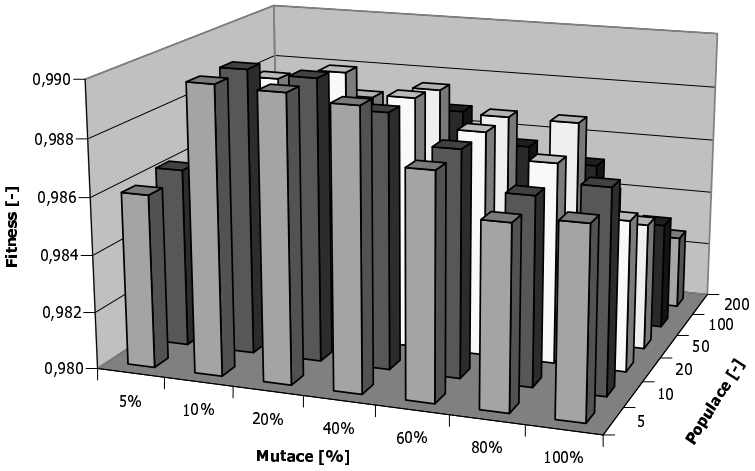
Další sada experimentů je zaměřena na hledání vhodných parametrů použitého evolučního algoritmu. Vhodnou volbou parametrů optimalizačního algoritmu můžeme dosáhnout lepších výsledků návrhové metody. V případě evolučního návrhu testovacích obvodů je důležitá zejména volba vhodné velikosti populace a počtu generací, dále počtu mutovaných spojů a nahrazované části rodičovské populace.

7.2.1 Volba velikosti populace a parametru mutace

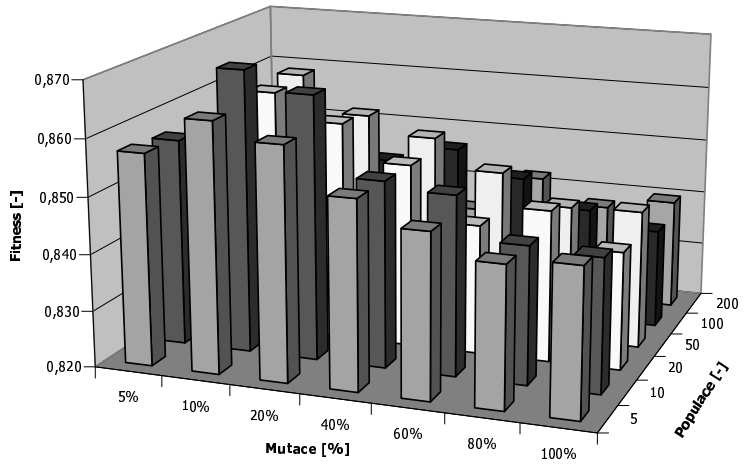
Cílem následujícího experimentu bylo nalezení vhodné velikosti populace a parametru mutace. Při hledání vhodné velikosti populace budeme vycházet z toho, že máme pro každý experiment k dispozici určitý počet možných ohodnocení kandidátních řešení. Volba větší populace tak povede na evoluční algoritmus s menším počtem generací a naopak. Cílem bude nalézt nejvhodnější poměr velikosti populace a počtu generací.

Experiment byl proveden pro 42 různých kombinací velikosti populace $N \in \{5, 10, 20, 50, 100, 200\}$ a parametru mutace $M \in \{5\%, 10\%, 20\%, 40\%, 60\%, 80\%, 100\%\}$ na obvodu o 50 (a) a 500 (b) prvcích. Parametr nahrazení R byl nastaven na 90%. Počet ohodnocení byl omezen na 2500. Počet generací je dán vztahem $G = 2500/N$. V grafu na obrázku 7.2 jsou zobrazeny průměrné hodnoty fitness pro 50 prvkový obvod získané z 15 nezávislých běhů a v grafu na obrázku 7.3 jsou průměrné hodnoty fitness pro 500 prvkový obvod získané z 10 nezávislých běhů.

Z výsledků experimentů je patrné, že velikost populace N je vhodné volit v rozsahu $N \in \langle 5, 20 \rangle$. Parametr mutace M, který určuje jaká část spojů má být v každém kroku



Obr. 7.2: Ukázka vlivu parametru mutace a velikosti populace na kvalitu výsledného řešení pro obvod s 50 prvky.



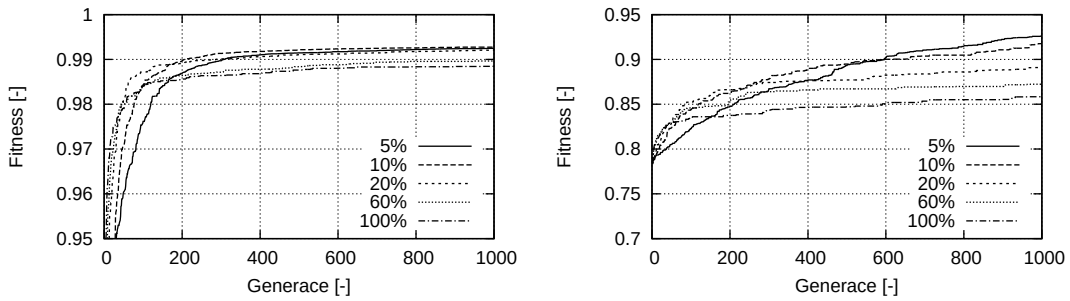
Obr. 7.3: Ukázka vlivu parametru mutace a velikosti populace na kvalitu výsledného řešení pro obvod s 500 prvky.

změněna, je vhodné volit v rozsahu $M \in \langle 8, 20 \rangle \%$.

7.2.2 Volba parametru mutace a průběh hodnoty fitness

Volba parametru mutace má vliv na rychlost konvergence návrhové metody. Pro použitý evoluční algoritmus je z hlediska dosažení lepšího řešení vhodné volit kombinaci nižší hodnoty parametru mutace a vyššího počtu generací. Vyšší hodnoty parametru mutace $M > 20\%$ zajišťují rychlou konvergenci, ale výsledné nalezené řešení je horší než při použití nižší hodnoty parametru mutace a vyššího počtu generací.

Na obrázku 7.4 je průběh hodnoty fitness v závislosti na hodnotě parametru mutace pro obvod s 50 (a) a 500 (b) prvky se stejnou strukturou, která byla použita v předchozím experimentu. Velikost populace $N = 10$, mutace $M \in \{5\%, 10\%, 20\%, 60\%, 100\%\}$ a parametr nahrazení



Obr. 7.4: Průběh hodnoty fitness pro různé hodnoty parametru mutace pro obvod s 50 (a) a 500 (b) prvky.

$R = 90\%$. Z grafů na obrázku 7.4 je zřejmé, že rychlost konvergence je vyšší pro vyšší hodnoty mutace. Pro nalezení optimálního řešení je však vhodnější volit nižší hodnotu mutace, čímž sice dosáhneme pomalejší konvergence, ale při dostatečném počtu generací získáváme lepší výsledné řešení.

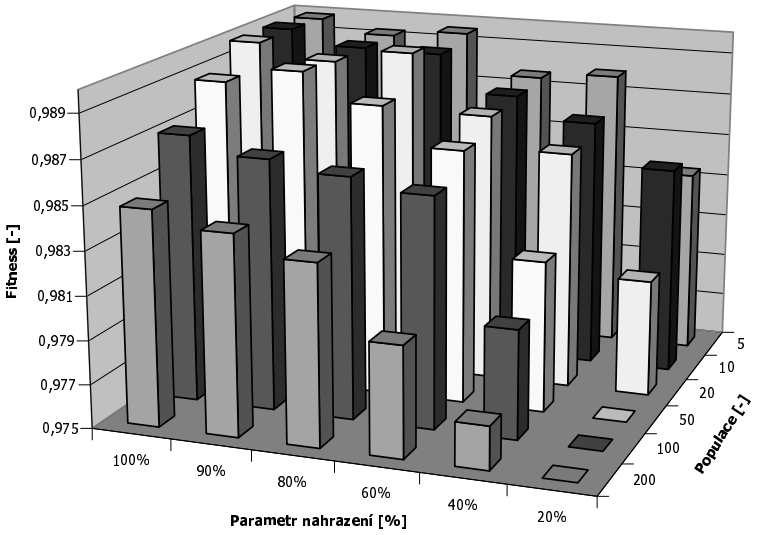
7.2.3 Volba velikosti populace a nahrazované části populace

Cílem dalšího experimentu bylo analyzovat vliv velikosti populace a parametru nahrazení na kvalitu vytvářených obvodů. Experiment byl realizován pro obvody o 50 (a) a 500 (b) prvcích se stejnou strukturou jako v předchozích experimentech a 36 kombinací velikosti populace $N \in \{5, 10, 20, 50, 100, 200\}$ a parametru nahrazení $R \in \{20\%, 40\%, 60\%, 80\%, 90\%, 100\%\}$. Parametr mutace M byl zvolen 20% a počet ohodnocení byl omezen na 2500. V grafu na obrázku 7.5 jsou zobrazeny průměrné hodnoty fitness pro 50 prvkový obvod získané z 15 nezávislých běhů. V grafu na obrázku 7.6 jsou průměrné hodnoty fitness pro 500 prvkový obvod získané z 10 nezávislých běhů.

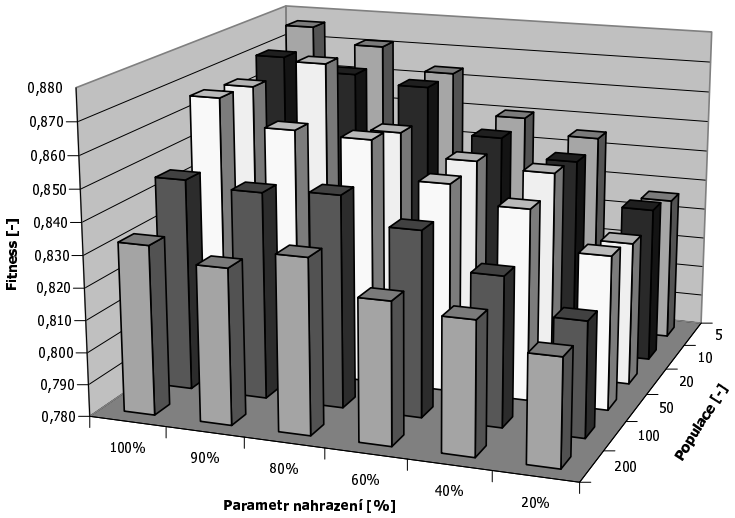
Výsledky tohoto experimentu ukazují, že velikost populace N je vhodné volit v rozsahu $N \in \langle 5, 20 \rangle$ a parametr nahrazení $R \in \langle 80, 100 \rangle\%$. Je potřeba si ale uvědomit, že volba $R = 100\%$ neznamená, že by všichni jedinci populace rodičů byly nahrazeni jedinci z populace potomků. Nahrazení je realizováno turnajem – úplné nahrazení by mohlo nastat teoreticky pouze tehdy, pokud by všichni potomci představovali lepší řešení.

7.3 Ověření možností návrhové metody

Cílem dalšího experimentu bylo analyzovat, jak složité obvody a v jakých časových relacích je vytvořená metoda schopna navrhovat a zároveň, zda je metoda schopna plnit uživatelem specifikované požadavky na testovatelnost i při různé složitosti vytvářených obvodů. Experiment byl realizován pro obvody, které obsahovaly 50, 100, 500 a 1000 prvků na úrovni RT. Pro všechny obvody byl stejný požadavek na testovatelnost obvodu – 33% řiditelnost a pozorovatelnost. V tabulce 7.1 jsou uvedeny diagnostické vlastnosti vytvářených obvodů (průměrné hodnoty z 5 nezávislých spuštění), jejich složitost vyjádřená v počtu základních prvků technologie TSMC a čas potřebný pro vytvoření těchto testovacích obvodů (počítač s procesorem Intel Xeon 2,8GHz CPU a 2GB RAM).



Obr. 7.5: Ukázka vlivu parametru nahrazení a velikosti populace na kvalitu výsledného řešení pro obvod s 50 prvky.



Obr. 7.6: Ukázka vlivu parametru nahrazení a velikosti populace na kvalitu výsledného řešení pro obvod s 500 prvky.

Z výsledků je patrné, že navržená metoda je schopna navrhovat obvody o složitosti až tisíce prvků na úrovni RT při splnění uživatelem specifikovaných požadavků na diagnostické vlastnosti vytvářených obvodů. Časová složitost návrhu obvodů této složitosti se pohybuje v řádu hodin. Výsledná složitost obvodů na úrovni hradel je závislá na složitosti prvků, z nichž se obvod skládá. Složitost obvodů uvedená v tabulce 7.1 je uvedena pro základní prvky úrovně RT jako jsou 16/32 bitové sčítačky, odečítačky a násobičky. Je potřeba si ale uvědomit, že prvky, z nichž se obvod skládá, mohou reprezentovat také složité obvodové konstrukce, v tom případě je složitost vytvářených obvodů omezena pouze nástrojem, který je použit pro syntézu navržených obvodů. Při použití nástroje Leonardo Spectrum se maximální složitost vytvářených

Tabulka 7.1: Složitost a testovatelnost navržených obvodů

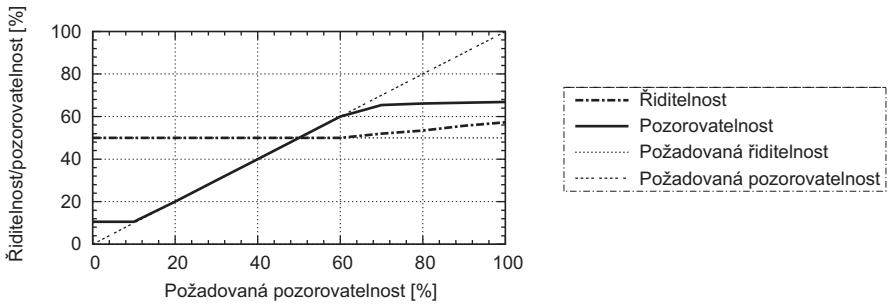
Požadavky		**Dosažené výsledky**				
Vst./Výst. [-]	Prvků [-]	Řiditelnost [%]	Pozorovatelnost [%]	Počet hradel [-]	Počet KO [-]	Čas [hh:mm]
5/5	50	32,76%	32,15%	27 452	2 509	00:10
10/10	100	32,12%	30,42%	55 720	5 176	00:17
40/40	500	33,57%	30,84%	277 674	25 584	02:47
80/80	1 000	34,35%	29,92%	565 193	51 717	11:52

obvodů pohybuje na hranici 600 000 hradel.

7.4 Rozsah plnění požadavků řiditelnosti a pozorovatelnosti

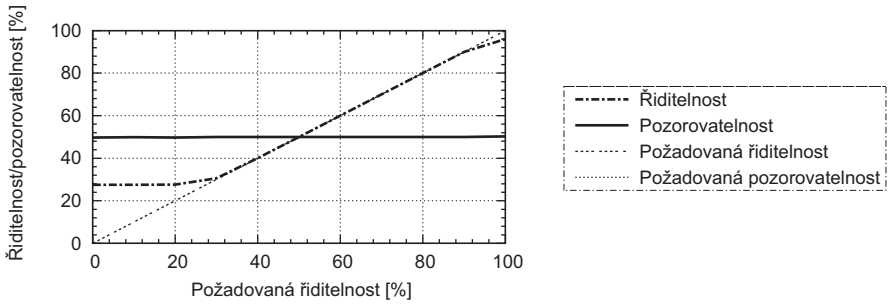
Další experiment je zaměřen na analýzu rozsahu, na němž je metoda schopna plnit uživatelem specifikované požadavky na testovatelnost obvodu v případě, že: (a) řiditelnost je pevně dána (=50%) a mění se požadovaná pozorovatelnost v rozsahu od 0% do 100% s krokem 10% a podobně (b) pozorovatelnost je pevně dána a mění se požadovaná řiditelnost. Experiment byl realizován na obvodu, který se skládá z 50 prvků: 8xADD(8bitů), 8xSUB(8bitů), 8xMUX2(8bitů), 8xADD(16bitů), 8xSUB(16bitů), 8xMUX2(16bitů) a 2xMUL(8,16bitů) a 8 primárních vstupů a výstupů. Evoluční algoritmus byl spuštěn s následujícími parametry – velikost populace $N = 30$, počet generací $G = 200$, procento mutovaných spojů $M = 2\%$ a nahrazovaná část populace $R = 95\%$

Na obrázku 7.7 jsou výsledky získané z 20 nezávislých spuštění pro každou kombinaci pevně dané požadované řiditelnosti (=50%) a požadované pozorovatelnosti $\langle 0, 100 \rangle$. Z výsledků je patrné, že navržená metoda je schopna, pro zadané primární vstupy a výstupy a zadaný počet a typ prvků, plnit požadavky na pozorovatelnost přibližně v rozsahu od 10% do 60%.



Obr. 7.7: Rozsah plnění požadavků testovatelnosti při pevně dané řiditelnosti (=50%) a měnícím se požadavkům na pozorovatelnosti ($\langle 0, 100 \rangle$).

Na obrázku 7.8 jsou výsledky stejného experimentu získané pro pevně danou pozorovatelnost (=50%) a požadovanou řiditelnost $\langle 0, 100 \rangle$. Z výsledků na obrázku 7.8 je vidět, že metoda je schopna plnit požadavky uživatele pro řiditelnost přibližně v rozsahu od 30% do 90%.



Obr. 7.8: Rozsah plnění požadavků testovatelnosti při pevně dané pozorovatelnosti (=50%) a měnícím se požadavkům na řiditelnost ($\langle 0, 100 \rangle$).

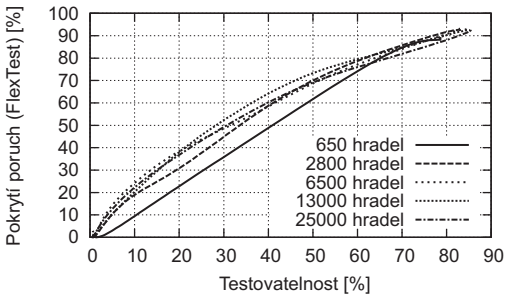
Metoda je tedy schopna plnit požadavky uživatele pouze na určitém rozsahu vstupních hodnot řiditelnosti a pozorovatelnosti, který je dán především počtem primárních vstupů a výstupů obvodu a vlastnostmi použitých prvků. Nelze totiž například vytvořit obvod s vysokou řiditelností, pokud jsou ve struktuře obvodu použity prvky bez transparentních režimů nebo je použit nedostatečný počet primárních vstupů obvodu.

7.5 Ověření metody profesionálními nástroji

Až dosud byly všechny experimenty zaměřené na hledání vhodných parametrů, ověření plnění zadaných požadavků, popř. hledání možností návrhové metody. Velmi důležitou částí experimentálního ověření je ale také ověření vlastností navrhovaných obvodů z hlediska možnosti jejich realizace a také ověření jejich diagnostických vlastností nejen pomocí metod použitých v této práci, ale také pomocí profesionálních nástrojů. Následující sada experimentů je tak zaměřena na ověření realizovatelnosti (syntetizovatelnosti) vytvářených obvodů pomocí nástroje Leonardo Spectrum a na ověření diagnostických vlastností vytvářených obvodů pomocí nástroje FlexTest.

Pro účely tohoto experimentu byla navrženou metodou vytvořena sada obvodů s různou složitostí (650, 2 800, 6 500, 13 000 a 25 000 hradel v technologii TSMC), kde pro každou úroveň složitosti bylo vytvořeno 50 obvodů s různými požadavky na řiditelnost a pozorovatelnost v rozsahu $\langle 0, 100 \rangle \%$. První část experimentálního ověření byla zaměřena na ověření realizovatelnosti navržených obvodů – všechny vytvořené obvody byly analyzovány pomocí nástroje Leonardo Spectrum.

Náročnější problém představuje ověření diagnostických vlastností vytvářených obvodů. V současné době neexistuje žádná uznávaná míra ohodnocení diagnostických vlastností, prostřednictvím které by bylo možné zjistit, zda obvody navrhované v této práci jako obtížně testovatelné (nízká řiditelnost/pozorovatelnost) jsou skutečně obtížně testovatelné a naopak. Jako referenční metoda pro ohodnocení diagnostických vlastností byl v této práci použit nástroj FlexTest – profesionální nástroj pro automatické generování testu. Tento nástroj byl použit pro generování testu pro všechny vytvořené obvody a jako míra obtížnosti testování daného obvodu byl pak použit parametr pokrytí poruch. Na obrázku 7.9 je zobrazena relace mezi hodnotou testovatelnosti obvodu získanou pomocí metody ohodnocení testovatelnosti použité v této práci (testovatelnost je dána jednoduchým vztahem *testovatelnost=(řiditelnost+pozorovatelnost)/2*) a parametrem pokrytí poruch získaným pomocí nástroje FlexTest.



Obr. 7.9: Vztah mezi testovatelností vyjádřenou parametry řiditelnosti a pozorovatelnoti a pokrytím poruch měřeným nástrojem FlexTest.

Z výsledku na obrázku 7.9 je patrné, že obvody s vysokou řiditelností a testovatelností jsou také obtížně testovatelné (mají nižší pokrytí poruch) a naopak obvody s dobrou řiditelností a pozorovatelností jsou snadno testovatelné (mají vyšší pokrytí poruch). Experiment tedy potvrdil, že existuje relace mezi výsledky metody analýzy testovatelnosti navržené v této práci a obtížností testování vytvořených obvodů vyjádřenou generátorem testu.

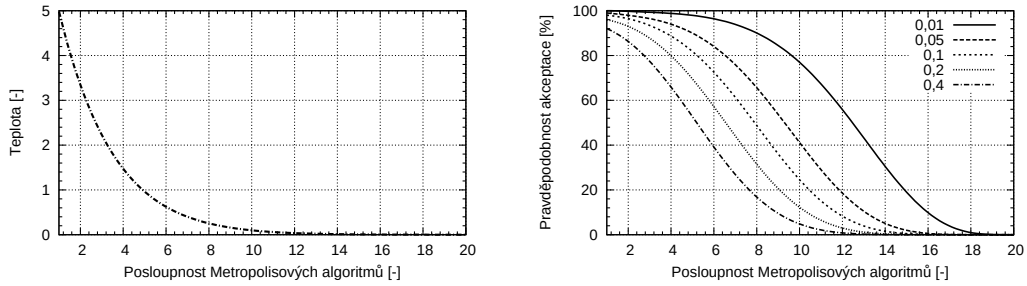
7.6 Použití alternativních optimalizačních technik

Návrh testovacích obvodů je v této práci realizován prostřednictvím optimalizačního algoritmu evolučního programování. Na základě provedených experimentů můžeme konstatovat, že tato metoda je vhodná pro návrh testovacích obvodů. Existují však také další optimalizační techniky (např. horolezecký algoritmus, simulované žíhání, . . .). Možností použití dalších optimalizačních metod pro návrh testovacích obvodů se v rámci své diplomové práce zabýval Tomáš Kantor [55]. Ten využil existující metodu návrhu testovacích obvodů a její implementaci ve formě nástroje Cirgen a metodu doplnil o možnost využít pro návrh struktury obvodu také metodu simulovaného žíhání. V této podkapitole tak můžeme provést srovnání výsledků dosažených pomocí evolučního programování a simulovaného žíhání.

Kantor ve své práci implementoval základní variantu simulovaného žíhání představenou v podkapitole 2.6.3. Základem pro správné fungování této optimalizační metody je vhodná volba počáteční teploty T_{max}, konečné teploty T_{min}, průběhu snižování teploty a počtu iterací k_{max} v rámci každé teploty. Na základě výsledků experimentů byl zvolen průběh snižování teploty podle vztahu $T = \frac{100}{e^x}$, kde $x \in \langle 3; 12,5 \rangle$. Počáteční teplota $T_{max} \approx 5$ a konečná teplota $T_{min} \approx 0$. Počet iterací v rámci jedné teploty závisí na velikosti navrhovaného obvodu. Experimentálně bylo zjištěno, že počet iterací v rámci jedné teploty x_{max} je vhodné volit jako $x_{max} = 0,1 \cdot |E|$, kde $|E|$ vyjadřuje počet prvků obvodu.

Na obrázku 7.10(a) je ukázka průběhu snižování teploty z počáteční teploty $T_{max} \approx 5$ na konečnou teplotu $T_{min} \approx 0$ pro 20 kroků. Na obrázku 7.10(b) je vyjádřena pravděpodobnost akceptování řešení horší kvality v průběhu jednotlivých kroků snižování teploty. Z grafu je patrné, že algoritmus při maximální teplotě akceptuje téměř všechny nové stavy a postupně se snižováním teploty jsou akceptovány pouze stavy představující lepší řešení než stav aktuální.

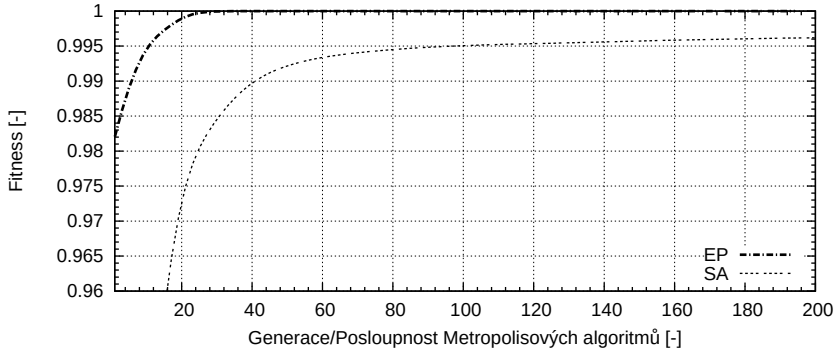
Přejděme nyní k vlastnímu experimentu, jehož cílem bylo porovnat výsledky dosažené pomocí optimalizačních technik simulovaného žíhání a evolučního programování. Oba algoritmy byly použity pro vytvoření obvodu, který tvoří 50 prvků: 8xSUB(8bitů), 8xADD(8bitů),



(a) (b)

Obr. 7.10: Ukázka průběhu snižování teploty (a) a pravděpodobnosti akceptování horšího jedince v průběhu snižování teploty (b).

8xMUX2(8bitů), 2xMUL(8,16bitů), 8xSUB(16bitů), 8xADD(16bitů), 8xMUX2(16bitů) a 10 primárních vstupů a výstupů. Parametry obou algoritmů byly zvoleny tak, aby byly vytvořeny srovnatelné podmínky – zejména pak počet volání hodnotící funkce, která má největší vliv na dobu potřebnou pro vytváření testovacích obvodů. Srovnání výsledků obou metod je k dispozici na obrázku 7.11. Výsledky byly získány jako průměrná hodnota z 30 nezávislých spuštění.



Obr. 7.11: Porovnání optimalizační metody evolučního programování (EP) a simulovaného žíhání (SA).

Z výsledků je zřejmé, že optimalizační algoritmus evolučního programování poskytuje rychlejší konvergenci k optimálnímu řešení a je tedy vhodnější pro evoluční návrh testovacích obvodů.

7.7 Shrnutí

Cílem této kapitoly bylo experimentálně ověřit implementaci metody návrhu syntetických testovacích obvodů. Navržená metoda byla nejprve ověřena na sadě experimentů s cílem analyzovat, zda skutečně umožňuje vytvářet syntetické testovací obvody s požadovanou strukturou a diagnostickými vlastnostmi. V další části pak byly analyzovány vhodné rozsahy parametrů použité optimalizační metody, možnosti návrhové metody a vlastnosti navržených obvodů. Kvalita výsledných obvodů byla ověřena pomocí profesionálních nástrojů firmy Mentor Graphics. Výsledkem provedených experimentů bylo zjištění, že návrhová metoda umožňuje navrhovat

syntetické testovací obvody s požadovanou strukturou a diagnostickými vlastnostmi až do řádu statisíců hradel. Je zajímavé, že omezující podmínkou je v našem případě maximální velikost obvodu, který je schopen syntézní nástroj Leonardo Spectrum zpracovat. Diagnostické vlastnosti vytvářených obvodů byly ověřeny pomocí profesionálního nástroje FlexTest. Provedené experimenty potvrdily, že obvody s vysokou řiditelností a testovatelností jsou také obtížně testovatelné (mají nižší pokrytí poruch) a naopak obvody s dobrou řiditelností a pozorovatelností jsou snadno testovatelné (mají vyšší pokrytí poruch). Experiment tedy potvrdil, že existuje relace mezi výsledky metody analýzy testovatelnosti navržené v této práci a obtížností testování vytvořených obvodů vyjádřenou generátorem testu. Právě ověření kvality výsledných obvodů pomocí profesionálních nástrojů pokládám za velmi přínosné.

Kapitola 8

FITTest_BENCH06: Nová sada testovacích obvodů

Hlavní motivací pro vznik této práce byl nedostatek testovacích obvodů na úrovni RT, které by bylo možné použít pro ověřování diagnostických metod a nástrojů. Nyní máme k dispozici metodu, která umožňuje vytvářet testovací obvody s požadovanou složitostí a diagnostickými vlastnostmi. Použijeme ji pro návrh sady syntetických testovacích obvodů FITTest_BENCH06 [76] (**F**aculty of **I**nformation **T**echnology **Test**ability **Bench**marks 2006).

8.1 Popis testovacích obvodů sady

Sadu FITTest_BENCH06 tvoří 31 syntetických sekvenčních testovacích obvodů. První část sady tvoří 20 obvodů s různou složitostí a různými diagnostickými vlastnostmi. Druhou část tvoří 11 obvodů se stejnou složitostí ale různými diagnostickými vlastnostmi. Testovací obvody této sady jsou k dispozici jako strukturou popsané obvody na úrovni RT v syntaxi jazyka VHDL a dále na úrovni hradel technologie 0,35μ TSMC popsané v syntaxi jazyka VHDL, Verilog a EDIF. Právě dostupnost těchto obvodů na různých úrovních a také v různých formátech umožňuje jejich jednoduché použití bez nutnosti provádět další konverze, při nichž může docházet k jistým ztrátám (např. v důsledku optimalizací). Podívejme se nyní podrobněji na obvody obsažené v této sadě.

Cílem při vytváření první části sady testovacích obvodů bylo poskytnout uživateli široké spektrum obvodů s různou složitostí a různými diagnostickými vlastnostmi. První část tak tvoří 20 obvodů se složitostí cca 2 000, 10 000, 28 000, 150 000 a 300 000 hradel a různými diagnostickými vlastnostmi. Vzhledem k tomu, že neexistuje obecně uznávaná míra testovatelnosti, byl v této práci jako míra testovatelnosti použit parametr pokrytí poruch získaný nástrojem FlexTest. Pro každou výše uvedenou složitost obvodů byly vytvořeny 4 varianty obvodů s různou úrovní pokrytí poruch – cca 0%, 33%, 66% a 100%. Vzhledem k zaměření této práce nemá význam popisovat jednotlivé obvody z hlediska jejich struktury a způsobu návrhu – tyto informace jsou k dispozici např. v [76, 100]. V této práci si představíme pouze základní parametry vytvořených testovacích obvodů.

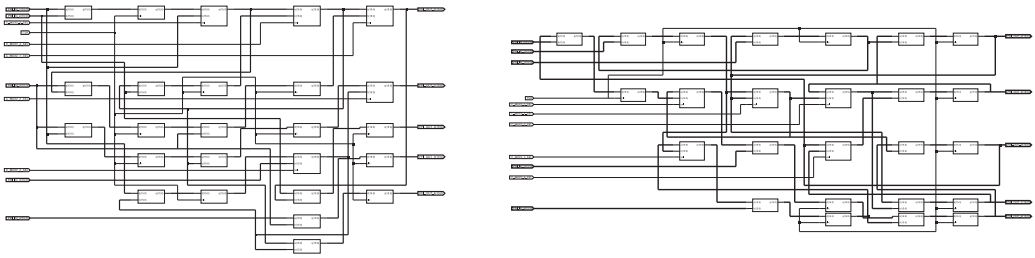
V tabulce 8.1 je uveden přehled základních parametrů obvodů první části sady. Pro každý obvod je k dispozici informace o počtu primárních vstupů a výstupů obvodu, počtu použitých hradel a klopných obvodů a také informace o počtu poruch a pokrytí poruch dosaženého

Tabulka 8.1: Testovací obvody sady FITTest_BENCH06 – obvody s různou složitostí a diagnostickými vlastnostmi.

Obvod	**Počet vst.**	**Počet výst.**	**Počet KO**	**Počet hradel**	**Počet poruch**	**Pokrytí poruch**
e01	86	80	160	1 985	6 830	90,45%
e02	86	80	144	1 657	5 716	60,69%
e03	86	80	160	2 046	7 114	39,43%
e04	86	80	160	2 221	7 900	0,00%
e05	186	160	792	10 011	33 146	90,11%
e06	186	160	831	9 999	33 028	43,90%
e07	186	160	785	9 894	33 338	22,87%
e08	186	160	778	9 559	31 358	0.00%
e09	211	192	2 020	28 065	93 556	91,90%
e10	179	208	1 979	27 853	93 132	64,22%
e11	211	200	2 058	28 231	94 488	27,46%
e12	203	208	2 106	28 438	93 948	0,00%
e13	1 669	1 904	6 304	155 046	572 722	89.38%
e14	1 621	1 904	6 368	155 380	572 890	64.46%
e15	1 701	1 840	6 368	155 207	572 430	31.84%
e16	1 589	1 744	6 368	155 045	572 025	12.50%
e17	3 833	4 272	12 672	310 122	1 146 522	81.73%
e18	3 913	4 512	12 608	309 856	1 147 050	56.72%
e19	3 833	4 320	12 576	309 874	1 146 978	40.28%
e20	3 961	4 352	12 736	310 610	1 148 130	23.13%

nástrojem FlexTest pro tyto obvody. V tabulce je vidět, že obvody první části sady poskytují uživateli široké spektrum složitostí a diagnostických vlastností. Na obrázku 8.1 je ukázka struktury dvou obvodů *e01* a *e04* ze sady FITTest_BENCH06. Oba uvedené obvody se skládají ze stejných prvků při jejich vytváření ale byly požadovány různé diagnostické vlastnosti – obvod *e01* (obrázek 8.1(a)) byl navrhován jako snadno testovatelný a obvod *e04* (obrázek 8.1(b)) jako obtížně testovatelný. Na obrázku 8.1 je vidět, jak se požadavky na testovatelnost projevily na struktuře výsledných obvodů – obvod *e04* má více zpětnovazebních smyček, které mají za následek obtížné testování obvodu.

Cílem při vytváření obvodů druhé části testovací sady bylo poskytnout uživateli množinu obvodů se stejnou složitosti a širokým spektrem diagnostických vlastností obvodů. Druhou část testovací sady tak tvoří 11 obvodů se složitostí přibližně 100 000 hradel a různými diagnostickými vlastnostmi vyjádřenými parametrem pokrytí poruch (pokrytí poruch se pohybuje přibližně od 1% do 95%). V tabulce 8.2 je opět základní přehled parametrů pro obvody druhé části sady. Pro každý obvod je k dispozici informace o počtu použitých hradel a klopných obvodů a také informace o počtu poruch a pokrytí poruch dosaženého nástrojem FlexTest pro tyto obvody.



(a) (b)

Obr. 8.1: Porovnání struktury snadno (a) a obtížně (b) testovatelného obvodu na úrovni RT.

Tabulka 8.2: Testovací obvody sady FITTest_BENCH06 – obvody stejné složitosti ale s různými diagnostickými vlastnostmi.

Obvod	**Počet hradel**	**Počet KO**	**Počet poruch**	**Pokrytí poruch**
a00	108 627	4 384	399 806	1,28%
a01	108 748	4 448	399 786	10,11%
a02	108 532	4 416	398 012	23,81%
a03	108 876	4 448	400 166	31,60%
a04	108 551	4 416	399 334	40,38%
a05	108 740	4 448	399 534	50,19%
a06	108 607	4 416	399 494	65,56%
a07	108 811	4 448	399 708	66,37%
a08	108 650	4 448	399 566	74,86%
a09	108 345	4 384	398 888	86,54%
a10	108 652	4 448	399 112	94,29%

8.2 Shrnutí

V této kapitole byla představena sada testovacích obvodů FITTest_BENCH06, kterou tvoří 31 syntetických sekvenčních testovacích obvodů vytvořených s využitím metody návrhu testovacích obvodů navržené v této práci. Obvody obsažené v sadě můžeme rozdělit na dvě části. První část tvoří 20 obvodů s různou složitostí (≈2 000, 10 000, 28 000, 150 000 a 300 000 hradel) a různými diagnostickými vlastnostmi (pokrytí poruch ≈0%, 33%, 66% a 100%). Druhou část představuje 11 obvodů se shodnou složitostí a různými diagnostickými vlastnostmi (pokrytí poruch ≈0-100%). Sada testovacích obvodů FITTest_BENCH06 je k dispozici (společně s nástrojem, který byl použit pro jejich vytvoření) na adrese http://www.fit.vutbr.cz/∼pecenka/cirgen. Pro každý obvod je k dispozici podrobná informace o struktuře obvodu společně s informacemi o diagnostických vlastnostech zjištěných ATPG nástrojem FlexTest.

Kapitola 9

Závěr

V poslední kapitole této práce jsou shrnuty dosažené výsledky, představeny hlavní přínosy práce a naznačeny možné směry dalšího výzkumu.

9.1 Shrnutí výsledků práce

Tato práce se zabývala možností návrhu syntetických testovacích obvodů s požadovanou složitostí a diagnostickými vlastnostmi, které by bylo možné použít pro ověřování diagnostických metod a nástrojů. Práce vycházela z poměrně rozsáhlé analýzy aktuálního stavu v oblasti existujících sad testovacích obvodů, metod vytváření syntetických testovacích obvodů, možností využití evolučních algoritmů pro návrh obvodů a metod ohodnocení testovatelnosti obvodu na úrovni RT. Na základě výsledků těchto analýz bylo konstatováno, že v současné době není k dispozici dostatek testovacích obvodů pro ověření diagnostických metod a nástrojů pracujících na úrovni RT a současně také neexistuje metoda, která by umožňovala vhodné testovací obvody vytvářet. V rámci analýzy aktuálního stavu v oblasti evolučního návrhu však byly identifikovány možnosti využití těchto metod pro návrh testovacích obvodů. Další část práce tak byla věnována návrhu metody založené na evolučním návrhu umožňující vytvářet syntetické testovací obvody s požadovanými diagnostickými vlastnostmi popsané na úrovni RT.

Navržená metoda představuje nový přístup k vytváření syntetických testovacích obvodů. Uživatel má možnost specifikovat požadavky na vlastnosti vytvářených obvodů ve formě počtu bran rozhraní, počtu a typu prvků, z nichž se má obvod skládat a požadovaných diagnostických vlastností obvodu. Vlastní návrh testovacích obvodů je realizován jako optimalizační proces, jehož cílem je najít nejvhodnější propojení uživatelem specifikovaných prvků tak, aby byly splněny požadavky na diagnostické vlastnosti vytvářených obvodů. Celý proces návrhu, včetně reprezentace obvodu a reprodukčních operátorů byl navržen tak, aby vytvářené obvody byly elektricky korektní.

Optimalizačním kritériem při návrhu jsou vlastnosti obvodu z hlediska struktury spojů a testovatelnosti. Pro ohodnocení těchto vlastností byly vytvořeny algoritmy, které pracují přímo nad formální reprezentací obvodu. Analýza struktury obvodu spočívá v identifikaci izolovaných prvků obvodu a nežádoucích konstrukcí z hlediska spojů. Analýza testovatelnosti kandidátního řešení představuje časově nejnáročnější část procesu návrhu syntetických testovacích obvodů. Při hledání metody analýzy testovatelnosti vhodné pro evoluční návrh bylo nutné najít kompromis mezi časovou složitostí a přesností použité metody. Metoda analýzy testovatelnosti

navržená v této práci vychází z principu metody *ADFT* [86] a dále tuto metodu rozšiřuje tak, aby byla vhodná pro evoluční návrh testovacích obvodů. Navržená metoda umožňuje realizovat analýzu testovatelnosti hierarchicky popsaného obvodu na úrovni RT a dokáže také zohlednit obtížnost testování jednotlivých prvků obvodu. Metoda také umožňuje přidávat nové prvky do knihovny prvků, se kterou pracuje a dokáže automaticky identifikovat transparentní režimy vkládaných prvků [75].

Součástí práce je poměrně rozsáhlé experimentální ověření návrhové metody. Navržená metoda byla podrobena řadě experimentů s cílem analyzovat nejvhodnější parametry použitého evolučního algoritmu, ověřit možnosti návrhové metody z hlediska rozsahu, na kterém je metoda schopna plnit uživatelem specifikované požadavky a také ověřit vlastnosti výsledných obvodů. Kvalita výsledných obvodů byla v této práci ověřena pomocí profesionálních nástrojů firmy Mentor Graphics. Výsledkem provedených experimentů je zjištění, že návrhová metoda umožňuje navrhovat syntetické testovací obvody s požadovanou složitostí a diagnostickými vlastnostmi v řádu až statisíců hradel. Omezující podmínkou v tomto případě však není samotná návrhová metoda, ale použitý syntézní nástroj Leonardo Spectrum.

Funkčnost navržené metody byla také demonstrována prostřednictvím návrhu sady 31 testovacích obvodů FITTest_BENCH06 [76,100] určených pro ověřování diagnostických metod a nástrojů.

9.2 Přínos práce

Hlavní přínosy této práce můžeme shrnout v následujících odstavcích.

Navržen formální model obvodu na úrovni RT, který umožňuje reprezentovat strukturu a umožňuje modelovat transparentní cesty ve struktuře obvodu na úrovni RT. Oproti existujícím modelům (viz např. [81, 86]) umožňuje navržený model kompaktní zápis struktury obvodu prostřednictvím trojice reprezentující informaci o rozhraní obvodu, prvcích, z nichž se obvod skládá a informaci o vzájemném propojení těchto prvků. Model podporuje také hierarchický popis struktury obvodu, dokáže modelovat bitovou šířku jednotlivých bran obvodu a pro účely analýzy testovatelnosti umožňuje rozlišit řídicí a datové brány. Výhodou formálního přístupu je zejména jednoznačnost zápisu a možnost transformovat problémy návrhu na známé a řešené problémy z oblasti diskrétní matematiky a teoretické informatiky a využití známých, efektivních a ověřených postupů a algoritmů pro jejich řešení.

Navržena metoda evolučního návrhu syntetických testovacích obvodů, která umožňuje vytvářet syntetické testovací obvody s požadovanou složitostí a diagnostickými vlastnostmi. Navržená metoda využívá nový přístup k vytváření syntetických testovacích obvodů, který je založen na evolučním návrhu. Uživatel má možnost specifikovat požadavky na vlastnosti vytvářených obvodů ve formě počtu bran rozhraní, počtu a typu prvků, z nichž se má obvod skládat a požadovaných diagnostických vlastností obvodu. Návrh testovacích obvodů je realizován jako optimalizační proces, který za pomocí algoritmu evolučního programování hledá nejvhodnější propojení uživatelem specifikovaných prvků tak, aby byly splněny požadované diagnostické vlastnosti.

Odstraněn problém škálovatelnosti evolučního návrhu. Základním problémem všech dosud existujících přístupů využívajících evoluční návrh pro návrh číslicových obvodů bylo, že byly schopny vytvářet pouze jednoduché číslicové obvody realizující požadovanou funkci o složitosti v řádu maximálně jednotek tisíc hradel [83]. Důvodem problematické škálova-

telnosti byl zejména exponenciální růst doby potřebné pro ohodnocení kandidátního řešení. Pokud zvýšíme počet vstupů číslicového obvodu o jeden, tak se počet všech možných vstupních kombinací, pro které je potřeba ohodnotit funkci daného obvodu, zdvojnásobí. Řešení problému škálovatelnosti použité v této práci spočívá v aplikaci evolučního návrhu pouze na problémy, u nichž jsme schopni dosáhnout ohodnocení kandidátního řešení v polynomiálním čase. Příkladem právě takového problému je návrh syntetických testovacích obvodů s požadovanými diagnostickými vlastnostmi, kdy je ohodnocení struktury a testovatelnosti obvodu realizováno v polynomiálním čase. V důsledku toho je navržená metoda schopna navrhovat číslicové obvody o složitosti až statisíců hradel, které představují doposud největší evolucí navržené obvody s požadovanými diagnostickými vlastnostmi.

Vytvořena metoda analýzy testovatelnosti vhodná pro evoluční návrh testovacích obvodů. Navržená metoda pracuje nad formální reprezentací obvodu zavedenou v této práci a umožňuje analyzovat řiditelnost a pozorovatelnost bran strukturou popsaného číslicového obvodu na úrovni RT. Analýza testovatelnosti pracuje nad knihovnou prvků, která obsahuje informace o transparentních vlastnostech jednotlivých typů prvků. Na základě této informace pak metoda analyzuje dostupnost jednotlivých bran obvodu z primárních vstupů a výstupů obvodu. Hodnota výsledné řiditelnosti a pozorovatelnosti je vypočtena na základě dostupnosti bran obvodu, jejich bitové šířky a obtížnosti testování prvků, ke kterým brány náleží.

Metoda návrhu syntetických testovacích obvodů byla experimentálně ověřena pomocí profesionálních nástrojů Mentor Graphics. Navržená metoda byla podrobena sadě experimentů s cílem nalézt vhodné parametry použité optimalizační metody a identifikovat možnosti metody z hlediska splnění uživatelem specifikovaných parametrů. Vlastnosti navržených testovacích obvodů byly analyzovány nástrojem Leonardo Spectrum z hlediska realizovatelnosti navržených obvodů v technologii $0,35\mu$ TSMC a nástrojem FlexTest z hlediska diagnostických vlastností vytvořených obvodů. Provedené experimenty potvrdily, že obvody s vysokou řiditelností a testovatelností jsou také obtížně testovatelné (mají nižší pokrytí poruch) a naopak obvody s dobrou řiditelností a pozorovatelností jsou snadno testovatelné (mají vyšší pokrytí poruch) – tzn. existuje relace mezi výsledky metody analýzy testovatelnosti navržené v této práci a obtížností testování vytvořených obvodů vyjádřenou generátorem testu.

Vytvořena sada testovacích obvodů FITTest_BENCH06, kterou tvoří 31 syntetických testovacích obvodů [100]. Testovací obvody sady FITTest_BENCH06 mohou díky nekonvenčnímu způsobu návrhu obsahovat konstrukce, které se obvykle nevyskytují v obvodech navržených pomocí klasických technik návrhu založených na dekompozici a minimalizaci [83]. Díky tomu tato testovací sada může pomoci odhalit i takové typy problémů, které by zůstaly při použití klasických testovacích obvodů skryty (viz například [52]). Pomocí jednoho z navržených testovacích obvodů byla například odhalena chyba v profesionálním ATPG nástroji FlexTest.

9.3 Možná rozšíření a další práce

Z hlediska možnosti směrování dalšího výzkumu je stále otevřená problematika návrhu efektivní metody inkrementální analýzy testovatelnosti, která by umožnila analyzovat testovatelnost pouze změněných částí obvodu. Návrh takové metody by představoval efektivnější návrh zejména složitějších obvodů, kde dochází při každé změně struktury obvodu k opakované analýze testovatelnosti celého obvodu. První výzkumné aktivity, které proběhly v rámci této práce, však narážely na NP časovou složitost algoritmu identifikace částí obvodu, jejichž

testovatelnost byla ovlivněna danou změnou.

Otevřená zůstává také otázka analýzy transparentních režimů prvku při vkládání prvku do knihovny prvků. V této práci byla navržena metoda identifikace transparentních režimů na úrovni strukturou popsaných obvodů. V rámci obecnosti navržené metody by bylo vhodné umožnit také analýzu transparentních vlastností prvků popsaných chováním.

Samostatnou kapitolou tvoří možná rozšíření metody z hlediska kvality vytvářených obvodů. Příkladem možného rozšíření je například možnost specifikovat maximální kombinační zpoždění obvodu.

Literatura

[1] 1522 IEEE Trial-Use Standard for Testability and Diagnosability Characteristics and Metrics. 5 2005, ISBN 0-7381-4492-4, 35 s.

[2] Abadir, M. S.; Breuer, M. A.: A Knowledge-Based System for Designing Testable VLSI Chips. *IEEE Design & Test*, ročník 2, č. 4, 1985: s. 56–68, ISSN 0740-7475.

[3] Abramovici, M.; Breuer, M. A.; Friedman, A. D.: *Digital Systems Testing and Testable Design*. Piscataway: IEEE Press, 1990, ISBN 0-7803-1062-4, 680 s.

[4] Bäck, T.: *Evolutionary Algorithms in Theory and Practice*. Oxford: Oxford University Press, 1996, ISBN 0-19-509971-0, 314 s.

[5] Bennetts, R.: *Design of Testable Logic Circuits*. Addison-Wesley, 1984, ISBN 0-2011-4403-4, 256 s.

[6] Bentley, P.: *Evolutionary Design by Computers*. San Francisco, CA: Morgan Kaufmann Publishers, 1999, ISBN 1-5586-0605-X, 446 s.

[7] Boyd, S.; Vandenberghe, L.: *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004, ISBN 0-5218-3378-7, 730 s.

[8] Breuer, M. A.; Lien, J.-C.: A Test and Maintenance Controller for a Module Containing Testable Chips. In *Proceedings International Test Conference 1988*, Washington, D.C., USA: IEEE Computer Society, 1988, s. 502–513.

[9] Brglez, F.; Bryan, D.; Kozminski, K.: Combinational Profiles of Sequential Benchmark Circuits. In *Proceedings International Symposium on Circuits and Systems (ISCAS)*, Portland, OR, 1989, s. 1924–1934.

[10] Brglez, F.; Fujiwara, H.: A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Simulator in Fortran. In *Proceedings International Symposium on Circuits and Systems (ISCAS)*, Kyoto, Japan, 1985, s. 695–698.

[11] Bushnell, M. L.; Agrawal, V. D.: *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Springer, 2000, ISBN 0-7923-7991-8, 712 s.

[12] Chen, C.; Menon, P.: An Approach to Functional Level Testability Analysis. In *Proceedings of the International Test Conference*, Washington, 1989, s. 373–380.

[13] Cohen, B.: *VHDL Coding Styles and Methodologies*. Norwell, MA, USA: Kluwer Academic Publishers, 1995, ISBN 0-7923-9598-0, 480 s.

[14] Corno, F.; Cumani, G.; Reorda, M. S.; aj.: Efficient Machine-Code Test-Program Induction. In *CEC2002: Congress on Evolutionary Computation*, Honolulu, Hawaii, USA, 2002, ISBN 0-7803-7282-4, s. 1486–1491.

[15] Corno, F.; Reorda, M. S.; Squillero, G.: High-Level Observability for Effective High-Level ATPG. In *VTS '00: Proceedings of the 18th IEEE VLSI Test Symposium (VTS'00)*, Washington, DC, USA: IEEE Computer Society, 2000, ISBN 0-7695-0613-5, s. 411–416.

[16] Corno, F.; Reorda, M. S.; Squillero, G.: RT-Level ITC'99 Benchmarks and First ATPG Results. *IEEE Design & Test*, ročník 17, č. 3, 2000: s. 44–53, ISSN 0740-7475.

[17] 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.

[18] Darnauer, J.; Dai, W. W.-M.: A Method for Generating Random Circuits and Its Application to Routability Measurement. In *FPGA '96: Proceedings of the 1996 ACM Fourth International Symposium on Field-Programmable Gate Arrays*, New York, NY, USA: ACM Press, 1996, ISBN 0-89791-773-1, s. 66–72.

[19] Davidson, S.: Characteristics of the ITC'99 Benchmark Circuits. In *IEEE International Test Synthesis Workshop (ITSW)*, Santa Barbara, CA, March 1999, str. 19.

[20] Dey, S.; Raghunathan, A.; Wagner, K. D.: Design for Testability Techniques at the Behavioraland Register-Transfer Levels. *Journal of Electronic Testing: Theory and Applications (JETTA)*, ročník 13, č. 2, 1998: s. 79–91, ISSN 0923-8174.

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

[22] Eigen, M.: New Concepts for Dealing with the Evolution of Nucleic Acids. In *Cold Spring Harbor Symposia on Quantitative Biology 52*, 1987, ISSN 0091-7451, s. 307–320.

[23] Černý, V.: A Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm. *Journal of Optimization Theory and Applications*, ročník 45, 1985: s. 41–51, ISSN 0022-3239.

[24] Fernandes, J. M.; Santos, M. B.; Oliveira, A. L.; aj.: A Probabilistic Method for the Computation of Testability of RTL Constructs. In *DATE '04: Proceedings of the Conference on Design Automation and Test in Europe*, Washington, DC, USA: IEEE Computer Society, 2004, ISBN 0-7695-2085-5, s. 176–181.

[25] Flottes, M. L.; Pires, R.; Rouzeyre, B.: Analyzing Testability from Behavioral to RT Level. In *EDTC '97: Proceedings of the 1997 European Conference on Design and Test*, Washington, DC, USA: IEEE Computer Society, 1997, ISBN 0-8186-7786-4, s. 158–164.

[26] Fogel, D. B.: *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ, USA: IEEE Press, 1995, ISBN 0-7803-1038-1, 272 s.

[27] Fogel, D. B.: The Advantages of Evolutionary Computation. In *Bio-Computing and Emergent Computation*, Singapore: World Scientific Press, 1997, s. 1–11.

[28] Fogel, L.; Owens, A.; Walsh, M.: *Artificial Intelligence Through Simulated Evolution*. John Wiley, 1966, 162 s.

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

[30] Fujiwara, H.: Needed: Third-generation ATPG Benchmarks. *IEEE Design & Test*, ročník 15, č. 1, 1998: str. 96, ISSN 0740-7475.

[31] Garvie, M.; Thompson, A.: Evolution of Combinatonial and Sequential On-Line Self-Diagnosing Hardware. In *5th NASA / DoD Workshop on Evolvable Hardware (EH 2003)*, Chicago, IL, USA: IEEE Computer Society, 2003, ISBN 0-7695-1977-6, s. 177–183.

[32] Garvie, M.; Thompson, A.: Evolution of Self-Diagnosing Hardware. In *Proc. 5th Int. Conf. on Evolvable Systems (ICES2003): From biology to hardware, LNCS*, ročník 2606, editace A. Tyrrell; P. Haddow; J. Torresen, Springer-Verlag, 2003, ISBN 3-540-00730-X, s. 238–248.

[33] Ghosh, I.; Fujita, M.: Automatic Test Pattern Generation for Functional RTL Circuits Using Assignment Decision Diagrams. *IEEE Transactions on CAD*, ročník 20, č. 3, 2001: s. 402–415, ISSN 0278-0070.

[34] Ghosh, I.; Jha, N. K.; Bhawmik, S.: A BIST Scheme for RTL Controller-Data Paths Based on Symbolic Testability Analysis. In *Proceedings of the 35th Annual Conference on Design Automation Conference*, ACM Press, 1998, ISBN 0-89791-964-5, s. 554–559.

[35] Gloster, C.: Dynamic Scan Testing: Investigating A New Paradigm. Technická zpráva, MCNC, Center for Microelectronics, 1993.

[36] Gloster, C.: ISCAS'89 Addendum Benchmark Set. In *ACM/SIGDA Benchmarks Electronic Newsletter*, 1993, str. 10.

[37] Goldstein, L. H.: Controlability/Observability Analysis for Digital Circuits. *IEEE Transactions on Circuits and Systems*, ročník 26, č. 9, 1979: s. 685–693, ISSN 0098-4094.

[38] Goldstein, L. H.; Thigpen, E. L.: *SCOAP: Sandia Controllability/Observability Analysis Program*. New York, NY, USA: ACM Press, 1988, ISBN 0-89791-267-5, 397–403 s.

[39] Grason, J.: TMEAS, a Testability Measurement Program. In *Proceedings of the 16th Design Automation Conference (DAC'79)*, Piscataway, NJ, USA: IEEE Press, 1979, s. 156–161.

[40] Grason, J.; Stephenson, J. E.: A Testability Measure for Register Transfer Level Digital Circuits. In *Proceedings of International Symposium on Fault Tolerant Computing*, 1976, s. 101–107.

[41] Gu, X.; Kuchcinski, K.; Peng, Z.: Testability Measure with Reconvergent Fanout Analysis and Its Applications. *The Euromicro Journal, Microprocessing and Microprogramming*, ročník 32, č. 1-5, 1991: s. 835–842, ISSN 0165-6074.

[42] Gu, X.; Kuchcinski, K.; Peng, Z.: An Approach to Testability Analysis and Improvement for VLSI Systems. *The Euromicro Journal, Microprocessing and Microprogramming*, ročník 35, č. 1-5, 1992: s. 485–492, ISSN 0165-6074.

[43] Hansen, M. C.; Yalcin, H.; Hayes, J. P.: Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering. *IEEE Design & Test*, ročník 16, č. 3, 1999: s. 72–80, ISSN 0740-7475.

[44] Harlow, J.: Overview of Popular Benchmark Sets. *IEEE Design & Test*, ročník 17, č. 3, 2000: s. 15–17, ISSN 0740-7475.

[45] Hellebrand, S.; Wunderlich, H.-J.: Synthesis of Self-Testable Controllers. In *Proceedings of European Design and Test Conference*, 1994, ISBN 0-8186-5410-4, s. 580–585.

[46] Hlavička, J.: *Diagnostika a spolehlivost číslicových systémů*. Vydavatelství ČVUT, Praha 1, Husova 5, 1978, 153 s.

[47] Homma, N.; Aoki, T.; Higuchi, T.: Graph-Based Individual Representation for Evolutionary Synthesis of Arithmetic Circuits. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, IEEE Press, 2002, ISBN 0-7803-7278-6, s. 1492–1497.

[48] Hutton, M.; Rose, J.; Grossman, J.; aj.: Characterization and Parameterized Generation of Synthetic Combinational Benchmark Circuits. *IEEE Transactions on CAD*, ročník 17, č. 10, 1998: s. 985–996, ISSN 0278-0070.

[49] Hutton, M. D.; Rose, J.; Corneil, D. G.: Automatic Generation of Synthetic Sequential Benchmark Circuits. *IEEE Transactions on CAD*, ročník 21(8), č. 8, 8 2002: s. 928–940, ISSN 0278-0070.

[50] Huynen, M.; Hogeweg, P.: Pattern Generation in Molecular Evolution: Exploitation of the Variation in RNA Landscapes. *Journal of Molecular Evolution*, ročník 39, 1994: s. 71–79.

[51] Iwama, K.; Hino, K.: Random Generation of Test Instances for Logic Optimizers. In *DAC '94: Proceedings of the 31st Annual Conference on Design Automation*, New York, NY, USA: ACM Press, 1994, ISBN 0-89791-653-0, s. 430–434.

[52] Iwama, K.; Hino, K.; Kurokawa, H.; aj.: Random Benchmark Circuits with Controlled Attributes. In *Proc. 1997 European Design and Test Conference*, Washington, DC, USA: IEEE Computer Society, 1997, ISBN 0-8186-7786-4, s. 90–97.

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

[54] Jiang, T.; Klenke, R. H.; Aylor, J. H.; aj.: System Level Testability Analysis Using Petri Nets. In *Proceedings of the IEEE International High-Level Validation and Test Workshop (HLDVT'00)*, Washington, DC, USA: IEEE Computer Society, 2000, ISBN 0-7695-0786-7, str. 112.

[55] Kantor, T.: *Metodika pro generování obvodů se zadanými diagnostickými vlastnostmi*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2004, 69 s.

[56] Škarvada, J.; Růžička, R.: Using Petri Nets for RT Level Digital Systems Test Scheduling. In *Proceedings of 1st International Workshop on Formal Models (WFM'06)*, 2006, ISBN 80-86840-20-4, s. 79–86.

[57] Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P.: Optimization by Simulated Annealing. *Science*, ročník 220, 4598, 1983: s. 671–680, ISSN 0036-8075.

[58] Kotásek, Z.: Uplatnění principů řiditelnosti/pozornosti při návrhu číslicových obvodů. Habilitační práce, FEI VUT v Brně, 1999, 80 s.

[59] Kuchcinski, K.; Peng, Z.: Testability Analysis in a VLSI High-level Synthesis System. *The Euromicro Journal, Microprocessing and Microprogramming*, ročník 28, č. 1-5, 1990: s. 295–300, ISSN 0165-6074.

[60] Kundarewich, P.; Rose, J.: Synthetic Circuit Generation Using Clustering and Iteration. *IEEE Transactions on CAD*, ročník 23, č. 6, 2004: s. 869–887, ISSN 0278-0070.

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

[62] Landman, B. S.; Russo, R. L.: On Pin Versus Block Relationship for Partitions of Logic Circuits. *IEEE Transactions on Computers*, ročník 20, č. 12, 1971: s. 1469–1479, ISSN 0018-9340.

[63] Lohn, J. D.; Larchev, G. V.; DeMara, R. F.: A Genetic Representation for Evolutionary Fault Recovery in Virtex FPGAs. In *Evolvable Systems: From Biology to Hardware, 5th International Conference, ICES 2003*, editace A. M. Tyrrell; P. C. Haddow; J. Torresen, Trondheim, Norway: Springer, 2003, ISBN 3-540-00730-X, s. 47–56.

[64] Makris, Y.; Collins, J.; Orailoglu, A.; aj.: TRANSPARENT: A System for RTL Testability Analysis, DFT Guidance and Hierarchical Test Generation. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, 1999, ISBN 0-7803-5443-5, s. 159–162.

[65] Makris, Y.; Orailoglu, A.: Test Requirement Analysis for Low Cost Hierarchical Test Path Construction. In *Proceedings of 11th Asian Test Symposium*, 2002, ISBN 0-7695-1825-7, s. 134–139.

[66] Mano, M. M.: *Digital Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001, ISBN 0-1306-2121-8, 516 s.

[67] Marinissen, E. J.; Iyengar, V.; Chakrabarty, K.: A Set of Benchmarks for Modular Testing of SOCs. In *Proceedings IEEE International Test Conference (ITC)*, Baltimore, MD, 2002, s. 519–528.

[68] Masner, J.; Cavalieri, J.; Frenzel, J. F.; aj.: Size versus Robustness in Evolved Sorting Networks: Is Bigger Better? In *NASA/DoD Workshop on Evolvable Hardware (EH'00)*, Palo Alto, CA, USA: IEEE Computer Society, 2000, ISBN 0-7695-0762-X, s. 81–90.

[69] Maunder, C. M.; Bennetts, R. G.; Robinson, G. D.: CAMELOT: A Computer-Aided Measure for Logic Testability. In *Proceedings of Intenational Conference on Computer Communication*, 1980, s. 1162–1165.

[70] Mazumder, P.; Rudnick, E.: *Genetic algorithms for VLSI Design and Test Automation*. Prentice Hall PTR, 1998, ISBN 0-1301-1566-5, 338 s.

[71] Mentor Graphics: Scan and ATPG Process Guide. [5/2007].
URL `http://www.fm.vslib.cz/~kes/bs/mg/atpg_gd.pdf`

[72] Miller, J.; Job, D.; Vassilev, V.: Principles in the Evolutionary Design of Digital Circuits – Part I. *Genetic Programming and Evolvable Machines*, ročník 1, č. 1, 2000: s. 8–35, ISSN 1389-2576.

[73] Miller, J. F.; Thomson, P.: Aspects of Digital Evolution: Geometry and Learning. In *Evolvable Systems: From Biology to Hardware, Second International Conference, ICES 98*, ročník 1478, Heidelberg: Springer-Verlag, 1998, ISBN 3-5406-4954-9, s. 25–35.

[74] Nuutila, E.: Efficient Transitive Closure Computation in Large Digraphs. *Acta Polytechnica Scandinavia: Math. Comput. Eng.*, ročník 74, 1995: s. 1–124, ISSN 1237-2404.

[75] Pečenka, T.; Kotásek, Z.: I-path Scheduling Algorithm for RT Level Circuits. In *MEMICS 2006 2nd Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, Mikulov, CZ, 2006, ISBN 80-214-3287-X, s. 174–181.

[76] Pečenka, T.; Kotásek, Z.; Sekanina, L.: FITTest_BENCH06: A New Set of Benchmark Circuits Reflecting Testability Properties. In *Proc. of 2006 IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop*, IEEE Computer Society, 2006, s. 285–289.

[77] Pistorius, J.; Legai, E.; Minoux, M.: Generation of Very Large Circuits to Benchmark the Partitioning of FPGA. In *ISPD '99: Proceedings of the 1999 International Symposium on Physical Design*, New York, NY, USA: ACM Press, 1999, ISBN 1-58113-089-9, s. 67–73.

[78] Raik, J.; Nommeots, T.; Ubar, R.: A New Testability Calculation Method to Guide RTL Test Generation. *Journal of Electronic Testing*, ročník 21, č. 12, 2005: s. 71–82, ISSN 0923-8174.

[79] Ratiu, I.: VICTOR: A Fast VLSI Testability Analysis Program. In *Proc. IEEE International Test Conference*, 1982, s. 397–401.

[80] Ravi, S.; Lakshminarayana, G.; Jha, N.: TAO: Regular Expression Based High-level Testability Analysis and Optimization. In *Proceedings of International Test Conference*, IEEE Computer Press, 1998, ISBN 0-7803-5093-6, s. 331–340.

[81] Růžička, R.: *Formální přístup k analýze testovatelnosti číslicového obvodu na úrovni RT*. Disertační práce, FIT VUT v Brně, 2002, 102 s.

[82] 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.

[83] Sekanina, L.: *Evolvable Components: From Theory to Hardware Implementations*. Natural Computing Series, Springer-Verlag, 2003, ISBN 3-540-40377-9, 194 s.

[84] Sekanina, L.; Růžička, R.: Easily Testable Image Operators: The Class of Circuits Where Evolution Beats Engineers. In *The 2003 NASA/DoD Conf. on Evolvable Hardware*, Chicago, 2003, ISBN 0-7695-1977-6, s. 135–144.

[85] Sheppard, J.; Kaufman, M.: IEEE P1522 Standard for Testability and Diagnosability Characteristics and Metrics (Draft). Informace o připravovaném standardu IEEE P1522 dostupné z http://grouper.ieee.org/groups/1232/pubs/, 2000, [10/2005].

[86] Strnadel, J.: *Analýza a zlepšení testovatelnosti číslicového obvodu na úrovni meziregistrových přenosů*. Disertační práce, FIT, VUT v Brně, 2004, 150 s.

[87] Stroobandt, D.; Depreitere, J.; Campenhout, J. V.: Generating New Benchmark Designs Using a Multiterminal Net Model. *Integration, the VLSI Journal*, ročník 27, č. 2, 1999: s. 113–129, ISSN 0167-9260.

[88] Stroobandt, D.; Verplaetse, P.; Van Campenhout, J.: Generating Synthetic Benchmark Circuits for Evaluating CAD Tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, ročník 19, č. 9, 2000: s. 1011–1022, ISSN 0278-0070.

[89] Thompson, A.: Evolving fault tolerant systems. In *Proc. 1st IEE/IEEE Int. Conf. on Genetic Algorithms in Engineering Systems*, IEE Conf. Publication, 1995, ISBN 0-8529-6650-4, s. 524–529.

[90] Thompson, A.: Evolutionary Techniques for Fault Tolerance. In *Proceedings of the UKACC International Conference on Control*, IEE, 1996, s. 693–698.

[91] Thompson, A.: Evolving Inherently Fault-Tolerant Systems. *Proc Instn Mechanical Engrs, Part I*, ročník 211, 1997: s. 365–371.

[92] Thompson, A.: *Hardware Evolution: Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution*. Distinguished dissertation series, Springer-Verlag, 1998, ISBN 3-5407-6253-1, 115 s.

[93] Vedula, V.; Abraham, J.: FACTOR: A Hierarchical Methodology for Functional Test Generation and Testability Analysis. In *DATE '02: Proceedings of the Conference on Design, Automation and Test in Europe*, Washington, DC, USA: IEEE Computer Society, 2002, ISBN 0-7695-1471-5, s. 730–734.

[94] Verplaetse, P.; Stroobandt, D.; Van Campenhout, J.: Synthetic Benchmark Circuits for Timing-driven Physical Design Applications. In *Proceedings of the International Conference on VLSI*, Las Vegas, Nevada, USA: CSREA Press, 2002, s. 31–37.

[95] Verplaetse, P.; Van Campenhout, J.; Stroobandt, D.: On Synthetic Benchmark Generation Methods. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, ročník IV, 2000, ISBN 0-7803-5483-4, s. 213–216.

[96] Wagner, K. D.; Dey, S.: High-Level Synthesis for Testability: A Survey and Perspective. In *Design Automation Conference*, 1996, ISBN 0-7803-3294-6, s. 131–136.

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

[98] Warshall, S.: A Theorem on Boolean Matrices. *J. ACM*, ročník 9, č. 1, 1962: s. 11–12, ISSN 0004-5411.

[99] Zhang, L.; Ghosh, I.; Hsiao, M. S.: Efficient Sequential ATPG for Functional RTL Circuits. In *Proceedings 2003 International Test Conference (ITC 2003)*, Charlotte, USA: IEEE CS, 2003, ISBN 0-7803-8106-8, s. 290–298.

[100] FITTest_BENCH06 Benchmarks & Cirgen Page. [5/2007].
URL `http://www.fit.vutbr.cz/~pecenka/cirgen`

[101] ISCAS Benchmarks. [5/2007].
URL `http://www.fm.vslib.cz/~kes/asic/iscas/`

[102] ITC'99 Benchmarks Web Site. [5/2007].
URL `http://www.cad.polito.it/tools/itc99.html`

[103] ITC'02 SOC Test Benchmarks Web Site. [5/2007].
URL `http://www.hitech-projects.com/itc02socbenchm/`

[104] Low Power Group Benchmarks. [5/2007].
URL `http://www.ece.cmu.edu/~lowpower/benchmarks.html`

[105] Intel: Moore's Law. [5/2007].
URL `http://www.intel.com/technology/mooreslaw`

Rejstřík

Příloha A

Charakteristiky testovacích sad

Tabulka A.1: Charakteristika testovacích obvodů sady ISCAS'85

Název obvodu	**Počet vstupů**	**Počet výstupů**	**Počet D-KO**	**Počet hradel**	**Počet poruch**
C432	36	7	-	160	524
C499	41	32	-	202	758
C880	60	26	-	383	942
C1355	41	32	-	546	1574
C1908	33	25	-	880	1879
C2670	233	140	-	1193	2747
C3540	50	22	-	1669	3428
C5315	178	123	-	2307	5350
C6288	32	32	-	2406	7744
C7552	207	108	-	3512	7550

Tabulka A.2: Charakteristika testovacích obvodů sady ISCAS'89

Název obvodu	**Počet vstupů**	**Počet výstupů**	**Počet D-KO**	**Počet hradel**	**Počet poruch**
s27	4	1	3	10	32
s208	11	2	8	96	215
s298	3	6	14	119	308
s344	9	11	15	160	342
s349	9	11	15	161	350
s382	3	6	21	158	399
s386	7	7	6	159	384
s400	3	6	21	162	424
s420	19	2	16	196	430
s444	3	6	21	181	474
s510	19	7	6	211	564
s526n	3	6	21	194	553
s526	3	6	21	193	555
s641	35	24	19	379	467
s713	35	23	19	393	581
s820	18	19	5	289	850
s832	18	19	5	287	870
s838	35	2	32	390	857
s953	16	23	29	395	1079
s1196	14	14	18	529	1242
s1238	14	14	18	508	1355
s1423	17	5	74	657	1515
s1488	8	19	6	653	1486
s1494	8	19	6	647	1506
s5378	35	49	179	2779	4603
s9234	19	22	228	5597	6927
s13207	31	121	669	7951	9815
s15850	14	87	597	9772	11727
s35932	35	320	1728	16065	39094
s38417	28	106	1636	22179	31180
s38584	12	278	1452	19253	36305

Tabulka A.3: Charakteristika testovacích obvodů sady ISCAS – Addendum'93

Název obvodu	**Počet vstupů**	**Počet výstupů**	**Počet D-KO**	**Počet hradel**	**Počet poruch**
s344	9	11	15	160	342
s499	1	22	22	152	583
s635	2	1	32	286	666
s938	34	1	32	446	931
s967	16	23	29	394	1066
s991	65	17	19	519	910
s1196	14	14	18	529	1242
s1269	18	10	37	569	1343
s1512	29	21	57	780	1357
s3271	26	14	116	1572	3270
s3330	40	73	132	1789	2870
s3384	43	26	183	1685	3380
s4863	49	16	104	2342	4764
s6669	83	55	239	3080	6684

Tabulka A.4: Charakteristika testovacích obvodů sady ITC'99

Název obvodu	**Počet vstupů**	**Počet výstupů**	**Počet D-KO**	**Počet hradel**	**Počet poruch**
b01	2	2	5	46	127
b02	1	1	4	28	64
b03	4	4	30	149	382
b04	11	8	66	597	1477
b05	1	36	34	935	2553
b06	2	6	9	60	151
b07	1	8	49	420	1120
b08	9	4	21	167	439
b09	1	1	28	159	417
b10	11	6	17	189	468
b11	7	6	31	481	1308
b12	5	6	121	1036	2777
b13	10	10	53	339	835
b14	32	54	245	4475	12643
b15	36	70	449	8893	23316
b16	M+1	1	N	f(M,N)	g(M,N)
b17	37	97	1415	24194	65324
b18	36	23	3320	68752	188458
b20	32	22	490	9419	25274
b21	32	22	490	9803	26516
b22	32	22	735	15071	40200

Tabulka A.5: Charakteristika testovacích obvodů sady ITC'99 (2. vydání)

Název obvodu	**Počet vstupů**	**Počet výstupů**	**Počet D-KO**	**Počet hradel**	**Počet poruch**
b01	2	2	5	49	114
b02	1	1	4	28	62
b03	4	4	30	160	386
b04	8	11	66	737	1646
b05	1	36	34	998	2440
b06	2	6	9	56	134
b07	1	8	49	441	1072
b08	9	4	21	183	442
b09	1	1	28	170	403
b10	11	6	17	206	485
b11	7	6	31	770	1726
b12	5	6	121	1076	2856
b13	10	10	53	362	830
b14	32	54	245	10098	22634
b14_1	32	54	245	6900	15492
b15	36	70	449	8922	21776
b15_1	36	70	449	13098	28787
b17	37	97	1415	32326	76485
b17_1	37	97	1415	39665	87958
b18	36	23	3320	114621	263967
b18_1	36	23	3320	108482	250717
b19	21	30	6642	231320	533142
b19_1	21	30	6642	219424	507476
b20	32	22	490	20226	45395
b20_1	32	22	490	14443	33191
b21	32	22	490	20571	46090
b21_1	32	22	490	14442	32884
b22	32	22	735	29951	67472
b22_1	32	22	735	21772	49881